MULTIPOS	

MULTIPOS

D3.1. Semantic positioning framework 1.0: State-of-the-art report on positioning technology and the related semantic technology ecosystem. Version 1.0

Technical Report

Contractual Date of Delivery:	$T_0 + 7$
Actual Date of Delivery:	$T_1 + 7$
Editor:	Alejandro Rivero Rodriguez
Author(s):	Ossi Nykänen, Alejandro Rivero Rodriguez
Participant(s):	*TUT*
Work package:	*WP3 – Technical Report*
Version:	1.0
Total number of pages:	19

Abstract:
Acknowledging the user context, e.g., position, provides a natural way to focus applications. How to actually model and exploit context in applications, however, is not self-evident, and assigning the related responsibilities to individual applications is tempting. We believe this confuses users and limits computing with user context. In this work, we discuss context modeling and identify the minimal responsibilities of the context engine, a novel software component able to compute and reason with the context for other applications. We suggest that the natural responsibilities of the context engine include reasoning with the sensor context and managing user preferences, thus clarifying the role of context-aware computing in applications. To achieve this, we review the current sensor frameworks, propose adopting a context ontology and point out a logic programming based approach for extending and reasoning with context. This allows applications to exploit the context without having to fully model/understand it.

Disclaimer:

T0: ESR Expected Start Date

T1: ESR Real Start Date

Document Control

Version	Details of Change	Review Owner	Date
0.8	Most content included	ON,AR	13.1.2014
0.9	First del. Draft, Sem Web section added	AR,ON	14.2.2014
1.0	Motivation chapter details	AR	25.2.2014

Executive Summary

This document describes the work of the MULTI-POS fellow for the deliverable D3.1, named: *State-of-the-art report on positioning technology and the related semantic technology ecosystem.* Please note that Submission of this deliverable was delayed due to the consortium-wide delay in the MULTI-POS ESR recruitment.

In this report we describe the status of the existing technologies for developing context-aware services, review most relevant research work and briefly propose some ideas for future research.

In mobile applications, positioning, time, calendar and other information- contextual information- establishes a convenient starting point for filtering, organizing, and providing access to information relevant to the end user, which can be extended with other kinds of information. However, we face two main issues:

1- Context-Aware Services use mainly position and time. However, much more device information is available via API (see table 3-1).

2- Context Information, especially from external servers, is not always available and its usage is not trivial for developers. That seriously hinders the development of context-aware computing.

In this report we describe a computational component, the so-called context engine, that manages contextual information. The context engine has been described by other authors, and we focused our effort towards outlining its main responsibilities: Answering questions about the current context, extending the currently known context and managing user preferences. Similar work has been proposed in the literature, and it seems that main issues are battery consumption and information privacy.

Different technologies can be used for modelling this contextual information. We have concluded that semantic technologies provide many advantages (see 3-3 Modelling Context) and therefore semantic modelling is our approach for future research.
Therefore we review the semantic technology ecosystem (see chapter 4). These technologies are the base for our future development. It is also relevant the existence of Semantic Sensor Network ontology, created by W3C, which describes sensors and sensor networks for its use in sensor web applications. A common and accepted SSN would semantically enable applications to use different sensorial data from different sources.

Authors

Partner	Name	Phone / Fax / e-mail
TUT	Ossi Nykänen	Phone: +358 40 849 0730 e-mail: ossi.nykanen@tut.fi
TUT	Alejandro Rivero Rodriguez	e-mail: alejandro.rivero@tut.fi

Table of Contents

List of Acronyms and Abbreviations

Term	Description
SSN	Semantic Sensor Network
W3C	World Wide Web Consortium
API	Application Programming Interface
SOCAM	Service-Oriented Context-Aware Middleware

1. Vision

Before talking about complex ideas and technologies, we would like to present a real case scenario to provide readers with an intuitive and practical idea of what can be achieved in this area of research. Please take a look at the picture below: on the left side (pink box) we have an example of how mobile devices work nowadays. On the right side (grey box), we have a scenario assuming that mobile devices are able to adapt to user's behaviour, context and preferences.

