Dynamic and Semantic Web Services Composition for Ubiquitous Computing

Frederico Lopes[1], Thais Batista[2], Everton Cavalcante[2], Thiago Pereira[2], Flávia C. Delicato[3], Paulo F. Pires[3], Paulo Ferreira[4]

[1] School of Science and Technology, Federal University of Rio Grande do Norte – Natal, Brazil
[2] Department of Informatics and Applied Mathematics, Federal University of Rio Grande do Norte – Natal, Brazil
[3] Department of Computer Science, Federal University of Rio de Janeiro – Rio de Janeiro, Brazil
[4] INESC-ID, Technical University of Lisbon – Lisbon, Portugal

fred.lopes@gmail.com, thais@ufrnet.br, evertonranielly@gmail.com, thiago.inf@gmail.com, fdelicato@gmail.com, paulo.f.pires@gmail.com, paulo.ferreira@inesc-id.pt

ABSTRACT

The development of complex applications is one of the major current challenges of Ubiquitous Computing. Such applications are difficult to develop because they use services provided by different context-provision middleware, thus being necessary to know the models used for communication and representation of context information for each one. This paper presents the semantic composition and Web service selection mechanisms provided by OpenCOPI, a platform designed to integrate context-provision middleware. In OpenCOPI: (i) the applications are described in a high-level of abstraction by using semantic workflows; (ii) the semantic composition is responsible for composing concrete services from abstract descriptions of the applications, and; (iii) the selection is based on service quality metadata and user preferences. This paper also presents an evaluation of the semantic composition and the Web service selection mechanisms considering a ubiquitous application.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems – *distributed applications.*

General Terms

Algorithms, Design, Experimentation.

Keywords

Ubiquitous computing, platforms integration, context provision middleware, Semantic Web services composition.

1. INTRODUCTION

Ubiquitous Computing [1] scenarios typically consist of a myriad of integrated devices, sensors, and networks in order to build a highly heterogeneous environment. Such heterogeneity requires powerful middleware infrastructures to make easier the development of ubiquitous applications thus providing mechanisms for managing sensors, interpreting and dealing with context information, etc. [2]. The current trend in Ubiquitous Computing is the emergence of complex context-aware applications in which platforms are based on heterogeneous networks, interfaces, services, and technologies are used to provide user-centric applications. In such scenarios, a *service* is an element that has functionalities provided by an entity and consumed by other software entities (*clients*). Thus, ubiquitous applications are built through the *composition of services* provided by several service providers, so that many of them can provide services with the same functionality. Thus, the composition process may consider *Quality of Service* (QoS) parameters to select services that best meet the client's needs.

In this perspective, this paper presents OpenCOPI (*Open COntext Platform Integration*), a platform that integrates context-provision services and provides an environment to support and make easier the development of context-aware applications. OpenCOPI is based on SOA (*Service-Oriented Architecture*) [3] and its implementation uses the Web services technology, thus exploring open standards and languages. Such platform enables the integration of services provided by distinct service providers, thus ranging from services provided by simple ad hoc sensor networks to more complex services, such as context-provision middleware platforms. Another important OpenCOPI's feature is the use of *semantic workflows*. SOA and workflows management are promising technologies to meet Ubiquitous Computing requirements since SOA allows the applications functionalities to be provided and consumed as a set of *services*, which can be invoked by consumers through service interfaces published by service providers. On the other hand, a *workflow* describes the order of a set of activities to be executed by several services in order to achieve a specific goal [4]. SOA and workflows were selected since they are useful in environments in which lots of services provided by several providers are available and some of these services have similar functionality. Moreover, these technologies enable selection, composition, and automatic orchestration of services.

The model of Semantic Web service composition used by OpenCOPI is based on the service functionality and service metadata. This characteristic enables a better choice among the available services with the same functionality, for example, services with the same inputs, outputs, preconditions and effects. Such compositions must be performed by choosing services that provide context data that meet the application requirements. Thus, OpenCOPI uses

Quality of Service (QoS) and *Quality of Context* (QoC)[1] attributes. The literature presents many workflow-based platforms for Ubiquitous Computing [6, 7, 8, 9], but they do not provide service composition and selection mechanisms based on quality metadata. Moreover, such platforms have limited support for heterogeneity since they are not able to access different context-provision middleware in a transparent and uniform way.

In short, OpenCOPI provides: (i) an automatic Web service composition and selection mechanisms; (ii) a standardized communication model and an ontology-based context model, so that the applications must deal only with the abstractions provided by OpenCOPI, and; (iii) a fault-tolerance mechanism [10] that is able to deal with service failures. In this paper, we present and evaluate the semantic Web services composition and selection mechanisms provided by OpenCOPI. The service integration through composion is crucial to make easier the development of applications, thus enabling the use of services from several providers in a transparent and automatic way. Therefore, the applications become independent from the underlying context-provision platforms.

This paper is structured as follows. Section 2 describes a motivational example of ubiquitous application. Section 3 presents an overview about OpenCOPI. Section 4 presents the workflow specification, service compostion, selection, and execution processes. Section 5 discusses about the evaluation of the processes presented in Section 4. Section 5 presents related works. Finally, Section 7 presents final remarks.

2. MOTIVATIONAL EXAMPLE

In order to clarify the need of a platform like OpenCOPI, we use an ubiquitous application from the gas and oil industry as our motivational example. This application monitors an oil well in production through a pumping unit machine to detect the need of changing the pumping unit configuration. 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 actions to make changes in the configuration or directly notify the human responsible for taking decisions about the pump 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 this application, we monitor the *burden* of the pump unit, which denotes the charge of oil extracted from a well at each cycle of movement of the pump unit. Each pump unit has a specific maximum value (*maxBurden*) of burden for its correct operation. If this value is abruptly reached, then the pumping unit operation must be quickly stopped to prevent its damage (a reactive strategy). Furthermore, there is an intermediary value (*intermValue*), in which actions may be taken (in a proactive way) to prevent the pump unit to be stopped, consequently avoiding loss of production and risks to the equipment. Each one of these actions may be performed by more than one service and those services may be provided by different service providers.

