# A Component-based Software Infrastructure for Ubiquitous Computing

Areski Flissi, Christophe Gransart, Philippe Merle

# A Component-based Software Infrastructure for Ubiquitous Computing

Areski Flissi
*LIFL / CNRS*
*59655 Villeneuve d'Ascq, France*
*Areski.Flissi@lifl.fr*

Christophe Gransart
*INRETS-LEOST*
*59666 Villeneuve d'Ascq, France*
*Christophe.Gransart@inrets.fr*

Philippe Merle
*INRIA / LIFL*
*59655 Villeneuve d'Ascq, France*
*Philippe.Merle@inria.fr*

## Abstract

*Multiplication of mobile devices and generalized use of wireless networks imply changes on the design and execution of distributed software applications targeting ubiquitous computing. Many strong requirements have to be addressed: heterogeneity and limited resources of wireless networks and mobile devices, networked communications between distributed applications, dynamic discovery and automatic deployment on mobile devices. In this paper, we present a component-based software infrastructure to design, discover, deploy, and execute ubiquitous contextual services, i.e. distributed applications providing services to mobile end-users but only available from a particular place. These ubiquitous contextual services are designed as assemblies of distributed software components. These assemblies are dynamically discovered according to end-users' physical location and device capabilities. Then, appropriate assemblies are automatically deployed on users' devices. We have implemented this approach (the software infrastructure and a ubiquitous application example) on top of the OMG CORBA Component Model and the OpenCCM open source platform.*

## 1. Introduction

With the multiplication of mobile devices and generalized use of wireless networks, the design, implementation, deployment, and execution of distributed software applications must take into account several problems: end-user's mobility, heterogeneity and limited resources of mobile devices and wireless networks, networked communication, dynamic discovery and automatic deployment of distributed software applications. More generally, all these problems are also encountered in any ubiquitous computing context where services provided to end-users must be accessible from anywhere, at anytime, and by anyone [1]. Currently, there is no well established approach to design, implement, deploy, execute, and manage ubiquitous applications [2]. More specifically, such an approach is required for ubiquitous contextual services, i.e. services provided to mobile end-users but only available from a particular place. For instance, a railway operator could want to provide a contextual railway station service allowing mobile end-users to consult from their own device (smartphone, PDA, laptop) the trains' schedule of the railway station. In this context, the key research challenges are firstly to define the appropriate model to design such services and secondly to provide the software infrastructure dedicated to these services and solving the problems related to ubiquitous computing.

The main contribution of this paper is to propose a component-based model and a software infrastructure to design, discover, deploy and execute ubiquitous contextual services on mobile end-user devices. However, security issues are not discussed in this paper.

In our approach, a ubiquitous contextual service is designed as a set of assemblies (or compositions) of distributed and interconnected software components. We distinguish two kinds of assemblies: fixed and mobile ones. On one hand, a fixed assembly represents the permanent part of a service: it is deployed on fixed nodes of the ubiquitous environment at the starting time of the service and is present as long as the service is available to end-users. On the other hand, each mobile assembly represents the part of a service dedicated to one end-user: it is automatically deployed on end-user demand and can be automatically destroyed when the user leaves the contextual wireless coverage zone (e.g. out of a railway station). Our components provide interconnection ports in order to be assembled together. They could be heterogeneous in

terms of hardware and software requirements.

At runtime, our infrastructure allows mobile end-users to dynamically discover the mobile assemblies of ubiquitous contextual services according to the end-users' physical location and also hardware/software device capabilities. For this purpose, we propose a multicast-based discovery protocol that reduces power consumption and network traffic and a negotiation protocol to present to end-users only the mobile assemblies adapted to their device capabilities. Then, our infrastructure allows end-users to automatically deploy the mobile assemblies of ubiquitous contextual services on their own devices. We have designed this infrastructure as a set of software components distributed over fixed and mobile computers.

As proof of concept, we have implemented our model and infrastructure on top of the OMG CORBA Component Model (CCM) [3] and the OpenCCM platform, a Java-based open source CCM implementation [4]. To illustrate our model and infrastructure, we present the design, implementation, discovering, deployment, and execution of a simple train service scenario.

