Journal of Universal Computer Science, vol. 20, no. 9 (2014), 1327-1351
submitted: 18/6/12, accepted: 5/5/14, appeared: 1/9/14 © J.UCS

A Middleware Architecture for Dynamic Adaptation in Ubiquitous Computing

João Lopes, Rodrigo Souza, Cláudio Geyer
(Federal University of Rio Grande do Sul, Porto Alegre - RS, Brazil
{jlblopes, rssouza, geyer}@inf.ufrgs.br)

Cristiano Costa, Jorge Barbosa
(University of the Vale do Rio dos Sinos, São Leopoldo - RS, Brazil
{cac, jbarbosa}@unisinos.br)

Ana Pernas, Adenauer Yamin
(Federal University of Pelotas, Pelotas - RS, Brazil
{marilza, adenauer}@inf.ufpel.edu.br)

Abstract: The development of applications that adapt to the environment and remain running even when the user is moving or switching device, remains an open research challenge. In this article we present a view of the EXEHDA middleware and a new service created for dynamic adaptation. EXEHDA is service-oriented, adaptive and was conceived to support the execution of ubiquitous applications. The main concept in the proposed design for the middleware and for the application is context awareness expressed in an adaptive behavior. The middleware manages and implements the *follow-me semantics* for ubiquitous applications. This is also a key to provide functionality adapted to the constraints and unpredictability of the large-scale environment. To achieve this objective, EXEHDA employs various strategies in its services to allow the adaptation to the current context, such as on-demand adaptive service loading, and dynamic discovery and configuration. In that sense, EXEHDA provides services for distributed adaptive execution, context recognition, ubiquitous storage and access, and anonymous and asynchronous communications. To evaluate the new service proposed for dynamic adaptation we developed a case study, implementing an application in medical area. Analyzing the results we can see that the users found the application easy to use and usefulness for health workers at a hospital. This work is sponsored by RNP, FINEP and CNPq - Brazilian Foundations.

Key Words: Ubiquitous Computing, Service-oriented Middleware, Adaptive Middleware, Context-aware Adaptation, Dynamic Adaptation.
Category: L.7, C.2.4, J.3

1 Introduction

The field of Ubiquitous Computing (UbiComp) presupposes the provision of computing environments with information and communication technology everywhere, for everyone, all the time. In this way, human beings will be citizens of both the physical world and the augmented reality that extends this world [Caceres and Friday 2012].

In Ubiquitous Computing, systems need to sense and adapt to the environment [Kakousis et al. 2010]. They have to understand the context in which they are interested. This new class of computer systems, adaptive to the context, enables the development of richer applications, more elaborated and complex, exploiting mobility of the user and the dynamic nature of modern computing infrastructures. Nevertheless, the development of applications that continuously adapt to the environment and remain running even when the user is moving or switching device, remains an open research challenge [Costa et al. 2010] [Knappmeyer et al. 2013].

We consider that a central issue in the context-aware adaptation is the degree of expressiveness that can be achieved in the description of contextual information [Da et al. 2011]. In that sense, the use of technologies to support semantic processing, such as ontologies, are suitable for supporting the reasoning services in the scope of ubiquitous applications [Baldauf et al. 2007] [Bettini et al. 2010]. Ontologies can capture the full range of concepts involved in a complex environment, increasing the expressiveness of context information and providing support for reasoning [Fujii and Suda 2009].

Previously published work about EXEHDA middleware [Lopes et al. 2007] discussed and proposed services to deal with aspects related to communication, physical resources, context information. In turn, this work discusses the context adaptation, and proposes a new EXEHDA service created for dynamic adaptation, called *DA Service*.

The main contribution of this work is an architectural model, and an ontology-based semantic model designed to support the development of adaptive applications. This proposal contributes particularly to the Context Recognition and Adaptation Subsystem of EXEHDA middleware.

The article is structured as follows. Section 2 outlines the EXEHDA Middleware software architecture. Section 3 describes the main features of the EXEHDA middleware. Section 4 discusses the EXEHDA middleware services. Section 5 details the context-aware adaptation in the EXEHDA middleware, introducing a service for dynamic adaptation control. In Section 6 we present a case study and the evaluation of the dynamic adaptation service. Finally, related work and concluding remarks are presented in sections 7 and 8, respectively.

2 EXEHDA Middleware Software Architecture

The foundation of our proposal is the EXEHDA Software Architecture, in which cells form a large-scale computing environment. These cells are composed of several mobile and stationary physical resources. The components of the computing environment, such as: data, code, devices, services and resources, are ubiquitous and managed by a middleware that provides continuous access to them [Augustin et al. 2008].

In this computing scenario, the mobile devices are increasingly used as interfaces. They do not store neither code nor data persistently (except for some caching strategy), but operate as portals that obtain the code to be executed and that can transfer the execution to other devices using proximity or resource availability as a selection criterion [Bellavista et al. 2012]. Furthermore, each user has a virtual environment that can be accessed at any location, and with the available device. Moreover, the user's location in the environment has a significant effect in the way ubiquitous applications are executed. As the user moves physically, i.e. by carrying his current device - user mobility - or changing the device being used - terminal mobility, his currently running applications, in addition to the user's virtual environment, need to be continuously available to him, following the user's movements in the ubiquitous space. Such behavior defines what we call *follow-me semantics* of ubiquitous applications [Costa et al. 2008].

The main properties of EXEHDA applications are: distributed, mobile, adaptive and reactive to the context, and capable of expressing the follow-me semantics - the application follows the user in his movement at a large-scale space. The application's code is installed on-demand on the devices used and this installation is adaptive to context of each device [Augustin et al. 2005].

Another behavior present in ubiquitous executions is the notion of planned disconnections. A mobile device, for example, would rather operate disconnected to reduce battery consumption and, at specific moments, reconnect to update the state of the global execution. Such disconnection/reconnection procedures should be, whenever possible, transparent to the applications.

The EXEHDA middleware software architecture, shown in Figure 1, aims to provide an integrated solution to build and execute large-scale ubiquitous applications. The execution of such applications is supported by EXEHDA middleware.

EXEHDA architecture is divided in a logical organization of three layers: (Upper) application layer; (Middle) support layer, and execution environment; and (Lower) basic systems' layer. The Upper Layer corresponds to the abstractions provides to the application designer to ease the development of ubiquitous context-aware adaptive application. This is mainly obtained by the provision of a Java Framework. We also have in this and the next layer the representation of context awareness. The reason for that is to underscores its importance in the architecture, highlighting their presence in the design of many components.

In the Middle Layer are the support mechanisms for the implementation of ubiquitous application and adaptation strategies. This layer consists of two levels: the first level consists of the application service modules, and the second level is formed by the EXEHDA basic services. These basic services enable features required for the upper level and cover various aspects, such as ubiquitous access, communication, distributed execution, context recognition,

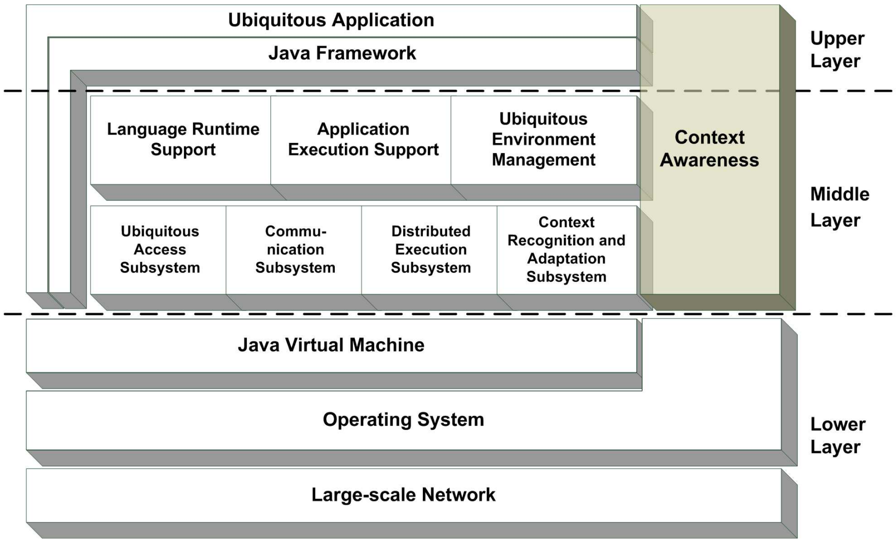

Figure 1: EXEHDA Middleware Software Architecture

and adaptation.

