# Chapter 13

# Context-Awareness System

**Paulo Costa, Bruno Gonçalves and Luis Miguel Botelho**

## 13.1 Introduction

Computer application basic inputs, such as keyboard strokes or pointing devices, supply only limited information about the surrounding environment. The necessity of context information grows as applications need to adapt to the environment in which they are used. This adaptation increases the application's performance and makes sure that the results are well adapted to the specific circumstances. The main objective of context-aware computing is the development of applications that, without being limited by usual input devices, acquire and use context information to better adapt to the circumstances in which interactions take place.

Since context can be acquired by a wide range of input devices (i.e., domain dependent sensors), context systems were created to provide a simple and unique source of information for applications. Context systems can then be understood as extensions of the basic input that an application can receive [4].

This chapter describes the CASCOM approach to context acquisition and management — GCMAS (General Context Management and Acquisition System). The proposal was developed under the assumption that specific information is considered context if it complies with the following definition:

> "Context is all the information related to persons, objects, locations and applications, participating or being referred in a specific interaction, which is not strictly necessary for the interaction to be accomplished, although the use of this information allows improving the quality of the interaction and often the system's performance."

The presented definition integrates aspects from the definitions put forth by Anagnostopoulos et al. [1], and by Dey and Abowd [4], which are probably the most accepted ones in the scientific community. It emphasizes the central role of interactions, and sets a clear distinction between information that is essential to the interaction and context information which, although not being essential, may be used to improve it. This distinction is humbly suggested by the authors of this chapter with the goal of providing some guidance about the difference between context-aware computing and computing in general. According to this view of context information, while non-context information (i.e., information that is strictly necessary for the task at hand) should be explicitly provided to the system when requesting it to perform some task, context information (i.e., information not strictly necessary) should be acquired by the system that receives the request, even if it has to ask it back to the requester.

The following example may shed some light on this issue. Imagine someone, say Tom, that wants to use the CASCOM Agent System to locate a Healthcare Centre. Tom might send a request to the CASCOM Agent System saying "Find me a Healthcare Centre“. This is the information explicitly sent in the message initiating the interaction. However, being a context-aware system, the CASCOM Agent System knows that Tom's location is important for discovering a Healthcare Centre more appropriate to Tom's situation. Therefore, it asks Tom's location to GCMAS, the CASCOM Context System. Having Tom's location, the CASCOM Agent System will be capable of discovering the Healthcare Centre closest to Tom. In this example, Tom's location is context information hence acquiring it is the responsibility of the CASCOM Agent System.

Other context definitions may be found in the context-awareness state of the art chapter (Chapter 5), in particular in Section 5.2.

GCMAS is responsible for acquiring, monitoring, representing and storing context information. The system is organized in two layers - an infrastructure layer and an application layer. The infrastructure layer incorporates mechanisms that allow applications to provide context information, request context information and subscribe information about selected context events. Application dependent context processing mechanisms are included in the application layer, which provides mechanisms for context modeling, aggregation, and reasoning adapted to the type of application that accesses the system.

The chapter begins with the system requirements. Following, it presents context representation decisions, namely the content and structure of the ontologies used to model context in GCMAS. Next, it presents the description of GCMAS architecture explaining each of its components and functionalities. Finally, in the last section, it presents a discussion of the context system, briefly describing an example in which GCMAS is used in one of the CASCOM medical emergency scenarios, and presenting results and conclusions.

## 13.2 System Requirements

This section presents a set of requirements assuming, without loosing generality, that context information is acquired by sensors, which is in fact the most accepted choice. The described requirements are classified in two types: functional and non-functional. The discussed requirements were identified through a review of the literature on context acquisition and management (see Chapter 5), in particular those aspects proposing design principles for context systems and context modeling (Section 5.3).

The following functional requirements, which apply in general to all context systems, are analyzed:

- Separating context capturing from context interpretation;
- Sensor information acquisition should not depend on the specific sensor but on its interface, which must comply with the specified ontology;
- Capability of acquiring information from sensors according to the specified ontology regarding the sensor;
- Easy communication between remote or local sensors and applications;
- Supporting client applications implemented using diverse paradigms and technologies;
- Provide historical context information;
- Provide static context information volunteered by applications;
- Transparent mechanism for locating the best source of context information for each context request, using sensor properties (e.g., sensor owner and hosting device);
- Run-time addition of new sensors; and
- Context information subscription mechanism.

Context capturing should be isolated from context interpretation. In this requirement, context capturing refers, for example, to reading "0300000050" from a specific location sensor; and context interpretation would be generating the following information "Ann's location is (x =30.0 m, y= 0.5 m) relative to the central point in the north doorway of the Interdisciplinary Complex building". This separation allows the interpretation mechanisms to be developed without the concern of how context information is obtained. In GCMAS this requirement is fulfilled because context interpreters, aggregators and reasoners (application layer) allow an additional abstraction of the data provided by the sensor widgets, generating application dependent information.

