Chapter 5

Context-Awareness

Bruno Gonçalves, Paulo Costa and Luis Miguel Botelho

5.1 Introduction

Context-aware computing has increasingly gained the attention of the research community because, as it is the case with human interactions, context information provides the background against which it is possible to more accurately interpret communicative acts without the need to explicitly state everything that might be relevant. If, within an agent negotiation for buying some specific service, the service provider says "*the price is 20 Euros*", the receiver would not be capable of fully interpreting the meaning of the message without using the context created by the whole conversation. Context information provides the basis for more efficient information processing mechanisms due to the possibility of discarding irrelevant information in early stages of information processing. For instance, if some patient's personal assistance agent is looking for a service that would sell him or her a specific medicine and deliver it in the patient's home, this would be achieved through the creation of a compound service consisting of an on-line pharmacy and a medicine transportation service. Using context information about the patient's location, the service composition process may discard service providers located far away from the client and create the compound service considering only a very small number of all existing services of the relevant categories. Context information also enables better adapted behavior since, being context-aware, it may be more directed towards clients requirements in the circumstances of the interaction. For instance, if a personal assistance agent is looking for an internet movie critique service for its owner, having to choose between services displaying a German, an Italian, or anEnglish user interface, the use of context information regarding the user's profile, will enable the agent to choose the service whose interface language is preferred to the client.

Context-aware computing increasing importance is manifest in the emergence of a growing number of applications that use context information captured by software and hardware sensors, such as the current time, the current temperature and

humidity, the user's location, current traffic in alternative internet connections, availability and load of some service provider.

The CASCOM Project designed and implemented an architecture for context-aware agent-based service coordination for static and mobile users. Context-aware service coordination agents may adapt their behavior to their clients taking into account the context in which interactions take place. For the sake of efficiency, modularity and specialization, service coordination agents should not have to care about the problems of acquiring context information from the large diversity of sources actually existing or coping with the enormous variety of representation and encoding formats used in these sources.

This chapter provides an overview of selected topics of context-aware computing, focusing the problems of context information acquisition, modeling, and management, which are those related with context acquisition and management systems. Context information acquisition refers to the process of acquiring information that is considered to be part of the context. Often, context acquisition is implemented through software sensors (e.g., user spoken languages) or hardware sensors (e.g., room temperature). Context modeling consists of creating the model according to which context is represented. Context modeling allows to convert raw data read from the sensors into something with meaning, generally following a given ontology. For instance, the string "*English*" provided by a software sensor implemented in the user's personal assistance agent might mean "*user spoken languages* = {*English*}". Context management refers to the whole activity of context processing within the context system including storing context information, taking care of context clients and their requests, and knowing when to discard particular pieces of context information. The chapter will review context definitions, theories and principles for context system design, context modeling, and context system architectures. There is of course much more about context, for instance, about the way context information may be used by context-aware systems. However, the chapter will not address such topics in detail.

Maybe the first idea of context information was the user location however, simultaneously with the effort to clarify and adequately extend its definition [2, 12, 13, 29], other kinds of information were used in context dependent applications, such as the state of network connections, the existing devices available to the user, and the social environment.

Several definitions of context can still be found, which does not contribute to creating a clear picture of context-aware applications and context acquisition and management systems. In spite of this diversity, maybe, the most accepted definition of context is the one proposed by Dey and Abowd [13], according to which "*context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves*". Although the proposal by Dey and Abowd is still the most accepted, the definition by Anagnostopoulos et al. is increasingly gaining more adepts [13]. They use the definition of Dey and Abowd but they circumscribe the notion of context to a

set of situations, which describe humans, applications, and environment related to specific activities.

A context-aware system is a set of services that adapt to environmental factors, such as the location in which the system is used, nearby people and objects, as well as the changes that occur in these objects over time. With the appearance of mobile devices, context became increasingly important to improve the performance and effectiveness of applications for mobile users [17]. There are several projects [1, 5, 6, 3, 4, 16, 21, 33] that investigate how context information can be useful to improve existing services and to create new services for the next generations of mobile networks.

One of these projects is the WWI Ambient Networks [1]. This project is aimed at creating solutions beyond the third generation, promoting a scalable and low cost network that allows an easy access to the offered services. These solutions include the use of the context-aware computing paradigm to select the best connection, location services and geographical orientation among others.

Other project presented by Chalmers and Sloman [5] proposes the use of a framework that allows the management of the quality of service in mobile networks, using context information to analyze the user characteristics.

Several architectures and approaches that deal with context [2, 8, 9, 6, 10, 11, 22, 24, 27, 28] have been discussed over time, however there is still no normalized solution that satisfies all possible uses of context information.

This chapter presents some definitions of context given by several authors. Following, it presents several context models focusing context acquisition, context modeling and context processing. Next, it describes some of the developed architectures of context-aware systems. Finally, overall comments about this subject are presented.

