POLITECNICO DI MILANO

Facoltà di Ingegneria

Corso di Laurea Specialistica in Ingegneria Informatica Dipartimento di Elettronica e Informazione

A Study of Context Aware Systems

Supervisor: **Prof. Letizia Tanca**

Master's Thesis of
Shaon, Md. Rajib Hasan
Matricula: 723236

Anno Accademico 2010 - 2011

A Study of Context Aware Systems

This Tesina is submitted to the Dipartimento di Elettronica e Informazione (DEI) Politecnico di Milano, Italy in partial fulfillment of the requirements for the degree of Master of Science in Computing Systems Engineering.

Md. Rajib Hasan Shaon

(Matricola # 723236)

Dipartimento di Elettronica e Informazione (DEI)
Politecnico di Milano, Italy
December 2010

Acknowledgements

First of all, I would like to express my gratefulness to the almighty God for His blessing that has lead to a flourishing end of this Tesina. It is He who has been always on my side when I felt feeble and troubled.

Secondly, I have to thank my respected supervisor **Prof. Letizia Tanca** for leading me to the area of multimedia research. She has brought me to the area of scientific research and offered me the opportunity to know how to be a good researcher. She used to encourage me always and I found her mind ever fresh and cooperative. Whenever I got the wrong direction, she prescribed me the right path with her deep knowledge and wisdom.

Thirdly, I should remember my parents and siblings who always conferred me kind support, love and encouragement that helped me to continue my study. I also would like to express my deepest gratitude to my friends for their lovely support throughout my study.

Finally, thanks to Politecnico di Milano, Italy for quality education without which it would have been very difficult for me to get quality education. In addition, I am grateful to all the staffs of the Dipartimento di Elettronica e Informazione (DEI) for their sincere co-operation and guidance.

Contents

Abstract

Context-aware computing refers to a general class of mobile systems that can sense their physical environment, and adapt their behavior accordingly. Such systems are a component of a ubiquitous computing or pervasive computing environment. Three important aspects of context are: (1) where you are; (2) who you are with; and (3) what resources are nearby. Although location is a primary capability, location-aware does not necessarily capture things of interest that are mobile or changing. Context-aware in contrast is used more generally to include nearby people, devices, lighting, noise level, network availability, and even the social situation; e.g., whether you are with your family or a friend from school. [1]

Now with the help advances in technology, Context-aware systems are increasing in popularity. Context aware system (CAS) is a large scale system which senses information and changes of surrounding environment, and adapts its response correspondingly. Recently, CAS is widely investigated and developed both in academy and industry. Therefore context modeling is becoming a relevant issue and an expanding research field.

For this reason, it is especially important to achieve a complete and safe context aware system. This paper provides a summary and survey of a chosen set of context aware systems, and categories their properties and use according to taxonomy. An overview of each system is provided.

Abstract (Italian)

I sistemi di “context aware computing” si riferiscono ad una classe generale di sistemi mobili in grado di rilevare il loro ambiente fisico e adattare il loro comportamento di conseguenza. Tali sistemi sono componenti di un ambiente di “ubiquitous computing” o “pervasive computing”.

Tre aspetti importanti del contesto sono i seguenti: (1) Luogo (2) Persone (3) Risorse nelle vicinanze. Sebbene il luogo sia un elemento primario, i sistemi “location aware” non necessariamente rilevano le cose mobili o in fase di cambiamento. I sistemi “context aware”, invece, sono utilizzati più in generale per integrare le persone vicine, i dispositivi, l’illuminazione, il grado di rumorosità, a disponibilità di rete, e anche la situazione sociale: ad esempio, se siete con la vostra famiglia o con un amico di scuola.

Ora, con il progresso tecnologico, i sistemi “context aware” si stanno diffondendo. Il sistema di “context aware (CAS) è un sistema su larga scala che rileva le informazioni e i cambiamenti dell’ambiente circostante, e si adatta di conseguenza. Recentemente il CAS è stato ampiamente studiato e sviluppato sia nel campo della ricerca che nell’industria. Pertanto le tecniche di modellazione del contesto stanno diventando sempre di più oggetto di ricerca. Per questo è particolarmente importante realizzare un completo e sicuro sistema di conoscenza del contesto CAS.

Il presente documento fornisce una sintesi e uno studio di un gruppo scelto di sistemi di conoscenza e definisce le loro proprietà secondo la tassonomia. Viene fornita inoltre una panoramica di ogni sistema.

Chapter 1

Introduction

People are quite successful at conveying ideas to each other and reacting appropriately. This is due to many factors: the richness of the language they share, the common understanding of how the world works, and an implicit understanding of everyday situations. When people talk with another people, they are able to use implicit situational information, or *context*, to increase the conversational and width. Unfortunately, this ability to convey ideas does not transfer well to people interacting with computers. In traditional interactive computing, users have an impoverished mechanism for providing input to computers. Consequently, computers are not currently enabled to take full advantage of the context of the human-computer dialogue. By improving the computer's access to context, we increase the richness of communication in human-computer interaction and make it possible to produce more useful computational services.

In order to use context effectively, we must understand what context is and how it can be used, and we must have architectural support. An understanding of context will enable application designers to choose what context to use in their applications. An understanding of how context can be used will help application designers determine what context-aware behaviors to support in their applications.

Finally, architectural support will enable designers to build their applications more easily. This architectural support has two parts: services and abstractions.

1.1 What is Context

To develop a specific definition that can be used prescriptively in the context aware computing field, we will look at how researchers have attempted to define context in their own work. While most people tacitly understand what context is, they find it hard to elucidate. Previous definitions of context are done by enumeration of examples or by choosing synonyms for context.

1.1.1 Previous Definitions of Context

In the work that first introduces the term ‘context-aware,’ Schilit and Theimer refer to context as location, identities of nearby people and objects, and changes to those objects. These types of definitions that define context by example are difficult to apply. When we want to determine whether a type of information not listed in the definition is context or not, it is not clear how we can use the definition to solve the dilemma. Other definitions have simply provided synonyms for context; for example, referring to context as the environment or situation. As with the definitions by example, definitions that simply use synonyms for context are extremely difficult to apply in practice. The definitions by Schilit *et al.* and Pascoe are closest in spirit to the operational definition we desire. Schilit *et al.* claim that the important aspects of context are: where you are, who you are with, and what resources are nearby. Pascoe defines context to be the subset of physical and conceptual states

of interest to a particular entity. These definitions are too specific. Context is all about the whole situation relevant to an application and its set of users. We cannot enumerate which aspects of all situations are important, as this will change from situation to situation. For this reason, we could not use these definitions provided.[2]

1.1.2 Definition of Context

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 applications themselves.