The *OilExploration* context ontology (Figure 1) describes concepts regarding oil exploration environments. In the context ontology, these concepts are classified as *tasks* and *objects* related to the oil exploration domain. A *task–object* relationship denotes an *activity* that can be used by the applications. For instance, the *Subscribe* task is associated with the *Burden*, *BSW*, and *LimitOilFlow* objects. Thus, applications may include activities such as a subscription to receive notifications about the burden of a pump unit (*Subscribe, Burden*). Another example is the relationship between the *Send* task and the *SMS* and *Email* object.

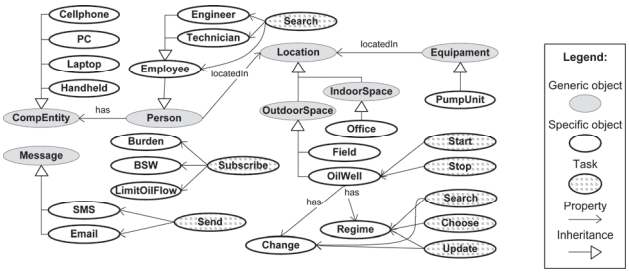

Figure 1. Description of the *OilExploration* domain ontology.

Such relationship means that it is possible to specify activities such as sending a SMS or an e-mail. Figure 2 shows the workflow that represents the motivational application in which each activity is performed at least by one service. The execution starts in the first activity, *SubscribeBurden*, which is related to a subscription to monitor the value of the pump unit burden. If the current burden value is between the pump unit's intermediary burden value and the maximum value, then the workflow follows *Flow1*. If the burden value is greater than the maximum value, then the workflow follows *Flow2*. *Flow1* encompasses activities to automatically change the *regime* (relation between the length of the pump unit's stem and its frequency in cycles per minute) of the pump unit. First, the *SearchRegimeOptions* activity looks for possible regimes of the pumping unit operation, in which each regime variable is composed of a stem length value and cycles per minute value. Then, the *searchPreviousChanges* activity finds the regimes previously used in this pumping unit. Next, the regime is changed (*ChangeRegime* activity) and this information is updated (*UpdateRegime* activity) in the registry of changes. Finally, it is performed a search for available technicians (*SearchTechnicians* activity) in the vicinity of the oil well and a message is sent to them (*SendMsgToEmployees* activity). Thus, they can check if ev-

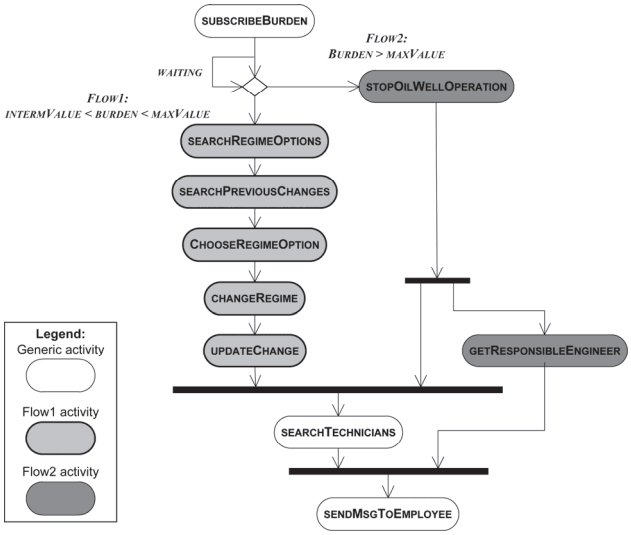

Figure 2. Workflow of the motivational example.

[1] QoC stands for any information that describes the quality of a context information, e.g. precision, probability of correctness, resolution, up-to-dateness, etc. [5].

erything is running as expected. In turn, *Flow2* describes the situation in which the burden is greater than the maximum limit of the pumping unit. To avoid damage to the pump unit, the operation of the well is stopped (*StopOilWellOperation* activity). Next, a search is performed to find the engineer that is responsible for this oil well (*GetResponsibleEngineer* activity) and the technicians near to the oil well (*SearchTechnicians* activity). Finally, warning messages are sent to them (*SendMsgToEmployees* activity).

Figure 3 presents the interaction among the application, OpenCOPI, and the underlying service providers (including context-provision middleware). *WellDatabase*[2] provides information about oil wells, being responsible for synchronously and asynchronously providing the current oil burden in each pump unit. This service performs the first workflow activity (*SubscribeBurden*). *BMDimensioner* is a platform that provides services to manage the configuration of the pump unit's operation regime. The services provided by this platform are responsible for presenting possible pump unit configurations and changing them (e.g. switching an operation regime to another one or stopping the pump unit operation). They are able to perform the *SearchRegime*, *ChangeRegime*, and *StopOilWellOperation* activities. *ChangeControlSystem* stores and retrieves configuration changes performed at the oil exploration equipment. In this case study, this system retrieves which regimes were already used in each pump unit. This system performs the *SearchPreviousChanges* and *UpdateChange* activities.

The *WifiLocalizationMiddleware*, *GPSLocalizationMiddleware*, and *CellularLocalizationMiddleware* plaforms are responsible for for providing the location of technicians spread over the oil field. Although they have the same functionality, each platform has a different quality level (QoS and QoC) and their quality will influence the selection of the service composition. Services provided by these platforms perform the *SearchTechnicians* activity. *HRDatabase* is a system that provides employees information, e.g. which employees are working at a given time. This system performs the *GetResponsibleEngineer* activity. Finally, *GSMPlatform* and *MailPlatform* provide services are used to send messages to employees. As well as location platforms, these platforms provide services with different quality level, thus performing the *SendMessageToEmployee* activity.

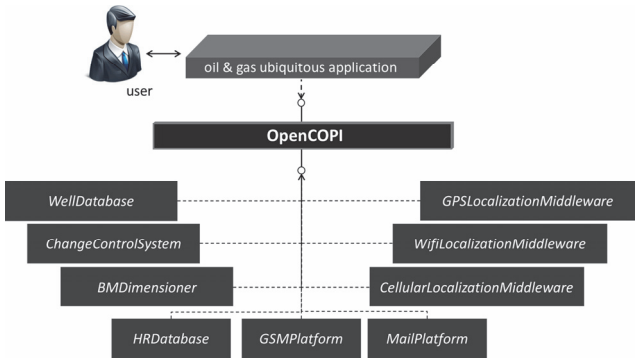

Figure 3. Service providers of the motivational example.

