AdaptUbiFlow: Selection and Adaptation in Workflows for Ubiquitous Computing

Frederico Lopes, Thiago Pereira, Everton Cavalcante, Thais Batista
DIMAp – Federal University of Rio Grande do Norte
Natal, RN, Brazil
{fred.lopes, evertonranielly, thiagosilva.inf}@gmail.com, thais@ufrnet.br

Flavia C. Delicato, Paulo F. Pires
NCE – Federal University of Rio de Janeiro
Rio de Janeiro, RJ, Brazil
{fdelicato, paulo.f.pires}@gmail.com

Paulo Ferreira
INESC-ID – Technical University of Lisbon
Lisbon, Portugal
paulo.ferreira@inesc-id.pt

***Abstract*—Ubiquitous environments still suffer from low availability given that any device may fail and it is hard to replace a failed element. In this paper we present AdaptUbiFlow (*Adaptive Ubiquitous Workflow*), an OpenCOPI´s element that aims to increase the availability of an ubiquitous system. When a device fails, AdaptUbiFlow supports the automatic reconfiguration of the system replacing the failed device (or service) by an equivalent one; this makes the system fault-tolerant without the need of any manual intervention. The replacing of a device/service is chosen taking into account not only the QoS and QoC (Quality of Context) provided but also the application's execution flow to ensure that the best adaptation option will be chosen. AdaptUbiFlow evaluation showed encouraging results**

***Keywords*—Ubiquitous computing, platform integration, context provision middleware, service selection, adaptation, semantic workflows.**

I. INTRODUCTION

The current trend in ubiquitous computing is the emergence of complex context-aware applications in which platforms based on heterogeneous networks technologies are used to provide user-centric applications. The user-centric perspective means that ubiquitous environments 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 [2]. Thus, ubiquitous applications are built through a process of service composition, where those services are available from various service providers. In such scenario, it is common that different providers offer services with the same functionality. In such situations, the composition process can consider the Quality of Service (QoS) and Quality of Context (QoC) to decide which services are the best ones to be selected to fulfill the client needs. QoC denotes "any data that describes the quality of information that is used as context, for instance, precision, probability of correctness, resolution, up-to-dateness, etc" [3].

Unlike services which are hosted by high-end servers and data centers, where service failures are rare, services in a ubiquitous computing system need to embrace service failures as the normal case [4]. Thus, there is a need of an adaptation process to handle failures in order to avoid the interruption of the execution. Moreover, an adaptation process can be started in case services with better QoS become available in the environment or when the QoS of a service used in a composition degrades. Adaptation process also can be started in case of user mobility. Consequently, a ubiquitous platform must consider both requirements (used to specify the initial composition of services to meet an application request and the adaptation of existing composite services) at runtime [5].

The goal of this work is to support automatic service selection and adaptation in ubiquitous environments to increase availability, better QoS and deal with user mobility. To meet this goal, this paper presents AdaptUbiFlow, the OpenCOPI's (*Open Context Platform Integration*) [6, 7] component to support added-value service selection, service composition, and adaptation in ubiquitous environments. OpenCOPI is a platform based on semantic workflows and SOA (*Service Oriented Architecture*) that integrates context provision services and provides an environment that allows quick and easy context-aware application development. SOA combined with workflow management are promising technologies to meet such requirements. SOA refers to policies, practices, and frameworks that enable application functionality to be provided and consumed as set of services, which can be invoked by consumers through service interface descriptions published by service providers. Workflow is the automation of a business process in which tasks and goals are passed from one participant to another according to a set of procedural rules [8]. Specifically, workflow describes the order that a set of activities is attended by various services to complete a given procedure [9]. Workflows are useful in environments in which several services provided by different sources are available, where some of such services have similar functionality. Moreover, workflows can handle environmental changes at runtime according to resources availability, service quality, and context changes. They can specify ways of undoing previous operations and going back to a legal state from where another path can be taken to reach the stated goal. This is essential in ubiquitous environments which are characterized by uncertainties and faults, and where a set of service provider are often available to offer multiple ways of reaching a same goal [8]. *AdaptUbiFlow* is the OpenCOPI's component responsible for the adaptation process in

978-0-7695-4552-3/11 $26.00 © 2011 IEEE
DOI 10.1109/EUC.2011.12

ubiquitous environments. 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 requirements of composition, selection, and adaptation based on service quality to increase the availability at highly heterogeneous environments typical of ubiquitous computing.

There are many SOA-based and workflow-based platforms for ubiquitous computing in the literature [10-14] but in general they do not provide service selection and composition based on quality metadata. Moreover, although a few of these platforms have an adaptation mechanism, they do not consider some factors that can make the adaptation process more efficient (i.e. rollbacks, avoidance of service re-execution, etc). Finally, these recent platforms do not intent to provide support for heterogeneity, and consequently, they do not enable the access to different context provision middleware platforms in a transparent and uniform way. The use of AdaptUbiFlow along with other features supported by OpenCOPI provides the ability to deal with requirements such service composition, selection, and adaptation in ubiquitous environments since OpenCOPI provides the required heterogeneity and workflow support and the AdaptUbiFlow complements OpenCOPI with an efficient and added-value service selection and adaptation process.

