INTEGRATING SEMANTIC NETWORKS AND OBJECT-ORIENTED MODEL TO REPRESENT AND MANAGE CONTEXT

June 2012

David Sorrentino

Master of Science in Computer Science

INTEGRATING SEMANTIC NETWORKS AND OBJECT-ORIENTED MODEL TO REPRESENT AND MANAGE CONTEXT

David Sorrentino
Master of Science
Computer Science
June 2012
School of Computer Science
Reykjavík University

M.Sc. PROJECT REPORT

ISSN 1670-8539

Integrating Semantic Networks and Object-Oriented model to represent and manage context

by

David Sorrentino

Project report submitted to the School of Computer Science
at Reykjavík University in partial fulfillment of
the requirements for the degree of
Master of Science in **Computer Science**

June 2012

Project Report Committee:

Hannes Högni Vilhjálmsson, Supervisor
Associate Professor, Reykjavík University

Emanuela Merelli, Supervisor
Professor, University of Camerino

Marjan Sirjani
Associate Professor, Reykjavík University

The undersigned hereby certify that they recommend to the School of Computer Science at Reykjavík University for acceptance this project report entitled **Integrating Semantic Networks and Object-Oriented model to represent and manage context** submitted by **David Sorrentino** in partial fulfillment of the requirements for the degree of **Master of Science** in **Computer Science**.

_____

Date

_____

Hannes Högni Vilhjálmsson, Supervisor
Associate Professor, Reykjavík University

_____

Emanuela Merelli, Supervisor
Professor, University of Camerino

_____

Marjan Sirjani
Associate Professor, Reykjavík University

_____

Date

_____

David Sorrentino
Master of Science

Integrating Semantic Networks and Object-Oriented model to represent and manage context

David Sorrentino

June 2012

Abstract

In few years context-aware computing will pervade almost every aspect of our lives. One of the crucial issues related to this field is to have a proper and convenient model to represent and manage the context. Existing representation models like ontologies constitute a well researched and mature solution. However, they are not made to represent continuously changing data; moreover, building and maintaining them might be highly error-prone, time-consuming, and non-scalable processes, and they can become tedious tasks if they are done manually.
This thesis proposes a model to represent and manage contextual information of different types, generated by a variety of heterogeneous sources and with different levels of granularity. The model is derived from the integration of Semantic Networks with the Object-Oriented software development model and has been implemented by exploiting the use of scripting languages and their properties, such as dynamic typing, meta-programming, and introspection. A context-aware infrastructure (CAFE) based on this model and written in Python is presented and, by showing an illustrative contextual scenario implemented in CAFE, it is demonstrated that the proposed model guarantees high readability, flexibility, scalability, general-purpose, and modularity.

A mio padre e mia madre, a cui devo tutto..

A mio fratello Marco..

Al mare..

Acknowledgements

This thesis dissertation marks the end of a long and eventful journey which represents the most important phase of my academic career. It is a pleasant task to express my thanks to all those who contributed in many ways to the success of this study and made it possible and an unforgettable experience for me.

Above all I would like to acknowledge the remarkable sacrifices that my parents **Gerardo** and **Rosalba** made to ensure that I had an excellent education. For this and for being a constant source of love, encouragement and support for me, I am forever in their debt. It is to them that I dedicate this dissertation. Grazie papà e mamma per avermi regalato una vita meravigliosa. Le soddisfazioni più grandi che ho avuto, sto avendo, ed avrò, sono solo merito vostro.

My brother and my best friend **Marco** needs a special mention, since he has been a true support always and I have not missed a single opportunity to trouble him and load him with my difficulties. This dissertation is also dedicated to him. Grazie infinite per esserci sempre stato Marco.

I would like to thank my "brothers-in-arms" **Andrea**, **Lillo** and **Alfredo** with which I spent an extraordinary year of my life in Reykjavik. Let's keep in touch mates!
A huge thank you to all the wonderful people I met during my staying in Reykjavik. Thank you **Arabella**, **Christian**, **Michael**, **Katrin**. If this experience has been unforgettable is also because of you. A special thank you to **Roxy**. Words fail me to express my appreciation for her massive support and generous care. She was always beside me during the happy and hard moments to push me and motivate me. Thank you Roxy, a journey is easier when you travel together.

Huge and warm thanks go to my uncle **Luigi**, my aunt **Ida Rosa**, and my cousin **Monica** for their continuous support. E' bello avervi in famiglia. Vi voglio bene.

Thank you, **Arianna**, for the memorable moments together.
Special thanks also to my friends **Lorenzo**, **Luca S.**, **Mirko**, **Luca O.**, **Daniele**, **Antonio**, **Iacopo**, **Gianluca**, and **Saverio**.

Last but not least, I would like to expand my thanks to my supervisors **Hannes** and **Emanuela**, and my committee member **Marjan**. If this dissertation looks in good shape, it is also because of your precious help and guidance.

David Sorrentino

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Human beings manage without many problems to exchange ideas among them and react properly. This is due to several factors, including the richness of the language they share and the common sense of understanding how daily situations work. When human beings talk to other human beings, they are able to use information retrieved from the current situation, or *context*, to increase the bandwidth of the conversation.
Unfortunately, this ability to exchange ideas is not applicable when human beings interact with computers. This happens since computers do not understand our language, do not understand how our world works, and they can not retrieve information from the current situation; at least they can not do it as easily as the majority of human beings do.
In traditional interactive or desktop computing, users use an impoverishment mechanism to provide information to computers, typically by means of a keyboard or a mouse. The result is that the information has to be explicitly provided to computers. In practice, the users translate what they want to perform into specific commands about how accomplish the task, and therefore they use keyboard and mouse to articulate these details to the computer, so that it can execute their commands. This is not similar at all to the interaction of human beings with other human beings. Consequently, computers are not able to fully exploit the context of a conversation with a human being. By improving the access to the context for computers, users can increase the richness of the communication within a human-computer interaction and make the production of computational services even more useful.
We can either improve the language that the human beings use to interact with computers, or increase the quantity of information related to a given situation, or context, which is available to the computers.
In particular the first approach tries to improve the above-mentioned interaction, allowing the human being to communicate in a much more natural way. However, this type of

communication is explicit, since the computer only knows what the user tells it. In fact, techniques such as speech and gestures recognition do not provide to computers more information than the explicit input.
As it is clear from interactions among human beings, information retrieved from the current situation, such as facial expressions, emotions, past and future events, the presence of other people in the room, and the relationships with these other people, are crucial in order to understand what is happening in a given environment and in a given moment. The process consisting in the building of this common sense of understanding between two people is called *grounding* (Clark & Brennan, 1991). Since both human beings in such interaction share the information retrieved from the current situation, there is no need to make it explicit. This need for explicit information does exist in human-computer interactions, for the computer does not share this implicit information which represents the context. Therefore, the goal of context-aware computing is to use the context as a cue to enrich the impoverished interaction among human beings and computers, making it easier and more efficient.
Indeed, researchers belonging to this field work to make it simpler for users to interact with computers and environments, allowing the users not to have to think consciously about how to use computers. For this purpose, the approach for developing context-aware applications is to gather *implicit* contextual information by using automatic means, making them easily available to the run-time environment of the computer, and letting the application designer free to decide which information is relevant and how to handle it. This is, without any doubt, a better approach, since it can remove the burden of the users to make all the information explicit and put the decisions about what is relevant in the designer's hands.

1.1 Research statement

One of the crucial issues related to context-aware computing is to have a proper and convenient model to represent and manage the context. Existing representation models like ontologies constitute a well researched and mature solution. However, they are not made to represent continuously changing data; moreover, building and maintaining them might be highly error-prone, time-consuming, and non-scalable processes, and they can become tedious tasks if they are done manually.
In this thesis work we propose a model that is *highly-readable*, *flexible*, *scalable*, *general-purpose*, and *modular*, in order to represent and manage contextual information of different types, generated by a variety of heterogeneous sources and with different levels of

granularity.
This model is based on the integration of Semantic Networks with the Object-Oriented model[1], enables reasoning, guarantees an easy handling of the atomic data forming the context, and is capable to manage incomplete or not accurate information due to incompleteness of data from sensors.

1.2 Overall project

This work was conceived as part of a larger project in Assisted Living[2]. The main project idea is a real-time tracking and simulation of a home environment to identify and avert potentially dangerous situations.

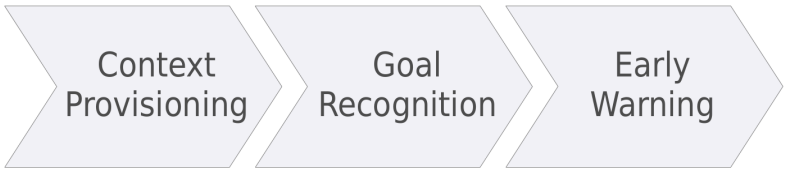

Figure 1.1: Project overview

The general approach is to embed a range of sensors in objects and appliances, in order to allow context awareness. By exploiting the context a state of the real world is described in an abstract language.
Eventually, a simulation, that simulates both the physics of objects and the behaviour of the human, runs a number of steps into the future, and watches for possible dangerous states (as defined by given rules). If the current behaviour of the human seems to be leading towards a disaster (according to the simulation), then the human is alerted through voice, sounds or projected visuals.
For this purpose, the goal of the user needs to be defined; conjoint use of activity recognition and prediction is exploited to achieve this.

[1] Object-oriented modelling is a software engineering approach that models a system as a group of interacting objects. Each object represents some entity of interest in the system being modelled, and is characterised by its class, its state (data elements), and its behaviour. [wikipedia.org]

[2] *Assisted Living* provides supervision or assistance with activities of daily living; coordination of services by outside health care providers; and monitoring of resident activities to help to ensure their health, safety, and well-being. [wikipedia.org]

1.3 Thesis organization

Chapter 2 and Chapter 3 provide some useful background information by giving to the reader a quick overview about the main concepts which this thesis work is based on. We chose not to dedicate a chapter to the Object-Oriented model, since we suppose it constitutes an already well-know subject to everybody. However, detailed information about it can be found in (Nierstrasz, 1989).
In Chapter 2 we clarify the concept of context, context representation, context awareness, listing also some canonical examples of context-aware applications.
In Chapter 3 we define the concept of Semantic Network, providing, in addition, simple examples with the purpose to make it clearer to the reader's eyes. Moreover, we also show the most important properties characterizing Semantic Networks, and the advantages and disadvantages related to the use of them.

In Chapter 4 we present the main existing infrastructures aimed at providing tools for building context-aware applications and systems, mostly focusing on the approach they use to deal with context representation and the knowledge management.

In Chapter 5 we explore some of the main models suitable for representing context, analysing their strong and weak spots and explaining the reasons why we finally preferred Semantic Networks. Furthermore, we describe in details the problem statement and how our approach addresses it. Finally, we point out the weak spots characterizing our methodology.

In Chapter 6 we introduce CAFE, a contextual infrastructure based on the integration of Semantic Networks with the Object-Oriented model, describing how we implemented it and listing its main features.

In Chapter 7 we propose an illustrative scenario in order to show how it is possible to use CAFE to represent and manage the context related to that scenario. We also use this demonstration to underline the results we achieved and point out the limitations of CAFE.

Finally, in Chapter 8 we summarise our thesis work, highlighting the results of our study and proposing possible future works related to it.

Chapter 2

Context-aware computing

Context-aware computing is a mobile computing paradigm in which applications can discover and take advantage of contextual information, such as user location, time of day, nearby people and devices, and user activity (G. Chen & Kotz, 2000).
Naturally, context-aware computing environments are based on the knowledge of the context. This is because users have the expectation that they can access whichever information and service they want, whenever they want, and wherever they are. In order to ensure that these expectations are satisfied, the need of a context is clear. Indeed, the context can be used with the aim of helping to determine which information or services should be made available to the users (Krumm, 2009).

Since our aim is to propose a model to represent and manage the context, in this chapter we chose to clarify basic concepts such as *context*, *context representation*, *context awareness*, and *context reasoning*, providing also some canonical examples of context-aware applications.

2.1 Definition of context

Merriam-Webster defines the context as "the interrelated conditions in which something exists or occurs".

In 1994 Schilit and Theimer introduced for the first time the term *context-aware*. At that juncture, they referred to the context as location, people's identity and close objects, and changes which characterize those objects. In 1997 Brown et al. gave a similar definition, defining the context as location, identity of people around the user, time of the day, season, temperature, etc. In 1998 Ryan et al. described the context as the user's location,

environment, identity and time. In the same year (1998) Dey defined the context as an emotional state of the user, focus of attention, location and orientation, date and time, and objects and people in the user's environment. These definitions describe the context by means of examples and they are quite difficult to apply. Indeed, if we wanted to determine whether a type of information not listed in the above-mentioned definitions is context, it would not be clear how to use those definitions to do it.
Other definitions simply provide synonyms of context, referring, for example, to the context as an environment or a situation. Someone claims that the context is the user's environment, whereas others look at it as the application's environment. In 1996 Brown defined the context as the user's environmental elements which are known by the computer. In 1997 Ward et al. considered it as the state of everything surrounding the application, and in 1998 Rodden et al. defined it as the application's setting. In 1997 Hull et al. took into account the entire environment, defining the context as the aspects of a current situation. Right as it happens with the definitions by example, definitions which simply use synonyms of context are extremely hard to be applied in practice. In 1994 Schilit et al. asserted that the crucial aspects of the contexts are the following ones: where you are, whom you are with, and which resources are surrounding you. Indeed they defined the context as the execution of an environment that changes continuously.
In fact, in their definition, they include the following environmental elements:

- *Computing environment*. Available processors, accessible devices for user input and display, network capability, connectivity, and computational costs.
- *User environment*. Location, list of close people, and social situations.
- *Physical environment*. Illumination, noise level, etc.

Dey et al. (1998) defined the context as the physical, emotional, or informational user's state. Finally, in the same year (1998), Pascoe described the context as the subset of physical and conceptual states which are interesting for a given entity. In effect, the context is entirely based on the situation judged relevant for an application and on its set of users. It is not possible to list which aspects of all the situations are important, since this would change from situation to situation. For example, in some cases the physical environment can be important, whereas in others it can be totally useless.
Finally, in 2000, Dey and Abowd defined the context as "... any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves".

In practice the context-aware computing looks at questions such as “who?”, “where?”, “when?”, and “what?” and use these information to determine which situation is occurring and why.
This type of context includes not only implicit inputs, but also explicit ones. For example, a user’s identity can be explicitly sensed by means of face recognition or can be explicitly determined when an user is asked to write his/her name by using a keyboard. From the application point of view, both are information related to the user’s identity and allow to perform further activities.
In conclusion, there are some types of context which are more important than others. These are *location* (where), *identity* (who), *time* (when), and *activity* (what). In fact, location, identity, time, and activity are crucial to characterize the situation related to a given entity. These types of context not only answer to questions like “who?”, “where?”, “when?”, and “what?”, but they also behave as indexes for other contextual information sources. For example, given the identity of a given person, we may acquire many pieces of related information, such as telephone numbers, addresses, email addresses, a birth date, lists of friends, relationships with other people belonging to the same environment, etc. With the location of an entity we could determine which objects or people are close to that entity and which activities are occurring around it (Poslad, 2009).
To improve the context categorization, it is also possible to include hierarchical or containment information. An example of this related to the location could be a spot in a room. That spot may be defined by means of coordinates within the room, the room itself, the floor of the building which the room is in, the building, the city, etc. (B. Schilit & Theimer, 1994).

2.2 Types of context

Since the concept of context was defined the first time, several ways to classify contexts have been proposed. Prekop and Burnett (2003) and Gustavsen (2002) referred to two types of context: one external and one internal, which correspond to physical and user contexts, respectively. In 2003, Hofer et al. referred to physical and logical contexts where the logical context is similar to the user context. In 2001, Dey and Abowd proposed rooms and buildings, people, either individuals or groups, and things such as physical objects and components as types of context. In 1995, Schilit et al. classified the context in three main categories: where you are (location context including resources belonging to the physical environment which are located close to the user), who you are with (social

context), and what ICT[1] resources are nearby. Chen and Kotz (2000) and Henricksen et al. (2002) performed a distinction between dynamic and static context-aware systems. A static context describes aspects of a pervasive system which remain invariant, like the birth date of a person. A dynamic context refers to a user or an environment context. These contexts may be highly changeable over the space and the time, e.g., temperature, blood pressure, etc. In 2000 Morse et al. modelled the context as six main dimensions described by means of the questions "what?", "who?", "where?", "how is it accessed?" and "why is it useful?".

Table 2.1 summarizes the characteristics of the types of context classified with respect to the type of environment: physical, human, and ICT, respectively. The following classification is based on the one of Morse et al. (2000) (Poslad, 2009).

Context type	**Characteristic**	**Description**
Physical	What	Type of physical environment context-awareness such as awareness of temperature, light intensity, etc.
	Where	*Spacial awareness* or *location awareness*: where an awareness of context can be exploited.
	When	*Temporal awareness*: when context-awareness is useful.
ICT	How	*ICT awareness*: awareness of how any context is created and adapted over an ICT infrastructure.
User	Who	*User context-awareness*: who may benefit from an awareness of someone's context. *User Activity or Task Context*: describes a user's current situation. *Social Context*: describes how the actions of someone may affect others.
Goal	Why	*User or application goal*: why a context is useful, the higher-level application or user purpose the context is used for. *Context Adaptation*: how the current context can perform a transition to the goal context.

Table 2.1: Types of context according to (Poslad, 2009)

Every individual context is, in turn, defined by means of a *meta-context*, information that describe the context. For example, the context related to the location has also to define which type of coordinates and units system are use. The most important invariant context is the user and his features, such as his identity and perhaps the details related to his contacts.

[1] *Information and Communications Technology* (ICT) is often used as an extended synonym for information technology (IT), but is usually a more general term that stresses the role of unified communications. [wikipedia.org]

The context often implies a situation influencing another system that is outside the system, the external conditions of the environment surrounding the system (*external context*), which is described by Morse et al. (2000). In the same way, the system can be influenced by the conditions of the internal system and the use conditions which a system is aware of (*internal context*). More generally, we can refer to a context as something related to the use of the system, i.e., by means of an internal or external interaction.
According to (Poslad, 2009), various types of context can have multidimensional characteristics and they can be modelled as shown in Figure 2.2.

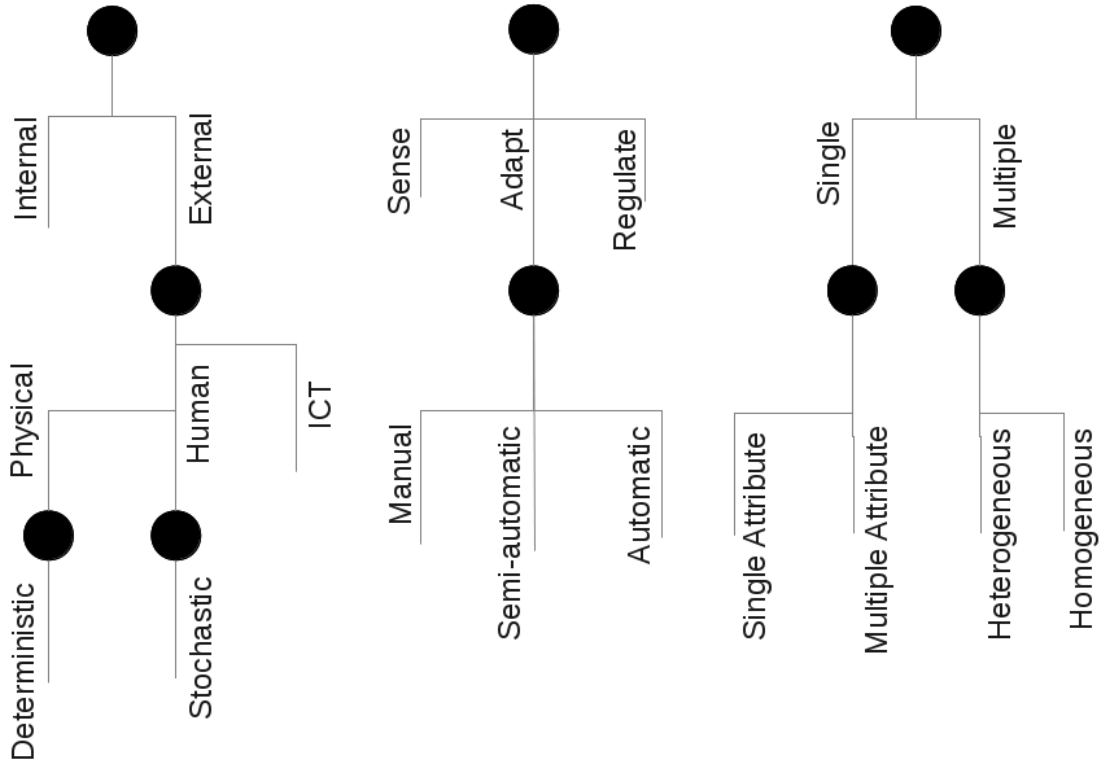

Figure 2.1: Multidimensional features of types of context according to (Poslad, 2009)

2.3 Context creation and context composition

New contexts can be created at run time by using sensors located in the physical environment, e.g., sensors for the temperature. Often, raw contexts belonging to a lower abstraction level and coming from sensors need to be processed in contexts belonging to a higher abstraction level which are relevant for users and applications. The data retrieved by the sensors may need to be scaled or transformed in different range of values or domains, e.g., an electrical signal emitted by a temperature sensor can be mapped to

a value on a temperature scale such as Celsius or Fahrenheit. Valued retrieved by means of sensors may also need to be connected to other entities in order to become useful, e.g., coordinates in absolute position may be mapped to a positional context such as a given building or a postcode. Rather than using the location coordinates themselves as a low-level context, it is often more useful to exploit an abstraction of the context, e.g., "this person is at this person's home at this moment".
Some contexts such as location, entity, activities and time, may act as contextual information sources, which other contexts can be determined from; in that case we can speak of *context-reuse*. By combining multiple individual contexts it is possible to obtain a more accurate comprehension of the current situation, rather than by taking into account every individual context. For example, knowing the current geographical position and the time, together with the user's calendar, allows an application to infer a new user context and understand whether the user is at work or waiting for the bus, etc. According to Poslad (2009) the approaches which can be used to determine user contexts are:

- Combining several simple contexts (*context composition*).
  - Combining homogeneous contexts: e.g., from multiple independent sensors in order to deal with the variation in individual measurements.
  - Combining heterogeneous contexts: e.g., determine a composite context combining two or more contexts.
  - Deriving high-level context from lower-level ones: e.g., get a higher-level context starting from lower-level contexts.
  - Deriving a lower-level context from a higher-level one: e.g., an absolute location coordinate can be determined from a street name or a building name.
- Consulting an user profile, e.g., check an user calendar to know the user's activity at a certain time.
- Asking users, in order to define directly their preferences.
- Observing users, e.g., use image processing technology to identify faces, fingerprints, etc, and then link these features to the user context.

