ABSTRACT
The use of context provision middleware is a promising approach to deal with the low-level functions involved in handling contextual events when building ubiquitous applications. Several middleware are currently available that provide support for context handling, each one adopting different models both to interact with applications and to represent contextual data. We present OpenCOPI (Open COntext Platform Integration), a platform that integrates several context provision middleware systems, providing a unified ontology-based context service for ubiquitous applications. OpenCOPI encapsulates the underlying middleware platforms and represents them as services to be provided to client applications thus facilitating the development of ubiquitous applications. This paper presents the OpenCOPI architecture, main implementation issues, and a case study that discusses the use of OpenCOPI in a ubiquitous scenario.

Categories and Subject Descriptors
C.2.4 [Computer-Communication Networks]: Distributed Systems – Distributed application;

General Terms
Design, Algorithms.

Keywords
Ubiquitous and pervasive computing, context-based middleware, context heterogeneity.

1. INTRODUCTION
Ubiquitous computing [1] consists of sensor-instrumented environments, often endowed with wireless network interfaces, in which devices, software agents and services are integrated in a seamless and transparent way and cooperate to meet high level goals of user applications. Key features of ubiquitous computing systems are the high dynamism and heterogeneity of their environments and the need for context awareness. Context awareness refers to the ability of such systems to supply services not only based on information that end users explicitly provide, but also based on implicit contextual information. Such implicit information is often gathered from a set of sensors that are connected via a wireless network structure. To cope with these features, ubiquitous systems often rely on sophisticated middleware infrastructures.

Differently from traditional middleware for distributed systems that provide generic services for applications of different domains, middleware for ubiquitous systems tends to provide services tailored for specific requirements in the sense that, some middleware focus on handling contextual information to provide context awareness, others focus on providing support to mobility, while others provide autonomic capacities and so on. Such trend in designing specific middleware creates a problem to developers of applications that are required to handle a wide range of programming models and APIs from different middleware platforms. Such problem is even more evident when dealing with middleware platforms for context provision, since, besides the different programming models and APIs, the developer also needs to know the different ways to represent contextual information (the middleware context model). However, ubiquitous computing envisages environments in which applications should be able of seamlessly accessing contextual data, originating from underlying sensing infrastructure and these applications should be agnostic of the various underlying middleware technologies and their specific characteristics [2]. The need of dealing with the specificities of each middleware platform would infringe the requirements of seamlessly and transparency.

Considering the current state of the art on this field, we propose OpenCOPI (Open COntext Platform Integration), a platform that integrates several context provision middleware in order to provide a unified context service for ubiquitous applications. OpenCOPI architecture is based on the SOA architecture and on Web Services technologies, using open and solid XML based standards and languages. SOA (Service Oriented Architecture) refers to policies, practices and frameworks that enable application functionality to be provided and consumed as set of services [3]. Services can be invoked by consumers through service interface descriptions published by service providers. SOA is becoming popular in the domain of ubiquitous computing since its characteristics of loose-coupling, stateless, and platform-independence make it an ideal candidate for integrating ubiquitous devices [4]. By adopting a SOA approach, middleware are service providers which delivery context data to consumer applications.

OpenCOPI provides a unified context model, converting the several representations adopted by different platforms so that applications only need to deal with the OpenCOPI model. According to the adopted SOA-based approach, context provision middleware platforms assume the SOA role of context service providers. Often it is necessary to have different providers to handle the application needs. To support this issue, OpenCOPI also provides a context service composition. Manual and semiautomatic service composition depends on user actions for accomplishing the composition task. Therefore, in ubiquitous computing, where user interaction must be minimized, automatic...
composition is the best solution to this end [5]. Such compositions are made by choosing context services that offer context data that match with applications needs through service profile’s attributes (input, output and preconditions) and Quality of Context (QoC) attributes. QoC is “any information that describes the quality of information that is used as context information”, for instance, precision, probability of correctness, resolution, up-to-dateness, etc [6]. The matching among application requirements and available context service providers is done through a reasoning mechanism based on Web Semantic technologies [7]. Following the Web semantic vision, OpenCOPI’s context model is represented as an ontology. Ontologies offer high and formal expressiveness, preventing different semantic interpretations of the same context information. Ontologies also offer context inference, knowledge sharing, as well as software interoperability, allowing transparent integration of services.

