The Knowledge Engineering Review, Vol. 23:1, 21–49. © 2008, Cambridge University Press
DOI: 10.1017/S0269888907001269 Printed in the United Kingdom

Dynamic context management for pervasive applications

JÉRÔME EUZENAT[1], JÉRÔME PIERSON[1,2] and FANO RAMPARANY[2]

[1]*INRIA Rhône-Alpes & LIG, 655 avenue de l'Europe, 38330 Montbonnot Saint Martin, France*
E-mail: Jerome.Euzenat@inrialpes.fr
[2]*Orange Labs, 28 chemin du Vieux Chêne 38240 Meylan, France*
E-mail: {Jerome.Pierson, Fano.Ramparany}@orange-ftgroup.com

Abstract

Pervasive computing aims at providing services for human beings that interact with their environment (encompassing objects and people who reside in it). Pervasive computing applications must be able to take into account the context in which users evolve, e.g., physical location, social or hierarchical position, current tasks as well as related information. These applications have to deal with the dynamic integration in the environment of new, and sometimes unexpected, elements (users or devices). In turn, the environment has to provide context information to newly designed applications. This requires a framework which is open, dynamic and minimal. We describe an architecture in which context information is distributed in the environment and context managers use semantic web technologies in order to identify and characterize available resources. The components in the environment maintain their own context expressed in RDF and described through OWL ontologies. They may communicate this information to other components, obeying a simple protocol for identifying them and determining the information they can provide. We show how this architecture allows introducing new devices and new applications without interrupting what is working. In particular, the openness of ontology description languages makes possible the extension of context descriptions and ontology matching helps dealing with independently developed ontologies.

1 Introduction

In a pervasive computing environment, basic services can be provided by physical devices, e.g., sensors, effectors, or human-computer interfaces, for acquiring and rendering information. More advanced services are delivered by active devices that can act as service aggregators, e.g., computing an average temperature from the data of several sensors, to applications involving communication with numerous services and assisting human users. Pervasive computing requires the perception of the environment within which devices and users evolve, in particular, spatial and temporal location as well as information related to the physical environment in which services are available [31]. These elements are part of the context in which applications operate and can be considered common to all services. Even if the context is highly dependent on the task to be accomplished, a large part of context information, e.g., the already mentioned location, can be shared with other applications. Figure 1 synthesizes the relationships between applications, devices, services and context.

Although not strictly necessary to the service function, context generally impacts its choice and behavior. For instance, it is only useful to coordinate a camera and a microphone if they are in the same room. Moreover, depending on the context, services must be adapted. Taking context into account when delivering adapted solutions to users is indeed one of the main goals of pervasive computing. These context elements are very often described by ontologies and provision is made in the context management system for users and application developers to introduce their own ontologies. So far, proposed solutions are rather

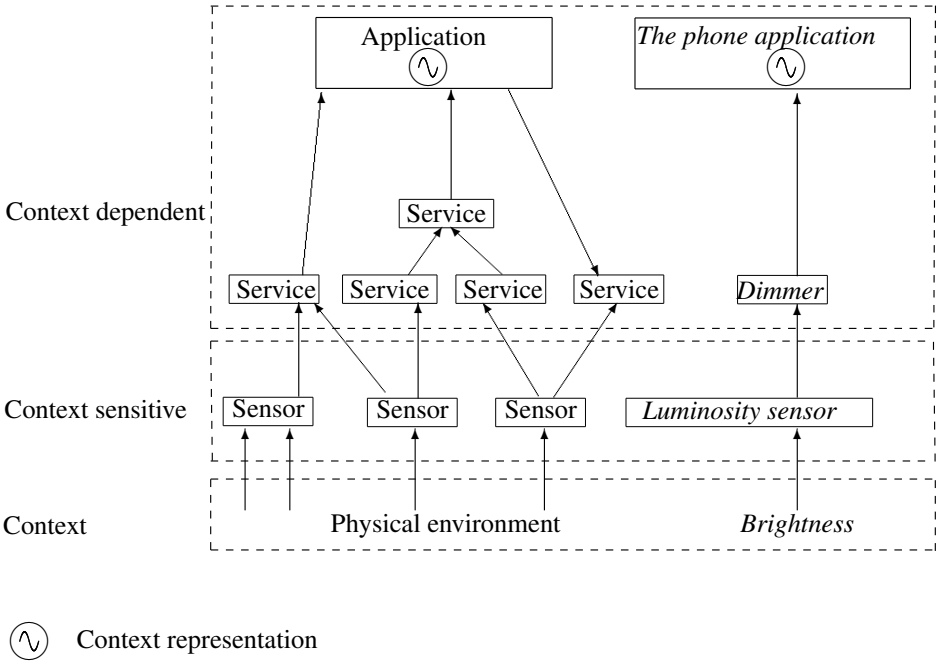

Figure 1 General architecture of a pervasive computing environment: sensors, e.g., Luminosity sensor, provide information about the environment (here the Brightness). These are provided as services enabling other services or applications to assess the context of some task (the luminosity at some place in the physical environment). Context representation is handled in applications which need it (mentions in italics denote examples, arrows denote information flow).

static as they cannot cope at runtime with applications needing or providing unknown context information [14, 13, 34, 30], but links between these ontologies must be established at design time. The same holds for new types of sensors that can be used and whose capabilities were not expected when designing services or applications. Thus a more flexible solution is required that allows new types of context information to be handled dynamically. In [9], the question of the evolution of contexts is raised. Our goal is to contribute solving this problem by using technologies enabling the use of open and heterogeneous context descriptions.

So, we aim at designing a context information management framework which is:

Open so that new devices and applications can be involved in the environment. It must thus rely on well accepted standards for expressing context information which guarantee that components will be able to interoperate.

Dynamic so that these devices and applications can be taken into account dynamically. This requires that the context manager can represent new types of context information and that it can match these representations so that old parties can take advantage of new ones and vice versa.

Minimal so that the framework does not put a non realistic burden on application and device developers. This requires to keep minimal the computing resources and specific interfaces needed for using this framework.

The two first goals have been considered by other efforts in pervasive computing [29]. More original is that of imposing minimal constraints to the applications and infrastructures, so that the context manager can be integrated in as many environments as possible. These constraints lead to reuse as many technologies as possible, to reduce the necessary interface to the strict minimum and to avoid requiring expensive computation. This also requires to find a trade-off between these requirements. Indeed, dynamics mandates the use of reasoning techniques in order to match service context needs while minimality would impose to keep these reasoning abilities to the strict minimum.

We define a context representation independent from applications and an architecture enabling application evolution. The openness of the system yields heterogeneous representations that have to be reconciled before being used. For that purpose, solutions developed for the semantic web are adopted.

We use the following vocabulary (see Figure 1): The *environment* denotes everything, i.e., the physical environment of the users, all devices within reach as well as the full description of the situation. By *device* is meant any material system available in the environment and able to communicate. A *sensor* is a context-sensitive device able to characterize some aspect of the environment; an *application* is a computer program dedicated to a user and a *service* is a computer program dedicated to other services or applications, i.e., a context aggregator or location service. This paper is concerned with context-aware applications, i.e., applications which adapt their behavior depending on the context [13].

This paper will first review existing work about context management in pervasive computing and artificial intelligence (§2). It then will explain how both approaches can contribute to a solution, though both taken in isolation have limitations. We propose a solution to this problem whose main ingredients are:

– A general representation of context information based on semantic web technologies and methods for accessing the context information (§3);
– An architecture for distributing context information so that it is possible to introduce new devices consuming and producing this information (§4);
– The use of semantic web techniques allowing to take advantage of the available information even if this was not anticipated at application design time (§5).

This paper is illustrated by a very simple scenario. Informally described here, it will be reconsidered at each step of the paper in order to show how it can be implemented (§6).

Example 1 (Motivating scenario)
Someone is quietly resting in her hotel room when a phone call occurs. The phone, detecting that the call is not of the utmost importance, will attempt to determine if it must ring, vibrate or route the call to an answering machine. In order to preserve the quietness of the user, it will determine her state (sleeping, in discussion with others, etc.). Information from the environment will support the decision: such as the consumer electronics appliance in use (computer, hair dryer, TV set), the presence of other people in the room, or the physical state of the environment (temperature, humidity, light).

Example 1 is a realistic scenario: most of the required sensors are available in today hotel rooms and providing them with wireless connectivity is considered by many appliance manufacturers.

2 Contexts

Context is considered here as the set of information (partly) characterizing the situation of some entity [14]. The notion of context is not universal but relative to some situation [15, 5]. This can be a physical situation, e.g., the spatio-temporal location of some person, or a functional one, e.g., the current task of the person or her hierarchical position.

Since context is not supposed to encompass all the information characterizing the situation (this is clearly not feasible), there is room for several context representations of the same situation. Moreover, these context representations can be compared on the basis of the information they provide, a context representation can be considered as more or less precise than another. Finally, a context representation must enable the aggregation and separation of context information so that a pervasive computing application can get in and out contexts: when users enter a building, the context related to the city they are in, is partially overruled by that of the building (which will be overruled by that of the room they are in, etc.). This building context must be forgotten upon exit of the building. This also means that several applications can share part of their context, so if the context is relative to the situation it does not depend on the application (instead the application is expected to be context-dependent). Hence, information such as pressure, temperature, sunlight, humidity and date are context elements (Figure 2). If one wants to implement a temperature service the context will be restricted to temperature, but for an air-conditioning

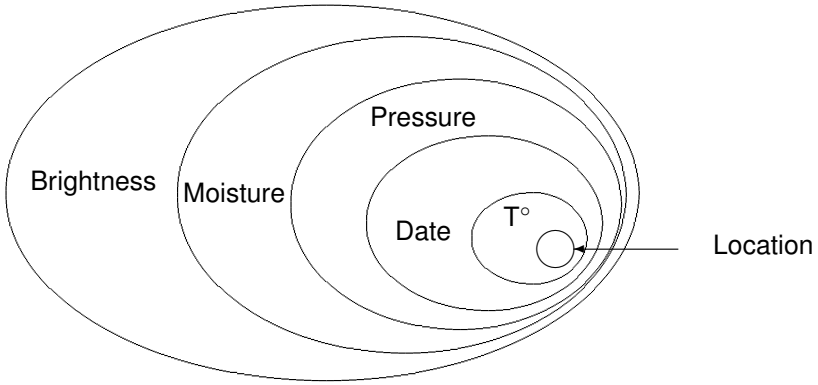

Figure 2 The application context of an air-conditioning monitoring station.

monitoring station, information of temperature, sunlight and date will be aggregated for regulating the heat sources. The context of a weather forecast service will gather the five types of information. Each set of Figure 2 is a context and can be necessary to various applications. There are also other possible contexts combining this rough information.

Although, several domains have considered the notion of context, the standpoints from which this notion is considered are different: in pervasive computing, the context of an application in terms of its physical parameters has been especially considered; in human-computer communication, the context is most often the user task and the history of the interaction with the computer [15]; in artificial intelligence, the context is rather considered as the conditions of validity of an assertion [25]. These points of view are not competing but complementary. Since technology and concepts from both pervasive computing and artificial intelligence will be used, more details are presented below.

2.1 Context in pervasive computing

In pervasive computing, the physical context is of the utmost importance. In general, it is acquired through sensor data, e.g., the temperature acquired by a thermometer. These data are further elaborated into context characterizations adapted to their use, e.g., "high temperature" for an air-conditioning controller. With regard to the sensor data (a temperature), the information has been weakened, i.e., made less precise, but more appropriate.