Context-aware systems may guarantee the possibility to interconnect heterogeneous contexts. These contexts may be, for example, of the same type, but represented in a different way and defined by using different meta-contexts.
For example, there could be two different types of temperature measurement scales, Celsius and Fahrenheit, so the same type of context can be described differently.

2.4 Context representation

In 2004 Strang and Linnhooff-Popien studied several ways to represent a context and they identified six different types of representation. Moreover, they have been discussing and evaluating these six types, according to the following parameters: partial validation, richness and quality of information, incompleteness and ambiguity of context gathered, and level of formality and applicability to existing environments. Table 2.2 summarizes this distinction, describing the above-mentioned six different types of context representation, according to Strang and Linnhoff-Popien (2004).

Model	Type of structure
Key value	Simple, flat, data structure for modeling contextual information.
Markup scheme	Hierarchical data structure, e.g., XML, consisting of user defined markup tags with attributes that can be arbitrarily nested.
Graphical	Graph data structures and richer data types, e.g., UML (Unified Modelling Language).
Object-Oriented	Context processing is encapsulated, hidden to other components. Access is through specified interfaces only.
Logic based	A logic defines the conditions in which a concluding expression or fact may be derived from a set of other expressions or facts.
Strong ontology	A combined expressive conceptual model with a logic.

Table 2.2: Types of context representation according to (Poslad, 2009)

Moreover, Table 2.3 points out the pros and cons of each type of context representation (Poslad, 2009).

2.5 Context awareness

Context-aware computing was described for the first time in 1994 by Schilit and Theimer as a software that "... adapts according to its location of use, the collection of nearby people and objects, as well as changes to those objects over time".
This first definition of context-aware applications moved the idea of context awareness from applications which are simply informed about the context to applications which adapt to it. In fact the term "context-aware" became synonym of "adaptive" (M. Brown, 1996), "reactive" (Cooperstock, Tanikoshi, Beirne, Narine, & Buxton, 1995), "responsive" (Elrod, Hall, Costanza, Dixon, & Rivieres, 1993), "situated" (Hull, Neaves, & Bedford-Roberts, 1997), "context sensitive" (Rekimoto, Ayatsuka, & Hayashi, 1998) and "environment directed" (Fickas, Kortuem, & Segall, 1997).

Model	Pros	Cons
Key value	Easy to manage and parse in embedded systems.	Uses exact matches, lacks expressive structuring, lacks efficient context retrieval algorithms, has weak formalism, handling incompleteness. May need multi-values.
Markup scheme	Distributed model, uses underlying resource identifier and namespace model; XML Web services are becoming pervasive; handling heterogeneity, handling incompleteness.	Expressive structuring and weak formalism.
Graphical	More expressive than key value, and hierarchies.	Support for distributed context model, handling incompleteness, lack of formalism.
Object-Oriented	Distributed Object-Oriented is mature, some partial validation but often not very formal. Reuse can be supported through inheritance and composition.	Handling incompleteness.
Logic based	Strong formalism, expressive structuring.	Handling uncertainty, time varying instances, heterogeneity, often difficult to partially validate, simple structuring, handling incompleteness.
Strong ontology	Expressive structuring, handling heterogeneity, partial validation.	Handling uncertainty, scalability in searching large data volumes, use in low resource embedded environments.

Table 2.3: Pros and cons of types of context representation according to (Poslad, 2009)

Previous definitions of context-aware computing fall into two main categories: using context and adapting to context. Finally, in 2000 Dey and Abowd defined context awareness in a more general way with the following statement: “A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task”.

Since the above-mentioned definition is more general, it can include all context-aware applications, both those that adapt to context and those that simply display the user’s context.

Therefore, context awareness can be seen as the ability of a system to sense the current environment and autonomously perform appropriate adaptations in regard to its optimal

operation, general behaviour and user interaction. When a user enters a new context, it is desirable that the applications on his devices be able to adapt to the new situation, and the environment be able to adapt its services to the presence of the new user.

2.6 Context-aware applications

Applications using the context, both in a desktop environment and in a mobile or ubiquitous computing one, are called *context-aware applications*. The growing availability of commercial sensing technologies is making it even more possible to sense the context in a wide variety of environments. The existence of networked powerful computers makes it possible to use these technologies and distribute the context in multiple applications. Mobile computing allows users to move through an environment carrying their computing power with them. By combining that with wireless communications it is possible to give users the access to information and services not directly available on their portable devices. Increased mobility creates situations where the user's context, such as his/her position, or the people and the objects around him/her, is more dynamic. With the introduction of ubiquitous computing users move though an environment and interact with computer-enhanced objects within the environment. Moreover, this allows users to access to information and remote services. Since there is such a wide range of possible situations for the users, we need a means to make services adapt properly, in order to best support human-computer and human-environment interactions. Context-aware applications are becoming even more frequent and they can be found in many areas, such as wearable computing, mobile computing, robotics, adaptive and intelligent user interfaces, augmented reality, adaptive computing, intelligent environments, and context-sensitive interfaces. It is not surprising that in the majority of these areas, the user is mobile and his/her context changes continuously and quickly (Krumm, 2009).

2.6.1 Types of context-aware applications classified by features

Classifying context-aware features provides two main benefits. Firstly it specifies the types of applications which researchers focus on. Secondly it describes the types of features which developers should consider when they build context-aware applications.
The first attempt to classify context-aware applications according to their features was made by Schilit et al. in 1994 and that taxonomy had two orthogonal dimensions: if the task is to obtain information or execute a command, and if the task is manually or automatically executed. Applications which retrieve information for the user in a manual way

going by the available context are classified as *proximate selection* applications. This is an interaction technique with which a list of objects, or places, is presented to the user and only the items which are relevant for the user's context are emphasized and made easier to choose. Applications which retrieve information for the user in an automatic way based on the available context are classified as *automatic contextual reconfiguration*. Basically, it is a system-level technique which automatically creates a resource and makes it available according to current context. Applications which execute commands for the user in a manual way going by the available context are classified as *contextual command* applications. They are executable services made available by means of the user's context. The execution of these can always be modified on the basis of the user's context. Finally, applications which execute commands for the user in an automatic way based on the available context use *context-triggered actions*. These are services which are automatically executed when the right combination of contexts occurs, and are based on simple if-then rules (Poslad, 2009).
The above-mentioned classification can be summarized in Figure 2.2.

	manually	automatically
retrieving information	proximate selection	automatic contextual configuration
executing commands	contextual command	context triggered actions

Figure 2.2: Classification of context-aware application by features

More recently, Pascoe (1998) proposed a taxonomy of context-aware features. This taxonomy presents a considerable overlap with the Schilit's one, but some important differences as well. In effect Pascoe's taxonomy has the aim to identify core features of context awareness, whereas the Schilit's one identifies classes of context-aware applications. The first feature is *contextual sensing* and it is the ability to retrieve contextual information and

present them to the user, trying to augment the user's sensory system. This looks quite similar to proximate selection, but it actually differs from it for the fact that the user does not necessarily need to select one of the context items for more information. The second feature is *contextual adaptation* and it is the ability to execute or modify a service in an automatic way going by the current context. This can be directly mapped on the Schilit's context-triggered actions. The third feature is *contextual resource discovery*, and it allows context-aware applications to locate and use resources and services which are relevant to the user's context. This can be directly mapped on an automatic contextual reconfiguration. The fourth and last feature, *contextual augmentation*, is the ability to associate digital data with the user's context. So, an user has the possibility to visualize the data when he is in that associated context. For example, a user can create a virtual note providing details about which food is missing in the fridge and attach the note to the fridge. When another user is close to the fridge, he will see the virtual node left previously.

In 2000 Dey and Abowd combined the ideas from the two above mentioned taxonomies in a new and innovative taxonomy. This taxonomy, like the Pascoe's one, is a list of the context-aware features that context-aware applications may support, and it is based on three major categories:

- *Presentation* of information and services to a user.
- Automatic *execution* of a service.
- *Tagging* of context as information for later retrieval.

The presentation can be seen as a combination of Schilit's proximate selection and contextual commands, with the addition of Pascoe's notion of presenting context to the user (context as a form of information). An example of this feature can be a mobile computer that dynamically updates the list of the closest printers to it. Automatic execution can be seen as the combination of Schilit's context-triggered actions and Pascoe's contextual adaptation. An example of this second feature could be when an user prints a document out and this is printed from the closest printer to him. Tagging can be mapped on the Pascoe's contextual augmentation. An example of this third feature could be when an application keeps track of the names of the documents the user printed out, the time in which they were printed out and the printer used. Therefore, these information may be accessed by the user later on, perhaps to help him to retrieve the printouts he forgot to pick up.

2.6.2 Examples of context-aware applications

The Active Badge system

The Active Badge system (Want, Falcao, & Gibbons, 1992) is commonly seen as the first context-aware system. The Active Badges were infrared transmitters transmitting a unique identity code. When the user moved trough a building, a database was dynamically updated with information related to the geographical position of each user, the closest phone, and the likelihood of meeting somebody in that location.
In this way, when a phone call was received for a given user, the receptionist was able to forward it to closest phone for that particular user. This applications was focused on the location-aware computing and today it could belong to the class of the location-based services.

Tour guides

If the Active Badge is considered the first context-aware system, the mobile tour guide is definitely considered the most canonical one, since the literature is practically full of examples about this system. A mobile tour guide is a hand-held device which a museum visitor usually receives. It can present information to the user about exhibit locations which he/she is interested in, using audio, video/images, or text. Nowadays, the majority of the tour guides ask the user to explicitly insert the name or the ID of the exhibit location which he desires more information about. Instead, context-aware mobile tour guides remove this need of an explicit input. Indeed, they automatically sense which exhibit location the visitor is closest to (e.g., using RFID[2] tags on every exhibit location and a RFID reader in the tour guide) and automatically present the proper information about that exhibit location (Krumm, 2009).
While early systems such as the Active Badge system were focused heavily on location, later systems started taking into account users' interests, such as the amount of time they spent at a exhibit location or the amount of time they had to tour in choosing what information to show, and what exhibit locations to recommend.

[2] *Radio-frequency identification* (RFID) is the use of a wireless non-contact system that uses radio-frequency electromagnetic fields to transfer data from a tag attached to an object, for the purposes of automatic identification and tracking.

Reminders

Another canonical context-aware application is the context-aware reminder system. Context-aware reminders present reminders to individuals, triggered by changes in context. An alarm clock uses a simple contextual trigger, time, to set off an alarm, a simple form of reminder. Similarly, location-based services can deliver reminders when users are at a particular location or within some proximity of each other (W. N. Schilit, 1995). More sophisticated reminder systems use a combination of different forms of context to trigger reminders. In being more sophisticated, these applications can remind users more appropriately, delivering the right reminder in the right situation (Dey & Abowd, 2000) (Ludford, Frankowski, Reily, Wilms, & Terveen, 2006).

Environmental controls

Another canonical context-aware application is a system to control an environment's heating and lighting, generally for the purposes of being energy-efficient or saving users effort. As many people often leave lights on unnecessarily, or have to manually change heating or cooling levels to remain comfortable, many systems have been developed that can control these on behalf of users. Some are based on simple rules (Elrod et al., 1993), while others use more sophisticated mechanisms to learn how users use a space and sets heating and lighting accordingly (Mozer, 1998).

2.7 Context reasoning

Context-aware systems need a reasoning mechanism mostly because of some basic characteristics of contextual data. These characteristics are *imperfection* and *uncertainty*.
In 2004, Henricksen and Indulska identified four different types of imperfect contextual information:

- *Unknown*. Because of sensor or connectivity failures, not all the contextual data are available all the time.
- *Ambiguous*. If contextual data come from different sources, the may be ambiguous.
- *Imprecise*. Imprecision might be really common in sensor-derived data.
- *Erroneous*. Erroneous contextual information is generally due to human or hardware errors.

In these cases reasoning can be used to detect possible errors by predicting missing values, and evaluating the quality and validity of the sensed data (Bikakis, Patkos, Antoniou, & Plexousakism, 2007).

Furthermore, context reasoning might also be used to allow a system to take decisions based on the gathered contextual information. By exploiting this mechanism, in fact, a system can change its behaviour depending on the changes in its context.

Mature and well-researched reasoning techniques are the *ontological* reasoning and the *rule-based* reasoning.
The ontological reasoning approaches are characterized by two main advantages. First, they are well-integrated with the ontology model, which is widely used as representation model for the context. Second, they have a relative low computational complexity; therefore they are made to be used with rapidly changing contexts.
Instead, rule-based reasoning approaches use a formal model to reason on the context. For this reason, they are easy to understand and many systems integrate them with the ontology model. Anyway, they have to use additional reasoning mechanisms to deal with imperfection and uncertainty.

2.8 Summary

Since our aim is to propose a model to represent and manage the context, in this chapter we clarified basic concepts such as context, context awareness, and context reasoning. In addition we introduced the important role played by the context representation in context-aware systems. Finally, we provided some canonical examples of context-aware applications.

In the next chapter we provide a quite detailed overview about Semantic Networks, since they constitute the representation model we chose to work on in our approach. 4

Chapter 3

Semantic networks

Natural language is extremely powerful. Without any effort it allows us to ask somebody how to find the nearest grocery store, to share our knowledge or to express our opinion about something. As a simple example, think about the following two sentences. Both are of the form "subject-verb-object", one of the simplest possible grammatical structures:

1. David owns a dog.
2. Dogs scare Rose.

Each of these sentences represents a piece of information. The words "David" and "Rose" refer to specific people, the word "dogs" refers to a class of mammalian, and the words "owns" and "scare" define the relationship between the person and the animal in question. Since we know from previous experience what the verbs "owns" and "scare" mean, and we have probably seen a cat before, we are able to understand the two sentences. And after reading them, we can say we are equipped with new knowledge of the world. This is a plain example of *semantics*: symbols can refer to things or concepts, and sequences of symbols express a meaning. We can now use the meaning that we derived from the two sentences to answer simple questions such as "Who owns a cat?" or "Who is scared by cats?".

Merriam-Webster defines semantics as "the language used to achieve a desired effect on an audience." Semantics is, in fact, the process of communicating enough meaning to result in an action. A sequence of symbols can be used to communicate meaning, and this communication can then affect behaviour (Segaran, Taylor, & Evans, 2009). For example, when we read a book, we integrate the ideas expressed in the book with all that we already know.

In conclusion the use of semantics is necessary, if we aim to make it possible to represent, combine, and share knowledge between communities of machines, and to write systems that can act on that knowledge.

Since in our approach we use Semantic Networks as knowledge representation for context, in this chapter we are going to give a definition of them, providing in addition some simple examples with the purpose to make them clear to the reader. Finally, we are also going to show the main properties characterizing Semantic Networks, and the advantages and disadvantages related to them.

3.1 Definition of Semantic Networks

Semantic Networks are graphical knowledge representation schemes consisting of nodes, and links between nodes (Marra & Jonassen, 1996). Computer implementations of semantic networks were first developed for artificial intelligence and machine translation, but earlier versions have long been used in philosophy, psychology, and linguistics (J. F. Sowa, 1991).

The nodes of the net represent objects or concepts and the links represent relations between nodes. The links are directed and labelled; therefore, a semantic network corresponds to a directed graph. From the graphical point of view, the nodes are usually represented by circles or boxes and the links are drawn as arrows or simple connectors between the circles. The structure of the network defines its meaning, depending on which nodes are connected to which other nodes. In practice, by defining a set of binary relations on a set of nodes, the network corresponds to a predicate logic with binary relations. Moreover, Semantic Networks are redundancy-free, since they can not have duplications of the same nodes.

3.2 Understanding Semantic Networks

In order to have a concrete example of what a Semantic Network is, let us look at Figure 3.1, which is just composed of two nodes and a link. As can be seen, the node on the left labelled "person" is linked to the node on the right, labelled “living being”. The link is labelled “is-a”. Thus, the Semantic Network in questions describes a person as an example of living being. Indeed, technically speaking, the diagram represents the fact

that there is a binary relation between a living being, such as a person, and the concept of person itself.

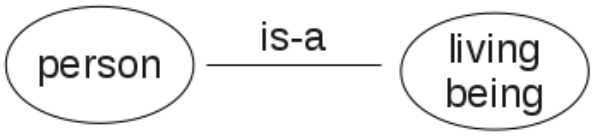

Figure 3.1: Example of Semantic Network

In Figure 3.2 another node with the label “cat”, as well as a “is-a” link from this node to the “living being” node, again representing that a cat is a type of living being.

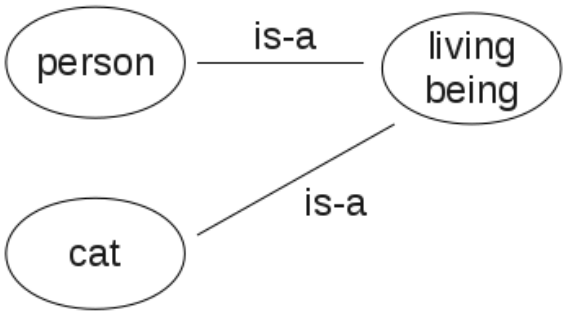

Figure 3.2: Example of Semantic Network (cont’d)

If a person called “David” and a cat called “Tom” are added, and David owns Tom, the structure of the network becomes apparent as shown in Figure 3.3. Clearly, a new link labelled "owns" would need to be added as well, in order to represent that David owns Tom.

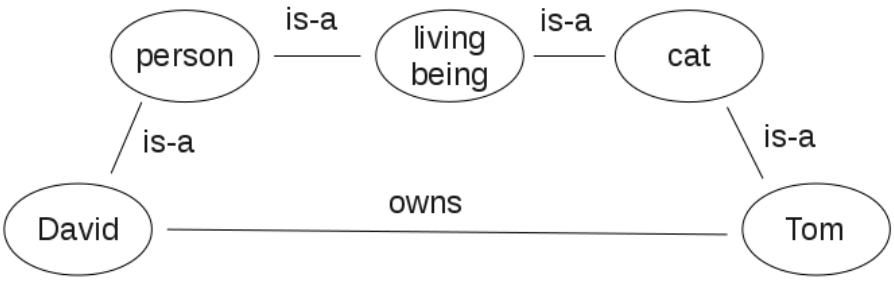

Figure 3.3: Example of Semantic Network (cont’d)

At this stage it is important to clarify a point which can create some semantic confusion. It is visible that the nodes belonging to this small network are not all of the same type.

Indeed the nodes labelled “living being”, “person” and “cat” represent the generic or meta or class concept of a living being, a person and a cat, respectively; in practice, they represent just abstract concepts. Instead, the nodes “David” and “Tom” represent an individual instance of the nodes “person” and “cat”, respectively; in fact David is a person and Tom is a cat. In conclusion it is crucial to notice that there are two types of context, classes and individuals, although they are represented in the same way.

Now, let us add another class node, labelled “place”, that represents the abstraction of places in a category. Along with that, an instance of a place, labelled “home”, is added. Thus, another “is-a” link and a new link, labelled “is-at”, must be added to the node “home” and the node “David”, respectively. These new additions are shown in Figure 3.4. The information now being represented is that David is a person and home is the place he is at.

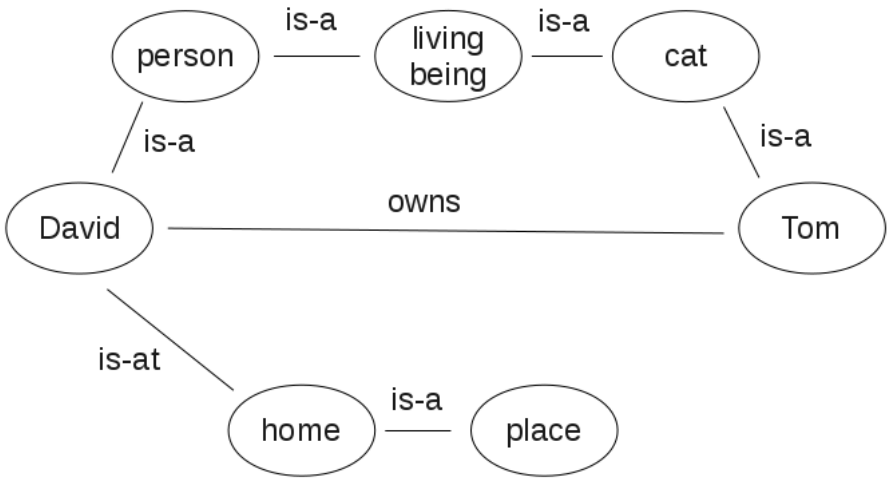

Figure 3.4: Example of Semantic Network (cont’d)

As the number of nodes increases, the meaning of the respective links need to be considered. It should be apparent that not all links are alike. Indeed, some links express only relationships between nodes, and are therefore *assertions* of the nature of the relationship between two different nodes. For example, the link “is-at” in Figure 3.4, which describes the relationship that the person David is at the place home. The “is-a” links in Figure 3.4, instead, are *structural* links, in that they provide “type” information about the node. It is clear since this information is about the node itself and not about the relationship it has to be a different type of node. For instance, the node “home” is an individual instance of the class node labelled “place”.

