PAAL, Stefan
KAMMÜLLER, Reiner
FREISLEBEN, Bernd

A Cross-Platform Application Environment for Nomadic Desktop Computing

published on netzspannung.org:
http://netzspannung.org/about/mars/projects/pdf/awake-2004-7-en.pdf
14 March 2005

First published: Proc. of the 5th International Conference for Objects, Components, Architectures, Services and Applications for a Networked World (NODE 2004). Erfurt: Springer, 2004.

A Cross-Platform Application Environment for Nomadic Desktop Computing

Stefan Paal[1], Reiner Kammüller[2], Bernd Freisleben[3]

[1] Fraunhofer Institute for Media Communication
Schloss Birlinghoven, D-53754 Sankt Augustin, Germany
stefan.paal@imk.fraunhofer.de
[2] Department of Electrical Engineering and Computer Science, University of Siegen
Hölderlinstr. 3, D-57068 Siegen, Germany
kammueller@pd.et-inf.uni-siegen.de
[3] Department of Mathematics and Computer Science, University of Marburg
Hans-Meerwein-Strasse, D-35032 Marburg, Germany
freisleb@informatik.uni-marburg.de

Abstract. The possibility to uniformly access the WWW using a standard web browser has fostered the development of *nomadic desktop computing*, allowing nomadic users to run their applications from nearly any location providing access to the Internet. In this paper, we propose an approach to nomadic desktop computing based on the idea of dynamically deploying and executing personalized applications on the desktop system currently used by a nomadic user. We present a *cross-platform application environment* that automatically adapts itself to the requirements and configuration of a nomadic desktop application and enables the seamless execution and migration of applications across heterogeneous desktop computer systems. The implementation of our approach is outlined and its use in ongoing research projects is demonstrated.

1 Introduction

The major goal of the Internet was to link spatially distributed computing resources in the real world into a virtual computing environment where their physical location becomes less important or even completely unknown to the user [1]. With the advent of the WWW, information published on HTML pages by web servers got transparently accessible from everywhere using an ordinary web browser. Subsequently, several proposals have been made to establish the web browser as the universal user interface for *nomadic desktop computing* [2]. Instead of deploying an application on each desktop system, it is centrally installed and executed on an application server [3]; a standard web browser on a desktop system is then used by the nomadic user to access the application on the application server, as shown in Fig. 1.

However, an HTML interface provided by a web browser can not really substitute the rich user interface of a native desktop application. Thus, several attempts have been made to provide a comparable user interface by implementing the *thin-client approach* and running a desktop Java applet [4]. This approach requires less administration effort due to the centralized installation of an application, but it heavily relies on

the processing power of the application server and the availability of a permanent network connection to access the remote application.

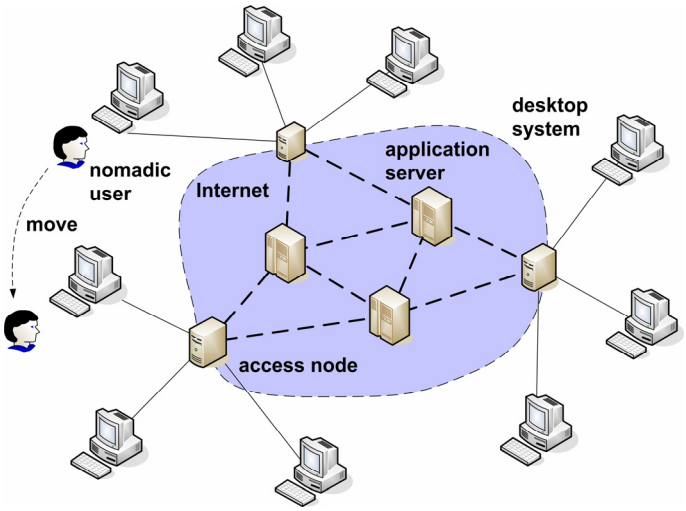

Fig. 1. A nomadic desktop computing scenario

In this paper, we propose a different approach based on the idea of so called *nomadic desktop applications*. Instead of installing and executing applications on certain application servers, they are dynamically deployed and executed on the desktop system currently used by a nomadic user. In this scenario, a desktop computer system is alternately shared among different and unknown users. Thus, a fundamental problem is the dynamic and automatic adaptation of the current desktop computer to transparently provide a pervasive desktop environment to each user and application. Clearly, in practice it is not possible to prepare and maintain a single installation or configure all applications which might be used in advance. We address this basic problem by presenting a *cross-platform application environment*. It automatically adapts itself without user intervention to provide a suitable application environment for each deployed application across heterogeneous desktop computer systems. In addition, it enables the migration of application and user configurations among different hosts.

The paper is organized as follows. In section 2, we discuss the features and requirements of nomadic desktop computing and present related work. Section 3 presents our approach towards a cross-platform application environment and illustrates its realization in Java. The application of our approach is demonstrated in section 4. Section 5 concludes the paper and outlines areas of future work.