Several proposals have been put forth for providing context support for pervasive applications [14, 10, 29]. They offer middleware for connecting sensors to applications in a flexible way. Data coming from sensors is both routed through the architecture components and abstracted through the conceptual layers of Figure 3.

Although these architectures are generic enough for supporting any pervasive applications, they do not specify how to describe context information. Many proposals have naturally used ontologies for that purpose [40, 6, 7, 19, 24, 9, 27]. However, the use of context infrastructure in pervasive computing very often remains related to an application or a particular domain: "high temperature" is not an absolute characterization. It depends on the use of the room (a sauna or a sleeping room). This dependency is legitimate because the goal of context-aware applications is to help users in their context. But in consequence, pervasive computing tends to manipulate a characterization of the context in the perspective of an application and it is difficult for applications to share context information, i.e., to dynamically implement new applications with the characterization of context made for previous ones.

However, the issue of multi-application context modeling has now been addressed by the pervasive computing research community. [9] raises the issue of considering context independently from applications. Figure 3 shows the way to progressively elaborate context information from sensors to applications.

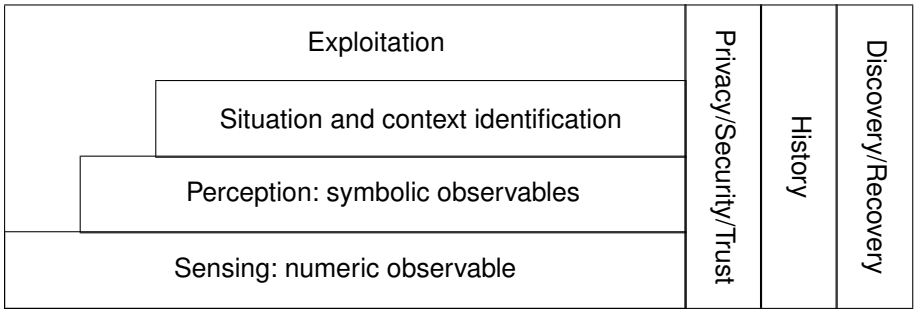

Figure 3 Model for context management in pervasive computing. Data coming from sensors are aggregated and elaborated into the context used by applications (reproduced from [9]). We do not consider the orthogonal aspects (discovery, history and security).

We will follow this approach and will precise the content of the perception and situation layers so that they can support the dynamic update of the environment (new sensors and applications).

2.2 Contexts in artificial intelligence

In artificial intelligence, the notion of context is, in general, concerned with the representation of information. It is used for accounting for two phenomena: the context of validity of information [11] and the efficiency of reasoning in narrower contexts [25].

A possible formalisation of context is based on context "reification" [33], i.e., considering contexts as objects, as well as the "meta-predicate" ist, $ist(p, c)$ meaning that assertion p is true in context c. This assertion being itself an assertion, it is possible to assert contextual assertions, e.g., $ist(ist(p, c'), c)$. It is also possible to reason across contexts: $ist(p, c) \Rightarrow ist(q, c')$ is totally different than $ist(p \Rightarrow q, c')$. More interestingly, it is possible to create new contexts. For instance, $assume(p, c)$ creates a context made from c assuming p. This allows to define hierarchical context structures corresponding to real life contexts. Context reification is particularly useful for navigating within contexts or reasoning about contexts.

The approaches of context in artificial intelligence allow grouping knowledge in micro-theories [25] and to reason within those. In this framework (that of Cyc), the context is a more precise frame for interpreting information. This kind of approach can be used in pervasive computing in order to integrate and interpret data provided by sensors. Taking advantage of the theory associated with the sensor enables reducing the ambiguity of the data it delivers. In that view, raw data issued from sensors, are generally not weakened but rather enriched (and aggregated with other information sources allowing to further precise their interpretation). [26] describes the way to express this kind of contexts within the semantic web by providing each triple with information on its origin ("quad"). The same model is implemented in modern RDF managers [31].

Although the former works consider contexts as independent theories related to some particular knowledge field, others consider contexts as concurrent viewpoints on the same information [22]. Thus contexts are independent logical theories which can be related by "mappings" used for importing information from some context into another. It is not possible to manipulate contexts as first class objects or to assert statements about these contexts. It is only possible to reason within a context with knowledge imported from other contexts. A comparison of both approaches is made in [39].

This approach can be useful in pervasive computing when several information sources provide comparable information, but it does not constitute an explicit representation of context (this is rather a representation of what is holding in some "context"). We will thus not consider this approach here. It has found its way within semantic web through the C-OWL language [3].

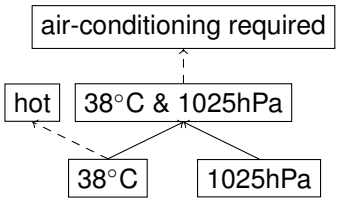

Figure 4 Context, aggregation (plain lines) and weakening (dashed lines). Data coming from sensors is manipulated and adapted to the needs of applications.

2.3 Synthesis

In summary, pervasive computing tends to consider context as what characterizes the situation while artificial intelligence rather characterizes the information itself. The goal of context in pervasive computing is to provide the most adapted service to users, while in artificial intelligence the goal is to be able to switch contexts and reason in different contexts.

In consequence, pervasive computing very often deals with the particular context of an application, i.e., the context of the users that the system is trying to help, thus information coming from sources is very often weakened in order to fit the application needs. Artificial intelligence rather tends to enrich further this information or to retain the most precise information. In artificial intelligence, the context is information centric, e.g., the context of validity of this information.

The validity of sources has also been considered in pervasive computing [28, 24], but not as part of the context, rather as a way to evaluate the quality of the gathered information: in the one case, the validity of context information is considered, in the other case, validity of information *is* the context. For instance, [38] identifies three subsorts of context: computing context, user context and physical context. None of these covers the context of validity of information. Both approaches consider different types of contexts, which explains why none of the mentioned artificial intelligence theories are used in pervasive computing or in adaptive interface.

Of course, nothing is so clear cut: these approaches are rather complementary than competing. In general, raw data can go through weakening and enrichment as shown in Figure 4. These approaches must be bridged further by offering an architecture that allows both views on context information.

2.4 Contexts, ontologies and the semantic web

As other works have shown [40, 6, 7, 19, 24, 9, 27], ontologies are appropriate tools for defining context information because their use does not require exact match of information requirements with available information. Ontologies will thus be used as a way to define the elements that can be found in context representation and the context elements that are sought by applications.

Semantic web technologies can be considered as a breed between knowledge representation and web technologies. They offer knowledge representation languages that are both expressive and open: two useful features for expressing contexts. In particular, openness allows the dynamic extension of ontologies and knowledge descriptions. Moreover, semantic web technologies come as standardized languages with available supporting tools. This lowers the barrier to their adoption.

The ground language for the semantic web is RDF (Resource Description Framework [32]). It enables expressing assertions of the form subject-predicate-object (called triples and denoted $\langle s\ p\ o \rangle$). It is possible to see an RDF document, i.e., a set of triples, as a labeled multi-graph. For instance, in Example 2, $\langle$ `#67` `ami:location` `#1345` $\rangle$ means that object `#67` is located within object `#1345`. `#1345` is also declared as an `HotelRoom` as defined in the namespace associated with the `ami` prefix (that of the AmiGo project). The strength of RDF is that the names of entities, either subjects, predicates or objects, are URIs (the identifiers of the web that can be seen as a generalization of URLs such as `http://www.w3.org/sw`). This opens the

possibility for different RDF documents to refer precisely to an entity (it is reasonable to think that a URI denotes the same thing for all of its users).

The OWL language [12] has been designed for expressing "ontologies" or conceptual models of a domain of knowledge. It eases the interpretation of RDF graphs concerning this domain. OWL offers to define classes of objects and predicates and to declare constraints applying to them, e.g., that the "output" of a "thermometer" is a "temperature".

In the semantic web, there exists no standard context management tools as defined above, though there are proposals like CDF [25, 31] or C-OWL [3]. However, the languages developed for the semantic web, and particularly RDF and OWL, are suitable for context representation in pervasive computing and particularly to the representation of dynamically evolving contexts for two reasons:

– These languages are open: they implement the open world assumption under which it is always possible to add more information to a context characterization. This is quite suited for dynamic and incomplete environments in which some unavailable information can become available. This is also adapted to a world in which new representations are dynamically added.
– These languages have been designed to work in a networked way. For that purpose, they use the global address space provided by URIs to avoid name conflicts when merging representations.

RDF and OWL will be used for representing context as illustrated below.

3 Modeling context within the semantic web

A simple context model is introduced for representing context information necessary to pervasive computing applications (§3.2). It is based on semantic web languages which were introduced above (§2.4) and more precisely defined below (§3.1). Their use within pervasive computing applications is further justified in the following sections (§4 and 3.3).

3.1 OWL ontologies for characterizing objects

As such, ontologies are not the context of applications but help characterizing these contexts. OWL ontologies are used in order to characterize the objects that can be found within the environment. There are general purpose ontologies such as SUMO[1], Cyc[2] or DOLCE[3] that can be used. The essential point is to have ontologies sufficiently generic to cover the various concepts involved in pervasive computing applications: resources, actors, places, dates, activities, permissions, etc. There are several ontologies of this type that have been designed for pervasive computing purposes [6, 42, 19, 7].

These ontologies are in general not very sophisticated because they were designed to make the machinery of pervasive computing applications work. For each domain, it is necessary to develop a more precise description of exploitable information. Thus, in [19], we proposed a semantic description of four different models (semantical, geometrical, graph theory based, and set theory based models) to represent indoor location information. To cover outdoor location, [20] proposed a geographical ontology which gathers several geographical datasets. [21] presents a semantic description of rules to selectively control who can access to contextual information and under which conditions. A spatio-temporal approach is developed in [35] to describe, with a dedicated ontology, user preferences in ubiquitous computing environments and their behavior routine. To describe users and their social relations, the FOAF[4] ontology is appropriate, as well as the GUMO[5] ontology described in [27].

Ontologies such as those of Figure 5 can be used by applications to characterize information which is necessary for them. Usually, devices will use the most precise refinements of these models to be exploited by the applications.

[1] http://reliant.teknowledge.com/DAML/SUMO.owl
[2] http://www.cyc.com/2003/04/01/cyc
[3] http://www.loa-cnr.it/ontologies/DLP_397.owl
[4] http://xmlns.com/foaf/spec/
[5] http://www.ubisworld.org/

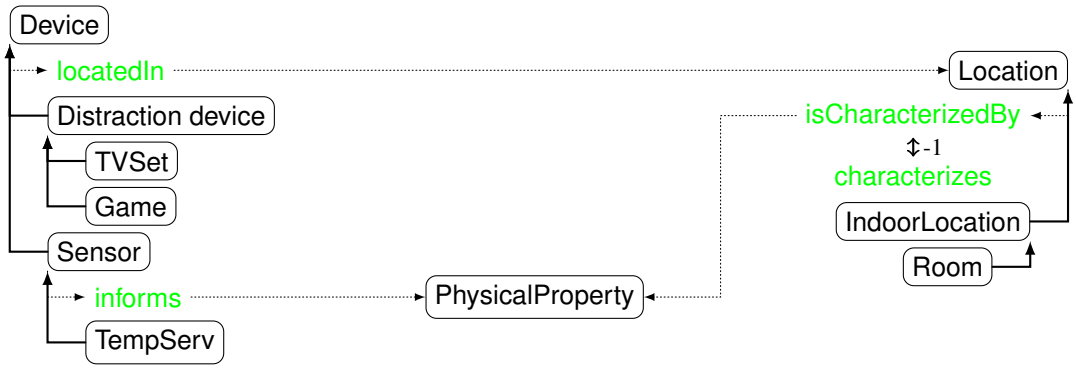

Figure 5 Sample of general purpose ontology concepts (classes are in rounded corner rectangles, properties are related by dotted arrows and plain arrows between classes denote sub-class relationships). The ontology provides classes for devices and locations as well as properties such as "locatedIn".