The reminder of this paper is organized as follows. Section 2 presents the train service scenario and discusses the notion of ubiquitous contextual services and the challenges that have to be addressed. Section 3 details our distributed component-based model for designing ubiquitous contextual services. Its principles are illustrated on the design of the train service. In Section 4, we present our component-based software infrastructure dedicated to ubiquitous contextual services. We especially focus on the problems of dynamic discovery and automatic deployment of services. Section 5 details the implementation of our infrastructure and some experimentation using the CCM and OpenCCM, and outlines current limitations. Section 6 presents some related work. Finally, Section 7 concludes this paper and gives some future works.

## 2. Context

This section presents a scenario of ubiquitous contextual service, i.e. the train service. According to this example, a definition of ubiquitous contextual services is given regarding to the user's terminal mobility, and the related challenges are outlined.

### 2.1. An example of contextual service

As an illustration, let us take the following example. A commuter with a wireless PDA is arriving into a railway station. When he/she is inside of the hall, his/her PDA automatically knows where his/her owner is and launches services to obtain some information about the departure platform, train schedule, etc. This service is composed of two parts: the fixed and mobile parts. The fixed part is composed of the back office railway operator information system and several railway station components. A mobile part is dedicated to each user, i.e. run on the user's device, and could be realized by several user interfaces: with a GUI or with a text to speech interface.

### 2.2. A definition of contextual service

Ubiquitous contextual services have a particular behavior. They are available into a well known place (e.g. a railway station). When users are outside of this place (Fig. 1a), the services are not presented and can not be used. Next, when users enter into the place, the infrastructure must show them the services which can be used on their terminals (Fig. 1b). When users choose a particular service, the mobile part must be deployed and instantiated on their terminals, so that it can be used (Fig. 1c). Finally, when users move outside of the network coverage zone, the services can not be used. According to the service, the infrastructure can save the code of the mobile part for another use later, the users can continue to use it in an offline mode, or the mobile part is totally removed from the terminal (Fig. 1d).

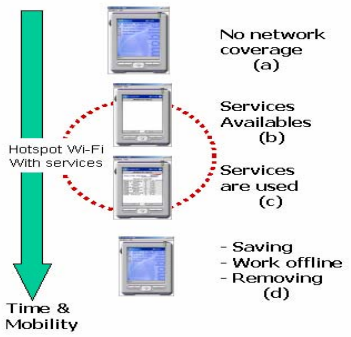


**Figure 1. The life cycle of ubiquitous contextual services.**

### 2.3. The challenges

To realize this train service scenario, we must take into account several issues related to ubiquitous contextual applications:

– **Heterogeneity of devices -** End-users can access services from various devices like a laptop, a PDA or a smartphone. These devices run different operating systems (Windows, Linux, WinCE, PalmOS, Symbian, etc.) and provide various hardware and software capabilities. Concerning the wireless link, several technologies are also available as Wi-Fi (IEEE 802.11a/b/g) or Bluetooth for

short/medium range network, GPRS or UMTS for national coverage.

- **Low resources -** Power consumption is a major problem with mobile devices. Moreover, PDA and smartphones have less memory and smaller processor than laptops.
- **Distributed applications -** The service to get information about the departure platform is distributed on several computers. The graphical user interface is running on the PDA and the process to extract data from the railway operator information system is running on fixed computers. Communication between fixed part and mobile part is done through the wireless network, and can be achieved using middleware like CORBA, Web Services, asynchronous messaging, etc.
- **Service discovery -** The next problem is how a user knows that some services are available in a particular location. Moreover, code of available services to deploy will be different if the terminal is a black and white PDA or a color laptop for instance. To make this choice, the system needs some information about the user's terminal. According to this information, it can present a subset of services which can potentially run on the user's terminal.
- **Service deployment -** Once the user knows that an interesting service is available, then the infrastructure must deploy the code on the user's device. Deployment consists of fetching software components from a repository, downloading them, instantiating, configuring, and interconnecting components. In our context, services must be incrementally deployed: components running on fixed computers are deployed in a preliminary step, and then components running on user's terminal are deployed on user's demand. On-demand deployment avoids us to preinstall on user's devices all the mobile parts of each service and to waste the limited device memory resources.

# 3. Model for ubiquitous contextual services

This section presents our distributed software component-based model for designing ubiquitous contextual services.

