Using Semantic Web Techniques for Middleware Integration in Ubiquitous Computing

Frederico Lopes¹, Flavia C. Delicato², Thais Batista¹, Paulo F. Pires², Everton Cavalcante¹, Thiago Silva¹, Reginaldo Mendes³

¹ DIMAp – Federal University of Rio Grande do Norte – Natal, Brazil
² DCC – Federal University of Rio de Janeiro – Rio de Janeiro, RJ, Brazil
³ SERPRO – Federal Service of Data Process – Brasilia, Brazil

{fred.lopes, fdelicato, paulo.f.pires, evertonranielly, thiagosilva.inf, jrsmjr}@gmail.com, thais@ufrnet.br

Abstract. In this paper, we present OpenCOPI, a middleware level platform that integrates context services and allows quick and easy development of context-aware ubiquitous applications. This is achieved by supporting automatic service composition that enables the use of context services from several sources in a transparent and automatic way, allowing applications to be independent of the underlying context provision environment. OpenCOPI is based on SOA, Web Semantics, and workflows technologies. This paper describes OpenCOPI architecture and implementation.

1. Introduction

The current trend in ubiquitous computing is the emergence of complex and added value context-aware applications in which platforms based on heterogeneous networks, interfaces and services technologies are used to provide user-centric applications. The user-centric perspective means that smart spaces are aware of user needs and activities and reactively, or even proactively, satisfy user demands by composing and deploying the appropriate services and resources [1]. In this scenario, service is a consistent piece of functionality made available over the network by a software entity and accessed by others – customer – software entities. Thus, ubiquitous applications are built through the composition of services made available through various services platforms. As a consequence, application developers have the burden of dealing with a wide range of service discovery mechanisms, programming models, APIs and non-interoperable protocols provided by distinct middleware platforms. It configures a conceptual paradox as ubiquitous computing considers environments in which applications are able to access context information provided by distributed underlying infrastructures, so that these applications do not need to know the existence of those platforms and their specificities [2]. The need to know the specificities of each platform violates the transparency requirement and increases the complexity of applications development.

The goal of this work is to present OpenCOPI (Open COntext Platform Integration), a platform that integrates context provision services and provides an environment that allows a quick and easy context-aware application development. OpenCOPI allows the integration of services provided by distinct service sources, from services offered by simple ad hoc sources to more complex services, as other context
provision platforms. Integration of services through *service composition* is crucial to facilitate the application development, enabling the use of services of several sources in a transparent and automatic way, and making the applications independent on the underlying distributed environment and on the underlying context provision mechanisms. OpenCOPI architecture is based on the SOA architecture and on Web Services technologies, using open and solid XML based standards and languages. The use of standardized context model and mechanisms are important to provide a unified environment in which applications only need to deal with OpenCOPI models. In short, OpenCOPI provides: (i) an automatic service composition based on user preferences and services metadata; (ii) a standardized context and communication model; (iii) an adaptive mechanism that cope with service failures, service’s quality fluctuation and user mobility.

Besides using WS* technologies, OpenCOPI is also based on semantic workflows. A workflow describes the order of a set of goals to be performed by several services to complete a given procedure [3]. The composition model adopted by OpenCOPI is based both on services functionality and on service metadata, enabling a better choice between available services with similar functionality, e.g. services with same inputs and outputs. Such compositions is performed by choosing context services that offer context data that match with applications needs through QoS and QoC (Quality of Context) attributes. QoC is “any data that describes the quality of information that is used as context, for instance, precision, correctness, resolution, etc [4].

The focus of this paper is to present OpenCOPI. Section 2 describes OpenCOPI features and architecture. Section 3 presents a motivational case study, while Section 4 discusses related works. Section 5 contains final remarks.