[2] *WellDatabase* abstracts a widget of *Context Toolkit* (CT) [11], a context-provision middleware. This widget monitors the pump unit operation and triggers events to OpenCOPI through a driver.

3. OPENCOPI: AN OVERVIEW

This Section presents OpenCOPI, a platform at the middleware level that integrates different service providers including, but not limited to, context-provision middleware platforms to make easier the task of developing context-aware adaptive ubiquitous applications. OpenCOPI enables the collaboration of different service providers to reach a high-level goal, which is to supply value-added services and contextual data to ubiquitous.

OpenCOPI provides its own communication model and an OWL [12] ontology-based context model, in which context is handled by adopting the Semantic Web services perspective [13]. Under this perspective: (i) service providers publish their services using the OWL-S language [14]; (ii) ubiquitous applications are services consumers; and (iii) OpenCOPI is a mediator that provides uniform acces to services used by ubiquitous applications. The adoption of an ontology-based context model leverages the interoperability among middleware platforms since ontologies enable knwoledgment sharing and prevent ambiguities [15]. Therefore, information received from other plaforms is translated to information that follows the ontology adopted by OpenCOPI, thus enabling applications only to know the unified context model.

OpenCOPI offers mechanisms for specifying and executing ubiquitous applications. Applications are specified through *semantic workflows*, which describe the abstract activities that represent the execution flow to reach the user goal, and each application has it own semantic workflow. A semantic workflow determines the sequence of abstract activities in terms of descriptions of semantic Web services. When specifying a semantic workflow, these activities are not linked to the services that would perform them since a semantic workflow is independent of specific concrete services, so that the link is made at the service composition phase. This approach of separating abstract activities from the concrete services able to perform them is useful mainly in cases in which there are several available similar services that are provided 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 activity.

Since a semantic workflow is an abstract representation of the execution flow, it is necessary to create at least one concrete specification for the workflow, which is called *execution plan* and contains a set of concrete, orchestrated Web services. Execution plans 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. In fact, when a semantic workflow is executed, it is performed a search for Web services which are compatible with each workflow activity, i.e. services which have the same inputs, preconditions, outputs, and effects of the activity. If there is no service that performs the activity, the OpenCOPI's composition mechanism tries to compose a set of services to perform the activity. Thus, the discovered and composed services are combined in order to build one or more execution plans. The set of generated execution plans is represented as a directed acyclic graph (DAG) in which each intermediary node represents a specific service anc each directed edge represents the execution direction between two services. Each DAG starts with an *initial node* and a *final node*, which are nodes that not represent services, but are used to indicate the beginning and the end of the DAG. Thus, each complete path between the *initial node* and the *final node* represents a possible execution plans in the workflow, so that the DAG represents the workflow with all possible execution plans.

Figure 4 shows a DAG that represents eight possible execution plans regarding the workflow presented in Figure 2. It is possible to create these eight execution plans since two workflow activities are performed by more than one service. For example, the *SearchTechnicians* activity can be performed by the services provided by *GPSLocalizationMiddleware, WifiLocalizationMiddleware* and *CellularLocalizationMiddleware*, and the *SendMsgToEmployee* activity can be performed by *GSMPlatform* and *MailPlatform*. Therefore, the combination of these services can create six execution plans. In order to increase the number of execution plans to enable a better analysis of the selection process, we created a replica of the services provided by *GPSLocalizationMiddleware*. The replica and the original service have distinct quality metadada. Using this replica, there are eight different execution plans created by combining four localization services (*GPS* and *GPS2*, *Wifi* and *Cellular*, represented in Figure 4 by the *S7*, *S7'*, *S7''* and *S7'''*) and two messaging services (*SMS* and *Email*, represented in Figure 4 by *S8* and *S8'*). For example, one of the execution plans represented in Figure 4 is the execution plan that includes the services from *SubscribeBurdenAssync* (*S1*) to *UpdateChange* (*S6*), besides the GPS-based localization service (*S7* or *S7'*) and the SMS-based messaging service (*S8*). Due to space restrictions, we omit the intermediary services, i.e. the services between the *SubscribeBurdenAssync* (*S1*) and *UpdateChange* (*S6*) services.

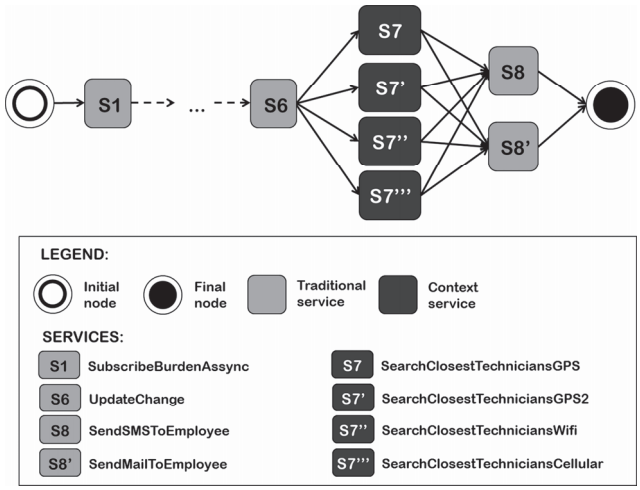

Figure 4. DAG representing the execution plans for the workflow depicted in Figure 2.

Since more than one execution plan is created, one of them must be selected. This selection is based on quality parameters regarding these services (QoS and QoC, for context services). Therefore, it is possible to select the service with the best QoS and QoC parameters among the several available services that provide the same functionality.

3.1 Architecture

OpenCOPI architecture encompasses two layers (Figure 5), namely *ServiceLayer* and *UnderlayIntegrationLayer*. *ServiceLayer* is responsible for managing abstractions of services (OWL-S descriptions) supplied by service providers. The components of *ServiceLayer* use such abstractions to support workflow creation and execution, service selection, service composition and adaptation. In addition, they support context reasoning, context storage, among other functionalities related to ubiquitous applications. These applications should implement the *IApp* interface to communicate with the OpenCOPI's *AppFacade*, which is responsible for receiving requests from the applications and forwarding them to the components of *ServiceLayer*. In turn, *UnderlayIntegrationLayer* is responsible for integrating service providers (mainly, but not only context-provision middleware), thus performing context conversion whenever it is necessary (from the middleware context model to the OpenCOPI one) and communication protocol conversion. The *IUnderlayIntegration* interface links service providers and OpenCOPI's *UnderlayIntegrationLayer*. This paper presents just the components related to the service composition and selection processes.