## 3.1. Distributed software component model

In order to address the challenges discussed in Section 2.3, we have chosen to design ubiquitous contextual services according to a distributed component-based approach. More precisely, a service is composed of a set of assemblies (or compositions) of distributed and interconnected software components. As ubiquitous contextual services are deployed and run on a distributed system composed of several fixed and mobile computers, we distinguish two kinds of assemblies: fixed and mobile assemblies. On one hand, a fixed assembly represents the permanent part of a service: it is deployed on fixed nodes of the ubiquitous environment at the starting time of the service and is present as long as the service is available to end-users. On the other hand, each mobile assembly represents the part of a service dedicated to one end-user: it is automatically deployed on end-user demand and can be automatically destroyed when the user leaves the contextual wireless coverage zone (e.g. out of a railway station). Each component has a set of configurable attributes and interaction ports that capture features provided by a component or required by other components [5]. We reuse the four kinds of ports defined by the CORBA Component Model: *facet*, *receptacle*, *event sink*, and *event source*. A facet and a receptacle expound an interface, *i.e.* a set of methods, respectively implemented or required by the component. An event sink and an event source symbolize the fact that a component can respectively consume or produce a specific kind of event. The binding between a facet and a receptacle, and between an event sink and an event source allows to realize respectively synchronous and asynchronous interactions between components.

Finally, each component has a base reference interface offering management and control methods, *i.e.* to configure attributes, to connect facets to receptacles and event sinks to event sources. This management interface is mainly used during deployment and reconfiguration of distributed software component applications.

## 3.2. The train service component-based design

The different concepts discussed previously are illustrated, on the train service example, in Figure 2. The "Train GUI" component is deployed on the user's device and interconnected with the "Railway Station" component to get the trains' schedule. It has to be differently implemented according to the type of terminal (a PDA with a small screen, a laptop or a smartphone). So, each component is composed of several implementation codes and of a description of context dependencies. The latter clearly captures the adequate running context in terms of hardware and software resources required to correctly deploy each implementation.

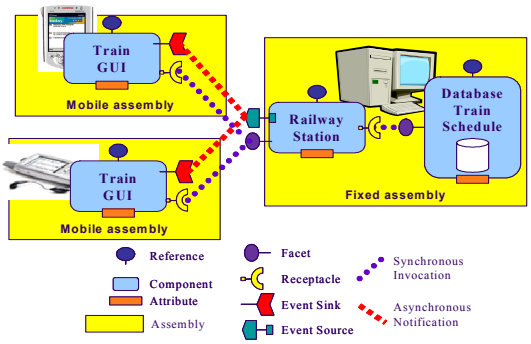


**Figure 2. The train service as a set of assemblies of software components.**

## 4. Our infrastructure for ubiquitous computing

This section describes our software infrastructure dedicated to ubiquitous computing. To design this infrastructure, we followed the distributed component-based approach exposed on Section 3.1.

### 4.1. The dynamic discovery of services

The discovery of services is one of the key points of ubiquitous contextual computing. The challenge to address is how a user entering a specific network coverage zone is informed about existence of available services. To solve this, we firstly introduce a component named *Service Registry* (SR), hosted in a fixed computer of the ubiquitous environment (*e.g.* a server on the railway station). The SR component is in charge of the following main points: 1) Manage the list of available services: the SR component offers a way for service provider administrators to register, update and remove services from the list, 2) Transmit its unique identifier: every SR has a unique ID allowing the differentiation of multiple service providers. Indeed, ubiquitous environments can co-exist from the user point of view in some particular cases. Actually, this happens in our sample scenario if the railway station is near a museum offering other ubiquitous contextual services as information on current exhibitions for instance, 3) Compute the appropriate mobile assembly and/or component implementations of services according to the user's device hardware and software capabilities, 4) Transmit the list of registered services: the list sent to a user should reference only available services that are adapted to its device.

Secondly, we propose to introduce a component named *Service Activator* (SA) hosted on mobile devices. The SA component interacts with the SR one as follows: 1) Receive SRs IDs: from them, the SA component will be able to access the list of available services registered in corresponding SRs, 2) Transmit the device characteristics: this information is sent by the SA component to the SR that has been selected by the user, 3) Receive and display the list of available contextual services that match to the device capabilities.