5.2 Context Definitions

Context definitions, in computer applications, have been adapted from the way context is used in everyday language. Since there are many everyday language uses for context, an adequate and generally accepted definition of context information and context-aware applications still does not exist. The meaning of context in everyday language is related with the interpretation of written and spoken text. Text is not an encapsulated representation of a specific meaning. Rather, it is an indication that allows the anticipated construal of a meaning. That construal is based on what comes with the text, namely its context. In a sentence, each word has a meaning but the sentence global meaning can only be determined by doing inferences over its context [31]. For instance, if someone looking from a window at a car being stolen, says "*isn't that our neighbor's car?*" the pronoun "*that*" can only be understood if the listener is also looking at the same scene, that is, if the listener shares at least part of the same context with the speaker. "*Our neighbor*" can also be understood by the listener depending on the context. If the

listener leaves in the same place as the speaker, then the expression gains a certain meaning. However the meaning would be different if the listener knows the speaker is talking about him and his wife.

Linguists and philosophers have made a big effort to identify the several context elements that give meaning to words. When trying to adjust everyday context definitions to computer sciences, several authors have created their own definitions of context for their applications, which lead to different views of context and different approaches to acquire context information from the environment.

Winograd [31] defines context as not only the data structures in the operating system (such as Windows and applications), but also something far beyond the application being used. Context is an operational term; something is considered context if it is used in an interaction.

For Schilit and Theimer [29], context consists of the identities of people, the objects near the application, as well as their changes. Dey [12] adds to the definition of context the emotional states, the user attention, location and orientation, date and time, and objects and persons in the user environment. The meaning of the noun phrase "*the car that has just appeared in front of you*" depends on the time in which the phrase is uttered, it depends on the direction the listener is headed to and on his or her location, and of course, on the objects (i.e., the car) on the listener's environment.

For Anagnostopoulos et al. [2], context is a set of situations that describe people, applications and environment related with a specific activity. This provides context to the context, which will enable to constrain the whole array of objects, people and events that may be considered context to only those related to a given activity. For instance, only the set of potential threats related to driving in a particular road in a given moment, not the set of all possible threats in the universe, is relevant to interpret the danger traffic sign.

The most accepted definition of context, for the scientific community, is the one by Dey and Abowd [13] which states that context is defined as any information that characterizes a situation or entity.

According to Schmidt et al. [30], context can be divided in two categories: human factors and physical environment. Human factors include user, social environment (people near the user, the relations among them, between them and the user, and between them and the application) and task (which plays a similar role to that played by the activity put forth by Anagnostopoulos and colleagues). The physical environment includes location, infrastructure (supporting the application, the user, the social context and the task) and conditions (e.g., current date and time).

Analyzing current definitions, we conclude that they are either too restrictive or too wide scoped, failing to distinguish context-aware computing from other kinds of computing.

Taken together, the points of view of Schilit and Theimer [29], and of Anagnostopoulos et al. [2] mean context includes applications, environment, and people related with a given activity, and their changes. Dey's proposal [12] also includes

the emotional states, the user attention, location and orientation, date and time, and the user environment. In a strict sense, these definitions would rule out for instance current traffic conditions in a given network connection, the average waiting time per request and the current number of requests of a given application. In a broad sense, this definition would include almost everything.

For Dey and Abowd [13], context is defined as any information that characterizes a situation or entity. For Winograd [31], something is considered context if it is used in an interaction. These are obviously too broad definitions. Winograd's definition would include even the messages exchanged in the interaction. And for Dey and Abowd, almost any information would be considered context. This way, context-aware computing would be basically information processing which is not a useful definition since it does not allow us distinguishing context-aware computing from other kinds of computing.

The proposal of Schmidt et al. [30] identifies different classes of context information but it also cannot distinguish context-aware computing from other kinds of computing.

We propose that often the decision of considering or not a specific information as context should be a design task. For instance, some applications would consider the user location to be part of the context, while for others, location would not be relevant. In any case, context information should be processed differently from other classes of information or else it would not make sense to be concerned about context-aware computing. A suggestion regarding the way context information should be handled could be "*in an interaction between the initiator and the participant, it is the responsibility of the participant to acquire relevant context information even if the participant has to ask the initiator to provide (part of) it*".

5.3 General Design Principles and Context Modeling Approaches

The design principles reviewed in this chapter are important to evaluate specific context system architectures presented in the next section. Ideally, specific architectures should comply with reviewed design principles. Whatever information is considered context in an application, it must be acquired, modeled and processed, which will transform context into something useful [2, 23]. According to Anagnostopoulos and his colleagues [2], a context system should implement a set of functionalities, such as acquisition, aggregation (creating new meaningful compound data structures integrating context information from different types of sources), discovery (discovering the relevant sources of context), and context search (discovering the relevant context information), among others.

The acquisition stage is normally associated with sensors. A great amount of context information is acquired from sensors implemented in software or hardware. Several approaches have been proposed that focus on the task of creating an

interpretation of the acquired context information that makes sense for the specific application. This interpretation process is usually guided by a context ontology conceptually close to the application. An example of context acquisition might be the reading "*001A*" from a given temperature sensor placed inside a pool. The result of context interpretation, in this case, could be "*pool water temperature in Tom's place = 26 C*".