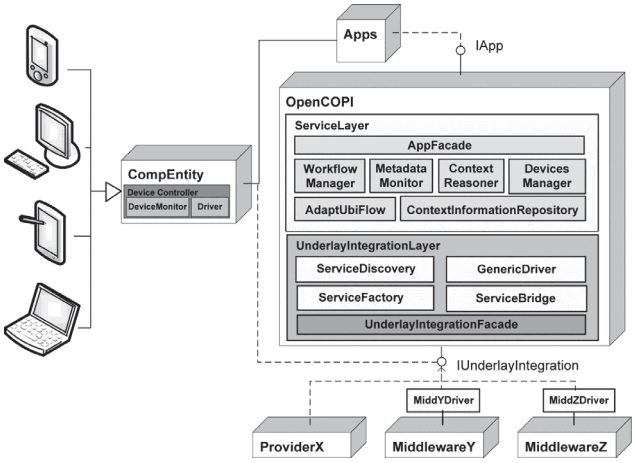

Figure 5. OpenCOPI architecture.

The *WorkflowManager* component manages workflows and is composed of the four following (sub)components, which provide support for specifying semantic workflows and generating execution plans. The *ServiceManager* component is responsing for importing OWL-S descriptions from service providers to OpenCOPI and validating such descriptions, and for proving capabilities to search for basic concepts in the knowledge base (ontology) using inputs, outputs, preconditions and effects of the available semantic Web services. *SemanticComposer* is responsible for discovering and composing Web services according to the semantc workflow specifications, i.e. it makes the mapping between abstract activities and Web services. First, it tries to discover the services (among those available at the *SemanticServicesRepository*) that can be used to compose the execution plan, given the goals in the workflow specification. Next, it tries to combine the discovered services in order to consume all inputs and preconditions and to produce all outputs and effects in the workflow specification. Then, 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* component stores both the ontologies that describe Web services and execution plans. Finally, the *WorkflowExecutor* component supports the workflow execution. It receives the execution plan selected by the *AdaptUbiFlow* component considering QoS and QoC parameters of the services that compose each execution plan, as detailed in Section 4. At runtime, *WorkflowExecutor* executes the selected execution plan by making calls to the services provided by the underlying context-provision middleware.

MetadataMonitor is responsible for gathering metadata about the services and context provided by underlying middleware to feed the *ContextInformationRepository*. OpenCOPI adopts an SLA (*Service-Level Agreements*) [16] approach in which service providers publish the quality metadata of their services and these

metadata are used to select the services to be provided to the consumers. If these metadata are not published, *MetadataMonitor* tries to assess them by invoking the services provided by underlying middleware. Details about this process are out of scope of this work.

AdaptUbiFlow component [12] is responsible for the selection and adaptation processes in OpenCOPI. After creating execution plans, *AdaptUbiFlow* selects the best execution plan (with the highest quality) to be executed. In addition, *AdaptUbiFlow* is responsible for performing an *adaptation process* in the workflow execution, thus meaning to replace the previously selected execution plan for another one, in which this new plan must meet the goals of applications specified in the workflow. This component directly works with the *MetadataMonitor* and *WorkflowManager* components to identify a fault and automatically change the execution flow in order to use another execution plan. Therefore, *WorkflowManager* and *AdaptUbiFlow* components are responsible for supporting the workflow specification and the composition, selection and execution processes.

4. FROM THE WORKFLOW SPECIFICATION TO ITS EXECUTION

This section presents the activities performed from the semantic workflow specification until its execution, also encompassing the composition, selection, and execution processes, as described in the following subsections.

4.1 Semantic workflow specification

OpenCOPI enables users to build semantic workflow even they are not familiar with the vocabulary of the process to be modelled. By using an assistant provided by OpenCOPI, the user can select the activities used for describing the application and identify the desired inputs, outputs, preconditions and effects (IOPEs) for each activity. The assistant also enables the user to specify some preferences, e.g. weights assigned to quality parameters at the selection phase.

The semantic workflow specification consists of describing the activities of a business process through the combination between a *task* and an *object* (classes in the context model). The assistant enables the user to select these activies without the need of creating each of them since they are already pre-defined according to the set of *tasks* and *objects* [17] specified in the context ontology. These activities are abstract, thus meaning that the binding to the concrete services that will perform the activity is done at runtime. The assistant shows a list of possible *tasks* and a list of possible *objects*, from which the user can select a <*task*, *object*> tuple (e.g. <*Choose*, *RegimeOption*>, <*Send*, *Message*>) to describe each activity of a semantic workflow. The assistant also shows the list of possible IOPEs related to each specified activity and that should be added to the activity definition. Thus, the user describes only the abstract activities of the workflow, so that when executing the workflow, OpenCOPI performs inferences on the ontologies to discover which services perform each workflow activity and composes the possible execution plans with these discovered services.

Figure 6 shows the process of creating workflow activities. The user requests, via OpenCOPI's GUI, the creation of a new activity. The request is received by the *WorkflowManager* component, which is responsible for creating and editing the workflow. This component asks the *ServiceManager* component for the list of possible tasks. Then, *ServiceManager* recovers the concepts described in the domain ontology stored in the *SemanticServiceRepository* component by loading all concepts in memory and returning the list of tasks and respective objects (objects related with each task) to the *WorkflowManager* component. Next, this component shows these concepts for the user, which selects the desired task and the object (e.g. the *Subscribe* task and the *Burden* object), so that this tuple is sent to *WorkflowManager*. Then, *WorkflowManager* asks the *ServiceManager* component for the list of respective IOPEs of this activity. With the concepts in memory, the *ServiceManager* returns the list of possible IOPEs for the *WorkflowManager*, which forwards this list to the user. Then, the user selects which IOPEs should be considered for this activity, thus finishing the process of including an activity. This process is repeated for each activity of the workflow.