Thanks to this component-based architecture, the challenge of contextual services discovery can be reformulated as follows: how a SR is discovered by the SA components? Different scenarios at this stage exist, depending on which component – the SA or the SR - is initiating the service discovery. Using a protocol based on multicast, the first case, *i.e.* the services search is initiated by user, implies the user's device being in a "transmit" mode (TX), which means start a periodically sending of search requests. The second case, *i.e.* the user's device is in a "receive" mode (RX), implies that the SR is in charge of the periodical sending of requests through the network. This case is more interesting if we take into account several aspects, and particularly energy aspect, as showed in some IEEE 802.11 wireless cards specifications summarized in Table 1. Devices' wireless card power consumption is reduced by an average of 30 percents in the RX mode. We propose then to implement this mode in our software infrastructure. What's more, having one emitter (the service provider server) and several receivers (the users' devices) that are waiting for requests generates less network traffic than the opposite. This has to be pointed out in the context of ubiquitous applications as the number of devices involved can be potentially important. At last, the RX mode allows a transparent discovery of the SR from users' point of view and updates are automatically and immediately notified to devices.

**Table 1. Power consumption of some IEEE 802.11x wireless cards**

| | TX | RX | RX/TX |
|---|---|---|---|
| Compaq WL110 [6] | 280 mA | 180 mA | 65 % |
| Buffalo AirStation G54 [7] | 550 mA | 350 mA | 64 % |
| Dlink DWL-AG660 [8] | 500 mA | 379 mA | 75 % |

Once the SR is discovered, the SA is able to send a request to get the list of contextual services that are adapted to the device. To realize that, we define a "negotiation" protocol between the SA (*i.e.* the user's device) and SR (*i.e.* the service provider server) components, as illustrated in Figure 3. After the SR discovery (1), hardware and software characteristics of device are sent by the SA to the SR (2). This one computes then appropriate assemblies according to this information.

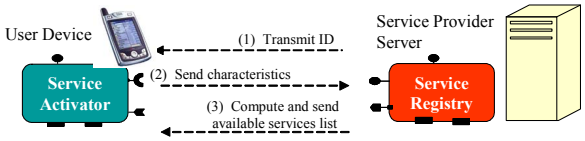


**Figure 3. The service discovery and the negotiation protocol.**

Indeed, many component implementation binaries of the mobile part of a service are possible. For instance, many implementations of the GUI for the train service are available: GUI for white and black or color screens, text to speech interfaces, etc. Moreover, different component assemblies can be proposed, not just in terms of component implementation choices but also in terms of application architectures. To illustrate this point, we can imagine that the user is a *non-French* speaker arriving in a French railway station, so that, if exists, a *Translator* component realizing the *French-to-user's language* translation has to be expressed in the alternate configuration. Finally, the SR sends the list of contextual services that are adapted to the user's terminal (3).

### 4.2. Automatic deployment on-demand

Once services are discovered and one of them selected by user, the infrastructure must deploy the mobile assembly of the service on the user's device. In the context of component-based applications, deployment consists of several steps: downloading component binaries from a (remote) repository to devices, instantiation of these components (*e.g.* the GUI showing the railway information) on the user's device, configuration of their business properties (e.g. the GUI title), and interconnection with the components that belong to the fixed part of the service (*e.g.* the component that processing data to send and the railway station information system). For this purpose, we introduce in our architecture a third component called *Service Deployer* (SD). As for the SA component, it must be previously deployed and instantiated on users' devices. Figure 4 completes the architecture of our software infrastructure for ubiquitous computing and details steps from the service demand by user (4) to deployment of the mobile part of the service (5, 6a and 6b).

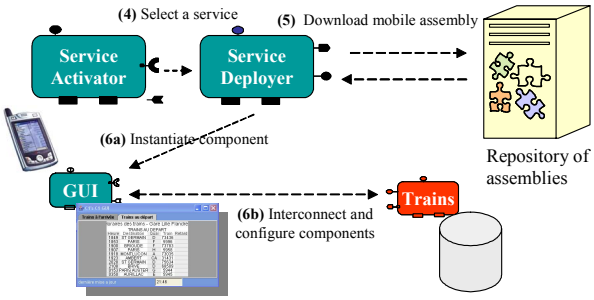


**Figure 4. On-demand service deployment.**

The components of the service that are running on fixed computers have been deployed and instantiated in a preliminary step according to the same deployment process. Assemblies downloaded on devices only contain mobile components which must be instantiated on user's terminal and, if necessary, interconnected to the previously instantiated steady components. The repository of assemblies mentioned in Figure 4 can be located on the service provider server or on a remote HTTP or FTP server (or anything else). Once all these deployment steps achieved, end-user can use features provided by the contextual service, that is, in our case, accessing the trains' schedule.

