The Software Station
A System for Version Controlled Development and Web Based Deployment of Software for a Mobile Environment

Lei Liu Philipp Obreiter

lei.liu@web.de, obreiter@ipd.uni-karlsruhe.de

August 5, 2003

Technical Report Nr. 2003-16

University of Karlsruhe
Faculty of Informatics
Institute for Program Structures and Data Organization
D-76128 Karlsruhe, Germany

Abstract

As mobile devices become more wide spread, making available software for such devices is crucial for making use of their capabilities. However, due to the inherent constraints of mobile devices, it is highly non trivial to do so. Especially for larger software projects, additional support for the development and deployment phase is required. Therefore, in this paper, we propose a system that provides such support, i.e., the Software Station. The system's requirements result from the analysis of our project environment. On the one hand, the development of software components is based upon a version control system. On the other hand, a web based interface facilitates the automated yet flexible deployment of software components to the respective mobile devices. We highlight design and implementation issues of the Software Station. In this context, our experiences with the applied technologies and tools are shared.

1. Introduction

Over the past ten years, there was a trend towards smaller and higher processing power of mobile devices on the market. Advanced technologies have made the mobile devices smaller while increasing their processing power and memory capacity. The technology advances enable also mobile devices to run applications which existed only on a desktop computer before. Another trend is that mobile devices go wireless. Many devices are equipped with a WLAN card to enable the access to resources in the internet or on a central data repository at any time, from anywhere and using any system.

According to the equipments, mobile devices can be divided into two main categories: high end, such as Notebooks and Tablet PC and low end, such as PDA and other appliances. All of these devices share the following characteristics up to a certain degree:

- Restricted screen size and, thus, restricted screen resolution makes it difficult to design and develop a complex GUI for mobile applications. The information which can be displayed on the screen at the same time is limited.
- Because of their diversity, mobile devices usually require complex settings for the operating system.
- The communication modules of these devices (such as the WLAN card, Bluetooth, IrDA) normally consume much more energy than the processor itself. In addition, the battery power implies inherent limitations of the device's usage patterns. Consequently, the mobile devices are prevented from being connected to the internet all the time. Furthermore, due to the nature of the wireless communication, it might make it difficult to connect to the internet or to some other devices.
- Most devices are difficult to handle with.
- Users of these devices are normally not professional users. They are interested in accessing distributed information in the internet but not in the related technical know-how.

These characteristics imposes severe burdens for the management of mobile applications.

In this paper, we will discuss the problem and a possible solution in context of the DIANE project [1] [4], in fact the project where this paper is from. In schools and universities, mobile devices like laptops, PDAs, and in particular cell phones become more and more popular. This fact has driven us to find out a way to support the students to use these devices for their studies. The basic idea of the project is to enable students to form ad hoc networks on demand and have access to the available resources in the network.

The aforementioned problem becomes very critical in DIANE project. In order to form an ad hoc network on demand, new protocols have been conceived in the project [2] [3]. Apart from them, there are a lot of program libraries which act as service providers in the ad hoc network. All of these program libraries should be present on the mobile devices so that it becomes possible to share resources in the network. How will these programs be delivered to the mobile devices? And how about the update of a single program? From the database point of view, we have a central data repository. The problem is how data can be delivered to the client and how a single component or even all components can be updated? Obviously, all of these problems should be resolved under the aforementioned constraints of mobile devices.

The remainder of this paper is structured as follows: In Section 2, we briefly describe some background knowledge about the lifecycle of an application from the software engineering point of view in order to be able to discuss the problems from inside of the mobile application lifecycle. In Section 3 and 4, we introduce our solution to resolve these problems. We share our experiences with the applied technologies in Section 5. The usage of the Software Station is exemplified in Section 6. Finally, we conclude our paper in Section 7.