3.2 RDF graphs for modeling context information

At this stage, the context model is very simple: a context is a set of RDF assertions (see Example 2). The context elements will thus be considered as RDF graphs. The benefit of using a general purpose language like RDF is that a common general interface can be defined for context aware components that exchange RDF triples. Interoperability is then trivially guaranteed by considering that context-aware devices are consumers and producers of RDF.

Example 2 (A context)
Part of the information concerning the hotel room can be represented by the following set of triples:

$$\begin{array}{rrr} \langle & \texttt{self} & \texttt{ami:locatedIn} & \texttt{\#1345} & \rangle \\ \langle & \texttt{\#1345} & \texttt{rdf:type} & \texttt{ami:HotelRoom} & \rangle \\ \langle & \texttt{\#67} & \texttt{ami:locatedIn} & \texttt{\#1345} & \rangle \\ \langle & \texttt{\#67} & \texttt{rdf:type} & \texttt{tv:tvset} & \rangle \\ \langle & \texttt{\#67} & \texttt{tv:state} & \texttt{tv:on} & \rangle \\ \langle & \texttt{\#67} & \texttt{tv:volume} & \texttt{tv:muted} & \rangle \\ \langle & \texttt{\#67} & \texttt{tv:channel} & \texttt{tv:arte} & \rangle \end{array}$$

It means that the application is located in a hotel room containing a TV set playing Arte with the volume muted. The typing of entities (tvset, HotelRoom) is denoted by the use of the `rdf:type` *predicate and the combination of various vocabularies or ontologies is denoted by the prefix rdf, ami, and tv.*

Example 2 illustrates that context information may not be expressed with regard to the general purpose ontologies as presented in Figure 5.

Devices will need to extract only the relevant information from context sources. For that purpose, a query language like SPARQL [36] may be useful for querying or subscribing to context sources. In order for applications to know which devices to query, devices must publish the query types to which they can answer. This can be achieved by publishing the classes of objects and properties on which the component can answer. Ontologies are the natural way to achieve this and OWL is particularly suited for designing shared ontologies.

3.3 Matching context requirements and context information

If devices can be added at any time (which occurs when new people enter a room for instance), it is not clear that they are easily usable. There is no reason, a priori, that added devices as well as new applications are really compatible. Indeed, each newly introduced sensor will provide more precision or information which has not been considered at application design time. In the same way, the applications

cannot know all kinds of available sensors. Ontology description languages can help solving this problem. Fortunately, knowledge representation techniques in the OWL language always permit to specify a concept or a property without questioning those which existed originally.

Thus, to achieve its task, the application does not need to know that a television set is a 16/9 television set with an integrated hard disk if the application only needs to know if the user is watching television (or if there is an activity going on in the room). But nothing prevents the producer of this equipment from declaring it like this. This will characterize the television set precisely in the ontologies of television equipments, distraction devices and equipments which consume electricity. Figure 6 shows how the ontologies of Figure 5 can be extended for the purposes of using an air-conditioner in room 1345. So ontologies contribute to matching the available contextual information and application needs. In this sense, they help mostly when they are homogeneous, i.e., the available information is expressed in a restriction of the ontology in which the application need is expressed.

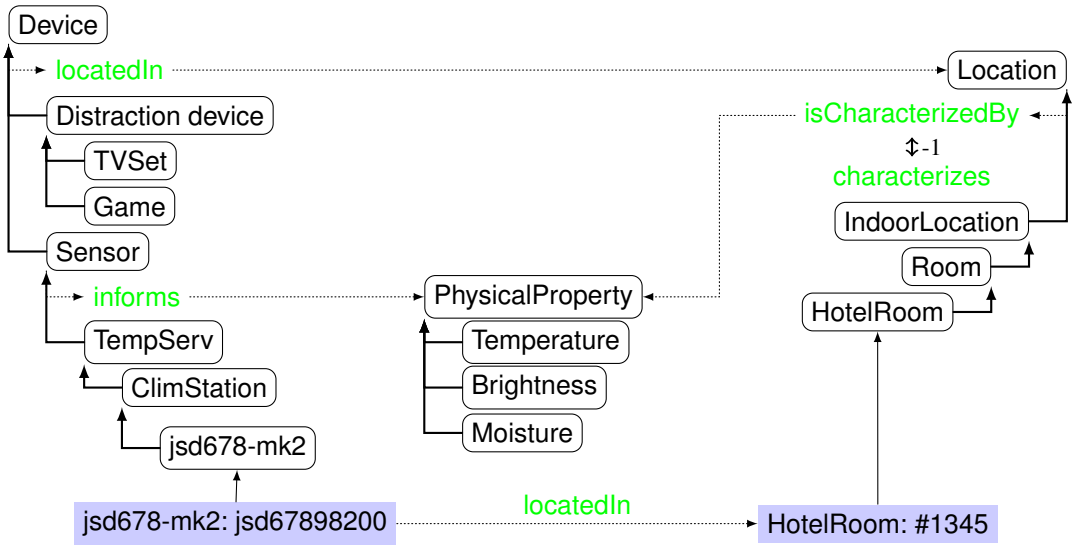

Figure 6 Specific ontologies related to the objects found in the environment. They refine the general purpose ontologies of Figure 5 with new concepts to be considered (instances are denoted by grey rectangles and linked to their classes by thin arrows).

In the given sample application, the important issue is that the class of the television set is a subclass of "Distraction device" so that the application can deduce an activity. On the other hand, if the purpose of the application is to quickly find a channel so that the user does not miss her favorite program, it is important to know the capabilities of the equipment and to ask it what it is capable of receiving.

Using ontologies to express context information permits a new equipment whose capabilities have not been known at application design time to enter and new applications to benefit from these possibilities. The applications must be as general as possible when describing their needed information (the room temperature, the distraction device activity) whereas the context management systems must be as precise as possible on what they produce. That will permit the most specialized applications to take advantage of them. This is illustrated by Example 3.

Example 3 (Matching queries)
Let assume that the application wishes to know the temperature in the room. A high level ontology enables to characterize its needs: the temperature is a physical property of the room; this property can be obtained by a ThermServer sensor, like a thermometer, located in the same room. The requested information (query) is expressed by the graph depicted in Figure 7 which corresponds to a set of RDF triples. Figure 8 presents the local context information that allows to answer the query (corresponding to the room temperature). The temperature will be obtained with the protocol to be presented in the next section.

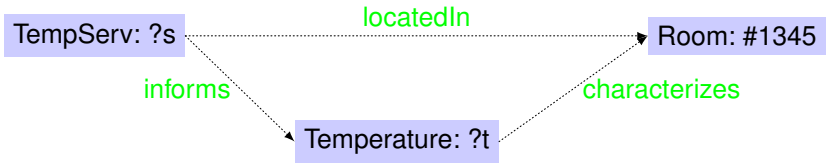

Figure 7 A query graph pattern (question marks instead of instance names introduce free variables). It can correspond to the information that an application requires from the environment (here the temperature measured in a room). This information is expressed in function of the specific ontologies of Figure 6.

The information is described by the device more precisely than in the query: The room (concept coming from a semantic model of location [19]) became a hotel room. In a similar way, the air-conditioning station, which is described as a temperature server, will play the role of a temperature sensor and provides temperature. So, ontologies are the glue which helps answering queries against sensor information.

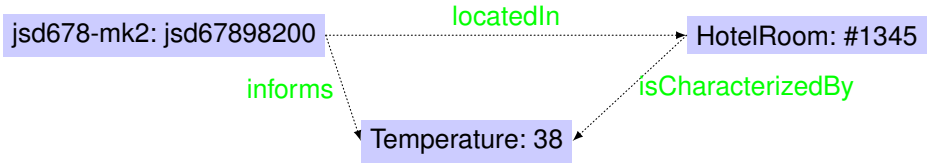

Figure 8 Local information that matches the information needs of the application expressed in Figure 7. The information as provided by the context sources are not an instantiation of the graph pattern of Figure 7, but by using the ontologies of Figure 5 and Figure 6 it becomes an answer to the query.

This example shows the benefit of using the semantic web technologies for dealing with context information: ontologies disseminated on the web provide the background knowledge necessary to interpret raw information.

Another benefit of using RDF and SPARQL is that services do not necessarily need to tell in advance to which query they are able to answer: applications can ask query that were not anticipated.

4 Context management architecture

In order to acquire context information, pervasive computing applications retrieve data directly or indirectly from sensors, or usually proxies, which are grounded in the physical environment. We propose an architecture which supports this process in the most flexible way, in the sense that applications will not need to directly connect to each available sensor and that adding a new sensor will not require all applications to be recompiled and redeployed and yet to be able to take advantage of the sensor.

4.1 Architecture

As seen earlier (Figure 1), collecting physical context information is done by exploiting sensor measurements. Such sensors include thermometers, light sensors, and generally do not embed enough computing resources for fully managing context. In this case, they are assumed to be accessed by pervasive computing systems through dedicated proxies that format and forward sensor information.

There are several alternative approaches for designing the target architecture. The first approach lets applications directly communicate with sensors they have an interest in [14, 38]. This approach requires applications to know in advance who to communicate with to get the required information. Furthermore it adds complexity to the process of information aggregation, as this process should then be handled by the applications themselves and overloads sensors activity. Finally this approach makes it difficult to insert new sensors into the environment and thus does not comply with the dynamicity requirement.

In the framework of service oriented architectures, the second approach consists of building a context management service [6, 23, 40] whose job is to collect sensor information and forward this information

to those applications that need it (Figure 9). This approach makes it possible to gather sensor information in a single place so that information could be easily aggregated. For example, a system that provides local temperature and atmospheric data is very useful in a home environment. At a city level, the same information is also useful; however it does not require the same degree of precision. The drawback of such systems is that they centralize the management of context information, which is contradictory to the concept of context. More specifically, this system provides information about the activity environment (a special case of context information), however this information is not contextual as it is independent from the current task or situation, i.e., that of the client application. Moreover, with such a system, the scope of context management would be only efficient in a limited physical area.