2 Nomadic Desktop Computing

In the following, we highlight the goals of nomadic desktop computing and identify the requirements to provide a suitable nomadic application environment. Then, we discuss related work and summarize our findings.

2.1 Goals

There are several models and visions of nomadic computing [2, 5, 6, 7], and many of them are mixed up with different variants of mobile computing, ubiquitous computing and pervasive computing. While some approaches are focusing on integrating computing systems into everyday scenarios and devices like car navigation systems or inventing new mobile devices like wearables, our focus is to particularly support nomadic users employing different desktop computing devices [2]. The basic idea is to enable people to use any desktop computer to run applications and to separate the one-to-one relationship between the *desktop computer* and the *nomadic user* as well between the *desktop computer* and the *desktop application*. Nomadic users can travel around and employ various desktop computers to work with their desktop applications, hence creating a *pervasive desktop environment*. Consequently, instead of having a desktop application installed on a single personal computer, the involved application has to travel with the nomadic user and become itself a *nomadic desktop application* which is dynamically deployed on unmanaged nodes, as shown in Fig. 2.

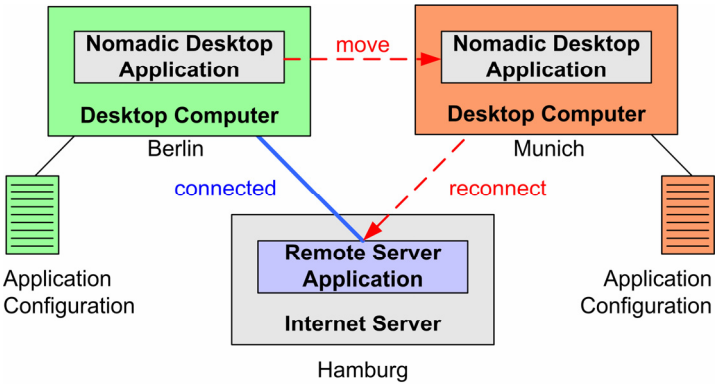

Fig. 2. Nomadic desktop computing

A crucial problem in this scenario is to provide the illusion of a pervasive desktop environment that is independent of a specific host. This does not only include the automatic deployment and composition of application binaries, but also the migration of customized application configurations. In addition, the access to remote resources, such as the server application in Hamburg in Fig. 2, should be transparently reconfigured according to the current platform capabilities. The user should not be bothered with manual application configuration tasks but in contrast the desktop computer should adapt itself and provide a suitable application environment on the fly.

2.2 Requirements

The functional requirements of an application environment supporting nomadic desktop applications are as follows:

Runtime Environment
An application environment has to support nomadic desktop applications which may migrate to another desktop computer when the user moves. In effect, the desktop computer does not know in advance which application will be employed but it has to adapt itself and to provide a suitable runtime environment according to the platform capabilities and application requirements.

Self-Management
While a desktop computer in a local network environment is well-known and can be easily managed, it is practically impossible to manage all possible desktop computers in a heterogeneous Internet environment. As a result, a potential Internet desktop computer has to self-manage the aspects of application deployment and composition as well as dynamic hosting and configuration of nomadic desktop applications.

Cross-Personalization
While traveling around, nomadic users and desktop applications pass different desktop systems. An important requirement for creating a pervasive desktop environment is the personal configuration and customization of the current desktop system [8]. With respect to current platform capabilities, each desktop system should offer a personalized application environment to the nomadic user and the desktop application.

Multi-Application Management
In order to share commonly required components and to enable the easy collaboration and integration of multiple started desktop applications like in a personal desktop computing system, a cross-platform application environment should offer facilities to concurrently host and manage multiple applications in a shared or several separated runtime environments.

Resource Sharing
If a nomadic user is moving to another desktop computer where some application components have already been deployed by a former nomadic user, the application environment should reuse these components and not download and maintain exclusive copies for each application installation. Moreover, common configurations of the currently involved desktop computing systems should be shared among applications.

2.3 Related Work

In the following, we examine various approaches proposed in the literature to support the requirements of nomadic desktop computing.

Native Application Environments
A desktop computer is typically used exclusively by a single user. The installed operating system represents a *native application environment* bound to certain hardware. It is usually managed by a single administrator who ensures that all resources (e.g. li-

braries) are installed and properly configured in advance. This pre-installation is usually perfectly tailored to the needs of a single user and does typically not allow hot deployment of new applications. Instead, the administrator has to manually install and configure each application. Moreover, different computer systems are used, and the required application binaries as well as the capabilities of the installed operating system vary from version to version. Therefore, native application environments basically do not support seamless migration of an application to another host, e.g. a Linux binary can not be executed on a MS Windows system. Finally, native application environments are not expected to migrate application configurations across heterogeneous desktop systems, although there are approaches which synchronize user profiles on a central server and retrieve them when the user logs in to another host, e.g. using Microsoft ADS, Novell NDS or LDAP [9]. As a result, nomadic desktop applications which rely on a native application environment and the related binary format can not be easily deployed across different desktop computers.