In Figure 3.5, more nodes and links are added to the original network. There is now a “posture” class node with an instance node labelled “sitting”. The link “has-posture”

conveys the information that the person David has the posture “sitting” in a given moment. We also added a class node labelled “appliance” with an instance node labelled “television”, which in turn is related to the person “David” by means of the link “uses”. Then, we added a class node labelled “room” and a respective instance labelled “living room”. Finally, we added a new link labelled “is-in”, that connects the nodes “David” to the node “living room”, and the node “living room” itself to the node “home”.

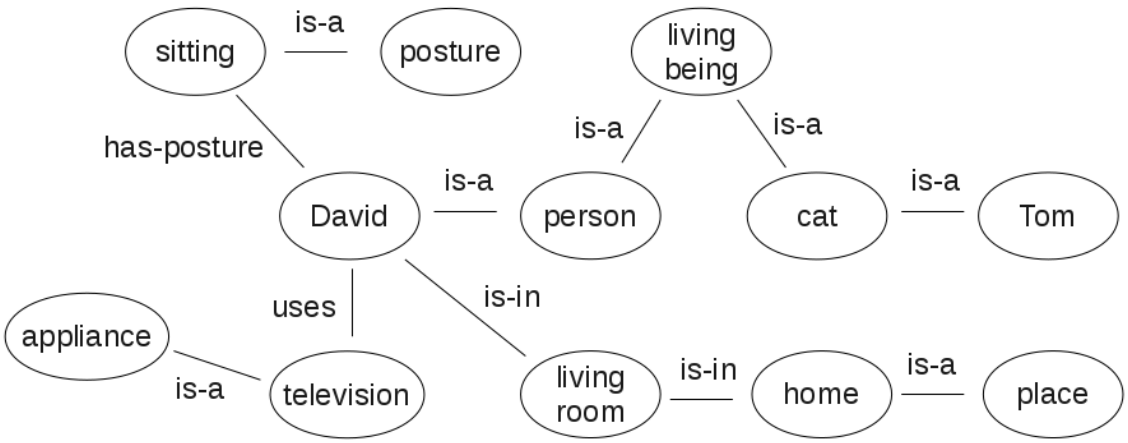

Figure 3.5: Example of Semantic Network (cont’d)

The network in Figure 3.5 now provides a representation for information about the nodes belonging to it. For instance, a person called David is the owner of a cat called Tom, and at the moment he is sitting in the living room, using a television.

Another important characteristic of the node-link representation is the implicit “inverse” of all relationships represented by a link. Indeed, if there is a link going from one node to another, this also implies the reverse, and it means that there is a link from the second node to the first. in Figure 3.6, for example, there are two nodes labelled “David” and “television” with the link labelled “uses”. The direction of the relationship is that “David uses a television”. In practice “David” is a subject and “television” is the object, and “uses” is the verb or action or link between them.

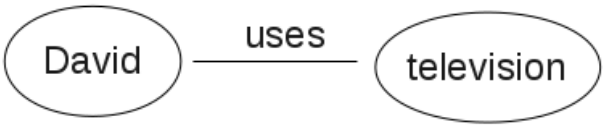

Figure 3.6: Symmetric relationships in Semantic Networks

This “David uses television” relation implies the inverse relationship that “television is-used-by David”, as shown in Figure 3.7.

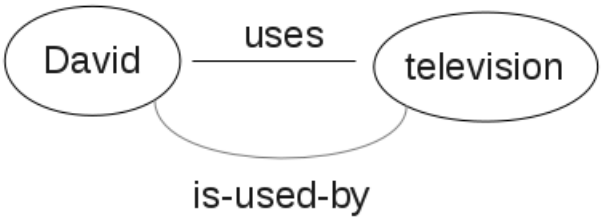

Figure 3.7: Symmetric relationships in Semantic Networks (cont'd)

3.3 Inferring knowledge with Semantic Networks

With any kind of knowledge representation scheme, it is possible to infer knowledge that is not directly represented by the scheme. The ability to work with incomplete knowledge sets a knowledge representation apart from a database (Marra & Jonassen, 1996).

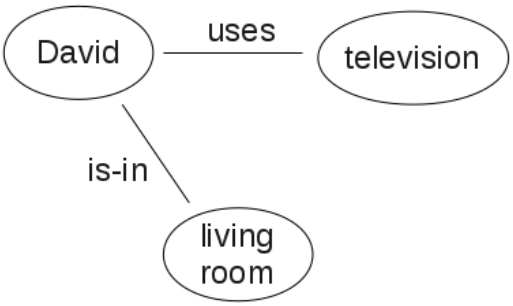

Figure 3.8: What can we infer from this extraction from 3.5

To give an example of what can be found out from the Semantic Network in figure 3.5 that is not directly represented, let us consider Figure 3.8. It is nothing but the an extraction of Figure 3.5 containing only three nodes and two links. The information explicitly represented is that a person called David is using a television and that he is in the living room.

By tracing the path from the node "living room" to the node "David" via the link labelled "is-in" and then from the node "David" to the node "television" via the link labelled "uses", it is possible to infer that the television is in the living room by inferring a link labelled "is-in" between the node "television" and the node "living room", as shown in Figure 3.9. This means that this information does not need to be explicitly represented in the original network, for it can be easily inferred later.

From a mathematical point of view, *composing* links occurs by placing them end-to-tail. This *composition* creates a new link. It is not possible to compose every pair of links,

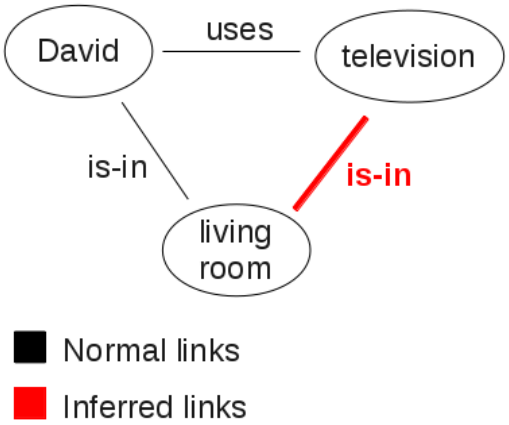

Figure 3.9: Example of knowledge inferring in Semantic Networks (cont'd)

only those whose destinations and sources correspond. The destination of the first must be the source of the second. By composing links, new relationships between nodes can be found and described. Such a process is also called *chasing links* and the terminology introduced comes from a branch of mathematics called *Category Theory*[1] (Marra & Jonassen, 1996).

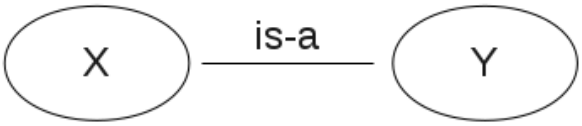

Figure 3.10: Simple example of instancing in Semantic Networks

Looking at Figure 3.10 and formalising the whole lot from a logical point of view, we can say that if x is an individual and y is class, the link "is-a" between them can be interpreted as the following formula:

$$y(x)$$

E.g.: $cat(Tom)$.

Instead, if x and y are classes, the link between them can be interpreted as the following formula:

$$\forall Z\, x(Z) \;\Rightarrow\; y(Z)$$

[1] *Category theory* is an area of study in mathematics that examines in an abstract way the properties of particular mathematical concepts, by formalising them as collections of objects and arrows, where these collections satisfy some basic conditions.

E.g.: $\forall Z\ cat(Z) \ \Rightarrow\ living\_being(Z)$.

Finally, if a class or an individual has some properties, these can be translated to binary predicates:

$$\forall Z\ y(Z) \ \Rightarrow\ property(Z, value) \qquad \textit{class}$$
$$property(x, value) \qquad \textit{individual}$$

In conclusion, coming back to our original example, Figure 3.11 shows the results of more link chasing. As you can see, additional relationships are derived, e.g., a person has a posture, may own a cat and may use appliances.

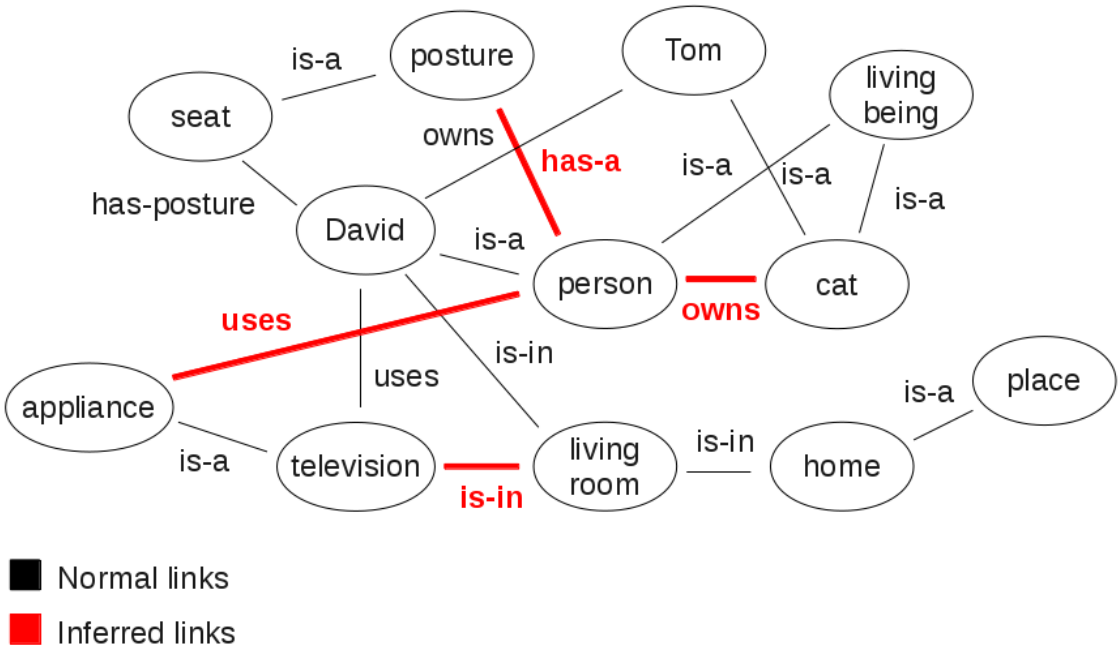

Figure 3.11: A more complicated example of inference in Semantic Networks

3.4 Advantages and disadvantages of Semantic Networks

As we saw thus far, Semantic Networks are characterized by a high representational and expressive power, which is why they constitute a powerful and adaptable method of representing knowledge. In particular, Semantic Networks present the following advantages:

- Many different types of entities can be represented in Semantic Networks.
- Semantic Networks provide a graphical view of the problem space and therefore they are relatively easy to understand.

- They can be used as a common communication tool between different fields of knowledge, e.g., between computer science and anthropology.
- They allow an easy way to explore the problem space.
- Semantic Networks provide a way to create clusters of related elements.
- They resonate with the ways in which people process information.
- They are a more natural representation than logic (using meaning axioms).
- They are characterized by a higher cognitive adequacy than logic-based formalisms.
- Semantic Networks allow the use of efficient inference algorithms (graph algorithms).
- They have a higher expressiveness than logic (e.g., they allow properties overriding).

Semantic Network also have some limitations, which frequently lead to some epistemological problems. Such limitations can be summarized in three main points.

1. A distinction between classes and individuals does not exist. The system is limited by the user's understanding of the meanings of the links in a semantic network. As pointed out previously, links between nodes are not all alike in function or form. Indeed, we need to differentiate between links that constitute some relationship and links that are structural in nature. Figure 3.12 shows an example of the same link used both to create a relationship between two nodes and to describe a structure. In fact, the link "is-a" behaves in two different ways: between the nodes "Tom" and "cat" it specifies an instance of a cat; instead, between the nodes "cat" and "living being" it specifies a category, a hierarchy.

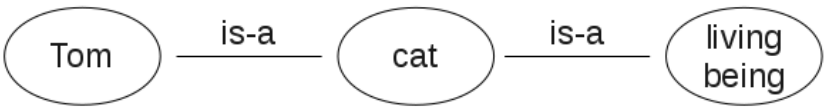

Figure 3.12: Example of a link used with different meanings

In 1983, Brachman on the subtleties of the "is-a" link revealed even more distinctions in the uses of this link.
A possible work-around to this problem could be to specify in a more detailed way the name of the links, distinguishing between relational and structural ones, as shown in Figure 3.13. In this case we re-wrote the link between the nodes "Tom"

and “cat” as an “instance-of” link; and the link between the nodes “cat” and “living being” as a “subtype-of” link.

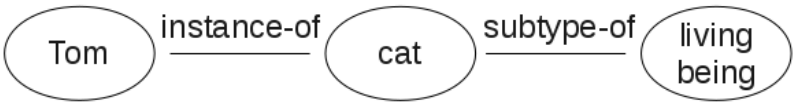

Figure 3.13: Removing ambiguity from a link

2. A distinction between attributes associated to a class and attributes inherited by the individuals of the class does not exist.
   So, on the one hand we may correctly say:
   - Tigers are an endangered species.
   - Cleopatra is a tiger.
   - Cleopatra is endangered.

   But on the other hand, we may also erroneously say:
   - Tigers are numerous.
   - Cleopatra is a tiger.
   - Cleopatra is numerous.
3. A formal semantic does not exist, so there is not an agreed-upon notion of what a given representational structure means. Indeed, Semantic Networks do tend to rely upon the procedures that manipulate them (J. F. Sowa, 1991). A solution to this problem could be either using conceptual graphs (J. Sowa, 1984), a formalism for knowledge representation, or a knowledge representation system such as KL-ONE, which allows to overcome semantic indistinctness in Semantic Network representation.

3.5 Summary

Since in our approach we use Semantic Networks as knowledge representation for context, in this chapter we gave a definition of them, providing in addition some simple examples with the purpose to make them clear to the reader. Finally, we showed the main

properties characterizing Semantic Networks, and the advantages and disadvantages related to them.

In the next chapter we are going to list and analyse the main existing contextual infrastructures, focusing on the approach they use to deal with the context representation and the knowledge management.

Chapter 4

Related works

Context-aware computing requires an environment for software development and deployment, where large quantities of different devices and sensors need to be integrated, building a programmable and auto-configurable infrastructure. So far several projects have developed prototypes of such environments, but usually with focus only on specific use cases, user tasks or application domains. However, looking at the current tendency in a few years nearly every public and private space will be equipped with sensors and smart appliances that are able to automatically adapt to the preferences and demands of the local users and provide special context-specific services to them.

In this section we present the main existing infrastructures aimed to provide tools for building context-aware applications and systems, mostly focusing on the approach they use to deal with the context representation and the knowledge management.

Gaia

Gaia provides a generic computational environment which integrates physical spaces and their ubiquitous computing devices into a programmable computing and communication system(Román et al., 2002). It is rather similar to a traditional operating system, since it manages the tasks common to all applications built for physical spaces (Ranganathan & Campbell, 2003). Each space is self-contained, but it may interact with other spaces. By specifying well-defined interfaces to services, applications may be built in a generic way so that they are able to run in arbitrary active spaces. Gaia is a mature project, since the first prototypes were implemented in 2002 and several applications for active-classrooms have already been developed.

The Gaia Context Infrastructure allows applications to get contextual information. The context is obtained from either sensors or other data sources by means of various components, called Context Providers. These Context Providers allow applications to query them for context information.
Gaia uses ontologies to represent the context and describe various concepts of an Ubiquitous Computing environment, such as kinds of applications, services, devices, users, data sources and other entities. There ontologies are written in DAML+OIL.
All the ontologies in Gaia are maintained by an Ontology Server. Entities contact the Ontology Server to get descriptions of other entities in the environment, information about context or definitions of various terms used in Gaia.

CoBrA

Context Broker Architecture (CoBrA) is an infrastructure that supports agents, services and devices that interact in order to explore context information in active spaces (H. Chen, 2004) (H. Chen, Finin, & Joshi, 2003). Its main component is an intelligent agent called *context broker*, which is responsible for providing a common model to represent context information, inferring higher-level context information not directly available from sensors (H. L. Chen, 2004).
CoBrA has a context-acquisition model in order to acquire contextual information from sensors, agents, and the Web. The acquisition phase is made by means of a library including procedures for collecting information from Smart Tag sensors (location) and environment sensors (temperature, sound, luminosity, etc.).
The base ontologies used for representing context information are the CoBrA Ontology[1] (COBRA-ONT) and SOUPA[2].
All the devices, services, and agents in the space of interest share a centralized model of context provided by the system. The knowledge owned by the context broker consists of RDF statements and is stored in a persistent knowledge base. In order to acquire contextual information, all agents have to send query messages to the context broker.

[1] *COBRA-ONT* is a set of ontologies to describe contextual information and to share context knowledge. It defines concepts for representing actions, agents, devices, meetings, time and space.

[2] The *SOUPA ontology* is a standard ontology for supporting pervasive and ubiquitous computing applications. It consists of vocabularies for expressing common concepts that are associated with person, agent, belief-desire-intention, action, policy, time, space and event.

Semantic Space

Semantic Space is a context infrastructure developed to address three key issues (X. Wang, Dong, Chin, Hettiarachchi, & Zhang, 2004).

1. It aims to provide an explicit representation of the raw context data obtained from various sources in different formats.
2. It provides means for the applications to access in a selective way a subset of context data through context queries.
3. It provides reasoning capabilities for inferring higher-level contexts.

Semantic Space uses the CONtext ONtology[3] (CONON) for modeling context in pervasive computing environments (X. H. Wang, Zhang, Gu, & Pung, 2004).
Every smart space contains a *Context Knowledge Base*, which provides a persistent context knowledge storage. The *context aggregators* are responsible for gathering context data from various sources such as hardware sensors and software programs, and then asserting the gathered data into the context knowledge base. The knowledge base is updated every time a context event occurs.

CHIL

CHIL (Computers in the Human Interaction Loop) is a middleware infrastructure providing features as context modeling, control of sensors and actuators, directory services for infrastructure elements and services (Soldatos, Dimakis, Stamatis, & Polymenakos, 2007). Mechanisms for modelling composite contextual information and describing networks of situation states are also available.
The system is a distributed multi-agent system where the agents are provided with fault tolerance capabilities, since they can migrate among hosts.
This infrastructure can use a wide range of sensors for context acquisition, and new sensors can be plugged into the framework to provide new information. Software agents are responsible for obtaining context information from sensors and making them accessible through the *Knowledge Base Agent*.
CHIL uses a modularized ontology, in order to allow different parts to be used in different contexts and applications. The core module may provide a merged version of the concept.

[3] *CONON* is an ontology aiming to be an extensible upper-level context ontology providing a set of basic concepts that are common to different environments.

In order to globally put together all the modules, the ontology consists of a main OWL[4] file, which imports all modules. In this way developers interested only in a smaller set of modules can define a main OWL file of their own that imports only the module of interest. The knowledge base server is accessible both locally and remotely through a unique interface. The server remote interface is programming language independent, so that there are not constrains about the programming language to use for building the client components.

CAMUS

Context-Aware Middleware for URC (Ubiquitous Robotic Companion) System, also known as CAMUS, is a context-aware infrastructure for the development and execution of a network-based intelligent robot system (Kim, Cho, & Oh, 2005).
Basically CAMUS gathers context information from different sensors and provides appropriate context information to different applications. In addition, CAMUS uses context-aware autonomous service agents that are capable to adapt to different situations.
A *sensor framework* has the aim to process raw data from various sources such as physical sensors, applications and user commands and transfer them to the *Context Manager* through an *Event system*. The Context Manager manages context information collected from the Sensor Framework. When a context event occurs in the environment, the Context Manager transfers the event to the Event System.
The context model in CAMUS is represented as a four-layered space, where each layer has a different abstraction level:

1. *Shared vocabulary layer*. It consists of a set of shared vocabulary used in the common ontology layer.

2. *Common ontology layer*. This layer contains the ontology concepts that are commonly used in various applications. It provides the high-level knowledge description to context-aware applications.

3. *Domain ontology layer*. This layer the domain specific knowledge to context-aware applications.

4. *Instance layer*. It is a space where all the instances of the ontology concepts are represented.

[4] The *Web Ontology Language* (OWL) is a family of knowledge representation languages for authoring ontologies. [wikipedia.org]

SAMOA

SAMOA is a framework that supports the creation of semantic context-aware social networks (Bottazzi, Montanari, & Toninelli, 2007). In details, SAMOA allows mobile users to create *roaming social networks* that, following user movements, reflect all nearby encounters of interest at each instant.
In order to support the creation of social networks in ubiquitous environments, SAMOA exploits geographical context information, e.g., users' location and proximity.
In SAMOA contextual data are modelled and represented in terms of semantic metadata. Places and users are the entities in the system, and they are associated with profiles describing their characteristics.
SAMOA does not provide a centralized database containing the contextual information. All the contextual data are maintained and analysed separately.

OWL-SF

OWL-SF is a distributed semantic service framework supporting the design of ubiquitous context-aware systems considering both the distributed nature of context information and the heterogeneity of devices that provide services and deliver context (Mrohs et al., 2005). It exploits OWL in order to represent high-level context information in a semantically well-founded form.
OWL-SF uses Super Distributed Objects[5] (SDOs) to encapsulate context providers such as sensors, devices, user's interfaces or services.
Each functional entity in OWL-SF has to be described using a dedicated ontology that provides an automatic classification of the object into appropriate service categories.