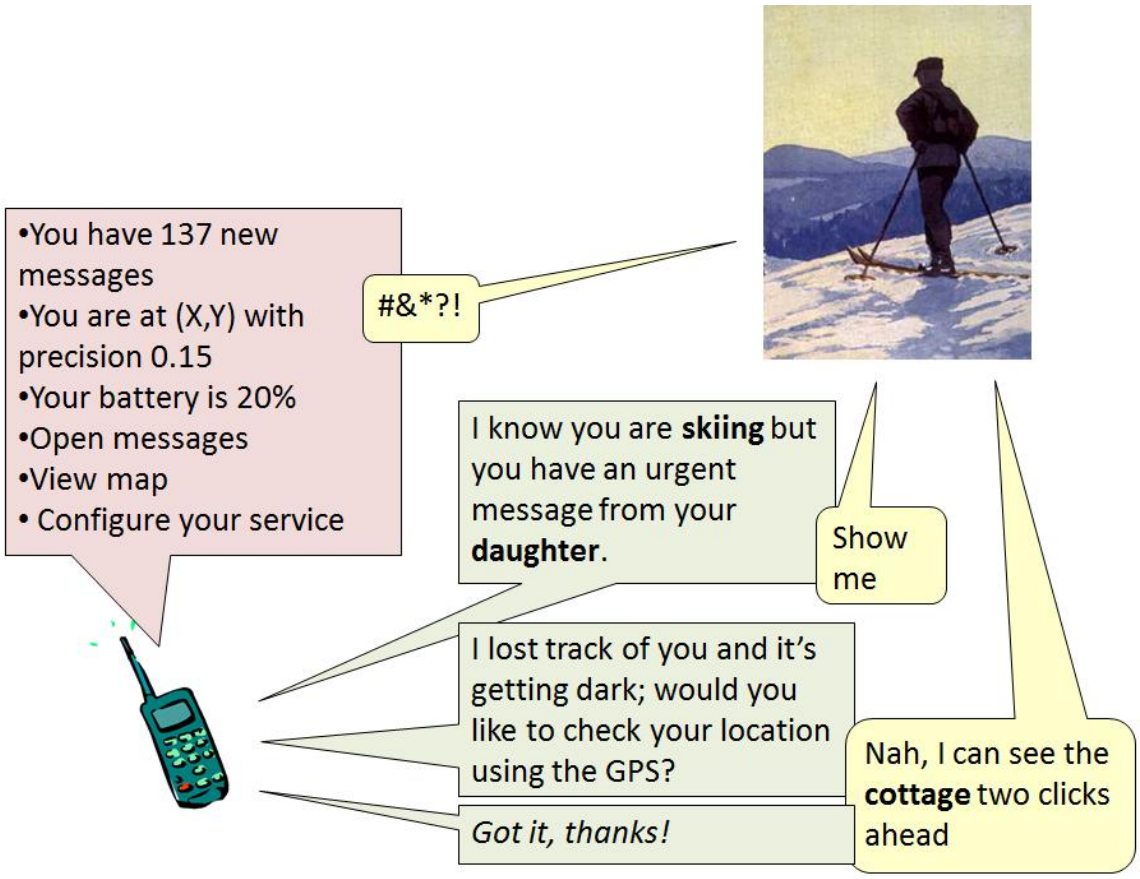

Figure 1-1 Mobile scenario using contextual information

This is a textbook example, but how can we achieve this in practice?
The key idea to solve this problem is an abstract entity called context. It can refer to any information that can be used to characterize the situation of an entity, where an entity can be a person, place, or physical or computational object (Anind, 1944). The systems that use contextual information to provider tailored information to users are called context-aware system. Currently, most common context-aware systems take into account just a few context types such as position or time.
However, context is something much wider. According to (Mehra, 2012), time and location are just the most intuitive forms of context. However, we should go beyond that to find more relevant information about the users. This context can be small context, such as what concrete task user is doing, what kind of trip the user is doing or in what project the user is working at the moment. Other higher-level ways of context include Web of people and knowledge, including user's interest, social networks, etc. The picture below shows different forms of context.

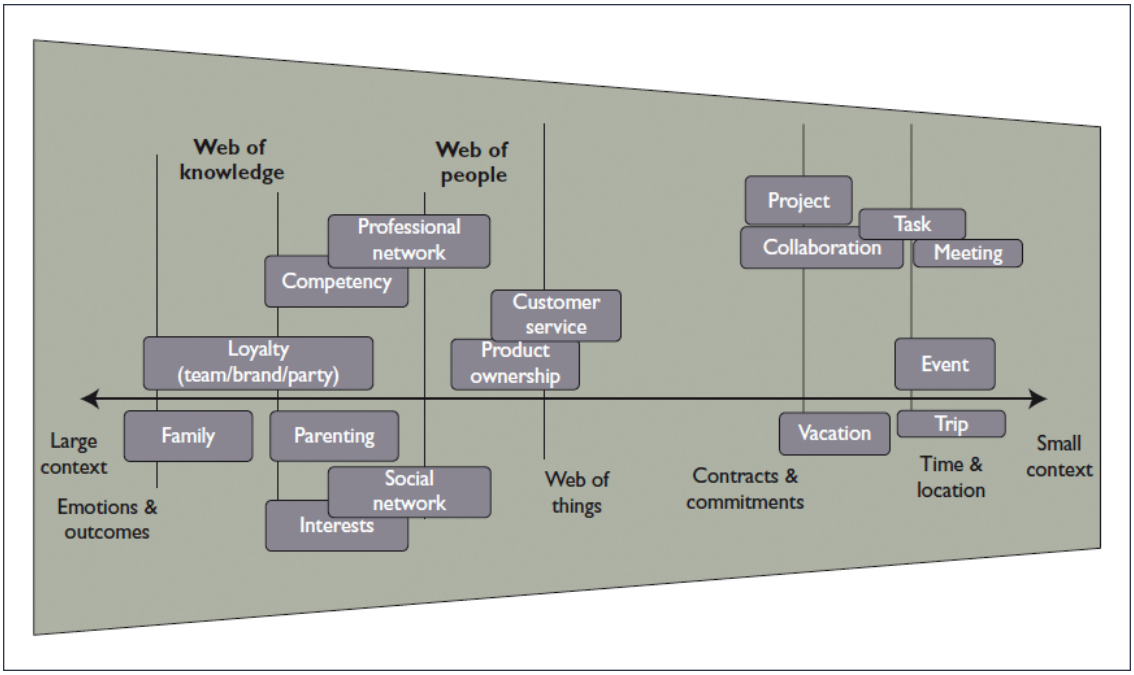

Figure 1-2 Types and levels of context

Please note that contextual information does not refer exclusively at information collected by the mobile device. For instance, sensors that measure the temperature of the living room or traffic congestion are also absolutely valid forms of contextual information.

2. Introduction

Acknowledging the user context provides a natural way to focus user attention and the use of resources in applications. In mobile applications, e.g. positioning, time, calendar and other information establishes a convenient starting point for filtering, organizing, and providing access to information relevant to the end user, which can be extended with other kinds of information.

Pioneering research in context-aware computing research dates back to early 1990s (Chen,2000). Since then, studying and building context-awareness have been first tackled in application-specific manner, then in terms of reusable toolkits, and finally, on infrastructure level (Gu, 2005). Nevertheless, most applications still address generic context-aware computing tasks in application specific manner. Considering toolkits and infrastructure-level support, however, significant development has been made in mobile applications (Palmieri, 2012), and it is fair to say that contemporary mobile sensor frameworks establish the de facto technology driver for context-aware computing.

Despite the two fronts of research, i.e. generic context-aware and mobile sensor frameworks, it is still not self-evident, however, how context-aware processing should be realized and what is the role of context in the application and service ecosystem. In particular, it is tempting to assign the responsibilities of managing and reasoning context to the end user applications, which in practice suggests that each application deals with the context as it sees fit. We believe that this both confuses users and seriously hinders the development of context-aware computing.

In this report, we outline the context-aware processing research and the current mobile frameworks, acknowledging a best practice for modeling context. We then propose the minimal responsibilities of the so-called context engine, a software component that is able to compute and reason with the context on the behalf of the end user applications (or services). We suggest that the natural responsibilities of the context engine include extending and reasoning with the context and managing user preferences, thus clarifying the role of context-aware computing in applications. These baseline responsibilities can be extended by adding functionality for accessing context provider etc. services, and implementing interfaces for application-specific activities.

