DEPLOYMENT OF COMPONENT-BASED APPLICATIONS ON TOP OF A CONTEXT-AWARE MIDDLEWARE

Dhouha Ayed and Nabiha Belhanafi
Computer science department
GET-INT CNRS UMR SAMOVAR 5157
Institut National des Télécommunications
9, rue Charles Fourier, Evry 91011, France
email: \{Dhouha.Ayed, Nabiha.Belhanafi\} @int-evry.fr

Chantal Taconet and Guy Bernard
Computer science department
GET-INT CNRS UMR SAMOVAR 5157
Institut National des Télécommunications
9, rue Charles Fourier, Evry 91011, France
email: \{Chantal.Taconet, Guy.Bernard\} @int-evry.fr

ABSTRACT
With the expansion of wireless communication and mobile hand-held devices, mobile applications are required to react to frequent changes in the environment, such as new geographical location and high variability of network bandwidth. This paper describes the installation of a deployment service on top of a context-aware middleware. This latter enables interaction with context by providing a meta-model of the context information and a set of components able to collect and analyse the context information. The deployment service has to describe its relevant context using the meta-model provided by this middleware. When relevant changes occur, deployment service is notified in order to apply its adaptation process.

KEY WORDS
Middleware, context-awareness, dynamic deployment, component-based applications, adaptation, metamodelisation.

1 Introduction

With recent advances in wireless networking technologies and mobile computing devices, making applications context-aware becomes essential. This is due to the scarce resources of these new technologies (e.g. memory, battery, CPU) and the variability of the application execution context (e.g. user location, device screen size) [1].

To achieve such level of awareness, application developer has to periodically query heterogeneous physical or logical sensors, represents context information, and adapt the application to context changes. As these are complex tasks for the developer, using a middleware upon the network operating system is necessary. This middleware automates all tasks dealing with context.

Context-aware deployment service which consists in installing, configuring, activating, reconfiguring, updating and deinstalling the software according to context is a particular type of context-aware service which has to be processed just-in-time (i.e. when a user accesses to a service). Just-in-time deployment enables users to automatically install and configure applications suited to their needs, the resources of their computers and their surrounding environments. It implies the automation of the deployment tasks and thus relieves the user from the deployment repetitive tasks. The context-aware deployment service is a part of the AMPROS project which aims to study and develop a middleware-based platform for wireless networks and validate it in the field of emergency aid and crisis management.

The purpose of this paper is to describe a deployment service placed on a context-aware middleware. This service enables the deployment of component-based applications for mobile users confronted with deploying applications in different contexts. Section 2 presents our motivations and introduces an adaptative deployment service of component-based applications on top of a context-aware middleware. The architecture of this context-aware middleware is presented in Section 3 and the structure of the components enabling the deployment adaptation will be described in Section 4. Section 5 discusses some related work and Section 6 concludes the paper.

2 Motivations

In order to adapt an application to the context, we follow three main steps: (1) The collection of context information. (2) The analysis of context information and decision of the adaptation actions. (3) Triggering the decided adaptation actions.

Dealing with direct access to context information during the development of a context-aware service or application is expensive, error-prone, and lead to complex and non-portable applications. That is the reason why we support the idea of separating context-aware infrastructure from context-aware applications. The context-aware infrastructure deals with activities like communicating with context sources, collecting context data, storing and managing context data, and finally transforming context data into high level context and refine them according to the applications needs.

The context-aware infrastructure can be a context-aware middleware that is responsible for collecting, delivering, and managing context on behalf of context-aware applications. It provides abstractions for the fusion of sensor information to obtain high-level context information whereas the context-aware applications are responsible for
adaptation and reaction to context changes.

Our context-aware deployment service is a middleware service that deploys context-aware applications according to their current context. In order to follow the steps described in the beginning of this section, we place our deployment service on a context-aware middleware, which carries out the collection and the analysis of context information. The adaptation of the deployment process will be carried out by a DeploymentAdapter component that we add to a Deployer component which represents a classical deployment tool (cf. Figure 1). The architecture of the context-aware middleware is presented in Section 3 and the structure of the DeploymentAdapter is described in Section 4.

Figure 1. Architecture Overview

3 Component-based Middleware for Context-aware Services and Applications

The purpose of this section is to describe the infrastructure of our component-based context-aware middleware. This middleware allows designers to build context-aware applications and to interact with context-aware services by providing a meta-model for describing context and adaptation policies; The middleware communicates with the underlying execution environment to collect context information, processes them to identify relevant changes, and propagates those changes up to the context-aware services or adapt the applications to context changes.