DRAGO

Distributed Reasoning Architecture for a Galaxy of Ontologies (DRAGO) is a distributed reasoning system, implemented as a peer-to-peer architecture (Serafini & Tamilin, 2005). Every peer may contain a set of different ontologies describing specific domains of interest. In each peer there are also semantic mappings defining semantic relations between

[5] A *Super Distributed Object* (SDO) is a logical representation of a hardware device or a software component that provides well-known functionality and services. One of the key characteristics in super distribution is to incorporate a massive number of objects, each of which performs its own task autonomously or cooperatively with other objects.

entities belonging to two different ontologies. New peers may be added dynamically to the system, providing new ontologies and semantic mappings.
DRAGO is implemented to operate over HTTP and access ontologies and mappings published on the web.

4.1 Summary

In this chapter we presented the main existing infrastructures aimed to provide tools for building context-aware applications and systems, mostly focusing on the approach they use to deal with the context representation and the knowledge management.
The majority of these contextual infrastructures use ontologies as model to represent and manage the context, since they constitute a fundamental knowledge representation structure in modern Artificial Intelligence. Existing representation models like ontologies constitute a well researched and mature solution. Anyway, they are not made to represent continuously changing data; moreover, building and maintaining them is highly error-prone and time-consuming process, and it can become a tedious and non-scalable task if it is done manually. (Zouaq, Gasevic, & Hatala, 2011).

In the next chapter we introduce a new model to represent and manage context, based on the integration of Semantic Networks and Object Oriented software development approach.

Chapter 5

Approach

In this thesis work we propose a new model that is highly-readable, flexible, scalable, general-purpose, and modular, in order to represent and manage context, where context is intended as information of different types, generated by a variety of heterogeneous sources and with different levels of granularity. This model is based on the integration of Semantic Networks with the Object-Oriented software development model, and it combines all the advantages of the two approaches. Furthermore, it enables reasoning, guarantees an easy handling of the atomic data forming the context, and is capable of managing incomplete or inaccurate information due to incompleteness of data from sensors.

Before choosing to combine Semantic Networks and the Object-Oriented model, we tried many other representation models for context, some of them very well researched and mature. In this chapter we explore them, analysing their strong and weak spots and explaining the reasons why we finally preferred Semantic Networks. Moreover, we describe in details the problem statement and how our approach addresses it. Finally, we point out the weak spots characterizing our methodology.

5.1 Choosing a representation model for the context

At the beginning of our work we faced a tough question: "which representation model should we use to represent contextual data?". To answer this question we went through several representation models, analysing them and pointing out their strong and weak spots.
The first kind of dataset we thought about, in order to represent the context, was the simplest one, and probably also the most common one: the table.

5.1.1 Context as a table

Tabular data is any data set organized as a table, such as an Excel spreadsheet or an HTML table. The major advantage of tabular data is, without any doubt, represented by the ease with which we can read and manipulate it. Let us consider the data shown in Table 5.1 describing people at home.

person	**posture**	**room**	**uses**
Andrea	standing	kitchen	toaster, oven, knife
Roxy	standing	hall	-
Marco	sitting	living room	dvd-player, projector

Table 5.1: Contextual data represented in tabular model

Data kept in a table are generally easy to display, sort, print, and edit. Actually, data in a table can be considered not modelled at all, but the fact that they are placed in rows and columns gives each piece a particular meaning. Indeed, unlike modelling methods to represent data, there is no variation in the ways we can look at tabular data. However, it is interesting to note that a data table or a spreadsheet is characterized by a semantics: in fact the row and column in which the data is explains what the data means to a person reading the table. The fact that "living room" is in the same row as "Marco" tells us immediately that Marco is in the living room. We simply know it, since we understand what people and rooms are.

This type of dataset has obvious limitations. Let us consider the column "uses". We just included a list of objects (used by somebody) into a single column. This turns out to be fine if we are simply going to read the table, but it causes some issues if we want to add more information, such as the type of object a person is using (e.g., appliance, kitchenware, etc). In theory it could be possible to add this information in parentheses after every object listed, as shown in Table 5.2.

person	**posture**	**room**	**uses**
Andrea	standing	kitchen	toaster (appliance), knife (kitchenware)
Roxy	standing	hall	-
Marco	seat	living room	DVD-player (appliance), projector (appliance)

Table 5.2: Multiple information in tabular model

However, the spreadsheet software will not understand that we have used an individual field to store multiple distinct information values, since it will not capture the deeper meaning of the text we entered.

Another problem with spreadsheet and tables occurs when we have multiple spreadsheet-s/tables which refer to the same data. For example, if we had another table containing the

addresses of the people described in Table 5.1, there would be no simple way to look for information combining both documents. For sure experts of the field could use macros and lookup tables to get the desired data, but this way is definitely rigid, limited, and usually not interchangeable among different users.
Therefore, we chose to look for a more sophisticated way to model and represent data in order to satisfy our expectations; in particular we considered to use relational databases, as shown in the next section.

5.1.2 Context as a relational database

Relational databases constitute an alternative form to the simple table/spreadsheet and they are considered very fast and powerful tools for storing big amount of data, where the data model is well defined and the way in which the data will be used is predictable. Basically, a relational database is composed of multiple tables joined in a standardized way. Therefore, in order to store the information previously represented in Table 5.2, we could define a schema like the one shown in Figure 5.1. This approach allows us to represent the same data in a more useful and flexible way. Moreover, it appears clear how in this data model the semantics of the data are more explicit. The meanings of the values are actually described by the schema: in fact whoever looks at the tables can easily notice that there are several types of entities modelled and that they have specific relationships between them. Moreover, although the database does not know what a "person" is, it is able to respond to requests to list all the people with given properties. This is why each datum is labelled with what it means, simply using the table and the columns in it.
In our approach we aim to represent data that could quickly change (e.g., the posture of a person), and that is not well understood from the beginning (we do not know how people may want to use it).

As a plain example, let us consider the database in Figure 5.1, and then let us assume that we receive a new database, with additional information about people which are not in the first schema (see Table 5.3).

person	**gender**	**country**
Andrea	male	Iceland
Roxy	female	Poland
Marco	male	Italy

Table 5.3: Dataset containing additional information

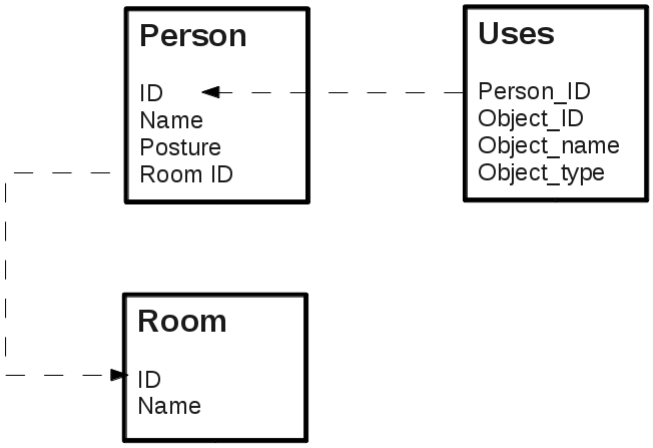

Figure 5.1: Contextual data represented in relational model

At this stage, in order to update the main database so that it supports the new data, we could just link the tables with another table, without being forced to change the existing structure. Figure 5.2 shows the new database structure including an additional table, that aims to link the existing table with the new one.

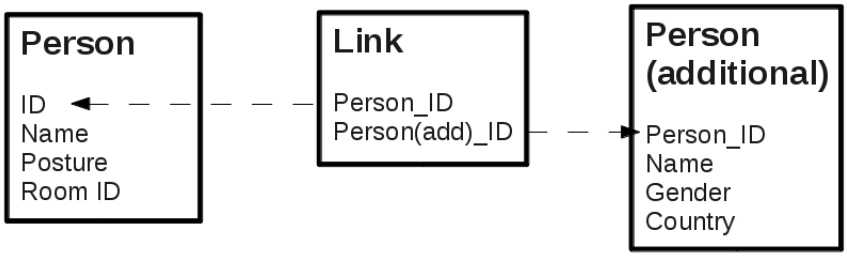

Figure 5.2: Linking additional data to the existing schema

This approach works, but it introduces a considerable problem: there are two name fields in the database, and if we wanted to query it by name, we need to look at both tables. In this way, adding and updating data is much more complicated.
A smart way to overcome this problem could be to have a middle table containing common data, as shown in Figure 5.3. It allows us to achieve our goal, but we need to transform the old data model to the new one, and usually this process (called *schema migration*) is really difficult to perform. In addition, all the queries that were written assuming a certain table structure have to be changed as well. Since our approach needs to deal with data from environments that are constantly using new datasets, performing a schema migration each time a new type of data is faced is simply not convenient.

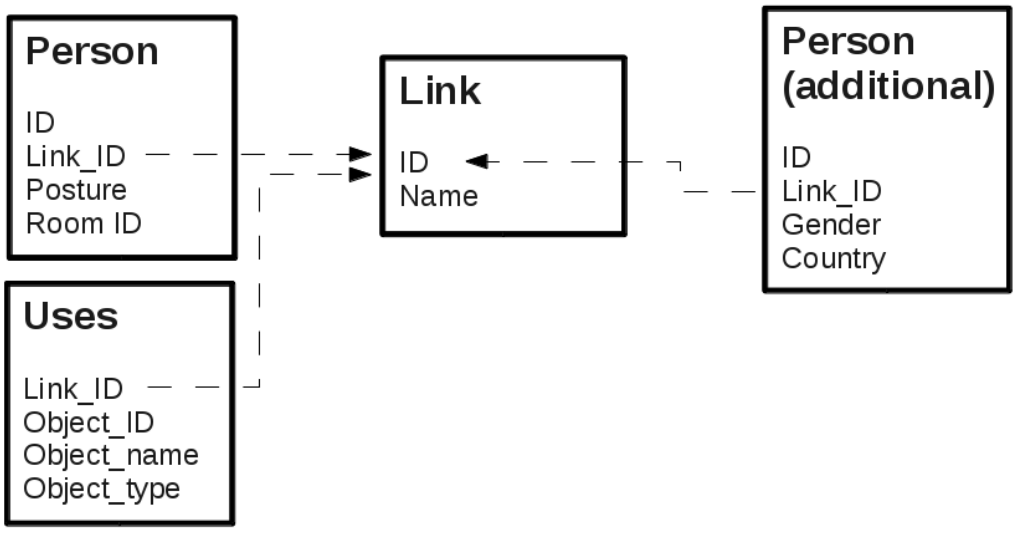

Figure 5.3: Normalized schema after adding a new dataset

In addition, another problem related to relational databases is that schemas can get extremely complicated when dealing with a large amount of data characterized by different types (see Figure 5.4).

In conclusion we did not find it appropriate for our purposes to use relational databases, since we did not judge it flexible enough to handle a wide variety of types of data which tend to change quite frequently. In addition its level of readability did not look high enough in our opinion.
Thus, looking for a more flexible and expressive solution with a high degree of readability, we next examined a graphical knowledge representation model: Semantic Networks.

5.1.3 Context as a Semantic Network

Semantic Networks, about which we provided a quite complete synthesis in Chapter 3, are characterized by a three-column format known as *triple*. In practice, triples form the fundamental building blocks of this type of knowledge representation. Each triple is composed of a *subject*, a *predicate*, and an *object*. For simplicity, we could think of triples just as linguistic statements, where each element corresponds to a piece of grammar used to express a sentence (see Figure 5.5).

Usually, the subject of a triple corresponds to an entity, that is whatever can have a conceptual class. In our approach examples of entities may be people, places, concrete objects

[1] Source: *http://alabra.com* .

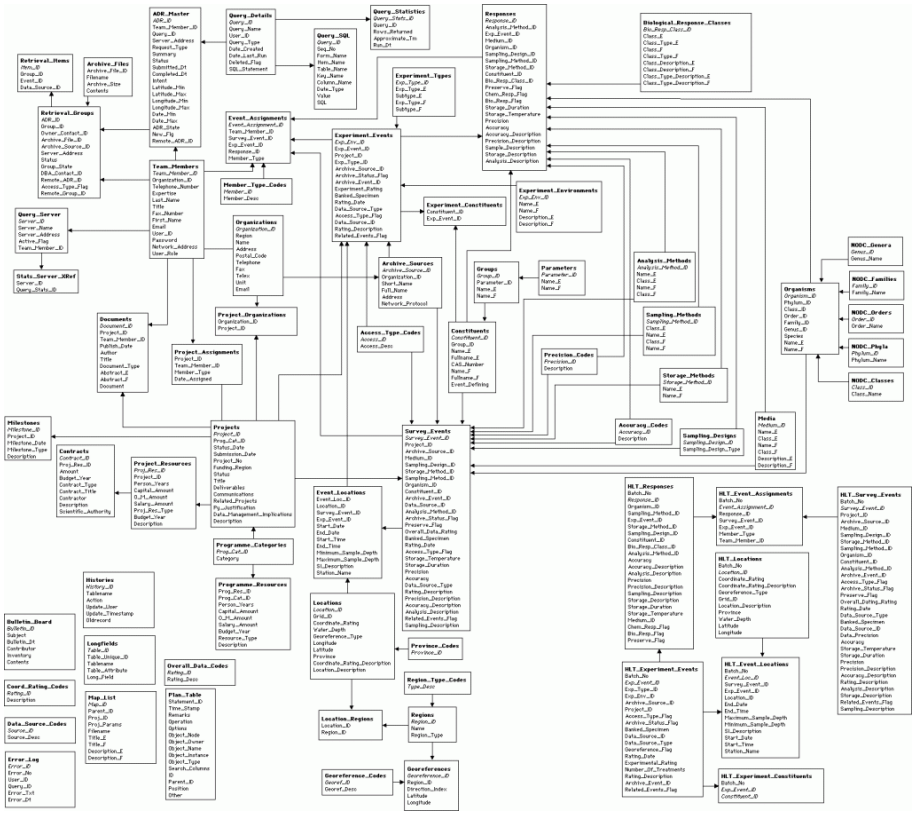

Figure 5.4: Example of a big relational schema[1]

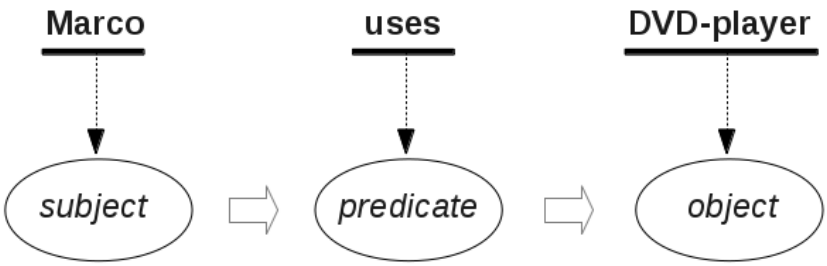

Figure 5.5: *Subject-predicate-object* model

(e.g., appliances, kitchenware) and also things like periods of time and ideas. In the network, subjects are represented as nodes.

Predicates are a property of the entity to which they are attached. A person's location or posture are examples of predicates. In the network, predicates correspond to the links.

Objects fall into two classes: entities that can be the subject of other triples, and literal values such as strings, boolean, or numbers. In the networks objects are represented as nodes.

Obviously, multiple triples can be tied together by using the same subjects and objects

in different triples, building in this way chains of relationships, and forming a directed graph.

Directed graphs are well-known data structures in computer science and mathematics, thus they constitute a reliable model to represent our data. One of the marvellous properties of using graphs to model information is that if we have two separate graphs with a consistent system of identifiers for subjects and objects, we can merge the two graphs without effort. This is because nodes and relationships in graphs are first-class entities, and each triple constitutes a piece of meaningful data (Segaran et al., 2009). In addition, if a triple belongs to both graphs, the two triples merge together in a transparent way, because they are identical. Figure 5.6 and Figure 5.7 show the ease with which we can merge arbitrary datasets thanks to this approach.

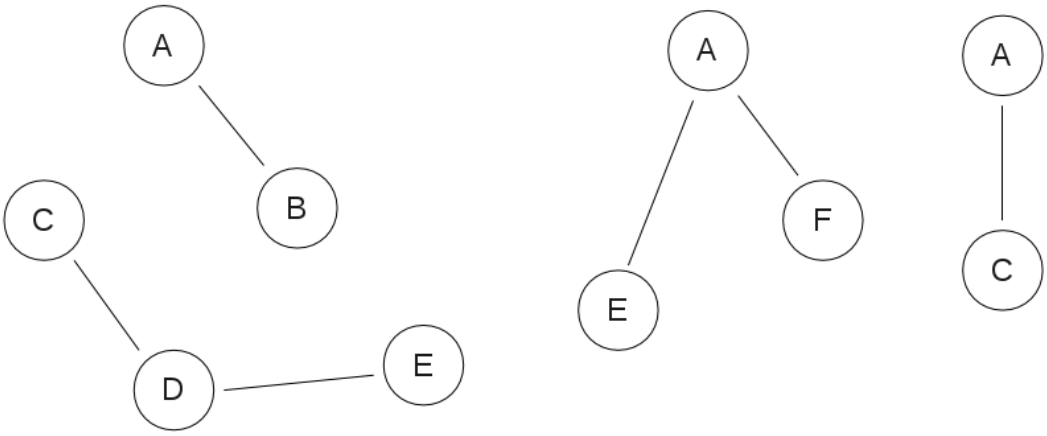

Figure 5.6: Separate graphs sharing some identifiers

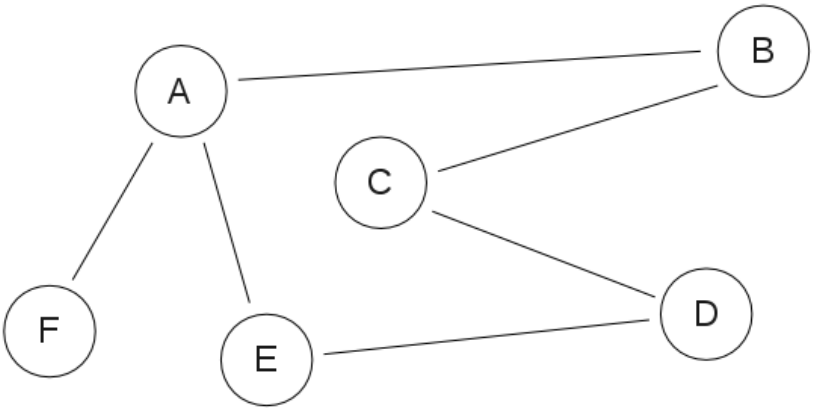

Figure 5.7: Merged graph obtained from the union of triples

Let us consider again some data contained in Table 5.1 and let us use a graph model to represent them, looking at people as entities. Figure 5.8 illustrates the high expressiveness and readability of this representation model.

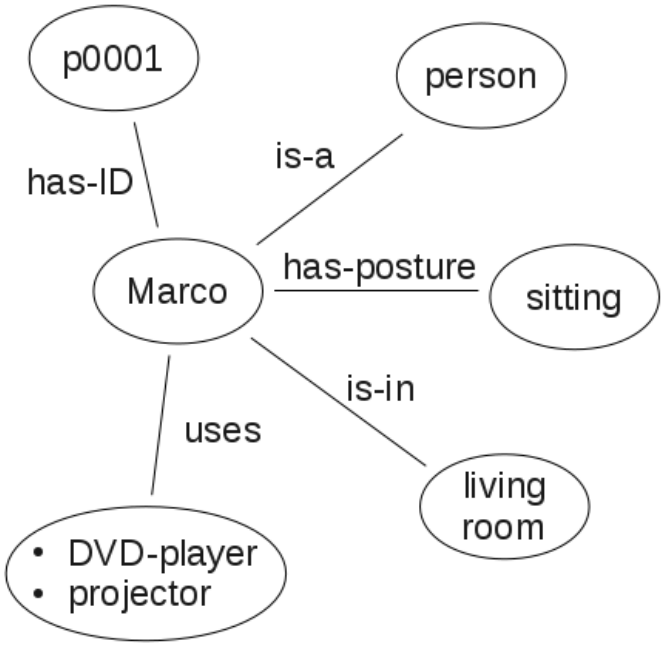

Figure 5.8: Contextual data represented as Semantic Networks

Furthermore, inserting additional information (such as the information contained in Table 5.3) appears to be quick and easy, since we do not need to transform our model to do that. This happens because of the properties of the graphs previously mentioned, and also because objects in one triple can be subjects in another triple. Figure 5.9 and Figure 5.10 show how different datasets can be easily merged without requiring changes in the model.

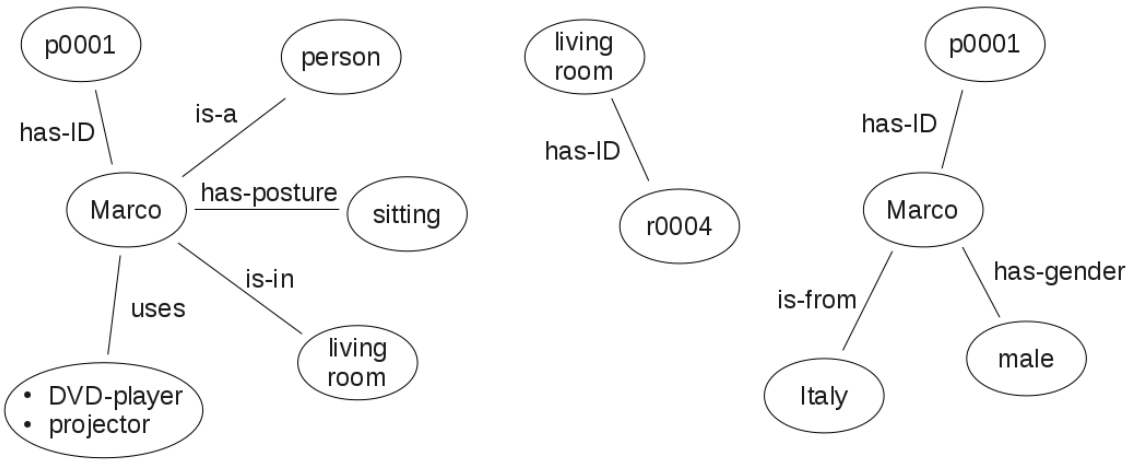

Figure 5.9: Separate datasets sharing some data

Now, for example, by following the chain of assertions, we could determine some basic information about a given person. We just have to know where to look.