Finally, the Lower Layer of the architecture is composed of native languages and systems that integrate the execution physical environment. For reasons of portability, in this layer the platform for implementation is the Java Virtual Machine in its different approaches. The architecture assumes the existence of a network to support the execution of components and services on a global scale.

3 EXEHDA Middleware Features

EXEHDA is service-oriented architecture that contribute in three perspectives: (i) provide a management through services to control the physical environment in which the processing will take place; (ii) support for execution of applications, by providing the services and abstractions needed to implement the follow-me semantics; and (iii) offers an API (Application Programming Interface) to foster ubiquitous application development [Yamin et al. 2005a].

The requirements of operation in a high heterogeneous environment, in which hardware capabilities and software availability on each device may vary, have motivated the use of pluggable services. In this approach, the middleware minimum core extends its functionalities according to the availability of resources. The loading of these pluggable services is done on-demand and, moreover, is context adaptive. In this way, we are able to employ implementations of services that are better tuned to each device and also reduce resource consumption by only loading services that are effectively used. Such scheme is possible because services are defined by their semantics and interface rather than by a specific implementation.

Additionally, in each node, a execution profile defines the loading policy that will be applied for each of the middleware services. This loading policy specifies one of two currently available modes: (i) bootstrap, meaning that the service should be loaded in the node startup process; or (ii) on demand, meaning that the service will be loaded in its first use.

The minimum core provides the loading policy for services and is included in each node that composes the execution environment. Using this feature, we can configure what is needed and when it should be loaded. However, there are two services that must always be present: (i) Profile Manager, in charge of interpreting the execution profiles, making these profiles available at runtime for the other middleware services; (ii) Service Manager, which activates services in anode based on the information provided by the Profile Manager. Service code is loaded on demand from a service repository, which may be local or remote depending on the device storage capacity and the nature of the service being loaded.

EXEHDA has the requirement of remaining operational during periods of planned disconnection. To support this feature, we split the services into two parts: a node instance, and a cellular instance. The former is local to each device, while the latter executes in the base node. Hence, the local device will remain operational during the planned disconnection, regarding that the node instance of that service would temporary renounce the access to resources that are in the network. On the other hand, the cellular instance of the service, in execution on the base node of the cell, acts as a reference point for services that require distributed, inter-node or inter-cell, coordination procedures [Augustin et al. 2005].

4 EXEHDA Middleware Services

EXEHDA is composed of several integrated services. An overview of these services is presented in this section. The new service created for dynamic adaptation (*DA Service*) is described in section 5.

The middleware services are conceptually organized in subsystems: data and code ubiquitous access, uncoupled spatial and temporal communication, large-scale distribution, context recognition and adaptation [Yamin et al. 2005b]. The subsystem integration is shown in Figure 2.

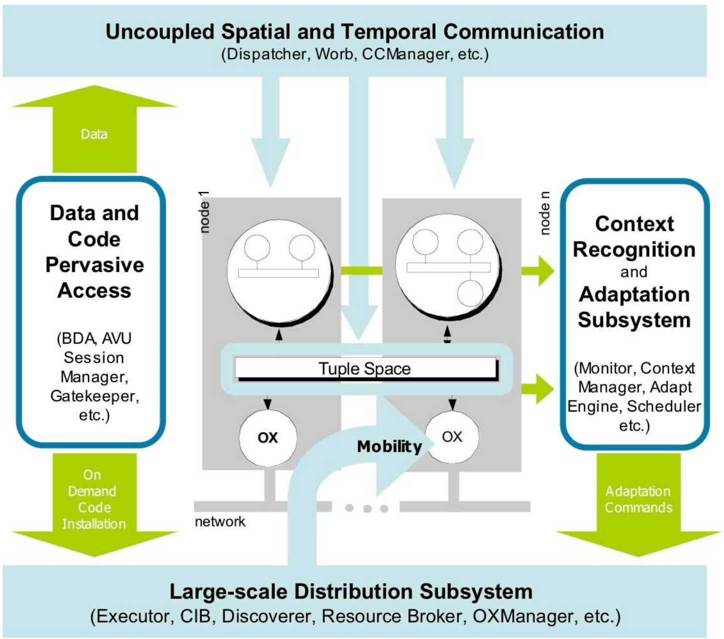

Figure 2: EXEHDA Middleware Subsystems

4.1 Multi-Level Communication Services

Regarding communications, EXEHDA currently provides, through the *Dispatcher*, *WORB*, and *CCManager* services, three types of communication primitives, each one addressing a distinct abstraction level.

The *Dispatcher Service* corresponds to the lowest abstraction level, providing message-based communications. Message delivering is done through per-application channels, which may be configured to ensure several levels of protection for the data being transmitted. Protection levels range from data integrity, using digital signatures, to privacy through encryption mechanisms. Additionally, the *Dispatcher Service* uses a checkpointing/recovery mechanism for the channels, which is activated when a planned disconnection is in course. This feature may or may not be activated by the upper communication layers depending on its particular demands.

In order to make the development of distributed services easier, EXEHDA also provides an intermediary solution for communications, based on Remote Method Invocations, through the *WORB Service*. The programming model is similar to Java RMI, but optimized to the ubiquitous requirements. More specifically, WORB remote method invocations, differently from Java RMI, do not require that the device keep connected during the entire execution of the method on the remote node. Instead, WORB was built on the functionality provided by the Dispatcher Service, including a per-invocation ID. The invocation ID remains valid during the disconnection, allowing the WORB to re-sync with the remote node after reconnection and obtain the returned values from the invocation.

At a higher level, the *CCManager Service* provides tuple-space based communications. It builds on the *WORB Service*, which also handles planned disconnections, providing to applications an anonymous and asynchronous communication support. This model is provided in the *CCManager Service* and is better suited to scenarios that application components might migrate among nodes, since it does not require both sides to coexist at the same time for the communication to take place.

4.2 Management Services of Physical Resources

From the middleware point of view, environment resources fit in one of two categories: processing node or specialized resources. The former corresponds to the nodes, which effectively execute and whose access is managed by the middleware. The latter corresponds to specialized devices, e.g. printers, scanners, etc., whose access is not done through one of the middleware services, but through the use of some specific libraries. Although not managed by EXEHDA, the specialized devices are also cataloged in the *CIB Service* in order to allow applications to locate and use them.

The *Discoverer Service* is in charge of finding specialized resources in the environment based on an abstract definition of the resource. Typically, this service interacts with the *CIB Service* from its own cell, aiming at satisfying the resource discovery request in the scope of the local cell. When the local resources fail to fulfill the request, the *Discoverer Service* interacts with the *Resource Broker service* of the neighbors cells. The strategy adopted in this extra-cell search is characteristic of the particular *Discoverer Service* instance in use. These services employ a language to describe resources and its interfaces are standardized. Since the Middleware does not manage specialized resources, the results of a *Discoverer Service* search do not imply resource allocation or even resource reservation.