2. Lifecycle of Mobile Applications

From software engineering we know that the lifecycle of applications can be divided into several phases [11]. There are the *Analysis Phase*, the *Design Phase*, the *Development Phase* and the *Maintenance Phase*. In case of application updates, it is a repetition of the whole lifecycle, maybe without *Analysis Phase*. Figure 1 illustrates the lifecycle.

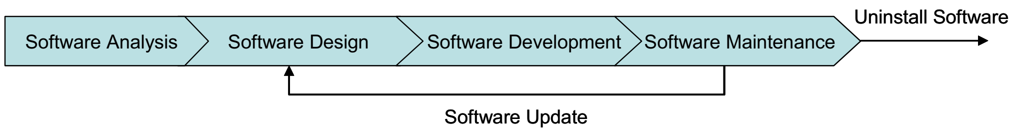


Figure 1: Lifecycle of mobile applications

In the context of this paper, we interested in mobile applications. Hence, among the four phases, only the *Software Development Phase* and the *Software Maintenance Phase* are within our scope. Therefore, we focus on these two phases. For this purpose, we adjust the standard lifecycle by aggregating the *Analysis Phase*, the *Design Phase* and the *Development Phase* to a single *Development Phase*. In the following, the problem is defined from phase to phase in more detail.

In the *Software Development Phase*, after a developer has finished his work and has compiled all the program files into a single software component, the new software component is checked in a version control system. How does the administrator know that there is a new component available and how can he get the new component? If many new components have arrived, how does he know which component should be forwarded to the client?

In the *Software Maintenance Phase*, all the components on the server are available to the client. How does a common user know about the components that are needed for his purpose? How can he determine the relations between the components so that all of the components work properly? If all of the required components are determined, how does he install them under the inherent constraints of his mobile device?

After this analysis, we can get a few design requirements for our solution:

1. In the *Software Development Phase,* there has to be some automating process which recognizes new components and subsequently makes it available in the central data repository.
2. An administrator should be able to organize the software components into several basic software packages and then into several installation profiles. Consequently, the user decides which profile to use instead of choosing among several components.
3. A client application for the automation of the software installation process should also be available to the user. Hence, the user does not have to install every component manually.

In Figure 2, the global architecture of our solution is shown. It matches the requirements that we listed above. The component filter we used in this architecture blocks the software components which should not be published by the software component service at all. And the additional software repository is used to increase the performance for our software component service. It stores all available versions of the components. Since the communication with the version control system is much more expensive than the direct access to the local file system.

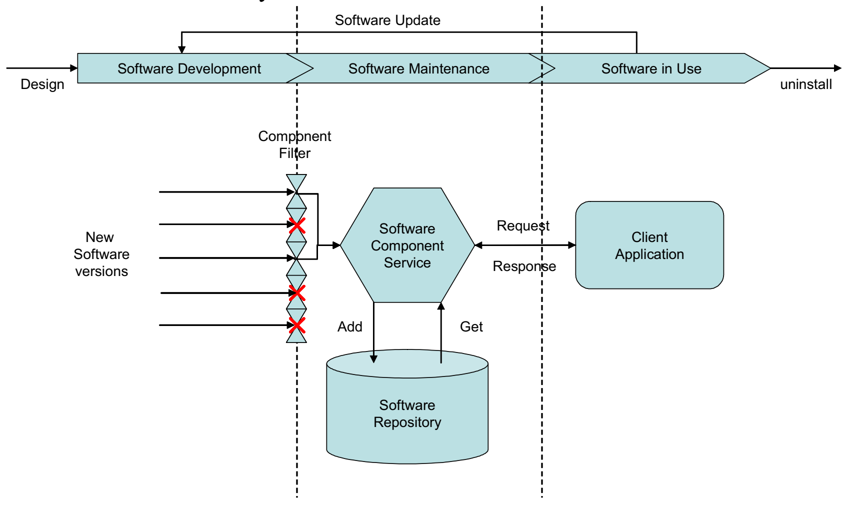


Figure 2: Global Architecture

3. Support for the *Software Development Phase*

In the DIANE project, we use a version control system to maintain the software components and to resolve version conflicts. Consequently, we have access to all of software components on a single central place. On the other hand, the update notification service of the version control system enables us to know about the availability of new updates on the server. Under these preconditions, we are able to design the part of our solution that automates the component update process.