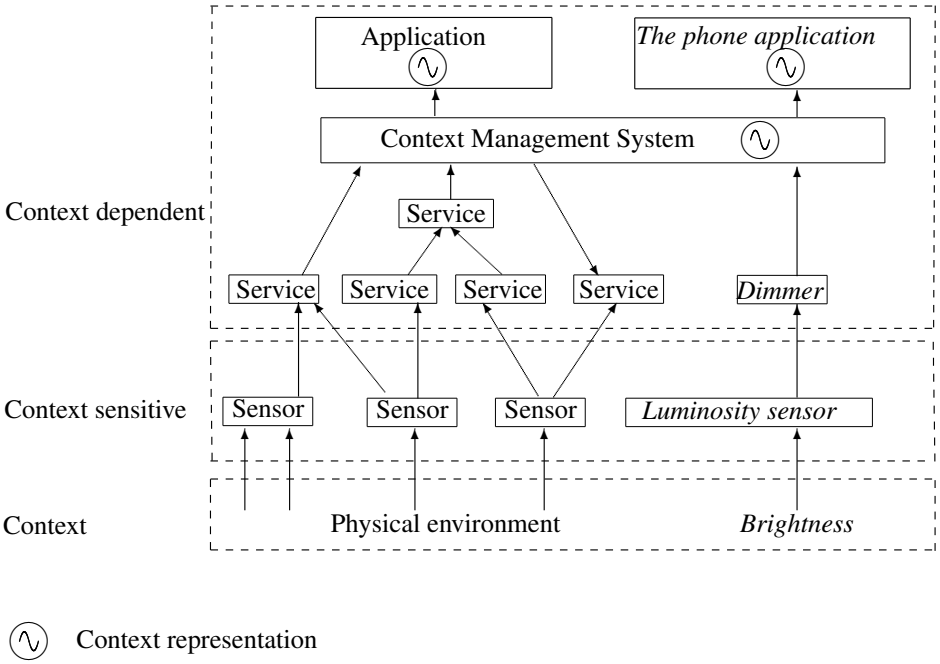

Figure 9 Centralised context-management system: context information coming from all sensors is aggregated in a central system which is contacted by all applications.

We have adopted a third approach in which each device or service embeds a context management component for maintaining context information for its own use or for the benefit of others (Figure 10) as it is done in [10, 29]. These devices can publish their capacities to service directories, so that they can be easily localized, or connect to a dedicated bus. The main advantage of this approach is that new devices can come in or go out online, without having to recompile or reinitialize any part of the whole environment.

Context management is then implemented as a library that could be embedded in any device from Figure 10. This library provides mechanisms for helping active devices to obtain context information from context sensitive devices.

4.2 Interaction

Applications should be able to query context information they are interested in and some services should be able to provide context information, such as aggregated context information to other devices.

For that purpose, we have designed a minimal protocol that makes the best of available services. It requires to be able to identify a service, to know what kind of context information it could provide and to interact with it to get access to this information. Thus the context management component provides a few methods. In the following descriptions, the first element is the query, the second is the response type:

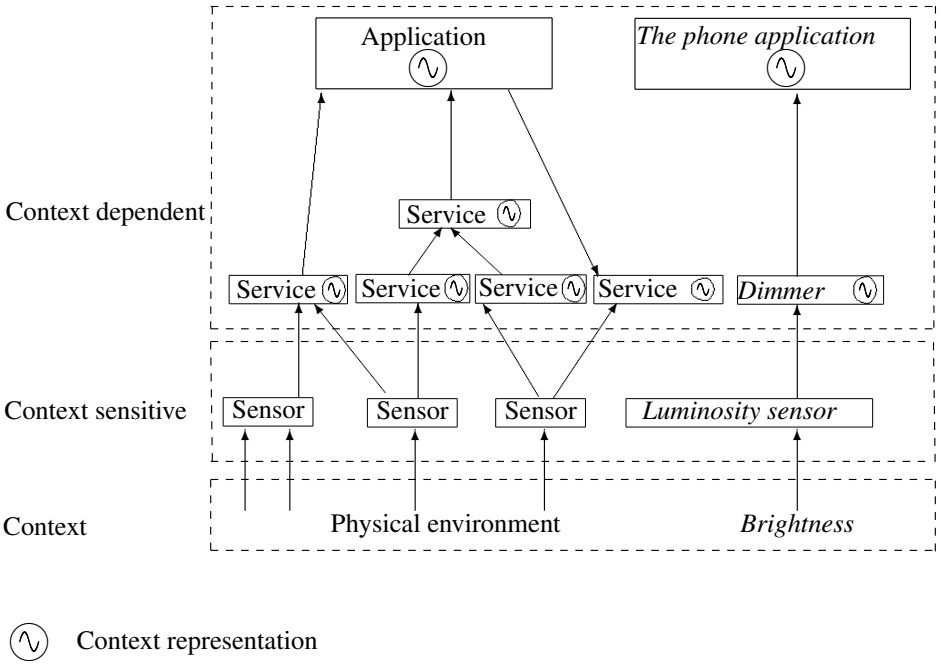

Figure 10 A context information manager component is embedded in each device and application: any component can embed a context management library which it uses for its own purposes. It can also use this library for providing context information to other components.

`Id()`→ `URI` asks for the identifiers of services available in the environment. This query is usually broadcast by the application since it does not know the identity of the services. This can be achieved either by an effective broadcast on a local network or by connecting to a directory.

`Cl(URI)`→ `URI` asks a particular service for its class. The identification of the class is a URI. It should allow the application to find this class description either on the web or in its local memory. Again, this information could come from the service itself or from a directory.

`Desc(URI)`→ `OWL` asks for the description of the information that the service can provide. This is usually given by a set of class and property identifiers in the OWL sense.

`Req(SPARQL)`→ `RDF` asks a query in the SPARQL language to a particular service. The query is usually a very simple query involving directly the classes and properties that the service has declared to be able to provide. It is returned as a SPARQL answer.

These methods allow identifying devices that are available in the environment (usually, using Bluetooth or another ad-hoc network protocol). The identifier can then be used to contact the device. Alternatively, it could be used to get a more detailed description of the device, e.g. in case the identifier is a URI identifying an object whose description can be stored on a network (directly through a URL or through some URI resolution mechanism). The second method identifies the class, in OWL terminology, of the device. In theory, this class should be accessible from the network and once its definition is found, it provides a detailed description of the device. For instance, manufacturers of each device can be expected to publish its features and description (in a specific query that the device could receive). The two first methods are mandatory. They are quite reasonable as they form the basis of other efforts such as the RFID/EPC protocols [41].

However, devices which operate without a globally available description, should provide the description of context information they offer. The method Desc provides this information in an OWL like language. The fourth method is used to post queries the device understands and to get the returned information. Thus any application is able to:

1. Find out, in its environment, services prone to provide information relevant to its own context.
2. Get features of services that have been found (for example, measurements precision).
3. Connect to the selected service to get the information sought.

This is illustrated by Example 4:

Example 4 (Service invocation)
Back to the phone handset example, the handset will first look up for available services in its immediate neighborhood:
`Id()`
Two services will send back their URI:
→ `http://tvbrand.com/3d23455651p/127765543`
→ `http://appliance.com/sdh/com/jsd67898200`
The first one is a television set from class `http://tvbrand.com/3d23455651p/`*. The second one is an air-conditioning control system. They could be further explored through the following method calls:*
`Cl(http://tvbrand.com/3d23455651p/127765543)`
→ `http://tvbrand.com/3d23455651p/`
`Cl(http://appliance.com/sdh/com/jsd67898200)`
→ `http://appliance.com/climctl/jsd678-mk2`
More information about these device capabilities can be found on the web. For instance, by finding an ontology describing `http://tvbrand.com/3d23455651p/` *as a particular type of high definition television set which can provide applications with their state and control over their behavior. The air-conditioning control system is able to provide high level information about its functionality (such as the uselessness of cooling an empty room). It is also able to transmit information its sensors have aggregated. In particular, it is able to check that all lights have been switched off. For example,* `Desc(http://appliance.com/climctl/jsd678-mk2)` *will return:*
→ `http://ambient.org/light/#GlobalLight`, `http://ambient.org/light/#state`
This will make it possible to answer the following query:
`Req(SELECT ?s WHERE ⟨amb:GlobalLight amb:state ?s⟩)` *by :*
→ {⟨`amb:off`⟩}
This informs that all lights have been switched off, so the user should probably be sleeping, hence it is better to not wake her up.

The main advantage of this approach is to avoid modifying devices when using them for building new applications, and conversely to avoid modifying applications for using new devices.

These primitives only rely on minimal requirements, drastically limiting the constraints on devices (or their proxies). In particular, the languages used are widespread standards (URI, RDF, OWL, SPARQL). Only Req(SPARQL) relies on inference capabilities, the other queries only require that the device returns static values. These inference capabilities can be more restricted than expected by the SPARQL language (for instance, by limiting queries to restrictions of the advertised classes and predicates).

5 Taking advantage of heterogeneous resources

The proposed context management system makes it possible to introduce new devices in the environment by extending the ontology in such a way that existing applications can make the best use of them. However, this view holds if all parties share the same ontology. Unfortunately, this is not always the case.

5.1 Context model interoperability in pervasive computing environments

Indeed, two sensor manufacturers will probably use two different ontologies to describe their products. Moreover, they will probably use domain ontologies related to the scope of their applications. For example, a location service will use a location ontology to describe the context information it can produce, and a user activity monitoring service will use a social ontology and an action ontology to describe its

needs. But the second service will probably need the context information produced by the first one and it needs a way to infer the compatibilities between their context information models.

Agreeing on standard universal and self-contained context ontologies is not a reasonable assumption. This raises the issue of matching context information produced and application context information requirements. There are three alternative approaches to address interoperability in pervasive computing environments:

– A priori standardization of ontologies: as mentioned earlier, this is not a reasonable option. Furthermore, such an approach will probably hamper the development of ontologies and technology.
– Setting up mediators among ontologies: on either a distributed mode or centralized one, matching from one ontology to the other could be made available (by sensors manufacturers, application developers or even end users) and applications could exploit them if they are facing the issue of matching two ontologies.
– An online ontology alignment service: this service matches two ontologies upon request.

These three approaches are not incompatible and might even be used concurrently. For example, parties could agree on sharing common high level ontologies and leaving more specific ontology evolve freely and independently. This is a strategy enabling a close account for a fast evolving domain.

5.2 Ontology alignment service

Ontology matching consists of finding the correspondences between two ontologies. It results in a set of correspondences, called an alignment, that can be used for various purposes such as merging ontologies, transforming data or querying. Ontology matching is actively researched and many algorithms have been provided for finding correspondences [18]. In context management, alignments can be used for two purposes:

– Finding out correspondences between two semantic context models which are represented using different ontologies to infer the compatibility between a context information producer and a context information consumer. This is useful for context information consumer, i.e., an application, to select the best context information producer, i.e., the sensor.
– Transforming queries from the applications to the devices and transforming the answers back. This is called query mediation.

An ontology alignment service [17] can be used. The goal of such services is to help agents (context managers in this case) to find an alignment between different ontologies they face. These services provide mechanisms for:

– Archiving (and retrieving) past alignments;
– Dynamically matching two ontologies;
– Translating queries and answers to queries between context managers that use different ontologies;
– Finding out an ontology close to a specific ontology (this can be useful for finding intermediate ontologies which will facilitate matching).

As for ontologies, alignment services should be made available to applications and context managers through network access. Alignment services use a functional interface that allows the explicit handling of ontologies alignments that have been developed in the framework of the semantic web [16].