This paper is organized as follows. Section II presents an overview of OpenCOPI. Section III contains the AdaptUbiFlow's service selection algorithm. Section IV shows how AdaptUbiFlow performs workflow adaptation. Section V presents an evaluation focusing on service selection and workflow adaptation. Section VI discusses related works. Finally, Section VII contains final remarks.

II. OPENCOPI OVERVIEW

OpenCOPI is 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 provides its own API and an OWL ontology-based context model, in which context is handled by adopting the Semantic Web Services perspective. In this perspective, service providers (including, but not restricted to, context provision middleware) publish their services using the OWL-S technology. Ubiquitous applications are services consumers; and OpenCOPI is a mediator, enabling that applications only need to know the OpenCOPI context model and interfaces.

A. Terminology.

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

Services. Services are the basic elements in the OpenCOPI architecture and their features and functionality are described by inputs, outputs, preconditions and effects (IOPEs) through OWL-S ontologies. There are two types of services in OpenCOPI: traditional services and context services. Such classification is not necessarily visible to final users but it is important to OpenCOPI service composition process. Traditional services are services provided by databases, legacy systems, message systems (SMS, e-mail, twitter), among others. They are selected through services IOPEs and QoS parameters. Context services are services that handle context information. They are provided by context provision middleware. In order to perform service selection with context services, additional quality metadata is required, for example, the values of QoC provided by the service.

Service dependency. In some cases, services may be dependent on other services. In OpenCOPI, dependency between services denotes a relationship in which the dependent service can be executed only if the service that is being depended upon is previously executed.

Semantic workflow. It is an abstract representation of a workflow described in terms of activities, representing the application execution flow, i.e., a workflow defines the sequence in which these activities must be executed. Activities are described in terms of Semantic Web services descriptions. Workflows are used to perform automatic service selection, composition, and orchestration. In OpenCOPI, each application has its own workflow and each workflow activity is a high level description of an application task. A workflow is independent of specific concrete services. This approach separating the abstract activity from the concrete services that are able to achieve it. This is useful mainly in cases where there are several similar services available, offered by different providers and consequently increasing the availability and QoS. 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.

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*. 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.

B. Architecture.

As mentioned before, OpenCOPI is a new middleware layer localized between applications and the integrated underlay context provision middleware platforms. This characteristic allows that applications consume services provided by these context provision middleware via the mediation of OpenCOPI. A more detailed description of OpenCOPI architecture can be found at [6] and *http://www.ppgsc.ufrn.br/~fred/opencopi/.*

OpenCOPI architecture encompasses two layers (*ServiceLayer* and *UnderlayIntegrationLayer*) as depicted in Figure 1. *ServiceLayer* is responsible for managing the abstractions of services (OWL-S descriptions) 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 *UnderlayIntegration Layer* is responsible for integrating service providers, performing context conversion whenever is needed (from each specific middleware context model to OpenCOPI context model) and conversion of the communication protocol (if necessary). The *IUnderlayIntegration* interface links service providers and OpenCOPI's *UnderlayIntegrationLayer*.

The *WorkflowManager* component manages the abstraction of available context services provided by context provision middleware platforms that are interacting with OpenCOPI. It supports the specification of semantic workflows and the generation of EPs. That component is responsible for discovering and composing Web services according to semantic workflow specifications, i.e., it maps workflow activities into Web services. The function of the *MetadataMonitor* component is to acquire metadata about services and context provided by context provision middleware to feed the *ContextInformationRepository* with metadata information. OpenCOPI adopts an SLA (*Service Level Agreements*) approach in which the service providers publish the quality metadata of their services and these metadata are used to select the services to be provided to the consumers.

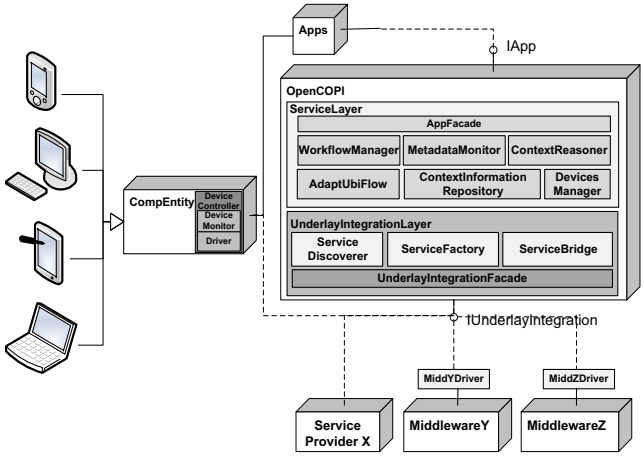

Figure 1. OpenCOPI Architecture.

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. *Context-InformationRepository* component stores context data and context metadata.

AdaptUbiFlow component is responsible for the adaptation process in OpenCOPI. In AdaptUbiFlow, an adaptation of an application means the replacement of the running EP by another EP (that achieves the same stated activities).This component works directly with the *MetadataMonitor* and *WorkflowManager* components to identify a fault (or other condition that triggers an adaptation, as for instance the availability of a new service with best quality) and automatically change the execution flow to use another EP.

