Automatic Resource and Service Management for Ubiquitous Computing Environments

S. Maffioletti  S.Kouadri M. B. Hirsbrunner
DIUF – Department of Informatics
University of Fribourg
1700 Fribourg, Switzerland
Email: \{Sergio.Maffioletti, Soria.KouadriMostefaoui, Beat.Hirsbrunner\}@unifr.ch
http://www.unifr.ch/diuf/pai/
Swiss National Science Foundation Project N. 2000-065301

Abstract

The high degree of dynamism and heterogeneity of the resources involved in a pervasive computing environment makes service adaptation and interoperability a difficult task. This paper presents UBIDEV, a service framework that faces the heterogeneity problem by hiding at the application level the dynamism of the underlying environment. We describe the UBIDEV architecture focusing on the description and the management of services and resources. We also describe how this approach decreases the complexity of the design and development of service-oriented applications. A prototype implementation of a unified messaging system is presented as a validation of the architectural design.

1. Introduction

Pervasive environments are inherently composed of heterogeneous and dynamic resources. Applications are characterised by a large number of services that have to configure the environment and the interaction with other services in order to carry out their execution. Moreover services perceive their environment, usually with the help of sensors, interpret this information and hence derive their model of context, based on their perception. Hence coordinating services in a continuously changing search space becomes a necessary but challenging task.

In a scenario characterized by a high degree of heterogeneity of the underlying resources, it is not possible for applications to rely on common base functionalities offered by a Network OS. What is required is a software infrastructure that defines a suitable coordination model to describe and manage the dynamism of the services composing the application and, at the same time, to relieve the application itself from directly handling the heterogeneity of the underlying environment as well as from maintaining a model of the context.

UBIDEV proposes a context centric management of the environment: is the application context that determines the semantics of resources and services involved as well as the contextual information. Consequently resource configuration, service instantiation description and composition, context model as well as client query solving are based on this semantics. This allows applications to automatically reconfigure themselves according to context changes. UBIDEV represent a unified management model for resources, services and context information, seen at application level as a homogeneous coordination space \[2\].

The rest of this paper is organised as follows: Section 2 briefly introduces the architectural model of UBIDEV. Section 3 presents the functionalities of the Ubiquitous Access layer of UBIDEV. Section 4 illustrates an application example. Section 5 discusses the main differences of UBIDEV with some popular systems. Finally conclusions and future perspectives are given in Section 6.

2. System Overview

UBIDEV is a lightweight infrastructure for pervasive computing environments conceived with the aim of supporting an application to configure and adapt itself to the underlying environment. In doing so it separates the coordination aspects from the resource and service management in order to hide at the application level the heterogeneity of the underlying environment. As a result an application can be described in terms of composition\[1\] of homogeneous services

\[1\] Composition is the process of building complex composite services
as in classical coordination-based distributed systems [4].

A key element in realising this architecture is the use of an application ontology that undergrid the communication and representation. Instead of introducing its own semantics, UBIDEV relies on the ontology defined by the application to determine the internal representation (concepts) of semantics and the relation to the environment (context and resources). That led to small topic-oriented ontology used to classify the whole environment as well as the contextual information. Figure 1 represents the actual implementation of UBIDEV based on such a model.

The Physical Entities layer represents the resources belonging to the environment. Resources describe all the entities that may be involved in the execution process of an application; they describe physical devices, software components and users. In order to maintain a consistent model of the physical dimension for the upper layer modules, the system has to classify these resources monitoring constantly their changes.

The Ubiquitous Access layer represents the core part of UBIDEV; it centralises all functional aspects related to resource and service management; it is also responsible of creating and maintaining a model of the context of the application. It is composed of four main modules as depicted in figure 1: The Resource Manager handles the communication with the underlying resources through standardized interfaces called adapters. It also supplies the corresponding classification that identifies the resource capabilities; in doing so, it relies on the Classifier [12] to create an abstraction of the resources in terms of high level concepts taken from an ontology. Service Manager is responsible for instantiating and monitoring services. It analyses the resource classification in order to bind resources and services according to the service requirements. Context awareness is achieved by the Context Manager that is responsible for gathering, processing and representing contextual information. It also classifies this information into context types. Coordination Manager is responsible for decomposing complex queries coming from the Application Layer in terms of composition of basic services. In doing so it relies on the contextual information provided by the context interface in order to adapt the service composition policy, hence the application behaviour, to the specific context configurations. Next section will give more details about functional aspects related to each module.

The Application layer is a generalisation of the view in a traditional Model Viewer Controller [6]; it works as a presentation module embodying some application dependent functionalities like visualisation that produces the input queries. Application GUI modules, together with the adapter are the only piece of code that is executed on the specific resource according with the application requirements. UBIDEV does not define any particular client-side module; in the UMS example, the Palm interface is realized by a simplified implementation of a remote framebuffer protocol.