Example 5 (Matching ontologies and mediating queries)
Consider the previous air-conditioning control system example. This system uses a specific air-conditioning domain ontology (`http://clim.org/ontologies/devices.owl/v1.1`*). On the other side the phone handset uses an ontology which models specific human situations (*`http://psycho.univ.edu/attitudes/onto.rdf`*). At this stage, there is no reason for these two ontologies to be identical.*

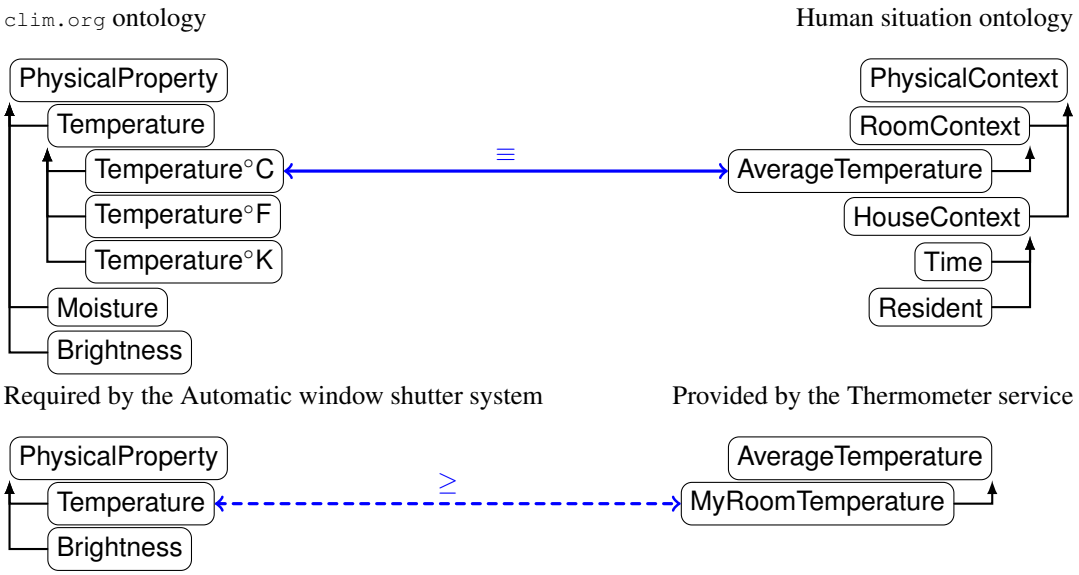

Figure 11 To infer relations between its required context information model and the model of a context information provider, the automatic window shutter service asks to an alignment service correspondences between its basic ontology (`clim.org`) and another basic ontology used by the context information provider (Human situation ontology). The equivalence relation (≡) returned by the Alignment server enables deducing that Temperature is more general than the MyRoomTemperature (and thus that the latter can be used as the required temperature information).

However, the phone handset simply needs to know if a person is too busy to be disturbed and, as air-conditioning systems are commonplace and nearly everyone owns a phone handset, it is likely that these two ontologies have already been aligned. This time, the phone will query the alignment service as follows (see Figure 11):

Align(http://clim.org/ontologies/devices.owl/v1.1,
http://psycho.univ.edu/attitudes/onto.rdf)
→ *al23889*

The service returns an alignment. Note that the requester is not aware if this alignment has been retrieved or computed online. The protocol allows specifying the parameters, the type of algorithm to use, or whether the alignment should be looked up in the library first or not. The result returned is not the alignment itself, but an identifier that allows the requester to access it if necessary. The application is then able to ask for a translation of the query to the context manager of the air-monitoring control system.

Translate(al24889, "SELECT ?s WHERE ⟨psy:room psy:lighting ?s⟩")
→ *SELECT ?s WHERE ⟨amb:GlobalLight amb:state ?s⟩*

It will post the query to the air-monitoring control system and ask for the correct interpretation of the returned answer:

Translate(inv(al24889), {⟨amb:off⟩})
→ {⟨*psy:dark*⟩}

Example 6 (Alignment between the clim.org ontology and the human situation ontology in XML format)

See a graphical representation of this alignment in Figure 12.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<rdf:RDF xmlns="http://knowledgeweb.semanticweb.org/heterogeneity/alignment#"
       xml:base="http://knowledgeweb.semanticweb.org/heterogeneity/alignment#"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
<Alignment>
  <xml>yes</xml>
  <level>0</level>
```

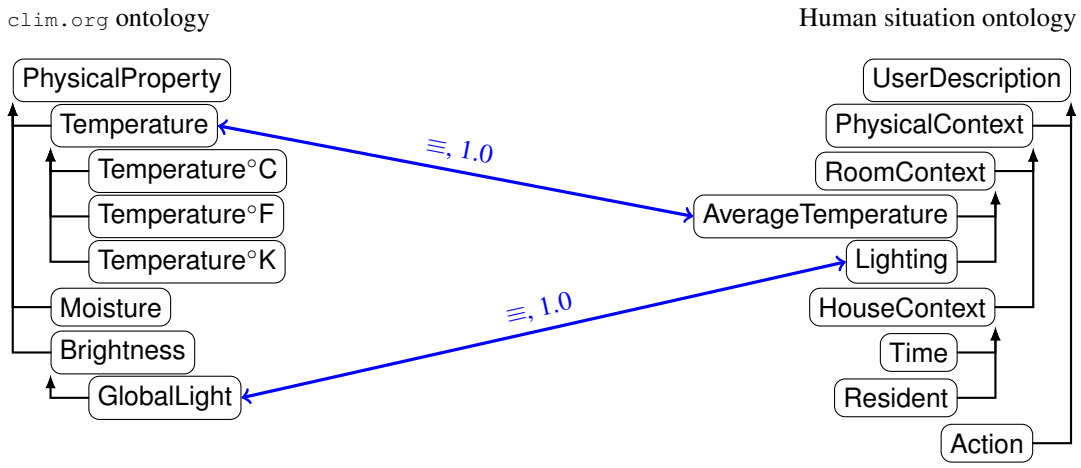

Figure 12 Matching the air-conditioning domain ontology and the human situation ontology yields two correspondences. An equivalence relation with a degree of confidence of 1.0 between the concept Temperature and the concept AverageTemperature, and an equivalence relation between the concept GlobalLight and the concept Lighting with a degree of confidence of 1.0. See the same alignment in a XML format in Example 6.