The *DevicesManager* component manages the user computational entities to allow the applications migration from a device to another in case of user mobility or in case of resource limitations of the currently 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), with each device type having its own set of context information. Each computational entity in ubiquitous systems (e.g. laptops, smartphones, tablets, and so on) has a *DeviceController* responsible for controlling and monitoring the entity activity. *DeviceController* sends the actual device's status to *DevicesManager*. This allows OpenCOPI to change the execution from the actual device to another one if it is needed. *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 (at development time) to implement the context model transformation (from the middleware context model to the OpenCOPI context model), if necessary, the driver needs to abstract away such different APIs and to 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. Since drivers and platform integration is not the focus of this paper, please visit *http://www.ppgsc.ufrn.br/~fred/opencopi/architecture.html* for more details about the process of creating drivers.

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 *WorkflowManager* co§mponent. Thus, each service provided by the middleware API is represented by a Web service, created by OpenCOPI, to represent the respective service API. Each OpenCOPI Web service created by the *ServiceFactory* uses the driver tailored for the specific underlying middleware.

C. Workflow Representation.

In OpenCOPI, a workflow is represented by a direct acyclic graph (DAG) in which each intermediary node represents a specific service and each directed edge represents the execution direction between two services. Each complete path between *initial node* and *final node* is an *execution route*, and an *execution route* represents a possible EP in the workflow. So, the graph represents the workflow with all possible EPs. Figure 2 shows a graph with some possible execution routes, for example, (i) S1 → S2 → S3 → S4 → S5, (ii) S1 → S2' → S3 → S4'' → S5, etc.

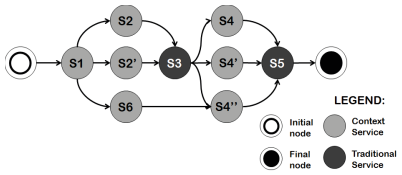

Figure 2. Example of graph representation.

III. ADAPTUBIFLOW'S SERVICE SELECTION

As mentioned before, OpenCOPI's *workflows* are abstract descriptions of applications and the generated *EPs* contain a set of concrete services that satisfy the respective abstract workflow. In general, there is more than one EP for each workflow and the number of these EPs depends on the amount of available services with the same functionality in the environment (at a given moment). Thus, it is necessary a service selection algorithm to choose which EP from the available ones should be executed. This section presents the service selection algorithm supported by AdaptUbiFlow.

The process of EP selection begins with the calculation of the quality of each EP. The quality of an EP is determined by quality parameters (QoS and QoC) values of all services contained in the EP. Before computing the EP's quality, it is necessary compute the global quality of each quality parameter. Global quality of a parameter means the quality parameter value for whole EP, i e., the value that represents the parameter of all services of an EP. This computation is needed to find the quality for the whole EP. The global quality of each parameter can be computed by aggregating the corresponding values for this parameter of all services in the respective EP. Different aggregation functions [15] are necessary to compute the global value of each parameter. Typical quality parameter aggregation functions are summation, multiplication, minimum and average relation (see Table 1). For example, *Responsing* is the QoS parameter used to measure the response time to execute each service. Thus, the value of the *Responsing* parameter for an EP is the sum of the *Responsing* values of all services ($q_{R(S)}$) that compose the EP ($q_{EP(R)}$). Another QoS parameter, *Availability*, can be aggregated ($q_{EP(A)}$) through a multiplication function of availability value of each EP's service ($q_{R(A)}$). *Performance* QoS parameter describes the number of service requests served by the service provider at a given period of time. Thus, the performance of EP is limited for the service with the smaller value for *Performance* attribute. Finally, *Freshness* QoC parameter describes the context information life span, i e., how long time ago the context information was created. Thus, the value of that QoC parameter is the average of context life span of all services of an EP.

Table 1. Aggregation functions examples.

Type	Example	Function
Summation	Responsing	$q_{EP(R)} = \sum_{s=1}^{m} q_{R(s)}$
Multiplication	Availability	$q_{EP(A)} = \prod_{s=1}^{m} q_{A(s)}$
Minimum	Performance	$q_{EP(P)} = \min_{s=1}^{m} q_{P(s)}$
Average	Freshness	$q_{EP(F)} = 1/m \times (\sum_{s=1}^{m} q_{F(s)})$

Once the value of all global (or aggregated) quality parameters were calculated, and considering that different quality parameters have different units and range, it is necessary to normalize these attributes into the same range to allow a unified and uniform measurement of EP's quality. 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 *Responsing* parameter). This process normalization of quality parameters is used by many authors [15, 16].

$$\begin{cases} q_{Ni} = (q_i - q_{\min})/(q_{\max} - q_{\min}), q_{\max} \neq q_{\min} \\ q_{Ni} = 1, q_{\max} = q_{\min} \end{cases}$$

Figure 3. Equation to normalize *positive* quality parameters (positive quality criteria).

$$\begin{cases} q_{Ni} = (q_{\max} - q_i)/(q_{\max} - q_{\min}), q_{\max} \neq q_{\min} \\ q_{Ni} = 1, q_{\max} = q_{\min} \end{cases}$$

Figure 4. Equation to normalize *negative* quality parameters (negative quality critéria).