Virtual Application Environments

A different approach is so called *virtual application environment*. Instead of tightly coupling the application binaries with the operating system, they typically employ an intermediate application format. While some approaches like Flash or Shockwave [10] are typically tailored to be used as a plugin in a web browser, other approaches like Perl or Python [11] do not really support desktop computing with a graphical user interface. In contrast, Sun's Java comes with a full-fledged Java Runtime Environment (JRE) which provides a uniform application system across heterogeneous platforms. Actually, it is concurrently installed with a standard browser as a plugin and is thus available on nearly every desktop system. The introduction of Java applets allowed simple desktop applications to be deployed on desktop computer systems using a web browser. However, an applet could not be executed offline and independent of the browser. With the invention of Sun Java Web Start (JWS), the web browser was only needed for requesting an application but then it could be used offline as well as independent of the browser [12]. Furthermore, with Java Web Start, software deployment was simplified by introducing additional features such as automatic caching and updating of already downloaded components as well as sharing them across various desktop applications. Finally, application development was made as simple as developing a local application. There was no need to employ particular programming models as with Java applets. However, even though Java Web Start helped to seamlessly deploy Java applications across heterogeneous Internet nodes, it still lacks some important features required for a nomadic application system. First of all, there is no support for the migration of an application from one desktop system to another. In contrast, an application is independently installed on each desktop system and there is no synchronization with already existing configurations on another one. Moreover, it is not able to directly communicate with JAR repositories. Instead, a *Java Native Launch Protocol (JNLP)* configuration file must be downloaded and evaluated from a web server which can not be personalized by the user.

Remote Application Environments

Other approaches avoid deploying desktop applications at all and rely on a *remote application environment*. For instance, the project *Cooltown* is a Hewlett-Packard research project which follows the *smart space* solution of web-based nomadic computing [1, 13]. It utilizes existing standard Internet technologies like a web browser, HTTP, XML and URL. Thus, it avoids the inherent complications associated with ubiquitous computing, such as programming language, operating system and hardware dependencies. A similar approach called *crossware* was already proposed by Netscape in the mid-nineties but never reached broad acceptance [14, 15]. Another example is the *network computer* approach developed a few years ago mainly by Sun and Oracle [4]. The crucial idea was to replace a desktop computer by a computer system which basically consists of a screen, a keyboard and some memory. Instead of deploying and installing applications on each desktop computer, everything including the configurations and user profiles are managed on an application server which also provides processing power to execute the application. Similar to the XWindows system, the network computer is used to connect to the application server and to display the related user interface, following a *thin-client* approach. As a result, the user can access every application and all of her or his files from each connected network computer using a uniform desktop interface. In addition, the administration of the system is limited to a few application servers. Nevertheless, the original network computer approach did never receive widespread acceptance, possibly because network computers could not be used offline but only online. A variant is the use of a native operating system to provide a graphical user interface which is connected to an application running on a remote application server as in [16]. Instead of executing the application on the desktop, only the required display data is transferred from and to the application server. However, this also inherits the online problem and it is limited to a certain operating system supporting the user interface.

2.4 Summary

The presented approaches basically differ in their support for heterogeneous desktop systems. While native application environments are inherently bound to specific operating systems, virtual and remote application environments are actually independent of the underlying platform capabilities. Although MS .NET actually represents a virtual application environment, it is still only available as an attachment to MS Windows and can not be employed on different operating systems [17].

Furthermore, there is also a difference in the customization of the application environment according to personalized application configurations. Approaches based on Java Web Start do not support the modification of a given application configuration within a JNLP file. In contrast, native application environments may be independently customized but the resulting configuration is not always portable to another employed desktop system. Consequently, the personalized configuration options left on the formerly employed desktop system are lost. Concerning the sharing of resources such as downloaded components, there is also the problem that application systems typically provide a separated runtime environment to each deployed application. For instance, a Java application hosted by Java Web Start is usually isolated from other concurrently

hosted applications and is not able to share common resources, e.g. already downloaded application components.

In summary, there are various approaches towards the distributed deployment of desktop applications. While they have proven their suitability in specific application scenarios, they typically introduce certain application models which are not suitable for all scenarios (such as Java applets and offline operation) or rely on fixed configurations (such as Java Web Start and its JNLP files). As a result, we think that there is still the need for a *cross-platform application environment* which is suitable to customize itself in a self-managed way according to the needs of the nomadic desktop application and the personalization of the nomadic user.

3 A Cross-Platform Application Environment

In this section, we present our approach towards a cross-platform application environment. We discuss its basic concepts and features concerning nomadic desktop applications. Finally, we illustrate its implementation and present an example of its use.