2. OpenCOPI

OpenCOPI integrates services provided by different service sources, including other provision context platforms. OpenCOPI enables different platforms to collaborate and complement functionalities offered by each one to reach a high-level goal: each platform supplies services and contextual data to applications. Therefore, OpenCOPI provides a platform with its own API and context model in which context is handled by adopting the perspective of Semantic Web Services. In such perspective, context data is represented through ontologies. Service providers publish their services using OWL-S technology, ubiquitous applications are service consumers, and OpenCOPI is a mediator, which enables applications only need to know the OpenCOPI context model and interfaces. Thus, the applications do not need to know details of others underlying context provision middleware platforms. According to the adopted SOA-based approach, context provision middleware platforms and other systems (databases, legacy systems, etc.) assume the SOA role of context service providers, applications are service consumers and the application logic is described as business process. Often it is necessary to have different providers to handle the application needs. The use of Web services technologies makes easy the interoperability and integration of services offered by different service providers.
OpenCOPI’s context model converts several representations adopted by different platforms so that applications only need to deal with the OpenCOPI model. Ontologies offer high and formal expressiveness, preventing different semantic interpretations of the same context information. Ontologies also offer, in the context of OpenCOPI, context inference, knowledge sharing and software interoperability, allowing transparent integration of services.

2.1. Terminology

This sub-section presents important terms and features necessary to fully understand OpenCOPI operation.

Services. They are the basic elements in the OpenCOPI architecture and their features and functionality are described using OWL-S. There are two types of services in OpenCOPI: traditional services and context services. Traditional services are services provided by databases, legacy systems, message (SMS, e-mail, twitter) systems, etc. In order to perform service selection and composition with context services (services provided by context provision middleware), additional quality metadata is required, for example, the values of QoC provided by the service.

Semantic workflow. It is an abstract representation of a workflow described in terms of goals, representing the application execution flow. A workflow defines the sequence in which these goals must be executed. Goals are composed of tasks, which are described in terms of Semantic Web services descriptions. Workflows are used to perform automatic service selection, composition, and orchestration. In OpenCOPI environment, each application has its own workflow and each workflow goal is a high level description of an application goal. A workflow is independent on specific concrete services. This approach, of separating the abstract goal from the concrete services that are able to achieve it, is useful mainly in cases where there are several similar services available, offered by different providers. In such cases, the service that best meets the user requirements can be chosen to be executed based on a given high level workflow goal.

Execution plan (EP). In order to execute a semantic workflow, it is necessary to create at least one concrete specification for the workflow, which is called execution plan (EP). Such EP contains a set of concrete Web services that are orchestrated through the execution of services in a particular order. EPs are built through an on the fly process of service discovery and composition, according to the semantic enriched interface of the selected services and the semantic workflow specification. Figure 1 presents an example of workflow’s graph in which each complete path between initial node and final node is an execution route so that it represents a possible EP. Therefore, the graph represents the workflow with all possible EPs. Some examples of EPs: (i) $S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_5$, (ii) $S_1 \rightarrow S_2' \rightarrow S_3 \rightarrow S_4'' \rightarrow S_5$, etc.
2.2. Context Model

The OpenCOPI context model is ontology-based. Following the CONON [5] approach, OpenCOPI OWL ontology is organized into two layers 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 (e.g. about ubiquitous environment, including only the basic features of these environments) that can be modeled using a general context model and shared by all the sub-domains (Generic Context ontology layer). However, there are particularities of each sub-domain (e.g. home, office, petroleum exploration, etc), which are detailed in separate, encouraging the reuse of general concepts, and providing a flexible interface for defining application-specific knowledge (domain-specific ontology layer). The OpenCOPI Generic Context ontology is structured around objects and tasks classes. Objects are physical or conceptual things, including, Person, Equipment, Computer Entity, Message and Location. Tasks are used to represent an activity implemented by web service(s). Objects can be used to describe inputs, outputs and pre-conditions related to a task. Furthermore, association among tasks and objects should be defined in order to express the relationship between them, facilitating the description of semantic workflows. This association is called workflow goal (Task, Object).

Figure 2 presents a partial definition of Oil Exploration domain ontology. This ontology describes the specific sub-classes (objects and tasks) used in a case study performed in our work (see Section 3.1). Moreover, this ontology shows some relationships between object-object and task-object. For example, the Subscribe task can be used to define tasks to monitor some parameters of the oil exploration, e.g. burden, BSW (amount of water dissolved in the oil) or limit of oil flow.
2.3. Service Representation and Metadata

**Service representation.** In OpenCOPI, each service has an OWL-S representation used to allow semantic service composition. Services are semantically composed through service ontologies to satisfy the semantic workflow built by the user. Each service ontology is built from WSDL description. In the process to create the OWL-S document, each operation (endpoint) of a Web service is represented as an atomic process of a semantic Web service. The operations’ parameters (input and output) are represented as objects in the context model ontology.