4.3 Services to Build the Context Information

The *Monitor Service* implements a monitoring scheme based on sensors, which employs indexes to describe specific aspect of the environment. These sensors can be customized through parameters. The whole set of sensors installed on a node is part of the node description information registered in the *CIB Service.*

The data generated by each sensor is gathered by the *Monitor Service*, which typically runs on the same node that sensors are installed. The gathered data is published by the *Monitor Service* to a *Collector Service*, which typically runs on the base-node.

Both the gathering of data by the sensors and the publication to the *Collector Service* by the Monitor occurs in discrete multiples of a per-node configured *quantum*. The quantum parameter allows the resource owner to control, externally to the middleware, the degree of intrusion of the monitoring mechanism in the host. After a *quantum* of time expires, the *Monitor Service* executes a pooling operation over the active sensors in the node. Then, it applies the publishing criteria specified for the sensor data, determining, or not, the generation of a publishing event for that sensor. Thus, the events generated after a quantum expiration are grouped into a single message, reducing the amount of data that the Monitor has to transmit to the Collector.

The *Collector Service* aggregates information from several monitors in the cell and forwards them to the registered consumers. Among such consumers are other middleware services like the *Context Manager* and the *Dynamic Adaptation Service*.

5 Dynamic Adaptation Control Service

In order to increase the flexibility for specifying adaptations, the Dynamic Adaptation Service (*DA Service*) is designed to support the definition of application requirements.

In the design of the service we considered that several elements of context could be relevant for a given application, and that the possibility of various adaptations for the same software component could help the maintainability and even the improvement in the quality of the execution, whenever a context change occurs.

This adaptation control model allows an incremental development of specifications, which includes policies, rules, parameters, constraints, and adaptation actions. This feature enables reuse and customization of these specifications in the development of adaptive applications.

The Dynamic Adaptation Service employs utility functions, which considers the parameters of adaptation and execution context, as well as user preferences. This function calculates the weighted sum of the differences between the parameters offered by the environment and required by the application, where the weights reflect the priorities of user needs. Within this model, the goal of *DA Service* can be formulated as a guarantee that at any time, the adaptation with the highest utility is performed and does not violate the resource constraints imposed by the environment.

The *DA Service* aggregates a semantic approach to the EXEHDA's services. It is based on a model represented by ontologies. The semantic model, an overview of this service architecture, and the integration with the others EXEHDA's middleware services are presented in the next subsections.

5.1 Semantic Model

The semantic model used by *DA Service* includes an ontology, called OntUbi, which represents the environment managed by EXEHDA. The OntUbi is composed by the following ontologies: (i) OntContext - Context Situation - it represents the collected contexts, notified contexts, and contexts of interest (see Figure 3); (ii) OntAdapt - Adaptation Policy - rules, parameters, operations and preferences, constraints and adaptation actions for the applications components.

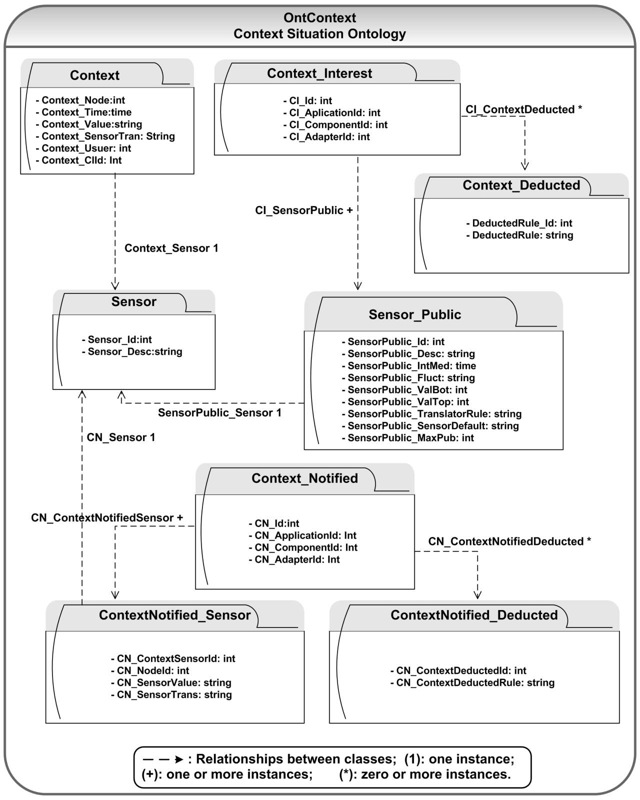

Figure 3: Context Situation Ontology

In OntUbi, elements of context are entities and have three basic categories of information: (i) hardware resources: devices, peripherals and information of nodes and sensors, such as location, memory, battery, bandwidth, CPU load, among others; (ii) software resources: description of the operating system, middleware and applications, and adaptation policy; (iii) user resources: user profile and preferences.

The Adaptation Policy is in charge of registering the profiles of the application software components. This ontology is composed by specification rules that manage the adaptive behavior of the application components.

In OntAdapt, we can set policies for various applications. Each application may consist of several components, instances of relationship "Application_Component". Each component can have several adaptations, instances of the relationship "Component_Adapter". Each adaptation can have multiple parameters, instances of relationship "Adapter_ParamType". A parameter can have multiple instances of lower value, higher value, and utility value, which will be instances of the relationship "ParamType_ParamValue".

The OntAdapt is instantiated in the application development time and used at runtime by the *DA Service* to guide the context adaptations of the components. The ontological model allows an incremental evolution of the OntAdapt instances, such as rules, parameters, adapters, and operations.

Figure 4 shows the attributes and relationships between classes of OntAdapt. This ontology is instantiated and maintained by the developer, using a framework, called FWADAPT [Warken et al. 2010]. The programmer employs this framework, in development time, to instantiate the adaptation policy.

The main aims considered in the design of this framework were: (i) to be general, allowing different kind of applications follow the EXEHDA's programming model; (ii) to provide procedures for editing or extending definitions previously established in applications; (iii) to enable resources that foster the reuse of existing specifications to adaptations, rules, and operations.

5.2 Architectural Model

The design of the *DA Service* considers the following architectural features: (i) monitoring of context, which must be kept detached from the application, and executes through the use of reusable elements in the middleware; (ii) context processing, which executes in the middleware and must be extensible, supporting the inclusion of new elements of context and new ways of reasoning.

In the adaptation decision-making, the model considers three categories of information: (i) the context, based on monitored data, semantic information and inferences; (ii) the adaptation policy of the application; (iii) the user preferences.