```
  <type>**</type>
  <onto1>http://clim.org/ontologies/devices.owl/v1.1</onto1>
  <onto2>http://psycho.univ.edu/attitudes/onto.rdf</onto2>
  <uri1>http://clim.org/ontologies/devices.owl/v1.1</uri1>
  <uri2>http://psycho.univ.edu/attitudes/onto.rdf</uri2>
  <map>
    <Cell>
      <entity1 rdf:resource="http://clim.org/ontologies/devices.owl/v1.1l#GlobalLight"/>
      <entity2 rdf:resource="http://psycho.univ.edu/attitudes/onto.rdf#Lighting"/>
      <relation>=</relation>
      <measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
    </Cell>
    <Cell>
      <entity1 rdf:resource="http://clim.org/ontologies/devices.owl/v1.1l#Temperature"/>
      <entity2 rdf:resource=http://psycho.univ.edu/attitudes/onto.rdf#AverageTemperature"/>
      <relation>=</relation>
      <measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1.0</measure>
    </Cell>
  </map>
</Alignment>
</rdf:RDF>
```

Alignments may be more precise than this one by providing confidence about the given correspondences and other relationships such as “more general than” [16].

6 Implementation and experimentation

The proposed context management system is implemented as a library which is first described (§6.1). Then, how this implementation actually performs the example is presented in more detail (§6.2).

6.1 A library for context-aware applications and context-sensitive devices

In order to provide a general interoperable framework to help developers to build context-sensitive devices and context-aware applications, a set of Java interfaces providing a common API is defined (see Figure 15). This API has been implemented and is provided as a library to be embedded in applications.

Interfaces ensure compatibility between components with different implementations. It enables them to take advantage of the use of the discovering protocol and it ensures that they use a semantic context information model. The *ContextManager* interface describes the component embedded in context-sensitive applications. It is identified by a URI and can be setup with a *ContextConsumer*, a *ContextProducer*

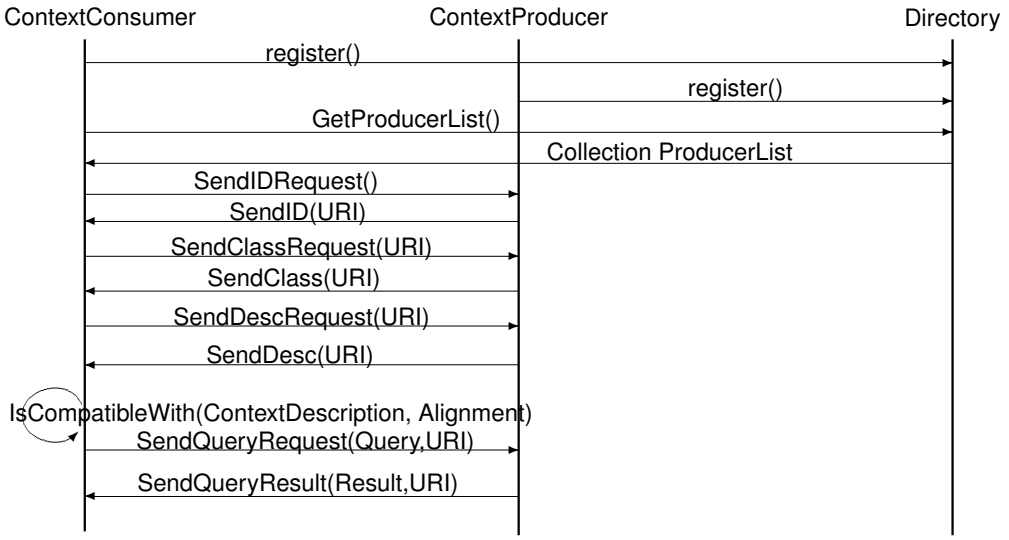

Figure 13 The activity diagram corresponding to the protocol.

object or both. These interfaces respectively stand for a context-aware component and a context-sensitive component. Each of these interfaces has methods to manipulate *ContextDescription* objects which are the semantic description of the context information it seeks or provides. This description is identified by a URI. It can be updated during the process and a method is provided to find the set of correspondences between two *ContextDescription* objects. This method returns an *Alignment* object. To get this alignment, the infrastructure queries an alignment server (see Section 5.2). First, a *ContextConsumer* asks to the server to get an alignment between its own *ContextDescription* ontology and the semantic context model of a *ContextProducer* that it obtained after the execution of the protocol (see Section 4.2). Then, thanks to this *Alignment*, representing the set of correspondences between these two ontologies, the context information consumer can decide which context information producer to select. It then can create a *QueryMediator* object to translate its query according to the *Alignment*.

ContextConsumer and *ContextProducer* interfaces provide methods in order to communicate using a *Message* interface and are useful to execute the protocol. There is one method at each step of the protocol for sending the corresponding message (Figure 13). Moreover, there are methods to subscribe and unsubscribe to a local service directory. In the *ContextConsumer* interface, methods enable to create, manage and update a list of the known *ContextProducer* instances, and a method is required to know if a *ContextConsumer* object and a *ContextProducer* object are compatible. In the *ContextProducer* interface a method is required for evaluating a SPARQL context query and getting back a *Result* object.

This framework has been implemented using the Java Agent DEvelopment framework (JADE)[6]. Each *ContextManager* is a JADE Agent and the *ContextConsumer* and the *ContextProducer* interfaces are implemented as agent behaviors which are independent processes. The implementation of the *Message* interface uses the JADE messaging system. So, the messages which are exchanged between agents can be sniffed by using JADE sniffer tools (see a snapshot of the GUI in Figure 14). The *ContextDescription* interface implementation is based on the Jena[7] RDF API library. The context information description is coded with the Jena ontology model and uses ARQ[8], the Jena SPARQL implementation, as query engine. The Alignment API[9] and its Alignment Server is used for filling the role of this component. The JADE connection with the Alignment Server has been specifically implemented for that purpose so that the Alignment server is seen as a JADE agent.

[6]http://jade.tilab.com/
[7]http://jena.sourceforge.net/
[8]http://jena.sourceforge.net/ARQ/
[9]http://alignapi.gforge.inria.fr/

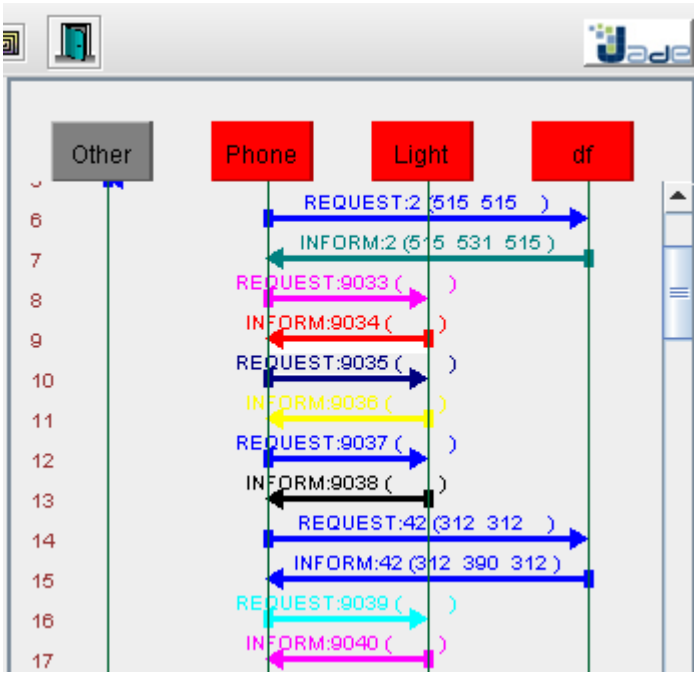

Figure 14 JADE sniffer interface. The phone agent is a context information consumer, the Light agent is a context information producer and the DF agent is the JADE directory agent. First, the consumer asks to the directory agent for all producers in the environment. Then, it asks all producers their IDs, their classes and their semantic context models. If it is interested in their context information, it sends them SPARQL queries.

In the current state, the infrastructure is fully functional, but it needs several tests in different environments to ensure its robustness. It could be added context reification and context manipulation operations. These manipulations will be supported by the architecture which enable components to acquire their context by combining those of other components. We also plan to produce some alignments between common pervasive computing domain ontologies, such as the Amigo[10] project ontologies, [24] or [40], or common general purpose ontologies, such as SUMO or CYC, and to publish them on the web. Then, these alignments could be exploited by an alignment server.

6.2 Experiment: The hotel room scenario

The proposed framework has been experimented in a simulation environment (agents simulate devices but use the software presented here). This experiment took advantage of existing ontologies to describe context information.

There are three context information producers in the environment of the subject's hotel room:

– A TV set which provides its information status, i.e., on or off,
– a light controller which provides the level of brightness of the room, and
– a location service which provides the number and the identification of the users who are in the room.

And there are two context information consumers in the environment:

– A phone which needs the number of users in the room, their activities as well as which devices are running, and
– the TV set which needs the same context information.

[10]http://amigo.gforge.inria.fr/home/index.html

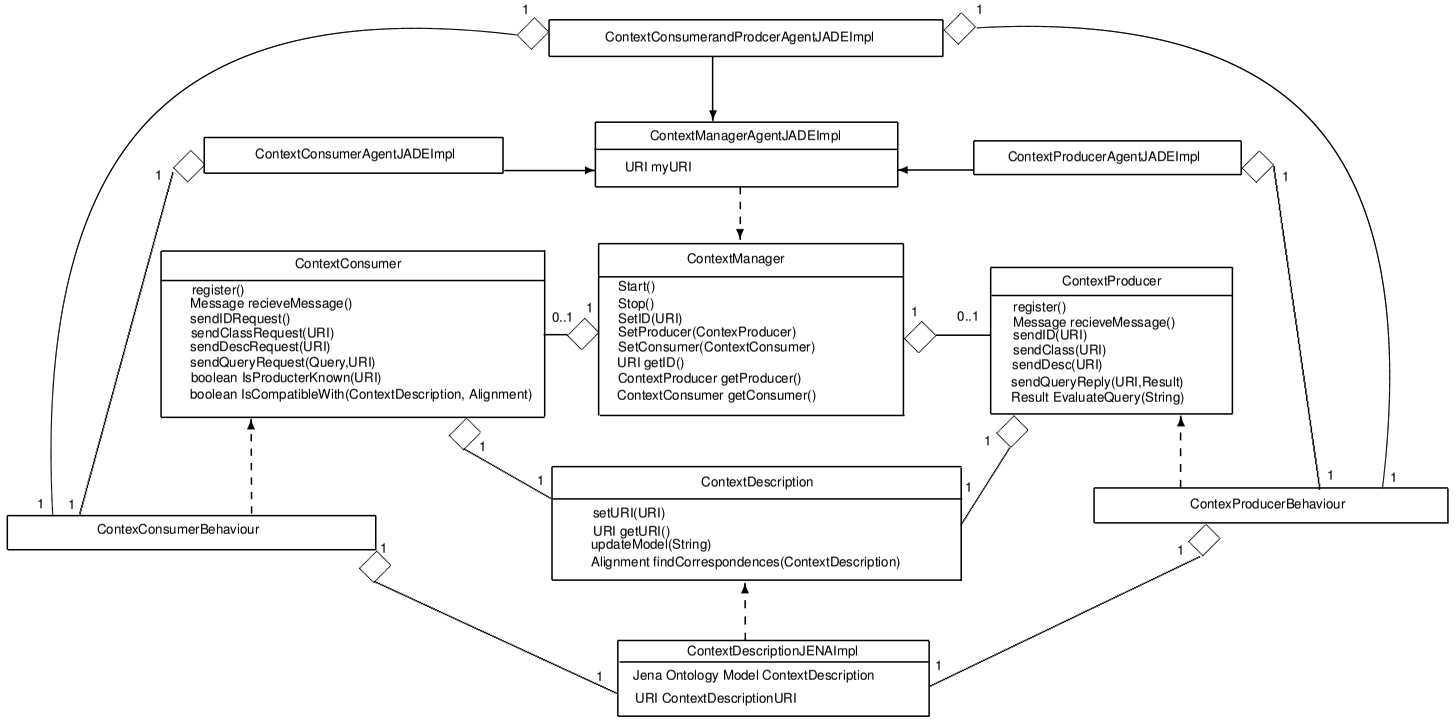

Figure 15 A class diagram of the main classes of the library. Three simple implementations of Agent (Consumer, Producer and Consumer and Producer) are provided to embed in a component. Plain lines stand for *extends* and dashed lines stand for *implements*.

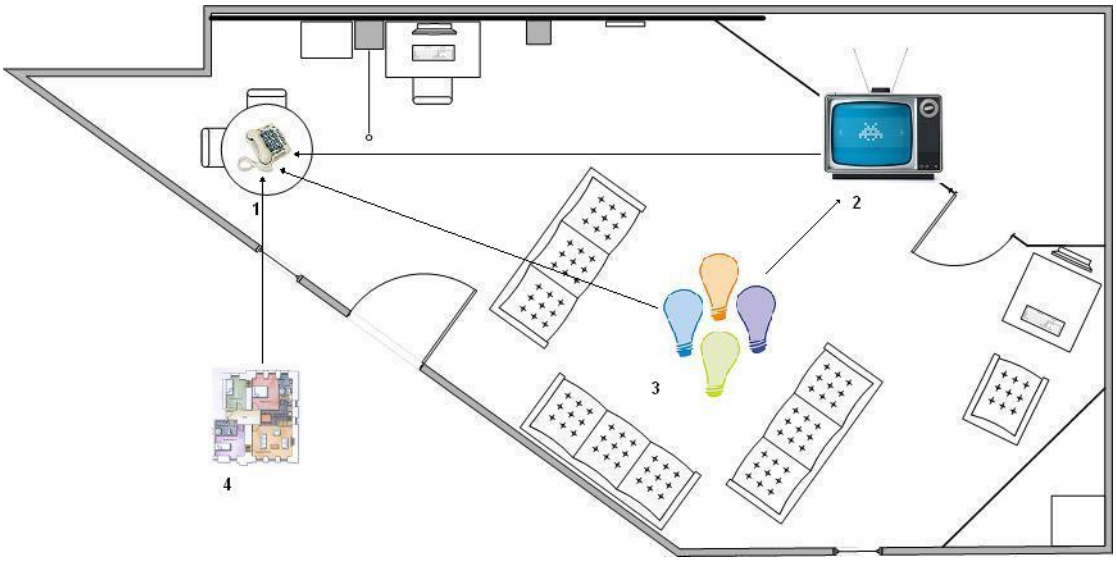

Figure 16 At the beginning of the experiment, the three context information providers, the television set (2), the light controller (3) and the location service (4), are connected to the phone (1).

When a phone call occurs, the phone will attempt to determine the user state (sleeping, in discussion with others, etc.) and adapt its behavior. Information from the environment will support its decision. The phone describes its needs with the following OWL description:

```
<owl:Class rdf:ID="PhoneContextDescription">
  <rdfs:subClassOf rdf:resource="ContextDescription"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&phc;action"/>
      <owl:allValuesFrom rdf:about="&pervact;#Activity"/>
    </owl:Restriction>
  </rdf:subClassOf>
  <rdfs:subClassOf>
   <owl:Restriction>
     <owl:onProperty rdf:resource="&Loca;#numberOfPeople"/>
     <owl:cardinality rdf:datatype="&xsd;#nonNegativeInteger">1</owl:cardinality>
   </owl:Restriction>
 </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&phc;activeDevices"/>
      <owl:allValuesFrom>
        <owl:Class>
          <owl:intersectionOf rdf:parseType="Collection">
             <owl:Class rdf:about="&amidom;ActivityMarker"/>
             <owl:Restriction>
               <owl:onProperty rdf:resource="&amidom;on"/>
               <owl:hasValue>true</owl:hasValue>
             </owl:Restriction>
          </owl:intersectionOf>
        </owl:Class>
      </owl:allValuesFrom>
    </owl:Restriction>
  </rdf:subClassOf>
</owl:Class>
```

This phone ContextDescription has three properties: the actions that are performed, the number of people in the environement and the marker of activities, i.e., some active devices. The application will attempt at filling its context needs by searching them in the environment through the protocol. It discovers the three context information producers and receives the following context information descriptions (see Figure 16).

The model of the context information that is produced by the light controller is:

```
<owl:Class rdf:ID="LightContextDescription">
  <rdfs:subClassOf rdf:resource="ContextDescription"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="http://www.ambient.org/Light#state"/>
      <owl:cardinality rdf:datatype="&xsd;#nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

The model of the context information that is produced by the location service is:

```
<owl:Class rdf:ID="AmilocContextDescription">
  <rdfs:subClassOf rdf:resource="ContextDescription"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&pervloc;#LocationContext">
      <owl:cardinality rdf:datatype="&xsd;#nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="http://www.ambient.org/measurable#Population">
      <owl:cardinality rdf:datatype="&xsd;#nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

The model of the context information that is produced by the television set is:

```
<owl:Class rdf:ID="TVContextProduceDescription">
  <rdfs:subClassOf rdf:resource="ContextDescription"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&amidom;#hasDeviceStatus">
      <owl:cardinality rdf:datatype="&xsd;#nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

As they do not use the same ontologies to describe their context information, the phone asks, to an alignment service, correspondences between its model and those of the other devices. It receives the alignment partly described in Section 5. Thanks to this alignment, it can decide which context information producers are the appropriate ones. Then, it can ask them queries to get the context information.