This definition makes it easier for an application developer to enumerate the context for a given application scenario. If a piece of information can be used to characterize the situation of a participant in an interaction, then that information is context. Take the canonical context-aware application, an indoor mobile tour guide, as an example. The obvious entities in this example are the user, the application and the tour sites. We will look at two pieces of information – weather and the presence of other people – and use the definition to determine whether either one is context. The weather does not affect the application because it is being used indoors. Therefore, it is not context. The presence of other people, however, can be used to characterize the user's situation. If a user is traveling with other people, then the sites they visit may be of particular interest to her. Therefore, the presence of other people is context because it can be used to characterize the user's situation. [2]

1.1.3 Context-Aware Computing

Context-aware computing refers to a general class of mobile systems that can sense their physical environment, and adapt their behavior accordingly. Such systems are a component of a

ubiquitous computing or pervasive computing environment. Three important aspects of context are: (1) where you are; (2) who you are with; and (3) what resources are nearby. Although location is a primary capability, location-aware does not necessarily capture things of interest that are mobile or changing. Context-aware in contrast is used more generally to include nearby people, devices, lighting, noise level, network availability, and even the social situation; e.g., whether you are with your family or a friend from school. [1]

1.7 Overview of the Tesina

The rest of the thesis goes as follows.

Chapter 2 presents the literature review of all chosen context systems.

Chapter 3 presents some background knowledge and analysis of the contest systems.

Finally, in chapter 4 I conclude the tesina with summary

Chapter 2

Literature review

In this chapter, I present the summary of selected context systems which I have studied;

2.1 Context Systems

I have studied following 6 context systems

1. Context-ADDICT
2. The MUSIC context System
3. The CoWSAMI Context System
4. Incontext
5. CONTEXT TOOLKIT
6. Gaia

2.1.1 Context-ADDICT

Goal of Context-ADDICT is to discover and wrap data sources whose contents are accessible and relevant with respect to the application, and make their data available on the users' mobile devices, appropriately tailored on the user's current context and information needs. In this section we briefly describe the Context-ADDICT [3] architecture and context model .

The overall system is composed by three main subsystems, each one devoted to a specific task:

- The Design-Time subsystem supports the designer in the context-modeling activity and in modeling the information domain. The latter can be represented as an ontology or, alternatively, by any data model: if the data model is not an ontology, the support of a domain ontology will still be needed, since it will become a precious tool for dynamic integration support.

- The Run-Time Schema Level subsystem, composed of several modules, performs data source discovery and wrapping, schemata integration and tailoring.

- The Run-Time Data Level subsystem, once the schemata have been integrated and tailored, is devoted to the actual data movement, to synchronizing and integrating the data instances only for the pieces of information considered relevant, and to query processing (both local and remote).

The Design-Time subsystem must, first of all, support the designer in the modification of an existing Domain Ontology or in its creation from scratch. The Domain Ontology captures the main concepts of the information domain. It provides a well accepted general taxonomy enriched of the main relations among

concepts, even if not all the concepts' attributes and relations are detailed. If the data model chosen to represent the information domain is not ontology, also the information schema, represented in the chosen data model, must be presented – either already existing or produced from scratch–. Along with it goes a specification of the relationships of its concepts with those of the Domain Ontology. The Domain Ontology supports the system at run-time, in the integration of the global information schema with the data source schemata. The Design-Time subsystem also supports the designer in the context design activity. The Context Dimension Tree is used to represent the knowledge of"What may the activities and interests of the various users be, within this application domain?". It is created by the designer via a syntax oriented Dimension Tree Editor. Following a precise methodology [BQ06], the tool guides the designer in the process of defining the context dimensions and their values. Once the Context Dimension Tree has been created, the designer has to associate each context (or chunk configuration), with the schema portion representing the data which are relevant with respect to that context, thus knowledge about "What part of the domain is relevant for a given user in a certain context?" This process is heavily application dependent, and cannot be performed automatically in any sense; the Context Integrator is a highly interactive tool, meant to give the designer an important role: setting the relationship between the context and the information domain schema means to be able to capture, for each context, the specification of the actual data to be delivered in that given context, discarding unnecessary information. This association is done in a semi-automatic way: the designer specifies the schema portions declared relevant to each context element, represented in the tree by a white node – e.g., in the archaeological example, a supervisor will be assigned a portion of data, different (at least partially) from the data related to the visitors' role –. The system completes the specification by appropriately combining, in each context, the data relevant to its elements. The result is the definition of a view over the domain information schema for each possible context. Run-Time Schema Level subsystem: ambition of Context-ADDICT is being able to capture data sources which might be fully heterogeneous in terms of schemata, data format and access interfaces. At run time, a Data Source Discovery Service will be in charge of actually discovering data sources and making them reachable. This module may heavily vary depending

on the specific scenario we are considering, from a centralized server to a set of fully distributed discovery procedures from a mere syntactic matcher to a semantical filter. The data sources may range from Relational Databases to XML data sources, to Web Services, to sensor networks, but the key point is that their schemata will be transformed into a common format and then operated upon in a uniform global representation. Some of these data sources will be cooperative, i.e., they will be aware of their participation to Context-ADDICT; in this case they will provide a description of the available data in the common format. Other, occasional data sources may offer heterogeneous interfaces such as the DDL specification of the database schema or a set of web pages: in the latter scenario, Context-ADDICT provides a set of wrappers and wrapper generators to automatically translate the schema into the common model. The domain ontology supports this phase, by mediating between the semantics of data sources' and domain concepts. In general, data sources may appear and disappear during system working time; however, considering a snapshot of the system life at a given time, a set of data sources schemata, a global information schema and a Domain Ontology are available; thus integration is needed. This operation is performed at run-time either on the user's device or on a dedicated machine, depending on the deployment choices of the designer. The integration operation is a rather general and well known problem, though far from having been solved. A lot of research effort has been devoted to make this process as automatic and precise as possible. In Context-ADDICT, the Integration Module makes intensive use of an ontology matcher we have developed: X-SOM At the end of the integration, all the data sources' information is coherently integrated with the global schema. Since the correspondences drawn between the Context Dimension Tree and the global schema are available, at this point the data tailoring part comes into play: to reduce the amount of data to be managed on the user device, the view definitions produced in the design phase are mapped to appropriate queries over the data source's and used to tailor the information coming from them, possibly instantiated by means of actual values describing the current user context. For instance, in the archaeological example, if the user (device holder) is an operator during on-site work, the data should contain information about the art pieces s/he is in charge of on that day, or in that period. Considering a medical database example, if the device user is a

doctor during house call time, the data should be tailored to contain his/her patients' information related to that day's envisaged calls. Thus, from each data source, only the information which is relevant to that context will be physically integrated on the device. During the described process, several metadata have been recorded, together with the data schema, in order to enable query processing and synchronization, in charge to the Data Level subsystem. The Run-Time Data Level subsystem deals with the actual data transfer, thus, together with on-line query processing, it is responsible for data synchronization and local data management. Each data source schema will contain a description of how each concept of the data source has been stored. This can be done by means of metadata or be explicitly coded within the transformation rules used by the source wrapper. For example, the fact that the concept "teacher" of the data source 1 is stored in a table named "professor" of a database named "university" reachable at a given URL is captured by transformation rules between the relational model and the Context-ADDICT internal model. In the current implementation, where the global information schema is represented as an ontology, we have a translator which interprets the DDL of a relational database and produces an ontology, in its turn integrated with the Domain Ontology. Here, the concepts "teacher" and "professor" are recognized as synonyms, and a special mechanism generates virtual identifiers for tuples of the relational database, in order to see them as ontology concepts and to be able to translate ontological queries into relational queries. Because of the tailoring phase, data synchronization and local data management are performed on a manageable amount of data, actually relevant to the user. A Local Data Management service such as the one presented in, as well as mechanisms for data synchronization are required together with a set of caching policies. Apart from the caching and storage policies, the process data synchronization will be similar to distributed and heterogeneous query processing , with the addition of local data storage.