Nevertheless, the role of the SD component is not limited to the on-demand deployment process. In fact, this component is in charge of the entire life cycle of the service. In the context of ubiquitous computing, users arrive and go outside of ubiquitous environments. In support of this view, just let us imagine the following scenario. The user has taken a train which has started. When user will come in the railway station again, as he/she remembers that the train service exists, he/she would like to use it immediately. That means not having to select one of the displayed SR, next waiting for the device characteristics transmission, choosing the appropriate service and so on. Thus, the idea is to keep in cache mobile assemblies of services user is likely to reuse. The SD component is in charge of this aspect. The management of service versions is done thanks a unique identifier attached to each mobile service assembly.

## 5. Implementation

We have implemented our infrastructure with the *OMG CORBA Component Model* (CCM) and on top of our OpenCCM platform. Firstly, this section briefly describes the motivations for these two implementation choices. Next, we detail the CORBA components for both our infrastructure and the train service. Finally, current limitations of our implementation are outlined.

### 5.1. The OMG CORBA Component Model

We have chosen the *OMG CORBA Component Model* because it is the only vendor neutral open standard for *Distributed Component Computing* supporting various heterogeneous programming languages, operating systems, networks and CORBA products seamlessly [3]. This model is appropriate to create ubiquitous, distributed, server-side scalable, component-based, language-neutral, transactional, multi-user and secure applications. The CCM defines a framework to support the whole life cycle of software development process: design, implementation, packaging, assembling, deployment, and execution. CORBA components provide the required concepts (reference interface, interaction ports and business attributes) to implement our component-based model for ubiquitous contextual services. At runtime, components are hosted by containers which transparently manage the system aspects like component life cycle, networked communication, transaction, security, and persistence. Finally, the CCM defines a XML-based packaging and assembling facility coupled to an automatic distributed deployment process.

We have chosen to use the OpenCCM platform, our open source Java-based CCM implementation [4], because it implements all the CCM features required by our infrastructure for ubiquitous contextual services, especially packaging, assembling, and automatic deployment facilities. Moreover, using OpenCCM allows us to experiment our infrastructure in truly heterogeneous environments as OpenCCM supports various operating systems like Linux, Solaris, Windows NT/2000/XP, Windows CE and Linux Familiar for PDA, and most of available Java-based CORBA products, *i.e.* JacORB, OpenORB, ORBacus or Borland Enterprise Server.

### 5.2. Software infrastructure implementation

First of all, we defined two CORBA components to respectively model the SR and the SA. However, we added a third CORBA component named *Multicast* to implement the interactions between the SA and the SR during the service discovery phase (Figure 5).

This generic *Multicast* component implements a "send/receive" multicast request function and is instantiated on each node (*i.e.* on all devices and servers). SA and SR components are both bound to it (but not the same instance) as follows. On one side, the *Multicast* component is bound to the SR's facet that provides the list of available ubiquitous services and which IOR (and the associated SR ID) is sent by multicast. On the other side, the SA component is bound, thanks to a receptacle, to the *Multicast*'s facet that provides the list of discovered SRs (*i.e.* the SR IDs received). Using of multicast requests is well suited for the implementation of the discovery protocol in the context of a ubiquitous environment. Indeed, SA and SR components are not directly connected (from a CCM point of view), which was the original goal as the user/device is not assumed to know about its environment before discovering it!

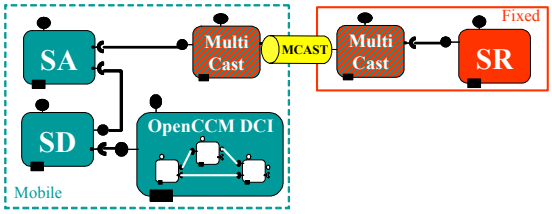


**Figure 5. The CORBA components assembly for service discovery and deployment.**

The code below (written in the OMG CCM *Interface Description Language*) defines the main interfaces and operations provided by our infrastructure components, and interaction ports between these components.

```
// Interfaces provided or required by components.
interface Receive {
                        IORList get_list_IOR(…);
                        void stop(); };
interface Services {
                        void add(…);
                        void update(…);
                        void remove(…);
                        IOR get_IOR(); };
interface Deployer  {
                        void deploy(…);
                        void undeploy(…); };

// SA, SR, SD and Multicast components
component ServiceActivator {
                        uses Receive receive;
                        uses Deployer deployer; };
component ServiceRegistry {
                        provides Services services;
};
component ServiceDeployer {
       provides Deployer deployer;
       uses ::DCI::DCIDeployment dci;
};
component Multicast {
                        uses Services services;
                        provides Receive receive;};
```

As far as the deployment is concerned, a CORBA component implements the denoted SD. It provides a facet, used by the SA component's receptacle, which offers operations for deploying the services. More precisely, the SD component is connected to our OpenCCM *Distributed Component Infrastructure* (DCI). The DCI infrastructure is also designed as a set of CORBA components to manage the deployment phases of CCM applications [9]. DCI components

provide facets offering deployment operations as install an assembly by downloading it from a repository to the device, instantiate and tear down it. Also, the SD implements the cache management of assemblies, as explained in section 4.2.