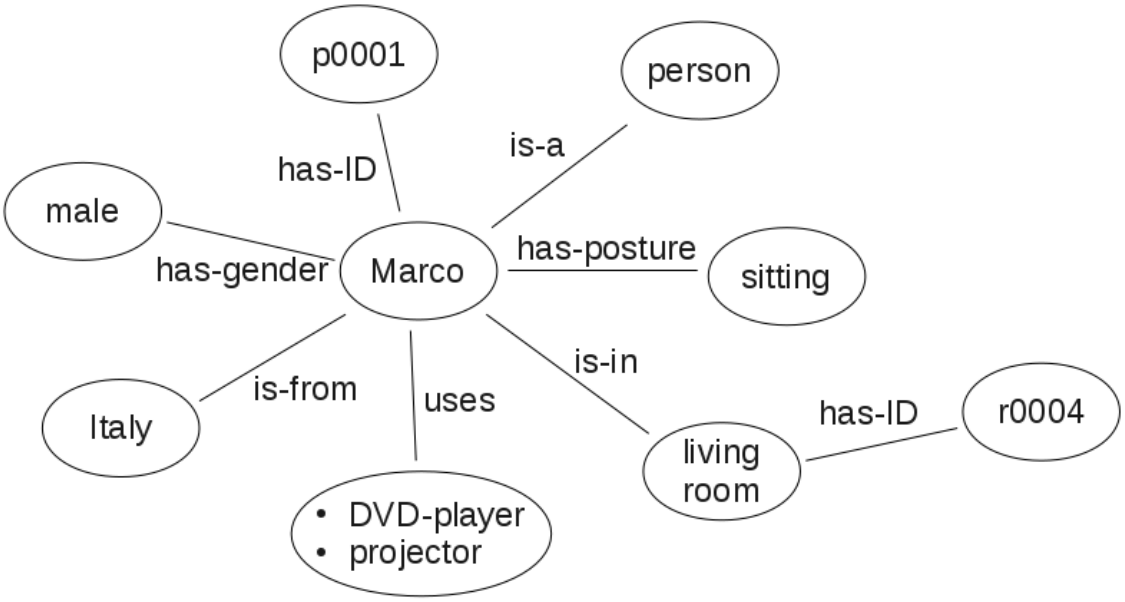

Figure 5.10: New dataset obtained by merging the existing datasets

In conclusion, after comparing pros and cons of the datasets mentioned above, we decided to use Semantic Networks as representation model to represent contextual data. In particular the advantages of Semantic Networks mentioned in Section 3.4 make them a good solution to integrate heterogeneous data characterized by varying structures from varying sources, where both schema and data evolve over the time.

5.2 Semantic Networks vs Object-Oriented model

In order to make Semantic Networks simple, modular, modifiable, extensible, maintainable, and re-usable, we chose to embed them in the Object-Oriented model, integrating nodes and links of Semantic Networks with classes, instances, and attributes of an Object-Oriented programming language. In practice we provide an API to map programming classes to classes of a Semantic Network.
Since the conceptual model and semantics of Semantic Networks differ substantially from the object-oriented paradigm, the gap between these two approaches is considerable, and traditional techniques to fill such a gap can not be applied directly.

5.2.1 Differences analysis

Classes and instances in Semantic Networks are characterized by an open-world and description logics-based semantics, whereas object-oriented type systems are closed-world and constraint based (Kalyanpur, Pastor, Battle, & Padget, 2004). This semantic gap leads to six crucial differences that our approach aims to minimize:

1. *Class inheritance*. In object-oriented systems, classes can inherit at most one superclass, whereas in Semantic Networks, classes can inherit more than one superclass. For example, in Figure 5.11 the class "engineer" inherits both the class "person" and the class "job", since an engineer is a person, but engineer is also a job.

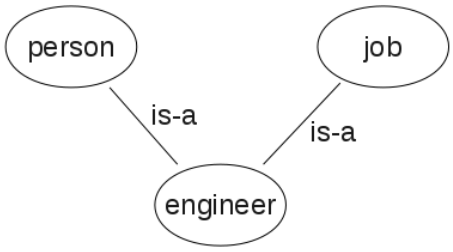

Figure 5.11: Multiple class inheritance in Semantic Networks

2. *Class membership*. In object-oriented languages, one object is member of one and only one class. Moreover, its membership is fixed (since it cannot change during the execution) and is defined during the object instantiation. In Semantic Networks, instead, a resource can belong to more than a class. For example, in Figure 5.12, the individual "Marco" is an instance both of the class "musician" and of the class "student", since Marco is a musician, but he is also a student. Furthermore, its membership is not fixed but defined by its *rdf:type* and the properties belonging to the resource.

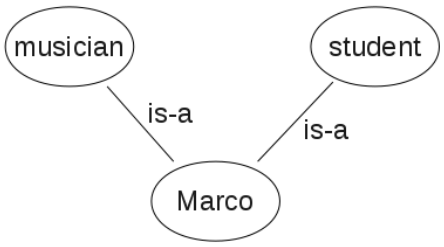

Figure 5.12: Multiple class membership in Semantic Networks

3. *Class flexibility*. Object-oriented languages generally do not allow class definitions to change at runtime. In contrast, Semantic Networks are designed right to integrate heterogeneous data characterized by varying structures from varying sources, where both schema and data evolve over the time.

4. *Object compliance*. Usually, in object-oriented languages, the structures of the instances have to perfectly match with the definitions of the classes. In Semantic Networks, instead, class definitions are not rigid and do not constrain the structures of the instances. In plain words, any resource can use any property.

5. *Attributes inheritance*. In object-oriented programming, objects inherit their attributes from their superclasses. In Semantic Networks, since properties do not belong to classes, they are not inherited, but their domains are propagated into the upwards direction of the class hierarchy.

6. *Attributes vs properties*. In the object-oriented model, attributes are defined locally inside a class, they can be used only by the instances of that class, and usually they have single-typed values. On the contrary, properties in Semantic Networks are independent entities that can be used by any resource of any class. Moreover, they can have different types of values.

5.2.2 Existing approaches

If we talk about relational databases, many object-relational mappings exist, such as Hibernate, Java Data Objects, ADO.Net and ActiveRecord (Fowler, 2002). Their approaches consist in mapping tables to classes. Table columns are mapped to class attributes, except for foreign keys, which are mapped to object relationships. Furthermore, every tuple in the relational model is mapped to an object and intersection tables[2] are mapped to object relationships (e.g., is-a relations).
This mapping methodology is not suitable when we deal with Semantic Networks, since the mismatches listed in Section 5.2.1 do not occur in relational data. Therefore, existing object-relational approaches are not able to address the problem.

[2] *Intersection tables* are introduced in the relational model to capture many-to-many relations. [msdn.microsoft.com]

5.2.3 Solution

First of all, we decided to set the bases of our approach on object-oriented scripting languages, such as Python, Perl, and Ruby, where the gap between the object-oriented paradigm and Semantic Networks is smaller than using compiled object-oriented languages.
Dynamic and general purpose scripting languages are generally interpreted, use dynamic typing, are characterized by strong *meta-programming*[3] capabilities, and give the possibility to exploit introspection at runtime (Ousterhout, 1998). These features allow us to fill the gap discussed in Section 5.2.1 and implement a domain-specific object-oriented language embedding Semantic Networks. In particular, we chose to use Python for its very clear syntax and its high code readability.
Thus, our approach solves (although in some cases just partially) the above-mentioned six differences as follows:

- *Class inheritance*. By supporting multiple inheritance, the use of Python as scripting language automatically solve the issue related to the class hierarchy.
- *Class membership*. Python, like the other scripting languages, uses dynamic typing. For this reason objects type does not require static specification; in fact, these are determined at runtime according to the capabilities of the object. Dynamic typing perfectly adapts to Semantic Networks class membership, which can also change dynamically. Moreover, even though objects in Python can have only one type at a time, we can use meta-programming and easily override that behaviour.
- *Class flexibility*. As a scripting language, Python is interpreted and is not based on strict and pre-defined classes. For this reason it constitutes a good solution to deal with flexible environments in which both data and schema can continuously change. Furthermore, the introspection provided by Python allows to investigate both schemas and data at runtime, that is a very interesting property in such a varying domain like the context one.
- *Object compliance*. Python does not require object to be strictly conformed to their class structure, allowing them to deviate from it. For example, in Python it is possible to define a different behaviour for every instance of the same class.
- *Attributes inheritance*. Actually, the lack of the attribute inheritance in Semantic Networks does not constitute a problem for our object-oriented mapping.

[3] *Meta-programming* is the writing of computer programs that write or manipulate other programs (or themselves) as their data, or that do part of the work at compile time that would otherwise be done at runtime. [wikipedia.org]

- *Attributes vs properties*. Since Python allows the use of meta-programming, we are enabled to add attributes of objects in a dynamic way. Moreover, by means of Python's dynamic typing we are allowed to assign to the attributes values of multiple types.

In conclusion by exploiting the properties of scripting languages such as dynamic typing, meta-programming, and introspection, our approach constitutes a solution to embed the Semantic Networks model into the object-oriented one. In this way we are able to exploit the high expressiveness of the Semantic Networks and all the advantages related to the Object-Oriented model, such as simplicity, modularity, modifiability, extensibility, maintainability, re-usability.

5.3 Weak spots of the approach

In this section we list six points that, according to us, constitute a weakness of this approach. As is clear, most of them are directly related to the limitations characterizing Semantic Networks (see Section 3.4).
The first three weak spots, in particular, are due to the graphical nature of the Semantic Networks. For this reason they actually do not affect our model, since, as shown in Chapter 6, we treat Semantic Networks as directed graphs, and not as a graphical representation.

1. Showing all the different inferred information using a network can be difficult.
2. Representing the context with Semantic Networks is less reliable than doing it with other knowledge representation techniques because inferring becomes a process of searching across a diagram.
3. When we deal with huge amounts of data, diagrams can become very complex. Therefore, this model may suffer scalability problems.
4. The wide range of possible types of links and the ways they might combine to form indirect links, plus the large number of concepts usually included in a Semantic Network, make this approach susceptible to a combinatorial explosion. However it is possible to mitigate this problem by dividing the model in several sub-models, avoiding, in this way, to deal with huge amounts of data.
5. The meaning of nodes and links in Semantic Networks is not clear and only defined procedurally by the inference algorithms. This weak spot is addressed as future work in Chapter 8.

6. This model requires knowledge of the problem space. This is actually a limitation related to Semantic Networks and ontologies, and in this thesis we do not propose any mitigation to it.

5.4 Summary

In this chapter we explored the main representation models for context, analysing their strong and weak spots and explaining the reasons why we finally preferred Semantic Networks. Moreover, we described in details the problem statement and how our approach addresses it. Finally, we pointed out the weak spots characterizing our methodology.

In the next chapter we are going to introduce CAFE, a contextual infrastructure based on the integration of Semantic Networks with the Object-Oriented model, describing how we implemented it and listing its main features.

Chapter 6

CAFE: a context-aware infrastructure

On the basis of the approach presented in Chapter 5, we created a direct, quick-to-understand, and easy-to-use solution to build highly-readable, flexible, scalable, general-purpose, and modular context-aware applications.
CAFE[1] (Context-Awareness Factory for Entities) is a context-aware infrastructure written in Python, that exploits the considerable power of the Object-Oriented model, and the strong expressiveness and the high readability of Semantic Networks.

Figure 6.1: CAFE's logo

CAFE is thought to work both with real and with virtual environments and aims to provide the entities present in a given environment a clear and precise knowledge of the context

[1] Source code available at https://github.com/dapids/Context-Awareness-Factory-with-Entities.

related to that environment, independent of the type of sensors used (adding or removing sensors, for example, does not affect the system).
Let us imagine, for example, smart appliances which have the task of automatically managing actions and reactions of a house according to the context. Thanks to CAFE, we can give the appliances independent behaviours, making them capable of auto-managing their activities and auto-configuring themselves when necessary, by auto-adapting to the context in which they are at a given moment. An independent behaviour guarantees continuous operation of the appliances, since they do not need a human intervention to be managed. This could lead to a considerable saving of time, money, and effort.
In practice, in order to guarantee an independent behaviour, we simply make the appliances aware of what is happening around them (context-awareness), so that they can act and react in an appropriate way.

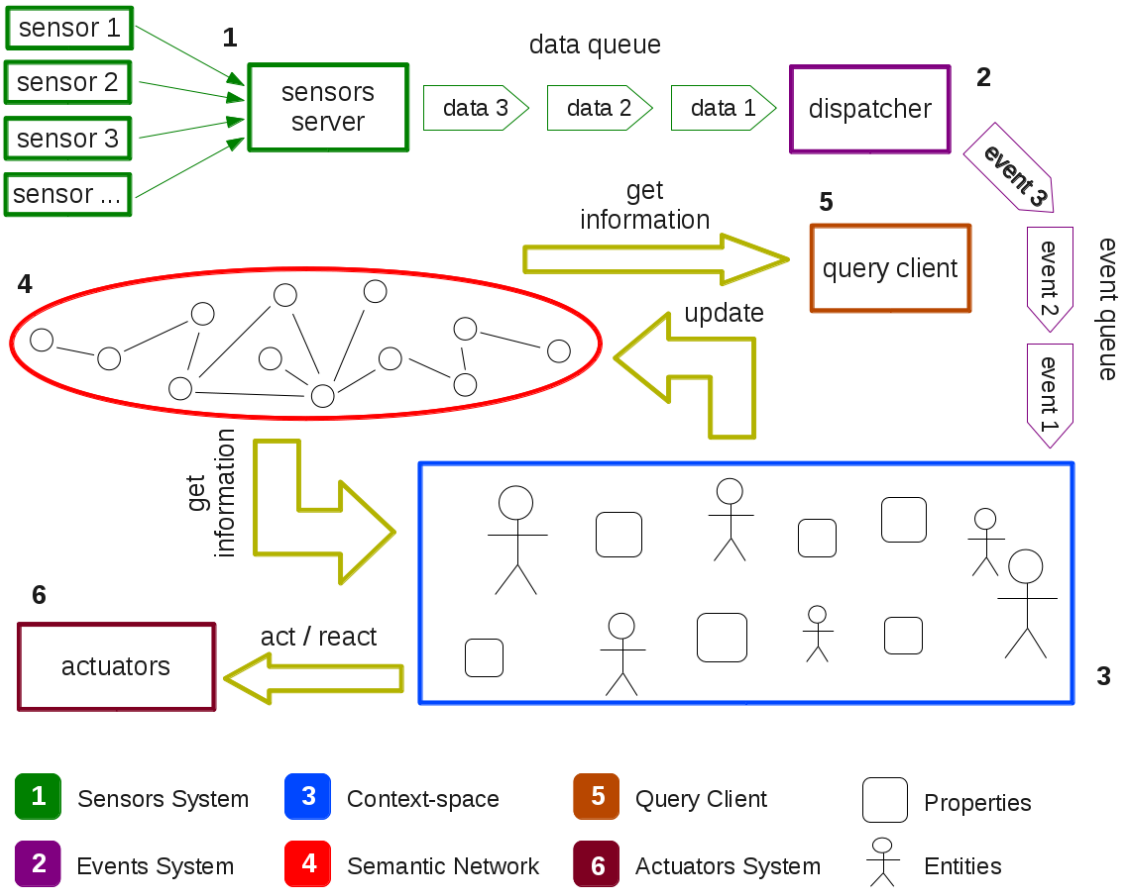

Figure 6.2: CAFE's generic schema

Figure 6.2 constitutes a generic snapshot describing how CAFE works. Data from sensors are gathered by means of a server and organized in a queue. A dispatcher reads data from this queue and generates an event for each of them. Every entity is subscribed to events regarding itself and has the task to update information about itself in the Semantic

Network. For example, when the sensors recognises the presence of a person in a room, the entity corresponding to that person updates the information about its location in the Semantic Network. In this way every entity does its part to keep the knowledge about itself up-to-date. Notice that we may also arbitrarily make an entity subscribe to a given event and react to it in a given way. For example, we could make the Light Manager System subscribe to events regarding the presence of people inside rooms, in order to switch a light off when somebody leaves a room. It is important to underline that every entity may access the whole knowledge contained in the Semantic Network, or just a part of it. Entities can also perform reasoning on the Semantic Network, in order to infer additional knowledge. In this way we provide context-awareness to the entities, since they are completely or partially aware of what is happening around them. Since entities are provided with context-awareness, they can act/react consequently and modify the environment by means of actuators[2]. Finally, CAFE also provides a way to get information from the network by means of an external client, but this aspect will be deepened in Section 6.4.
Figure 6.3 shows a UML Sequence Diagram describing a simple scenario in CAFE, where a camera notices that a person identified as "Marco" is leaving a room identified as "kitchen".

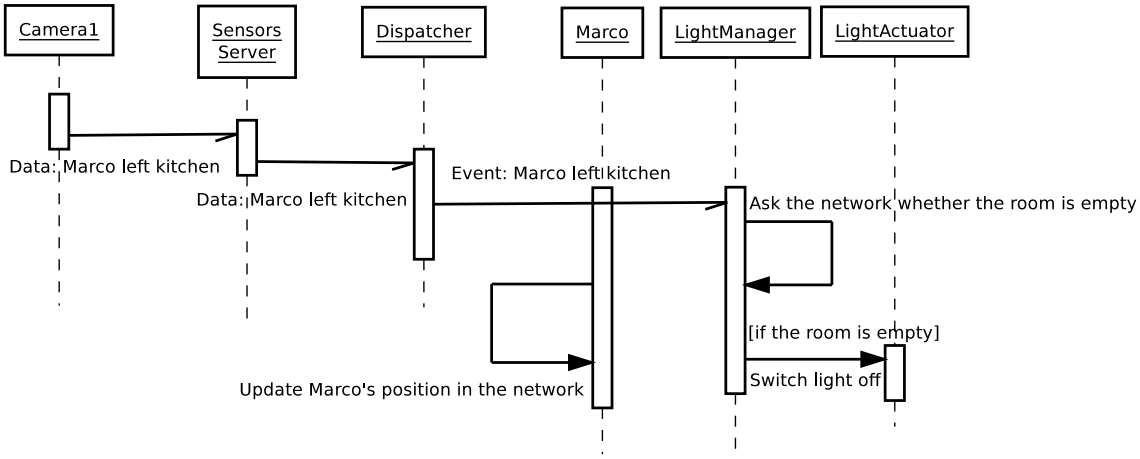

Figure 6.3: UML Sequence Diagram of a simple scenario in CAFE

In this case the Sensors Server gets to know that Marco is leaving the kitchen and forwards the information to the Dispatcher. The Dispatcher, in turn, generates the event related to that information. The entity "Marco" and the entity "LightManager" are subscribed to that event, therefore they are aware of it. At this stage, the entity "Marco" updates the information about its location in the Semantic Network (deleting the triple "Marco;is-located-in;kitchen", or probably simply updating it with a new location). The

[2] An *actuator* is the mechanism by which an agent acts upon an environment. [wikipedia.org]

LightManager, on the other hand, asks the network whether the room Marco has left is now empty; in that case it provides for switching off the light.

6.1 Context-Spaces, entities and properties

In CAFE the context is considered a space, which takes the name of *Context-Space*. This space contains all the *entities* and the *properties* characterizing the context. Regarding the concept of entity, following the definition of entity given in Section 2.1, in our approach we consider an entity "whichever person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves". Regarding the concept of property, in our approach we consider a property "whichever relationship among entities that belong to the context".

The Context-Space, in turn, can be divided in a set of *local-contexts*, where every local-context describes a part of the global context related to the environment. The Context-Space, therefore, can be seen as a puzzle, and the local-contexts as the pieces composing that puzzle. Local-contexts can be used to describe the context from different points of view. For example, we can define a local-context for the part of the context regarding the geographical locations, another one for that part regarding the actions performed, and another one for that part regarding the information about the entities. In practice, local-contexts contain all the properties related to the entities that populate a given environment. The benefit of having different local-contexts is clear, since in this way it is possible to access even only a small part of the context, making the research and the management of information more accurate and easier.
Figure 6.4 shows an example of a Context-Space describing a home environment. The Context-Space shown contains a series of entities and local-contexts, in turn containing a series of properties which constitute the relationships among the above-mentioned entities.

Looking from the point of view of the Object-Oriented model, Context-Spaces and local-contexts are represented as simple objects, containing information about entities and properties related to the context, respectively. Entities and properties are, in turn, objects, and more specifically attributes of a Context-Space.
Considering the Semantic Networks point of view, instead, the Context-Space is seen as a graph, where entities and properties correspond to nodes and links, respectively. As we already remarked in Chapter 3, this type of representation is redundancy-free, therefore ambiguous situations due to some repetitions of information are automatically avoided.

Moreover, thanks to the graph properties, it is possible to merge two or more graphs practically without any effort, which allows us to bind information coming from different Context-Spaces in order to have an even wider view of the context.