2.1.2 The MUSIC context System

This section describes the context model of the MUSIC project. This approach is based on several existing approaches and tries to overcome their limitations.[4]

Three layers of abstraction

This approach identifies three basic layers of abstraction that correspond to the three main phases of context management: the conceptual layer, the exchange layer and the functional layer. The conceptual layer aims to be leveraged by the developers and to be exploited in the model-

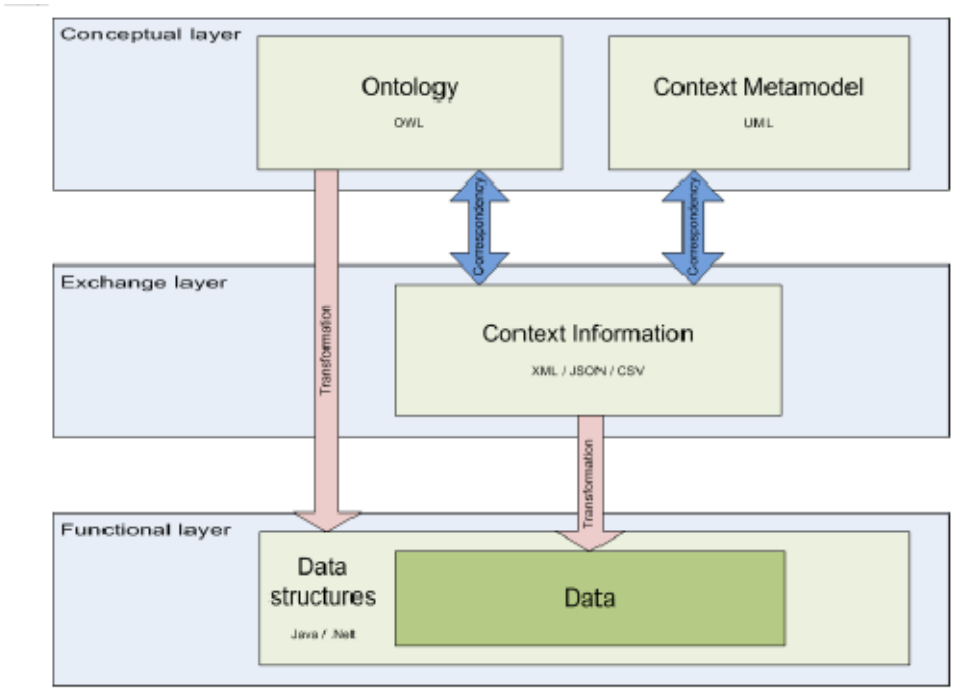

Figure: 1 the three layers of the MUSIC context model

driven development approach. The layer enables the definition of context artifacts such as elements, scopes, entities and representations based on standard specification languages, like UML and OWL. The exchange layer aims to be utilized for interoperability between devices. At this layer, the context information can be expressed in any of the XML JSON and simple CSV based representations. These can be used for communication of context data between nodes or between sensors and nodes. Finally, the functional layer refers to the actual implementation of the context model representation and the internal

mechanisms used in the different nodes. This model can be object-based, but it does not necessarily need to be interoperable as different devices might use different implementations of it, using for example Java and .NET. The main objective of this layer is efficiency, both in terms of processing speed and resource consumption... Figure [1] illustrates how these concepts fit into the three layers of the MUSIC context modeling approach. For this context model, author decided to incorporate the concept of ontologies in the conceptual layer of the context model for several reasons. (1) Ontologies facilitate the establishment of a common understanding of the semantics of context elements and their associated metadata and therefore boost interoperability. (2) Similar to the ASC model proposed by Strang *et al*, ontologies can also be used to define the internal structure of context data, thus allowing several representations, their interpretation and automatic conversions between them. (3) By incorporating ontologies, it is possible to model a wide range of relationships between context elements, which is essential for a flexible context reasoning approach. Also, a context metamodel is defined to facilitate automatic transformation between the different layers of the context modeling approach and to define basic guidelines for modeling the ontology.

The ontology is described in OWL and the context meta-model is specified in UML. Together they form the conceptual layer of the MUSIC context modeling approach. The context metamodel defines the general structure of context information and shows how concepts and/or individuals/entities specified in the ontology are referenced. In turn, as the context meta-model defines a general representation of context information it can also be considered as a kind of schema for defining the concrete representations of context elements in the ontology. At the exchange layer, an instance of the conceptual model is represented in XML (or alternatively in JSON or CSV). The representation in XML is quite straight forward, as it is the common way to represent individuals of the ontology (which can be seen as contextinformation). At the functional layer, a set of data structures for storing the context information are defined. As the internal structure of context elements is specified in the ontology, it is possible to

automatically generate the corresponding data structures for specific platforms along with appropriate serialization and de-serialization methods. Thus, the data structures can easily be filled with the information represented at the exchange layer without much overhead spent for interpretation. It is also worth noting here, that the information concerning the ontology is only transferred once or on demand. All these features take into account the quite limited resources of mobile devices in pervasive computing environments.

2.1.2.1 The MUSIC Context Meta-model.

Figure [2] illustrates the proposed Context Meta-model. As already mentioned before, it is just intended to define how context information is structured and how semantic concepts and/or entities specified in the ontology are referenced.