The context extraction mechanisms should not depend on sensors, but on the interfaces that sensors present to the system. This requirement is satisfied in GCMAS through sensor widgets, which provide an abstraction of the data acquired by each sensor.

Context systems should be capable of acquiring information from sensors according to the specified ontology regarding the sensor. This ensures compatibility between sensors and applications. GCMAS is capable of acquiring context information from any registered sensor, in accordance with the specified ontology.

Both remote and local sensors and applications should be allowed to easily communicate. The fact of sensors and applications being distributed (remote) should not have a negative impact on the easiness with which they communicate. GCMAS includes interfaces that allow sensors and applications, both remote and local, to connect to each other without difficulties. The same interface is used irrespective of whether or not the client application and the sensor widget are running on the same or on different devices.

Context systems should be prepared to receive requests from applications implemented with diverse paradigms and technologies (e.g., object oriented technologies, component-based technologies, or agent-based technologies), which communicate using different protocols. This was achieved through the inclusion of several types of system interfaces. Currently, we have implemented interfaces suitable for component-based applications and for agent-based applications. The application designer just has to select the kind of interface more adequate to the application technology.

Context systems should provide historical context information. When requested by the application, the system should provide historical information about specified aspects of the context. GCMAS includes a history storing mechanism that allows accessing to historic context information. The interface for requesting context information allows specifying the desired number of context information samples.

Context systems should also store static context information provided by their client applications. Static context is context information that does not change much over time. Rapidly changing context information is dynamic or volatile context. GCMAS has a context repository and a specific interface through which it may receive and store context information requested or volunteered by its client applications. For example, some Personal Assistance Agent may want to provide its owner gender, nationality and birth date to the context system. This information may then be requested by the physician agent where the user is being treated.

Context systems must be capable of transparently locating the best source of context information for each context information request. GCMAS has a yellow pages service where sensors register themselves as they are added to the system. This service allows GCMAS to locate required sources of context information. For instance, if a given client application requests the patient's location in a medical emergency scenario, and GCMAS has different classes of context information sources (e.g., location, gender, spoken language, birth date and nationality among others), and several specific sensors within the same class (e.g., location sensors for several users), GCMAS yellow pages service will identify the location sensor of the specific patient. Sensor descriptions in the GCMAS yellow pages service allow the inclusion of sensor properties which may be used to guide their location.

It should be possible to add new sensors to the system anytime while the system is running, without requiring the system to be reinitialized. GCMAS supports the dynamic addition of new sensors in run-time. Besides, it should also be possible to add new sensors even if the information they provide is not specified in the system ontology at the moment of their addition. GCMAS supports this requirement by allowing the sensor registration mechanism to add the definitions of the new sensor information to the context ontology. Imagine, as an example, that a software sensor providing the room temperature is to be added to GCMAS. Suppose also that the used context ontology does not include the "room temperature" concept. The sensor addition mechanism will start adding the "room temperature" concept to the ontology, and then it will register the new sensor in the system's yellow pages.

Finally, context systems should implement an alert mechanism that notifies their client applications when changes occur in specified aspects of the context. GCMAS design includes a context subscription mechanism through which client applications may subscribe specified classes of context information. When context information of the specified classes changes, the subscription mechanism sends the subscribed context information to the subscriber. In an fictitious scenario, the hospital Intensive Care Unit Agent needs to be informed of the updated heart rate and blood pressure values of a patient being carried by an ambulance to the hospital. Instead of repeatedly issuing requests for the patient's heart rate and blood pressure to GCMAS,it just needs to subscribe these two classes of context information. GCMAS will autonomously send updated information to the agent.

This section considers also a set of non-functional requirements, which apply to context systems in general and even to the generality of information management systems: performance/response time, usability, generality and reliability/robustness.

Context systems should have short response times. To fulfill this requirement GCMAS was developed with the concern of optimizing its performance, by avoiding unnecessary processing, component communication and memory usage. Since GCMAS allows the existence of redundant components, adequate load balance policies could be defined which would improve the systems response time.

To ensure their usability, context systems should provide intuitive and complete interfaces to their functionalities. GCMAS implements a set of interfaces that enable adding new sensors easily, and a simple access of applications to the system functionalities.

Context systems should be as general as possible so that they can be used by several types of applications. Any application of any domain can use GCMAS, providing that it knows its communication protocols and the applicable context ontology.

To ensure their reliability context systems should possess sensor failure control system so that they continue to function even if several sensors fail. GCMAS fulfills this requirement. GCMAS allows redundant components, for instance redundant sensors. This way, even if one instance of a given component fails, the redundant one will replace it.

## 13.3 Context Representation

To enable GCMAS and their client applications to interpret context, GCMAS represents context according to specified publicly available ontologies, as suggested by most authors, such as Dey and Abowd [4], Anagnostopoulos [1], and Chen and Kotz [2], whose work is described in the section on design principles and context modeling (Section 5.3) of Chapter 5. The definition of the context ontology, also known as context modeling, is a very important step for the development of a context system.