The *DA Service* supports both functional (that selects the code being executed) and non-functional (related to scheduling and resource allocation)

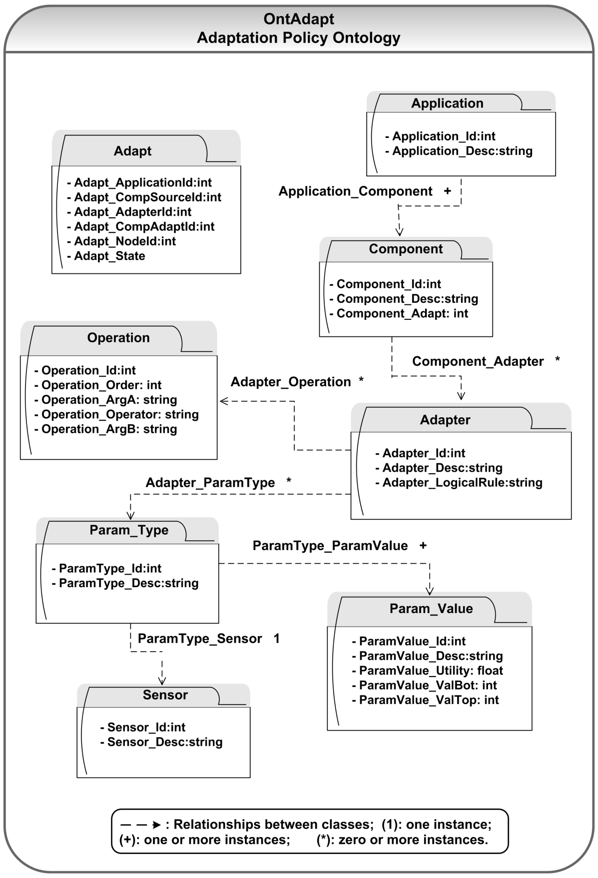

Figure 4: Adaptation Policy Ontology

adaptations [Lopes et al. 2007]. Moreover, the service includes a Multi-level Collaborative adaptation organized in two levels. At the application level, in development time, two approaches are provided: (i) creation of the following ontologies: Adaptation Policy and Context of Interest; (ii) programming of the adaptive commands in the software components. Furthermore, at the application level, at runtime, the middleware activates commands that trigger the EXEHDA adaptive services. At the same time, in the service level, the middleware makes the decision of adaptation, considering the state of the context, the adaptation policy, and of user preferences.

DA Service communicates through predefined interfaces, both with the applications, by the commands of adaptation, as with other middleware services. The service encompasses communication, contextual data, adaptation decisions, and can be developed, modified and extended independently.

We present the definition of *DA Service* software architecture in Figure 5. The service is organized in the following modules.

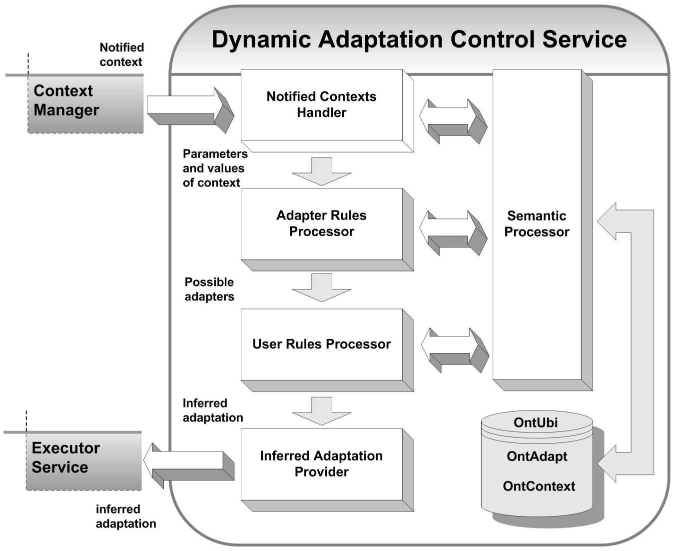

Figure 5: Software Architecture of the *DA Service*

Notified Contexts Handler module receives the identification of notified contexts from the Context Recognition Subsystem. These contexts are accessed in order to define the possible interest of the application component, and the sensor values associated with it. This mechanism also deals with application policy, using the semantic processor, and accesses the information needed to determine the adaptation rule. In this module DA Service prepares all the needed information for deciding which adaptation to make.

A list with one or more actions to adapt, classified by criteria of utility, is produced by Adapter Rules Processor module. It infers the possible options for adaptive actions, using the adaptation rule and the parameters values used to evaluate it. Possible actions are classified using the utility function, from higher to lower use of adaptive action. Functional and non-functional adaptations are computed the same way, through the adaptation rule.

User Rules Processor module accesses the user's preferences for the application in the User class on OntUbi. These preferences are sorted by utility value. This way, it selects the one with the highest utility value to the user, from the possible adaptive actions inferred in the previous module.

The information necessary for adaptation action, obtained from the previous module, is provided by Inferred Adaptation Provider module. The implementation of the adaptation is inserted into the code of the running component. At this moment, the *Executor Service* of the EXEHDA access the information for adaptation execution.

Semantic Processor module accesses, instantiates, and infers the information needed to compute the adaptation rules. The product of these rules is delivered to the mechanism that makes the retention of the adaptation decisions for the applications components.

5.3 Integration with Other EXEHDA Middleware Services

The generation of the context state information, which guides many of the middleware operations and also the application adaptive behavior, is accomplished by the EXEHDA *Context Recognition Subsystem*, through the cooperative operation of the *Monitor*, *Collector* and *Context Manager* services. The produced context state information feeds both functional and non-functional adaptation processes, which are managed by the *DA Service*.

When a new component is instantiated through the use of the *Executor Service*, other middleware services are activated. Even though the component being instantiated may not be adaptive itself, at least non-functional adaptations would take place. Thus, information about the current context state becomes necessary.

As previously stated in this article, the adaptation model is collaborative. Such a collaborative adaptation process occurs in two forms: (i) adaptation commands, by explicit calls to some of the middleware services, and (ii) adaptation policies, which implicitly guide middleware operations. Adaptation policies are managed by EXEHDA *DA Service* in form of OWL documents, instantiated from the Adaptation Policy Ontology and deployed together with the application code when it is installed in the BDA ubiquitous repository.

The *adaptation policy* of an application is defined in the *DA Service*. The adaptation policy defines not only the requirements for the resources to be allocated in the application, but also affinity criteria among the components and between the components and the external resources used. The *DA Service* combines these abstract definitions with run-time gathered information, obtained from the Context Recognition Subsystem, when deciding on component placement. This service also negotiates resource allocation with the *Resource Broker Service*.

Whenever a component migrates, the *DA Service* re-evaluates the affinity criteria defined for that component, triggering the migration of other components when necessary, thus contributing for the maintenance of the application cohesion and implementation of the follow-me semantics.

Furthermore, the functional adaptations, also managed by the *DA Service*, receive meta-data generated at development-time in the form of *adaptation policies*. In this case, an adaptation policy binds component implementation to specific context element states. The *DA Service* is activated when an adaptive component is instantiated or restored after a migration, in order to select the proper implementation for the current execution context state. Additionally, the *DA Service* also takes care of the management of the run-time functional adaptations. This is accomplished by notifying the adaptive components when a context element for which they registered interest has changed its state. The information about the current state of the execution context used by the *DA Service* is provided by the Context Recognition Subsystem.

6 Case Study

Since the middleware is constantly under modeling and implementation, we are using a case study strategy to evaluate the middleware services that were already developed. The strategy employed in this evaluation uses ubiquitous applications specially devised to assess the *DA Service*.

In the adaptation control model we can configure specific rules to the adaptation control model. This allows its use in multiple scenarios, as well as in different context domains. In this article, we employ an application that allows functional adaptation in medical area.

The application developed uses the follow-me semantics and the context-aware adaptation, provided by the EXEHDA services, particularly the *DA Service* described in section 5. We defend a model of collaborative adaptation between application code and the execution system, as well as a semantic-oriented approach for dynamic adaptation. In that sense, the application was modeled considering these principles.

6.1 Ubiquitous Monitoring of Patients

The Ubiquitous Monitoring of Patients is an application targeted to the medical area. The functions are designed to explore the context-aware adaptation control promoted by the *DA Service*, considering the demands of ubiquitous behavior of this application.

The application uses a database created specifically for this case study. The values, alert levels, rules, and parameter adaptation are approximate averages, with no commitment to translate an accurate medical data. This is necessary to manipulate the data during the tests, avoiding the incursion at ethical aspects.