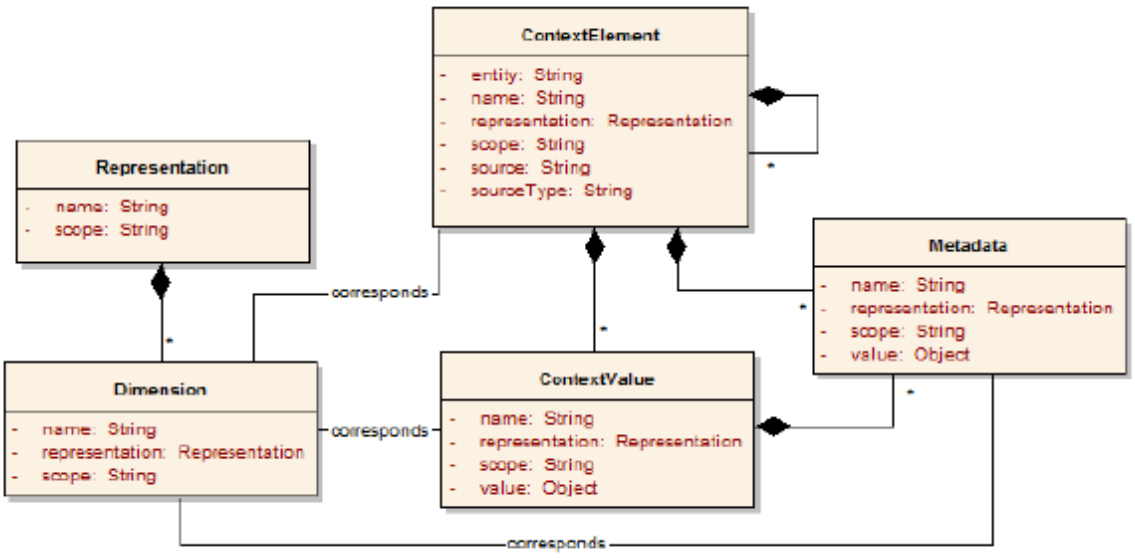

Figure 2: The MUSIC Context Meta-Model

Context information is encompassed in *context elements* which provide information about *context entities* and *context scopes* specified as semantic concepts in the ontology. *Context scopes* group context values belonging to the same context domain. For example, the *scope* 'Position' groups context values like: 'longitude', 'latitude' and 'accuracy'. The *context entities* refer to concrete entities in the world, for example a certain user, room, device, *etc*. *Context elements* can be composed of other context elements and can contain a number of *context values*. For example, a context element *network connection* in a *device's* context can contain the elements *Wi-Fi* and *Bluetooth*, and both elements can have the values *Cost* and *Bandwidth*. Metadata can be associated with context elements and context values. Here we distinguish between *predefined* (or suggested) *metadata* and *user-specific metadata*. For context elements, the proposed model includes the *predefined* metadata *name*, *entity*, *scope*, *representation*, *source* and *sourceType*, which are associated as attributes. The *name* serves as an identifier; *scope* provides a reference to the characterized context scope defined as semantic concept in the ontology; *entity* references the concrete individual of a certain entity type of the ontology to which the context information is associated, e.g. "My Windows XP Laptop" which is of entity type 'device'. The *representation* refers to the internal representation of the context information which is also specified in the ontology. With these types of metadata, it is specified that a context element characterizes the semantic concept *scope* for the individual *entity* and its internal structure corresponds to *representation*. The *source* is a unique identifier of the component that provides the context information (e.g. a context sensor or reasoner). Similar to Henricksen and Indulska *sourceType* can have values like *sensed*, *derived* or *user-provided*. Sensed means that raw-data are provided by the context sensor, derived indicates that reasoning or pre-processing was used to derive the data and user-provided means that the context information was given by the user. For *context values* the suggested types of metadata are *name*, *scope* and *representation* that have the same meaning as the corresponding metadata types of the context element. In addition, we allow to associate *user-specific metadata* to context elements and context values. In a way, these *metadata* can be seen as additional *context values* and they are also represented in the same way. However, in contrast to *context values*, *metadata* can be associated to *context elements* and *context values*.

It is noteworthy here, that context elements, context values and metadata characterize the same entity, but the context scopes they refer to may be different. This is quite obvious as e. g. a context value that belongs to a context element just provides part of the information of the whole context element. Therefore, in a way, the referenced scope of the context value can be considered as part or sub-scope of the context scope referenced by the context element.

Each *context element*, *context value* and *metadata* has a *representation*. According to aspects in the Aspect Scale Context (ASC) model described in Strang *et al*, each *representation* (in the ASC model called aspect) aggregates one or more *dimensions* (scales in ASC). Each *dimension* corresponds to a certain *context element*, *context value* or *metadata* element. A *dimension* itself has its own *representation*, which again can consist of several *dimensions*. With these concepts, the internal structure of the context information is defined through the context element.

2.1.3 The CoWSAMI Context System

In this view, an AmI environment consists of networked entities, each one of which includes an instance of the CoWSAMI[5] infrastructure (Figure [3]). The CoWSAMI entities may be connected through a LAN or WLAN. CoWSAMI also allows the formulation of pure ad-hoc environments where the entities communicate through technologies such as Bluetooth. Finally, the environment may comprise entities

located on the Internet. The entities vary from typical stationary workstations and PCs, to handheld and embedded devices such as PDAs, smart-phones and sensors. In the open environments that authors examine, every CoWSAMI entity may play the role of a context user, the role of a context source, or both. A context source provides one or more Web services that offer context information. The

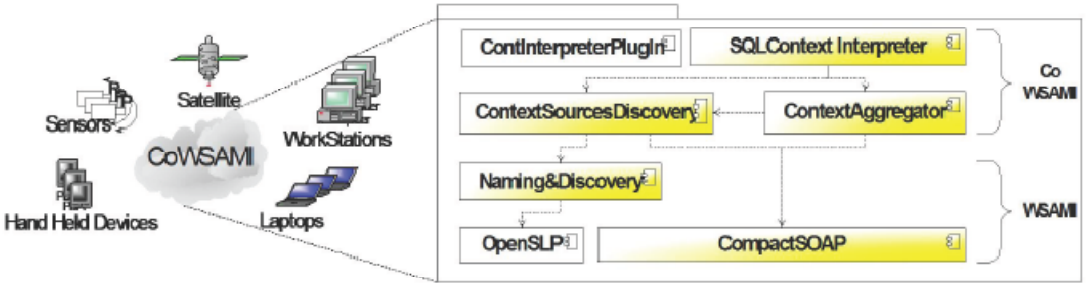

Figure 3: CoWSAMI Architecture.

Web services are specified using the WSAMI language, which extends standard WSDL with features enabling a more detailed behavioral description. A service specification consists of an abstract and a concrete part. The abstract part describes the interface of the service in terms of a WSDL document, whose URI (Uniform Resource Identifier) is included in the abstract part. Multiple service specifications may have the same abstract part, as long as they describe services providing the same interface. The concrete part of the service specification contains binding information (e.g., the endpoint address of the service, the communication protocol used, etc.). Figures [4](a) and (b) give examples of service specifications, provided by two different CyberCars. The upper parts of the figures give the abstract specifications of these entities, while the lower parts provide the WSDL interfaces referenced in these abstract specifications. CyberFrogs (Figure [4](a)) offer an homonymous Web service whose interface comprises 6 operations that can be invoked to obtain a unique identifier that characterizes each CyberFrog, the CyberFrog's brand, velocity, fuel, and coordinates. On the other hand, Cyber- Cabs

(Figure [](b)) provide a Web service that comprises a single operation, which can be invoked to obtain all the characteristics of a CyberCab (i.e., its identifier, brand, velocity and coordinates). In CoWSAMI, the users specify the context that interests them as a set of context attributes (Figure [4](a)). The information that corresponds to a context attribute is a value provided by a Web service. In general, the AmI environment may comprise multiple context sources that report semantically equivalent information (e.g., two different CyberCars reporting their current velocity). Moreover, a context source may provide values for more than one context attribute (e.g., the velocity and the brand). Therefore, related context attributes are organized into relations (Figure [4](a)). A context relation is characterized by a name and consists of a finite set of context attributes. The context information modeled by a relation is a finite set of tuples, i.e., a finite subset of the cartesian product of the domains of the attributes that constitute the relation. The relational-based approach we employ for modeling context is also followed by other context-aware middleware infrastructures. However, in most of these cases, context modeling is controlled in that