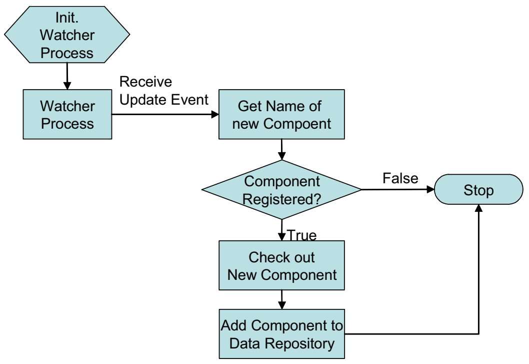


Figure 3: Automating process for the *Software Development Phase*

As shown in Figure 3, a watcher process is initialized and started by the main process. This watcher process checks regularly if an event has been fired by the version control system. If an update event is found, it will start the update process as a sub process. The update process parses the event and retrieves the component name. Using this name, the process makes sure that the component has been registered by the administrator before. In case of a negative result, the process terminates itself; otherwise it checks out the new component version from the version control system and add it directly to the data repository. At the end of the process, a new log entry is created in the log file and the sub process is terminated.

Apparently, the whole update process for new component does not require any intervention by the administrator. He only has to administer the component register file used by the component filter. This is done via the program interface provider of the component service.

4. Support for the *Software Maintenance Phase*

In this section, we take a closer look at how the deployment process is automated for the administrator and the user. The key problem is how the components are organized while considering the dependence between different software components. For this purpose, we have to introduce a new deployment configuration document the structure of which is illustrated in the following Figure 4.

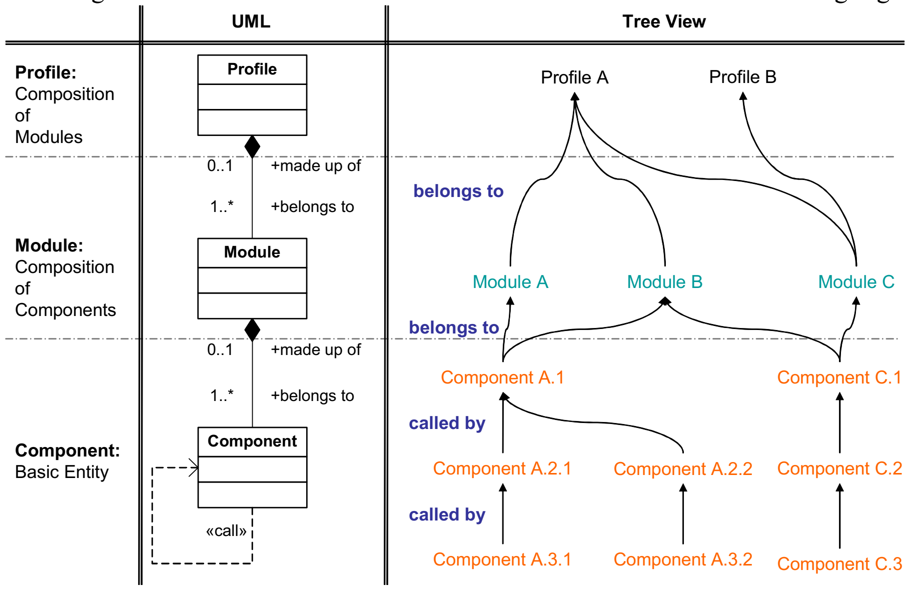


Figure 4: Structure of the deployment configuration document

A deployment configuration document is divided into three sections: the component section, the module section, and the profile section. Furthermore, there are three different data objects available in this document: component, module, and profile. The basic entities are components. A component entry in the configuration document corresponds to a component in the data repository. In this entry, the administrator should specify the name of the component and, if necessary, also the version of the component. Without any version input, the newest available version will always be used. By using the attribute *dependsOn*, the *called-by* relation between two components is explicitly specified. A module object in the module section is composed of component objects. As shown in the UML diagram a module must contain at least one component. It may reference to any component which is defined in the component section so that a module can contain any node or even the leaves of the component tree.