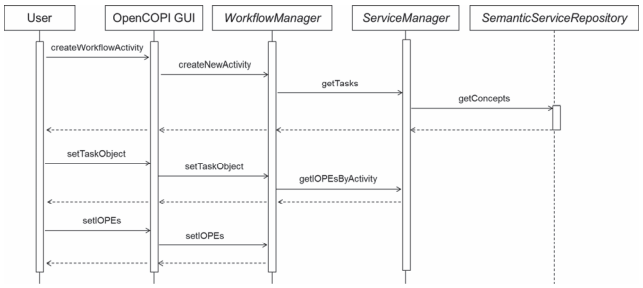

Figure 6. Process of creating a workflow activity.

OpenCOPI assistant also allows the use of flow control connectors (condition, repetition, etc.) in the semantic workflow specification. Figure 7 presents the assistant screenshot in which three activities have already been added to the workflow and the activity *Choose_RegimeOption* composed by the *Choose* task and the *RegimeOption* object is being added. Figure 7 also presents a screenshot showing that the user selected two inputs (*RegimeList* and *ChangeList*) and one output (*Regime*) for the current activity.

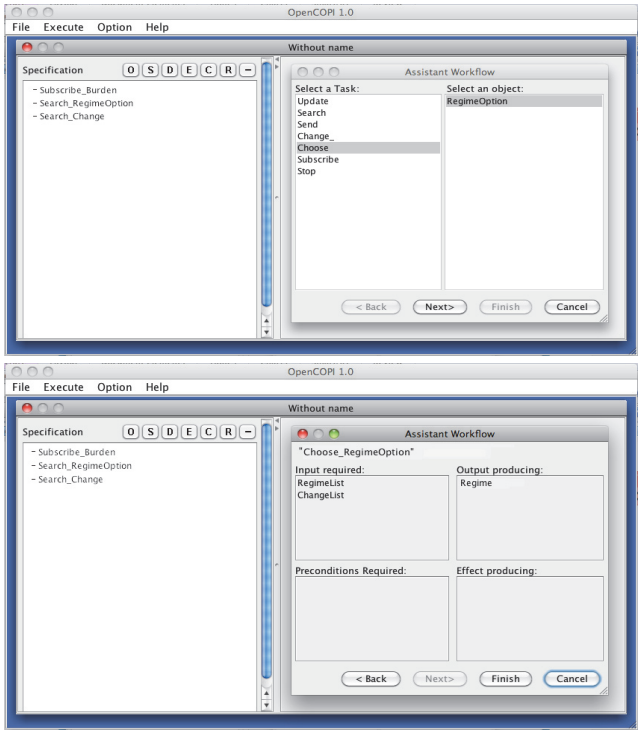

Figure 7. Assistant for creating a workflow.

4.2 Composition: Creating execution plans

Once the workflow is completed, the next step is the creation of the execution plans by the *SemanticComposer* component. First, this component discovers which registered services can be used to satisfy the activity's IOPEs. Next, it matches the selected services in order to consume inputs, produce outputs, and meets preconditions and effects of each workflow activity, so that a workflow activity may be performed by a service composed of atomic services as well as a single service may perform more than one activity. Finally, *SemanticComposer* orders the services of each created execution plan by following the sequence of activities defined in the workflow specification (service orchestration). OpenCOPI uses the matching algorithm presented by Mendes et al. [18] that composes Web services from a semantic workflow specification taking into account the required inputs and pre-conditions and the produced outputs and effects regarding each activity. With these lists, OpenCOPI generates the possible execution plans. In the presented case study, each one of these execution plans was naes as *EP1*, *EP2*, …, *EP8* to facilitate the explanation of the evaluation. Table 1 shows the services that are different in the execution plans. The services that perform the other activities are the same for all execution plans since there is only one available service for each of these activities.

Table 1. Execution plans for the motivational example.

EPs	Distinct services in the execution plans
EP1	*SearchTechnicians**GPS**, Send**SMS**ToEmployee*
EP2	*SearchTechnicians**GPS2**, Send**SMS**ToEmployee*
EP3	*SearchTechnicians**GPS**, Send**Mail**ToEmployee*
EP4	*SearchTechnicians**GPS2**, Send**Mail**ToEmployee*
EP5	*SearchTechnicians**Wifi**, Send**SMS**ToEmployee*
EP6	*SearchTechnicians**Wifi**, Send**Mail**ToEmployee*
EP7	*SearchTechnicians**Cellular**, Send**SMS**ToEmployee*
EP8	*SearchTechnicians**Cellular**, Send**Mail**ToEmployee*

4.3 Execution plan selection

OpenCOPI selects execution plans according to quality metadata (QoS and/or QoC parameters) regarding the services that compose the execution plan. Although we have defined a built-in set of parameters, new parameters can be added to this set and others can be removed from it. OpenCOPI considers the following QoS parameters: (i) *Response time*, which represents the time between sending a service request to a service provider and receiving the response; (ii) *Availability*, which represents the probability that the service provider is up and the service is running, i.e. if the service is ready for immediate use, and; (iii) *Performance*, which describes the number of service requests fulfilled by the service provider at a given period of time. Moreover, OpenCOPI considers the following QoC parameters, as proposed by Buchholz et al. [6]: (i) *Correctness*, which denotes the probability that a piece of context information is correct (since internal problems in a temperature sensor can generate wrong temperature values, for example), and; (ii) *Freshness*, which represents the time elapsed between the generation of context data and the retrieval of this data by applications, so that a particular context data may be prioritized if is is newer than other similar data.

4.4 Aggreggation functions

The process of selecting an execution plan begins with the calculation of the quality of each plan. The quality of an execution plan is determined by quality (QoS and QoC) parameters values of all services contained in the execution plan. Before computing the execution plan quality, it is necessary to compute the *global quality* of each quality parameter. Global quality of a parameter means the quality parameter value for the whole execution plan, i.e. the value that represents the parameter of all services of this execution plan. The global quality of each parameter can be computed by aggregating the corresponding values for such parameter of all services in the respective execution plans. Different aggregation functions [19, 20] are necessary to compute the global value of each parameter. Typical quality parameter aggregation functions are addition, multiplication, minimum, and average relation. For instance, *Response time* is the QoS parameter used to measure the response time to execute each service. Thus, the value of the *Response time* parameter for an execution plan *EP* ($q_{EP(R)}$) is the sum of the *Response time* values of all services ($q_{R(s)}$) that compose *EP*. The *Availability* QoS parameter can be aggregated ($q_{EP(A)}$) through a multiplication function of the availability value of each service of the execution plan *EP* ($q_{A(s)}$). The *Performance* QoS parameter describes the number of service requests served by the service provider at a given period of time. Thus, the performance of an execution plan *EP* ($q_{EP(P)}$) is limited to the service with the smaller value for *Performance* attribute. *Freshness* QoC parameter describes the context information life span, i.e. how long time ago the context information was created. Thus, the value of this QoC parameter for an execution plan *EP* ($q_{F(s)}$) is the average of context life span of all services of *EP*. Similar to the *Availability* parameter, the aggregation function for the *Correctness* parameter is also a multiplication; however it refers to the correctness level of context information provided by the execution plan's services ($q_{C(s)}$).