To achieve this, we review the current sensor frameworks, propose adopting a context ontology and point out a logic programming based approach for extending and reasoning with context. These establish the

basic tools for modeling, understanding, and computing with the user context, so that each application or service module does not have to fully understand and implement each aspect of the user context and tasks.
The main contribution of this work lies in outlining the path for implementing context-aware computing based on existing sensor frameworks, in clarifying context-aware computing tasks and responsibilities with respect to the novel context engine architecture, and in suggesting a resolution-based mechanism for computing and reasoning with context.
We believe that this line of research is very important because understanding user context will have an increasingly significant impact in application and service development. This seems obvious when acknowledging that mobile devices dominate consumer markets, the Internet/Web of things is underway, and the next industrial revolution is likely to adopt the so-called Industrie 4.0 paradigm.

3. Background

In brief, context can refer to any information that can be used to characterize the situation of an entity, where an entity can be a person, place, or physical or computational object (Anind, 1999). Contextual information may include physical information such as accelerometer data, virtual information such as calendar events, recognized patterns such as observed user activities, and predictions such as weather forecasts.
Acknowledging the key characteristic of context as a relevant data source, the notion of a "sensor" is typically generalized. We may acknowledge three different types of sensors providing contextual data (Baldauf, 2007):
- Physical sensors. These are the most frequently used sensors, capable of capturing physical data (e.g. position, orientation, and acceleration).
- Virtual sensors. These provide contextual information from applications and services. Virtual sensors may further be based on local or external data sources (e.g. user calendar vs. weather service).
- Logical sensors: These provide new contextual information by combining and computing information from physical and virtual sensors.

Today, for practical reasons, physical sensors (or physical sensor middleware frameworks) establish the major technology driver in context-aware computing. This is largely due to the fact that developers and application vendors are still in the process of learning how to model and compute with context.
Let us next briefly outline the related research, take a closer look at the current technology driver of mobile sensor context, and identify a best practice for modeling context. This discussion will then be used as a basement for identifying the pivotal context-aware computing architecture and the core responsibilities of context engines.

3.1 Current Technology Driver: Physical Sensor Context

To get hands-on insight in the current state-of-the-art of sensors in context-aware computing, let us briefly review what types of contextual information are actually accessible in the widespread mobile platforms, Android and iOS.
Android developers can make usage of contextual information in several ways (Android Developers). The first approach is using the Android sensor Framework, which includes the motion sensors (e.g. accelerometers), environmental sensors (e.g. temperature) and position sensors (e.g. orientation sensor). It is also possible to access location information with Location API and other additional location services, such as Geofence API to alert user or applications when the user is entering a certain region. iOS developers can mostly access the same type of sensor information, with some exceptions, such that it is not appropriate to access WiFi access point information directly (Fankhauser, 2012).
In addition to device-specific interfaces, various browser APIs are being developed. Indeed, accepting the obvious challenges in generalizing the sensor context of different operating systems, an interesting research perspective on context providers is established by cross-platform tools. These abstract the details of the various platforms, aiming to allow implementation of an application and its user interface for several mobile platforms more efficiently (Palmieri 2012). Table 1 below lists the most popular cross-platform development tools and information categories, effectively pointing out what sensor information is currently available (access languages in parentheses).

Table 3-1 APIs supported by main cross-platform development tools (adapted from (Palmieri,2012))

API \ Tools	Rhodes (JS)	PhoneGap (JS)	MoSync(JS)	MoSync(C,C++)	DragonRad
Accelerometer		X	X		
Barcode	X	X			X
Bluetooth	X	X		X	
Calendar	X	X	X	X	X
Camera	X	X		X	
Capture		X	X	X	X
Compass		X	X		
Connection		X	X	X	
Contacts	X	X			X
Device	X	X	X	X	X
File	X	X	X	X	
Geolocation	X	X	X	X	X
Menu	X				X
NFC	X	X	X	X	X
Notification	X	X	X	X	
Screen Rot	X	X	X	X	
Storage	X	X	X	X	X

This review is important since it has a major impact both in application development and in the current strategies of modelling context. We have seen that, in practice, developers have (easy) access to physical sensorial information.

3.2 Research and Basic Concepts.

The term context-aware (computing) appeared first time in early 1990s, with the beginning of context-aware system research (Chen, 2000). In addition to solely computing with respect to time and place, context-aware systems should capture many other things as well, such as places, things, commitments, and user knowledge and preferences (Mehra, 2012). A typical application area is context-aware search, which includes the phases of data acquisition, context reasoning and state updates, and contextualized output (Yndurain, 2012).

The main components of a context-aware system include context providers and context-aware services, perhaps associated with service locating services (or brokers) (Gu, 2005). When defining context, computing context, user context, and physical context are often differentiated (Chen, 2000), as already described at the beginning of this chapter. Processing contextual information is carried out by a component called context interpreter, and the relevant data is stored in a context database. The basic activities include context assertion, i.e. making contextual information available, and context retrieval, i.e. exploiting the context in an application (Mehra, 2012).

In brief, we may identify three complementary approaches on how the context providers acquire contextual information (Chen, 2000):

– Direct sensor access. In this approach, the client gathers the needed information directly from the sensors.

– Middleware infrastructure. This approach introduces a layered architecture that enhances reusability and concurrent sensor access. Therefore, instead of accessing directly the raw data from sensors, the client gathers information from this intermediate layer that manages sensorial data.

– Context Server. In addition to the above, this approach permits us to gather information from remote data sources, which allows reusing sensorial information, accessing a potentially much richer set of contextual information sources, and distributing the costs of measurements and computations.
Strictly speaking, direct sensor access is not usually feasible since sensor access needs to be encapsulated for multi-tasking, concurrency etc. The middleware approach is thus favoured, but in mobile applications the power consumption and computing costs are typically still a major concern. Thus, in addition to simply requesting additional or more high-level context information from context servers such as weather forecasts and traffic congestion, applications might also seek means to exploit servers in delegating and perhaps orchestrating their local tasks.