The queries sent to the light controller and their replies are:

```
Translate(al58963, "SELECT ?s WHERE <amidom:ActivityMarker amidom:on ?s>")
→ SELECT ?s WHERE <amb:GlobalLight amb:state ?s>
Translate(inv(al5863), {<amb:on>})
→ {<true>}
```

The queries sent to the television set and their replies are:

```
Translate(a589632, "SELECT ?s WHERE <amidom:ActivityMarker amidom:on ?s>")
→ SELECT ?s WHERE <amigo:DistractionDevice amigo:hasDeviceStatus ?s>
Translate(inv(a589632), {<amigo:off>})
→ {<false>}
```

The queries sent to the location service and their replies are:

```
Translate(a678523, "SELECT ?s WHERE <Loca:Context Loca:NumberOfPeople ?s>")
→ SELECT ?s WHERE <amiloc:NumberofPeople amb:Population ?s>
Translate(inv(a678523), <1>)
→ {<1>}
```

Thanks to this information, the phone can infer that the user can be interrupted by the call and it can decide to ring.

Later, another user enters the room for having a meeting with the first user. This user carries a smartphone which is also a context information producer providing the user's schedule and a context information consumer seeking the number of people present in the room in order to adapt its behavior as the other phone does.

The model of the context information that is consumed by the smartphone is:

```
<owl:Class rdf:ID="SmartPhoneContextConsumeDescription">
  <rdfs:subClassOf rdf:resource="ContextDescription"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&amigo;#hostedUser">
      <owl:cardinality rdf:datatype="&xsd;#nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

The model of the context information that is produced by the smartphone is:

```
<owl:Class rdf:ID="SmartPhoneContextProduceDescription">
  <rdfs:subClassOf rdf:resource="ContextDescription"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&amigo;#involvesActivity">
      <owl:cardinality rdf:datatype="&xsd;#nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

After it has executed the protocol, the smartphone had discovered the three context information producers and had received their context information descriptions (see Figure 17). Thanks to the alignment server, it finds that the location service can provide the context information it needs. At the same time, the phone application has discovered the smartphone and decided to use it as a context information producer to know what the smartphone user is doing. It can query it about this user's schedule and try to deduce his activity, as shown by the following query.

The queries sent to the smartphone by the phone application and their replies are:

```
Translate(a444586, "SELECT ?s WHERE <pervact:Action pervact:Name ?s>")
⟶ SELECT ?s WHERE <amigo:User amigo:involvesActivity ?s>
Translate(inv(444586), <amigo:Meeting>)
⟶ {<pervact:Meeting>}
```

If a phone call occurs during the meeting, the phone knows that users are in a meeting and do not want to be interrupted unless it is an emergency call.

7 Related work

As mentioned previously, there are many different management systems for context information. Dey and colleagues [14] have worked early on a general architecture to help designers in their context-aware computing applications building task. They provide the Context Toolkit composed of a set of context abstractions which can be used to construct context-aware applications. The context widget abstraction encapsulates acquisition from sensors and handling of a piece of context information through methods for several applications. It can be combined with interpreters in order to transform low-level information into higher-level information or with aggregators to gather related context information about relevant entities (people, places and objects) to best fit application requirements. To finally construct these applications, they provide discoverers to locate suitable widgets, interpreters, aggregators and services. They are used to connect applications to actuators in the environment. Context abstractions are implemented as Java objects using a basic communication protocol based on HTTP and XML. This toolkit is a powerful tool for designers who want to construct a context-aware environment. It covers the transportation and the transformation of the context information flow from sensors to actuators. But it does not specify how the context of an application is described and how its model is used to enable the discovery of the abstraction which enables the construction of the information flow. This solution seems to be adapted to an

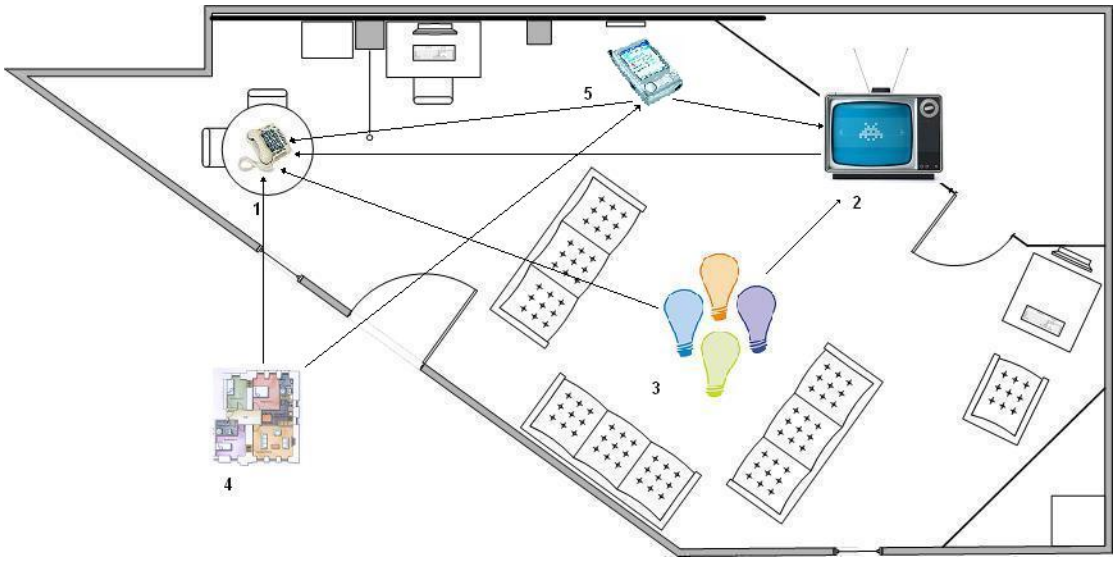

Figure 17 When the smartphone (5) is entering in the room, it is discovered by the other devices and applications. It is a context information provider for the phone (1) and the television set (2), and it asks context information to the location service (4).

environment that is already known, whereas our infrastructure is more optimized for open and unknown environments where sensors and actuators can interact together describing their needs and capabilities.

Closer to what is presented here is the work on contextors [10]. It offers a library of elements, called contextors, which are software abstractions modeling a relation between variables of a composition of situations as it is observed by the system. Contextors provide context information: They enable combining contextual information on a distributed mode. The contextors approach is very powerful to construct context-aware applications. It enables developers to build a chain of contextors going from numeric observables to situation and context identification. But it does not tell how to dynamically add devices without stopping the system or other devices. As the Context Toolkit, this approach lacks openness and dynamism to be used in an unknown and unpredictable environment.

Another toolkit approach is the PACE middleware [29] which offers context and preference management and a programming toolkit for interacting with them. This is a ready-to-use infrastructure to build context-aware applications, but it cannot dynamically link new context models to the old ones so that the old applications can take advantage of new sensors. Adding a new context-aware or context-sensitive device to an already running context-aware environment using this infrastructure mandates the use of the ontology already in this environment to describe context information. Several independently developed ontologies cannot be used for this purpose.

These three toolkits impose their own middleware to implement context-dependent applications. This is usually justified by the the lack of support for advanced features, e.g., mobility, in standard solutions. We have tried here to follow another road and to adopt standards as far as they are available.

There are several proposals to extend semantic web languages to contextualize the assertions [3, 26, 31]. The applications that are described here may use one of these proposals. The use of C-OWL [3] for representing contextual information may seem very similar to our proposal. However, C-OWL provides support for statically dealing with multiple context in OWL, while the present proposal concentrates on the dynamic aspects.

With regard to the use of OWL to represent context information, [42] introduces a high level ontology of contextual information for pervasive computing. The interest of this approach is the same as what is presented in §3.3: being able to extend ontologies. However, authors do not propose any path to extend it dynamically. A similar approach is proposed with CoBra [6] which can accept incoming devices thanks

to its context broker middleware, but provides no way to dynamically extend this ontology to cope with incoming unknown devices. [8] proposes three fundamental ontologies using a where, a when and a who ontologies to describe a shared context between applications and to ensure interoperability between them. Applications can define their own context model by extending the three ontologies or not, but the interoperability between each context model relies on the use of the core ontologies. GUMO [27] also provides ontologies targeting at ambient computing. [1] develops a contextual ontology in OWL where the context is composed of user centers of interest, making possible for the system to adapt the presentation of web pages. However, in this system, all interactions are centralized on a server and applications never interact with a dynamic or unknown environment. The existence of so many ontologies for pervasive computing emphasizes the need for reconciling ontologies.

A last and original use of ontology in ambient intelligence is made by the Gaia project agents and ontology in [37]. The infrastructure is a multi-agent system where agents can be context provider, context synthetizer or context consumer. An ontology server stores ontologies which are used to establish a joint terminology between context model agents. Even if agents can add new concepts in these ontologies, there is no provision for ensuring the interoperability with these new concepts.

8 Discussion

The work presented in this paper puts emphasis on particular properties of context management support for pervasive applications:

Openness is the first requirement, also considered as *Support for heterogeneity* in [29]. Thanks to semantic web technologies, our infrastructure supports introduction of unknown and unforeseen devices, services and applications at run-time. Using standard technologies encourage applications to adopt them, and these technologies have been built as open. Moreover, standards help developers by providing the opportunity to take advantage of tools that have already been developed for them, e.g., reasoners. [4] also required *standards* and *extensibility* which are covered here.

Dynamics support or flexibility is the ability to deal with heterogeneity (in devices, applications and context representation) on the fly. This is again provided by semantic web technologies which can always be extended by new representations and by the use of ontology alignment technology for reconciling representations.

Minimal commitment which, in some respect, corresponds to the *Ease of deployment and configuration* of [29], is ensured through the use of standard technologies, the definition of minimal interaction protocols and not requiring computationally expensive operations. This lowers the adoption barrier as much as possible while still ensuring the functionality of the proposal. These technologies are already recommended and used and do not represent a requirement out of proportion for industry parties. This also contributes filling the *standard* requirement of [4].

Requirements for context representations introduced in [4] call for a *structured*, *interchangeable*, *uniform*, *extensible* and *standardized* representation. These are satisfied by the use of semantic web technologies like RDF or OWL. The only additional property is *decomposability* that has not been specifically studied here.

There are other properties, introduced in [29] or [9], that have not particularly been addressed in this work. Relying on other standard technologies would help to meet these properties. They are:

Mobility support [29] No special support to mobility has been provided. The discovery protocol should work if devices, applications and services are reachable through a network.

Scalability [29] This proposal has not been sufficiently tested to demonstrate its scalability. But, as it uses simple and lightweight technologies, it should support the deployment of pervasive computing applications in a domestic environment. The main threat to scalability is the availability of alignment servers.