From the reviewed approaches we have identified several important aspects to be considered when developing context systems. First, we have to separate context information acquisition from context information interpretation. This separation allows context interpretation to be independent of sensor interface details. Context information acquisition can be done by software or hardware sensors. Context interpretation normally requires tools and ontologies defined in or used by the context system.

Context acquisition is not limited to only capturing context information in the moment in which it is required. Context acquisition also includes the storage of acquired context information as well as its changes over time. The variation of context information over time is usually called historic context information [19].

During acquisition, we should take into account the errors and delays introduced by processing this information. A way to avoid these errors is to use data fusion [16] (i.e., using information from several sensors to try to identify and correct possible errors). For instance, if we have time readings from several clocks, errors pertaining the reading of one of the clocks may be overcome if we use the readings of the other clocks.

Some of the acquired context information is static, while other kinds of context information may change over time. Examples of static context information are the time schedule of a given service or the nationality of a given user. Examples of dynamic context information are the user location, current time and date, and current temperature. According to Henricksen et al. [19], context information is considered static when it does not vary much over time. Static context may be directly acquired from the user or a service and stored in a central repository. Dynamic context information should be acquired by sensors and locally stored in the sensors themselves.

The proposal of Cortese et al. [8] shows the complexity of managing a large number of sensors. The proposed model assumes that the whole interaction with the user is made through sensors. This implies that the used context model has to be extensible so it can be applied to different situations with more, less, or with different sensors. These authors define two methods to get information from sensors - the methods push and pull. Using these methods, the sensors can be both proactive, always sending information to the system, or passive, sending the information only when a request is received.

Context interpretation should draw upon the definition of context ontologies. Context ontologies allow representing context information following a structure and a level of abstraction independent from context sensors and other used sources of information. Any entity that receives context information represented according

to some specified context ontology can understand it, if it knows the ontology [15]. Context ontologies may be organized according to several aspects, such as used devices, application, and location, among others. The way context is acquired (from software and hardware sensors) also represents a context aspect [18].

The proposal of Anagnostopoulos [2] and his colleagues concerning context modeling states that context should be represented by classes with associations. These associations connect context elements and deal with both dynamic and static context. Additionally, the context model should allow the definition of dependencies between context elements. Christopoulou et al. [7] present a similar type of association, the synapse. These associations represent preferences and needs of the associated elements. According to this proposal, the context model should be defined by an ontology with two levels. The first level defines the model used to describe the context ontology. Following this very proposal, the first level would include the definition of "*context element*", and "*synapse*". The second level describes the context ontology using the model defined in the first level. Sticking to the same example, the second level would be the particular context elements and the particular synapses in a given application domain. The context information models presented by Anagnostopoulos and his colleagues and by Christopoulou et al. are very comprehensive models. Both synapses and dependencies are important aspects to focus when identifying the context elements.

A context acquisition and management system should be presented to applications as an abstract (i.e., hardware independent) context capturing and storage component ensuring the independence of the application with respect to the used context acquisition sensors [20]. The context system core can be built of components that implement its functionalities. Each sensor can also be built as a component that implements an abstract interface. This allows using the advantages of the component-based systemsparadigm such as modularity and the unification of sensor access in a single nterface [14]. As an example of a sensor implementing an abstract interface, we could think of a temperature sensor that extracts the reading "*001E*" from the environment but converts this into "*environment temperature = 30 C*" before making this information available to its clients.

The storage of context information in a system can be implemented by a centralized repository modeled following a given ontology. This repository allows the centralized access to context by context information producers and consumers [31]. The context system can also be presented as a peripheralware in a service network [26]. Generally, peripheralware consists of additional software layers placed between the middleware and services, and between the middleware and the client. Those layers perform tasks transparent to the services and to the clients. Using peripheralware allows context-awareness in services that are not prepared to deal with context. All the context information processing is done by the peripheralware in a transparent way to the services and the clients.

Prekop and Burnett [25] define a context model centered on the user activity, which is significant only when the activity takes place. This vision differs significantly from those previously mentioned because, in the previous ones, context

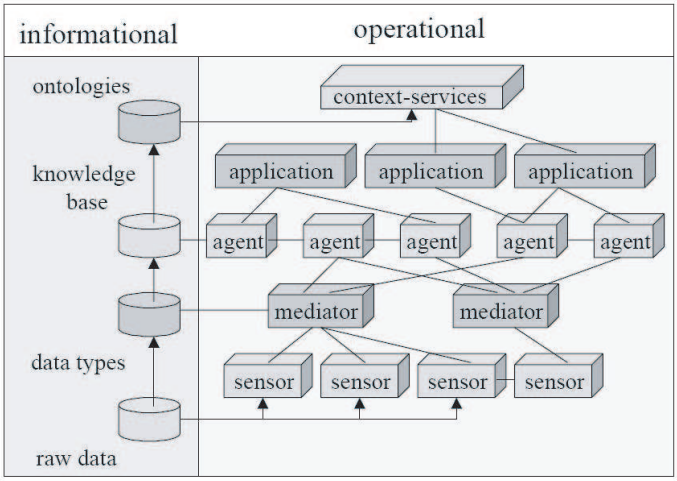

Figure 5.1: Context System Architecture Levels

information relates to entities, while in this one, context information relates to activities in which the entities participate. For instance, in previous mentioned models, the price of a book is context information relating to the book, and the available money to buy it is context information related to the client; in this model, both are context information of the book buying activity. This model assumes the context definition of Anagnostopoulos et al. [2], according to which, context is a set of situations that describe people, applications and environment related with a specific activity.