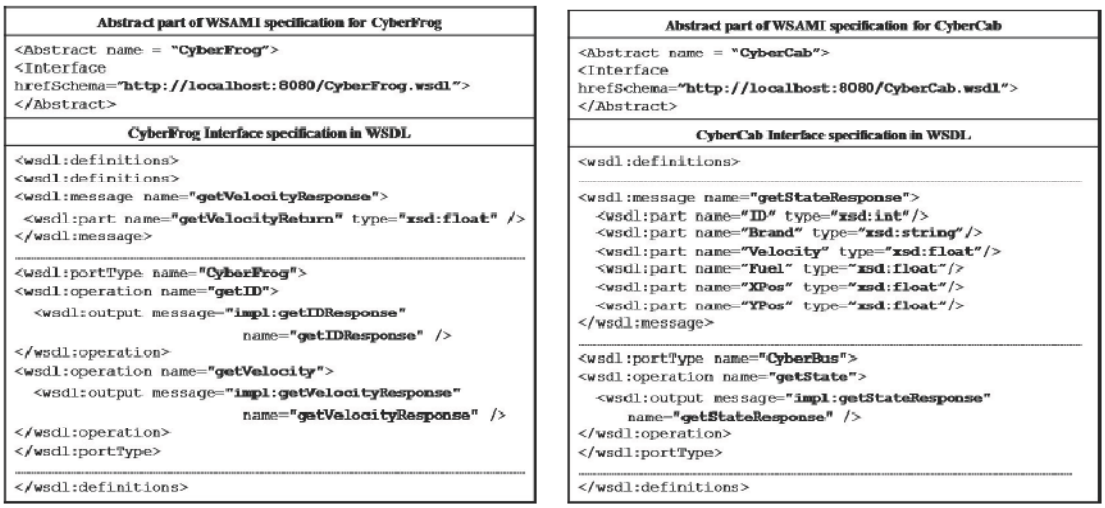

Figure 4: Different types of CyberCar interfaces.

there is a unified model defined for the environment. In CoWSAMI we aim at supporting open AmI scenarios in which the users have the ability to define their proper context relations. The relational-based modeling of context information allows formulating queries against relation instances, which are populated at runtime through the use of the CoWSAMI infrastructure. In detail, the CoWSAMI infrastructure is structured as depicted in Figure [3]. Its lowest layer comprises CSOAP (Compact SOAP), a lightweight communication mechanism, which allows deploying, invoking and executing Web services on resource-constrained devices. The same layer comprises a standard SLP (Service Location Protocol) server which serves for locating networked CoWSAMI entities in the environment. On top of the SLP server, the Naming&Discovery service realizes a primitiveWeb service discovery protocol. On top of the primitive Naming&Discovery service, a context sources discovery mechanism is incarnated by the ContextSourcesDiscovery service. The same layer further comprises a context aggregator mechanism, realized by the ContextAggregator service. Finally, the upper layer of CoWSAMI comprisesat least a simple context interpreter, realized by the SQLContextInterpreter service. This service provides user-friendly means for gathering context information through the use of its underlying aggregator and discovery services. Figure [](b) gives the interface of the ContextAggregator service which allows performing the following tasks: (1) Defining the particular context relations that interest a user; (2) define the context rules that customize the gathering of context information with respect to the interfaces of the context sources that become dynamically available and the context relations of interest; (3) compile the tuples that constitute the context relations of interest.

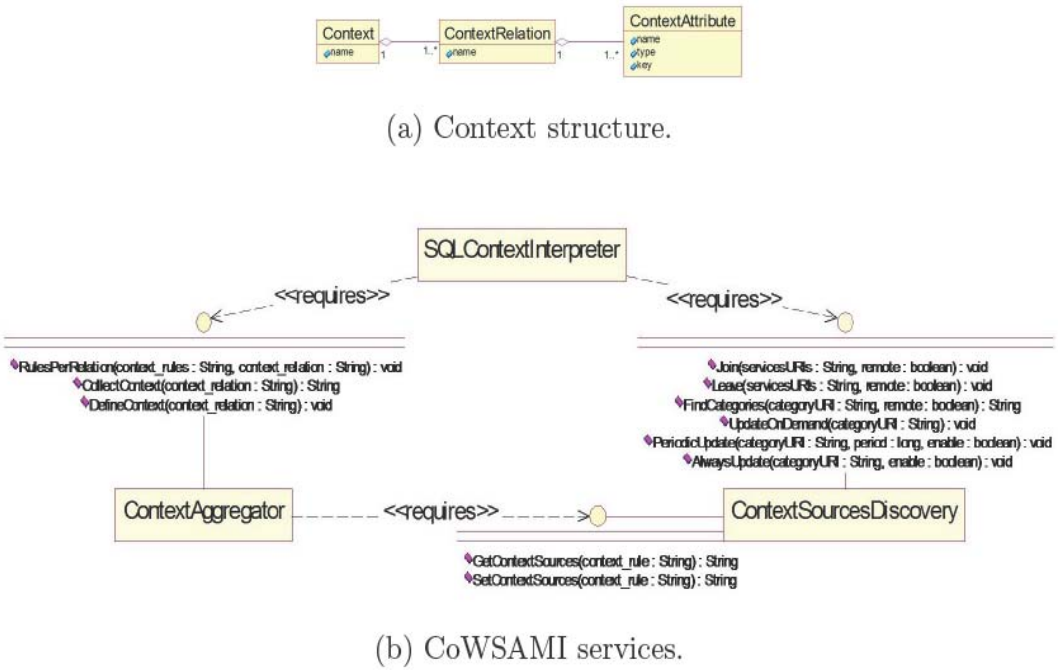

Figure 5: Context structure and CoWSAMI service interfaces.

Figure [5](b) details the interface of the ContextSourcesDiscovery service. Through this service a CoWSAMI entity that plays the role of a context source can perform the following tasks: (1) Join the AmI environment, i.e., make itself available as a context source to other CoWSAMI entities; (2) leave the AmI environment, i.e., resign from playing the role of a context source. Through the same service, an entity that gathers context information can perform the following tasks: (1) Discover the different Web service interfaces offered by context sources that become dynamically available; (2) populate a repository managed by the ContextSourcesDiscovery service with addressing information concerning context sources that become dynamically available. Finally, the ContextSourcesDiscovery service provides the ContextAggregator service

with means for acquiring addressing information concerning the available context sources that contribute in a particular context relation of interest. The repository of the ContextSourcesDiscovery service consists

of a number of categories (Figure [](a)). A category is characterized by the URI of the abstract service description that specifies the particular interface provided by the context sources, belonging in this category. The category contains URIs that identify service specifications, whose abstract parts reference the URI that characterizes the category. The concrete parts of these service specifications comprise specific addressing information for the available context sources that offer the specified services. The category is further characterized by the names of the context relations to which it contributes. Getting back to our reference example, Figure [5](b) gives a possible snapshot of the repository, managed by the ContextSourcesDiscovery service that is