OpenCOPI adopts the CONON ontology [8]. The CONON ontology is organized into two layers in order to deal with the fact that in ubiquitous environments the applications and services are commonly grouped as a collection of sub-domains. There are common concepts that can be modeled using a general context model and shared by all the sub domains. However, there are particularities of each sub-domain, which are detailed in separate, encouraging the reuse of general concepts, and providing a flexible interface for defining application-specific knowledge. The two-layer approach of CONON supports separation of concepts considering both their generality and specificity. The first layer is called Upper Ontology and the second layer is called Specific Ontology. “Upper Ontology is a high-level ontology which captures general features of basic contextual entities. Specific Ontology is a collection of ontologies set which define the details of general concepts and their features in each sub-domain” [8].

The use of OpenCOPI simplifies the development of ubiquitous applications. It can be used along with different platforms of context provision, providing an additional layer, aiming at supplying value-added information to the applications in their adaptive decisions. It takes advantage of the specific features of individual middleware, abstracting away their differences for the application. Therefore, OpenCOPI allows the integration of different context middleware by using an approach of automatic service composition where applications are clients of services and context middleware are providers of context services. Moreover, the service composition mechanism of OpenCOPI provides fault tolerance by dealing with situations in which a service provider available at the service discovery time becomes unavailable at runtime. This is possible since different execution plans can be generated for a given specification of service composition. For instance, in the case where two or more middleware platforms provide the same contextual information, OpenCOPI can build different plans to execute the same service composition, each one including a different middleware platform.

The remainder of this paper is organized as follows. Section 2 describes OpenCOPI features. Section 3 presents the case study. Section 4 discusses related works. Section 5 contains final remarks.

2. OPENCOPI

OpenCOPI (Open CONtext Platform Integration) is a ubiquitous platform specifically designed to integrate context services provided by several context provision middleware. OpenCOPI enables different middleware to collaborate and complement functionalities offered by each one to reach a high-level goal: supplying services and contextual data to applications. Therefore, OpenCOPI provides a platform with its own API and unified context model, in which context is handled by adopting the perspective of Semantic Web Services. In such perspective, context data is represented through ontologies; context provision middleware are service providers that publish their context services using OWL-S technology; ubiquitous applications are service consumers, and OpenCOPI is a mediator, enabling that applications only need to know the OpenCOPI context model and interfaces. Therefore, applications do not need to know details of the several different context provision middleware platforms existing in the environment. OpenCOPI hides different context provision middleware so that ubiquitous applications do not need to know the specificities of each middleware API. Figure 1 shows OpenCOPI sitting between applications and ubiquitous middleware platforms. In the proposed approach, applications request services and context information to OpenCOPI, which uses services offered by the underlying context middleware to supply the necessary information to the applications.

![Figure 1. OpenCOPI platform between applications and context middleware](image)

OpenCOPI adopts the CONON ontology (Figure 2) as its context model. The CONON ontology is OWL-based [9] and it is composed of two ontology layers.

The first layer (upper context ontology) captures general concepts about ubiquitous environments, comprising only features of basic contextual data. The second layer is composed of domain-specific ontologies which define the details of general concepts in each sub-domain (for instance, Home domain, Office domain, etc) [8]. Such separation is interesting since it allows that second layer ontologies be created and/or adapted according to the needs of each application and execution context.

Furthermore, OpenCOPI provides a mechanism for service composition. This mechanism is necessary whenever an application requires context data managed by distinct middleware platforms. So, this mechanism must discover, select, and compose services that, in conjunction, can provide all context data requested by an application and, as a consequence, meet the QoC requirements of such application. The composition service mechanism allows dealing with fluctuations in the quality of the provided service (context data) as well as with the dynamic nature of ubiquitous environments, in which sensors responsible for acquiring contextual data are prone to unexpectedly fail. Another feature of the service composition mechanism is to build different execution plans for a given specification of service composition with the goal of providing fault tolerance, for instance, in situations in which a service provider becomes unavailable in runtime. Thus, whenever a service of an execution plan fails or the actual provided QoC is lower than those requested by an application, the mechanism can replace the execution plan by
another that meets the same requirements and provides all services needed by the application. This mechanism is implemented as a component of OpenCOPI called Workflow Manager (presented in Section 2.1).