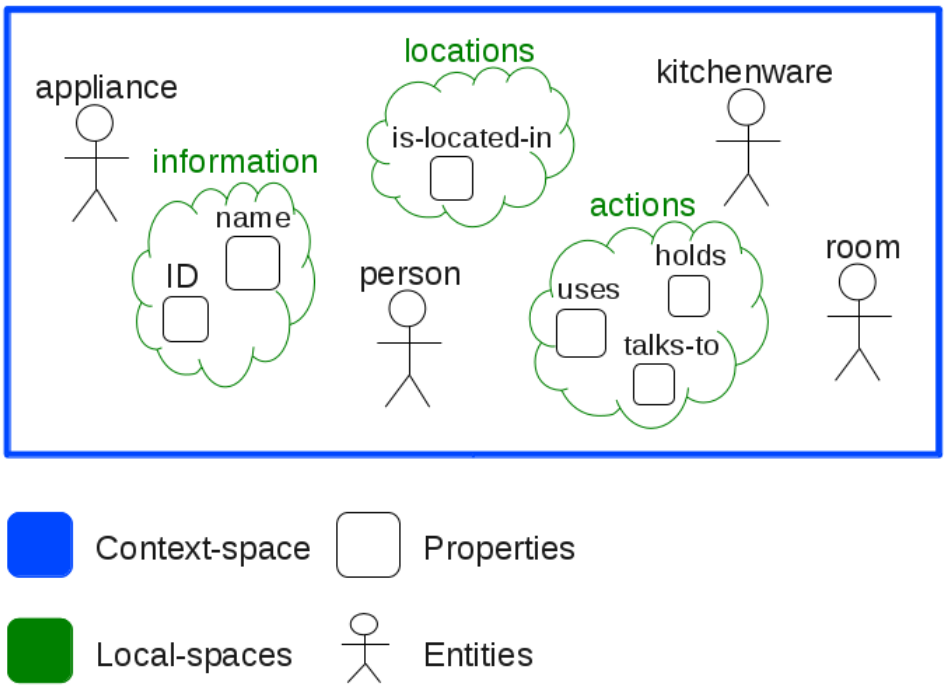

Figure 6.4: A Context-Space describing a home environment

In this section we show how to accomplish the main tasks in CAFE, in order to manage the context related to a given environment by exploiting the combination of Semantic Networks and Object-Oriented model.

With CAFE it is possible to create a Context-Space in such a simple and intuitive way. Listing 6.1 shows how to create a Context-Space simply importing the class ContextSpace and instantiating it.

Listing 6.1: Creation of a Context-Space

```
import ContextSpace
contextspace = ContextSpace("http://cafe.ns/home-environment#")
```

To instantiate a new Context-Space we need to specify a *namespace*[3] as parameter. Namespaces are thought to be unique, and they can be seen as a label attached to the environments. Thanks to namespaces, in fact, resources belonging to different environments can be distinguished with a unique identifier sanctioning the belonging to their

[3] In general, a *namespace* is a container for a set of identifiers (names), and allows the disambiguation of homonym identifiers residing in different namespaces. [wikipedia.org]

own environment. For example, the resources “http://cafe.ns/marco-home#fridge” and “http://cafe.ns/andrea-home#fridge” are different, since one fridge belongs to Marco’s environment, and the another one to Andrea’s environment.

Creating an entity is also an easy task in CAFE. In fact, Listing 6.2 shows that it is enough to import the class we want to instantiate, obviously, and call the method “createEntity” from an object of type “ContextSpace”, specifying the name of the entity to create.

Listing 6.2: Creation of an entity

```
import Person
marco = contextspace.createEntity(Person, "Marco")
```

Starting from its creation, every entity and all their properties will be mapped on a Semantic Network.

Before creating a property we need to create a local-context to contain it. As shown in Listing 6.3, creating a local-context is possible by means of the method “createLocalContext”, called from an object of type “ContextSpace”.

Listing 6.3: Creation of a local-context

```
info = contextspace.setLocalContext("information")
```

After creating a local-context we are able to attach to it some properties. This operation is possible by calling the method “defineProperties” from an object of type “LocalContext”, specifying the name of the property, its cardinality, and whether it is a symmetric property or not. In detail, since we previously created a local-context containing all the personal information about entities, in Listing 6.4 we create some properties related to it.

Listing 6.4: Creation of some properties

```
info.defineProperties(("name", 1), ("age", 1), ("isCloseTo", 0, "s"))
```

A cardinality “0” represents a one-to-infinite property (a person can have only one age), whereas a cardinality “1” represents a one-to-one property (a person can be close to more that one person). Moreover, by inserting “s” in the definition of the property we can define a symmetric property (the fact that Marco is close to Andrea means that Andrea is close to Marco).

Moreover, CAFE provides an easy way to serialize the information contained in a Semantic Network to *RDF*[4], *RDF/XML*[5], *turtle*[6], *n triples*[7], *n3*[8]. They are all knowledge representation formats that support the inference of additional knowledge by means of reasoning. Therefore, by calling the method "serialize" from an object of type "ContextSpace" (as shown in Listing 6.5), it is possible to serialize all the information representing the current context to the above-mentioned formats.

Listing 6.5: Serialization of the context

```
contextspace.serializeContext(format="n3")
```

After serializing the context, we may use the method "writeOntology" to store the ontology either on a local memory or on a FTP server. For instance, Listing 6.6 shows how to store the ontology produced by CAFE on a FTP[9].

Listing 6.6: Write the context as an OWL ontology on a FTP server

```
contextspace.writeOntology(contextspace.serializeContext(), "ftp")
```

In this way we are free to access it from every networked device and reason on it using an external OWL reasoner.

In addition CAFE allows us to make our knowledge about a given environment persistent, simply storing the graph containing it into a binary file. This can be made by calling the method "saveContext" from a Context-Space object (see Listing 6.7).

Listing 6.7: Save the context in a binary file

```
contextspace.saveContext("/home/david/myContextSpace")
```

This feature allows us to take a snapshot of the context related to a given environment in a given moment. In this way we might have a history of the context over the time, that gives us the possibility to reason on events happened in the past.

[4] The *Resource Description Framework* (RDF) is a family of World Wide Web Consortium (W3C) specifications originally designed as a metadata data model. [wikipedia.org]

[5] *RDF/XML* is a syntax, defined by the W3C, to express (i.e. serialize) an RDF graph as an XML document. [wikipedia.org]

[6] *Turtle* (Terse RDF Triple Language) is a serialization format for Resource Description Framework (RDF) graphs. [wikipedia.org]

[7] *N-Triples* is a format for storing and transmitting data. It is a line-based, plain text serialization format for RDF (Resource Description Framework) graphs, and a subset of the Turtle (Terse RDF Triple Language) format. [wikipedia.org]

[8] *Notation3*, or *N3* as it is more commonly known, is a shorthand non-XML serialization of Resource Description Framework models, designed with human-readability in mind: N3 is much more compact and readable than XML RDF notation. [wikipedia.org]

[9] The address and access data of the FTP have to be specified in an external configuration file.

Finally, CAFE provides us the possibility to visualize the context from a graphical point of view, showing the respective Semantic Network as an image. This functionality can be really useful if we need to have a highly-readable representation of what is happening in a given environment. As shown in Listing 6.8, this feature can be used by calling the method "visualizeContext" from an object of type "ContextSpace".

Listing 6.8: Visualization of the context

```
contextspace.visualizeContext()
```

6.2 Architecture

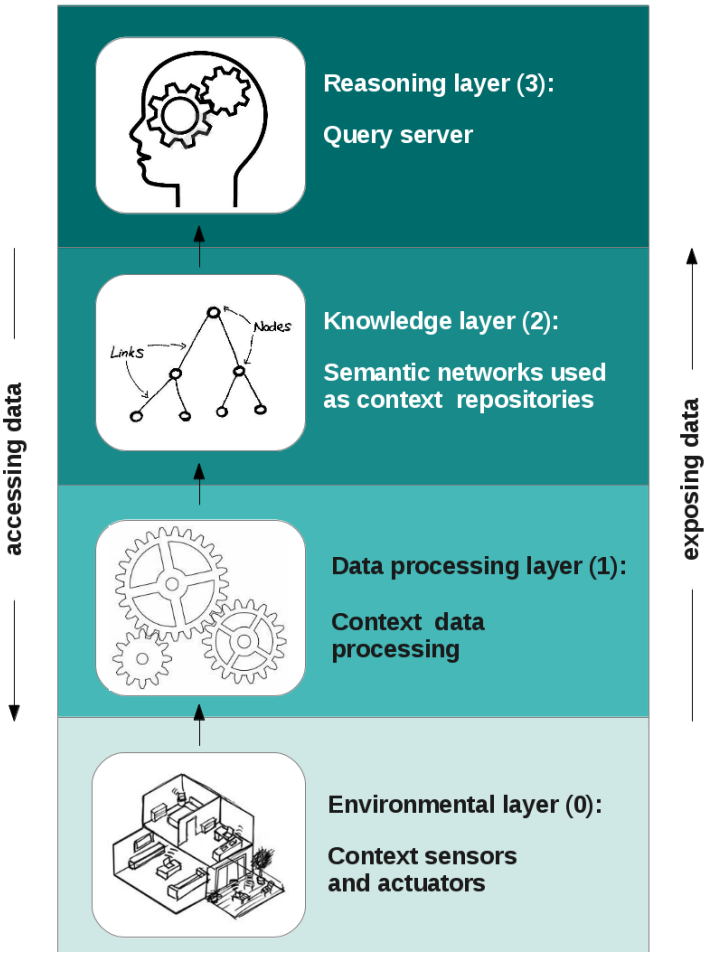

Figure 6.5: CAFE's architecture

As shown in Figure 6.5, CAFE's architecture consists of four layers. As is visible, in our approach entities do not manage contextual data in an isolated way and do not have the task to analyse them. They simply access information already properly structured

and organized, which represent a knowledge related to a situation of interest. Therefore, entities do not directly access data, but they do it by means of a middle level (knowledge layer), in which contextual information is organized in a sort of structured sets of atomic data related among them, that are the Semantic Networks. Exploiting the expressiveness of the Semantic Networks, entities do not need to access raw data from sensors to reach context-awareness. Thus, the knowledge organization and the analysis is externalized in a separate layer, and entities access information pre-digested and in a shape that is simpler to use.

6.2.1 Environmental layer

The *environmental layer* represents the lowest abstraction level of CAFE's architecture. At this level data are gathered from different sensors located in the environment. Our approach is independent from the number and type of sensors used. Let us think, for example, about the possibility to track the user's position inside a room: for this purpose we could use a kinect[10], or some cameras, or a pressure sensor on the ground. CAFE is designed so that our choice, whichever it is, does not require any modification to the system.
Actuators also belong to this layer. By means of them entities can act/react according to context, modifying their own environment.

6.2.2 Data processing layer

In the *data processing layer*, the data gathered from sensors are processed, analysed and used to build a Semantic Network. The Semantic Network is updated every time a new event occurs. In this way, the network represents a literal snapshot of the context related to a given environment in a given moment. The basic idea behind this layer is to continuously and autonomously analyse atomic data from sensors, in order to aggregate them and build a series of relationships among them, giving them the exact meaning. This task is accomplished without a direct human intervention. Moreover, it is general and flexible, since it can be adapted to a wide range of data types.

[10] *Kinect* is a motion sensing input device by Microsoft for the Xbox 360 video game console and Windows PCs. [wikipedia.org]

6.2.3 Knowledge layer

The *knowledge layer* is composed of a Semantic Network used as repository of information. Using a Semantic Network allows us to describe the context in a semantic way, disregarding whatever programming language, operating system, or middleware. However, the major benefit related to this model is that it facilitates the formal analysis of the knowledge. Thanks to this approach, in fact, it is possible to exploit reasoning methodologies which use first-order logic[11], temporal logic[12], etc. As already pointed out in Chapter 3, the knowledge is stored as a set of triples, where every information has the form *subject-predicate-object*, for example "David is-located-in kitchen".

6.2.4 Reasoning layer

The *reasoning layer* constitutes the highest abstraction level of the CAFE's architecture. This layer offers the possibility to query the Semantic Network in order to get a clear and precise knowledge about the context related to a given environment. By querying the Semantic Network, entities simply improve their knowledge about what is happening around them in a given moment. The queries to the Semantic Network are performed using SPARQL, a standard query language characterized by a simple syntax and a high expressiveness. We are going to speak more in details about the CAFE's reasoning features in Section 6.4.

6.3 Resource mapping

CAFE maps Python classes to classes of a Semantic Network, Python objects to individuals of a Semantic Network, and attributes on Python objects to links of a Semantic Network. In this section we explain in details how the above-mentioned mappings work. All the mapping features are handled by using RDFlib, a Python package intending to provide core RDF types and interfaces for working with RDF.

[11] *First-order logic* is a formal system used in mathematics, philosophy, linguistics, and computer science and it is also known as first-order predicate calculus, the lower predicate calculus, quantification theory, and predicate logic (a less precise term). [wikipedia.com]

[12] In logic, the term *temporal logic* is used to describe any system of rules and symbolism for representing, and reasoning about, propositions qualified in terms of time. [wikipedia.org]

6.3.1 Classes mapping

In CAFE every Python class belonging to a Context-Space corresponds to a node in the Semantic Network representing that same Context-Space.
Thanks to the use of meta-programming, the mapping is performed in an automatic way. The only care we must take is to inherit the class "Entity" for each class we want to insert into the Semantic Network. For example, Listing 6.9 shows how to map the class "Student" represented in the UML class diagram in Figure 6.6.

Listing 6.9: Mapping a class in CAFE

```
import Student
```

It is evident how intuitive the mapping is; in practice, it is enough to perform a normal Python import of the class we want to insert in the network, and CAFE will do the rest.

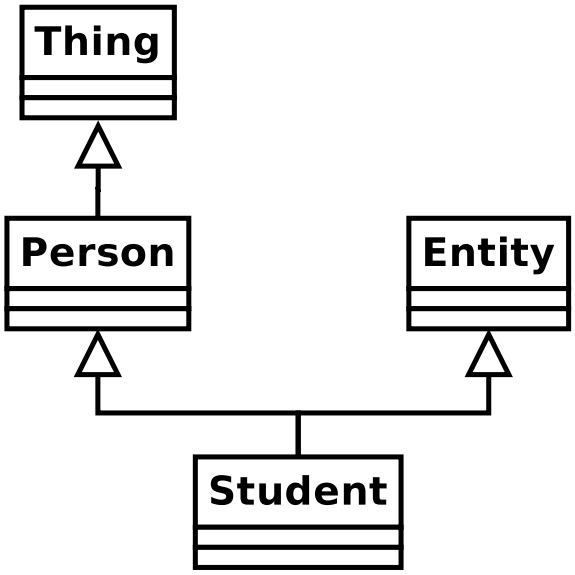

Figure 6.6: UML Class Diagram of an illustrative class mapped in CAFE

As the result in Figure 6.7 shows, CAFE maps the class "Student" inheriting "Entity", and also all its superclasses, building a hierarchy by means of "is-a" links.

Figure 6.7: Result of a class mapping in CAFE

6.3.2 Objects mapping

As for the classes, in CAFE every Python object belonging to a Context-Space corresponds to a node in the Semantic Network representing that same Context-Space. As already pointed out in Section 3.4, in fact, Semantic Networks treat classes and individuals in the same way, without distinguishing between them.
As shown in Listing 6.10, the mapping of objects can be performed by using the method "createEntity", which takes as parameters the class and the name of the individual to be mapped. CAFE checks whether an individual with the same name already exists in the same Context-Space, and whether the class which the individual belongs to inherits the class "Entity". Obviously, if the class which the individual belongs to has not been mapped yet, CAFE automatically provides for mapping it as previously explained in Section 6.3.1.

Listing 6.10: Mapping an individual in CAFE

```
marco = contextspace.createEntity(Student, "Marco")
```

6.3.3 Attributes mapping

In CAFE attributes of Python objects belonging to a Context-Space correspond to links in the Semantic Network representing that same Context-Space.
In Section 6.1 we saw how to define a property, corresponding to a link in the Semantic Network. But if we want to use this property, we also need to specify which classes are going to use that property, that corresponds to specifying which nodes are going to use that link. For this purpose, as shown in Listing 6.11, in CAFE we use the normal Python syntax to connect two classes and a property, that correspond to two nodes and a link from the Semantic Network point of view.

Listing 6.11: Specifying a link's domain and range in CAFE

```
Person.NAME = str
Person.AGE = int
Thing.ISLOCATEDIN = Room
```

As is visible from the previous example, it is enough to consider the property as an attribute of the subject class and assign to it the respective object class. In this way we create a *backbone* of the Semantic Network, in which all the domains and ranges of the properties are clearly specified. The backbone corresponding to Listing 6.11 is represented in Figure 6.8.

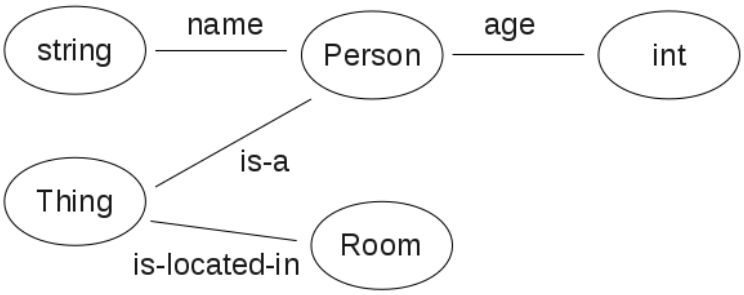

Figure 6.8: An example of Semantic Network backbone in CAFE

It is important to notice that a backbone, apart from giving to the network a precise structure and avoiding misunderstandings, is also useful to reason on, since it may contain many relevant information.
Once the backbone has been defined, CAFE is ready to automatically map on the Semantic Network the attributes of the Python objects belonging to the respective Context-Space (also known as entities). To do that, as Listing 6.12 shows, we simply need to assign a value to the attribute we want to map on the network.

Listing 6.12: Mapping a link in CAFE

```
marco.AGE = 26
```

Obviously, looking at backbone, CAFE will make sure that the type of the assigned value corresponds to the expected one.
The property taken into account in the previous example is a property with cardinality "1", therefore we can not assign more than one object to it (Marco has only one age). But let us consider the property "talks-to" and the triple "Marco;talks-to;Andrea". What happens if, while Marco is talking to Andrea, Rose approaches Marco and joins the talk? In that case our triple should become something similar to "Marco;talks-to;Andrea,Rose". Well, CAFE automatically manages properties with multiple cardinality, without requiring changes in the syntax used to map a single property. In fact, in that case CAFE uses a list to store all their objects. However, thanks to meta-programming, this is hidden to our eyes, since we can go on treating these properties in the same way as we treat properties with cardinality "1". In this respect, Listing 6.13 shows the mapping of a property with multiple cardinality.

Listing 6.13: Mapping a link with multiple cardinality in CAFE

```
marco.TALKSTO = andrea
marco.TALKSTO = rose
```

It is important to point out that, since the property “talks-to” is also symmetric, CAFE will manage it creating symmetric links among the nodes related to that property. This means that we just need to know that Marco is talking to Andrea and Rose, and CAFE will know that, consequently, Rose is talking to Marco and Andrea, and Andrea is talking to Marco and Rose.
But what happens if Rose eventually gets bored and decides to leave the conversation with Marco and Andrea? In that case we would probably need to remove from the network the triple “Rose;talks-to;Marco,Andrea”. Well, CAFE allows us to do it simply by using the normal Python syntax for deleting an attribute, as shown in Listing 6.14.

Listing 6.14: Removing a link between two or more nodes in CAFE

```
del rose.TALKSTO
```

Instead, let us imagine that Rose decides to leave only the conversation with Andrea, but she goes on talking to Marco. In that case, CAFE allows us to remove only the triple “Rose;talks-to;Andrea” simply by specifying the object to remove, as shown in Listing 6.15.

Listing 6.15: Removing a link between twos nodes in CAFE

```
del rose.TALKSTO[andrea]
```

Last but not least, let us imagine that today is Marco’s birthday and we want to update his age. In this case we can just assign a new value to Marco’s age, as shown in Listing 6.16, without worrying about deleting the previous age. Indeed, CAFE knows that “age” is a property with cardinality “1”, and it will automatically replace the old age with the new one. Obviously, this does not work with properties with multiple cardinality, since CAFE will just add the new value to the existing list of old values.

Listing 6.16: Updating a link between two nodes in CAFE

```
marco.AGE = 27
```

6.4 Inference and reasoning

As shown in Section 6.2.4, CAFE also provides reasoning features, allowing us to query the network in order to get information about the context stored in it. Queries have to be formulated in SPARQL (Simple Protocol and RDF Query Language), a query language

that matches patterns in the graph and binds wildcard character[13] variables as it finds solutions.

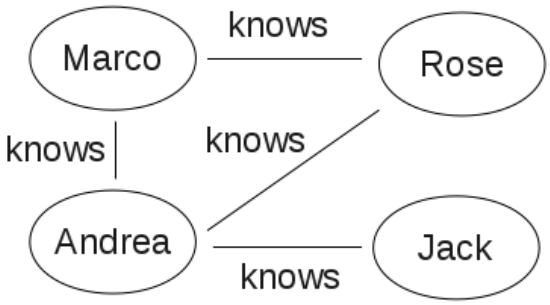

Figure 6.9: Example of a Semantic Network

Considering the Semantic Network shown in Figure 6.9, a plain example of SPARQL query could be the one in Listing 6.17.