Several architectures were developed from the models described in this section.

5.4 Context Dependency Architectures

Context systems architectures may have two levels: operational and informative levels, as presented in Figure 5.1. The operational level comprises the system modules such as sensors, mediators that convert sensor data into higher level information, intelligent agents that gather the system knowledge, and context-aware applications (if a global perspective is adopted that views both the context system and its clients as unique system). The informative level comprises the acquired context information and knowledge. This knowledge can be represented in a simple data model, in an object-oriented model, or in an ontology model [2]. Context information is acquired by sensor networks and further subject to processes that convert it into higher level representations, usually following a context ontology, which might be more abstract or more specific of the application that requires

it. Sensors can be used simply as data acquisition mechanisms but they can also be more sophisticated. Often sensors are coupled with adaptation mechanisms that create context information representations independent of the specific type of sensor. This is called sensor adaptation. Pure sensor architectures only have the operational level, since the way context information should be presented is not defined in sensor networks. However, if context adaptation performed by the sensors is done according to a given ontology, the informative level will also be present.

This section starts with sensor network architectures such as the Smart-Its Architecture. These simple architectures are totally distributed context systems consisting of a network of sensors that exchange context information packets among them. Each sensor may create new context information packets or add information to received packets. When completed, packets are sent to the context clients that have requested them.

The Merino architecture represents a sophistication of pure sensor networks because it has three kinds of sensors of different sophistication; and it includes a centralized context repository, and a user model.

All of the other reviewed architectures use similar ontology-based context modeling techniques for providing a sensor-independent abstract view of context to their clients. Besides providing sensor abstraction, all other architectures have a central repository for context information. Besides the instantaneous context, often the context repository stores historic context information. In addition to these common features, each of these reviewed architectures introduce specific differences with respect to the others.

WASP, CoBrA, Context Taylor architectures as well as the one proposed by Cortese and colleagues separate sensor information capturing from its processing. In all of them, the lower level layer extracts context information from sensors. Then, a higher level layer adapts the acquired information according to a defined ontology. This abstract representation of the context information is then subject to diverse kinds of information processing such as context fusion and inference, which result in additional pieces of context. Acquired and generated context information is stored in a repository. WASP and CoBrA have a system manager that has knowledge about all elements belonging to the architecture, manages context information requests, acquires information from the repositories and the context interpreters, and delivers it to context clients. CoBrA manager, denominated Context Broker, is a distributed agent that communicates with client agents, using an agent communication language. Besides context fusion and inference, the Context Broker also supports privacy by imposing access policies defined by each client, using a declarative language. Context Taylor has learning mechanisms that extract patterns from the context information. These patterns may be used in future context information requests.

Often, context acquisition and management architectures support both context information requests and context information subscription (push and pull). Information request mechanisms allow context clients to acquire context infor-

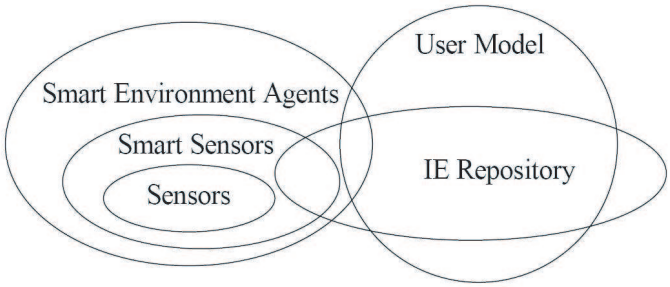

Figure 5.2: Merino Architecture

mation when needed (on demand); information subscription mechanisms allow context clients to receive desired context information whenever it changes.

5.4.1 Smart-Its Architecture

The decentralized architecture proposed by Michachelles and Samulowitz [24] is ideal for mobile environments and ad hoc networks. It stores the context information acquired by sensors (*Smart-its*) in packets that are passed from sensor to sensor. These packets are denominated *sCAP* (*Smart Context-Aware Packets*). This architecture does not have a central control mechanism. Instead, sensors get to know the information acquired by their neighbors through the context packets they receive from them. A sensor only adds the context information to a packet it receives if this information has some similarity with the context contained in the packet. Each packet is organized in three parts: the acquisition plan, the probable context, and the acquisition path. The acquisition plan is a plan based on an initial model that is adapted each time the packet visits a sensor. The probable context is the information retrieved from the sensors. The acquisition path represents the list of sensors already visited. After visiting all the sensors specified in the acquisition plan, the packets are directly sent to the user or system that has requested them. The architecture proposed by Samulowitz et al. [28] also uses packets, in a similarly way as the Smart-its architecture.