Similarly to modules, a profile in the profile section is a composition of modules. Every profile includes one or more modules which are defined in the module section.

After having explained the structure of the deployment configuration document, we take a look at the process in the *Software Maintenance Phase*. Again we explain the process by the means of a flowchart. It is illustrated in Figure 5.

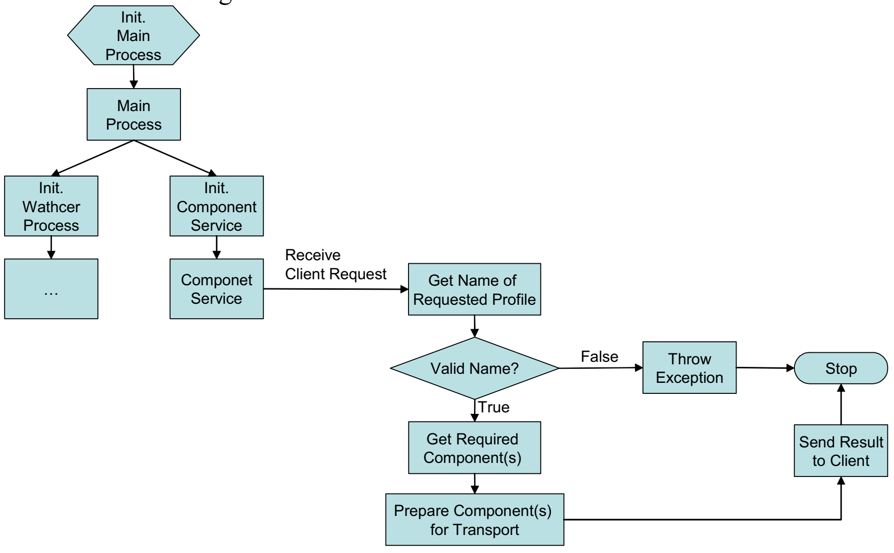
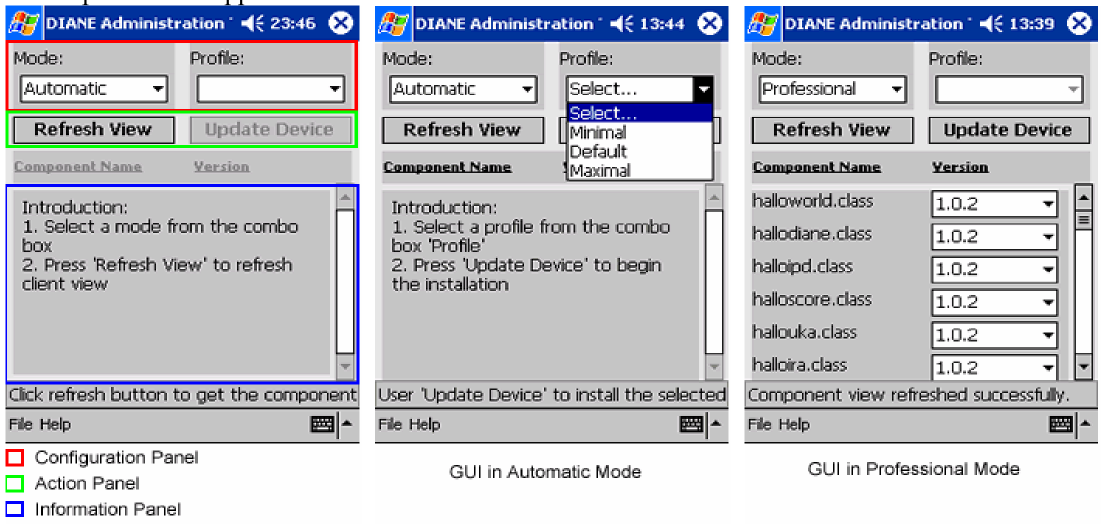