3.1 Conceptual Approach

A basic problem of nomadic desktop computing is the provision of a suitable application environment that is uniformly customized across different desktop systems following application-specific configuration requests and user-specific personalization options. We address this problem by separating platform, application and user related concerns and introduce various configuration options, as shown in Fig. 3.

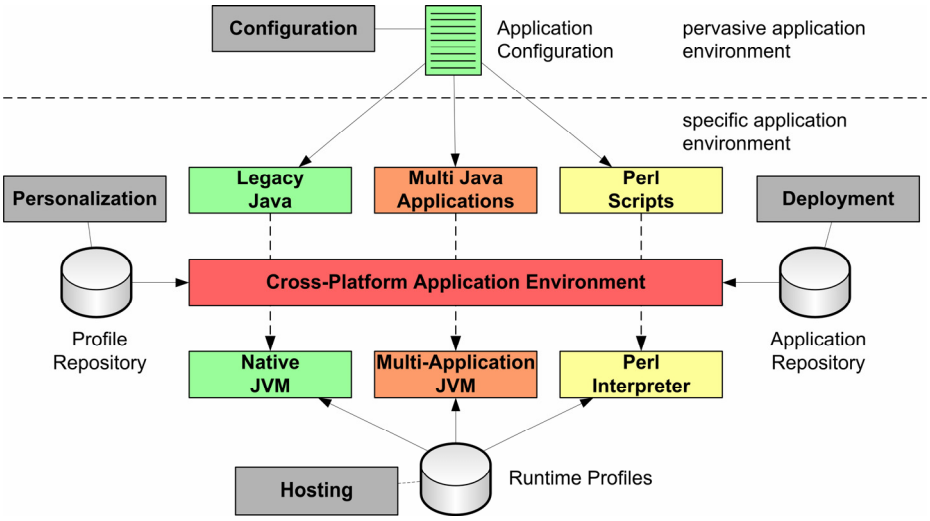

Fig. 3. Cross-Platform Application Environment

The basic idea of our approach is to provide a pervasive application environment on top of a specific application environment which is customized by the *application configuration* passed by the nomadic user. In detail, particular *runtime profiles* are responsible for defining the hosting capabilities of the platform and are maintained by the platform administrator. The *application repository* is used by application developer to deploy application components and to describe the dependencies on other components. The *profile repository* contains the personalization for each nomadic user and nomadic application. In contrast to existing approaches, the cross-platform application environment does neither impose a single application model nor how the runtime environments are prepared or the required application components are deployed. It can be easily extended with additional runtime variants like Java or Perl whereby each extension is responsible for interpreting the passed application configuration and to prepare a suitable application environment.

3.2 Realization

Our current prototype of the proposed cross-platform application environment is implemented in Java and therefore greatly benefits from the uniform runtime environment provided by the Java Virtual Machine across heterogeneous operating systems and platforms. In addition, we have virtualized further aspects of nomadic desktop computing, such as the *deployment* and *composition* of Java applications. For instance, the composition of multiple Java applications within a single JVM is decoupled from the actual deployment scenario found on a certain platform, as shown in Fig. 4. Commonly used resources and classes can be shared and do not have to be loaded in separate JVM. Moreover, the collaboration of concurrently used applications is facilitated, e.g. exchanging data using object references.

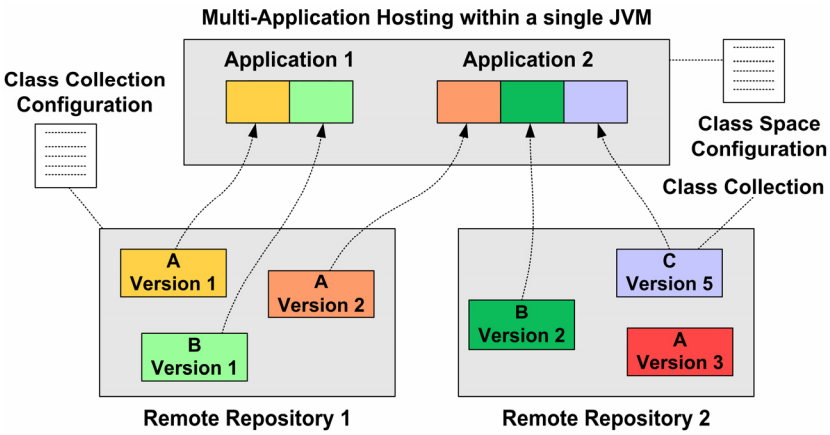

Fig. 4. Multi-application hosting within a single JVM

Class collections [18] define the location as well as the content of a class repository. In turn, *class space* [19, 20] configurations are used to organize which classes may be loaded by the JVM, which are shared with and which are shielded from other

concurrently loaded applications. Further, starting a single Java application by passing the class containing the static method *main* on the command line is not possible when multiple applications are hosted within the same JVM. Thus, we determine the *main* method using *Java Reflection* [21] and finally call it after having initialized the related application environment, as shown in fig. 5.

```
app = createClassSpace("application");
Class mainClass = app.loadClass(szMain);
String[] args = new String[mainArgs.size()];
mainArgs.toArray(args);
Object params[]={ args }; Class t[]={ String[].class };
Method meth = mainClass.getDeclaredMethod("main", t);
meth.invoke(null, params);
```

Fig. 5. Determining the main method using Java reflection

In the first line, the class space is created and configured to host the application. After that, the application class containing *main* is loaded and the subsequent call with *invoke* is dynamically assembled and performed using Java reflection. Further details about *class collections* and *class spaces* are described in [18, 19, 20, 22].