5.4.2 Merino Architecture

The Merino architecture presented by Kummerfeld et al. [22] integrates three classes of sensors: normal sensors, intelligent sensors, and environment agents. The architecture also has a context information repository and a user model (see Figure 5.2). Sensors in higher layers produce higher level information, promoting a more complex vision of context. Sensors in lower level layers are confined to acquiring information from the environment. The repository stores the context information acquired from the sensors. Agents retrieve context information from

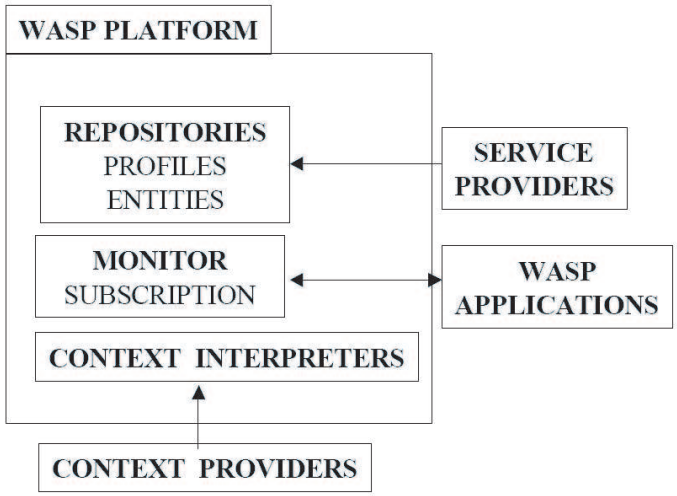

Figure 5.3: WASP Architecture

the repository and produce new context information. The user model, which is controlled by an intelligent personal assistant, represents the needs of the user.

5.4.3 Architecture proposed by Cortese et al.

The architecture proposed by Cortese et al. [8] defines a logical model of architecture with two layers. This division separates sensor information capturing from its processing. In the lower layer, denominated sensors layer, the sensor information is extracted. In the upper layer, denominated semantic layer, the acquired information is adapted according to a defined ontology. The information is published in a repository where fusion agents generate additional information with a higher abstraction level.

5.4.4 WASP Architecture

The WASP architecture (*Web Architectures for Service Platforms*) [9] defines a general development environment that supports the execution of mobile services with context dependency (see Figure 5.3). The fundamental idea of this architecture is to hide the complexity introduced by context acquisition and processing from the context clients. This is done using interpretation modules that offer context to applications. These modules gather context information and make it available for the remaining platform. The platform includes repositories to support the monitoring component, which has knowledge about all elements belonging to the system. This monitoring component is responsible for the integration of WASP

applications, for managing context information subscription requests, and for acquiring information from the repositories and the context interpreters. Context information is subscribed by the services registered in the platform, being further processed in the context interpreter. Ontologies are used to model context, enabling the architecture components to share knowledge among them. In order to obtain more complex context, different context supplying entities must share the same context representation. The presented architecture enables applications to obtain context information in a transparent way. Context processing problems are solved within the architecture. However, context information acquisition must be handled by the services that provide that information. The idea of hiding the context information processing complexity is an important feature of a context system.

5.4.5 CoBrA Architecture

The CoBrA architecture (*Context Broker Architecture*) [6] is an agent-based architecture that supports context awareness in intelligent systems, such as the systems that make up an intelligent house, or an intelligent vehicle (see Figure 5.4). This architecture has a central element - the context broker - that supplies a general picture of the context to the remaining agents. The context broker also supports privacy by imposing access policies defined by each client agent. The architecture incorporates the operational level in its design. The informative level is represented by the context information model. The CoBrA architecture requires the definition of a collection of ontologies to model the context. The CoBrA architecture provides a declarative language of policies that users and devices may use to limit the access to protected information. CoBrA architecture uses OWL [32] as ontology language. The context broker is an agent created to manage the shared context model. It is associated to the smart space in which the system operates, for example an intelligent house. This agent aggregates several other agents that represent smaller parts of the space.

Using this decentralized approach, communication overhead problems related with the access to a centralized mediator can be avoided. The context broker can also infer context information that cannot be easily acquired by sensors, which can be used to complete missing context elements. The context agent main function is the acquisition of context information from several sources, the fusion of this information in a coherent model and the subsequent sharing of this model with other entities in the environment. This architecture is ideal to agent networks. The use of an agent as a context broker enables CoBrA to communicate with other agent architectures, using an agent communication language. The distributed context broker results in a highly robust system, since the failure of one of the mediators does not compromise the functioning of the remaining system parts.