**Service metadata.** The service selection is based not only on the workflow created by the user but also by some metadata parameters handled in a uniform way. These parameters are categorized (didactically) in QoC (precision, correctness, resolution and freshness) and QoS (responding time, availability and performance) and their possible values are represented as ontology concepts.

2.4. Architecture

OpenCOPI architecture encompasses two layers (*ServiceLayer* and *UnderlayIntegrationLayer*) and two major interfaces (*IApp* and *IUnderlayIntegration*) as depicted in Figure 1. *ServiceLayer* is responsible for managing the abstractions (OWL-S descriptions) of services supplied by service providers. The components of the *ServiceLayer* use such abstractions to support workflow creation and execution, service selection, service composition and adaptation. Such components also support context reasoning, context storing, among other functionalities related to ubiquitous applications. *IApp* interface links applications with the OpenCOPI *ServiceLayer*. The *UnderlayIntegrationLayer* is responsible for integrating service providers, mainly (but not only) the context provision middleware, performing context conversion whenever is needed (from middleware context model to OpenCOPI context model) and communication protocol conversion. This latter type of conversion is required to integrate middleware that are not compliant to Web services protocols and standards. The *IUnderlayIntegration* interface links service providers and OpenCOPI's *UnderlayIntegrationLayer*.

*WorkflowManager* component manages the abstraction of available context services provided by context provision middleware. It is composed of four (sub)components that provide support for the specification of semantic workflows and for the generation of EPs. The *ServiceManager* component is responsible for importing OWL-S descriptions from service providers to OpenCOPI and validating such descriptions. The *ServiceManager* also provides capabilities to search for basic concepts in the knowledge base (ontology) using inputs and outputs of available semantic Web services. The *SemanticComposer* component is responsible for discovering and composing Web services according to semantic workflow specifications, i.e. it makes the mapping between workflow goals and Web services. First, it tries to discover (among a range of services available in the *SemanticServicesRepository*), those that can be used to compose the EP, given the goals specified in the application request. After this, it tries to combine the discovered services in order to consume all inputs and pre-conditions and produce all outputs specified in the request. The combined services are organized by the *SemanticComposer* according to the message flow between outputs of
a service and inputs of the subsequent service. The SemanticServicesRepository stores both the ontologies that describe the Web services and the EPs. The WorkflowExecutor component supports the workflow execution. It receives EPs generated by the SemanticComposer for a specific workflow, and chooses a plan to run (taking into account the QoS/QoC of the service providers included in the plan). At runtime, this component performs remote calls to the underlying context provision middleware, represented as Web services operations and included in the EP. In case of failure in one EP, the WorkflowExecutor chooses another EP (if any) in an attempt to successfully execute the workflow.

The function of the MetadataMonitor is to acquire metadata about services and context provided by context provision middleware to feed the ContextInformationRepository with metadata information. The techniques for acquisition and transformation of quality metadata (QoS/QoC deduction, filtering or extrapolation) are out of scope of or work.

The ContextReasoner component makes inferences about context data (low-level context), acquired through the several context provision middleware, to supply high-level and consistent context information for the applications. ContextInformationRepository component stores context data and context metadata.

AdaptUbiFlow component is responsible for the adaptation process in OpenCOPI. As it was previously mentioned, ubiquitous computing environments are highly susceptible to changes, several of them unpredictable. AdaptUbiFlow was specifically designed to deal with the requirement of adaptation. In AdaptUbiFlow, an adaptation of an application means the replacement of the running EP by another EP (that achieves the same stated goals). This component works directly with MetadataMonitor and WorkflowManager component to identify a fault and automatically change the execution flow to use another EP in the presence of fault.