To experiment our infrastructure, we have also implemented the train service using two CORBA components. Figure 6 illustrates on a PDA this service at work, *i.e.* after the dynamic service discovery and its deployment. The SA component (1) is represented by a GUI displaying, firstly the list of SRs that have been "detected" (left column of the SA GUI), secondly the list of available services registered on each SR selected by user (right column). The client part of the service is represented by a GUI showing the trains' schedule (2).

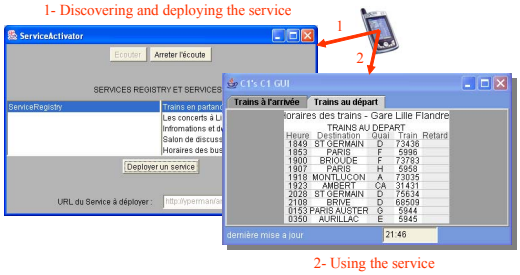


**Figure 6. An example of what the user is finally seeing on his/her device.**

Both infrastructure and service have been tested using a *Dell Lattitude* laptop with Windows XP and the JRE 1.3.1 (as the fixed server), a set of *Compaq iPAQ* PDA with Windows CE and the IBM J9 JVM (as mobile devices), and a 802.11 wireless network. The ORB used is the free JacORB product [10]. The fact that our experiences are based on the OpenCCM platform (which is written in Java) implies component implementations to be written in Java. Nevertheless, as CCM supports platform heterogeneity, it is obviously possible to use component binaries written in other languages (e.g. C++) for both infrastructure components and service components. These binaries must be deployed on appropriate containers (e.g. a C++ container provided by the CIAO [11], MicoCCM [12], or Qedo [13] platforms for instance) instead of the OpenCCM one.

### 5.3. Limitations

Some limitations of the CCM implementation of our infrastructure should be emphasized:

**Deployment -** The big size of assembly archives may cause some download time problems during the installation step of the deployment. This is due to the limited throughput of current wireless networks, and the way containers are generated in OpenCCM. Dynamic generation of container classes at execution time could address this problem by reducing size (then downloading time) of assembly archives.

**Multicast -** The multicast protocol had to be implemented using a specific hand-written component (*Multicast*). This component would be useless if multicast communications between CORBA components were directly supported by the CCM.

**Thread -** Our infrastructure components (but also many ubiquitous services) use threads to deal with parallel activities. Management of these threads is done explicitly in the code of component business implementations. This point can be avoided if component containers are able to manage this system aspect transparently.

## 6. Related works

A key element in ubiquitous and contextual services is the **service discovery function**. The OMG/ISO trader [14] and the RM-ODP [15] trader mediate advertisement and discovery of services. The implementation of our service registry could use such technologies. JINI [16] from Sun Microsystems offers several ways to discover services. The Multicast Request Protocol permits to user's terminal to send a multicast request to a lookup service. The response is sent via a TCP unicast message. The opposite solution, named Multicast Announcement Protocol, is used by lookup services to announce their presence to any interested parties that may be listening and "in range" of the multicast scope of the lookup service. Some other protocols like UpnP, Salutation, SLP, etc., exist and globally offer the same functionalities. A comparison of these protocols can be found in [17].

Concerning the **global infrastructure** to design ubiquitous services, the Appear Provisioning Server [18] from AppearNetworks is an infrastructure dedicated to service discovery and deployment. This allows deployment of wireless infrastructure in which applications and documents are available on users' terminal on dedicated places. Applications are developed independently of the infrastructure.

Concerning **CORBA based** related work, the Smart Deployment Infrastructure (SDI) [19] is based on an OMG trader service to discover available services. However, they don't explain how a terminal discovers the trader service in a particular location. The work in [20] is based on the *Satin* model. They developed component assemblies with safety properties. This work is complementary to our work.