The proposition in this case study is to improve the monitoring of patients that require continuous observation, avoiding the need of admitting then in an Intensive Care Unit. The main objectives of this proposition are: (i) to show patient data, acquired dynamically using a mechanism of signals sensing; (ii) to present the different alert levels to health workers, such as physicians and nurses, in an automated way, based on the sensed data; (iii) to integrate the alerts service application in the open communication network (Google Talk); (iv) to provide access from both mobile devices and desktops; and (v) to enable health workers ubiquitous access to the history of sensed data of the patients.

The health workers can run two modules: (i) dynamic capture of patients' vital signs; and (ii) history of patients' alerts. The signals considered in this version are heart rate, blood pressure and temperature. Different alert levels to health workers are produced according to the values of the signal data. On the other hand, the history includes the date and time of the alert, its level, and the values of vital signs. This information assists the physician's decision-making regarding the patient's clinical status.

The application is designed to explore functional adaptations, managed by the *DA Service*. The adaptations are specified in the ontology OntAdapt, considering two situations: (i) the context data sensed from the patient (vital signs), determining the software component used to display the alert level; and (ii) the device (desktop, smartphone or tablet) used by the health workers, selecting the component with the most appropriate interface to the device.

To build the adaptation policy, considering the specifications provided by health workers, the application developer uses the framework FWADAPT, instantiating classes and relationships of the ontology OntAdapt, with their parameters and adaptation rules. The application is instantiated in the Application class and its software components in the Component classes of OntAdapt ontology. Furthermore, the components' adapters are instantiated in the ontology classes Adapter, Operation, Param_Type, Param_Value, and Sensor.

Figure 6 shows the values associated with the properties of the class Application and the software components instantiated in the class Component.

The software component "Patient Monitoring (500)" can adapt to the component "501" to "514", depending on vital signs collected and the type of device used by health worker. The software component "500" is associated with adapters "Automatic Alert" and "Device Type", as Figure 7. These adapters are related to certain types of parameters, for example, the Figure 8 shows the parameters related to the adapter "Automatic Alert".

Listing 1 shows an instance of the class "Adapter", in which the rule for determining the adapter "Automatic Alert"is specified. This rule uses information instantiated in the ontology "OntContext" by the Context Recognition Service and in the ontology "OntAdapt" by the framework FWADAPT. The module "Semantic Processor" handles this rule using SPARQL (http://jena.apache.org/tutorials/sparql.html) and Jena API (http://jena.apache.org), and the result corresponds to the adaptive soft-

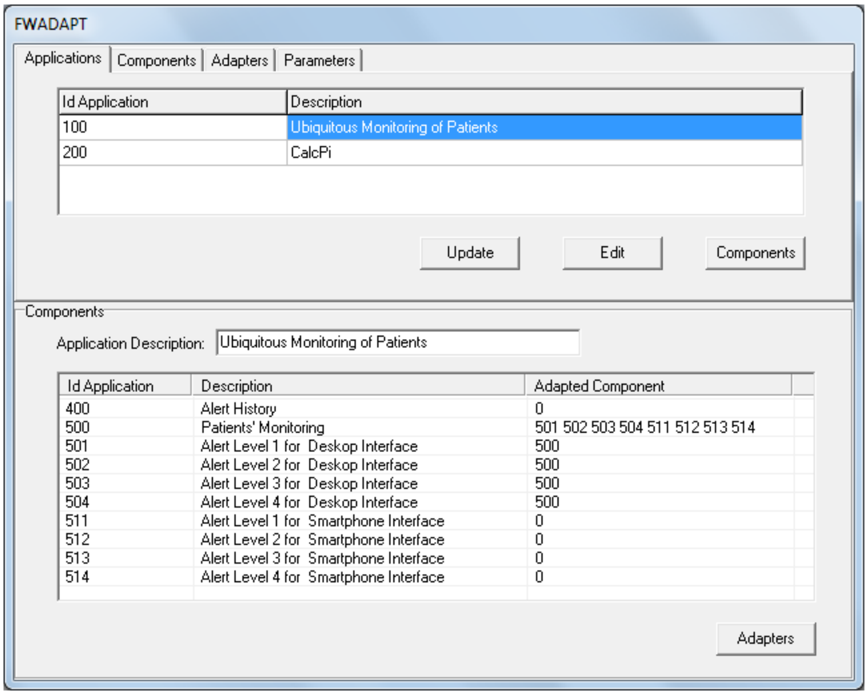

Figure 6: Application Components

ware component relative to the alert level.

The devices considered in the adaptive process are smartphones and desktops. Figure 9 shows the access to the "Alert History of the Patient", one of the features planned for the application both in smartphones and desktops.

Listing 1: Class "Adapter"

```
Adapter_Id = 1
Adapter_Desc = Automatic Alert
Adapter_LogicalRule = (ROUND ((CN_ContextNotifiedSensor.CN_SensorTrad (CN_Sensor.
    Sensor_Id = 100) * ParamType_ParamValue.ParamValue_Utility (ParamType_ParamValue
    .ParamType_Id = 001) + CN_ContextNotifiedSensor.CN_SensorTrad (CN_Sensor.
    Sensor_Id = 101) * ParamType_ParamValue.ParamValue_Utility (ParamType_ParamValue
    .ParamType_Id = 002) + CN_ContextNotifiedSensor.CN_SensorTrad (CN_Sensor.
    Sensor_Id = 102) * ParamType_ParamValue.ParamValue_Utility (ParamType_ParamValue
    .ParamType_Id = 003) + 0,9) + Context_Notified.CN_ComponentId
```

The automatic alert levels are defined by physicians for patients. The *DA Service* runs the adaptation procedures using the following levels:

– "Automatic Alert" Level 1: normal signs. Application with identification and patient vital signs: name, specialty, heart rate (HR), temperature (T), and blood pressure (BP). Interface Level 1 (green);

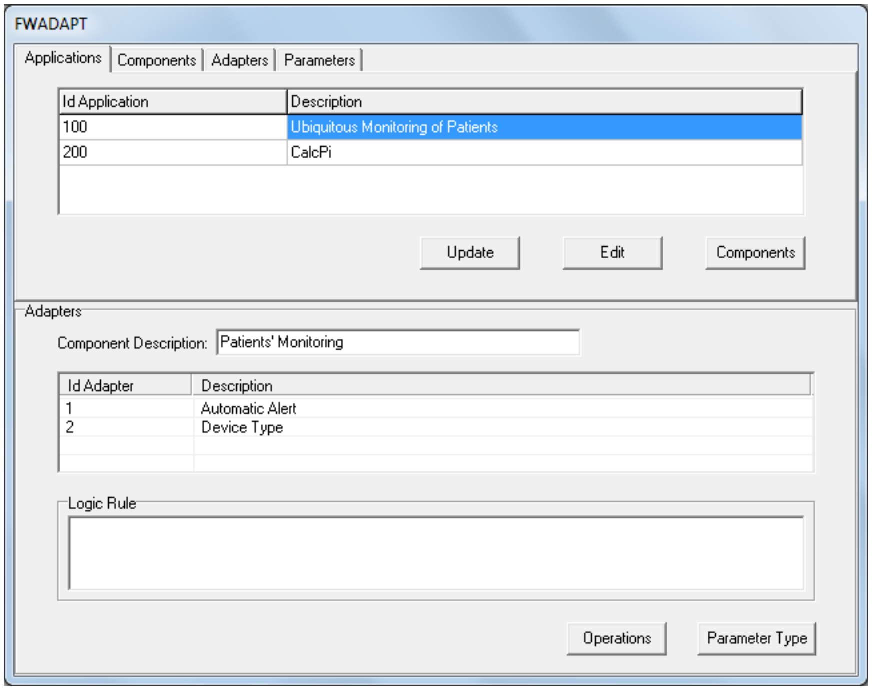

Figure 7: Component Adapters

- "Automatic Alert" Level 2: some signs are outside normal range (HR or T). Interface Level 2 (yellow);
- "Automatic Alert" Level 3: medium problem. Just BP is outside the normal or HR and T outside the normal. Interface Level 3 (orange) and gtalk message to nurse;
- "Automatic Alert" Level 4: maximum alert. BP is outside the normal, T and/or HR are outside the normal. Interface Level 4 (red) with gtalk message to nurse and message delivery option for physicians.