Figure 5: Automating Process for the *Software Maintenance Phase*

After initialization by the main process, the component begins to wait for a request from the client. On receiving a client request, the component service verifies if the given profile name is valid. If it is not, the component service throws an exception and stops itself; otherwise it gets all components of this profile from the central data repository. Before the required data is sent back to client, it will is serialized for the transport over the internet. Afterwards the transport back to the client takes place and the sub process is stopped.



Introduction:
1. Select a mode from the combo box
2. Press 'Refresh View' to refresh client view

Click refresh button to get the component

Introduction:
1. Select a profile from the combo box 'Profile'
2. Press 'Update Device' to begin the installation

User 'Update Device' to install the selected

Component Name	Version
halloworld.class	1.0.2
hallodiane.class	1.0.2
halloipd.class	1.0.2
halloscore.class	1.0.2
hallouka.class	1.0.2
halloira.class	1.0.2

Component view refreshed successfully.

Figure 6: Sample GUIs from the Client Application

After having talked about server processes, we take a look at the client side. On the mobile device there is a client application which supports the user in installing, uninstalling, and managing the software components. Figure 6 shows three selected screenshots from the Client Application. On the left, the three panels are indicated. From the *mode* combo box of the configuration panel, the user chooses between the *automatic*, the *professional* and the *view history* modes. In the middle, Figure 6 shows the *automatic* mode. It is for the majority of users who do not worry about which components are installed. In this mode, a user just need to select an existing profile from the profile combo box and then the installation is done automatically for him. On the right, the *professional* mode is illustrated. It aims at users with a stronger technical background. In this mode, the user is free to combine components as he likes. This also includes the choice of which version of the component to use. In the *view history* mode *(not shown in the figure)*, the user takes a look at the installation history on the respective mobile device.

5. Applied Technologies and the Experiences with Them

As it is true for desktop computers, there is competition between runtime environments and developments tools for mobile devices. The two main competitors are J2ME [6] from Sun Microsystems and .NET Compact Framework [7] [8] from Microsoft Cooperation. Both are compact versions and promise “write once, run everywhere” by building upon the integrated Byte-Code interpreter.

With regard to the *time-to-market* factor, .NET Compact Framework gives better support to new technologies like web services, since it has just been published by Microsoft this year. Furthermore .NET Compact Framework offers better access to the resources of the operating system since it is usually provided by the same company.

In the DIANE project, we implement two applications for the mobile devices (Pocket PC 2002), one in unmanaged native code (the so-called *Initiator*) which runs directly on the operating system and the other one in managed code (the so-called *Client Application*) which runs on the base of compact framework. According to our experiences, the development time for the managed application is shorter than the one in native code because of the support from the compact framework and the advance of the programming language VS C#. Due to the extra steps that the compact framework needs to compile the byte code on the fly, it takes longer to execute a managed application. Yet, this additional delay seems to be acceptable.

During implementation of the Initiator and the Client Application, we have not only debugged on the emulator included in the Pocket PC 2002 SDK [9] but also on a real device. The Pocket PC emulator worked fine during the whole development phase but the real device has raised problems during the debug process. It often failed because the development environment can not establish a connection to the mobile device. According to our experience, the only solution is to reinstall the whole development environment. It is not very efficient but effective.

The aforementioned version control system manage the concurrent versions of the software components. For this purpose, we deploy a CVS server converted for the use under Windows 2000 in our project [10].

We have chosen Web Services [5] as the technology for the implementation of the software component service. As a machine-readable web application, Web Services are language and platform independent because of their common industry standards: XML, SOAP, and WSDL etc. Their self-descriptive and self-discoverable characteristics facilitate deployment and use of Web Services. Using the development pack, it is seems to be feasible to develop the respective Web Services.