Listing 6.17: Example of a SPARQL query

```
base <http://cafe.ns/home-environment#>
SELECT DISTINCT ?aname ?bname
        WHERE {
        ?a :knows ?b .
        ?a :name ?aname .
        ?b :name ?bname .
    }
```

The result of this query (see Listing 6.18) is simply the name of all the people knowing other people:

Listing 6.18: Example of a SPARQL query

```
Marco knows Rose
Marco knows Andrea
Rose knows Marco
Rose knows Andrea
Andrea knows Marco
Andrea knows Rose
Andrea knows Jack
Jack knows Andrea
```

[13] A *wildcard character* can be used to substitute for any other character or characters in a string. [wikipedia.org]

Listing 6.17 shows just a plain example, but SPARQL is a power tool and it provides plenty of advanced features to extract information from the network and infer new knowledge in it.
Entities belonging to a Context-Space can directly query the Semantic Network to get useful data by using the method “ask”, as shown in Listing 6.19.

Listing 6.19: Example of a query performed by an entity

```
response = self.ask(query)
```

However, every Context-Space in CAFE is also equipped with a server aiming to serve requests of queries from outside, and an external client, which allows others to contact the server and ask for information about the context. Therefore, by using the client, an external user having the right privileges can query the Semantic Network in order to understand the context.
Once we queried the network for information, CAFE also allows us to convert that information into a new set of triples that can be added to the network. In this way it is always possible to create new knowledge, simply by performing queries.
Also the management of the SPARQL networking of queries has been implemented by using the already mentioned RDFlib, which provides useful and powerful API for that.

6.5 Summary

In this chapter we introduced CAFE, a contextual infrastructure based on the integration of Semantic Networks with the Object-Oriented model, describing how we implemented it and listing its main features.

In the next chapter we are going to propose an illustrative scenario in order to show how it is possible to use CAFE to represent and manage the context related to that scenario. We will also use this demonstration to underline the results we achieved and point out the limitations of CAFE.

Chapter 7

Results

In the previous chapter we introduced and described CAFE, a context-aware infrastructure based on the integration of Semantic Networks with the Object-Oriented model. In this chapter, we propose an illustrative scenario in order to show how it is possible to use CAFE to represent and manage the context related to that scenario. Furthermore, we exploit this demonstration to underline the results we achieved and to point out the limitations of CAFE.

7.1 Scenario

The scenario presented in this chapter is just a plain scenario about a home environment, but it is a good example to easily understand how CAFE works and which benefits we can get from it. The scenario includes a house, some people living in it, and some objects that you can typically find in a house.

Our main idea is to build context-aware application able to manage the context related to a given environment. For this purpose, the first step is to individuate the entities belonging to the environment and classify them in classes. Then, the following step is to individuate the properties of the entities and the relationships among them, and to represent them as attributes of the above-mentioned classes.
Let us consider the following entities and properties for the scenario taken into account:

- **Child, Adult, Elderly**. We decided to distinguish among children, adults, and elderly people, since they usually tend to behave differently. However, they share the following properties: ID, name, location, age, gender, posture, temperature in Cel-

sius, blood pressure, people they are talking to, appliances they are using, objects they are holding.

- **Appliance**. All the household machines, using electricity or some other energy input. They have the following properties: ID, name, location, description, status (e.g., on, off, stand-by, etc).
- **Kitchenware**. Utensils, dishes, and other tools made to be used in the kitchen. They have the following properties: ID, name, location, description.
- **Generic Object**. All the objects that are not appliances or kitchenware. They have the following properties: ID, name, location, description.
- **Room**. The rooms of the house taken into account. They have the following properties: ID, name, location, temperature in Celsius.

The UML Class Diagram in Figure 7.1 shows how the scenario may be modelled according to the Object-Oriented model.

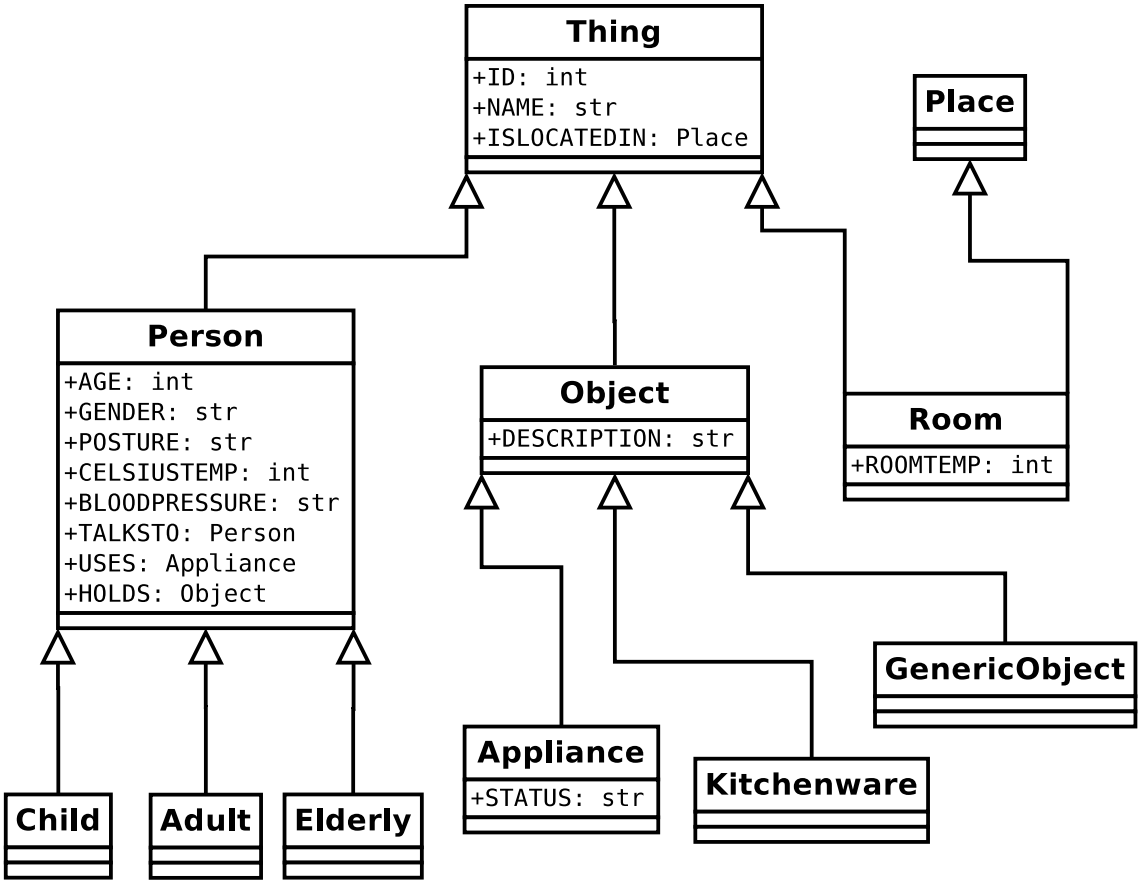

Figure 7.1: UML Class Diagram of the scenario

7.2 Demonstration

Once we individuated classes and properties characterizing the environment which we are working on, we can create the Context-Space that will represent the context of the home environment taken into account.

```
>>> from cafe.contextSpace import ContextSpace
>>> cs = ContextSpace("http://cafe.ns/home#")
Creating new context-space: HOME
Graph: [a rdflib:ConjunctiveGraph;rdflib:storage [a rdflib:Store;rdfs:label 'IOMemory']]
Global namespace: http://cafe.ns/home#
>>> 
```

Figure 7.2: Creation of a new Context-Space in CAFE

As shown in Figure 7.2, when we create a new Context-Space, CAFE initializes the Semantic Network that will be used to represent the context, associating to it a namespace. As already mentioned in Chapter 5, the Context-Space is nothing but a graph, that means that we can merge two or more Context-Spaces without any effort. This feature underlines the scalability of our approach. In fact, we might choose to divide a given context in more sub-contexts and manage them separately, and finally merge the sub-contexts without any effort and consider them again as a single context during the reasoning phase.
It is important to underline that the use of Semantic Networks makes CAFE capable of managing incomplete or inaccurate information due to incompleteness of data from sensors. This shows that our approach is flexible, since it allows CAFE to adapt to lacks of information.

The next step is to define the classes of the entities populating the environment (see Figure 7.3).

```
>>> from environment import child,adult,elderly,appliance,kitchenware,genericObject,room
Init'ing class Thing
Init'ing class Person
Init'ing class Child
Init'ing class Adult
Init'ing class Elderly
Init'ing class Object
Init'ing class Appliance
Init'ing class Kitchenware
Init'ing class GenericObject
Init'ing class Place
Init'ing class Room
>>> 
```

Figure 7.3: Definition of entities classes in CAFE

Clearly defining an entity class in CAFE is easy and direct, since this task can be accomplished using the normal Python syntax to import a class. It is important to notice how CAFE automatically manages the hierarchy of the classes. The class “Person”, for exam-

ple, is not mentioned in the import command, however CAFE initializes it, since an adult is also a person.

Figure 7.4 shows the next step, that is to define the entities present in the environment, which are supposed to belong to the previously defined classes. Notice how every entity we create starts to be mapped on the Semantic Network, constituting a piece of our knowledge.

```
>>> rose = cs.createEntity(Adult, "rose")
Mapping individual: 'rose' (Adult)
>>> marco = cs.createEntity(Adult, "marco")
Mapping individual: 'marco' (Adult)
>>> andrea = cs.createEntity(Adult, "andrea")
Mapping individual: 'andrea' (Adult)
>>> kitchen = cs.createEntity(Room, "kitchen")
Mapping individual: 'kitchen' (Room)
>>> livingRoom = cs.createEntity(Room, "livingRoom")
Mapping individual: 'livingRoom' (Room)
>>> bathroom = cs.createEntity(Room, "bathroom")
Mapping individual: 'bathroom' (Room)
>>> bowl = cs.createEntity(Kitchenware, "bowl")
Mapping individual: 'bowl' (Kitchenware)
>>> spoon = cs.createEntity(Kitchenware, "spoon")
Mapping individual: 'spoon' (Kitchenware)
>>> bottle = cs.createEntity(Kitchenware, "bottle")
Mapping individual: 'bottle' (Kitchenware)
>>> broom = cs.createEntity(GenericObject, "broom")
Mapping individual: 'broom' (GenericObject)
>>> television = cs.createEntity(Appliance, "television")
Mapping individual: 'television' (Appliance)
>>> toaster = cs.createEntity(Appliance, "toaster")
Mapping individual: 'toaster' (Appliance)
>>> dishwasher = cs.createEntity(Appliance, "dishwasher")
Mapping individual: 'dishwasher' (Appliance)
>>> 
```

Figure 7.4: Definition of entities in CAFE

At this stage we could prefer to divide the Context-Space in different local-contexts, as already explained in Section 6.1, and define the properties related to each local-context. For this scenario, we chose to have three local-contexts: information, actions, and location. The first one constitutes the part of the context containing information about the entities. The second one constitutes the part of the context containing the actions performed by the entities. Finally, the third one constitutes the part of the context containing locations and movements of the entities.
Figure 7.5 shows how the local-context “actions” and the properties related to it are defined.

After defining the properties characterizing the environment taken into account, we need to assign to each of them a subject entity and an object entity. In order to do that, as shown in Figure 7.6, we can simply use the Python syntax to assign a value to a class attribute.

```
>>> actions = cs.setLocalContext("actions")
Creating new local context: ACTIONS
Graph: <http://cafe.ns/home/context#actions> a rdfg:Graph;rdflib:storage [a rdfs:label 'IOMemory'].
Global namespace: http://cafe.ns/home#
Local Namespace: http://cafe.ns/home/context#actions

>>> actions.defineProperties(("holds", 0), ("uses", 0), ("talksTo", 0, "s"))
Defining class property: holds
Cardinality: *
Characteristics: []

Defining class property: uses
Cardinality: *
Characteristics: []

Defining class property: talksTo
Cardinality: *
Characteristics: ['simmetry']
```

Figure 7.5: Example of definition of a local-context in CAFE

```
>>> Person.USES = Appliance
Mapping class property: Person -> USES -> Appliance
>>> Person.HOLDS = Kitchenware
Mapping class property: Person -> HOLDS -> Kitchenware
>>> Person.TALKSTO = Person
Mapping class property: Person -> TALKSTO -> Person
>>>
```

Figure 7.6: Example of definition of properties in CAFE

Once the contextual backbone discussed in Section 6.3.3 is ready, we might use the CAFE's feature providing a graphical overview of the context. As already pointed out in the previous chapter, this feature is accessible by invoking the method "visualizeContext" from a Context-Space object. Notice how the possibility to have a graphical representation of the context gives to the model proposed in our approach a very high readability. Figure 7.7 shows the image that CAFE produces in output to graphically represent the contextual backbone previously created.

Now that the structure of the context is ready we can start to fill it in with data gathered from the sensors. CAFE needs that all the input data are in the format subject-predicate-object, as the example in Listing 7.1 shows. For this reason we assume that every sensor is provided with an interface aiming to transform raw data to the subject-predicate-object format.

Listing 7.1: Example of input data in CAFE

```
marco talksto andrea;
```

In order to test CAFE, we decided to create a series of simulated sensors, connecting to the Sensors Server and sending to it data about the environment. These simulated sensors work in a very simple way. Each of them has a list of possible sensed data. After sleeping for a random time interval, they randomly pick from the list one of the possible sensed

data and they send it to the Sensors Server. After sending data, they come back sleeping for another random time interval.

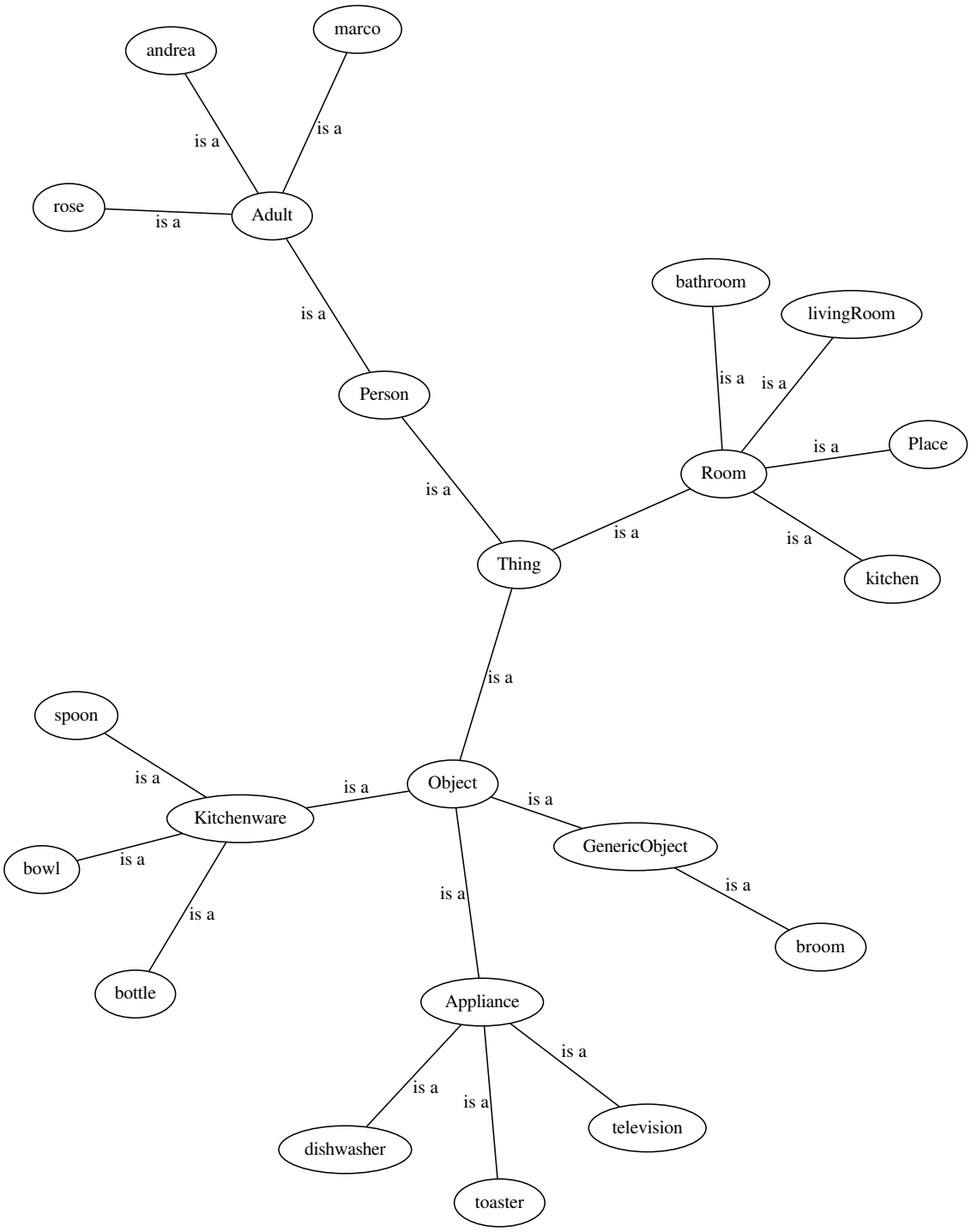

Figure 7.7: Visualisation of a contextual backbone in CAFE

In this scenario we consider seven types of sensors: cameras, microphones, RFID readers, a blood pressure sensor, a heart rate sensor, a body thermometer, and environmental thermometers. In addition, we assume that every appliance can communicate with the

Sensors Server to send data about its status (i.e., on, off, stand-by, etc). By means of cameras we could have information about people's posture and location. By using microphones and voice recognition we could know when somebody is speaking and who he/she is. RFID readers could provide us information about *who* is holding *what*. The blood pressure sensor and the body thermometer could give us information about people's health status. Finally, the environmental thermometers could let us know what is the temperature of every room.
The fact that CAFE can handle information of different types, generated by a variety of heterogeneous sources and with different levels of granularity, shows that our approach is general-purpose, since the system is able to easily change in response to different system requirements and types of data. Furthermore, CAFE also demonstrates that our approach is flexible, by guaranteeing an easy handling of the atomic data forming the context, and being capable to manage incomplete or inaccurate information due to incompleteness of data from sensors.

Figure 7.8 shows an illustrative execution of CAFE, in which simulated data from cameras are sent to the Sensors Server. Naturally the system is going to use those data to update its knowledge about the environment. In this particular case the system receives data about people's locations and posture.

```
david@alpha:~/workspace/Context-Awareness-Factory-with-Entities
File  Edit  View  Search  Terminal  Help
Query server available @ ['10.2.26.12', 7777]

Sensors server available @ ['10.2.26.12', 9999]

Type 's' to shutdown the servers or press any other key to visualise the context.

SENSORS_SERVER: got data from the client @ 10.2.26.12
Mapping individual property: 'rose' -> ISLOCATED -> 'livingRoom'

SENSORS_SERVER: got data from the client @ 10.2.26.12
Mapping individual property: 'rose' -> POSTURE -> 'standing'

SENSORS_SERVER: got data from the client @ 10.2.26.12
Deleting individual property: 'rose' -> ISLOCATED -> 'livingRoom'

Mapping individual property: 'rose' -> ISLOCATED -> 'bathroom'

david@alpha:~/workspace/Context-Awareness-Factory-with-Entities/clients
File  Edit  View  Search  Terminal  Help
Connecting to the server..
The server received 'rose islocated livingRoom;'.

Connecting to the server..
The server received 'rose posture standing;'.

Connecting to the server..
The server received 'rose islocated bathroom;'.
```

Figure 7.8: Simulated data from cameras sent to the Sensors Server

However, in this demonstration we involve all the types of sensors above-mentioned in sending simulated data to the Sensor Server. Therefore, after some time of execution, we may ask CAFE to print the context to the screen.

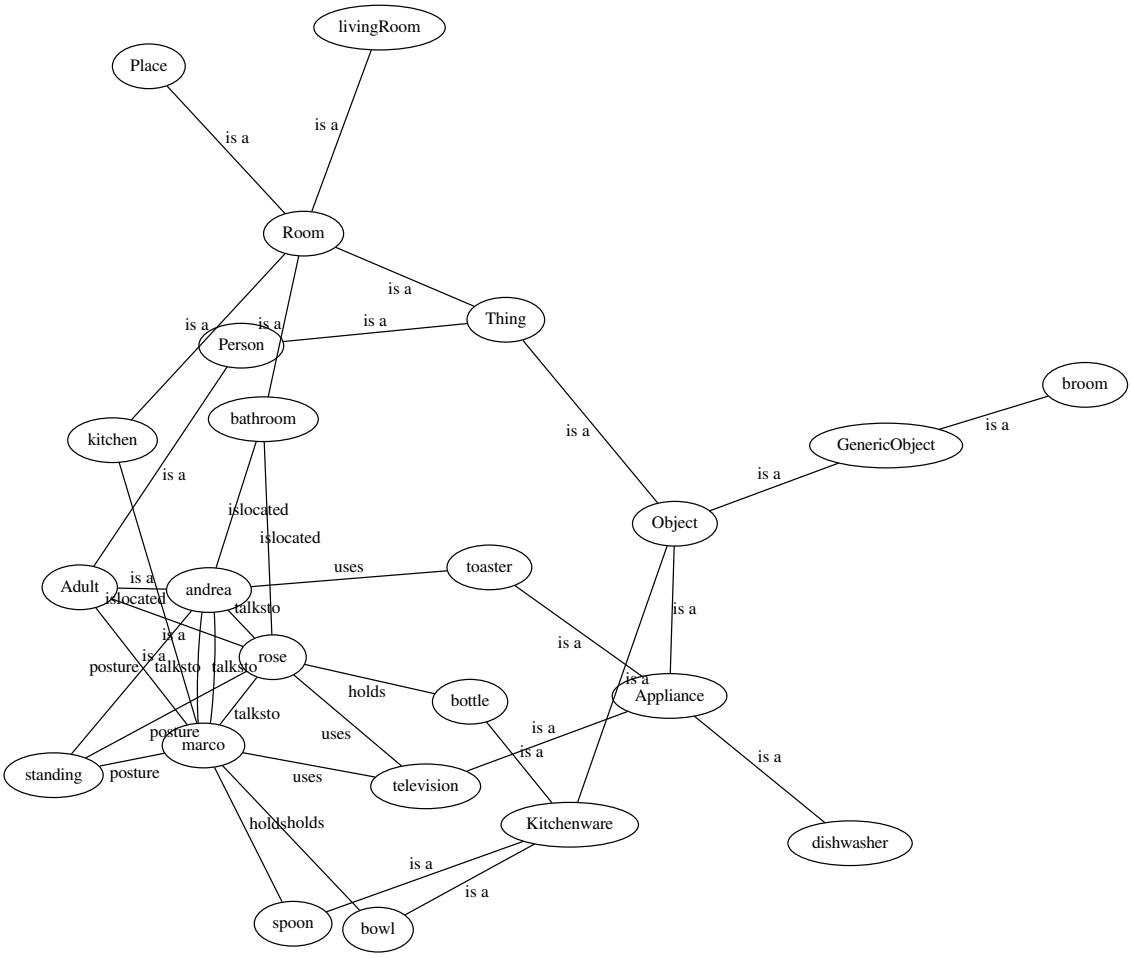

Figure 7.9: Graphical representation of the context in CAFE

As shown in Figure 7.9, the image produced by the system gives us a graphical representation of the context related to the environment taken into account. It is clear that the readability of this model decreases proportionally with the increasing of the amount of data we deal with.