Figure 3 and Figure 4 show the equation used in the normalization process for *positive* parameters and *negative* parameters, respectively. In those figures, q_{Ni} represents the aggregated quality value for parameter q. q_{max} and q_{min} are the maximum and minimum value for parameter q in the available EPs. In this process, each normalized parameter results in a value between 0 and 1 by comparing it with the minimum and maximum possible value according to the same parameter value about alternative EPs.

The normalization process is then followed by a weighing process to consider user priority and preferences. Thus, users can prioritize some quality parameters and minimize the importance of other quality parameters according to their needs. The weight (w) of each parameter (i) should be between 0 and 1 and the sum of weight of all parameters should be 1. Figure 5 presents the function to maximize the EP quality according a set of quality parameters (QoS and QoC). q_{EP} is the EP quality and it is calculated by the sum of the products between each quality parameter (where each parameter $i \in \{1, \ldots, m\}$) and its respective weight.

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

Figure 5. Function to maximize the execution plan quality.

At the selection phase, the *utility* of each EP is just the quality of the respective EP. The EP with biggest quality is selected. At the adaptation phase, the *utility* is represented for both: EP quality and adaptation cost for the respective EP (see Section IV).

IV. ADAPTUBIFLOW'S ADAPTATION PROCESS

As it was previously mentioned, ubiquitous computing environments are highly susceptible to changes, several of them unpredictable. In AdaptUbiFlow, an adaptation of an application means the replacement of the running EP by another EP (that achieves the same stated activities). Changes occurred at execution time may affect the application execution and performance. When a change happens, some actions may be needed to ensure that the applications continue running. If an adaptation is needed, AdaptUbiFlow analyzes the best strategy for this adaptation with minimal user awareness (thus, also promoting the autonomy of the application). This Section shows the types of changes that can trigger an adaptation process and the techniques and algorithm used by OpenCOPI to perform the adaptation.

In OpenCOPI architecture, a service is considered *in fault* when there is a problem that prevents it to meet/reply a received request. Examples of service failures are (i) a service provider that loses the connection with OpenCOPI and consequently cannot reply the service requests; (ii) a service that crashes in consequence of service provider's internal errors; (iii) a sensor device that has its energy depleted; (iv) a service that comes out of reach of the user due to user mobility. Thus, services failures are hard to handle, requiring the replacement of faulty services by other equivalent ones. Besides failures, the services and service providers are subject to other type of problem: *quality degradation*. In highly dynamic environments, the service quality can degenerate significantly due to network's bandwidth fluctuation, extensive use of a service, among other factors. This is a less severe problem, since such degradation does not necessarily mean a fault; it means that some quality parameters (QoS and/or QoC) may deteriorate at execution time. In addition, the emergence of new available services also needs to be taken into account since these new services can have better quality than services previously selected. Finally, while mobility can make some services unreachable, other services with better quality may become reachable. When quality degradation of a service is detected or new services emerge or yet services become reachable due to user mobility, it is necessary to assess the need to reconfigure the application execution.

A. Factors that trigger adaptation

There are four types of environment changes that may trigger the adaptation process in AdaptUbiFlow strategy.:

Service fault. Service failures can potentially lead to failure of ubiquitous applications. So, if a service fails, all running workflows that use this faulty service must be analyzed and automatically reconfigured.

Quality degradation. AdaptUbiFlow can reconfigure the application execution whenever there is a significant degradation in the quality of a service quality and there is an equivalent service option with higher quality to run the application. The decision about adaptation in these cases can be influenced by the user and sometimes a replacement of the EP leads to services re-execution, etc (meaning that there is a *cost* associated with the adaptation process).

Emerging of new services. New services and service providers can be added in the ubiquitous environment during execution time. These services can be used to give a better option of EP for a specific application already running. So, it is also necessary assessing the need of adaptation in these cases.

User mobility. Some context services are dependent on the user location. For example, a service that provides context information about the current user's room. If a user changes room, perhaps he/she can leave from radio range of a service in which he/she is using. At the same time, he/she can arrive at a new place covered by other services. In these cases, the workflow must automatically change the EP in use to another one that makes use of the services related with the new current location.

B. Factors that affect adaptation

AdaptUbiFlow's adaptation process chooses an EP for replacement in case of workflow adaptation. Section III presented the computation process for EP's quality. It was mentioned that the EP with the best quality is selected to be executed. In the adaptation process, an optional (not the first choice) EP needs to be selected to substitute the running EP. The adaptation selection is based not only on the EP quality but also on the cost of the adaptation process with the purpose of reducing the adaptation overhead, i.e., for improving the efficiency of adaptation. Our adaptation approach tries to reuse the services execution performed before the need to adapt. The adaptation cost of optional EPs is variable and consists in the number of services to be performed after adaptation (including services to be executed after the change of EP, services that require rollback and services that require compensatory action). Thus, we defined a relationship between the quality of optional EPs and the cost to replace the actual EP to them. Some important characteristics for this process:

Quality of execution plan. The quality (QoS / QoC) of an EP is used in this replacement process. Although it is a very important factor, it is not sufficient to ensure an efficient adaptation. In cases where some services had already run in the application's workflow, a choice of an EP very similar to the current one may be a better option than an EP with the best quality. The similar EP can reuse the output of the services executed before the fault, without violating services dependencies, neither perform rollbacks, thus decreasing the adaptation cost.