The simplest way to define the context ontology is proposed by Dey and his colleagues [5]. According to these authors, two of the main components of the context ontology are context elements (sources of context information), such as sensors; and context entities representing persons, places, applications and objects which the sensors refer to.

GCMAS ontology uses a similar approach. It defines the class “context element”, representing a context information supplier such as a sensor or a record in the context repository. It also defines the class “context entity”, representing the person, object, application or place referred by each context supplier (i.e., a location sensor is represented by a context element; the person that is located by that sensor is represented by a context entity). Each individual sensor can then be located in the context ontology as a “context entity”/“context element” pair. The context ontology used in GCMAS is organized in three layers: the base ontology, the distribution ontology and the context data ontology (see Figure 13.1). The base ontology describes the basic concepts of the GCMAS ontology such as context element, context entity, context value, context property and the relations between these concepts. The base ontology may also be seen as a meta-ontology defining the primitive concepts used to actually represent the context ontology of a specific domain.

The distribution ontology represents the distribution of context elements and entities on a specific scenario. The distribution ontology uses the base classes defined in the base ontology. Context information of a specific domain is described in terms of context elements, context entities, context values, and the relations between them.

The context data ontology represents a view of the context information that is closer to object oriented implementation languages. It defines the classes of context information, from the point of view of the implementation.

Both the base ontology and the distribution ontology are described in OWL ontology definition language [9]. Since the context data ontology needs to be close to the description of an object oriented programming language, it is defined in XML Schema [8].

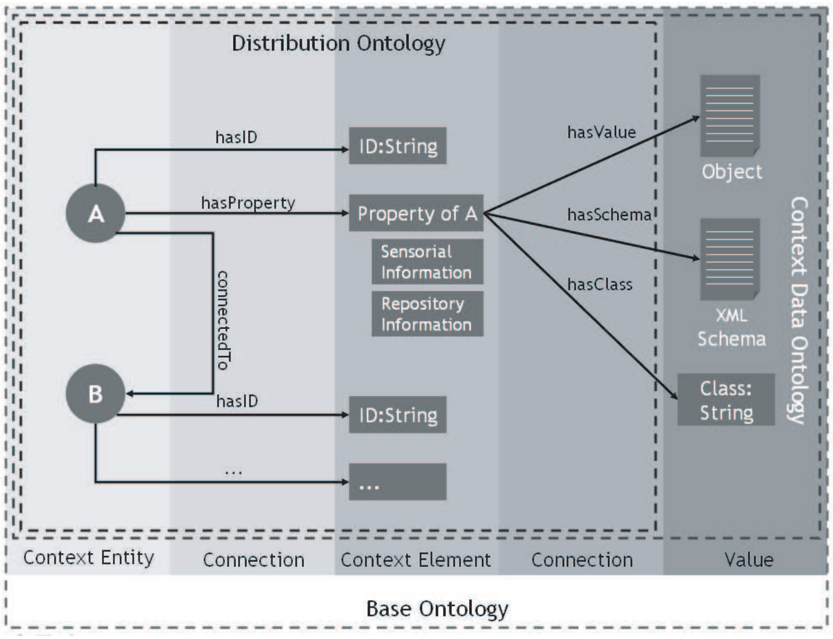

Figure 13.1: GCMAS ontology description

### 13.3.1 Base Ontology

The base ontology defines all the basic concepts that may be used to define a concrete domain context ontology. These include context elements, which represent context sources; context entities, which represent relevant persons and objects; and relations, which represent associations between context elements and entities. All elements defined in the domain context ontology must extend the classes defined in the base ontology. Besides the fundamental concepts (context elements, context entities and relations), the base ontology also defines relation links to the context data ontology, which represent the path of the classes of a specific context information, the XML Schema of the context data ontology defining those context information classes, and the actual value of the context information. These are represented by the *hasClass*, *hasSchema* and *hasValue* relations, as can be seen in Figure 13.1.

The classes defined in this ontology represent general context entities, general context elements and the basic relations between them. All context entities must be uniquely identified through the *hasID* relation. Entities are associated with their context objects by the *hasProperty* relation. Entities may also be associated with other entities using the *connectedTo* relation (see Figure 13.1). This relation allows the association between entities and elements that are related on the application

| Class | Description |
|---|---|
| *ContextObject* | Represents the context element and associated context information. Contains the context element ontology class, the object oriented class path, the XML Schema representation of context, and the context information itself. |
| *ContextEntity* | Represents a context entity. Contains the entity unique identifier, and the entity ontology class and description. |
| *Link* | Represents a relation that connects two ontology classes. |
| *SensorProperty* | Represents sensor properties. This class does not have direct connection to the ontology but uses its information. It is used for sensor registration and discovery. |

Table 13.1: Base ontology classes description

scenario (i.e., the entity *userA* connects to entity *deviceA* through a *connectedTo* relation meaning that this user has an associated device).