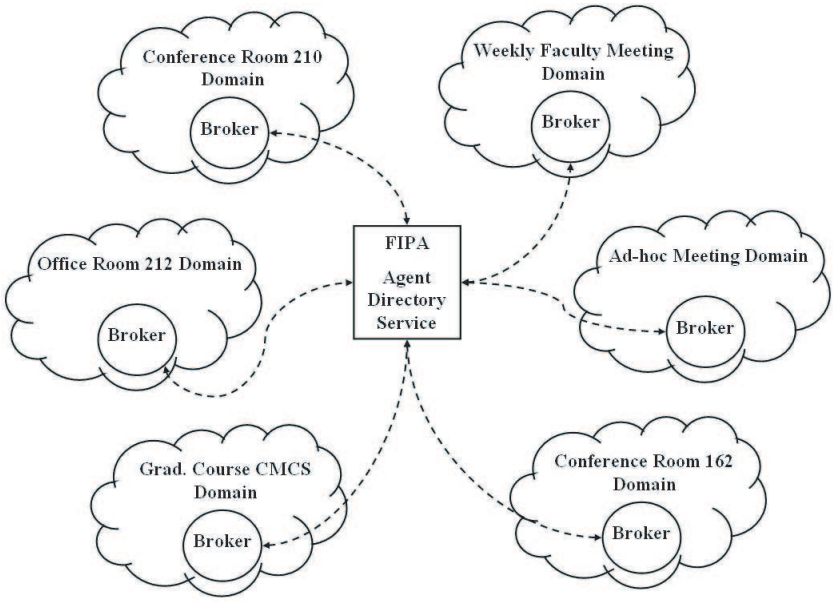

Figure 5.4: CoBrA Architecture

5.4.6 Context Taylor

This architecture proposed by Davis et al. [10, 11] is a component-based architecture that has a context service that acquires data from a set of context generation sources. The acquired context information is stored in a repository and made available to applications via an API. Learning mechanisms extract patterns from the context information. These patterns may be used in future context information requests. The components in the architecture include generation sources, a context history repository, a learning engine, a context patterns repository, a context patterns activator, and a server that coordinates the interaction between these components. The context service works as a middleware repository that provides context about specified entities. This service manages the connection with each source of context, providing context information to applications. The structure of this architecture is presented in Figure 5.5. The server registers the context requests sent by context clients and stores all the provided context information in the context repository service. Each context entry is composed of four fields: temporal mark, user id, context type, and context state. The temporal mark allows selecting context information pertaining to a specified time interval. The user identification allows to store and access context information for different users. Each type of context corresponds to a specific representation format. The context state contains information about context of a certain type, which was observed in a certain mo-

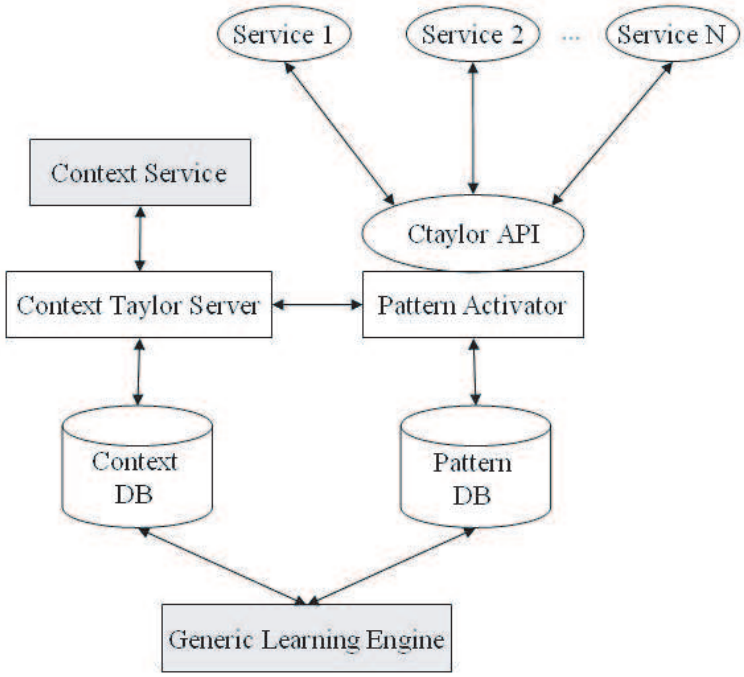

Figure 5.5: Context Taylor Architecture

ment. The learning mechanism applies learning algorithms to context information in the repository to abstract context patterns. These patterns are then stored in the patterns repository. Each pattern is composed of a condition, a pre-condition, a likelihood level (a value between 0 and 1 that represents the probability that the precondition predicts the condition) and a support. The conditions and pre-conditions define sets of events, and each event represents an instance of context attributes.

5.5 Summary

From the presented set of definitions, models and architectures, we conclude that a definitive solution to deal with context still does not exist. None of the described proposals addresses the whole context subject, only presenting solutions to some of the several problems related with context.

Some context definitions are too restricted ruling out important aspects of context. However most of them are too general failing to provide criteria for distinguishing context information from other kinds of information. We propose that a suitable definition of context, in the scope of context-aware computing, must allow

domain and application dependent context information to be identified at design time (since particular information would righteously be considered and treated as context information in some applications but not in others); and most importantly, it should provide a clear basis for distinguishing context information from other kinds of information in terms of the way context information, but not other kinds of information, is processed. That is, definitions must have something to say about the way context information is processed in context-aware applications.