Cost of adaptation for each execution plan. The factors which influence the computation of adaptation cost are: (i) *reuse of service execution* -some services can be used in two or more EPs; in case of adaptation, it may be advantageous to give priority to EPs that reuse the result of services already executed by the current EP; (ii) *service dependencies* - in case of a service fault, all EPs that use this service and/or its dependent services cannot be chosen to substitute the current plan; (iii) *rollback* - in case of replacement of an EP, some services that have already been executed may require a rollback to return to the previous execution state (rollbackable services); (iv) *compensatory action* - in case of replacement of an EP, if a service needs to return to a

previous execution state but this service does not support rollback, a compensatory action can be provided by the driver that handles the communication between OpenCOPI and the respective service provider. Drivers can store the original state of a service before the service execution and, if necessary, this state can be recovered.

These factors can have different degree of importance in the adaptation process and such importance depends on the application's configuration made by user. Section IV.C presents how the user can influence the adaptation process so that the efficiency (cost) of the adaptation can be trade by the final quality of service delivered to the user.

C. Adaptation process

The adaptation process is composed of two phases. The first phase consists of selecting an optional EP to replace the current EP. The second phase consists in execution restarting.

Selection of a substitute execution plan. The choice of a new EP to replace the currently executing plan uses two categories of parameters: *EP's quality* and *adaptation cost*. The first one is the *quality value* calculated in Section III. For this parameter, the value desired is a *high* value of quality. The second parameter means the necessary actions to be performed in case of adaptation. For this parameter, the value desired is a *low* value of adaptation cost. The users can prioritize the new selection according to their needs. To do so, AdaptUbiFlow adopts an approach based on assigning weights to each parameter. Thus, users can choose different weights for quality and adaptation cost in the decision about which EP will replace the actual EP, as explained below. This allows trading quality by cost, tailoring the decision process to the user's needs.

Unlike the selection phase, the *utility* of an EP also is influenced by the adaptation cost at the adaptation phase. The utility is defined by a weighted average of *EP's quality* and *adaptation cost* parameters. There are five possible configurations for these *EP's quality* and *adaptation cost* parameters weights, producing what we call an *adaptation profile*: (a) *full service quality* adaptation profile gives full priority to services quality, with *service quality* weight (w_{SQ}) equal to 1 (w_{SQ} = 1.0) and *adaptation cost* weight (w_{AC}) equal to zero (w_{AC} = 0.0); (b) *service quality* profile gives priority to service quality but adaptation cost has some influence in the decision, with weights w_{SQ} = 0.75 and w_{AC} = 0.25; (c) *balanced*, the default configuration, gives equal weights between both parameters, i.e., w_{SQ} = w_{AC} = 0.5; (d) *low cost adaptation* gives priority to adaptation cost but service quality has influence in the substitute EP, with weights w_{SQ} = 0.25 and w_{AC} = 0.75; finally, (e) *lowest cost adaptation* gives full priority to adaptation cost: w_{SQ} = 0.0 versus w_{AC} = 1.0. Figure 6 shows the function to maximize the execution utility (μ) of each EP. This function consists in a weighted average equation between quality of EP and adaptation cost. The plan with the maximum EP's utility (μ max) is chosen to substitute the current one.

$$\mu = (q_{EP} \times w_{SQ}) + (c_{EP} \times w_{AC})$$

Figure 6. Equation to calculate the execution plan's utility.

q_{EP} and c_{EP} are respectively the service quality and adaptation cost relative values of an EP. These relative values represent the relationship of the best service quality and adaptation cost among the EPs in the same workflow. Thus, in a workflow with some EPs (*workflow* = {ep_1, ep_2, …, ep_n}), the best service quality and adaptation cost among these EPs are used to calculate the relative quality (biggest quality value) and cost values of each plan (smallest adaptation cost means the best adaptation utility). Section III showed the computation of q_{EP}. Figure 7 present the equations to compute the relative adaptation cost of an EP (c_{EP}). Figure 7 (*i*) shows the function to minimize the relative adaptation cost since a smaller c_{EP} value means a better adaptation quality. c_{EPabs} is the absolute adaptation cost value and c_{EPmax} is the biggest absolute adaptation cost value among the workflow's EPs. Figure 7 (*ii*) presents the c_{EPabs} computation equation, that is the sum of the number of services to be executed after the change of EP (*e*), services that require rollback (*r*) and services that require compensatory action (*c*). The number of services to be executed (*e*) is defined by service reuse and dependences among services.

$$(i): c_{EP} = 1 - (c_{EPabs} / c_{EP\,\max}) \qquad (ii): c_{EPabs} = e + r + c$$

Figure 7. Formule to calculate the adaptation quality of execution plans.

Execution restarting. Since the selection of a new EP process is finished, the process responsible for changing to the new EP starts. This process consists in making all necessary actions (rollbacks, compensatory actions and restart execution) in a seamless way for the user.