The adaptations of the application are chained: the decision of the adapter "Device Type" uses the decision previously taken with the adapter "Alert Level". Figure 10 shows the Alert Level 4 for desktop and also smartphone.

6.2 Evaluation

In this section we present the experiment details and the results obtained with the application's evaluation. The main target is to assess the effective impact of using dynamic adaptation in the application usability.

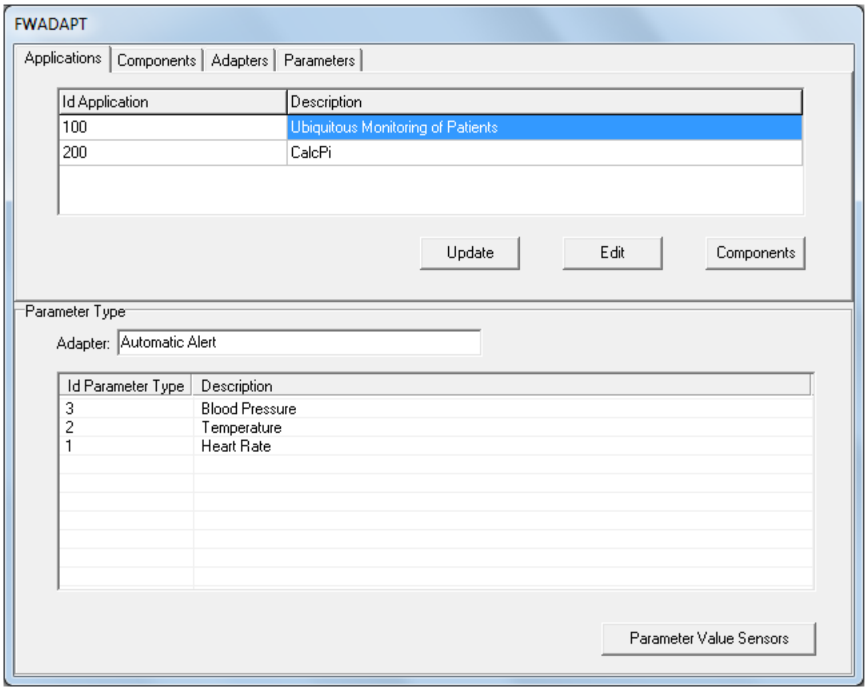

Figure 8: Adapter "Automatic Alert"

The evaluation regards the application acceptance, which involved São Francisco Hospital (a hospital that belongs to the Catholic University of Pelotas, Brazil) volunteer users, among physicians and nurses. These volunteers answered a questionnaire, after they used the application.

For the study we considered 10 physicians and 20 nurses, who were selected based on their activities in the hospital. Each participant used a mobile device (smartphone) and a desktop, with the application installed. We performed a basic training on the application operation beforehand. Participants were asked to use the application and respond to an evaluation questionnaire regarding the experience in the use of the system.

The answers should be within a range of five points (http://psycnet.apa.org/psycinfo/1933-01885-001), ranging from 1 point (totally disagree) to 5 points (totally agree). To evaluate the model acceptability, checking the system usability, the questionnaire were defined based on the Technology Acceptance Model [Yoon and Kim 2007]. The TAM model considers the following main themes for application acceptance: (i) Ease of Use: means the degree in which users evaluating the application may reduce their effort; (ii) Usefulness: means the degree in which users evaluating the application may

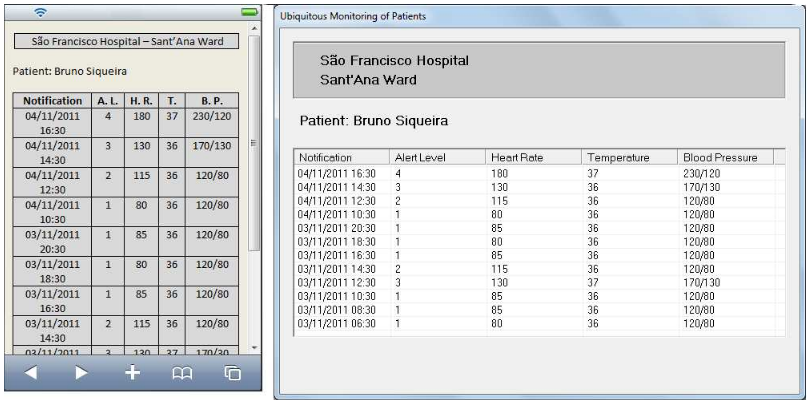

Figure 9: Alert History (smartphone and desktop versions)

improve their performance.

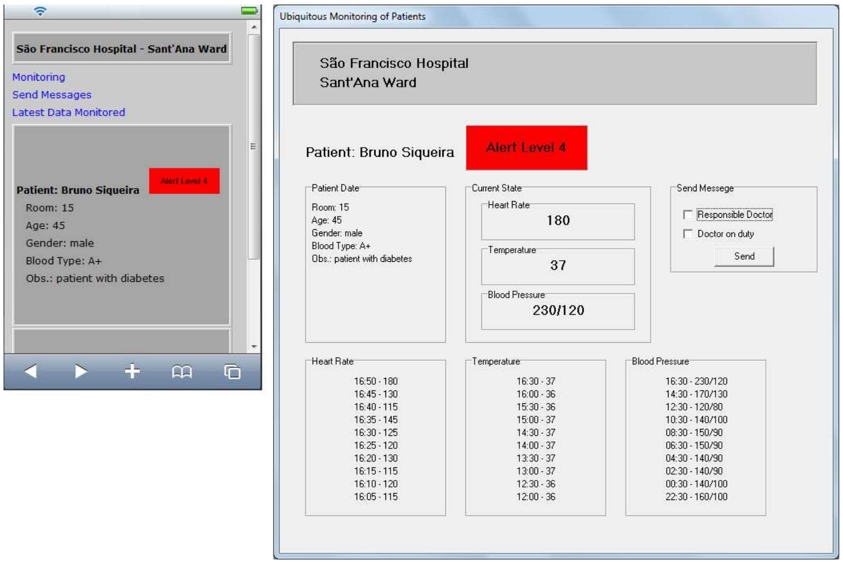

Figure 10: Automatic Alert Level 4 (Smartphone and Desktop)

Table 1 and Table 2 contain the questionnaire applied to users, and answers obtained. The questions were designed in order to be simple, short and direct. Table 1 presents the questionnaire with answers regarding Ease of Use with the following guidance: “Regarding the ease of use of the application, tell us in which degree you agree with the following statements". The table presents the question at the first column and the percentage obtained with the number of users in brackets following from "Totally Disagree" to "Totally Agree". The last column shows the consolidation average percentage score obtained by the responses, which varied between zero and five.