Work of more theoretical nature especially focused on context modeling and on general principles regarding context acquisition and processing proposes that context acquisition should be clearly separated from context interpretation. This work also proposes that static context information should be directly acquired from the user or other applications and may be stored in centralized repositories; while dynamic context information should be acquired by sensors and should be locally stored.

According to some authors, context representation, as specified by context ontologies, should contain several dimensions, the most important of which are entities, context elements, activities, and several kinds of associations between these (e.g., dependencies and needs). Besides individual samples of context information, it is also useful to keep historic context information.

Domain independence, improved interoperability, and the possibility to dynamically extend the context model (context ontology) are desirable properties of the context representation framework. These goals can be achieved if context ontologies have two levels: the first level describing the model that is used to represent the context ontology; and the second level representing the context ontology using the representation model defined in the first level.

Context acquisition and management systems play an important role in context-aware computation because they provide an abstraction of the context acquisition and management processes, hiding low level domain and hardware dependent details from context users and client applications. These systems should also support the two main modes of information conveying - push and pull - allowing context clients to passively receive context information whenever it changes or to receive it only upon request.

Several context acquisition and management system architectures have been proposed. These may be organized in two groups: the sensor network systems, which are more focused on the context acquisition problem; and the complete architectures, addressing both the context acquisition stage and the context processing stage, which should be totally separate processes.

Each of the proposed architectures addresses specific aspects of context acquisition and processing. For instance, sensor network architectures, such as the Smart-its, are focused on context acquisition and representation. The Merino architecture main innovation is the organization of sensors according to their level of sophistication / intelligence. It also proposes to use a context repository. Other proposals such as the WASP architecture emphasize the interaction with other applications instead of the context acquisition process.

The described complete architectures focus on important aspects that should be taken into account when designing a context system (e.g., independence of context processing from context acquisition, fusion and inference over context information, learning, and context delivery). The CoBrA architecture is more adequate for agent networks, since it provides an agent-based interface with applications, through context broker agents. The access to context information, by applications, in the other architectures is ensured by APIs. Ideally, these APIs should be flexible enough to allow adding several types of information and sensors, and to support flexible types of context searching requests. Unfortunately this is not the case.

The described architectures propose different solutions to deal with specific aspects of context-aware computing. However, none of them addresses the whole array of relevant problems. A more complete context acquisition and management system should be based on the integration of ideas put forth by the described proposals. Most of the presented architectures store all context information in central repositories which might not be a good idea, especially when there are many different sources of context acquiring a huge amount of information, and many client applications competing for system resources. A new proposal should give more attention to the integration of the sensors layer, allowing the existence and management of several types of sensors, with the context processing layer. The acquired context information should be stored in a distributed fashion. Static context information may be stored in centralized repositories; while dynamic context information should be stored locally in the sensors.

None of the architectures can be dynamically extended with new sensors of new classes of context information, in run-time. None of them supports the dynamic addition of new ontology definitions in run-time either. This is also an important feature of the context acquisition and management system developed in the CASCOM architecture.

Finally, each of the described architectures provides only one type of interface (e.g., agent-based, or API). Since context systems should be independent of their client applications, it would be a good idea to implement at least the most common types of interface.

References

[1] Ambient Networks Consortium. Ambient Networks. http://www.ambient-networks.org, 2006.

[2] C. Anagnostopoulos, A. Tsounis and S. Hadjiefthymiades: Context Awareness in Mobile Computing Environments: A Survey. Mobile e-conference, Information Society Technologies, 2004.

[3] L. Capra, W. Emmerich and C. Mascolo: Reflective Middleware Solutions for Context-Aware Applications. Proceedings of the Third international Con-

ference on Metalevel Architectures and Separation of Crosscutting Concerns LNCS, Vol. 2192. Springer-Verlag, London, 126-133. 2001.

[4] L. Capra, W. Emmerich and C. Mascolo: CARISMA: Context-Aware Reflective mIddleware System for Mobile Applications. IEEE Transactions on Software Engineering, vol. 29, no. 10, pp. 929-945, Oct., 2003.

[5] D. Chalmers and M. Sloman: QoS and Context Awareness for Mobile Computing. Proceedings of the 1st international Symposium on Handheld and Ubiquitous Computing, LNCS Vol. 1707. Springer-Verlag, London, 380-382. 1999.

[6] H. Chen, T. Finin and A. Joshi: An Intelligent Broker for Context-Aware Systems. Adjunct Proceedings of Ubicomp 2003, Seattle, Washington, USA, October 12-15, 2003.

[7] E. Christopoulou, C. Goumopoulos, I. Zaharakis and A. Kameas: An Ontology-based Conceptual Model for Composing Context-Aware Applications. In Research Academic Computer Technology Institute, 2004.

[8] G. Cortese M. Lunghi and F. Davide: Context-Awareness for Physical Service Environments. Ambient Intelligence, IOS press, 2004.

