Smart Cards: A System Support for Service Accessibility from Heterogeneous Devices[1]

Raphaël Marvie*, Marie-Claude Pellegrini, Olivier Potonniée****

* LIFL, UPRESA 8022 CNRS, Bât M3, UFR d'IEEA, 59655 Villeneuve d'Ascq, France
raphael.marvie@lifl.fr

** Research Lab, Gemplus Labs, BP 100, 13881 Gémenos Cedex, France
{marie-claude.pellegrini,olivier.potonniee}@research.gemplus.com

Abstract

With the proliferation of computers that don't look like computers, such as mobile phones, TV's, PDA's, end-users wish to take advantage of them to access their applications and information, wherever they are and whatever the device they use.
This article discusses a common mean to access a same service from various kinds of devices, and presents how the smart card, a medium compatible with most terminal kinds, can be used as application bootstrap.

1. Introduction

Nowadays, people are more and more nomadic. The scope of their information system is not reduced to a single PC anymore. Most of today's applications are designed in the context of local area networks. However, LANs do not ideally address the users' needs because they reduce the application accessibility to the scope of one activity (work, home, *etc.*). Thus, today's applications suffer from exiguous scope and do not benefit totally from the potential of large-scale networks.

The number of different kinds of devices used as access points to information systems increases from day to day. Beyond the PC, the emergence of mobile devices such as PDA's or mobile phones yields to a new point of view about how services have to be designed in order to be uniformly accessed. The issue related to the deployment of value-added services to end-users from heterogeneous devices is not trivial. Today, this issue is generally addressed by offering different service incarnations for every kind of terminal. For example, web-dedicated services are available on PCs and mobile phones, but their incarnations are different from one device type to the other. Such a situation yields to market segmentation that makes difficult for one service provider to develop and deploy a same service for different target devices.

In this article, we introduce a mean to support mobility and to hide heterogeneity to the end user. We intend to provide a framework to build applications for large-scale networks in a way that a single application is adapted to the device on which it is used. We then discuss the use of smart cards to address these two points.

Section 2 discusses the new requirements raised by users' mobility over a large range of terminals. Section 3 presents our approach to address the problem and discusses the use of smart card. Section 4 summarizes the article and outlines our ongoing work on the subject.

2. Towards a mean to support mobility and to manage heterogeneity

2.1. Users' mobility

Users being more and more nomadic due to work or leisure, solutions have to be found to provide them their applications and personal information whatever their location. Furthermore, applications content and configuration must be relevant to the user context, such as his location and his access device.

[1] This work is part of the CESURE project—RNRT project number 98—partly granted by the French government (Department of Research, Education, and Technology).

Traditional IT solutions for this problem rely on a central storage of user's information and applications, that is to say the user's environment. Data are downloaded to the users' terminal when he connects himself (identifying and authenticating himself). This technique has some flaws, especially when using large-scale networks. First of all, the availability of users' environment relies on the accessibility to the remote server. Any failure in the communication infrastructure or the server itself prevents the user from accessing his application and personal data.

Furthermore, personal information is sent over the network from the storage server to the user terminal. So confidentiality of users' personal information is not guaranteed. Indeed, even if the data is enciphered before emitted over the network, the user cannot trust the deciphering mechanism of the terminal, especially if the terminal is public.

This solution guarantees neither the availability of the service and users' personal information, nor their confidentiality.

2.2. Heterogeneous terminals

The number of computing device kinds increases from day to day: Set Top Box, mobile phone, PC, PDA, etc. All of these terminals can be used like computers without looking like them. They are sort of tomorrow's PCs. Nowadays, most of these terminals could be connected to a network. Connected terminals might be private, owned and used by a specific person, or public, owned by an organization. According to the types of terminals, needs are different in terms of security.

The heterogeneity of terminals implies multiple implementations of a same application. Each implementation has to be available to any potential users. Three solutions could be considered to store and provide these implementations for each kind of terminals. First, centralized servers could be used, second a medium containing all the implementations could be provided to each user, and third each application could be installed on every terminal.