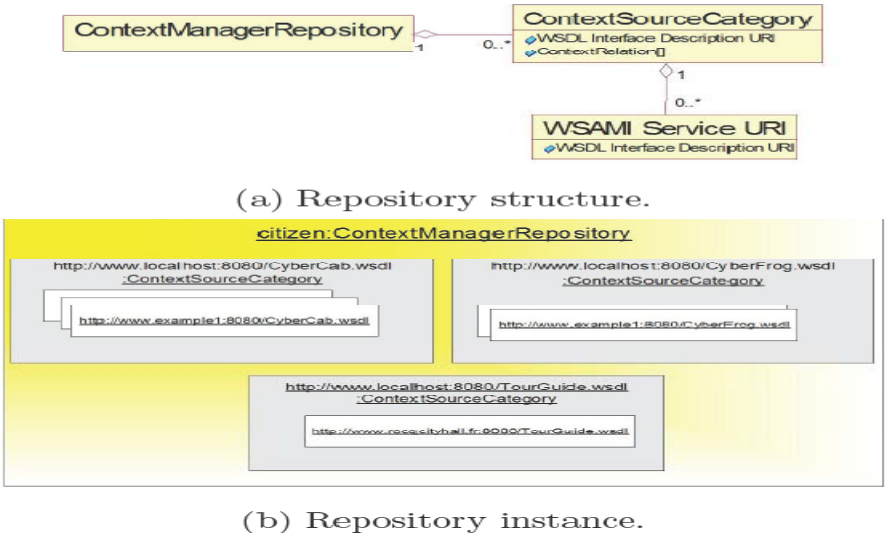

Figure 6: ContextSourcesDiscovery repository.

deployed on the French citizen's PDA. The repository comprises 2 categories for CyberCar URIs and a single category for the Rocquencourt city-hall service. The first of the CyberCar categories is characterized by the URI of theCyberFrog interface, given in Figure [6](a). Similarly, the second category is characterized by the URI of the CyberCab interface, given in Figure [6](b).The CyberFrog category

contains 2 URIs of available services, providing this interface. On the other hand, the CyberCab category contains 3 URIs of available CyberCab services.

2.1.4 Incontext

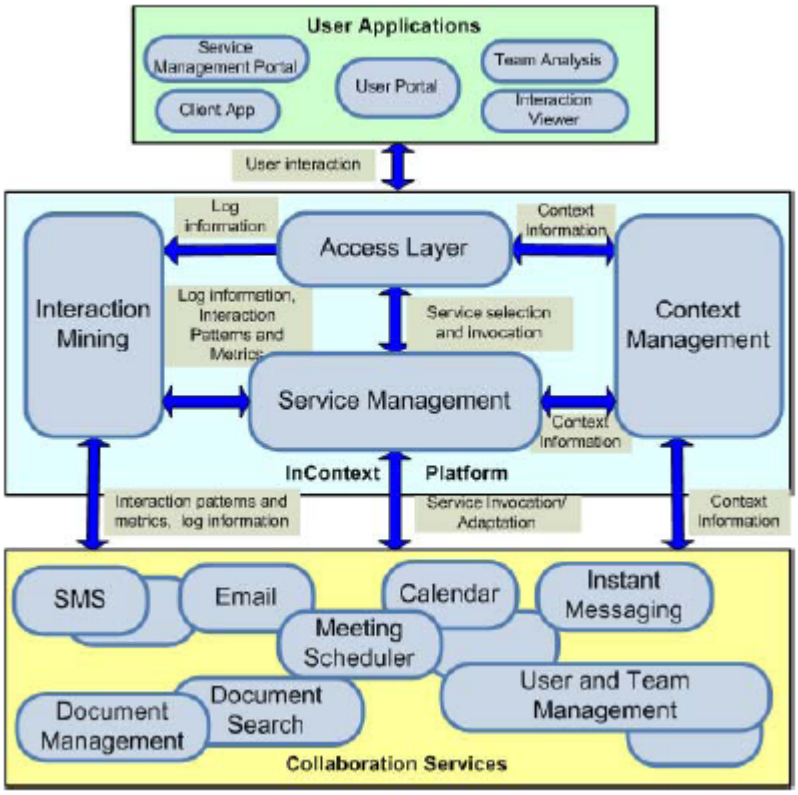

Figure 7: inContext architectural overview

Figure 7 depicts the *inContext[6]* environment which basically comprises three main parts: *Collaboration Services*, *inContext Platform* and *User Applications*. *Collaboration Services* include services that are normally required in team collaboration. Such collaboration services are for document sharing (e.g., Document Management and Document Search), communication (e.g., SMS (Short Message Service),

Instant Messaging (IM), Email, etc.), team and project management (e.g., User and Team Management and Activity Management). Those services could be specific to particular projects, but many are generic services which can be reconfigured to fit into particular purposes. The *inContext platform* is the central part of the *inContext* project, where all aspects are brought together. This part includes novel services that support advanced, dynamic collaboration of emerging teams based on context and interaction model. The *Access Layer* acts as an intermediate, receiving requests from the client side and invoking services. The *Interaction Mining* is used to extract and analyze interactions inherent within team collaborations. The *Context Management* manages context associated with human, services, teams and activities. It supports reasoning mechanisms to both infer new context information and to enrich existing context information. The *Service Management* is responsible for selecting the right services, ranking and invoking the services according to requests from *Access Layer*. All the above-mentioned components can be deployed in and operate in a distributed manner. The architecture of the *inContext* environment shown in Figure 7 is a reference implementation of the so-called *Pervasive Collaboration Service Architecture* (PCSA) that we have developed in the *inContext* project. By introducing new core services that support context- and interactionbased collaboration, the *inContext platform* is able to integrate various existing collaboration services to establish a network of PCSAs deployed in multiple organizations.

2.1.5 Context Toolkit

The context toolkit[7] has developed relies on the concept of context widgets. Just as GUI widgets mediate between the application and the user, context widgets mediate between the application and its operating environment. Authors analyze the benefits of GUI widgets, introduce the concept of context widget, detail its benefits, and explain how context-enabled applications are built using these widgets.

Learning From Graphical User Interface

Widgets

It is now taken for granted that GUI application designers and programmers can reuse existing interaction solutions embodied in GUI toolkits and widget libraries. GUI widgets (sometimes called integrators') span a large range of interaction solutions: selecting a file; triggering an action; choosing options; or even direct manipulation of graphical

objects . GUI toolkits have three main benefits:

- They *hide specifics* of physical interaction devices from the applications programmer so that those devices can change with minimal impact on applications. Whether the user points and clicks with a mouse or fingers and taps on a touchpad or uses keyboard shortcuts doesn't require any changes to the application.

- They *manage the details* of the interaction to provide applications with relevant results of user actions. Widget-specific dialogue is handled by the widget itself, and the application often only needs to implement a single callback to be notified of the result of an interaction sequence.

- They *provide reusable building blocks* of presentation to be defined once and reused, combined, and/or tailored for use in many applications. Widgets provide

encapsulation of appearance and behavior. The programmer doesn't need to know the inner workings of a widget to use it. Although toolkits and widgets are known to have limitations such as being too low-level or lacking flexibility, they provide stepping stones for designing and building user interfaces and developing tools such as User Interface Management Systems (UIMS). With context widgets, we aim at providing similar stepping stones for designing and building context-enabled applications.

What Is A Context Widget?

A context widget is a software component that provides applications with access to context information from their operating environment. In the same way GUI widgets insulate applications from some presentation concerns, context widgets insulate applications from context acquisition concerns. Context widgets provide the following benefits:

- They *hide the complexity* of the actual sensors used from the application. Whether the presence of people is sensed using Active Badges, floor sensors, video image processing or a combination of these should not impact the application.