4.5 Normalization of quality parameters

Once the value of all global (or aggregated) quality parameters were calculated, and considering that different quality parameters have different units and ranges, it is necessary to *normalize* these attributes into the same range to allow a unified and uniform measurement of the quality of the execution plans. Some quality parameters could be *positive*, i.e., a parameter in which the quality is better if the value is greater (for example, *Correctness* parameter). Other parameters are *negative*, i.e. the quality is better if the quality value is smaller (for example, the *Response time* parameter)[3]. This quality parameters normalization process is used by several authors [19, 20, 21].

Equations 1 and 2 present the formula used to normalize positive and negative parameters, respectively. In these equations, q_{Ni} is the normalized value of the parameter i, q_i is the global value of this parameter for the current execution plan, and q_{max} and q_{min} are respectively the minimum and the maximum global values of this parameter for all considered execution plans (if $q_{max} = q_{min}$, then $q_{Ni} = 1$). In this process, each normalized value results in a value between 0 and 1.

$$(1) \qquad q_{Ni} = \frac{q_i - q_{max}}{q_{max} - q_{min}}, q_{max} \neq q_{min}$$

$$(2) \qquad q_{Ni} = \frac{q_{max} - q_i}{q_{max} - q_{min}}, q_{max} \neq q_{min}$$

4.6 Utility of execution plans

The normalization process is followed by a weighting process to consider the user priority and preferences. Thus, users can prioritize some quality parameters and minimize the importance of

[3] We **are not** saying that the values are negative.

other quality parameters according to their needs. To do this, the user assigns a weight w_i to each quality parameter i, which must be between 0 and 1 and the sum of all of these weights must be 1. Equation 3 presents the function to calculate the execution plan quality (q_{EP}) according to a set of QoS and QoC parameters. In this formula, q_{EP} is the quality of the execution plan and it is calculated through a weighted sum between these normalized values (q_{Ni}) according each quality parameter i ($1 \leq i \leq m$) and its respective weight (w_i).

$$q_{EP} = \sum_{i=1}^{m} (q_{Ni} * w_i) \quad (3)$$

At the selection phase, the *utility* of each execution plan is just the quality of the respective execution plan, so that the exection plan with the highest quality is selected.

4.7 Execution

After selecting an execution plan, it is started. Executing a plan means invoking each service that compose it according to the execution process defined in the workflow that have originated this selected execution plan. Thus, OpenCOPI orchestrates the invocation of each service.

5. EVALUATION

The evaluation presented in this paper is twofold. The first evaluated aspect is the execution plan *selection*, which is based on QoS and QoC parameters of each service that compose each execution plan. The second evaluated aspect is the overhead generated by OpenCOPI. This overhead is natural since OpenCOPI introduces an abstraction layer between middleware platforms and applications. This evaluation uses the motivational example presented in Section 2.

5.1 Evaluation of execution plan selection

The evaluation was performed in order to confirm the selection of the best execution plan according to different configurations in terms of prioritizing the weights regarding the quality parameters. Two or more execution plans within the same workflow may differ from each other only in a few services, so that the calculations in the selection process are really influenced by the services that differentiate an execution plan from another one. For instance, an execution plan *EP1* may differ from another execution plan *EP2* only in terms of one service, which will make a substantial difference in the final quality of these execution plans. Therefre, the quality (QoS/QoC) metadata regarding the services were populated with ficticious values (see Table 2 and Table 3) in order to verify which execution plan would be selected for each tested prioritization configuration. For this evaluation, the motivational example presented in Section 2 contains eight possible execution plans (each one named as *EP1*, *EP2*, …, *EP8*), which differ from each other in terms of combinations of four possible localization services (*Wifi*, *Cellular*, *GPS* e *GPS2*[4]) and two messaging services (*SMS* and *Email*).

The execution plans were submitted to the selection algorithm considering the metadata of each service. After calculating the global value of each parameter for each one of the eight execution plans, these values were normalized, so that all parameters are represented in an adimentional way. Table 4 shows the utility of the execution plans for different prioritization configurations for the quality parameters. For instance, the execution plan *EP1* is selected when all parameters have the same prioritization, while the execution plan *EP4* is selected when only the *Availability* parameter is prioritized. The execution plan *EP5* is selected when only the *Response time* parameter is prioritized, and the execution plan *EP7* is selected when the parameters *Availability* and *Response time* have the same prioritization (0.5). The values highlighted in Table 4 refer to the highest utility for each tested configuration. The selection process was performed as expected, *always* selecting the best execution plan.

Table 2. QoS metadata for traditional and context services.

Services	*Availability*	*Performance*	*Response time*
SubscribeBurdenAssync	2	1	3
SearchRegimeOptions	2	1	3
SearchPreviousChanges	2	1	3
ChooseRegimeOptions	3	1	1
ChangeRegime	2	1	3
UpdateChange	2	1	3
SearchTechniciansGPS	9	8	4
SearchTechniciansGPS2	9	6	4
SearchTechniciansCellular	4	2	7
SearchTechniciansWifi	8	3	6
SendSMSToEmployees	7	7	9
SendMailToEmployees	9	5	7

Table 3. QoC metadata for context services.

Services	*Correctness*	*Freshness*
SubscribeBurdenAssync	5	2
SearchTechniciansGPS	10	2
SearchTechniciansGPS2	9	2
SearchTechniciansCellular	7	6
SearchTechniciansWifi	5	5

Table 4. Utility of execution plans (EPs) for each configuration.