Support for privacy [29, 9] and *Security* [9]. In creating context managers, developers choose the context information that they want to exhibit on the network and to transmit to other devices and applications. This enforces privacy. No specific method have been provided to ensure more elaborate security features.

Traceability and control [29] or *History* [9]. Information history can be very useful in pervasive computing environments, in particular to infer the future from the past, and to adapt behaviors to user's habits or preferences. No method is provided in this work to manage context information history. The information flows between context information consumers and context information producers are hidden from users at run-time, but the multi-agent implementation using the JADE system offers a sniffer service for debugging (see Figure 14).

Tolerance for component failures [29] The openness and distributed character of the system should help it being tolerant. When a context information producer fails, the context information consumers which use it can find a substitute in the environment by reprocessing the protocol. The only centralized components are directories and alignment servers. Redundancy in these components (which seems good practice in such environments) should be the way to overcome failures.

9 Conclusion

We have proposed context representation and management support for pervasive computing applications. Our objective is to consider specifically the dynamic aspects of context management so that the intrinsically changing world of pervasive computing (people enter rooms, go to other places) is taken into account in any applications. Dynamics also require openness, i.e., the ability to extend the environment with new components (new devices with unknown characteristics are made available). Moreover, we insisted on lowering adoption barriers by adopting technologies which require minimal commitments.

This problem has been tackled by providing a framework to help developers to construct context-sensitive devices and context-aware applications. This framework is based on:

a Distributed component architecture which enables the addition, at any moment, of new devices able to provide information about the context of applications;

Context representation using semantic web technologies (RDF and OWL) which ensure the interoperability between independently developed components by taking advantage of both the open character of these technologies and their capacity to incorporate new descriptions;

a Minimal protocol for acquiring context information through other standard semantic web technologies (SPARQL);

Ontology matching which helps coping with semantically heterogeneous descriptions.

The main strength of the proposal reside in the tight composition of these elements.

We have shown how these technologies interplay in order to solve dynamics and heterogeneity problems. This proposal has been implemented based on available technologies. Field experiments are planned to show that the approach can scale to many applications and devices.

There are a number of limitations of the approach. Like most of ontology-based approaches, this one is certainly more flexible but slower. In particular, a full-fledged system as the one presented above may require efficient reasoning, matching and query answering which can be very complex operations. However, the complexity is not a reason for not using such techniques. Instead research should consider adapted solutions to these problems in this particular context. For instance, [2] investigated particular compilations of the ontologies for faster query answering (that is called matching in this context). Using approximate reasoning methods is another approach that can be used in the context of pervasive computing: providing a good enough support for users is the goal, not providing optimal support.

Another research track worth considering is what can be called "active context" in which context strengthening is not statically performed by components but dynamically achieved for answering a

particular query of an application, i.e., the query specifies the granularity required by the answers and services are able to provide the adequate support, not a more precise support.

Acknowledgment

Fano Ramparany and Jérôme Pierson are partially supported by the European project Amigo (IST-2004-004182); Jérôme Euzenat was partially supported by the European network of excellence Knowledge Web (IST-2004-507482) and NeOn integrated project (IST-2005-027595).

References

[1] F. Abel and J. Brase. Using semantic web technologies for context-aware information providing to mobile devices. Technical report, University of Hannover, Hannover (DE), 2004.

[2] S. Ben Mokhtar, A. Kaul, N. Georgantas, and V. Issarny. Efficient semantic service discovery in pervasive computing environments. In *Proc. 7th Middleware conference*, volume 4290 of *Lecture notes in computer science*, pages 240–259, Melbourne (AU), 2006.

[3] P. Bouquet, F. Giunchiglia, F. V. Harmelen, L. Serafini, and H. Stuckenschmidt. C-OWL: Contextualizing ontologies. In *Proc. 2nd International Semantic Web Conference (ISWC)*, volume 3298 of *Lecture Notes in Computer Science*, pages 164–179, Sanibel Island (FL US), 2003.

[4] S. Buchholz, T. Hamann, and G. Hübsch. Comprehensive structured context profiles (CSCP): Design and experiences. In *Proc. 2nd IEEE Conference on Pervasive Computing and Communications Workshops (PERCOMW)*, pages 43–47, Washington (DC US), 2004.

[5] M. Chalmers. A historical view of context. *Computer supported cooperative work*, 13(3):223–247, 2004.

[6] H. Chen, T. Finin, and A. Joshi. An ontology for context-aware pervasive computing environments. *Knowledge engineering review*, 18(3):197–207, 2004.

[7] H. Chen, F. Perich, T. Finin, and A. Joshi. SOUPA: Standard ontology for ubiquitous and pervasive applications. In *Proc. 1st International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pages 258–267, Boston (MA US), 2004.

[8] A. Clear, S. Knox, J. Ye, L. Coyle, S. Dobson, and P. Nixon. Integrating multiple contexts and ontologies in a pervasive computing framework. In *Proc. ECAI workshop on Contexts and ontologies*, pages 20–24, Riva del Garda (IT), 2006.

[9] J. Coutaz, J. Crowley, S. Dobson, and D. Garlan. Context is key. *Communications of the ACM*, 48(3):49–53, 2005.

[10] J. Coutaz and G. Rey. Foundations for a theory of contextors. In C. Kolski and J. Vanderdonckt, editors, *Proc. 4th International Conference on Computer-aided design of user interface (CADUI)*, pages 13–34, Valenciennes (FR), 2002.

[11] J. de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28(2):127–162, 1986.

[12] M. Dean and G. Schreiber (eds.). OWL web ontology language reference. Recommendation, W3C, February 2004.

[13] A. Dey. Understanding and using context. *Personal Ubiquitous Computing*, 5(1):4–7, 2001.

[14] A. Dey, D. Salber, and G. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16:97–166, 2001.

[15] P. Dourish. Seeking a foundation for context-aware computing. *Human-Computer Interaction*, 16:2–3, 2001.

[16] J. Euzenat. An API for ontology alignment. In *Proc. 3rd International Semantic Web Conference (ISWC)*, volume 3298 of *Lecture Notes in Computer Science*, pages 698–712, Hiroshima (JP), 2004.

[17] J. Euzenat. Alignment infrastructure for ontology mediation and other applications. In M. Hepp, A. Polleres, F. van Harmelen, and M. Genesereth, editors, *Proc. 1st ICSOC international workshop on Mediation in semantic web services, Amsterdam (NL)*, pages 81–95, 2005.

[18] J. Euzenat and P. Shvaiko. *Ontology matching*. Springer, Heidelberg (DE), 2007.

[19] T. Flury, G. Privat, and F. Ramparany. OWL-based location ontology for context-aware services. In *Proc. Artificial Intelligence in Mobile Systems*, pages 52–58, Nottingham (UK), 2004.

[20] F. Fu, C. Jones, and A. Abdelmoty. Building a geographical ontology for intelligent spatial search on the web. In *Proc. IASTED International Conference on Databases and Applications (DBA)*, pages 167–192, Innsbruck (AT), 2005.

[21] F. Gandon and N. Sadeh. Semantic web technologies to reconcile privacy and context awareness. *Journal of Web Semantics*, 1(3):241–260, 2004.

[22] F. Giunchiglia and P. Bouquet. Introduction to contextual reasoning: an articial intelligence perspective. In B. Kokinov, editor, *Perspectives on cognitive science*, pages 138–159. New Bulgarian University Press, 1997.

[23] T. Gu, H. Pung, and D. Zhang. A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications*, 28(1):1–18, 2005.

[24] T. Gu, X. H. Wang, H. K. Pung, and D. Q. Zhang. An ontology-based context model in intelligent environments. In *Proc. Communication Networks and Distributed Systems Modeling and Simulation Conference*, pages 270–275, San Diego (CA US), 2004.

[25] R. Guha. *Contexts : A Formalization and Some Applications*. PhD thesis, Stanford university, Stanford (CA US), 1995.

[26] R. Guha, R. McCool, and R. Fikes. Contexts for the semantic web. In *Proc. 3rd International Semantic Web Conference (ISWC)*, volume 3298 of *Lecture Notes in Computer Science*, pages 32–46, Hiroshima (JP), 2004.

[27] D. Heckmann, E. Schwarzkopf, J. Mori, D. Dengler, and A. Krner. The user model and context ontology gumo revisited for future web 2.0 extensions. In *Proc. 3rd Contexts and ontologies workshop*, pages 37–46, Roskilde (DK), 2007.

[28] K. Henricksen and J. Indulska. Modelling and using imperfect context information. In *Proc. 2nd IEEE Conference on Pervasive Computing and Communications Workshops (PERCOMW)*, pages 33–37, Washington (DC US), 2004.

[29] K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam. Middleware for distributed context-aware systems. In *Proc. International Symposium on Distributed Objects and Applications (DOA)*, volume 3760 of *Lecture Notes in Computer Science*, pages 846–863, Orlando (FL US), 2005.

[30] K. Henricksen, J. Indulska, and A. Rakotonirainy. Modeling context information in pervasive computing systems. In *Proc. 1st International Conference on Pervasive Computing*, pages 167–180, London (UK), 2002.

[31] O. Khriyenko and V. Terziyan. Context description framework for the semantic web. In *Proceedings Context 2005 Context representation and reasoning workshop, Paris (FR)*, 2005.

[32] G. Klyne and J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. Recommendation, W3C, 2004.

[33] J. McCarthy. Notes on formalizing contexts. In *Proc. 13th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 555–560, Chambéry (FR), 1993.

[34] A. Narayanan. Realms and states: a framework for location aware mobile computing. In *Proc. 1st international workshop on Mobile commerce*, pages 48–54, New York (NY US), 2001.

[35] K. A. P. Ngoc, Y.-K. Lee, and S. Lee. OWL-based user preference and behavior routineontology for ubiquitous system. In *OTM Conferences (2)*, volume 3761 of *Lecture Notes in Computer Science*, pages 1615–1622, Agia Napa (CY), 2005.

[36] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. Working draft, W3C, 2006.

[37] A. Ranganathan and R. Campbell. A middleware for context-aware agents in ubiquitous computing environments. In *Proc. 4th Middleware conference*, volume 2672 of *Lecture notes in computer science*, pages 143–161, Rio de Janeiro (BR), 2003.

[38] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, pages 85–90, Santa Cruz (CA US), 1994.

[39] L. Serafini and P. Bouquet. Comparing formal theories of context in AI. *Artificial intelligence*, 155(1-2):41–67, 2004.

[40] T. Strang, C. Linnhoff-Popien, and K. Frank. CoOL: A Context Ontology Language to enable Contextual Interoperability. In *Proc. 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS)*, volume 2893 of *Lecture Notes in Computer Science*, pages 236–247, Paris (FR), 2003.

[41] K. Traub, G. Allgair, H. Barthel, L. Burstein, J. Garrett, B. Hogan, B. Rodrigues, S. Sarma, J. Schmidt, C. Schramek, R. Stewart, and K. Suen. The EPCglobal architecture framework. Specification, EPCglobal, 2005.

[42] X. H. Wang, D. Q. Zhang, T. Gu, and H. K. Pung. Ontology based context modeling and reasoning using OWL. In *Proc. 2nd IEEE Conference on Pervasive Computing and Communications Workshops (PERCOMW)*, pages 18–47, Washington (DC US), 2004.