The architecture overview of this middleware is described in Section 3.1, then the context meta-model that it provides is specified in Section 3.2; In this latter we focus on the use of the meta-model to install context-aware services on top of the described middleware.

3.1 Middleware Architecture Overview

The component-based middleware for context-aware services and applications enables gathering context information by interacting with sensors, and interpreting the collected data. A sensor is a software or a hardware device used to measure a physical quantity such as temperature, location or battery power. Each sensor is responsible for collecting a type of context information. As showed in Figure 2, this middleware includes different components:

The CollectionManager is responsible for collecting context information by interacting with sensors through agents. An agent is an entity allowing interaction with a sensor type to collect data from it, the CollectionManager has to activate and deactivate agents according to the context to which applications or services are sensitive.

The ContextAnalyser has to filter context information to determine relevant ones, and notify the subscribed component of these context changes. Context filtering consists in detecting context changes by comparing a context value saved in the ContextRepository with its new context value. If a change occurs, the ContextAnalyser verifies if this new value is relevant for a service component or an application, then saves it in the ContextRepository.

The ContextInterpreter abstracts raw or low level context information into richer or higher level information according to interpretation rules described by using the context meta-model provided by the middleware (cf. Section 3.2 and Section 4.1).

The ComponentAdapter is responsible for adapting application’s component to context changes after being notified by the ContextAnalyser, according to the adaptation rules defined by application designers.

If a component belonging to a service is sensitive to a given context, it subscribes itself to the ContextAnalyser in order to be notified about a specific context changes, the CollectionManager activates agents, which are not activated yet, responsible for collecting context information,
then the following process is executed for each context type. (1) Each agent collects context information. (2) CollectionManager sends context value to the ContextAnalyser and the ContextInterpreter. (3) The ContextInterpreter interprets high level context information from sensed information and transfers the interpreted data to the ContextAnalyser. (4) The ContextAnalyser filters the sensed and the interpreted data to determine if relevant changes occur for any application or service component, and saves the new value in the ContextRepository. (5) The ContextAnalyser notifies all components and services subscribed to this relevant context. (6) Go to step 1.

With these components, our middleware takes in charge context data collection, interpretation and analysis in order to determine relevant changes. In Section 3.2, we specify how a context-aware service describes context to which it is sensitive, interpretation rules, and the relevant context which requires adaptation.

3.2 The COntext Meta-model for Context-Aware Services

In this section, we describe our context meta-model for context-aware services which allows interaction between the described context-aware middleware and context-aware services. This meta-model called COntext Meta-model for Context-Aware Services (COMCAS) allows description of context to which services are sensitive, interpretation rules, and relevant context which necessitates adaptation. COMCAS is based on an ontology description [2], which provides a vocabulary for representing information about context, sensors from which data are collected and interpretation rules for high level context deduction.

The objective of this model is to provide flexibility and extensibility with the possibility to dynamically add specific context for various service domains and new sensors interacting with the middleware. Each service installed on top of our middleware can use the existing descriptions of context or add dynamically new context descriptions.

As shown in Figure 3, COMCAS is divided into four ontology domains related to each other (this ontology is written in the OWL language [2]). For each domain, the designer has to include description of classes, properties, their instances, and the relations which bind them. The Context domain represents context information to which services are sensitive, those context can be direct i.e. data captured from sensors or indirect data deduced from other contexts by using interpretation rules. The Sensor domain allows representation of sensors that the middleware may interact with. The Policies&Rules domain represents all adaptation policies and interpretation methods which can be used by the middleware for adapting a context-aware application to context changes, or interpreting high level context information. The CA Services domain represents all the context-aware services installed on top of the proposed context-aware middleware.