DevicesManager manages the user computational entities to allow the application migration from a device to another in case of user mobility or in case of resource limitations of the active device (e.g., low level of energy, low level of free memory). These devices can provide services to be consumed by applications, including services to provide device’s context information (e.g., location, battery level, free memory level), in which each device type has its own set of context information. The computational entities can be laptops, smartphones, tablets, etc. Each computational entity has a DeviceController to control and monitor the entity activity. DeviceController sends the actual device’s status to OpenCOPI (DevicesManager) and allows OpenCOPI to change the execution from actual device to another device. DeviceController is also responsible for supporting communication between the device and OpenCOPI.

The components of UnderlayIntegrationLayer are in charge of integrating service providers. ServiceDiscovery is the component that discovers services in the environment and registers them in OpenCOPI. When discovered, Web services need to be integrated with OpenCOPI. For each context provision middleware, it is necessary to build a driver to implement the context model transformation (from the middleware context model to the OpenCOPI context model). For context provision middleware which does not provide APIs complaint to the Web services technology it is necessary to
build drivers in order to abstract away the such different APIs and allow the transparent access to the context data provided by these context provision middleware. So, the driver is also responsible for issuing context queries and subscriptions from OpenCOPI to the underlying context provision middleware. Each driver should extend the GenericDriver component. This component implements the OpenCOPI side of the interface and defines operations for context model transformation and communication between a specific context provision middleware and OpenCOPI.

The drivers for context provision middleware are built at development time through a tool to convert underlay context representation from underlay platforms to OpenCOPI and to provide communication between them. Regarding context transformation, currently there is not a well-known and widely employed standard to represent context information and metadata and not even a consensus about the best formalism to model context-aware applications. Most of context provision middleware platforms do not have a well-defined context model. Others have context model based on approaches different from OpenCOPI, which is an ontology-based model, (particularly the CONON ontology model is adopted). In this perspective, a MDD approach [20] performs an efficient and automated mapping between models of underlying context provision middleware, conformed to a meta-model adapted from ARCAMODE model [21], and the OpenCOPI context model. This transformation between context models is performed through ATL [22] transformation rules.

![Figure 3. OpenCOPI architecture](image)

3. Motivational Case Study

The case study is a Gas & Oil Industry application that monitors an oil well in production through a pumping unit machine to detect the need to change the pumping unit settings. These modifications may be necessary to increase the oil production and/or decrease the abrasion of the equipment. Depending on the situation, the application can trigger necessary actions to make changes or directly notify the responsible (human) for taking decisions about the pumping unit reconfiguration. This application was chosen because it uses different types of context information provided by many sources.

To exemplify the use of OpenCOPI in the context of such an application, we selected the *Burden* variable to be monitored, which denotes the charge of oil extracted from a well at each cycle of movement of a pumping unit. Each pumping unit has a
specific maximum value of burden for its correct operation. If this value is reached abruptly, the operation of the pumping unit must be quickly stopped to prevent its damage. Furthermore, there is another specific and intermediary value. This means a value that needs attention, where actions can be taken to prevent the pumping unit to be stopped, consequently avoiding loss of production and risks to the equipment.

The service providers in this case study are: WellDatabase provides information about oil well. It is responsible for asynchronously providing the value of burden of each pumping unit. It abstracts a component of a context provision middleware, the Context Toolkit (CT) [6], used in the case study to allow that events related to the pumping unit burden are transferred to OpenCOPI. This abstraction is a web service to wrapper the services provided by CT. This was necessary since the context services provided by CT are not Web services compliant. Moreover, it was necessary a driver to convert context data from CT context model to OpenCOPI context model needed to be developed. Moreover, BMDimensioner provides services related to discover possible regime configuration, where the regime is the relation between the length of pumping unit’s stem and the cycles per minute of this stem. Furthermore, BMDimensioner is responsible for acting in the operation of the pumping unit, e.g. changing its regime or stopping the unit. ChangeControlSystem stores and retrieves the changes made earlier in the pumping units. WifiLocalizationMiddleware and GPSLocalizationMiddleware are responsible for providing service about user localization. Each one has different QoC, which is used to select one of them to be executed in the workflow. HRDatabase is a system that provides information about employees and GSMPlatform provides a service to send short messages (SMS) to employees.