3. UBIDEV middleware

This section presents the functional aspects of the four modules composing the Ubiquitous Access layer. The focus is on how resource and service management are faced, and what is the resulting abstraction at the coordination level.

3.1. Resource Manager

Resources have a tight coupling between hardware and software, on one hand because of the limitations of the device itself and on the other hand due to the dependencies of software on specific hardware features like pen-input. Therefore configuring and describing these resources is an important issue.

Resource Configuration: UBIDEV focus resource configuration issue on the federation management aspect; it defines a federation manager that associates to every resource a corresponding adapter. An adapter is considered as a virtual representor of a resource inside a UBIDEV environment; it defines uniform access mechanisms for exchanging

---

2 In our model the user is considered as an ultimate resource involved in the interaction process. As such it could correspond to the lowest level, which is in direct contact with the devices.
data in a seamless way. All resources access requests are managed by the resource manager that address the corresponding adapter. The adapter approach is particularly suitable for integrating handheld devices because it embodies the specific resource access information.

**Resource Description:** UBIDEV shifts the focus of resource description from a resource centric to a context centric. In UBIDEV resources are classified relying on a set of abstract concepts collected in ontology. In UBIDEV the meaning of these concepts is implicitly given by the Classification (figure 1). Classifiers are services that given a resource and ontology, output concepts of that ontology. This basically means that a classifier associates one or more concepts it knows with a resource in a context and tags the resource as an instance of that concept. Classifications of resources are stored and used as a cache when services request an instance of a concept. The process of requesting an instance of a concept is called “addressing by concept”, because the instance is referred by a concept instead of specific resource identification, such as memory address, name or URL. Thanks to the classifier, UBIDEV decouples the high-level concepts (abstractions) from the instances implemented by a context. Similar approach has been considered for semantic service discovery like in [14] or in [1] where a XSB-based engine is used to interpret DAML queries.

The main advantage of this approach in facing resource management problem is that resources selection is based on their semantics

### 3.2. Service Manager

In UBIDEV a service is defined as an atomic action that transforms an input resource yielding a new resource as output. Additionally, a service is constraint by (1) the concept it accepts as argument (input concept) and (2) the concept produced as a result of the action (output concept).

To execute such actions, resources that match the service’s semantics requirement have to be found (like printer for a print service). This requirement represents the service EXecution Environment (EXE) and is expressed by the service itself in terms of concepts the system should provide. The service EXE contains all resource access information the service requires to complete its execution. A run-time instance of a service inside its environment is represented by a homogeneous entity called capsule.

In systems like [10] the infrastructure is aware of the service requirements and discovers resources that match these specifications. However these requirements are expressed in terms of resource capabilities and basically are defined for compatibility reasons (Memory = 20Mb, OS = Solaris).

The main advantage of facing the heterogeneity problem with the capsule abstraction is that UBIDEV can present at application level a homogeneous coordination space seen as an unified mechanism for dynamic communication, coordination, and sharing of objects [4].

### 3.3. Context Manager

There are different types of context that can be used by the application: physical, environmental, personal, social, application, etc. The structure of the context types [5] is defined by the application ontology. Each type of context corresponds to a class in the ontology. The use of ontology to model context is useful for checking the validity of contextual information because of the unified class typing. It also makes it easier to specify the behaviour of context-aware application since the description relies on these context types.

This is similar to what Chen et al. denote in [3] by sensing the contextual information, the acquisition and sharing of contextual information, and reasoning about contextual knowledge. For each of these steps we have associated a computational module, respectively, the context manager, the context database and the context interface (figure 1).

The context manager is responsible for gathering and processing context information and for providing this information to the context database. In order to be used by a context-aware application, this context information needs to be structured according to the application conceptual model of the context. The Classification phase structures this context information according to the application ontology. The result is passed to the context database to be stored as context knowledge.

The context database is acting as a repository; it receives the specifications of the knowledge from the Classification and stores them in a collection of Prolog predicates. The actual implementation of the system makes use of a Logic Programming system engine called XSB [11] to implement the reasoning system that solves the coordination manager queries.

The context interface allows coordination manager to query the context database in order to instantiate the contextual rules that drive the service composition policy.

Using this approach, the coordination manager can retrieve the specific contextual data in a way that is decoupled from the service used for acquiring the data [5].