Even if the mobile development frameworks do not yet provide abstract means for context-aware computing, various theoretical modelling approaches exist. We may identify at least six major strategies for context modelling (Baldauf, 2007, Strang 2004):
1. Key-Value Models. Simple key-value pairs are used to model context.
2. Markup Scheme Models. Context is modelled as a hierarchical structure (cf. XML).
3. Graphical Models. Using modelling techniques such UML to model context, which provides a generic structure, ER-formalism and an intuitive graphical presentation.
4. Object Oriented Models. Using object-oriented technologies, enabling inheritance, encapsulation with interfaces, reusability, etc.
5. Logic-Based Models. In these approaches, context is defined as facts, expressions and rules, defining the conditions on which a context can be inferred. Its main strength is context reasoning.
6. Ontology Based Models. Combining several of the above approaches, based on formally specifying the basic concepts and interrelations related to context(s).
Currently, there are no commonly agreed standard models or systems for sensing contextual information from various sources to enable reuse across various middleware systems and frameworks (Baldauf, 2007). Ontology based models, however, seem to offer many desirable properties such as information alignment, dealing with incomplete or partially understood information, and formally working with context model of varying level of detail (Strang, 2004).

Let us review the proposed frameworks and architectures.

3.3 Proposed Frameworks

There have been many attempts for designing context-aware frameworks since late 90s. We take a look at some of the most relevant proposed frameworks.
Context Broker Architecture (CoBrA) is an agent-based architecture for context-aware computing in intelligent spaces. It uses ontologies for context processing, and the architecture is compound by the following components: Context Knowledge Base, Context Inference Engine, the Context Acquisition Module and Privacy Management Module.
Hydrogen project is specialised for mobile devices. Its main strength is to avoid the dependency to a centralised component in the majority of context-aware systems. Therefore they describe local context, information the device knows about context, and remote context, the information another device knows about context. The idea is that devices in proximity are able to share the contextual information in peer-to-peer manner. It uses three-layered architecture so-called Application, Management and Adaptor later, and the modelling approach is object-oriented.
Context-Awareness Sub-Structure (CASS) presents a middleware architecture composed by Interpreter, Context Retriever, Rule Engine and a Sensor Listener. The high-level context inference is based on a knowledge base and an inference (rule) engine. Furthermore, there are many other modelling frameworks such as CORTEX, SOCAM, Gaia, Context Management Framework or Context Toolkit. More detailed information about these systems can be found in other articles (Baldauf, 2007).

However, for reasons mentioned in last section, we think the most appropriated approach for context modelling is ontology-based models. We identify a highly promising ontology based modelling approach, the Service-Oriented Context-Aware Middleware (SOCAM) architecture (Gu, 2005). In brief, SOCAM aims providing efficient infrastructure support for building context-aware services in pervasive computing environments. Context modelling is carried out in (OWL) ontologies, based on two-level information architecture: the general context concepts are captured in the common upper ontology and application-specific concepts in domain ontologies (see Figure 1).

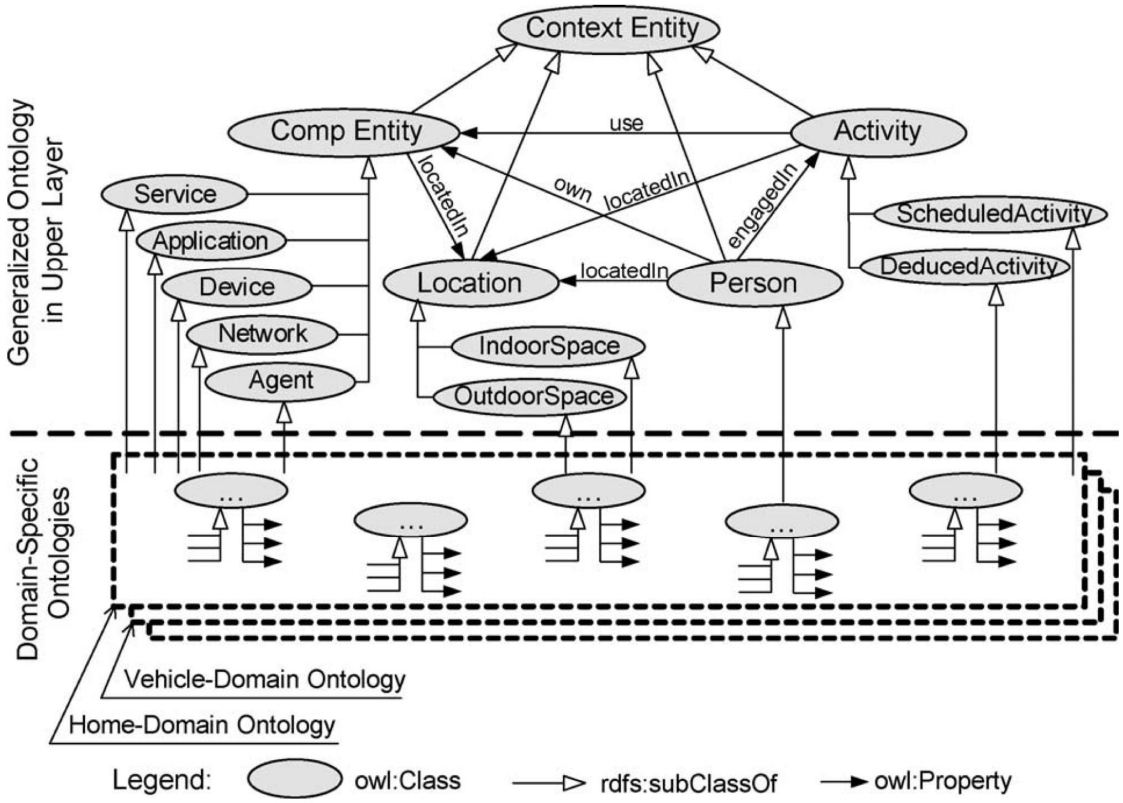

Figure 3-1 Class hierarchy of the upper (SOCAM) ontology (Gu, 2005)

As a best practice, let us next consider the context technology driver discussion in the previous subsection. It seems evident that a revised version of a context ontology might include parts of the readily available state-of-the art sensor context APIs, as built-in primitives already at the upper ontology layer. Further, specific context information might not be always available, for instance due energy saving considerations (e.g. GPS being switched off). This means that contextual information needs to be archived. Also, practical sensor information can be managed only approximately, suggesting that some build-in modelling mechanism for vagueness is needed.
From the perspective of our work, however, the SOCAM approach has several significant merits, two of which worth explicitly mentioning here. First, it demonstrates viable candidate of a standard approach for context modelling, introducing the crucial distinction between upper-level and domain-specific model concepts, with concretely appealing design. Second, it provides a potentially rich system of context services and providers, which raises the question of identifying the characteristic properties of context-aware computing, i.e. keeping the core responsibilities of applications, context interpreters, and context providers at minimum.