V. EVALUATION

The main purpose of the evaluation of AdaptUbiFlow presented in this paper is to assess its service selection and workflow adaptation approach. To achieve this goal, services used in a case study (see Section V.A) are forced to fail so that the adaptation process is triggered. Regarding our approach for service selection, it was evaluated by comparing it with a random selection approach. We also evaluated if the prioritization of specific quality parameters really selects the EP according to our expectative, i e, if the selected EP really is the best available one regarding the specific parameter. We also assessed the effect of using different *adaptation profiles*, changing the weights for EPs' quality and adaptation efficiency (adaptation cost) to evaluate if the prioritization of one of them properly selects the substitute EP. Finally, we evaluated the overhead generated by composition, selection, adaptation and execution processes considering different number of EPs for the same workflow.

A. Case study

The conducted case study is an application from the Gas & Oil Industry domain. Specifically, it is an 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 this 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 (*maxBurden*) of burden for its correct operation. If this value is reached abruptly, the pumping unit operation must be stopped quickly to prevent its damage (a reactive strategy). Furthermore, there is an intermediary value (*intermValue*), that calls for attention, where actions can be taken (in a proactive way) to prevent the pumping unit to be stopped, consequently avoiding loss of production and risks to the equipment. The complete case study description, including service providers and their respective services, possible EPs and services metadata can be found at *http://www.ppgsc.ufrn.br/~fred/opencopi/case_studies.html*.

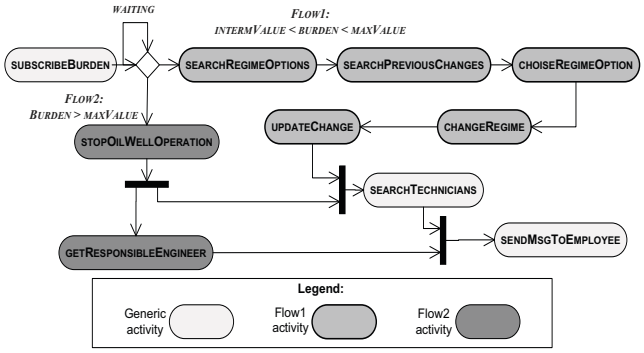

Figure 8. Case study workflow.

Figure 8 shows the semantic workflow representing the case study application in which each activity is realized at least by one service. The execution starts in the first activity: subscribe to monitor the value of pumping unit *burden*. If the current *burden* value is between pumping unit's *intermediary burden* value and *maximum* value, the workflow follows *Flow1*. If the *burden* value is greater than the maximum value, the workflow follows *Flow2*. *Flow1* encompasses activities to automatically change the regime of the pumping unit operation, where regime is the relation between the length of pumping unit's *stem* and the cycles *per minute* of this *stem*. First, the *searchRegimeOptions* activity looks for possible regimes of pumping unit operation, in which each regime variable is composed of a stem length value and *cycles per minute value.* Then *searchPreviousChanges* activity 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* (stores and retrieves the changes made earlier in the pumping units). Finally, a search for technicians available 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 in which *burden* is greater than the maximum limit of the pumping unit. To avoid the pumping unit damage, the operation of the well is stopped. After this, a search is performed to find the engineer responsible for this oil well and the technicians near to the oil well. At last, warning messages are sent to them.

B. Evaluation results

To evaluate AdaptUbiflow, we made replicas of some services to enable the generation of multiple EPs. The services replicated were four services that perform the *SearchTechnicians* activity (four replicas representing different technologies for user location) and *SendMsgToEmployee* activity (two replicas). Each service replica has different values for quality metadata. This configuration resulted in eight distinct EPs (different replicas combination) with different qualities. We named each EP as *EP1, EP2,…, EP8* to facilitate the explanation of the evaluation. We divided AdaptUbiFlow evaluation in three distinct aspects to be analyzed: (i) selection process; (ii) adaptation process, and (iii) generated overhead.

Selection process evaluation. We compared the AdaptUbiFLow selection approach with EPs randomly selection. Considering that there are 8 possible executions plans, we executed the selection process one hundred times for each approach (random and AdaptUbiFlow's) and the results are: the best EP was selected in 13.33% of cases for the random approach. Using AdaptUbiFlow, the best EP was selected for all (100%) execution rounds.

We also evaluated if the prioritization of some quality parameter really selects the EP as expected. Figure 9 presents the utility (or quality) of each EP and the selected plan for each different prioritization tested. For example, when the maximum priority (weight = 1) was given to *Availability* parameter, *EP4* was selected; when *Responsing* was prioritized, *EP5* was selected; the prioritization of *Availability* (0.5) and *Responsing* (0.5) resulted in selecting *EP7* (q_{EP7}= 0.66). When the same priority was assigned for all parameters, *EP1* was selected (q_{EP1} = 0.78). We found that, for all parameters prioritization, the selected substitute plan was the expected plan for that configuration.