Apart from the composition and hosting of multiple Java application, there is a related problem concerning the control of each application and its threads within a single JVM. The native JRE offers only little support for managing and signaling Java threads. For instance, there is no standard way to control a *foreign* thread or to process received signals in a pre-defined way, e.g. registering signal handlers. This means that for a multi-application environment there is no way to implement a common *task manager* to uniformly control various applications, threads and processes. In turn, an application can not be informed when a certain event is issued, e.g. when the application environment is shut down. As a solution, we developed a *managed thread model* which transparently wraps the original Java threads and provides synchronous and asynchronous access to each created thread within the JVM, as shown in Fig. 6.

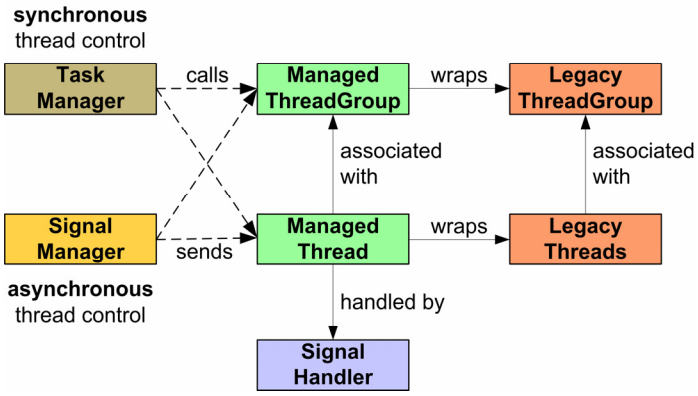

Fig. 6. Managed thread model and signal handling in Java

The implementation is mainly based on the Java feature of *thread group inheritance* which automatically associates each newly created thread with the current

thread group [23]. At the startup of the application environment, we create a particular managed thread group which is then able to address all later created threads. This way we can distinguish legacy and managed threads and can trigger callbacks that are used by the foreign thread to process asynchronous thread signals and synchronous thread control requests. In contrast to the native thread model where a thread is not longer allowed to call suspend, resume or stop, the managed thread model offers an alternative way to implement thread control. Of course, this assumes that the target thread is not refusing to be controlled but is regularly calling the signal handler method.

3.3 Use

A basic problem of a cross-platform application environment is its initial deployment on separately managed and remotely located computing systems. To this end, we distribute a Java-based management component with the Java Network Launch Protocol (JNLP). It is then used to load and start the requested applications as described in [20]. Thus, using the JNLP, the presented approach can be basically employed on any target system where a native Java Runtime Environment has been previously installed, e.g. as part of an Internet browser installation. If the nomadic user wants to move to another desktop computing system, he or she closes all applications and the application configuration is written back to the profile repository. He or she then moves and starts the cross-platform application environment on the next computer, the application profiles are retrieved from the profile repository and the user is able to start his or her applications again with personalized settings. In the following, we describe step-by-step how the actual cross-platform application environment is configured.

Step 1: Runtime Profiles (Platform Administrator)
Apart from launching Java applications within the same JVM as described in [20], we now use runtime profiles to specify additional runtime variants, as shown in fig. 7.

```
<application-runtime id="{8A750732}">
    <property name="runtime" value="native-java" />
    <property name="version" value="1.4.2" />
</application-runtime>
```

Fig. 7. Runtime profile

The runtime profile is used to specify a common configuration setup which can be referenced by the application deployment, as depicted below. Its major objective is to configure runtime profiles depending on the resources of the underlying platform instead of the yet unknown applications. In the example above, we define a runtime profile for launching an application in a separate JVM using Java version 1.4.2.