- They *abstract context information* to suit the expected needs of applications. A widget that tracks the location of a user within a building or a city notifies the application only when the user moves from one room to another, or from one street corner to another,

and doesn't report less significant moves to the application. Widgets provide abstracted information that we expect applications to need the most frequently.

- They *provide reusable and customizable building blocks* of context sensing. A widget that tracks the location of a user can be used by a variety of applications, from tour guides to office awareness systems. Furthermore, context widgets can be tailored and combined in ways similar to GUI widgets. For example, a *Presence* widget senses the presence of people in a room. A *Meeting* widget may rely on a *Presence* widget and assume a meeting is beginning

when two or more people are present. These benefits address issues 1 and 2 listed in the introduction. From the application's perspective, context widgets encapsulate context information and provide methods to access it in a way very similar to a GUI toolkit. However, due to the characteristics of context and notably issues 3 and 4 mentioned in the introduction, distribution and dynamicity, the context toolkit has some unique features. We briefly describe the similarities with GUI toolkits and point out some major differences.

2.1.6 Gaia

A middleware infrastructure called Gaia[8] that has been used for ubiquitous computing in classrooms, offices, presentation rooms. Gaia coordinates software entities and networked devices contained in a physical space. It provides services for location, context and events about an entity in the active space.

The system is composed of three major building blocks: Gaia kernel, Gaia application framework and the applications. The kernel contains five services: event manager, context services, presence services, space

repository and context file system. The event manager provides event channels. All events are shared with other devices using the event channels. Hence the event communication is an asynchronous communication. The event manager is implemented using CORBA Event service. Context service provides context information about all entities in the active space. Eg: context providers for location of people, conditions within a room, weather conditions, etc. Components can infer higher level of contexts. A context predicate is defined as: Context(<ContextType>, <subject>, <relater>, <object>), eg: Context(temperature, room 3231, is, 98F). Higher level contexts can be inferred using conjunction, negation, implication and disjunction operations. Presence service maintains information about resources present in the active space. It uses a beaconing mechanism where each entity transmits a presence beacon, called heartbeats, at regular intervals. If the entity fails to transmit a beacon, Gaia assumes that the entity has left the space. Gaia defines four types of entities: application, service, device and person.

Space repository learns about entities entering and leaving the active space by subscribing to the channels provided by presence service. Applications use space repository during their instantiation to find suitable resources. All active space resources have an XML. When new resources are added to the space, the space repository contacts them to obtain the XML description and stores the information. This helps in sharing the entity with all other entities. Space repository is implemented using CORBA Trader.

Context file system stores the context along with user data. It is used to: 1) automatically make personal data available to applications, conditioned on user presence. 2) Organize data to simplify the location of data. 3) Retrieve data in a format based on the context of user's device properties. Context is presented in directories, where path components represent context types and values, eg: /location:/RM2401/situation:/meeting directory is generated when a meeting is in process in room 2401. These directories are automounted according to the presence of other entities. Eg: the above directory will be automounted when the speaker of the meeting enters the room.

Application framework is composed of four components: model, presentation, controller and coordinator. The model implements the application logic, the presentation exports application data, and controller maps input events (eg touch screen events, context changes) into method requests for the model. The coordinator provides meta-level functionality of the application.

Gaia uses the LuaOrb scripting language to program and configure active spaces and to coordinate the entities.

Critique:

1) Transmitting beacon is network heavy operation. This leads to heavy traffic if there are many devices. Instead other methods like GPS should be used.
2) If there are multiple presentations in a room and the order of presentations is not fixed, the system can display wrong presentation on the screens.
3) Latest update information on Gaia is not available.

2.2 Summary

I have studied these context systems to analyze the characteristics of them

Chapter 3

Background and Analysis

In this chapter, I present some background knowledge analysis framework for analysis the context systems which are essential for the understanding of my analysis

3.1 The Analysis Framework

Many approaches defining the notion of context have been proposed and several adaptive applications have been designed and implemented, by introducing the notions of user profile and context. Although interesting comparisons of context models already exist, we felt the need to establish a framework to systematically evaluate them, by defining a set of relevant, objective and rather general categories. The analysis framework[9] proposed is intended for designers that are about to develop context-aware applications and need to decide which context model is best suited for their goals. This framework, used to analyze and compare the available context models, is built on a rich set of features which characterize the models from various perspectives. These features have been derived from the analyzed systems, by

selecting the most peculiar and common ones. The first step of the analysis is the identification of the key issues for the application being developed; in this phase the designer should define which features are more relevant for his/her target application, or whether new features should be added to address specific application requirements. Here we assume data tailoring as our target application and, with respect to it, we show the most relevant features among the presented ones. The second step is the classification of the existing context models with respect to each feature. The result is a structured view of the state of the art, which enables the designer to consciously compare the various models, focusing the attention on the key issues isolated in step one. As a result, the best model is selected or, in case no satisfactory models are available for the target application, the designer might consciously engage in the proposal of a new, more appropriate context model. The features we isolated and classified are now briefly discussed: Modeled aspects: The set of context dimensions managed by the model.

• Space: does the considered context model deal with location-related aspects?

• Time: does the considered context model allow the representation of temporal aspects?

• Absolute/relative space and time: are the space and time parameters (if any) represented absolutely (e.g., GMT time reference and GPS coordinates) or relatively (e.g., “near something”, “last month”, “after that”)?

• Context history: is the history of previous contexts part of (relevant for) the context itself, i.e., the current context state depends on previous ones, or is the context a pure snapshot of the user's current environment?

• Subject: who or what is the subject of the described context? This feature refers to the point of view used to describe the context itself; some models describe the context as it is perceived by the user, while others assume the application point of view, considering, as a consequence, the user itself as part of the context;

• User profile: is the user profile (in terms of preferences and personal features) represented in the context model? And if so, how is it represented (i.e., does the system describe the user's characteristics one by one, or does it provide a role-based model of user classes)? Representation features: General characteristics of the model itself.

• Type of formalism: class of the conceptual tool used to capture the context (key-value-, mark-up scheme-, logic-, graph-, ontology-based). Different classes provide different features (e.g., high or low intuitiveness, possibility to be automatically processed, reasoning support, formal semantics) and are more or less adequate for certain applications;

• Level of formality: the existence of a formal definition and whether the formalization well expresses the intuition;

• Flexibility: the model's ability to easily adapt to different contexts: a model can be "application-domain bounded" if it is substantially focused on a single application or on a specific domain, or "fully general" if it can naturally deal with different domains or applications (i.e., is it possible to capture any kind of context with this model and how easy is it?);

• Variable Context Granularity: the ability of the model to represent the characteristics of the context at different levels of detail.

• Valid Context Constraints: the possibility to reduce the number of admissible contexts by imposing semantic constraints that the contexts must satisfy for a given target application. Context management and usage: The way the context is built, managed and exploited.