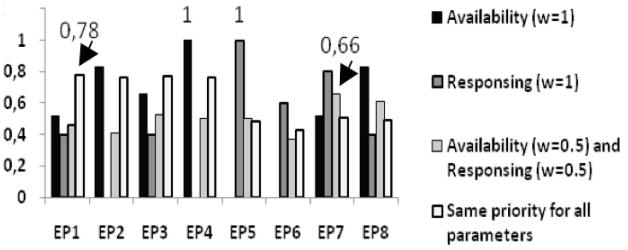

Figure 9. Selection based on different parameters prioritization.

Adaptation process evaluation. For the adaptation process, we evaluated if each possible adaptation profile (weights of EP's quality and adaptation cost) really selects the best candidate for the substitute EP. Note that distinct

adaptation profiles can select distinct substitute EPs. Considering that *EP1* was selected to execute at the selection phase (same priority – see Fig. 9), we forced the failure of a service encompassed in *EP1* to trigger the adaptation process. Table 2 presents the selected substitute plan for each different adaptation profile tested. For example, *Full service quality*, in which just EP quality is considered, selected *EP4*; *Balanced* configuration selected *EP4* again, but *Lowest cost adaptation* selected *EP2*. We found that, for all assessed adaptation profiles, the selected substitute plan was the expected plan for that configuration.

Table 2. Adaptation based on different configurations of weights.

Adaptation profile	Selected plan
Full service quality, Service quality, Balanced	EP4
Low cost adaptation, Lowest cost adaptation	EP2

Overhead evaluation. Since OpenCOPI represents an additional layer between ubiquitous applications and services provided by many platforms, it is expected that its use increases the overall processing time and consequently the response time for users. So, it is important to measure the impact of OpenCOPI in the application running time. Moreover, it is important to relate this time overhead with the benefits that OpenCOPI provides to the implementation and the execution of ubiquitous applications. The experiments conducted to assess the overhead were executed on Mac OS X operating system, using a computer with processor Intel® Core™ 2 Duo 2.4 GHz and 4 GB of RAM memory. Firstly, we measured the overhead generated by the service composition, the service selection, and the adaptation processes. We ran the workflow with a different number of possible executions plans – 2, 4, 6, and 8. Figure 10 shows the overhead average (in milliseconds) with a confidence interval of 95% for composition (Figure 10.A), selection and adaptation (Figure10.B).

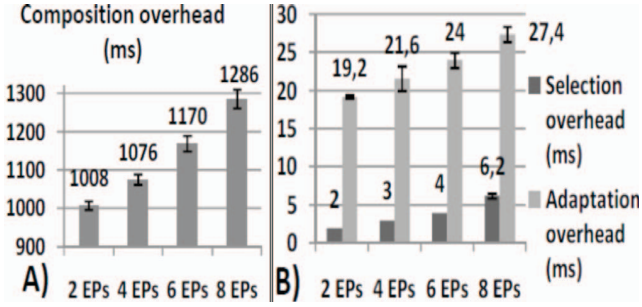

Figure 10. Overhead of service composition, EP selection and adaptation

The service composition is the process responsible for discovering services able to perform each workflow activity and for creating possible EPs. This is the most expensive process since it requires analyzing the ontologies of each available service. However, we consider that the spent time achieved in the results is not relevant for ubiquitous applications since this process transforms an abstract specification into concrete service compositions, enabling late service selection and decoupling the application and used services. For the simplest configuration (two possible EPs) the semantic composition spent 1008 ms on average o build both EPs and for the more complex configuration (8 possible EPs) it spent 1286 ms on average. The EP selection is the cheapest process. For the simplest configuration the selection spent 2 ms on average and for the more complex configuration it spent 6 ms on average. The adaptation process (presented in Section IV) consists of choosing a substitute EP and the adaptation preparation (re-start execution). For the simplest configuration the adaptation process spent 19 ms on average and the more complex configuration it spent 27 ms on average.

Another aspect analyzed was the application execution time. For this purpose, two versions of the case study were built. The first one is the workflow specified and executed using OpenCOPI. The second application directly invokes the same services through Java source code. The services are executed according to the sequence followed by the workflow executed by OpenCOPI. The average running time of the application without OpenCOPI was 1.2 seconds to call all services involved in the case study. In the application that uses OpenCOPI, the execution time increased to 1.9 seconds.

We consider the evaluation results very encouraging. Firstly, AdaptUbiFlow's selection and adaptation processes run as expected, always resulting in the best EP selection in selection and adaptation process. Second, the time intervals spent in the service composition, EP selection and adaptation processes were very low. Moreover, the difference (about 0.7 seconds) between the application executed with OpenCOPI (including the AdaptUbiFlow component) and the application directly developed in Java was not significant compared to the benefits provided by OpenCOPI. For instance, to build the application without the use of OpenCOPI, 139 lines of code were necessary only to call all the 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 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. As a consequence, the development is harder, reuse is impracticable, and it is difficult to dynamically select services and also to support adaptation.

VI. Related Work

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 the highly dynamic and heterogeneous ubiquitous environments. In general, existing proposals do not allow dynamic service composition and adaptation based in quality metadata. Even few platforms that enable adaptation do not consider factors that allow an efficient adaptation, such as dependence between services, rollbacks, service re-execution, etc.