Step 2: Collection Deployment (Component Developer)
We start with the deployment of a Java class collection. The developer has to create a class collection configuration with metadata about the collection, such as version property and where the Java classes can be actually downloaded. In addition, the re-

quired classes can be further selected by specifying regular expressions, as shown in Fig. 8. Only matching classes are loaded from the specified repositories.

```
<collection id="{553C6E73}">
    <property name="version" value="1.0" />
    <repository url="http://crossware.org/clock.jar" />
        <resource name="org/crossware/clock/.*"/>
    </repository>
</collection>
```

Fig. 8. Class deployment using class collections

Step 3: Application Deployment (Application Deployer)
Next, the *application deployer* has to specify the required runtime environment in a separate application deployment configuration, as shown in Fig. 9. Along with that, she or he defines the required collections defined in the second step and which ones are shared or shielded using a class space configuration. The purpose of this separation is to enable the platform to look for alternative compatible collections in case the exact collection is not available on the current platform. In addition, the main class where the application execution starts has to be specified.

```
<application-environment id="{DC488997}">
    <runtime="native-java" />
    <main-class="org.crossware.clock.Main" />
    <classspace name="shared">
        <collection id="{553C6E73}" />
            <property name="version" value="1.0"/>
        </collection> </classspace>
    <classspace name="shielded">
        <collection id="{3283A542}">
            <property name="version" value="2.2"/>
        </collection> </classspace>
</application-environment>
```

Fig. 9. Application deployment

Step 4: Application Configuration (Application Installer)
Another configuration is dynamically passed to a cross-platform application system when an application is to be deployed, e.g. a nomadic application migrating from one host to another. The application configuration shown in Fig. 10 contains a unique reference to the deployment description as well as a reference to the application profile. Both are evaluated by the cross-platform application environment and used to provide an appropriately configured runtime environment.

```
<application-configuration id="{5C015A86}">
    <deployment="{DC488997}"/> <profile="{DB92B18C}"/>
</application-configuration>
```

Fig. 10. Application configuration

We want to point out that the application description does not contain any reference to the actual implementation, e.g. package name, JAR file location or name of

the main class to start the application. Instead these settings are dynamically determined by the application environment evaluating the application deployment, collection deployment and runtime profile.

Step 5: Application Profile (Application User)
Finally, the application profile used for the personalization of an application installation is retrieved by the application system before the application is started. As shown in Fig. 11, the profile contains an individual display name and icon as well as certain parameters passed to the application.

```
<application-profile id="{DB92B18C}">
    <name="Clock" />
    <icon="class://org/crossware/clock/clock.png" />
    <args name="format" value="digital" />
</application-profile>
```

Fig. 11. Application profile

This example represents only a simple view of the system. Of course, real collection configurations and application profiles are more complex than the mentioned ones. They are also usually not edited manually but with related tools.

3.4 Discussion

In this section, we highlight the basic features of the proposed cross-platform application environment with respect to nomadic desktop computing.

Separation of Concerns
We have identified various tasks for employing an application, such as deployment, composition, configuration, personalization and hosting of an application. Consequently, we have separated these concerns and provide options to customize each independently. In detail, the desktop computer system can be configured regardless of the nomadic desktop application. In turn, the application developer can also implement his or her application without actually knowing where it will be employed.

Extensible Runtime Support
Our cross-platform application environment is not limited to employ a certain runtime environment but is supposed to prepare and control arbitrary runtime environments as long as there is a suitable runtime profile available on the current platform. For this purpose, it can be extended with additional runtime configurator plugins. In effect, a particular application environment may be prepared and started in a separate process and window, e.g. executing a native C++ command line tool.

Self-Managed Customization
The required customization and provision of a suitable runtime environment is performed without intervention of the nomadic user. This is a particular feature concern-

ing a pervasive application environment that travels with a nomadic user and does not force him or her to manually configure and synchronize different application installations on each involved platform. Moreover, due to the diversity of platform configuration and capabilities, it would be actually not possible for the user to perform this task in heterogeneous environments such as the Internet.

Personalized Application Profiles
A basic objective of a pervasive application environment is cross-platform personalization. To this end, we have introduced a user-specific application profile which is used to maintain the application personalization independent of a particular desktop system and is synchronized with locally modified application profiles. Apart from personalizing a single application, it is also used to personalize selected settings specific to a certain host like the address of the proxy server.

Multi-Application Management
The concurrent hosting of various applications is a typical feature of desktop computing, though it is often a complex task. Especially for Java applications we have introduced so called *class spaces* that allow organizing the classes of more than one Java application within the same JVM. Moreover, we have also invented a new thread control model that allows uniformly controlling and signaling *foreign* Java threads of concurrently loaded Java applications.

Resource Sharing
Commonly used resources like an XML parser can be shared among concurrently hosted applications in a multi-application environment. Apart from eventually downloading required components and the preparation of a suitable runtime environment, this does also not cause further performance overhead with respect to the execution of an application in a single platform environment.

4 Application of the Approach

In the following, we present a so called *Internet Application Workbench* which has been built as part of our cross-platform application environment to provide a graphical user interface. Currently, we use it in our project netzspannung.org [24] as an advanced workspace interface to access its document pools and start certain applications, as shown in Fig. 12.