Some of the most relevant issues to solve are contextual reasoning with missing or imprecise contextual data, resource consumption and privacy issues. Some examples include (Suman, 2012), which exploits the relationship between context attributes in order to achieve better energy-efficiency, and (Mileo, 2012), (Augusto, 2008) using non-classical models to reason with imprecise or missing data.

3.4 Modelling context with Semantic Sensor Network ontology

W3C Incubator Group worked on the idea of Semantic Sensor Network (SSN) ontology, which describes sensors and sensor networks for its use in sensor web applications (W3C SSN, 2011). A common and accepted SSN would semantically enable applications to use different sensorial data from different sources. Furthermore this standardization is very important to bridge the Internet of Things and the Internet of Services. The picture below shows the SSN ontology structure, to intuitively see how it works. It is fair to point out that this ontology has not been built from scratch. The process started by reviewing

17 ontologies to describe sensors with their capabilities and observations, in order to understand similarities and differences between them.

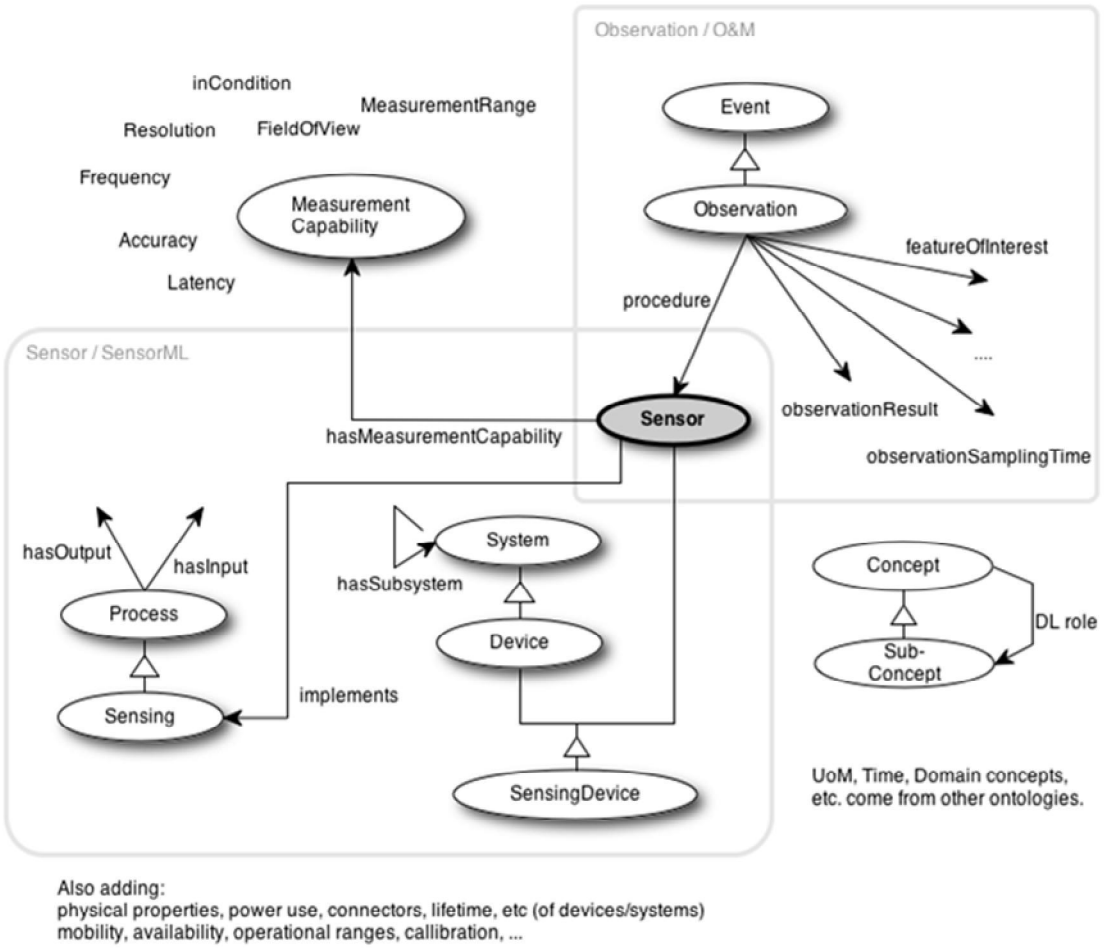

Figure 3-2 Overview of the ontology structure

Please note that, if we look at of use cases, SSN merely helps organizing contextual information. In order to successfully share and link the data, data provider should provide access to their data, and perhaps use adapters for some sensors devices and publish it as sensorial information. Context consumers should create application using SSN ontology for some purpose, such as for modelling context.