• Context construction: highlights if the context description is built centrally or via a distributed effort; this indicates whether a central, typically design-time, description of the possible contexts is provided, or if a set of partners reaches an agreement about the description of the current context at run-time;

• Context reasoning: indicates whether the context model enables reasoning on context data to infer properties or more abstract context information (e.g., deduce user activity combining sensor readings);

• Context information quality monitoring: indicates whether the system explicitly considers and manages the quality of the retrieved context information, for instance, when the context data are perceived by sensors

• Ambiguity and incompleteness management: in case the system perceives ambiguous, incoherent or incomplete context information, indicates if the system can “interpolate” and “mediate” somehow the context information and construct a reasonable “current context”;

• Automatic Learning Features: highlights whether the system, by observing the user behavior, individual experiences of past interactions with others, or the environment, can derive knowledge about the context; e.g., by studying the user’s browsing habits, the system learns user preferences;

• Multi-Context Modeling: the possibility to represent in a single instance of the model all the possible contexts of the target application, as opposite to a model where each instance represents a context. This characterization covers the focus of the model, its representation and the way context data are used; the result is a rich set of features, emphasizing that context modeling is a varied and complex problem. Depending on the specific purpose it is designed for, each model may “include” several of the listed features; we envision five classes of use, which share general sets of features, and more important, the same target field of application. These classes can be considered as a coarse-grained categorization of the context models, or as a decomposition of the context problem itself (in boldface the key features of each class).

A. *Context as a matter of channel-device-presentation.* Systems of this class are characterized by: variable context granularity, the application as subject of the model, limited or absent management of location and time dimensions, feature-based user profiling, low level of formality, limited flexibility (often considering only specific applications), and a centrally defined context. While automatic learning features can be available, context quality monitoring, ambiguity management and context reasoning are in general not supported.

B. *Context as a matter of location and environment*. Models of this class in general provide: precise time and space management, high degree of flexibility and centralized context definition. Context reasoning may be provided, offering a powerful abstraction mechanism. Information quality management and disambiguation may be available, in particular when the context information is acquired by sensors. Automatic learning is rarely exploited.

C. *Context as a matter of user activity*. The focus of this class of models is on "what the user is doing," consequently context history and reasoning are important issues. Time and space are considered relevant as far as they provide information about the user current activity2. While the level of formality may vary, the context definition is in general centralized and the user is the subject of the model. When available, the automatic learning is used to guess user activity from sensor readings.

D. *Context as a matter of agreement and sharing (among groups of peers).* Approaches of this group focus on the problem of reaching an agreement about a context shared among peers; clearly the context definition is distributed; context reasoning, context quality monitoring and ambiguity and incompleteness

management, are key issues. Sophisticated location, time and user profiling features are uncommon in models of this class. The level of formality is rather high, due to the need of information sharing.

E. *Context as a matter of selecting relevant data, functionalities and services (data or functionality tailoring*). The models of this group focus on how the context determines which data, application functionalities and services are relevant. Context definition is typically centralized, context history and reasoning are often not provided; time, space and user profile are in general highly developed and well formalized. The flexibility is usually high while automatic learning features, ambiguity management and information quality are not key issues and are often not available. The key features of this group are: the application as subject, the possibility to express both variable context granularity, valid context constraints, and multi-context models.

These classes and the identified relevant features constitute the analysis framework we propose, used in the next section to review some of the most interesting approaches to the context modeling problem.

3.2 Analysis

System	Space	Time	Space/Time coordinate (Relative or Absolute)	Context History	User Profile User Role	Variable context granularity	Valid context constraints	Type of formalization (Key valued based)	Type of formalization (Markup Based)	Type of formalization (Logic Based)	Type of formalization(Graph based)	Type of formalization (Ontology Based)	Formality level	Flexibility	Context distribution	Context reasoning	Context quality monitoring	Ambiguity /incompleteness	Automatic learning feature	Multi-context model
Music	+	+	R/A		A	+	+		+		+	+	H	+						
Contex ADDICT	+	+	R/A		A	+	+		+		+	+	H	+	C					
GAIA	+				A								H		C			+		
Contex Toolkit	+	+	R/A		A	+	+		+		-	+	H		C/D					
CoWA SAMI	+	+			A				+						C					
inContext	+	+		+	A				+			+			C	+				

3.4 Summary

In this chapter, I have discussed some essential analysis framework those are used for my analysis.

Chapter 4

Conclusions

Although a lot of work has been done, the representation and management of context can hardly be considered as an assessed issue. Due to the complexity of the “context modeling problem” as a whole and to the multitude of different applications, at the end of this comparison we advocate those models that, although being fully general, have a well defined focus, and try to support only a specific context sub problem. Indeed, we feel that the systems whose aim is to be completely general and to support the context modeling problem as a whole for any possible application, often fail to be effective. In fact, the practical applicability and usability, although not discussed because rather subjective, are important parameters, and are often inversely proportional to the generality of the model: the more expressive and powerful, the less practical and usable. Different context sub problems and applications have almost incompatible requirements, and common solutions are still not available; as a consequence, the context model should be chosen depending on the target application. The analysis framework we have proposed can, in this sense, be used by an application designer either to choose among the available models or to define the requirements of a new context model.

References

[1] From http://en.wikipedia.org/wiki/Context-aware_pervasive_systems

[2] ANIND K. DEY Understanding and Using Context *Future Computing Environments Group, College of Computing & GVU Center, Georgia Institute of Technology Atlanta, GA, 30332-0280, USA*

[3]. Context-aware views for mobile users, Extended Abstract, Cristiana Bolchini, Carlo A. Curino, Giorgio Orsi, Elisa Quintarelli, Rosalba Rossato, Fabio A. Schreiber and Letizia Tanca Dipartimento di Elettronica e Informazione – Politecnico di Milano (Italy)

[4] Initial research results on methods, languages, algorithms and tools to modeling and management of context, Telecom Italia S.p.A., Italy, 19 December 2007

[5]. Dionisis Athanasopoulos, Apostolos Zarras , Valerie Issarny, Evaggelia Pitoura Panos Vassiliadis , CoWSAMI: Interface-Aware Context Gathering in Ambient Intelligence Environments 2007

[6] Hong-Linh Truong, Schahram Dustdar, Dino Baggio, Stephane Corlosquet, Christoph Dorn,Giovanni Giuliani,Robert Gombotz, Yi Hong, Pete Kendal, Christian Melchiorre, Sarit Moretzky,Sebastien Peray, Axel Polleres, Stephan Reiff-Marganiec, Daniel Schall,

Simona Stringa, Marcel Tilly, HongQing Yu, inContext: a Pervasive and Collaborative Working Environment for Emerging Team Forms

[7] Daniel Salber, Anind K. Dey and Gregory D. Abowd GVU Center, College of Computing Georgia Institute of Technology, The Context Toolkit: Aiding the Development of Context-Enabled Applications

[8] Gaia: A Middleware Infrastructure to Enable Active Spaces Manuel Roman, Christopher Hess, Renato Cerqueira, Anand Ranganat, Roy H. Campblell, Klara Nahrstedt IEEE Pervasive Computing, pp. 74-83, Oct-Dec 2002,

[9] A Dataoriented Survey of Context Models, Cristiana Bolchini, Carlo A. Curino, Elisa Quintarelli, Fabio A. Schreiber, Letizia Tanca Dip. di Elettronica e Informazione – Politecnico di Milano (Italy)