**Workflow Manager:** This component manages the available context services. It is composed of four (sub)components that provide support for the specification of semantics workflows and for the generation of execution plans [10]. The Service Manager component is responsible for importing and validating OWL-S descriptions of semantic web services. The Service Manager also helps the Workflow Manager in specifying semantic workflows by providing capabilities to search the basic concepts of the knowledge base through inputs, outputs and pre-conditions of a semantic web services. The Semantic Composer component is responsible for discovering and composing web services into a semantic workflow. First, it tries to discover from among a range of services available in the Semantic Services Repository, those that can be used to compose the execution plan given the goals of the application request. After this step, this component tries to combine the discovered services in order to consume all inputs and pre-conditions and produce all outputs desired by the request. Finally, the combined services are organized by the Semantic Composer according to the message flow between outputs of a service and inputs of the subsequent service. The Workflow Executor component supports the workflow execution. For that, it receives execution plans generated by the Semantic Composer for a specific workflow, and chooses a plan to run (taking into account the QoC measured at runtime of the service providers included in the plan). At runtime, this component issues remote calls to the underlying context provision middleware platforms, represented as web services operations and included in the execution plan. In case of failure in the execution plan, the Workflow Executor chooses another execution plan (if any) in an attempt to successfully execute the workflow. Finally, the Semantic Services Repository component is responsible for storing both the ontologies that describe the context Web Services and the execution plans.

**Context Information Repository:** This component stores the context data acquired by the underlying context provision middleware as well as ubiquitous ontologies. It supplies context data to both the Workflow Manager and the Context Reasoner components.

**Context Reasoner:** Context reasoning refers to the process of producing high-level context deduction from a set of low-level contexts [11], checking and solving inconsistent context knowledge due to imperfect sensing [12]. The context reasoner component is responsible for making inferences about context data acquired through the context provision middleware to supply high-level and consistent context data for the applications, where the high-level context should be deduced through low-level context provided by the different middleware platforms.

**UnderlayMiddlewareInterface:** This interface is responsible for the interaction between OpenCOPI and context provision middleware platforms that are based on Web Services technology. Through this interface, OpenCOPI is able to receive context data produced by a context provision middleware. To support this issue, the middleware needs to publish their context services in OpenCOPI by using OWL-S profiles. These profiles are stored in the Semantic Services Repository through the Service Manager and are used by the Semantic Composer to create semantic workflows.
Underlay Service Manager: This component is responsible for the communication between OpenCOPI and context provision middleware platforms which are not compliant with the Web Services technology. Since these middleware platforms implement their own communication API it is necessary that OpenCOPI abstracts away these different APIs in the access of the context data provided by these classes of provision context middleware platforms. The ServiceFactory component is responsible for creating context services that encapsulate the specific middleware APIs while the ServiceBridge component makes the link between these context services and the Workflow Manager component. Thus, each service provided by the middleware API is represented by a Web Service, created by OpenCOPI, to represent the service API. For each context provision middleware not compliant with Web Services, it is necessary to build a driver. This driver implements a context model transformation (from the middleware context model to the OpenCOPI context model) and it issues context queries and subscriptions from OpenCOPI to context provision middleware. Therefore each OpenCOPI web service created by the ServiceFactory uses a driver tailored for the specific middleware.

2.2 Implementation
This Section describes important decisions at the design level taken in the development of OpenCOPI.

2.2.1 Service profiles
OWL-S Profiles provide a way to describe the services offered by the providers and the services needed by the requesters [13]. These profiles are used by applications to describe their interests in specific contextual data.

OpenCOPI has two OWL-S Profiles used to describe its provided services as the way by which applications acquire context data. One profile addresses the gathering of context data on a synchronous way and the other is for asynchronous acquiring of context data. The synchronous profile is composed of the specification of inputs required by the service and the generated outputs. Thus, the inputs describe the context data synchronously requested by ubiquitous application and the outputs are the context data to be delivered by the service after its execution. The asynchronous profile is similar to the synchronous profile, but it includes additional information called preconditions. This profile’s attribute describes the necessary condition (for instance, a minimum value for a context data) for a service to deliver context data to ubiquitous applications. Figure 4 shows the Asynchronous service profile describing functional attributes (input, output and preconditions).