Each context element has three properties: *value*, representing the context information itself, linked through the *hasValue* relation; *schema*, representing its XML Schema [8], linked through the *hasSchema* relation; and its class path information, representing the class of context information (as represented in the object oriented implementation language), linked through the *hasClass* relation (i.e., the *LocationSensor* context element is represented by the object oriented class *gcmas.ontology.element.LocationSensor*). Context elements are also associated to their related context entities by the *hasOwner* relation.

The base ontology concepts defined above are represented by the classes *ContextObject*, *ContextEntity*, *Link* and *SensorProperty*. Table 13.1 represents a brief description of these classes. All context information supplied by the system is represented as instances of the *ContextObject* class. The system is responsible for translating sensorial information into *ContextObjects*. These objects represent both the context element and the context information itself. Context information is structured according to the corresponding XML Schema and stored, in XML serialized form, in the value attribute, representing the *hasValue* ontology relation.

Storing context in serialized form allows the system to store all relevant types of context information (i.e., a *ContextObject* object for a room temperature sensor could be like this one: context element — *temperatureSensorForRoomA*, the ontology instance of the class *TemperatureSensor*, representing the sensor that measures that room temperature; *hasClass* - *gcmas.ontology.element.Temperature*, the class path to the context class (as represented in the object oriented implementation language); *hasSchema*, XML Schema representation context class or a path to the file where it is represented; *hasValue*, an XML representation of context information).

As described in the base ontology, the *ContextObject* is related to a context entity represented by a *ContextEntity* object. This object represents context entities used in the context system and possesses information about them.

Link objects represent relations between ontology classes. Although most of the relations between *ContextObjects* and *ContextEntities* are made directly through the defined class attributes, it may be necessary to represent other relations such as those between entities.

*SensorProperty* objects were created to normalize sensor registration in the system. These objects specify the context element the sensor belongs to, and the context entity associated to it (i.e., a *SensorProperty* object for a room temperature sensor could be like this one: context entity *roomA*, the instance of the class *Room* (which extends the base ontology class *ContextEntity*) representing the room where the temperature is measured which is associated with context element *temperatureSensorForRoomA*, the instance of the class *TemperatureSensor*, representing the sensor that measures that room's temperature).

### 13.3.2 Distribution Ontology

The distribution ontology represents the specific context information in a given domain. This ontology extends the classes defined in the base ontology to represent the context entities and context elements identified in the given scenario.

In a simple example of a distribution ontology, in a scenario where there is a need to locate people, each person is identified as a context entity and belongs to the ontology class *Person*, extending the *ContextEntity* base class. All persons are instances of this class. The location sensor is identified as a context element and belongs to the *LocationSensor* ontology class, extending the *ContextElement* base class. All location sensors will be instances of this class (i.e., *BobSensor* is an instance of *LocationSensor*). Each instance of a *LocationSensor* is linked to the corresponding instance of *Person* by the *hasSensor* relation. This relation extends the *hasProperty* base ontology relation (see Figure 13.1).

The context distribution ontology represents all defined context elements, context entities and all their instances for a given application scenario.

### 13.3.3 Context Data Ontology

The context data ontology defines the structure of context information in a way that is closer to object oriented implementation languages.

Although Jena [6] framework is used to process OWL descriptions, there is not a direct link between OWL classes and those in object oriented programming languages.

XML Schema language is used to define context in the context data ontology layer because XML Schema definitions are very close to object oriented programming language descriptions. The context data ontology creates a bridge between the OWL ontology description and an object oriented implementation language. The XML Schema declarations define classes of context information (as represented in its object oriented implementation language) and are directly connected to the sources of context information, the context elements, through the hasSchema relation. The other properties, *hasValue* and *hasClass*, point to the value and class path of the context element.

The *LocationSensor* context element is a simple example of a context data ontology definition. This element connects to the *gcmas.ontology.element.LocationSensor* program class, defined by the *hasClass* relation. The XML Schema description of this class is defined by the *hasSchema* relation. The description specifies that location is represented by three floating point values — latitude, longitude and altitude.

When the system acquires information from a given sensor, the information is transformed according to the defined XML Schema and stored into the *hasValue* attribute of the returned *ContextObject* object (see Section 13.3.1).

## 13.4 Context System Architecture

GCMAS architecture is a mixed of a component-based system and a development framework. System components represent sensors, repositories and system functionalities as identified in the system requirements section. To enable applications to easily access the system, GCMAS provides a set of alternative kinds of interfaces to all its functionalities. The system also provides the necessary tools to convert information provided by low level context sources such as sensors into higher abstraction level representations. The provided tools also support the simple inclusion of new sensor types into the system in run-time.

Different architectures were presented in the section on context system architectures (Section 5.4) of the context state of the art review chapter (Chapter 5). GCMAS includes many of the desirable features, which are scattered by several of these architectures.