At this stage all the entities have the possibility to be aware of the context (or a part of it) around them, since they can use simple SPARQL queries to reason on every information they are interested in. For example, the LightManager is probably interested in knowing whether everybody left the house, since it could switch the lights off.
However we also might be interested in accessing some information about the context related to a given environment from outside the system. In this respect, CAFE provides to us the possibility to query a Context-Space by using a client expressly implemented for

this purpose. Obviously the client can be used independently, since it does not impede the normal execution of CAFE. Figure 7.10 shows an example of a SPARQL query performed by means of the above-mentioned client. In this case, for instance, the query allows us to know the list of empty rooms, where by empty rooms we mean the rooms that contain no people.

```
david@alpha:~/workspace/Context-Awareness-Factory-with-Entities/clients
File  Edit  View  Search  Terminal  Help

:::::::::::::::::::::::::::::::::::::::::::::::::::::::
:::                Information retriever            :::
:::::::::::::::::::::::::::::::::::::::::::::::::::::::
[ctrl+D to delete your query] [ctrl+C to quit]

Please, insert your SPARQL query and be sure that it ends with '!!':

base <http://cafe.ns/home#>
SELECT DISTINCT ?roomName
        WHERE {
                ?a rdf:type :Room ;
                        :name ?roomName .
                OPTIONAL { ?b :islocated ?a } .
                ?b rdf:type     ?c .
                {?c rdfs:subClassOf :Person . } UNION {?c rdf:type :Person . }
                FILTER ( ! (bound( ?b ) ) )
        }!!

Connecting to the server..
Answer from from the server:
->      Bathroom
->      Kitchen

Press any key to formulate a new query.
```

Figure 7.10: An example of a SPARQL query performed by means of our query-client

Now let us assume we want to make this knowledge persistent, perhaps in order to share it among different applications.
Well, by using *Shelve*[1] CAFE provides us the possibility to store the context of a given environment into a binary file.

```
>>> contextspace.saveContext("/home/david/myContext")
Context snapshot taken at the time '2012-05-15 00:58:49.366507'.
>>> contextspace.saveContext("/home/david/myContext")
Context snapshot taken at the time '2012-05-15 01:09:44.128560'.
>>> contextspace.saveContext("/home/david/myContext")
Context snapshot taken at the time '2012-05-15 01:16:57.392196'.
>>>
```

Figure 7.11: Saving a context's snapshot into a binary file

[1] *Shelve* is a powerful Python module for object persistence. [python.about.com]

In practice CAFE takes a snapshot of the context at a given moment. In this way we may access a shelf containing all the history of the context, that allows us to reason on events happened in the past. Figure 7.11 shows how CAFE saves some snapshots of the context into a binary file.

Figure 7.12 shows how to load the history of a context. As is visible, every snapshot constituting the history is identified by a timestamp and is composed by the namespace of a Context-Space and a graph containing all the contextual information about that Context-Space.

```
>>> from cafe.tools.utils import Utils
>>> print Utils.loadContext("/home/david/myContext")
2012-05-15 01:09:44.128560 -> ('http://cafe.ns/home#', <Graph identifier=tmpZzcJL7
(<class 'rdflib.graph.ConjunctiveGraph'>)>)
2012-05-15 01:16:57.392196 -> ('http://cafe.ns/home#', <Graph identifier=tmpZzcJL8
(<class 'rdflib.graph.ConjunctiveGraph'>)>)
2012-05-15 00:58:49.366507 -> ('http://cafe.ns/home#', <Graph identifier=tmpZzcJL9
(<class 'rdflib.graph.ConjunctiveGraph'>)>)
```

Figure 7.12: Loading a context's history from a binary file

In addition, CAFE allows us to serialize the context to an OWL ontology and upload it on a FTP server. In this way we are free to access it from every networked device and reason on it using an external OWL reasoner. Furthermore, we may choose to convert it again into a graph, in order to update its content. Figure 7.13 shows how CAFE serializes a context into an OWL ontology, updates it on a FTP server, and finally gets it from the server converting it again into a graph.

```
>>> ontology = contextspace.serializeContext()
Context serialized in OWL.
>>>
>>> contextspace.writeContext(ontology, True)
OWL ontology uploaded on 'ftp.ilbello.com'!
>>>
>>> Utils.parseOntology("http://dapids.ilbello.com/cafe/context.rdf")
Ontology 'http://dapids.ilbello.com/cafe/context.rdf' parsed into the graph:
        '[a rdfg:Graph;rdflib:storage [a rdflib:Store;rdfs:label 'IOMemory']].'
```

Figure 7.13: Context as OWL ontology

Finally, as already explained in Section 6.1, in CAFE Context-Space are treated as simple graphs, and that allows us to exploit all the properties related to this kind of data structures. Since the graph model is redundancy-free, ambiguous situations due to some repetitions of information are automatically avoided. Therefore, merging two or more graphs it is possible to bind information coming from different Context-Spaces in order to have a wider view of the context.

Figure 7.14 shows how CAFE merges two Context-Spaces representing two different environments. The result is nothing but a third graph containing all the information belonging to the merged graphs.

```
>>> newContext = Utils.mergeContextSpaces([context1, context2])
Resultant graph:
        [a rdfg:Graph;rdflib:storage [a rdflib:Store;rdfs:label 'IOMemory']].
>>> 
```

Figure 7.14: Merging two different Context-Spaces

7.3 Summary

In this chapter we proposed an illustrative scenario in order to show how it is possible to use CAFE to represent and manage the context related to that scenario. We also used this demonstration to underline the results we achieved and point out the limitations of CAFE.

Chapter 8

Conclusions

One of the crucial issues related to context-aware computing is to have a proper and convenient model to represent and manage the context. Existing representation models like ontologies constitute a well researched and mature solution. However, they are not made to represent continuously changing data; moreover, building and maintaining them might be highly error-prone, time-consuming, and non-scalable processes, and they can become tedious tasks if they are done manually.
This thesis work aimed to propose a model that is highly-readable, flexible, scalable, general-purpose, and modular, in order to represent and manage contextual information of different types, generated by a variety of heterogeneous sources and with different levels of granularity. Furthermore, the proposed model allows reasoning, guarantees an easy handling of the atomic data forming the context, and is capable to manage incomplete or not accurate information due to incompleteness of data from sensors.
This model is based on the integration of Semantic Networks with the Object-Oriented model, therefore it exploits all the advantages of the two approaches.

In this essay, first of all we analysed the most popular and important representation models suitable to represent contextual data, pointing out, for each one of them, advantages and disadvantages.
After that, we focused on Semantic Networks, explaining in details their strong and weak spots, and the reasons that led us to choose them.
Then, we presented the approach we followed in order to integrate Semantic Networks and the Object-Oriented model, showing that the use of scripting languages and their properties, such as dynamic typing, meta-programming, and introspection, guarantees a good integration of the two models.
In addition, we showed the implementation of CAFE, a contextual infrastructure written

in Python that exploits the integration of Semantic Networks and the Object-Oriented model.
Finally, we introduced an illustrative scenario in CAFE, showing how the presented model works, and underlining the achieved results and the gaps which can be plugged in the future.

In conclusion, in this section we summarise the results of our work, highlighting the reasons why the integration of Semantic Networks with the Object-Oriented constitutes a model characterized by high readability, flexibility, scalability, general-purpose, and modularity.

- **General-purpose**. The model proposed does not need to work in a specific domain. Indeed, the high expressiveness of Semantic Networks makes the system able to easily change in response to different system requirements and types of data.
- **High readability**. The possibility to have a graphical representation of the context (Semantic Networks are a graphical model) gives to the model proposed in our approach a very high readability. However, it is clear that the readability of this model decreases proportionally with the increasing of the amount of data we deal with.
- **Flexibility**. Thanks to the use of Semantic Networks, the model proposed guarantees an easy handling of the atomic data forming the context, and is capable of managing incomplete or inaccurate information due to incompleteness of data from sensors.
- **Scalability**. The model proposed enjoys a good scalability. For example, we might choose to divide a given context in more sub-contexts and manage them separately, and finally, since Semantic Networks can be considered graphs, we can merge the sub-contexts without any effort and consider them again as a single context during the reasoning phase.
- **Modularity**. According to the Object-Oriented approach, every entity forming the context constitutes a separate entity, whose internal workings are decoupled from other parts of the system. In particular, modularity brings with it the following advantages concerning the system design and development:
  - **Simplicity**. Thanks to the Object-Oriented model, the complexity of the proposed model is reduced and its structure is very clear.
  - **Modifiability**. In the proposed model it is easy to make minor changes in the data representation. In fact, the integration of the Object-Oriented model

guarantees that changes of an entity do not affect any other part of the model, since in the Object-Oriented approach the only public interface that the external world has to a class is through the use of methods.

- **Extensibility**. The model proposed is extensible, since, thanks to the Object-Oriented model, adding new features or responding to changing operating environments can be solved by introducing a few new objects or modifying some existing ones.
- **Maintainability**. The model proposed enjoys a high maintainability, since the Object-Oriented approach allows objects to be maintained separately, making locating and fixing problems easier.
- **Re-usability**. The Object-Oriented model makes the model we proposed reusable. This means that entities characterizing a given context can be reused in different contexts.

8.1 Future works

Following the approach described in this essay and analysing CAFE, the context-aware infrastructure presented in this thesis work, a series of possible future works could be taken up.

First of all, we claim that it would be beneficial to integrate our model with a standard ontology language, such as OWL, in order to overcome the limitations that affect regular Semantic Networks. In that way, in fact, the resulting model would be even more complete and therefore more convenient to use in order to represent and manage context.

Second, although our approach allows us to merge different contexts (since they are treated like graphs), so far CAFE does not provide the possibility to manage the unification of distributed Context-Spaces. In order to handle this issue, we consider interesting the idea to use a tuple space[1]. In fact, by using a tuple space, every Context-Space could post its own context as a set of tuples in the space, and reasoners could retrieve the data which they are interested in directly from that space, with the purpose of reasoning on them.

Furthermore, since CAFE needs data in subject-predicate-object format, it assumes that every sensor is provided with an interface aiming to transform raw data to the above-

[1] A *tuple space* is an implementation of the associative memory paradigm for parallel/distributed computing. [wikipedia.org]

mentioned format. For this reason, we chose to overcome this issue by working with simulated sensors. Thus, a future work would be to make CAFE work with real sensors, by designing and implementing simple interfaces capable to take raw data from sensors as input and give a subject-predicate-object representation of them as output.

Finally, in this essay we claimed that the Object-Oriented approach which CAFE is based on makes it a direct, quick-to-understand, and easy-to-use contextual infrastructure. However, an interesting future work could be to use HCI usability techniques, such as think-aloud protocol[2], heuristic evaluation[3], and cognitive walkthrough[4], in order to perform a qualitative evaluation that effectively demonstrates the claimed ease-of-use and simplicity-of-understanding.

[2] *Think-aloud protocol* (or TAP) is a method used to gather data in usability testing in product design and development. [wikipedia.org]

[3] A *heuristic evaluation* is a discount usability inspection method for computer software that helps to identify usability problems in the user interface (UI) design. [wikipedia.com]

[4] The *cognitive walkthrough* method is a usability inspection method used to identify usability issues in a piece of software or web site, focusing on how easy it is for new users to accomplish tasks with the system. [wikipedia.org]

Bibliography

Bikakis, A., Patkos, T., Antoniou, G., & Plexousakism, D. (2007). A survey of semantics-based approaches for context reasoning in ambient intelligence. In .

Bottazzi, D., Montanari, R., & Toninelli, A. (2007, sep). Context-aware middleware for anytime, anywhere social networks. *IEEE Intelligent Systems*, *22*(5), 23–32.

Brachman, R. (1983, October). What is-a is and isn't: An analysis of taxonomic links in semantic networks. *Computer*, *16*(10), 30-36.

Brown, M. (1996). Supporting user mobility. In *In proceedings of ifip world conference on mobile communications* (pp. 69–77). Chapman and Hall.

Brown, P. J., Bovey, J. D., & Chen, X. (1997, October). Context-aware applications: from the laboratory to the marketplace. *IEEE Personal Communications*, *4*(5), 58-64.

Chen, G., & Kotz, D. (2000). *A survey of context-aware mobile computing research* (Tech. Rep.). Hanover, NH, USA.

Chen, H. (2004). *An intelligent broker architecture for pervasive context-aware systems*. Unpublished doctoral dissertation, University of Maryland, Baltimore County.

Chen, H., Finin, T., & Joshi, A. (2003). An ontology for context-aware pervasive computing environments. *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review*, *18*, 197–207.

Chen, H. L. (2004). *An intelligent broker architecture for pervasive context-aware systems*. University of Maryland, Baltimore County.

Clark, H. H., & Brennan, S. E. (1991). Grounding in communication. perspectives on socially shared cognition. In L. B. Resnick, J. M. Levine, & S. D. Teasley (Eds.), *Perspectives on socially shared cognition* (pp. 127–149). American Psychological Association.

Cooperstock, J. R., Tanikoshi, K., Beirne, G., Narine, T., & Buxton, W. (1995). Evolution of a reactive environment. In I. R. Katz, R. L. Mack, L. Marks, M. B. Rosson, & J. Nielsen (Eds.), *Chi* (p. 170-177). ACM/Addison-Wesley.

Dargie, W. (2009). *Context-aware computing and self-managing systems* (1st ed.). Chapman & Hall/CRC.

Dey, A. K., Abowd, G., Pinkerton, M., & Wood, A. (1998). Cyberdesk: A framework for providing self-integrating context-aware services. In *Knowledge-based systems* (pp. 47–54). ACM Press.

Dey, A. K., & Abowd, G. D. (1999). Towards a better understanding of context and context-awareness. In *In huc '99: Proceedings of the 1st international symposium on handheld and ubiquitous computing* (pp. 304–307). Springer-Verlag.

Dey, A. K., & Abowd, G. D. (2000). Cybreminder: A context-aware system for supporting reminders. In (pp. 172–186). Springer-Verlag.

Elrod, S., Hall, G., Costanza, R., Dixon, M., & Rivieres, J. des. (1993). Responsive office environments. *Commun. ACM*, *36*(7), 84-85.

Fickas, S., Kortuem, G., & Segall, Z. (1997). Software organization for dynamic and adaptable wearable systems. In *In proceedings first international symposium on wearable computers (iswc'97)* (pp. 13–14).

Fowler, M. (2002). *Patterns of enterprise application architecture*. Addison-Wesley Longman, Amsterdam.

Henricksen, K., & Indulska, J. (2004). Modelling and using imperfect context information. In *Proceedings of the second ieee annual conference on pervasive computing and communications workshops* (pp. 33–). Washington, DC, USA: IEEE Computer Society.

Hull, R., Neaves, P., & Bedford-Roberts, J. (1997). Towards situated computing. In *Proceedings of the 1st ieee international symposium on wearable computers* (pp. 146–). Washington, DC, USA: IEEE Computer Society.

Kalyanpur, A., Pastor, D. J., Battle, S., & Padget, J. A. (2004, June). Automatic mapping of owl ontologies into java. In F. Maurer & G. Ruhe (Eds.), *Proceedings of the 16th int'l conference on software engineering & knowledge engineering (seke'2004)* (pp. 98–103). Banff, Alberta, Canada.

Kim, H., Cho, Y.-J., & Oh, S.-R. (2005). *Camus: a middleware supporting context-aware services for network-based robots.*

Krumm, J. (2009). *Ubiquitous computing fundamentals* (1st ed.). Chapman & Hall/CRC.

Ludford, P. J., Frankowski, D., Reily, K., Wilms, K., & Terveen, L. (2006). Because i carry my cell phone anyway: functional location-based reminder applications. In *Chi 06 proceedings of the sigchi conference on human factors in computing systems* (Vol. 8, pp. 889–898). ACM Press.

Marra, R. M., & Jonassen, D. H. (1996). Transfer effects of semantic networks on expert systems: mindtools at work. In *Proceedings of the 1996 international conference on learning sciences* (pp. 219–226). International Society of the Learning Sciences.

Morse, D. R., Armstrong, S., & Dey, A. K. (2000). *The what, who, where, when, and how of context awareness.*

Mozer, M. C. (1998). The neural network house: An environment hat adapts to its inhabitants. *Proc AAAI Spring Symp Intelligent Environments*, 110–114.

Mrohs, B., Luther, M., Vaidya, R., Wagner, M., Steglich, S., Kellerer, W., et al. (2005). Owl-sf: A distributed semantic service framework. *Proc. of the Workshop on Context Awareness for Proactive Systems (CAPS 05), Helsinki*.

Nierstrasz, O. (1989). A survey of object-oriented concepts. In *Object-oriented concepts, databases and applications* (pp. 3–21). ACM Press and Addison Wesley.

Oren, E., Delbru, R., Gerke, S., Haller, A., & Decker, S. (2007). Activerdf: object-oriented semantic web programming. In C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, & P. J. Shenoy (Eds.), (p. 817-824). ACM.

Ousterhout, J. K. (1998, March). Scripting: Higher-level programming for the 21st century. *Computer*, *31*(3), 23–30.

Pascoe, M. J. (1998). Adding generic contextual capabilities to wearable computers. In *Proceedings of the 2nd ieee international symposium on wearable computers* (pp. 92–). Washington, DC, USA: IEEE Computer Society.

Poslad, S. (2009). *Ubiquitous computing: Smart devices, environments and interactions* (1st ed.). Wiley Publishing.

Ranganathan, A., & Campbell, R. H. (2003). A middleware for context-aware agents in ubiquitous computing environments. In *Proceedings of the acm/ifip/usenix 2003 international conference on middleware* (pp. 143–161). New York, NY, USA: Springer-Verlag New York, Inc.

Rekimoto, J., Ayatsuka, Y., & Hayashi, K. (1998). Augment-able reality: Situated communication through physical and digital spaces. In *In proceedings of the 2 nd international symposium on wearable computers* (pp. 68–75).

Rodden, T., Chervest, K., Davies, N., & Dix, A. (1998). Exploiting context in hci design for mobile systems. In *in workshop on human computer interaction with mobile devices.*

Román, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R. H., & Nahrstedt, K. (2002, October). A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, *1*(4), 74–83.

Ryan, N. S., Pascoe, J., & Morse, D. R. (1998, October). Enhanced reality fieldwork: the context-aware archaeological assistant. In V. Gaffney, M. van Leusen, & S. Exxon (Eds.), *Computer applications in archaeology 1997.* Oxford: Tempus Reparatum.

Schilit, B., & Theimer, M. (1994). Disseminating active map information to mobile hosts. *IEEE Network*, *8*(5), 22–32.

Schilit, W. N. (1995). *A system architecture for context-aware mobile computing.*

Segaran, T., Taylor, J., & Evans, C. (2009). *Programming the semantic web*. Cambridge, MA: O'Reilly.

Serafini, L., & Tamilin, A. (2005). Drago: Distributed reasoning architecture for the semantic web. In *In eswc* (pp. 361–376). Springer.

Soldatos, J., Dimakis, N., Stamatis, K., & Polymenakos, L. (2007, February). A breadboard architecture for pervasive context-aware services in smart spaces: middleware components and prototype applications. *Personal Ubiquitous Comput.*, *11*(3), 193–212.

Sowa, J. (1984). *Conceptual structures: Information processing in mind and machine*. Addison-Wesley.

Sowa, J. F. (1991). Principles of semantic networks. In . Morgan Kaufmann.

Wang, X., Dong, J. S., Chin, C.-Y., Hettiarachchi, S., & Zhang, D. (2004). Semantic space: An infrastructure for smart spaces. *IEEE Pervasive Computing*, *3*(3), 32-39.

Wang, X. H., Zhang, D. Q., Gu, T., & Pung, H. K. (2004). Ontology based context modeling and reasoning using owl. *Pervasive Computing and Communications Workshops, IEEE International Conference on*, *0*, 18.

Want, R., Falcao, V., & Gibbons, J. (1992). The active badge location system. *ACM Transactions on Information Systems*, *10*, 91–102.

Ward, A., & Jones, A. (1997). *A new location technique for the active office.*

Zouaq, A., Gasevic, D., & Hatala, M. (2011). Unresolved issues in ontology learning - position paper. In . Vancouver, BC, Canada.

School of Computer Science
Reykjavík University
Menntavegi 1
101 Reykjavík, Iceland
Tel. +354 599 6200
Fax +354 599 6201
www.reykjavikuniversity.is
ISSN 1670-8539