2.2.2 Inference machines
OpenCOPI Context Reasoner is composed of several inference machines, implementing distinct technologies (description logic, first order logic, temporal logic, fuzzy logic, neural networks, etc) for context reasoning which are accessed through a single API. The possibility of using several inference machines is important since depending on the context information to be inferred and the available low-level context, the ideal machine can be appropriately chosen to be used for such inference. For example, inferences based on various events might need to use some form of temporal logic to express the rules. Inference based on uncertainties may require some form of fuzzy logic. In other cases may be used other types of logic like first order logic, higher order logic, description logic, etc [14]. Pellet [15], an open source OWL-DL reasoner developed by Clark & Parsia, is the first inference machine to be integrated in the Context Reasoner module. It was chosen because it is already used in the Workflow Manager component to perform inferences on the choice of services to build the execution plans.

2.2.3 Provision middleware drivers
As it was previously explained, for each middleware which is not web service compliant a specialized driver implementation is required. This driver is a specialization of a GenericDriver class and implements the methods illustrated in Figure 5.

The register() and unRegister() methods are used to register the context provision middleware in OpenCOPI. The getcontextData() method is used to acquire synchronously context data from the respective middleware. subscribe() and unsubscribe() are used to support asynchronous communication, i.e. to respectively subscribe and unsubscribe the interests for events handled by the underlying context provision middleware. Finally, the callback() method is used by a context provision middleware to notify OpenCOPI about the occurrence of subscribed events. A driver must also implement two methods (convertOpenopiModelToUnderlayModel() and convertUnderlayModelToOpenopiModel()) to convert the OpenCOPI context model to the context provision middleware context model and vice-versa.

The input is the ontology context entity which the applications is interested in, the output are the context data consisting of the context entity and its properties. The precondition context entity is composed of the properties restrictions for the publish-subscribe communication. For example, the input could be a cardiac patient (Person entity), the output could be his/her cardiac activity properties and the preconditions could be Person.BPM (beats per minute) greater than 150. An additional precondition should be used to describe quality of context (QoC) attributes.

Figure 4. Asynchronous Service Profile.
3. CASE STUDY
This Section presents a motivational case study used for discussing the benefits of our proposal and for highlighting the differences between OpenCOPI and other solutions, explaining the need of using several middleware concurrently. We choose a Day-by-Day application that consists of a scenario in which a man (Alfred) is equipped with his handheld to facilitate his day-by-day activities anytime and everywhere. The following subsections first present a scenario that contextualizes this application and then a motivating discussion about the case study.

3.1 Scenario
Alfred uses his handheld all day long, anywhere he goes. For instance, when he is driving, his handheld receives context information about the car engine, as engine oil level, brake fluid level, brake pads wastage, tires wastage, etc. Therefore, Alfred is notified whenever something is out of the standard and that might risk the operation or performance of his car. Furthermore, Alfred can access information about the traffic and also, when he is approaching a particular mall, he can receive a list of products with sale discount that he has previously marked as of his personal interest. With that list, Alfred stops in the mall to buy some products. After this, he queries his handheld to discovery if there is some available friend, Alfred accesses the friend’s current location and goes to meet him. Supposing that there is no available friend, Alfred receives indications about movies sessions ongoing in mall’s cinemas based on his preferences (category of movie, actors, directors, etc) and with the trailer of each movie. If Alfred decides to see some movie, he makes the ticket booking automatically through his handheld.

Already at home, Alfred turns on the living room television, and according with his preferences, the television switches to a movie channel. This channel is presenting a violent movie, inappropriate for children. Suddenly, Alfred’s son with his MP3 player, equipped with 802.11 interface, arrives in the room and the TV automatically switches to another channel (favorite Alfred’s son channel), but begins to record the movie in the background. When Alfred’s son leaves the room Alfred can return to see the film in the exactly point where he had stopped.

3.2 Discussion
This heterogeneous scenario depicts some environments that Alfred is inserted in during the day. In each environment (car, mall, house, etc) there are different types of information handled by the Alfred’s device’s application. Furthermore, each environment is composed of different technologies and context provision middleware which are transparent to the application and consequently to Alfred. For example, the car environment is composed of a car platform to provide information acquired by sensors and GPS equipment embedded in the car. In this environment it is also necessary the communication with the transit department middleware responsible to supply traffic data. Another infrastructure is necessary to provide marketing context information as the used by the mall to provide the products list with sale discounts. In the mall environment, as well as in the car, Alfred uses location context information. However, in the car he uses a GPS service, while at the mall he must use another technology, for example, the RFID-based service or the 802.11 triangulation algorithm based on the signal strength. In the mall environment is necessary also an infrastructure to supply context information about the cinemas. The home environment is composed of a middleware to provide location context in order to identify the current location of residents along with users preferences regarding environmental conditions (light intensity, room temperature, etc). Moreover, the Set-top box middleware, linked with the TV, is used to manage the user’s channel preferences, switches channels automatically and control digital TV applications.