### 13.4.1 System Overview

GCMAS operates in two perspectives, as a sensor development framework and as an application interface.

In the sensor developing perspective, the system is responsible for sensor adaptation which provides hardware independent access to sensor information. Sensor adaptation consists of acquiring and transforming the context information provided by sensors into a higher level representation compliant with the context ontology. This adaptation is made by specific components, the sensor widgets.

Since there is a large range of sensor types, in order for the system to be easily adapted to all of them, general sensor adaptation tools were developed and are distributed with the system. These tools provide the means to interact with the system core components in a general way. New sensors and corresponding ontology definitions may be dynamically added to the system using these tools.

From an application perspective, the system has to implement ways of communicating with different types of applications, by providing basic interfaces to the system functionalities. The system allows the search for context and context historic information, the subscription of context events, and it also allows applications to provide context to the system. System functionalities are directly associated to specific components.

Each of the system functionalities is accessible through several types of interfaces (component-based interfaces and agent-based interfaces). This variety allows different types of applications to communicate with the system using more convenient communication protocols.

The system architecture consists of two main layers (see Figure 13.2) — the system infrastructure layer, which contains most of the system core functionalities; and the application specific layer, which contains functionalities more specific to each type of application. The application layer can be seen as an extension to GCMAS.

The system infrastructure layer possesses the basic components for context acquisition, storage and delivery. These components provide methods to process context information as defined by the context ontology, allowing context to be treated in a transparent way. By abstracting away from sensor extracted data, the system becomes open to all classes of context information, as long as they can be defined in the context ontology. The context ontology may be modified in run-time (e.g., new elements may be added when a new sensor is added to the system in run-time), allowing the system to handle new types of context information without being previously specifically prepared for that.

The functionalities provided by the system infrastructure layer allow applications to request context and context historic information, using the context query component; to register themselves on the system for receiving notifications each time context changes, using the event subscription component; and to provide context information to the system, so that it can be available to other applications, using the context storage component. Core components such as ontology manipulators, yellow pages server, and the sensor development tools also belong to the system infrastructure layer.

The application specific layer allows converting existing context information into a format more close to the application, aggregating information from several sources into single objects that make more sense to the application, and inferring new context information from existing context information acquired by the context system. Given its domain dependent nature, the components in the application layer must be developed on the side of the application.

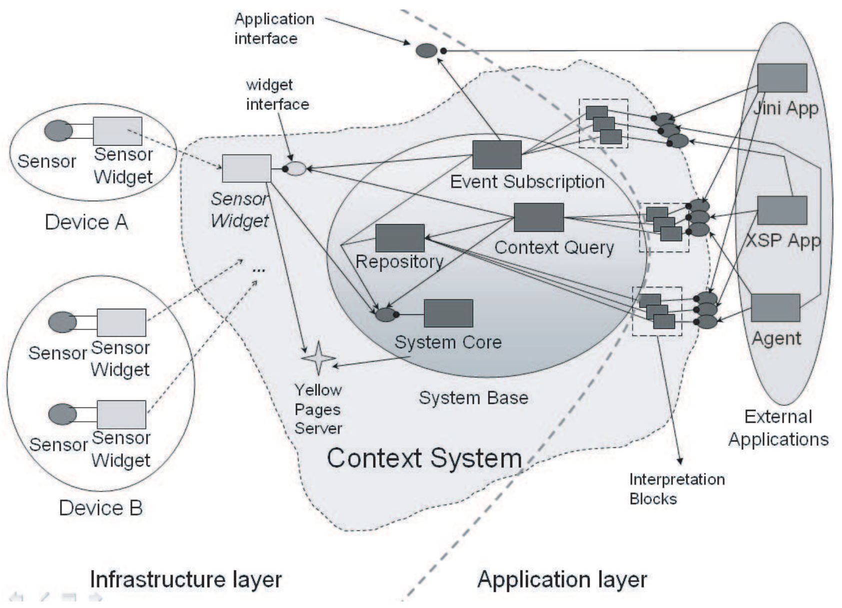

Figure 13.2: GCMAS architecture

## 13.4.2 Detailed Component Description

GCMAS is presented to applications as a black-box system. Its architecture is depicted in Figure 13.2. The figure shows only the existing application interfaces but the system can be extended with new ones. The system infrastructure layer is composed of sensors and sensor widgets, repository components, context query components, event subscription components, yellow pages server and the system core. Currently, the application layer consists only of the context interpreter which converts context information as provided by the system into a different application oriented format. The context interpreter has been designed but it has not been implemented yet.

In order to maintain the desired system redundancy, which is responsible for its robustness and efficiency, each of the system infrastructure layer components can have multiple instances running simultaneously possibly on different devices, allowing the distribution of the received requests. This is mandatory for sensor widgets since they run on the sensor side.