The basic purpose of the workbench is to provide a uniform graphical interface to access nomadic desktop applications from arbitrary Internet hosts. For this purpose, the workbench is implemented in Java and initially deployed using Sun Java Web Start. It comes with a basic configuration for a Java application environment, e.g. the location of a Java application repository, as described in [18]. Thus, the user can immediately start to request and configure Java applications and run them within the workbench. In detail, the workbench can host multiple Java applications within the same JVM, which is realized by self-organizing Java class loaders using class spaces,

as described in [19]. Concerning the way how users work with multiple applications within a native single desktop system, our Internet application workbench represents a *cross-platform desktop system* which is dynamically adapted to the current platform without bothering the user with manual configuration or installation tasks. In fact, the user is not aware of this adaptation and always gets the illusion of a pervasive application environment when moving across various desktop computers.

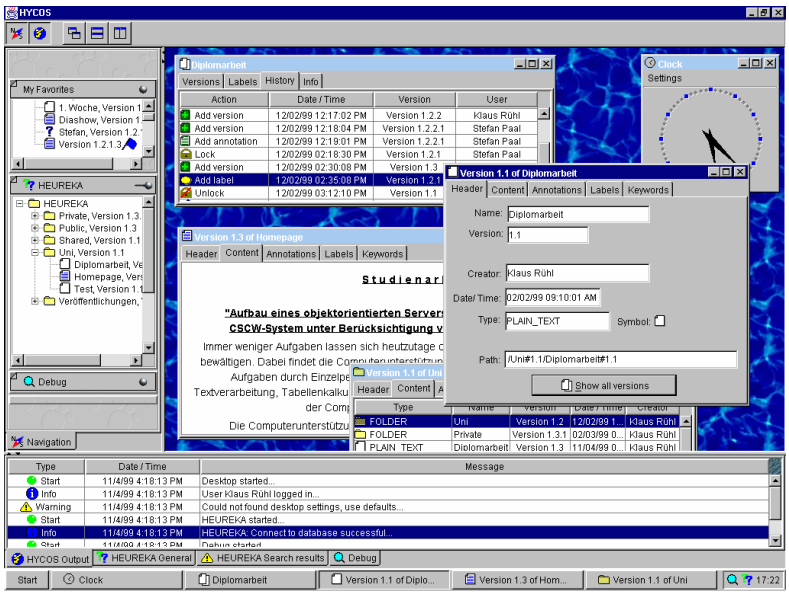

Fig. 12. Internet Application Workbench

In addition, the user can further configure the application environment according to the platform capabilities as well as specific program settings like the path to a Perl interpreter. The workbench can then be used to some degree as a program launcher for running legacy applications in a separated process. It downloads the suitable application binary from a remote application repository, passes the appropriate program parameters to the operating system and controls the resulting process. However, the automatic migration of customized application configurations is typically not possible due to application specific approaches to store the configuration data.

5 Conclusions

In this paper, we have discussed the goals and requirements for nomadic desktop computing. Since existing approaches are either limited to fixed application scenarios, do not support the seamless migration of a desktop application and its configuration from one desktop system to another, or are not capable to synchronize user and multiple application profiles across various Internet hosts, we have presented our development of a cross-platform application environment that tackles these problems by sepa-

rating the concerns of application deployment and composition, runtime customization and personalization by introducing various configuration files such as application descriptions, platform-specific runtime profiles and personalized profiles. Its implementation in Java, utilizing Java Web Start for the initial deployment on a new desktop computer system, was described. Finally, we have introduced a managed thread model that allows to uniformly controlling and signaling threads of multiple loaded Java applications. Our approach has been used to create an Internet application workbench that provides a uniform graphical interface across various Internet hosts. In effect, nomadic desktop users get the illusion of a pervasive application environment that travels with them and presents their personalized applications on different desktop computers in a uniform way.

There are several areas for future work. For example, the development of the cross-platform application environment is still going on. A major drawback of the current implementation is the lack of security features and protection against malicious applications. This problem could be solved by introducing cryptographic keys to protect downloaded and cached application components as well as application profiles. Another ongoing development is directed towards recommendation of application profiles. Currently, we investigate a recommender solution that evaluates existing profiles of other users which have already employed the same application and assists the user in creating the initial personalization. Finally, we use the current implementation only as part of our own research projects. After the first release, we want to test and evaluate it in further application scenarios, e.g. deploying the Internet Application Workbench as an advanced pervasive interface to netzspannung.org and its knowledge discovery tools [26].

Acknowledgements

The presented approach has been applied in the ongoing research projects CAT [25] and AWAKE [26] which are financially supported by the German Ministry of Education and Research (BMBF). The projects are conducted by the research group MARS of the Fraunhofer Institute for Media Communication, Sankt Augustin in cooperation with the University of Siegen and the University of Marburg, Germany. Special thanks go to Monika Fleischmann, Wolfgang Strauss, Jasminko Novak and Daniel Pfuhl.