Other interesting point is the adaptation of the application necessary whenever a user enters a new environment. Whenever this occurs, the ontology to be loaded in the handled is changed at runtime to the specific ontology of the respective environment (car ontology, mall ontology, house ontology, etc) and OpenCOPI only uses the services of middleware that are spread in the respective environment.

In order to contextualize the OpenCOPI execution process, a fragment of the described scenario will be explored. When Alfred arrives at the mall, the handheld application subscribes to OpenCOPI to receive information: (i) about the items Alfred whishes to buy; (ii) if there are some Alfred’s friend at the mall; and (iii) the list of movies in exhibition filtered according to Alfred’s preferences. Since there are several web services provided by the various middleware registered in the Semantic Services Repository component, including two web services that provide information on location (one based on 802.11 RSSI and other based in RFID), OpenCOPI receives Alfred requests and the Semantic Composer component starts the workflow creation process. The Semantic Composer component looks for web services that can implement the workflow and with these services creates the possible execution plans for this workflow. In this example scenario, the OpenCOPI Semantic Composer creates two execution plans, both composed of a location service (RFID-based or 802.11-based), a service to announce products with sale discounts, a service that lists movies in exhibition and a service to book movie tickets. After this, the Workflow Executor component uses the services QoC values to choose a plan to run. In this case, the execution plan chosen is the plan that includes RFID-based location-aware web service because this service has the highest precision and a higher probability of assuring the QoC. After choosing the execution plan, the Workflow Executor can execute the plan in order to provide the required contextual data to the application. So, firstly the service to discover products in sale and the location service are executed. In case of there is no Alfred’s friend at the mall, the services to list movies and to book movie tickets are executed.

4. RELATED WORK
In this Section, we present works that are directly related to our proposal. Several middleware have emerged over the last years to assist the development of context-aware systems but they usually do not meet all the requirements demanded by the highly dynamic and heterogeneous ubiquitous environments. For example, some middleware focus on handling contextual information to provide context awareness, while others focus on providing indoor or outdoor location-awareness and others on providing support to mobility, etc. Furthermore, each middleware has its own contextual representation and API for contextual acquisition. In this context, some middleware are domain-specific, for instance, PHM (Pervasive HealthCare Middleware [16]) and Gong et al [17] which are specific to meet the needs of HealthCare applications.
Context Toolkit (CT) [18] is a framework that aims at simplifying the design and implementation of general context aware applications. Widgets are CT components responsible for acquiring contextual information from sensors and providing this context information to applications. CT adopts an attribute-value context model which lacks capabilities for sophisticated structuring that enables efficient context retrieval and inference algorithms.

SOCAM [19] is a service-oriented middleware that uses an ontology context model and provides contextual information through Web Services described in OWL-S. SOCAM proposes and uses the CONON ontology, the same OWL ontology chosen for OpenCOPI. SOCAM (and OpenCOPI) have an inference structure composed of various inference machines and supports QoC analysis. Comparing to OpenCOPI, SOCAM does not: i) provide location-aware context information and, ii) provide resilience in the presence of service fails, iii) provide mechanisms for middleware integration.

Differently from OpenCOPI, all the above described middleware platforms lacks a mechanism for dealing with the inherent heterogeneity of ubiquitous and pervasive environments. Hence, OpenCOPI is unique in enabling accessing different context provisioning middleware platforms in a transparent and uniform way, allowing applications to use the best resources of each middleware, simplifying the application code and consequently contribute to reduce its size. For the best of our knowledge, there is no middleware for ubiquitous computing with all the integrated features that OpenCOPI provides.

5. CONCLUSION

In this paper we introduced OpenCOPI, a ubiquitous platform that integrates several context provision middleware and provides a unified ontology-based context services for the development of ubiquitous applications. In OpenCOPI the underlying context provision middleware platforms are service providers that publish their context services and ubiquitous applications are service consumers that use OpenCOPI API without knowing the underlying context platforms. OpenCOPI provides an automatic service composition mechanism that discovers, selects and composes services from different underlying middleware and generates execution plans. This mechanism works together with a reasoning mechanism to match application Quality of Context requirements with the context information provided by the underlying middleware platforms.

6. REFERENCES