EPs	Same priority (w = 0.2)	*Availability* (w = 1.0)	*Responsing* (w = 1.0)	*Availability*/ *Responsing* (w = 0.5)
EP1	0,785	0,528	0,400	0,464
EP2	0,772	0,660	0,000	0,530
EP3	0,766	0,830	0,400	0,415
EP4	0,760	1,000	0,000	0,500
EP5	0,515	0,528	0,800	0,664
EP6	0,496	0,830	0,400	0,615
EP7	0,480	0,000	1,000	0,500
EP8	0,430	0,150	0,600	0,375

5.2 Overhead in OpenCOPI

The overhead generated by OpenCOPI was also evaluated under three aspects: (i) composition; (ii) selection, and; (iii) execution. Some overhead was expected since an additional abstraction layer was inserted between ubiquitous applications and services provided by several underlying platforms, thus increasing the response time to the users. Therefore, it is important to measure the impact of OpenCOPI in the application execution time. Furthermore, it is important to relate this overhead time with the benefits propitiated by OpenCOPI in terms of implementation and execution of ubiquitous applications.

In order to evaluate the overhead of the composition and selection processes, it was considered a single workflow and the amount of possible execution plans for this workflow was varied in eight,

[4] *GPS2* is a replica of the GPS-based location service (*GPS*). However, these services have distinct quality metadata.

six, four and two execution plans by considering all available services in the motivational example. To decrease from eight to six execution plans, the *SearchTechniciansCellular* service was removed from the OpenCOPI service repository since this service performs the *SearchTechnicians* activity in two execution plans. This process was repeated to the *SearchTechniciansWifi* and *SearchTechniciansGPS2* services in order to have four and two available execution plans. The conducted experiments were carried out on Mac OS X operating system, using a computer with processor Intel® Core™ 2 Duo 2.4 GHz and 4 GB of RAM.

Figure 8 illustrates the average overhead (in miliseconds with a confidence interval of 95%) regarding the composition (Figure 8.A) and selection (Figure 8.B) processes. As the composition mechanism discovers services to perform each workflow activity and then creates the possible execution plans, this process is more computationally expensive since it requires analyzing the ontologies of each available service in the service repository. However, we can observe in the results that the time spent for the composition process is not relevant since it transforms an abstract specification (semantic workflow) in compositions of concrete services (execution plans), thus enabling a dynamic service selection and decoupling the applications form the used services. For the simplest configuration (two possible execution plans), the time spent in the semantic composition was 1008 ms on average to build both execution plans, and for the most complex configuration (eight possible execution plans) it was 1286 ms on average. The process of selecting an execution plan is cheaper. For several executions for each configuration and considering a confidence interval of 95%, the average times spent for the simplest and the more complex configurations were 2 ms and 6 ms, respectively.

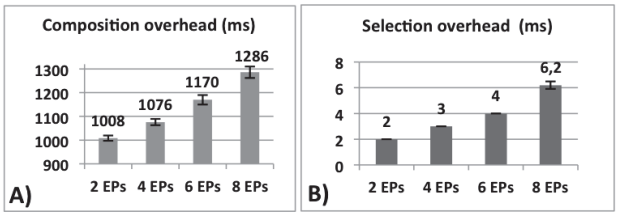

Figure 8. Overhead of the composition and selection processes.

Another aspect analyzed in the evaluation was the application execution time. Since OpenCOPI represents an additional layer between ubiquitous applications and services provided by many platforms, it is expected that the use of OpenCOPI increases the overall processing time and therefore the response time for users. Thus, it is important to measure the impact of OpenCOPI in the application execution time and if this impact is either significant or negligible from the point of view of the user's experience. For this purpose, two versions of the case study were built. The first one considered the specification and execution of a workflow using OpenCOPI, while the second one directly invokes the same services through Java source code in the same sequence followed by the workflow executed by OpenCOPI. The average execution time of the application without OpenCOPI was 1.2 s to call all services involved in the case study. In the application that uses OpenCOPI, the execution time increased to 1.9 s. According to Nah [22], the difference (about 0.7 s) between these two versions was not significant for this kind of application. Moreover, the second version does not take advantage on the benefits provided by OpenCOPI. For instance, to build the application without OpenCOPI, 139 lines of code (LOC) were necessary only to call all services specified in the workflow. However, the process to build the workflow using OpenCOPI is simpler since it was not necessary to implement source code but only to build the workflow by defining the applications activities, combining *tasks* and *objects* to satisfy the application goal. Moreover, without OpenCOPI, it is essential to know the services available in the environment and their interfaces. Consequently, the development is harder, reuse is laborious, and it is difficult to dynamically select services and also to support adaptation

6. RELATED WORKS

Several workflow-based middleware platforms have emerged over the last years to assist the development of ubiquitous applications. However, most of them do not meet the wide range of requirements demanded by highly dynamic and heterogeneous ubiquitous environments. In general, these proposals do not allow dynamic and semantic service composition based on several quality metadata parameters.

Ranganathan and McFaddin [6] present a workflow approach for modelling and managing the user interaction with the ubiquitous environment. In such approach, users can determine their overall goal and preferences and the system generates a customized workflow describing how various services should interact with one another. Montagut and Molva [7] present an architecture that supports the distributed execution of workflows in pervasive environments based on decentralized control. Both proposals lack mechanisms to allow dynamic service composition and workflow adaptation.

Tang et al. [8] present a context-adaptive workflow management algorithm, which can dynamically adjust workflow execution policies in terms of current context information and supports service selection based in bandwidth and user location. Finnaly, Mostarda et al. [9] present a workflow management approach that matches orchestration and choreography in ubiquitous environments. However, this approach does not consider metadata quality when selecting the services that will compse the workflow, thus hampering the selection of services with better quality.

Unlike all the previously mentioned proposals, this paper investigates how to automatically manage workflows by semantically composing execution plans from abstract descriptions of the application activities and then automatically selecting one of the possible execution plans, which may be composed of services provided by several context-providion middleware. In OpenCOPI, this selection process is based on several quality parameters, and each one of them has its weight defined by the user.