Figure 4 shows the case study semantic workflow. The execution starts in the first goal: subscribe to pumping unit burden. In case of the burden value between pumping unit’s intermediary burden value and maximum value, the workflow follows Flow1. In case of the burden value greater than the maximum value, the workflow follows the second flow. Flow1 encompasses activities to automatically change the regime of the pumping unit operation. First, a search is performed to find possible regimes, then another search is performed to find the regimes previously used in this pumping unit. The next step is to change the regime and update this information in the registry of changes provided by ChangeControlSystem. Finally, a search for available technicians in the vicinity of the oil well is performed and a message is sent to them so they can check if everything is running as expected. Flow2 describes the situation that burden is greater than the maximum limit of the pumping unit. To avoid the pumping unit damage, the operation of the oil well is stopped. After this, a search is performed to find the engineer responsible for this oil well and the technicians near the oil well. At last, messages are sent to them.
4. Related Work

This section describes some works directly related to OpenCOPI. According with the authors, the platform presented in [7] was the first platform to adopt workflows to perform Semantic Web service composition. It adopts an OWL context model. The lacks of this platform are: it has not a QoS/QoC and adaptation mechanism, service composition is not automatic and it has not a GUI to application development. [8] presents a workflow approach of modeling and managing the user’s interaction with the ubiquitous environment. In this approach, the users can determine their overall goal and preferences and the system generates a customized workflow describing how various services should interact with one another. This proposal lacks mechanisms to allow dynamic service composition and workflow adaptation. uMiddle [9] is a bridging mechanism that provides interoperability between devices over diverse middleware platforms and networks domains. uMiddle translates communication protocols and data types from middleware platforms protocols to uMiddle protocols. Applications built on top of uMiddle are free of platform dependencies and “can avail of all the devices from the various other platforms” [9]. GridKit [10] is a self-configuring and self-adapting middleware that can be deployed for applications that run on environments with a high degree of heterogeneity.

The aforementioned middleware platforms deal with a different subset of the UC application requirements. However, none of them simultaneously meet all these requirements. In particular, the proposals lack from a comprehensive solution to deal with interoperability issues at the middleware platform level. In the next Section we present OpenCOPI, an example of middleware platform that tackles such issues.

4. Conclusion

In this paper we presented OpenCOPI, a platform that integrates several context provision middleware and provides a unified context model for the development of ubiquitous applications. Regarding the context model, OpenCOPI uses ontology to represent context. Concerning the programming model, we consider that SOA provides a good metaphor for the type of interaction occurring in ubiquitous systems. So, OpenCOPI adopts a SOA based approach, decoupling applications from the underlying
context provision middleware while providing value added functionalities of
discovering, selecting and composing services that fulfill the application needs. We
believe that our approach can effectively contribute to the leverage of ubiquitous
computing, allowing that the full benefits of such environments can be widely spread.

References
1. Davidyuk, O., et al., MEDUSA: Middleware for End-User Composition of
Ubiquitous Applications. Handbook of Research on Ambient Intelligence and
Applications in Wireless Sensor Networks, in European Workshop on Wireless
Management, in International Conference on Information Management and
4. Buchholz, T., A. Küpper, and M. Schiffer. Quality of Context: What it is and
why we need it. in Workshop of the HP OpenView University Association. 2003.
Geneva, Switzerland.
5. Wang, X.H., et al., Ontology based context modeling and reasoning using OWL,
in Second IEEE Annual Conference on Pervasive Computing and
Communications Workshops2004: Orlando, USA. p. 18--22.
supporting the rapid prototyping of context-aware applications. Human-
Computer Interaction, 2001. 16(2).
7. Li, J., et al. FollowMe: On Research of Pluggable Infrastructure for Context-
Awareness. in 20th International Conference on Advanced Information
8. Ranganathan, A. and S. McFaddin. Using Workflows to coordinate Web Services
in Pervasive Computing Environments. in Proceedings of IEEE International
Pervasive Systems. in International Conference on Distributed Computing
Distributed Systems. in Proceedings of the Second International ICST