As shown in Figure 13.2, GCMAS provides interfaces to communicate with Jini-based applications [7], XSP-based[1] applications (both of which are component-based interafces), and agent-based applications. GCAMS allows the inclusion of

[1]XSP (eXtended Service Platform) is a tool for component-based systems development and deployment developed by the *We, the Body, and the Mind* research group of Adetti[2] and by Accedo Consulting[3].

more types of application interfaces.

**Sensor Widgets**

Sensor widgets allow sensors to communicate with GCMAS. They provide mechanisms to register sensors on the system's yellow pages server. They convert sensorial information into objects of the defined context ontology. Sensor widgets implement interfaces that handle system requests, such as context queries, and context event subscriptions.

Sensor registration allows GCMAS to be aware of their existence. Sensor registration is done by registering the widget associated to that sensor in the system's yellow pages server (see Section 13.4.2). Since widgets are seen by the system as any other component, their registration in the system's yellow pages is similar to the registration of other components. The widget must supply information about the instance of context element that represents the sensor, and the instance of context entity that represents the entity associated to the sensor (see Section 13.3). This information is encapsulated in a *SensorProperty* object and stored on the yellow pages to be used as an identifier.

When the context query component (see Section 13.4.2) processes the received context request, it locates the appropriate sensor widget and sends it the request. In response, the sensor widget, after extracting and transforming the sensorial information, supplies the requested context information, encapsulated in a *ContextObject* object, to the query component (see Section 13.3.1).

Context extracted from the sensor is transformed according to the defined context ontology. This results in creating an object defined by the corresponding XML Schema included in the context data ontology. This object is then serialized and encapsulated inside a *ContextObject* object, so that the system can manipulate it in a transparent way.

A set of tools named *Schema Class Builder* are used to support the XML Schema manipulation and serialization. These tools allow the creation of Java classes from XML Schema, and the serialization and de-serialization of instances of these classes. The *Schema Class Builder* was developed by the "We, the Body, and the Mind" research lab of ADETTI in partnership with *Accedo.*

System requests for event notification (i.e., event subscription) work in a different way. The system component responsible for this functionality, the event subscription component (see Section 13.4.2), requests the sensor widget to start supplying it the new context information each time it detects a change in the subscribed context. The event subscription component supplies the sensor widget an internal event notification interface in order for the widget to send the information to applications through it. Each time a change is detected, the widget sends the new context information encapsulated in a *ContextObject* through the supplied interface.

Context historic information is also managed by the sensor widget. Context history is locally stored, in the sensor widget, as a limited list of context samples, and can be requested by applications. Each time a new sample of context is acquired from the sensor it is stored on the historic list. Older samples are deleted when the list's retention time or list size is reached. Context historic information is made available by the sensor widgets as an array of *ContextObjects*, each one containing an encapsulated sample of context.

Since the sensor widget component is the only one in the system infrastructure layer that needs to be developed according to each specific type of sensor, the system provides basic sensor development tools named the general sensor adaptation tool. These tools allow the development of new sensor widgets, including all necessary system interfaces, yellow pages registration and ontology updating.

New classes of context information can be added to the system's ontology by the sensor widget. The general sensor adaptation tools provide the necessary means for sensor widget developers to incorporate new ontology information into GCMAS during sensor registration. New ontology definitions can include new classes of context elements and entities, along with their instances, and new XML Schema definitions of context information, or only new instances of already defined context elements and entities. Ontology update requests are then processed by the system core (see Section 13.4.2).

#### Repository Component

The repository component, which relies on a relational database, is responsible for storing and supplying context information provided by applications. Since the process of information storage in the repository is slower than context storage in sensors, the repository should only be used for context information that does not change frequently, that is, for static context.

The information sent by applications must be transformed accordingly to the XML Schema defined by the context ontology and incorporated into a ContextObject object (see Section 13.3.1). This way, any type of information can be received and stored as long as it is defined in the ontology. This same type of object is returned by the repository component when a context query is made.

The information in the repository is stored in serialized format, which is retrieved from the *hasValue* attribute of the received ContextObject. This ensures that GCMAS remains independent of the type and contents of context information.

The external interfaces implemented by this component only define context storage methods and ontology update methods. Context queries are made through the context query interface (see Section 13.4.2).

Although the context repository should in principle be used for static context information, it is also possible to obtain context historic information from the repository. Context historic information samples are stored in the database in the same way as context information. Historic information is retrieved from the repository as an array of *ContextObjects*, each containing a context sample.

### Context Query Component

The context query component is responsible for processing and responding to context queries. Context queries may be encapsulated in one of two possible objects: the *QueryObject* and the *HistoryQueryObject*. Both objects provide information about the entity which the context refers to and the context element that provides it. History queries also require information about the timestamp or number of samples to be retrieved from the history. This information allows the query component to precisely identify the necessary source of context information, and the associated sensor widget (see Section 13.4.2) or repository database records (see Section 13.4.2).