Using a remote server implies for users to connect themselves to it. Thus, the information location has to be known in order to choose and download the right implementation. Using this solution, access to the application relies again on the accessibility and on the availability of the remote server.

Using a medium to carry implementations of the application for all kinds of terminals is an other way to bypass this heterogeneity. Different mediums are nowadays available like CD-ROM, floppy disk, or memory stick [4]. But this choice is unrealistic for multiple reasons. First of all, these mediums are not compatible with all terminal types: Set Top Box, mobile phone, and PDA have no drivers for such mediums. Then, the medium content would have to be changed each time a new connection point kind is made available. Finally, it implies the user to install himself the right application implementation according to the connection point, each time he wants to access his service.

The third solution is to install on each connection host all the applications that can be required and used by any potential user. This solution is unfeasible. First, it requires storing all applications on each host, thus implying a huge amount of information to store. Then, it implies for each new application to be installed everywhere, which is also unrealistic.

A solution is required to offer users applications whatever the user connection point. Indeed, it is necessary to offer a simple solution for the user to abstract himself of the heterogeneity of terminal. The fact that numerous applications are used to provide a same service according to the terminal has to be transparent for the end user.

2.3. Towards a unified application bootstrap

In order to address mobility and to hide heterogeneity, a solution is required to allow application access and their execution from most kinds of devices. This solution has to be simple for users and to guarantee the availability, the confidentiality, and the reliability of information.

The smart card constitutes an easy and secure means for users to bring their application-related information to many different terminal kinds, and for the application provider to protect the access to his services. It is also an execution support for applications requiring a secure execution environment [7].

To be able to match heterogeneous devices, we define *adaptive applications* using a component-based approach [5]. In this scheme, an application is a set of interconnected components, distributed over a network [3]. Depending on device capacities, component implementations and distribution may vary.

Application descriptions and user information are contained in the smart card. An application description defines the types of components to be used and their interconnection in order to offer the service content. This *abstract description* only defines the architecture of the application.

In order to perform the deployment of such applications [1], an application bootstrap is necessary. This bootstrap uses the application description to provide a ready-to-use instance of the application. In our proposition, this bootstrap is a deployment pilot stored and executed in the smart card. The use of smart cards as an external system support allows users to carry their applications during journeys and use them from a variety of devices.

This abstract description has to be enriched to be adapted to the context of the device on which the application is deployed. The result is a *concrete description*, which is context (location and device) dependent.

Using the concrete description, the bootstrap searches component implementations using properties such as component type or system constraints, and instantiates them when and where needed. Each element of the concrete description is thus mapped to a real software entity, leading to an *application instance*.

This approach allows adaptation of a same application description, contained in a smart card, to numerous device kinds. In fact, the same service content is provided but not in the same manner. It is impossible and useless to provide the same GUI on a mobile phone and on a PC. The important aspect is that the same information is provided on both devices. The means to provide information is secondary compared to information themselves. So, in our approach, the construction of an adaptive application is based on descriptors, progressively refined during the bootstrap process performed on smart card system.

3. Launching adaptive applications

The infrastructure to which applications are targeted is composed of a set of execution environments. Two kinds of such environments exist in the system:

- Terminals able to communicate with smart cards and to host some application component instances, and
- A network of computers which stores component implementations and permits applications and instances to execute themselves.

Cooperations between all these elements are done through the network, about which no hypothesis of reliability, security, nor quality of service are expressed. These elements, described in next sections, are shared between applications.