6. Usage of the Software Station

In this section, we take a closer look at the different steps in which the administrator, the developer, and the user are involved while using the Software Station.

At first, the *administrator* compiles a list of software components that are released for the user and the deployment configuration document which configures the software components into modules and profiles. After that, he makes these two documents available for the component service through an integrated interface of the component service. The component service gets these documents, parses them, saves the result locally, and starts to monitor the update event from CVS server.

A *software developer* compiles all of his program files into a single program library whose name is listed in the component list composed by the administrator earlier. After that the software component is checked in to the CVS server using a CVS client. The component service handles the new software component automatically and respectively.

For a *user*, the initial work is to use the Initiator to install all the runtime environments and the Client Application on his device. Therefore, he visits our web page and executes the initiator directly from the internet. After the installation, he executes the Client Application, selects the installation mode and, if necessary, specifies also the profile provided by the administrator. After confirming his selection, the Client Application begins to download and install the selected software components without any intervention of the user.

In the lab course “Mobile Databases” of the summer semester 2003 [12], the Software Station has been successively applied for the deployment of various Java applications to the Pocket PC 2002. The evaluation shows that the requirements of the Software Station are met.

7. Conclusion

In this paper, we have discussed the problems which occur during the lifecycle of mobile applications from the software engineering point of view. Because of the characteristics of mobiles devices, it is difficult to develop and deploy software for mobile device. In order to resolve this problem, we have suggested a system that automates parts of the development process and the deployment process. We use a central software component repository and deliver software component through a web service to the Client Application on the mobile device. Furthermore, the available technologies for mobile devices have been compared and we have shared our experiences of applying them in our project. Lastly, the usage of the system is explained from the administrator, developer, and user's point of view.

References

[1] *Koenig-Ries, B., Klein, M.*: Information services to support e-learning in ad-hoc networks. In: First International Workshop on Wireless Information Systems (WIS2002). (2002) 13–24

[2] *Klein, M., Koenig-Ries, B., Obreiter, P.*: Service rings – a semantical overlay for service discovery in ad hoc networks. In. The Sixth International Workshop on Network-Based Information Systems (NBIS2003), Workshop at DEXA 2003, Prague, Czech Republic. (2003)

[3] *Klein, M., Koenig-Ries, B., Obreiter, P.*: Lanes – a lightweight overlay for service discovery in mobile ad hoc network. In. 3rd Workshop on Applications and Services in Wireless Networks (ASWN2003). Berne, Swiss.(2003)

[4] *IPD, University of Karlsruhe, Germany*: DIANE Project, http://www.ipd.uka.de/DIANE

[5] *WebServices.org*: http://webservices.org

[6] *Sun Microsystems, Inc.*: Personal Java^{TM} Technology White Paper, http://java.sun.com/products/personaljava/

[7] *MSDN, Microsoft*: Development Tools for Mobile and Embedded Applications, http://msdn.microsoft.com/library/en-us/dnppcgen/html/mobdevtools.asp

[8] *MSDN, Microsoft*: Microsoft .NET Compact Framework Overview, http://msdn.microsoft.com/vstudio/device/compactfx.aspx

[9] *Microsoft: Pocket PC 2002 SDK* http://www.microsoft.com/mobile/developer/downloads/ppcsdk2002.asp

[10] *CVSNT*: http://www.cvsnt.com

[11] *Ian Sommerville*: Software Engineering (6.th edition), Addison Wesley. (2002)

[12] *Koenig-Ries, B., Klein, M.*: Auf der Jagd nach mobilen Informationen - Erfahrungsbericht zum Feldversuch des Praktikums "Mobile Datenbanken". 2nd Workshop of the GI-Arbeitskreis "Mobile Datenbanken und Informationssysteme": "Persistence, Scalability, Transactions -database mechanisms for mobile applications", Karlsruhe, Germany, April 10-11, 2003