Table 1: Ease of Use Evaluation

Question	Totally Disagree	Partially Disagree	Neutral	Partially Agree	Totally Agree	Average
1. The application is easy to understand.	0,0%(0)	0,0%(0)	0,0%(0)	30,0%(9)	70,0%(21)	4,7
2. The application is easy to use.	0,0%(0)	0,0%(0)	0,0%(0)	30,0%(9)	70,0%(21)	4,7
3. With little effort I can make the dynamic capture of patients' vital signs.	0,0%(0)	0,0%(0)	0,0%(0)	20,0%(6)	80,0%(24)	4,8
4. With little effort I can access the history of patients' alerts.	0,0%(0)	0,0%(0)	0,0%(0)	20,0%(6)	80,0%(24)	4,8
5. The application interface is properly adapted to the device (mobile or desktop).	0,0%(0)	0,0%(0)	10,0%(3)	20,0%(6)	70,0%(21)	4,6

Analyzing the results we can observe that approvals were higher regarding ease of use. In general most people approved the application for this requirement, since average grades were high, up to four (4.5), which means approval of more than ninety percent (90%).

Table 2 presents the questionnaire with the user's answers on the subject usefulness, with the following guidance: "Regarding the application usefulness, tell us in which degree you agree with the following statements". The last two questions were posed in order to determine the true intent of the evaluator in using the application. We could verify that the approval was also high (over eighty percent), but with values slightly lower than the previous questions. This assessment is probably not due to usability issues, since they have high grades, but because of other users concerns such as security and privacy, regarding the use in a hospital. In that sense, security and privacy are being addressed in the software architecture trough a support for the data be transferred in encrypted way. So, the unauthorized access to data is avoided.

Considering the relevance of this issue for the applications' domain like medical area, others ongoing work in our research group are specifically dealing with security and privacy and must improve these aspects in the software architecture [Machado 2013] [Almeida 2013].

Finally, analyzing all the results we can see that in general the users found the application easy to use. Regarding the perception that these users had about the application usefulness, they also considered that application would be useful for health workers at a hospital.

7 Related Work

CARE [Agostini et al. 2009] is a middleware to support context adaptation. CARE uses policies expressed through rules to define how context data should be derived, indicating the reasoning that could be used. CARE has a narrower focus, including the adaptation of usual services of Internet in the context of mobile computing. In turn, EXEHDA has a comprehensive focus, handling the functional and non-funcional adaptation of mobile, context-aware, and ubiquitous applications. Differently from CARE, EXEHDA employs utility functions, besides rules, to select the best adaptation.

Table 2: Usefulness Evaluation

Question	Totally Disagree	Partially Disagree	Neutral	Partially Agree	Totally Agree	Average
1. The options presented are relevant.	0,0%(0)	0,0%(0)	0,0%(0)	40,0%(12)	60,0%(18)	4,6
2. The application makes it easy to do the dynamic capture of patients' vital signs.	0,0%(0)	0,0%(0)	0,0%(0)	40,0%(12)	60,0%(18)	4,6
3. The application makes it easy to access the history of patients' alerts.	0,0%(0)	0,0%(0)	0,0%(0)	30,0%(9)	70,0%(21)	4,7
4. The application is useful for a hospital.	0,0%(0)	0,0%(0)	20,0%(6)	30,0%(9)	50,0%(15)	4,3
5. I would use this application in my work at hospital.	0,0%(0)	0,0%(0)	30,0%(9)	20,0%(6)	50,0%(15)	4,2

WComp [Ferry et al. 2013] is a middleware for ubiquitous computing, based on a software infrastructure, an architecture for service composition, and a mechanism for adaptation. The proposal has some limitations in dealing with context adaptation. The main constraint is not to define an expressive model for the representation of contextual information. In turn, EXEHDA defines a semantic model, based on ontologies, which improves the expressiveness of contextual information for the context adaptation process.

Rainbow [Garlan et al. 2009] consists of a framework, a language and an incremental process engineering of self-adaptation. The adaptation approach, proposed in Rainbow, is similar to that used in EXEHDA, regarding the separation of the adaptation from the logical application. However, Rainbow does not focus on the requirements relevant to environments with mobile devices. Moreover, the strategies for adaptation are based on situation-action rules, which specify exactly what to do in certain situations. EXEHDA, on the other hand, uses the extended objectives policy expressed as utility functions, which is a higher-level specification for adaptation strategies.

Madam [Geihs et al. 2009] is a European research project with partners in industry and universities. Madam, in addition to a middleware, includes a methodology for model-driven development, based on adaptation models and the corresponding changes model-to-code. EXEHDA does not focus on software development tools, but rather in the provision of support for adaptations that can be used by components of the applications. Another consideration is that Madam does not use semantic modeling for adaptation policy application. The proposal does not explicate whether adaptation policies can be reused for further adaptations or other software components.

Proteus [Toninelli et al. 2009] employs a semantic model in the adaptation management of the users' access. Proteus executes adaptations in the access policies to resources, used by applications. In the management of the adaptive process, similar to EXEHDA, Proteus contemplates the use of a semantic model. However, EXEHDA provides a wider range of adaptations. Different functionalities of the applications, with different natures, are involved in the adaptive process.

SECAS [Chaari et al. 2009] is a project that deals with the context adaptation, considering the preferences of the user, environment, and devices involved. SECAS covers only the functional adaptations for healthcare applications. The proposal uses logical expressions to context settings for adapters. Unlike EXEHDA, SECAS does not use semantic modeling for adaptation rules. However, SECAS employs a formalism based on Petri networks.

Table 3 resumes the comparison among the proposals. The features considered in the comparison correspond to functionalities used in the modeling of the EXEHDA's Dynamic Adaptation Control Service: (i) functional adaptation of the application and/or middleware, (ii) non-functional adaptation, (iii) external control of adaptation, (iv) semantic modeling for the adaptation policy, (v) mobile devices, (vi) reuse policies from a catalog, (vii) autonomic rule-based treatment of the adaptation, (viii) utility function.

We consider that using utility functions provide greater adaptive capabilities to user needs. This feature distinguishes EXEHDA from most of the related work. Moreover, EXEHDA distinguishes from most of the researched projects, because it provides a semantic model that increases the expressiveness of context representation, as well as addressing both functional and non-functional adaptations.

EXEHDA allows, at any time, that the application developer includes new

Table 3: Comparison among the proposals

	CARE	WComp	Rainbow	Madam	Proteus	SECAS
Functional adaptation of the application and/or middleware	yes	yes	yes	yes	yes	yes
Non-functional adaptation	yes	yes	no	yes	no	no
External control of adaptation	yes	yes	yes	yes	yes	yes
Semantic modeling for the adaptation policy	yes	no	no	no	yes	no
Mobile devices	yes	yes	no	yes	no	yes
Reuse policies from a catalog	no	no	yes	no	no	no
Autonomic rule-based treatment of the adaptation	yes	yes	yes	yes	yes	yes
Utility function	no	no	no	yes	no	no

instances of classes in the adaptation policy ontology, such as new adapters, and new rules. This is due to the fact that application policy can be maintained externally, describing the configuration of the profiles of the applications, through the rules, parameters and operations of its adaptation policy.

8 Conclusion

In this article we presented the EXEHDA middleware, highlighting the Dynamic Adaptation Control Service (*DA Service*). This service enables the instantiation of adaptation policy, externally to the application code, using a framework. The proposal eases the development and the inclusion of new adaptations, new contexts of interest, new rules, and operating parameters.

The semantic model introduces aspects related to formalism, expressiveness, possibility of dynamic relationships, and inference between information that allow the definition of an adaptation policy model in high level. Furthermore, this model allows the maintenance, reuse, standardization, and sharing of information maintained in the ontological model among the various services of the middleware.