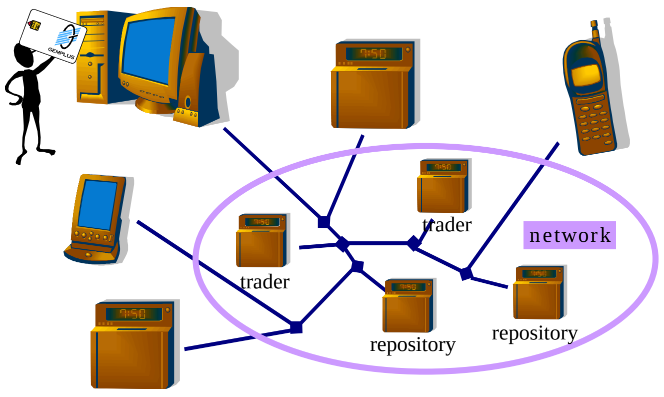

Figure 1: Deployment infrastructure

In order to launch an adaptive application in such an environment, the system needs a set of elements: The application description, the context knowledge, and the distributed infrastructure to locate and create instances of components implementations. We propose in this section a complete and consistent framework to achieve this goal.

3.1. Providing the abstract description

In our approach, the smart card is the pilot of application deployment. It does not only contain the abstract description but also the deployment logic, which makes it a portable application bootstrap. The idea is to externalize the evaluation of the bootstrap from the terminal in order to hide the heterogeneity of this one.

The smart card never provides the full application description to the terminal, it only sends elementary deployment commands which are built from the abstract description [2]. In that, the smart card keeps the deployment process under control, and may choose not to send some of the instructions because, for example, the connection point seems not enough trustable. Smart cards contain the deployment process that reacts when a deployment error occurs. This algorithm is able to analyze the potential errors and allow different politics of reaction, such as degrading the service, or suspending its execution.

3.2. Constructing the concrete description

In order for the smart card to be useful, terminals in which it is inserted must be ready to receive deployment commands. Thus, each terminal, whatever its kind, has to hold a basic and generic software service, that we call *Deployment Portal*, which is in charge of processing the bootstrap commands.

This portal first detects the insertion of a smart card and authenticates the user. Then, it offers access to the user's services according to the information contained in the card. Once the application has been chosen, the portal receives the deployment commands through a standard interface. This interface represents the vision of the device for the smart card. It offers functions to search component implementations, to connect component instances, to configure them and to start the application. A simplified version of this interface is presented in Figure 2.

```
interface DeploymentPortal {
   void retreive ( Component c );
   void establish ( Connection c );
   void configure ( Configuration c );
   void run ( );
  }
```

Figure 2: The deployment portal interface

The Deployment Portal analyses and enriches the commands with context dependent properties. It translates abstract descriptions into concrete ones, based on the knowledge it has of its location and execution capacities (underlying operating system, dialog interfaces, etc.). Figure 3 illustrates how an abstract description is enriched. However, the Deployment Portal does not perform the deployment itself, as it may be a too small device to do so. It forwards the enriched commands to a *Deployment Engine* that will really create the application instance.

Abstract: "*A user interface connected to a regional weather forecast server*"

Concrete: "*A* ***GSM*** *user interface connected* ***by WAP*** *to a* ***south of France*** *weather forecast server* "

Figure 3: From abstract to concrete description

3.3. Instantiating the concrete description

The application instance is created by a distributed framework, based on the deployment requests emitted by the Deployment Portal. Its role is to create an application instance in the system from the concrete description. This process involves a set of elements from the infrastructure that are presented in this section.

3.3.1 Component implementation repositories

As discuss above, the smart card only contains the description of the application. It does not contain, mainly for storage capacity reasons, the component implementations. This description is not enough to create an application instance, the component implementations are also required at runtime.

Component implementations are supplied by providers through *component implementation repositories*, which are storage servers where components are placed in order to be retrieved later by deployment engines. Component implementations are packaged in a uniform manner throughout all repositories. Thus, deployment engines can homogeneously access and use them. A component package contains the component type, the component implementation, and a descriptor. This descriptor identifies the name, version, static and user configurable properties, interface signature of the component, as well as implementation constraints (OS, JVM, GUI kind...).

3.3.2 Traders

Component implementation repositories are intended to store component implementations. However, it does not define how to organize the implementations nor how to retrieve them. In order to find back implementations, i.e. to search in the various repositories, traders are used [6].

Each time a component implementation is inserted in the system, it is also bound in a trader with its properties (type, functional description...). This permits the implementation to be usable at runtime, otherwise no one would know it is available. Similarly, when component instances are created in order to be shared, like servers, they also have to be bound in a trader in order to be found back later.