## 7. Conclusion

Ubiquitous and context aware applications are one of the new challenges in distributed computing area. Until now, there is no well established approach from the design step to the execution step to build such applications. To address this challenge, we have presented in this paper a distributed component-based software approach to build both ubiquitous context aware applications and the dedicated software infrastructure that addresses service discovery and automatic deployment on demand on mobile devices. We have implemented and tested this infrastructure on top of the OMG's CORBA Component Model and the OpenCCM platform. We have also designed, prototyped and tested the train service scenario on a heterogeneous infrastructure composed of a laptop, a set of PDA and a wireless network.

Our future works will go in two directions. Firstly, according to the current limitations listed in Section 5.3, we will enhance the OpenCCM platform in several directions: dynamic container generation to reduce the size of assembly archives and improve downloading time on slow wireless networks, integration of multicast communications in the CCM model, and adding a threading service inside CCM containers. Secondly, we will experiment more examples of ubiquitous applications in order to propose a model-driven approach for building both services and infrastructure for this domain, independently of underlying technological platforms.

## 8. References


[1] M. Weiser, "The computer for the 21st century", *Scientific American*, 3(265):94-014, 1991.

[2] C. Endres, A. Butz, "A Survey of Software Infrastructures and Frameworks for Ubiquitous Computing", *Mobile Information Systems Journal*, 1(1), Jan-Mar 2005.

[3] Object Management Group, *CORBA Components Specification*, version 3.0. OMG TC Document formal/2002-06-65. OMG, Boston, 2002.

[4] The OpenCCM Team, *The OpenCCM Project*, ObjectWeb Consortium, 2002. http://openccm.objectweb.org

[5] C. Szyperski, *Component Software: Beyond Object-Oriented Programming* (Second Edition). Component Software Series, Addison-Wesley and ACM Press, 2002.

[6] Compaq, *WL 110 Wireless Card User Manual*, 2001.

[7] Buffalo Tech, *WLI-CB-G54A User Manual*, 2002.

[8] D-link; *DWL-AG660 User Manual*, 2004.

[9] A. Hoffmann, T. Ritter, J. Reznik, M. Born, B. Neubauer, F. Stoinski, H. Boehme, B. Folliot, M. Vadet, U. Lang, P. Merle, C. Contreras, *Specification of the Deployment and Configuration*, IST COACH Document Deliverable #2.4, 2003. http://www.ist-coach.org

[10] The JacORB Team, *JacORB 2.1 Programming Guide*. May 2004. http://www.jacorb.org

[11] N. Wang, D. Schmidt, and D. Levine, "Optimizing the CORBA Component Model for High-Performance and Real-Time Applications". In Work-in-Progress session at the IFIP/ACM Middleware 2000 Conference, Pallisades, New York, April 2000. http://www.cs.wustl.edu/~schmidt/CIAO.html

[12] Frank Pilhofer, *The MICO CORBA Component Project*. 2000. http://www.fpx.de/MicoCCM/

[13] T. Ritter, M. Born, T. Unterschütz, T. Weis, "A QoS Metamodel and its Realization in a CORBA Component Infrastructure", In Proceedings of the 36th Hawaii International Conference on System Sciences, Software Technology Track, Distributed Object and Component-based Software Systems Minitrack, HICSS, Honolulu, Jan. 2003. http://qedo.berlios.de/

[14] Object Management Group, *Trading Object Service Specification, version 1.0*. OMG TC Document formal/2000-06-27. OMG, Boston, 2000.

[15] P. Leydekkers, *Multimedia Services in Open Distributed Telecommunications Environments*. PhD Thesis CTIT, 1997.

[16] Edwards, W. K. *Core Jini Introduction*. Prentice Hall PTR, 1999.

[17] R. E. McGrath, M. D. Mickunas, R. H. Campbell, *Semantic Discovery for Ubiquitous Computing*, http://citeseer.ist.psu.edu/485904.html

[18] AppearNetworks, *The Appear Provisioning Server*. http://www.appearnetworks.com.

[19] C. Taconet, E. Putrycz, G. Bernard, "Context Aware Deployment for Mobile Users", In Proceedings of the Computer Software and Applications Conference (COMPSAC 2003), pages 74-81, 2003.

[20] A. Occello, A.M. Dery-Pinna, "Sûreté de fonctionnement d'applications nomades construites par assemblage de composants", In Proceedings of UbiMob'05, pages 73-80, Grenoble, France, 2005.