![Figure 3. The Ontology-based COMCAS](image)

For each context-aware service the designer has to describe the context to which the service is sensitive, the relevant contexts and the interpretation rules. This description is done by creating sub classes of Context, Policies & Rules, and CA Services, and describing the following relations which exist between these classes. The HowDeduce relation is a meta-model which binds direct class and the indirect class for describing interpretation rules. Two types of deduction are possible: the built-in deduction which allows computation of the indirect context using calculating methods, these methods have to be described as an instance of the Policies & Rules domain, and the logic deduction which combines different conditions to specify the value of the indirect context. This meta-model allows creation of an interpretation descriptor. The DependsOn relation is a meta-model for binding CA Services domain to context domain for specifying the relevant context to which the context-aware service is sensitive. These services have their meta-model for describing the adaptation process, the DependsOn meta-model allows them to create a relevant context descriptor to specify relevant context to which they are sensitive in order to be notified, by the middleware, when these relevant changes occur. TakesDataFrom relation is a simple OWL ObjectProperty [2] used in order to specify for each context the sensor from which data are collected.

COMCAS allows middleware designers to add any context-aware service on top of our middleware. The interaction between the added context-aware service and this middleware has to be done by describing additional context to which the service is sensitive (direct and indirect context), interpretation rules by describing the HowDeduceProperty, and relevant contexts by describing the DependsOn property. When those relevant contexts occur, the middleware notifies the context-aware service for applying adaptation.

Currently, we have implemented the COMCAS meta-model, and the ContextAnalyser which filters context information and notifies the subscribed component about relevant changes.
Function of the Deployment Adapter on top of a Context-aware Middleware

As described in Section 2, the deployment adaptation is carried out by the DeploymentAdapter component. This component is able to vary four parameters of the application to be deployed:

- The application architecture i.e. the number of the components to be deployed and the connection between them. For example, in the case of AMPROS crisis management, if a rescuer is located in a zone near to a network-disconnected zone, the DeploymentAdapter deploys an additional component on the rescuer terminal which manages the disconnection by allowing the user to work locally and make reconciliation with distant components whenever it is possible to reconnect. The function of this additional component is well described in [3].

- The placement of each component on the nodes of the deployment domain. For example, if the resources of the rescuer terminal are insufficient to install the overall application components, only a part of the components will be installed on the user terminal, the others will be installed on one of the ambulances servers. These servers are selected according to the availability of their resources.

- The choice of one of the implementation versions of each component. This choice depends on the chosen node. Let us take the case of two rescuers, one using a laptop, the other one using a PDA, who want to deploy a graphical user interface component. The DeploymentAdapter deployment will install a normal screen version for the former and a small screen version for the second one.

- The configuration property values of each component. For instance, if a component has a property which represents the user’s language, the value of this property depends on the user’s language used at the deployment time.

To adapt these parameters during the deployment of a given application, the deployment service starts by collecting the relevant contexts filtered by the underlying context-aware middleware then it takes adaptation decisions.

Collection and analysis of context information for the deployment purpose

In order to be able to subscribe for particular relevant contexts and indicate to the underlying middleware how to filter them. Each application to be deployed has a description of contexts to which it is sensitive and the relevant context to be filtered out. This description is specified by the application designer.

Figure 4 shows the description of the "UserConnectionZone" context, which allows the deployment service to decide if a disconnection and reconciliation manager component has to be added to the deployed application. The "UserConnectionZone" is an indirect context deduced from the location context. Location is a direct context sensed from a location sensor. The location sensor is a GPS which determines the user coordinates X and Y. Figure 5 shows how the indirect context can be interpreted from the direct context by describing the interpretation rule. UserConnectionZone is deduced by triggering the determineUserConnectionZone method with the user coordinates as parameters. This method returns three possible values NearDisconnectedZone, DisconnectedZone or ConnectedZone.

Figure 6 presents a description of the determineUserConnectionZone method as an instance of the Policies&Rules domain by specifying the location of the code of this method, its name, the type of its parameters, and the type of the returned value. This method allows the middleware to determine if a user location expressed by its coordinates X and Y is in a network-connected zone, near a network-disconnected zone, or in a network-disconnected zone by using a Data Base where all the network-connected and disconnected zone are stored. Figure 7 presents a description of the relevant context which represents the case where the user is in a network-disconnected zone.
These descriptions allow the Context-Aware Deployment service to interact with the context-aware middleware (cf. Section 3) and to subscribe to the relevant context that it requires. The middleware notifies the DeploymentAdapter when a relevant change occurs to take deployment decisions suited to the context.

4.2 Deployment Adaptation

The DeploymentAdapter collects all the filtered relevant contexts and uses a set of adaptation rules described in the context-aware deployment plan of the application to be deployed in order to take deployment decisions. The context-aware deployment plan structure is well described in [4]. To adapt the four parameters described in Section 4.1, the DeploymentAdapter has three components: The ArchitectureAdapter, the Placement&ImplementationAdapter, and the PropertyConfigurationAdapter.