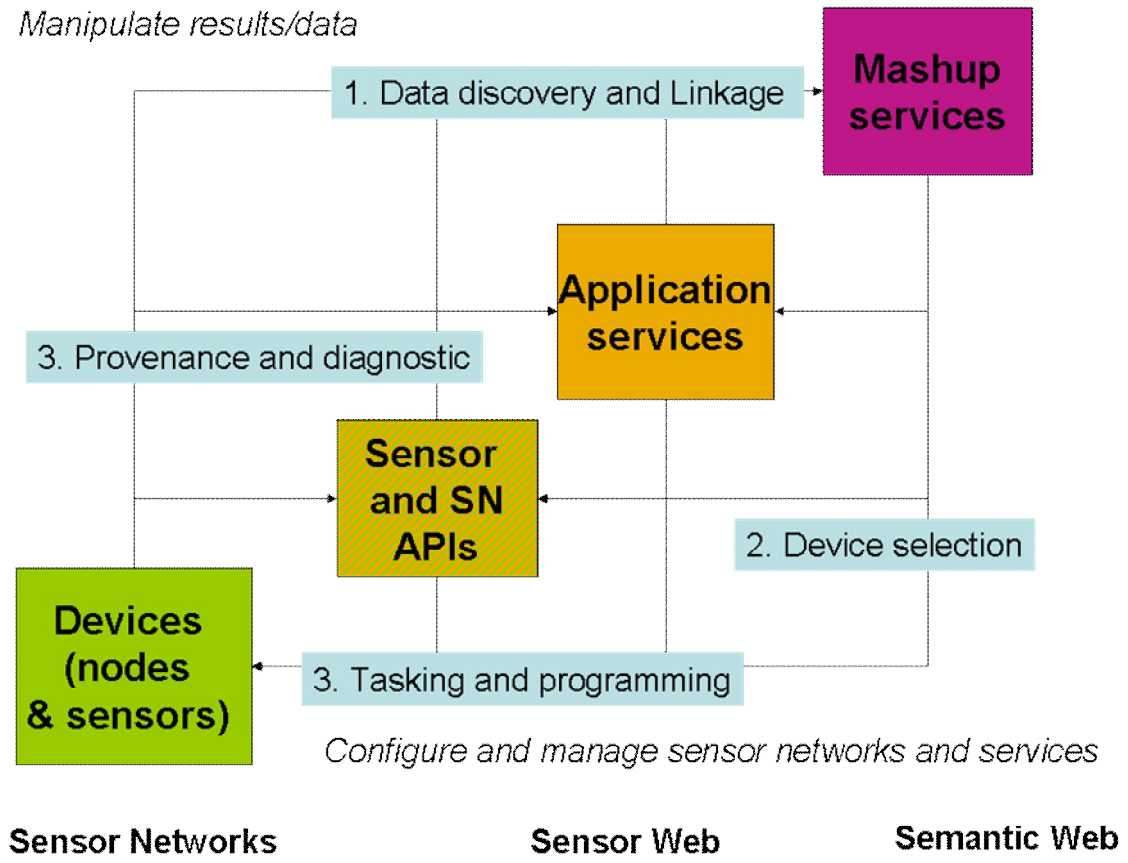

Figure 3-2 Technologies and Use Cases

4. Semantic Technology Ecosystem

Let us then briefly outline the underlying semantic (web) technology ecosystem.

The de facto semantic technology is based on the World Wide Web Consortium (W3C) Semantic Web specifications (W3C Semantic Web, 2014). Semantic Web provides the fundamental enabling technology standards, common data models, and best practices for realizing applications exploiting data with machine-readable semantics.

Description of the W3C Semantic Web Activity, alongside with the technical specifications and other documentation, are freely available at W3C website (W3C Semantic Web, 2014). In December 2013, W3C merged Semantic Web groups with the new W3C Data Activity, emphasizing the vertical application opportunities in development (W3C Data, 2014).

The foundation of the technology is established with the Resource Description Framework (RDF) and the related specifications, originally published in 1999 and slightly refactored and revised in 2001-2003. It is expected that the version 1.1 of RDF will be finalized in 2014, adding some minor extensions to the original model and serialization syntax.

Though its most important serialization syntax, RDF/XML, RDF can be founded on the XML family of specifications. In addition to the low-level structural level access, the related XML standards provide mechanisms for, e.g., signing and encrypting semantic information, and allow integration of the semantic technologies with service-oriented systems. While most applications follow the client-server model of the mainstream web architecture with dynamic client and server roles, it is also possible to use semantic technologies in the context of ad hoc and peer-to-peer networks (Nykänen, 2009).

The most recent RDF development includes development of JSON and CSV serialization syntaxes. In brief, together with the URI (IRI) and XML integration, these seamlessly bridges Semantic Web technologies with the extensive set of Web technologies in general, conceptually coined as the Open Web Platform (W3C Standards, 2014).

From the perspective of formal semantic modelling, a significant family of standards was introduced in 2004, in terms of the Web Ontology Language (OWL). Version 2 of the OWL standard was published in 2009 and slightly revised in the second edition in 2012.

The OWL 2 Web Ontology Language, informally OWL 2, is an ontology language for the Semantic Web with formally defined meaning. OWL 2 ontologies provide classes, properties, individuals, and data values and are stored as Semantic Web documents. OWL 2 ontologies can be used along with information written in RDF, and OWL 2 ontologies themselves are primarily exchanged as RDF documents. (W3C OWL 2 Overview, 2012)

The OWL (DL) language provides the technological basis for perhaps the most promising semantic applications, enabling the standardization of terminology in numerous application areas, including the Semantic Sensor Network (SSN) ontology from the W3C (W3C SSN, 2011). Commonly known ontology repositories include the DAML Ontology Library (DAML, 2014) and the Protégé Ontology Library (Protégé, 2014).

Not all semantic applications, however, require strict, machine-understandable semantics, i.e. application of which that can be reduced to search due resolution, tableau, etc. algorithms. In some cases, it is sufficient to document application information for purposes of human designer consumption and associative linking, i.e. where semantics can be expressed informally, essentially in the choice of common names with well-established narrative documentation.

For this purpose, two overlapping lines of semantic technology development are thus worth emphasizing. The first is (broader) "Linked Data" and the second (narrower) "Semantic Web". Roughly speaking, the distinction lies in level of machine-understandability of the related application data. Linked data provides best practices and enabling technology for a broad class of applications for publishing and linking (extralogical) data in various formats, in order to provide machine and human processable associations between data. Semantic Web, on the other hand, establishes a subset of Linked Data applications. It in addition emphasizes that the published information is modelled according to the RDF or OWL DL models, and yields logical consequences (entailments) according to the formal interpretation of the model(s).