### Event Subscription Component

Event subscription implies that external applications must register themselves on the system in order to receive the new context information each time the desired context changes. Each new sample of context information is encapsulated in a ContextObject object. In order to do so the system must know the interface to invoke on the application side so that applications can receive the new context. The system offers several application side interfaces, developed to simplify the task of sending context to applications. All applications that want to receive context events must implement these interfaces. For each type of application, a specific interface of this kind can be developed as long as it extends the basic application side interface.

Context subscriptions are encapsulated in *SubscribeObject* objects. This object contains the same information as *QueryObject*. Additionally, applications must also supply a link to the application side interface that will receive the new information. After locating the responsible sensor widget on the yellow pages, the event subscription component requests it (see Section 13.4.2) to start sending context information to the specified event subscription interface. Each time new context information is received from a sensor, the event subscription component consults the relation of applications that have subscribed it, and subsequently sends them the new information.

### Yellow Pages Server

The system's yellow pages server is the point where all system components are registered, allowing system components to locate each other. In the proposed architecture, it is possible to have multiple instances of the system components running simultaneously. The yellow pages server is responsible for differentiating each instance of the components and for locating the requested instances. Each component instance may be distinguished from the others through an attribute representing its sequential number.

**System Core**

The system core is responsible for system startup and for ontology related functionalities. System startup may be configured in configuration files. These files define which interfaces are made available by the system, the number of instances for each component to be launched, and the specification of the file containing the ontology. Ontology manipulation tools enable system components to search for ontology descriptions and for relations between sensors, context elements and context entities. It also supplies the system with methods for translating ontology instances into ontology representative objects (see Section 13.3.1). The system uses the Jena framework [6] to store and manipulate its ontology. The ontology is stored in a system database for faster access, and made available for applications through an ontology definition file.

**Context Interpreter**

In the application layer, the context interpreter (i.e., context interpretation module) converts the gathered context information into an application suitable representation, in accordance to the specified context ontology. The context interpreter uses the aggregation mechanism to integrate information from several sensors into a single data structure, and the context reasoning mechanism to infer new context information from already existing information. The aggregation mechanism is used when it is necessary to create new compound pieces of context information from existing correlated context information. The reasoning mechanism is used when it is necessary to infer new context. It defines reasoning mechanisms that produce new context from collected context information, enriching it with new information.

### 13.4.3 System Deployment

Until this moment, the current section has presented GCMAS as a general purpose tool for the creation of particular context acquisition and management systems for specific applications. This subsection presents some brief guidelines pertaining the development and deployment of a context system for a specific application. Since the application to be developed is a context information acquisition and management system, the analysis process includes the identification of the context entities and the context elements in the chosen application domain. Following the context definition presented in the introductory text of this chapter, the identification of relevant context elements and entities may be achieved, starting with identifying the relevant interactions of the context client application. After identifying all entities, we identify the context information associated with them, their properties and relationships.