For the trader service to be scalable, traders should be federated. Through a federation, the traders associated with component implementation repositories or instances servers become one trader. Thus the system knowledge becomes a whole, and is accessible from any portal of the system. Inserting a component implementation in one repository leads to its availability from any portal using the same trader federation.

3.3.3 Deployment engines

Deployment engines are the heart of the deployment process. They control the installation of component implementations and the creation of instances on the required hosts of the system. Even if deployment engines are driven by deployment portals, they are not necessarily co-located with them. A GSM phone, for example, only contains the deployment portal.

Deployment engines rely on two fundamental blocks. First, as discussed in the previous section, the trader is used to find back the pieces of the application. Second, in order to deploy the application where and when required, the deployment engine uses component containers and their associated instance managers. Containers permit implementation downloading and instance creation.

When searching a component instance using properties, the sequence of answers returned by the trader could be of two kinds. First, implementations that the deployment engine has to download and instantiate. Second, instances that are already running in containers. All these instances, reused or newly created, are then potentially configured and connected together. Connection is performed through the component type capacities.

Then, once the application has been successfully created (component instances exist, are configured, and connected) the deployment engine starts the application execution. When the user do not need it anymore, the deployment engine will remove from the system all the no-more-useful component instances.

4. Conclusion

In this article, we have identified new requirements arising from the evolution of users' behavior, and in particular their increasing mobility. Wherever the user asks for it, and whatever the device he uses, it is necessary to provide him his applications and his personal information, preserving their confidentiality and integrity. Using different device kinds implies the use of numerous implementations of a same service. However, the service content must remain identical, and the implementation differences may only concern part of the application. Our goal is to provide a way to launch these applications while masking the heterogeneity to users.

We thus proposed a solution to launch applications in a homogeneous way, whatever their location and access device. This solution supposes a modular conception of applications based on component technology, allowing to distribute processing, and to interchange implementation pieces according to the execution context. We call this kind of applications *adaptive applications*, as they are able to adapt themselves to their environment at launch time.

We have presented an application *bootstrap* mechanism for adaptive applications, based on a smart card that stores the abstract description of the application and its deployment logic. Smart cards provide a portable and secure storage and execution unit, compatible with most of existing terminal kinds. We outlined the distributed framework necessary to translate the abstract application description to an application instance. This set of tools constitutes a system support to mask heterogeneity of access devices while preserving a same service content.

Early prototypes have validated the main concepts presented in this article. Current work focuses on the construction of a multi-environment framework, and on the development of smart card services to handle applications description and support deployment policies.

5. References

[1] CARZANIGA A. et al., "A Characterization Framework for Software Deployment Technologies", Technical Report CU-CS-857-98, CS Dept. Univ. of Colorado, 1998.

[2] HALL R.S., HEIMBIGNER D., WOLF A.L., "Cooperative Approach to Support Software Deployment Using the Software Dock", Technical Report CU-CS-871-98, CS Dept. Univ of Colorado, 1998.

[3] PELLEGRINI M.-C., RIVEILL M., "Dynamic Architecture Management of Component Based Applications", Proc. Of Parrallel and Distributed Processing Techniques and Applications, PDPTA'99, Las Vegas, Nevada, USA, vol 2., pp. 800-806, juin 1999.

[4] SONY Digital Imaging, "Memory stick", 1999
http://www.sel.sony.com/SEL/consumer/dimaging/browse_the_products/memory_stick/index.html

[5] SZYPERSKI C., "Component Software – Beyond Object-Oriented Programming", Addison-Wesley 1998

[6] TERZIS S., NIXON P., "Component Trading: The basis for a component oriented development framework", of WCOP'99, 1999.

[7] VANDEWALLE J.J., "Construction of Distributed Applications based on Smart Cards", Course material for the Third ERSADS European Research Seminar on Advances in Distributed Systems, Spring School and Workshop, Madeira Island, Portugal. 23-28 April 1999.