In practice, the above distinction is not crisp, and there exists a certain "continuum" between the two classes of semantic applications. Either way, when conforming to the underlying RDF data model, both Linked Data and Semantic Web data may be queried and processed using the standard technologies, including the SPARQL query language.

Finally, standards and information alone are obviously not sufficient since tools are required. According the common conceptualization (Hebeler et al., 2009), the Semantic Web Software Development Environment includes the components of ontology editor (e.g. Protégé), integrated (software) development environment (e.g. Eclipse), Semantic Web (development) framework (e.g. Jena), and reasoner (e.g. Pellet). While certain tools remain experimental, the baseline technology is readily applicable. At the time of writing, however, application stability and response-time requirements still pose non-trivial challenges for developers.

5. Context Engine

When designing an ecosystem for context-aware systems, the first step is to provide a rich ecosystem of toolkits and services, which are then applied to design ad hoc applications. While middleware infrastructure solutions provide general-purpose frameworks for developers, they do not as such elaborate what it actually means to compute with contextual information.

As a research and design rationale, we believe that three things need to be elaborated in context-aware processing:

- A software component called context engine (cf. Mehra, 2012) should be explicitly introduced, to accept the responsibilities of computing with context on the behalf of applications. This includes the basic tasks of the local context provider, logical context interpretation, and user preferences management (which typically exceed the boundaries of individual applications).
- The core responsibilities of the context engine should be kept at minimum. The essential task is providing context information to the applications via various logical queries in terms of a standard communication interface, and managing user preferences. More complex content engines can be derived via extensions, by extending the related knowledge bases and adding new methods to the related API.
- Domain-specific knowledge about how to actually utilize context in applications is not a responsibility of the context engine. In particular, domain-specific knowledge is communicated in terms of references to domain-specific ontology modules and user preferences. When needed, any extralogical computation (including heuristics, predictions, etc.) can be delegated to internal or external services. This also means that an individual application does not have to fully understand the knowledge base of the context engine for making simple queries or asking questions about the current context, and vice versa.

We anticipate that eventually, a context engine is a service provided by a sensor framework, including an operating system level utility similar to personal details or privacy settings. For end users, the context engine thus provides the main access point to context-aware settings and user preferences.
In the scope of this work, however, our main interest lies in analysing the behaviour and the minimal responsibilities of the context engine. To elaborate this, let us next define the basic context engine architecture, and specify the basic forms reasoning with context.

5.1 Architecture

The context engine architecture includes four main components, shown in the Figure 2. In brief, the end user interacts with an application, which executes user activities and access contextual information through the context engine. Typical end user applications include information management and communication applications, such as calendar, messaging and telephony applications, and novel software agents.

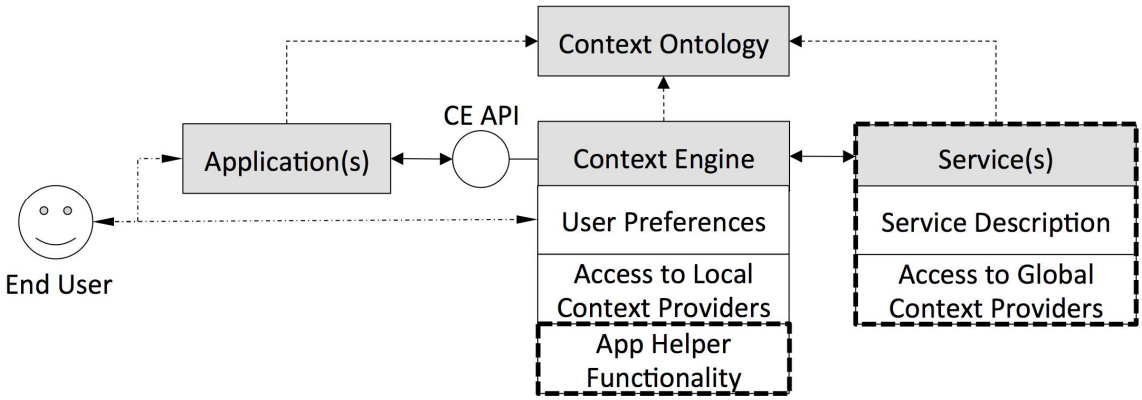

Figure 5-1 Context Engine Architecture (optional extensions marked with dashed rectangles)

When context-aware semantic processing is needed, the user application utilizes the context interpreter and methods provided by the context engine. For this purpose, the context engine has access to local context providers and possibly to external services. For instance, application might simply ask the current context such as user activity, or, when provided a context (location, time), the context engine may infer additional information about the context, e.g. (location, time, current calendar activity) or (location, time, weather), exploiting physical, virtual, and logical sensors.
From the perspective of the end user, the context engine also manages user preferences that are taken into account in context-aware computing. For instance, the user might prefer not accepting certain kinds of phone calls outside the office hours.
The terminology of the context is specified in the context ontology, which includes generalized and domain-specific modules (see also Figure 1). In addition, the context engine manages (rule-based) knowledge about reasoning and answering questions about the context that is needed for knowledge alignment etc. We will later see that in practice, the context ontology might need to be extended with concepts related to user preferences and application functionality.
Implementation of a context engine can be conveniently founded on some existing sensor framework. In addition, the context engine architecture depicts optional extensions. These include external services that encapsulate useful context providing and other services, and context engine helper functionality for applications, including additional, perhaps domain-specific methods and context listener services.
From the software engineering point of view, the context engine implements the context engine Application Programing Interface (API), which provides the main interface to the context-aware information processing system. A typical user application needs to understand only certain aspects of the context ontology. For instance, a simple telephony application might only need to know whether it currently is office hours and if the user is in a meeting or not.
This simple architecture helps discussing and analysing the role of context-aware semantic processing in applications. In particular, acknowledging the categories of context-aware applications proximate selection, automatic contextual reconfiguration, contextual information and commands, and context-triggered actions (Chen, 2000), we may acknowledge that interactions between application and the context engine may be triggered either by the application (e.g. context retrieval query) or by the context engine (e.g. message triggered by a specific contextual event listener). According to our context engine

architecture, however, the latter case is not part of the core context engine functionality, but is rather considered as an extension.
Further, we may conclude that e.g. in proximate selection, i.e. emphasizing currently "nearby" objects or information, it is the task of the context engine to provide the necessary information about the context, but the actual prioritization of the information presented to the user is the task of the application. It is worth observing that from this perspective, e.g., a context-aware search is actually an end-user application that exploits the services of the context engine.