References

1. Kindberg, T., Barton, J. A Web-Based Nomadic Computing System. Computer Networks. Vol. 35, Nr. 4. Elsevier 2001. pp. 443-456.
2. Amor, D. Internet Future Strategies: How Pervasive Computing Services Will Change the World. Prentice Hall 2001.
3. Zhu, J., Törö, M., Leung, V.C.M., Vuong, S. Supporting Universal Personal Computing on Internet with Java and CORBA. Concurrency: Practice and Experience, Vol. 10, Nr. 11-13. John Wiley & Sons 1998. pp. 1007-1013.

4. Gentner, D., Ludolph, F., Ryan, C. Designing HotJava Views. JavaSoft 1997. http://java.sun.com/products/hotjavaviews/hjv.white.html
5. Kleinrock, L. Nomadic Computing and Smart Spaces. IEEE Internet Computing. Vol. 4, Nr. 1. IEEE 2000. pp. 52-53.
6. Kleinrock, L. Breaking Loose. Communications of the ACM. Vol. 44, Nr. 9. ACM 2001. pp. 41-45.
7. Lyytinen, K., Yoo, Y. The Next Wave of Nomadic Computing. Information System Research. Vol. 13, Nr. 4. Informs 2002. pp. 377-388.
8. Wood, K. R., Richardson, T., Bennett, F., Harter, A., Hopper, A. Global Teleporting with Java: Toward Ubiquitous Personalized Computing. IEEE Computer. Vol. 30, Nr. 2. IEEE 1997. pp. 53-59.
9. Sheresh, D., Sheresh, B. Understanding Directory Services. Macmillan Computer Pub 1999.
10. Kerman, P. Macromedia Flash MX 2004 for Rich Internet Applications. New Riders Publishing 2003.
11. Wall, L., Christiansen, T., Orwant, J. Programming Perl. O'Reilly 2000.
12. Srinivas, R. N. Java Web Start to the Rescue. JavaWorld. IDG 2001. Nr. 7. http://www.javaworld.com/javaworld/jw-07-2001/jw-0706-webstart_p.html
13. Cooltown. HP Research Labs 2004. http://www.cooltown.com/cooltown/index.asp
14. Cusumano, M. A., Yoffie, D. B. What Netscape learned from Cross-Platform Software Development. Communications of the ACM. Vol. 42, Nr. 10. pp. 72-78. ACM 1999.
15. Andreessen, M. Building Crossware. Netscape Techvision 1997. http://wp.netscape.com/columns/techvision/crossware.html
16. Extent Solutions. http://www.exent.com/solutions/products.asp.
17. Meyer, B. .NET is coming. IEEE Computer. Vol. 34, Nr. 8. IEEE 2001. pp. 92-97.
18. Paal, S., Kammüller, R., Freisleben, B. Java Class Deployment with Class Collections. Proc. of the 2003 Int. Conf. on Objects, Components, Architectures, Services, and Applications for a NetworkedWorld. Erfurt, Germany., LNCS 2591, Springer-Verlag, 2003. pp. 135-151.
19. Paal, S., Kammüller, R., Freisleben, B. Java Class Separation for Multi-Application Hosting. In Proc. of the 3rd Conference on Internet Computing (IC 2002). Las Vegas, USA. CSREA Press, 2002. pp. 259-266.
20. Paal, S., Kammüller, R., Freisleben, B. Customizable Deployment, Composition and Hosting of Distributed Java Applications. Proc. of the 4th Int. Symposium on Distributed Objects and Applications (DOA 2002). Irvine, USA, LNCS 2519, Springer-Verlag, 2002. pp. 845-865.
21. Richmond, M., Noble, J. Reflections on Remote Reflection. Proc. of the 24th Australasian Computer Science Conference (ACSC 2001). IEEE 2001. pp. 163-170.
22. Paal, S., Kammüller, R., Freisleben, B. Separating the Concerns of Distributed Deployment and Dynamic Composition in Internet Application Systems. Proc. of the 5th Int. Symposium on Distributed Objects and Applications (DOA 2003). Catania, Italy., LNCS 2888, Springer-Verlag, pp. 1292-1311.
23. Venners, B. Inside The Java 2 Virtual Machine. McGraw-Hill. 1999.
24. netzspannung.org, Communication Platform for Digital Art and Media Culture. http://netzspannung.org
25. Fleischmann, M., Strauss, W., Novak, J., Paal, S., Müller, B., Blome, G., Peranovic, P., Seibert, C., Schneider, M. netzspannung.org - An Internet Media Lab for Knowledge Discovery in Mixed Realities. In Proc. of 1st Conference on Artistic, Cultural and Scientific Aspects of Experimental Media Spaces (CAST01). St. Augustin, Germany. pp. 121-129. Fraunhofer 2001.
26. AWAKE - Networked Awareness for Knowledge Discovery. Fraunhofer Institute for Media Communication. St. Augustin, Germany. 2003. http://awake.imk.fraunhofer.de