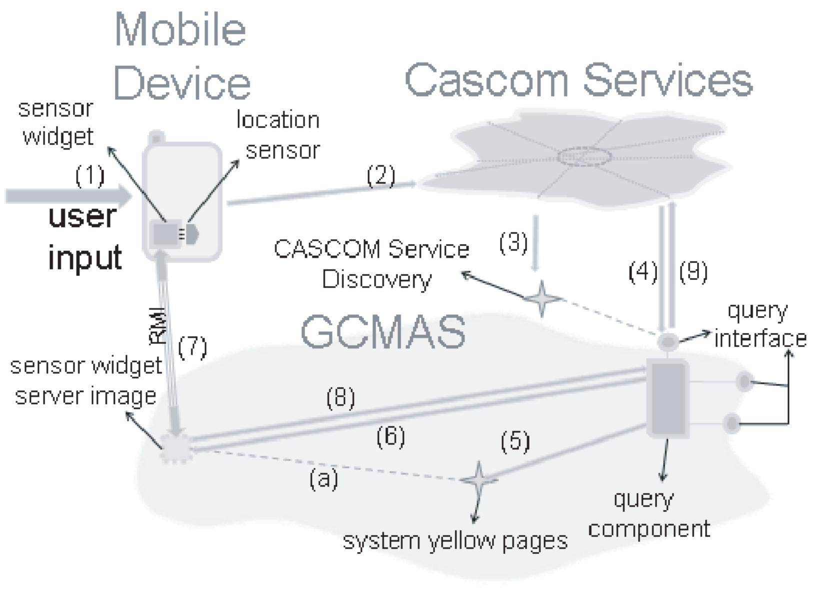

Figure 13.3: Example of interaction

## 13.5 Summary

GCMAS was developed to be used with the CASCOM system. The following example shows the way GCMAS and the CASCOM Agent System cooperate to provide valuable and context-aware assistance in a specific imagined scenario. As described in Section 1.3 (emergency assistance application), when the CASCOM Agent System needs to find the medical care centre nearest to the patient, it first needs to know where the patient is. The CASCOM Agent System simply needs to request the patient's location to the developed context system. This interaction is depicted in Figure 13.3. The interaction starts when the user logs in the CASCOM system, through his or her Personal Agent, and issues a help request, requiring medical assistance (1). The request is passed to the CASCOM Agent System (2). Since user location information is necessary to select the closest medical centre, the CASCOM Agent System queries the CASCOM service discovery for a GCMAS instance (3) and requests the user location from GCMAS query interface (4). The query component of GCMAS locates the sensor widget representing the user location's sensor, using the GCMAS internal yellow pages (5). For this, the query component uses the user and sensor classes described in the ontology supplied by the CASCOM Agent System. After locating the adequate sensor widget (a) (the widget of the user location sensor, which has a remote interface coupled with the sensor on the user's mobile phone), the query component requests it to provide its current context information (6). The sensor widget gets the information from its sensor (7), transforms it into an ontology defined object and sends it to the query component (8), which in turn passes it to the CASCOM Agent System (9), which requested it. The CASCOM Agent System then uses the user's location to locate the medical centre closest to the user.

Since GCMAS is an innovative application we cannot establish a direct comparison with other approaches. Analyzing the proposals described in context awareness state of the art chapter (see Chapter 5), it is possible to state that these applications were designed to solve specific problems of context-aware computing.

GCMAS, the proposed context system, represents a domain independent and adaptable solution to most of the problems related with context acquisition and management. GCMAS is adaptable to any situation as long as its context can be described using the proposed framework for context ontology representation. The system domain independence (i.e., the possibility to be used in different domains) arises from the ontology design and from the total independence from particular context information contents.

Tests made to GCMAS after its development provide evidence regarding the relative advantages of its architecture over related approaches. In terms of domain independence and adaptation, GCMAS goes beyond other analyzed systems. These results can be seen on Table 13.2.

The system evaluation was based on some of the interactions extracted from the CASCOM main scenario. Comparison with other approaches was based on a mere qualitative analysis of the architectures. The possibility of having multiple instances of the system components improves its robustness. If an instance of the component fails other instances can replace it. The system continues to work even if some of its components are not available.

The existence of several external interfaces allows client applications to interact with the context system using the application preferred type of interface.

The system may be extended to the application level through the context interpretation module. This allows the system to become more adapted to specific domains requiring more information than explicitly available from the system's context sources. Context interpretation also changes the way context information is represented so that applications can better understand it.

GCMAS supports the introduction of ubiquitous computing paradigm into applications. By removing the burden of obtaining information from sensors, registering and locating sensorial devices, and managing stored contextual information, application developers can focus only on application specific problems and on using the information provided by GCMAS. Applications and application designers may use the context ontology to know what kind of context information is available in the system.

By combining this system with other platforms, for instance the CASCOM agent-based service coordination platform, it is possible to take advantage of context-awareness without the burden of acquiring the relevant context information. CASCOM service coordination system uses context information to find,

| Test | Result |
|---|---|
| Context Infor-mation Retrieval | Applications were able to retrieve context information (regarding context entities and elements in the context ontology) from the system. These results apply to both sensorial context information and repository context information. |
| Context Infor-mation Diversity | Applications were able to retrieve different types of context information from the system, all of them defined by the system ontology. |
| Application Connec-tivity Diversity | Different types of applications interacted with GCMAS using different interfaces, and obtaining the same context information for the same type of request. Tests were made with three different application interfaces — XSP, Jini and agent-based interfaces. |
| Context Infor-mation Storage | The system was able to store a predetermined number of context samples in its sensors. Queries where made to each one of these in order to obtain this information and to show its availability. Location sensors and battery sensors where used in these tests. |
| Sensor Property Search | The system was capable of distinguishing two instances of sensor widgets of the same class. Queries to both instances were made in order to test their availability. These tests involved two location sensors belonging to different users. |
| Ontology Updates | The system was able to update a new definition in the context ontology. This new definition was necessary to the addition of a new sensor in run-time. After the new sensor has been added to the system, the client application was able to retrieve context information from it. Ontology update was made during sensor registration. |

Table 13.2: Functional Test Results

compose and execute services required by the user so that they become better adapted to the situation. Future versions of this system may include the definition and implementation of new types of external interfaces and the implementation of context interpretation modules.

## References

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

[2] G. Chen and D. Kotz: A Survey of Context-Aware Mobile Computing Research. Dartmouth College, Hanover, NH, USA, 2000.

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

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

[5] A. K. Dey, G. D. Abowd, Gregory, D. and Salber, D: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Human-Computer Interaction - Special Issue: Context-aware Computing, 16, 2-4, pp. 97-166, 2001.

[6] B. McBride: Jena: A Semantic Web Toolkit. IEEE Internet Computing, 6-6, 1089-7801, pp.55-59, IEEE Educational Activities Department, Piscataway, NJ, USA, 2002.

[7] Sun Microsystems. JINI Network Technology. http://www.sun.com/software/jini, 1999

[8] World Wide Web Consortium. XML Schema 1.1 Release. http://www.w3.org/XML/Schema, 2004.

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