5.2 Minimal Responsibilities: Interpreting Context

The context engine architecture points out a clear distinction between end-user applications and the context engine, realized in the context engine API. Every application, however, does not need to know all details of context processing, or understand the entire context ontology. This is possible because context is expressed in syntax that all compliant applications can parse, even if they don't fully understand its semantics, perhaps even at the generalized upper level.
We may use the simple architecture to identify the minimal responsibilities of a context engine:
1. Answering questions about the current context.
2. Extending the currently known context.
3. Managing user preferences.
In addition, a context engine may be extended to adopt also other tasks. For purposes of this report, let us identify three:
4. Extending the locally known context with the help of other services (with indirectly accessible sensors).
5. Implementing event listeners for registering application (context) event handlers.
6. Implementing application-specific API extensions, e.g. to enquiry whether it is currently ok to disturb the user.
By design, the first three requirements coincide with the tasks of logic programming.

6. Conclusions

In this report we have described elements of context-aware computing, and discussed how and what contextual information is used. We looked at it from two different perspectives, applications development and research:

1- Practical development of applications: Contextual information used for application development is very limited (mostly positioning and time). In any case, developers have great access to physical sensorial information, but not many possibilities when accessing to external information (e.g. weather information from external server).
2- Other research approaches. We have reviewed other research work, and select a good base for our future research. Some of these are: SSN (Semantic Sensor Network) for representing sensorial information to enable reuse of sensorial information across multiple applications; and SOCAM (Service-Oriented Context-Aware Middleware), which presents a textbook architecture to manage contextual information.

The main component of this context-aware system is so-called context engine, responsible for managing and reasoning with contextual information.
In brief, we believe that the natural responsibilities of the context engine include extending and reasoning with the context, and managing user preferences. To support this claim, we have demonstrated how to actually interpret and compute with context in terms of logic programming. Acknowledging that necessary sensor information is readily available due to the current (mobile) sensor frameworks, this also points out a straightforward implementation strategy.
Finally, we have identified some open problems, including how to manage temporarily missing sensor information in modelling, and suggested recording and archiving (past) contexts as a potential approach. Other interesting research question is to see if power-consumption is prohibitive for developing context engines in practice. This is very interesting since battery-consumption is one of the biggest concerns in mobile computing. As suspected, these topics appear on our future research agenda.

7. References

Android Developer. Location and Sensors APIs,
http://developer.android.com/guide/topics/sensors/index.html

Anind, K. D., 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, 304–307. Springer-Verlag, 1999.

Augusto, J.C., Liu, J., McCullagh, P.J., Wang, H., Jian-Bo, Y (2008). Management of uncertainty and spatio-temporal aspects for monitoring and diagnosis in a Smart Home. International Journal of Computational Intelligence Systems, 1(4) 361-378 (2008)

Baldauf, M., Dustdar, S., Rosenberg, F. (2007). A Survey on Context Aware Systems. Int. J. Ad Hoc Ubiquitous Comput. 2, no. 4, 263–277.

Chen, G., Kotz, D. (2000). A survey of context-aware mobile computing research. Technical Report TR2000-381. Dartmouth College (2000).

DAML 2014. DAML Ontology Library. Available at http://www.daml.org/ontologies/

Fankhauser, T. (2012). iOS sensors, http://www.slideshare.net/thomasfankhauser/ios-sensors-15579340

Gu, T., Hung Keng, P., Da Qing, Z. (2005). A Service-Oriented Middleware for Building Context-Aware Services. J. Netw. Comput. Appl. 28, no. 1, 1-18

Hebeler, J., Fisher, M., Blace, R., Perez-Lope, A. (2009). Semantic Web Programming. Wiley

Mehra, P. (2012). Context-Aware Computing: Beyond Search and Location-Based Services. IEEE Internet Computing 16, no. 2, 12 - 16.

Mileo, A., Merico, D., Pinardi S., Bisiani, R (2010). A logical approach to home health-care with intelligent sensor-network support. Computer Journal
53(8): 1257-1276 (2010), Oxford University Press

Nykänen, O. (2009). Semantic Web for Evolutionary Peer-to-Peer Knowledge Space. In Birkenbihl, K., Quesada-Ruiz, E., & Priesca-Balbin, P. (Eds.) Monograph: Universal, Ubiquitous and Intelligent Web, UPGRADE, The European Journal for the Informatics Professional, Vol. X, Issue No. 1, February 2009, ISSN 1684-5285, CEPIS & Novática. Available at http://www.cepis.org/upgrade/files/issue%20I-2009-nykanen.pdf

Palmieri, M., Singh, I., Cicchetti, A. (2012). Comparison of Cross-Platform Mobile Development Tools. In 2012 16th International Conference on Intelligence in Next Generation Networks (ICIN), 179–186.

Protégé 2014. Protégé Ontology Library. Stanford.edu. Available at http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library

Strang, T., Linnhoff-Popien, C. (2004): A Context Modeling Survey. In Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England.
(Baldauf, 2007, Strang 2004)

Suman N. (2012). ACE: exploiting correlation for energy-efficient and continuous context sensing. In Proceedings of the 10th international conference on Mobile systems, applications, and services (MobiSys '12).

W3C Data (2014). W3C Data Activity. World Wide Web Consortium (W3C). Available at http://www.w3.org/2013/data/

W3C Semantic Web (2014). W3C Semantic Web Activity. World Wide Web Consortium (W3C). Available at http://www.w3.org/2001/sw/

W3C SSN (2011). Semantic Sensor Network XG Final Report. W3C Incubator Group Report 28 June 2011. World Wide Web Consortium (W3C). Available at http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/

W3C Standards (2014). W3C Standards. World Wide Web Consortium (W3C). Available at http://www.w3.org/standards/

Yndurain, E., Bernhardt, D., Campo, C. (2012). Augmenting Mobile Search Engines to Leverage Context Awareness. IEEE Internet Computing 16, no. 2, 17–25.