7. FINAL REMARKS

This paper presented the service composition and selection process performed in OpenCOPI to make easier the development of ubiquitous applications and to enable the use of services provided by several context-provision middleware in a transparent and automatic way. We presented the workflow management process by detailing the activities performed from the semantic workflow specification until its execution, also encompassing the composition, selection, and execution processes. In addition, we conducted computational experiments to evaluate OpenCOPI under two distinct aspects: (i) selection of execution plans to show that the best execution plan was selected in all tested configurations in terms of weights regarding the quality parameters, and; (ii) the overhead introduced by OpenCOPI: this overhead is natural since OpenCOPI represents an additional abstraction layer between ubiquitous computing and underlying context-provision middleware. First, the selection mechanism was performed as expected always selecting the best exectution plan. Second, the time spent

in the service composition and execution plan selection processes were very low. Moreover, the difference (0.7 s) between the application executing with OpenCOPI and the application developed using the Java programming language was not significant compared with the benefits propritiated by OpenCOPI since delays in the order of 1 s are acceptable for Web applications, as stated by Nah [22]. Finally, without OpenCOPI, it is essential to know the available services and their interfaces, thus making the development more complex and the reuse impraticable.

The execution plan selection process presented in Section 4 considers all services that compose all execution plans. Therefore, the selection overhead considerably increases when there is a high number of services since it is necessary to combine the services and then calculate all aggregated values for each quality parameter for each execution plan. In this perspective, we are currently developing a strategy that aims to reduce the number of services to be considered in the selection processing, thus making the aggregations faster. Such strategy consists of desconsidering services that would be in all execution plans and thus equally contribute in the aggregation calculations. Once this new selection algorithm is implemented, we will carry out new computational experiments comparing the performance of both algorithms.

8. ACKNOWLEDGMENTS

This work is partially supported by the Brazilian National Institute for Web Science Research (Web Science Brazil), founded by the Brazilian National Council for Scientific and Technological Development (CNPq) through the grant 557128/2009-9. Thais Batista, Flavia C. Delicato, and Paulo F. Pires are partially supported by CNPq through the grants 307269/2012-8, 311363/2011-3, and 311515/2009-6 (respectively). Flavia C. Delicato is also partially supported by CNPq through the grant 470586/2011-7.

9. REFERENCES

[1] Weiser, M. 1991. The computer of the Twenty-First Century. *Scientific American* 265, 3, 94-104.

[2] Judd, G., Steenkiste, P. 2003. Providing contextual information to Pervasive Computing applications. *Proc. of the First Int. Conf. on Pervasive Computing and Communications*. PERCOM'03. IEEE Computer Society, USA, 133-142.

[3] Erl, T. 2007. *SOA principles of service design.* Prentice Hall, USA.

[4] Abbasi, A., Shaikh, Z. 2009. A conceptual framework for smart workflow management. *Proc. of the Int. Conf. on Information Management and Engineering.* IEEE Computer Society, USA, 574-578.

[5] Buchholz, T., Küpper, A., Schiffers, M. 2003. Quality of Context: What it is and why we need it. *Proc. of the 10th Workshop of the HP OpenView University Association*.

[6] Ranganathan, A., McFaddin, S. 2004. Using workflows to coordinate Web services in Pervasive Computing environments. *Proc. of the IEEE Int. Conf. on Web Services*. IEEE Computer Society, USA, 288-295.

[7] Montagut, F., Molva, R. 2005. Enabling pervasive execution workflows. *Proc. of the Int. Conf. on Collaborative Computing: Network, Applications and Worksharing*. IEEE Computer Society, USA.

[8] Tang, F. et al. 2008. Towards context-aware workflow management for Ubiquitous Computing. *Proc. of the Int. Conf. on Embedded Software and Systems*. IEEE Computer Society, USA, 221-228.

[9] Mostarda, L., Marinovic, S., Dulay, N. 2010. Distributed orchestration of pervasive services. *Proc. of the 24th IEEE Int. Conf. on Advanced Inf. Networking and Applications*. IEEE Computer Society, USA, 166-173.

[10] Lopes, F. et al. 2011. AdaptUbiFlow: Selection and adaptation in workflows for Ubiquitous Computing. *Proc. of the 9th IEEE/IFIP Int. Conf. on Embedded and Ubiquitous Computing*. EUC 2011. IEEE Computer Society, USA, 63-71.

[11] Dey, A., Abowd, G., Salber, D. 2001. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human Computer Interaction* 16, 2, 97-166.

[12] OWL 2 Web Ontology Language. 2009. W3C. Available at: http://www.w3.org/TR/owl-overview/

[13] Martin, D. et al. 2004. Bringing semantics to Web services: The OWL-S approach. Proc. of the First International Workshop on Semantic Web Services and Web Process Composition. SWSWPC 2004. In *Lecture Notes in Computer Science*, J. Cardoso, A. Sheth, Eds. Springer-Verlag Berlin Heidelberg, Germany, 26-42.

[14] OWL-S: Semantic Markup for Web Services. W3C. Available et: http://www.w3.org/Submission/OWL-S/

[15] Gruber, T. 1993. A translation approach to portable ontology specifications. *Journal of Knowledge Acquisition* 5, 2, 199-220.

[16] Keller, A., Ludwig, H. 2003. The WSLA Framework: Specifying and monitoring Service-Level Agreements for Web Services. *Journal of Network and Systems Management* 11, 1, 57-81.

[17] Guarino, N. 1998. Formal ontology and information systems. *Proc. of the Int. Conf. on Formal Ontology and Information Systems*. FOIS'98. 3-15.

[18] Mendes, R. et al. 2011. Using Semantic Web to build and execute ad-hoc processes. *Proc. of the 9th IEEE/ACS Int. Conf. on Computer Systems and Applications*. AICCSA 2011. IEEE Computer Society, USA, 233-240.

[19] Alrifai, M., Skoutas, D., Risse, T. 2010. Selecting skyline services for QoS-based Web service composition. *Proc. of the 19th Int. Conf. on World Wide Web*. ACM, USA, 11-20.

[20] Yanwei, Z. et al. 2010. A dynamic Web services selection based on decomposition of global QoS constraints. *Proc. of the IEEE Youth Conference on Information Computing and Telecommunications*. IEEE Computer Society, USA, 77-80.

[21] Ardagna, D., Mirandola, R. 2010. Per-flow optimal service selection for Web services based process. *Journal of Systems and Software* 83, 8, 1512-1523.

[22] Nah, F. 2004. A study on tolerable waiting time: How long are Web users willing to wait. *Behavior and Information Technology* 23, 3, 153-16.