The ArchitectureAdapter uses the collected relevant contexts to determine which of the component instances and the connections described in the context-aware deployment plan will be deployed.

The Placement&ImplementationAdapter studies the charge state of the deployment domain nodes and determines a placement for each component to be deployed by choosing the implementation version which satisfies the nodes properties. The Placement&ImplementationAdapter uses a placement algorithm which maximises the free nodes' resources and the satisfied user preferences.

The PropertyConfigurationAdapter uses the context-aware deployment descriptor to assign the values suited to the context to the properties of a given component instance.

Whenever a context change occurs that is relevant to an application deployment, the DeploymentAnalyser analyses the change and classifies it by sending a request to one of the three adaptation components.

The DeploymentAdapter uses the deployment decisions taken by its components to create dynamically a deployment plan [5] suited to the context and sends it to a Deployer component which represents a component-based deployment tool. The deployment plan describes the components making up the application assembly and specifies how are these components instantiated, how are they placed and how are they connected to each other. The Deployer will instantiate the components, configure and connect them. The structure of the generated deployment plan depends on the used deployment tool. Currently, we have developed a prototype of our deployment solution on the top of the CCM deployment tool [6] and we intend to extend it in order to support the reconfiguration of the deployed application after context changes.

5 Related Work

5.1 Context-aware middleware

Several research efforts have been addressed for creating context-aware middleware which enables context-aware applications and services development. We can cite SOCAM [7], RCSM [8], and CARISMA [9] projects.

The SOCAM architecture aims to enables rapid prototyping of context-aware services in pervasive computing environment. This middleware takes into account context acquisition and interpretation, it offers an API for context subscription but there is no mean to describe relevant context, so each service has to analyse and to filter the acquired context to detect relevant changes.

The RCSM project provides an Interface Definition Language which generates the code for triggering application action according to developer description. The CAIDL language provides a limited context type description without taking into account context interpretation nor description of relevant context without specifying adaptation actions.

The CARISMA project provides reflexive API for creating context-aware applications, it enables them to deliver the same service in different ways when requested in different context. CARISMA does not provide any interpretation mechanism and adaptation is always carried out by the middleware.

5.2 Context-aware deployment

Various component models were defined, the most prominent ones being .NET[10], Enterprise JavaBeans [11], and the CORBA Component Model [6]. These models are based on component middlewares, which do not manage context information. All of them provide a static deployment tool which deploys applications whose structure is fixed for any context and the placements of the components have to be imagined before the deployment of the application without studying the nodes capacity or fixed by the user manually at the deployment time.
The OMG Specification of the Deployment and Configuration [5] and the deployment tool of COACH [12] support only the context which presents the deployment target environment and do not consider the context in general case.

6 Conclusion

Component-based software technology is becoming an increasingly popular approach. The objective of this technology is to take elements from a collection of reusable software components (i.e., components-off-the-shelf) and build applications by simply plugging them together. It allows the applications to be adapted to changing requirements more easily than using traditional approaches by reconfiguring components, adapting existing components, or introducing new components.

The expansion of wireless communication and mobile hand-held devices has motivated us to take into account the context information and introduce it in the component-based middleware level in order to be able to react to frequent changes in the environment, such as new location and high variability of network bandwidth.

In this paper, we proposed a middleware architecture which interact with sensors for collecting context data, interpreting high level context information, and detecting context changes in order to notify the concerned component for applying adaptation policies. We proposed also a meta-model based on an ontology description. This meta-model provides an efficient support for a dynamic description of context, interpretation rules, and adaptation policies.

We have placed on top of this middleware a deployment service. The context-aware deployment tool proposed in this paper allows the installation and reconfiguration of an application according to the context. Its architecture is based on a set of lightweight adaptive components able to modify several deployment parameters such as the architecture of the application, the placement of the components and their configuration. The adaptive components may be added to existing deployment tools without modifying the tools themselves, as it has been implemented with the CCM deployment tool.

As next steps, we intend to implement the ComponentAdapter, the CollectionManager and the ContextInterpreter for the middleware architecture. For deployment purpose, we intend to look further into the reconfiguration of already deployed applications to improve our implementation by integrating the dependency and consistency managers and to make measurements of the adaptation delays added to the time necessary to deploy an application.

References