### 3.4. Coordination Management

The coordination manager acts as a control unit of the whole system. It centralizes the control of the underlying environment by presenting at application level a homogeneous coordination space. It proposes a model of the context structured as an organisational domain composed by autonomous entities. An entity is defined by its structure
obtained by a recursive composition of entities. The application model of the context can be described using this hierarchical organization: an entity hierarchy represents the structure of the context. Each entity can be mapped to concrete context space like a building or a room in the case of the Ubiquitous Message System prototype, or can represent a logical context like the user space. The building blocks of this structure are atomic entities that are in direct correspondence with the underlying resources. Entities can interact with each other through communication endpoints that define a set of actions. In this model a service is described by an active connection between the input and the output entity. The main difference from the classical coordination models is that it defines context-dependent behavioural rules [13]. These rules are applied by the coordination manager to the communication endpoints: a set of rules may determine whether two entities have to be inter-connected with each other or not. Rules describe the actions that should be taken in different contexts. A rule consists of a condition that, when satisfied, leads to a set of actions. Actions are interpreted by coordination manager to drive the service composition strategy and hence the application behaviour.

A concrete example of how actions are composed and controlled is given in the following section.

4. Ubiquitous Message System

Ubiquitous Message System (UMS) is a simplified implementation of commercial unified messaging systems like General Magic Portico [8]. It has been developed to show the added value of an infrastructure like UBIDEV in terms of application design.

Figure 2. Ubiquitous Message System. The application can compose capsules in a more structured entity (document to display).

UMS allows users to exchange messages through a virtual service called document to display without concerning them about how this service is realised in every user context (figure 2). The scenario is characterised by an application context organized as a hierarchy of sub-contexts to model the environment and the users. A portable phone, a mailer, a PDA or the terminal where the user just logged in represents an example of a user context.

Resource management in UMS is based on an event driven mechanism that allows the federation manager to inform its subscribers about all the changes occurring in the federation. The dynamic control of the federation is based on context services instantiated according to the availability of specific resources (like when a IrDA port is plugged or when the Ethernet interface is brought up). Federation manager registers to these services in order to acquire information about the resources (a PDA is authenticated through the IrDA port or by querying the DHCP server on the Ethernet interface).

Service manager reacts to the federation event changes adapting the behaviour of the application in terms of service instance execution. In doing so the coordination manager controls the execution of the capsules embodying the service and the resource reference. Once a service manager is notified of a resource that left the environment, it stops the execution of the corresponding capsules and tries to re-initialise the execution of the service inside a new capsule according to the actual resources classification.

The actual implementation of the Ubiquitous Message System allows the use of PalmOS-based PDA, Zaurus PDA, speakers, any kind of personal phones, fixed phones, fax machines, X terminals or mail clients.

Figure 3. Ubiquitous Message System. the ontology used in the prototype.

Coding UMS in UBIDEV requires to:
- Define the application ontology. An example is shown in figure 3.
- Code the classifiers that allow UBIDEV to tag resources with concepts of the ontology. For example a portable phone will be classified as instance of voice, b/w, display and Resource.
- Code the services and their specification. Services will be automatically instantiated according with the actual configuration of the resources federation. The sound_to_voice service, for example, requires a voice display so when the classifier tag a resource as instance of voice and display con-
cepts, service manager will create a capsule containing the instantiated service and its EXE represented by the access reference of the tagged resource. The bind between a service and a resource is transparent to the service because of the uniform access granted by the adapter and the access protocol. That means a service like ascii_to_textual that requires a textual display can be equally instantiated on a PDA, on a portable phone or on an X term.

- Code the interfaces that will produce the input queries. The only query generated by the actual implementation of UMS is document_to_display. The coordination manager will solve this query inside each user context by composing available instances of services; that means, for example, it can compose ascii_to_wav, wav_to_adpcm and adpcm_to_voice services in order to reach a user context composed of her personal phone (figure 2).

An ongoing work of a UbiDEV compliant coordination models [9], will permit to have a structured composition policy with the introduction of contextual social rules. A typical example of a social rule can be the hierarchical preference of multiple instances of the same service on the basis of user preferences or contextual scenarios (i.e. sms over phone call if the user is in a conference room).

5. Conclusion

This paper has presented how UbiDEV eases the construction and coordination of highly dynamic distributed services without forcing the application to deal with the heterogeneity of the underlying environment. In doing so it relies on the application ontology to describe and manage the underlying environment in terms of resources, services and contextual information. UbiDEV proposes a holistic management of the environment and allows the application to be described in terms of its functional aspects instead of configuration and interaction rules. A prototype of a unified messaging system has been presented in order to underlying the advantages of this system especially for what concerns the application design. UbiDEV is not intended to be a fully open distributed system; it focuses on model the environment to allow an application to configure and to adapt to it. For this reason aspects related to inter-platform and inter-environment interoperations are not directly addressed by the model (a more exhaustive explanation can be found in [7]).

References