[10] presents a workflow approach for modeling and managing the user's 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. [11] presents an architecture that supports the distributed execution of workflows in pervasive environments based on decentralized control. Both proposals lack of mechanisms to allow dynamic service composition and workflow adaptation. [12] presents 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. In such work, context information is limited to bandwidth and location and user configuration and workflow adaptation are not supported. The mechanisms presented in [14] support workflow adaptation but just in case of service failure. The adaptation process is modeled before workflow execution. It does not consider QoS to service selection and workflow adaptation. [13] presents an interesting set of tools and principles to support context-aware run-time deviations and changes in the workflow execution, allowing workflow adaptation in case of service fail but not allowing the user to configure the adaptation preferences in case of quality degradation of the services. Moreover, unlike AdaptUbiFlow, it does not consider the cost of adaptation to select the new flow in case of adaptation.

Differently from all the previously mentioned proposals, this paper investigates how to automatically manage workflows, selecting the best option of EP and the automatic adaptation decisions at runtime according to user preferences. In AdaptUbiFlow, users can configure the workflow's service selection and adaptation process in an easy way through a XML configuration file. This feature increments the involvement of the end-user with the application.

VII. CONCLUSIONS

In this paper we introduced AdaptUbiFlow, an automatic and adaptive mechanism to support service selection, service composition, and workflow adaptation in ubiquitous applications. It deals with services failures in these environments, increasing the availability of such systems. When a device fails, AdaptUbiFlow supports the adaptation of the system replacing the failed device (or service) by another equivalent; this makes the system fault-tolerant without the need for any manual intervention. The replacing device/service is chosen taking into account not only the QoS provided but also the application's execution flow to ensure the best adaptation option will be chosen.

Initial experiments performed with AdaptUbiFlow showed promising results, demonstrating that in the occurrence of faults, the mechanism selects the best option for adaptation according to user configuration, considering the quality of services and the overhead of adaptation process. Moreover, analyzing AdaptUbiFlow together with OpenCOPI focusing on service selection and workflow adaptation, we believe that our approach can effectively contribute to the leverage the benefits of ubiquitous computing.

ACKNOWLEDGMENT

The authors wish to thank the Brazilian National Oil and Biofuels Agency (ANP), through PRH-22 Program, and the National Council for Scientific and Technological Development (CNPq), through grants 477229/2009-3, 306938/2008-1, 311515/2009-6 and 480359/2009-1, who partially supported this work.

REFERENCES

1. Davidyuk, O., et al., *MEDUSA: Middleware for End-User Composition of Ubiquitous Applications. Handbook of Research on Ambient Intelligence and Smart Environments: Trends and Perspectives*, 2010.

2. Hepner, M., R. Baird, and R. Gamble. *Dynamically Changing Workflows of Web Services*. in *Congress on Services - I*. 2009.

3. Buchholz, T., A. Küpper, and M. Schiffers. *Quality of Context: What it is and why we need it*. in *Workshop of the HP OpenView University Association*. 2003. Geneva, Switzerland.

4. Xia, J., et al. *Fault-resilient ubiquitous service composition*. in *3rd IET International Conference on Intelligent Environments*. 2007.

5. Funk, C., et al. *Adaptation of Composite Services in Pervasive Computing Environments*. in *IEEE Int. Conf. on Pervasive Services*. 2007, Turkey.

6. Lopes, F., et *al. Context-based Heterogeneous Middleware Integration.* in Workshop on Middleware for Ubiquitous and Pervasive Systems. 2009.

7. Lopes, F., et al. *On the Integration of Context-based Heterogeneous Middleware for Ubiquitous Computing*. in *Int. Workshop on Middleware for Pervasive and Ad-hoc Computing (MPAC'08)*. Leuven.2008.

8. Allen, R., *Workflow: An Introduction*, in *Workflow handbook*, L. Fischer, Editor 2001

9. Abbasi, A. and Z. Shaikh. *A Conceptual Framework for Smart Workflow Management*. in *Int. Conf. on Info Management and Engineering*. 2009.

10. Ranganathan, A. et. al. *Using Workflows to Coordinate Web Services in Pervasive Computing Environments*. *IEEE Int. Conf. on Web Services*. 2004.

11. Montagut, F. and R. Molva. *Enabling Pervasive Execution of Workflows*. in *Collaborative Computing: Networking, Applications and Worksharing*. 2005. IEEE.

12. Tang, F., et al. *Towards Context-Aware Workflow Management for Ubiquitous Computing*. in *Int. Conf. on Embedded Software and Systems*. 2008.

13. Marconi, A., et al., *Enabling Adaptation of Pervasive Flows: Built-in Contextual Adaptation.* LNCS, 2009. 5900/2009: p. 445-454.

14. Mostarda, L., S. Marinovic, and N. Dulay. *Distributed Orchestration of Pervasive Services*. in *IEEE Int. Conf. on Advanced Information Networking and Applications* 2010. Perth, Australia.

15. Yanwei, Z., et al. *A Dynamic Web Services Selection based on Decomposition of Global QoS Constraints*. in *IEEE Youth Conference on Information Computing and Telecommunications*. 2010.

16. Ardagna, D. et. al. *Per-flow optimal service selection for Web services based processes.* The Journal of Systems and Software, 2010. 83: p. 12.