[9] P. D. Costa, J. G. P. Filho and M. van Sinderen: Architectural Requirements for Building Context-Aware Services Platforms. IFIP workshop on Next Generation Networks, Balatonfured, Hungary, 8-10 September, 2003.

[10] J. S. Davis, D. M. Sow, M. Blount and M. R. Ebling: Context tailor: Towards a programming model for context-aware computing. Proceedings of the first International Workshop on Middleware for Pervasive and Ad Hoc Computing (MPAC)., pages 68-75, Rio De Janeiro, Brazil, 16-20 June, 2003.

[11] J. S. Davis, D. M. Sow and M. R. Ebling: Context-sensitive Invocation Using the Context Tailor Infrastructure. System Support for Ubiquitous 94 Computing Workshop at the Fifth Annual Conference on Ubiquitous Computing, October 2003.

[12] A. K. Dey: Context-Aware Computing: The CyberDesk Project. AAAI 1998 Spring Symposium on Intelligent Environments, Technical Report SS-98-02, pp 51-54, 1998.

[13] A. K. Dey and G. D. Abowd: Towards a better understanding of context and context awareness. In GVU Technical Report GIT-GVU-99-22, College of Computing, Georgia Institute of Technology, 1999.

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

[15] J. G. P. Filho and M. van Sinderen: Web Service architectures, semantics and context-awareness issues in Web Services platforms. WASP/D3.3, 16-26, 2003.

[16] H. W. Gellersen, A. Schmidt and M. Beigl: Multi-sensor context-awareness in mobile devices and smart artifacts. Mobile Networks Applications 7, 5, 341-351, October, 2002.

[17] R. Gold and C. Mascolo: Use of Context-Awareness in Mobile Peer-to-Peer Networks. Proceedings of the 8th IEEE Workshop on Future Trends of Distributed Computing Systems. IEEE Computer Society, Washington, DC, 142, 2002.

[18] K. Goslar, S. Burchholz, A. Schill and H. Vogler: A Multidimensional approach to Context-Awareness. In Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI2003), 2003.

[19] K. Henricksen, J. Indulska and A. Rakotonirainy: Modeling Context Information in Pervasive Computing Systems. In Pervasive '02: Proceedings of the First International Conference on Pervasive Computing, pp. 167-180, 2002.

[20] J. I. Hong and J. A. Landay: An Infrastructure Approach to Context-Aware Computing. Human-Computer Interaction, 16:287-303, 2001.

[21] P. Korpipää and J. Mäntyjärvi: An Ontology for Mobile Device Sensor-Based Context Awareness. Fourth International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT 2003): 451-458. Stanford, California (USA), June 23-25, 2003.

[22] B. Kummerfeld, A. Quigley, C. Johnson and R. Hexel: Merino:Towards an intelligent environment architecture for multigranularity context description. User Modeling for Ubiquitous Computing, 2003.

[23] H. Laamanen and H. Helin: Contex-Awareness, Overview and State-of-Art. CASCOM project Technical Report, TeliaSonera, 2004.

[24] F. Michahelles and M. Samulowitz: Smart CAPs for Smart Its Context Detection for Mobile Users. Personal Ubiquitous Computing 6, 4, 269-275. January, 2002.

[25] P. Preko and M. Burnett: Activities, context and ubiquitous computing. Elsevier Science PII: S0140-3664(02)00251-7, 2002.

[26] M. Ritchie: Pre and Post Processing for Service Based Context-Awareness. Technical Report Equator-02-023, University of Glasgow / Department of Computing Science, 2002.

[27] H. K. Rubinsztejn, M. Endler, V. Sacramento, K. Gonalves and F. Nascimento: Support for Contex-Aware Collaboration. Mobility Aware Technologies and Applications, LNCS 3284, pp. 37-47, 2004.

[28] M. Samulowitz, F. Michahelles and C. Linnhoff-Popien: Adaptive interaction for enabling pervasive services. Proceedings of the 2nd ACM international Workshop on Data Engineering For Wireless and Mobile Access (Santa Barbara, California, United States). S. Banerjee, Ed. MobiDe '01. ACM Press, New York, NY, 20-26. 2001.

[29] B. Schilit and M. Theimer: Disseminating Active Map Information to Mobile Hosts. IEEE Network, 8(5):22-32, 1994.

[30] A. Schmidt, M. Beigl and H. W. Gellersen: There is more to Context than Location. Proceedings of the International Workshop on Interactive Applications of Mobile Computing (IMC98), Rostock, Germany, November 1998.

[31] T. Winograd: Arquitectures for Context. HI Journal, 2001.

[32] World Wide Web Consortium. OWL-S 1.0 Release. http://www.daml.org/services/owl-s/1.0, 2005.

[33] S. S. Yau and F. Karim: Reconfigurable Context-Sensitive Middleware for ADS Applications in Mobile Ad Hoc Network Environments. In Proceedings of the Fifth international Symposium on Autonomous Decentralized Systems. ISADS. IEEE Computer Society, Washington, DC, 319. March, 2001.