The proposed adaptation model can be used at runtime by applications and by the middleware. Unlike most of related work, which deal only with one type of dynamic adaptation, EXEHDA model supports both functional and non-functional adaptations. The adaptation management is proactive, because it can be started at any time and without user intervention in response to changes in context.

Adaptation policies are expressed using utility functions and promote an adaptation, which may be composed by several others (compositional way), targeted to the components of the application software. We consider that the utility functions can provide a better tuning in the adaptation process, better addressing users' needs. This aspect also distinguishes our proposal from the related works.

Among others, the following aspects should be considered in future works: (i) to implement adaptation rules that modify the parameters of other rules, characterizing a context adaptation of their own middleware; and (ii) to consider the historical context and the history of adaptations in the adaptive decision.

References

[Agostini et al. 2009] Agostini, A., Bettini, C., and Riboni, D. (2009). Hybrid reasoning in the care middleware for context awareness. *Int. J. Web Eng. Technol.*, 5(1):3–23.

[Almeida 2013] Almeida, R. (2013). Segurança da informação e gerenciamento de eventos: Uma abordagem explorando consciência de situação. Trabalho acadêmico, Bacharelado em Ciência da Computação - Universidade Federal de Pelotas.

[Augustin et al. 2005] Augustin, I., Yamin, A., and Geyer, C. F. R. (2005). Managing the follow-me semantics to build large-scale pervasive applications. In *Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, MPAC '05, pages 1–8, New York, NY, USA. ACM.

[Augustin et al. 2008] Augustin, I., Yamin, A. C., and Silva, L. C. d. (2008). Building a smart environment at large-scale with a pervasive grid middleware. In Wong, J., editor, *Grid Computing Research Progress*, volume 1, chapter 10, pages 323–344. Nova Science, New York, NY, USA.

[Baldauf et al. 2007] Baldauf, M., Dustdar, S., and Rosenberg, F. (2007). A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.*, 2(4):263–277.

[Bellavista et al. 2012] Bellavista, P., Corradi, A., Fanelli, M., and Foschini, L. (2012). A survey of context data distribution for mobile ubiquitous systems. *ACM Comput. Surv.*, 44(4):24:1–24:45.

[Bettini et al. 2010] Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., and Riboni, D. (2010). A survey of context modelling and reasoning techniques. *Pervasive Mob. Comput.*, 6(2):161–180.

[Caceres and Friday 2012] Caceres, R. and Friday, A. (2012). Ubicomp systems at 20: Progress, opportunities, and challenges. *Pervasive Computing, IEEE*, 11(1):14–21.

[Chaari et al. 2009] Chaari, T., Zouari, M., and Laforest, F. (2009). Ontology based context-aware adaptation approach. In Stojanovico, D., editor, *Context-Aware Mobile and Ubiquitous Computing for Enhanced Usability: Adaptive Technologies and Applications*, volume 1, chapter 2, pages 26–58. IGI Publishing.

[Costa et al. 2010] Costa, C. A., Silva, L. C. d., Barbosa, J. L. V., Yamin, A. C., and Geyer, C. F. R. (2010). A primer of ubiquitous computing challenges and trends. In Neto, F. M. M. and Neto, P. F. R., editors, *Designing Solutions-Based Ubiquitous and Pervasive Computing: New Issues and Trends*, volume 1, chapter 15, pages 282–303. IGI Global Publishing, Hershey.

[Costa et al. 2008] Costa, C. A., Yamin, A. C., and Geyer, C. F. R. (2008). Toward a general software infrastructure for ubiquitous computing. *IEEE Pervasive Computing*, 7(1):64–73.

[Da et al. 2011] Da, K., Dalmau, M., and Roose, P. (2011). A survey of adaptation systems. *International Journal on Internet and Distributed Computing Systems*, 2(1):1–18.

[Ferry et al. 2013] Ferry, N., Hourdin, V., Lavirotte, S., Rey, G., Riveill, M., and Tigli, J.-Y. (2013). Wcomp, middleware for ubiquitous computing and system focused adaptation. In Calvary, G., Delot, T., Sedes, F., and Tigli, J.-Y., editors, *Computer Science and Ambient Intelligence*, volume 1, chapter 6, pages 89–120. ISTE Ltd and Wiley Sons Inc.

[Fujii and Suda 2009] Fujii, K. and Suda, T. (2009). Semantics-based context-aware dynamic service composition. *ACM Trans. Auton. Adapt. Syst.*, 4:12:1–12:31.

[Garlan et al. 2009] Garlan, D., Schmerl, B., and Cheng, S.-W. (2009). Software architecture-based self-adaptation. In Zhang, Y., Yang, L. T., and Denko, M. K., editors, *Autonomic Computing and Networking*, pages 31–55. Springer US.

[Geihs et al. 2009] Geihs, K., Barone, P., Eliassen, F., Floch, J., Fricke, R., Gjørven, E., Hallsteinsen, S., Horn, G., Khan, M. U., Mamelli, A., Papadopoulos, G. A., Paspallis, N., Reichle, R., and Stav, E. (2009). A comprehensive solution for application-level adaptation. *Software Practice & Experience*, 39(4):385–422.

[Kakousis et al. 2010] Kakousis, K., Paspallis, N., and Papadopoulos, G. A. (2010). A survey of software adaptation in mobile and ubiquitous computing. *Enterprise Information Systems*, 4(4):355–389.

[Knappmeyer et al. 2013] Knappmeyer, M., Kiani, S., Reetz, E., Baker, N., and Tonjes, R. (2013). Survey of context provisioning middleware. *Communications Surveys Tutorials, IEEE*, 15(3):1492–1519.

[Lopes et al. 2007] Lopes, J. L. B., Pilla, M. L., and Yamin, A. C. (2007). Exehda: a middleware for complex, heterogeneous and distributed applications. *Iberian-American Conference on Technology Innovation and Strategic Areas*.

[Machado 2013] Machado, R. (2013). Loga-dm: uma abordagem de análise dinâmica de log com base em mineração de dados. Trabalho acadêmico, Bacharelado em Ciência da Computação - Universidade Federal de Pelotas.

[Toninelli et al. 2009] Toninelli, A., Corradi, A., and Montanari, R. (2009). A quality of context-aware approach to access control in pervasive environments. In Stojanovico, D., editor, *MobileWireless Middleware, Operating Systems, and Applications*, volume 7, chapter 18, pages 236–251. Springer Berlin Heidelberg.

[Warken et al. 2010] Warken, N., Venecian, L., Rodrigues, S., Dilli, R., Lopes, J. L. B., and Yamin, A. (2010). A model of autonomous control of the adaptation to the context in ubicomp. *11th Symposium on Computing Systems - WSCAD-SSC - IEEE Computer Society*, 0:104–111.

[Yamin et al. 2005a] Yamin, A. C., Augustin, I., Barbosa, J., da Silva, L. C., Real, R. A., Filho, A. S., and Geyer, C. F. R. (2005a). Exehda: Adaptive middleware for building a pervasive grid environment. *Frontiers in Artificial Intelligence and Applications - Self-Organization and Autonomic Informatics*, 135:203–219.

[Yamin et al. 2005b] Yamin, A. C., Augustin, I., da Silva, L. C., and Geyer, C. F. R. (2005b). Exehda middleware: Aspects to manage the isam pervasive environment. *XXV International Conference of the Chilean Computer Science Society*.

[Yoon and Kim 2007] Yoon, C. and Kim, S. (2007). Convenience and tam in a ubiquitous computing environment: The case of wireless lan. *Electron. Commer. Rec. Appl.*, 6(1):102–112.