On Integrating Mobile Devices into a Workflow Management Scenario

Stefan Müller–Wilken, Frank Wienberg and Winfried Lamersdorf
University of Hamburg, Germany
{smueller,wienberg,lamersd}@informatik.uni-hamburg.de

Abstract

While the desire to gain full access to stationary information sources (e.g. the company´s backoffice database) from abroad is only natural, inherent design limitations such as a lack of computational power, extremely limited resources and closedness to modifications, let mobile system integration be still a difficult issue. In contrast to other approaches to the field, the "Java Border Service Architecture" introduced in this article does not require modifications to the mobile device´s system software or the application environment to be accessed from the mobile system.

1 Introduction

The demand for seamless mobile system integration and optimum access to existing information systems has been identified in its relevance to distributed systems in research and in many application areas several years ago. Various publications have stated problems (e.g. [3], [5]) and several approaches on how to integrate mobile devices into existing system infrastructures have been proposed over time (e.g. [2], [7], [10]). While approaches differ from one another, most of them have in common, that they are based on modifications to the mobile device and/or require special system support on the application side. Especially with mobile systems most frequently being of "closed" design and not open to any modification to the device itself, such integration platforms will not suffice as a universal approach to the problem. They will rather be restricted to just a few device types open for third party modifications or extensions. In addition to that, it will not be acceptable for numerous application contexts to make use of sometimes very specialized software libraries when realizing mobile system integration. Consequently one will have to offer an integration platform that will not have such an impact on system design or the devices to be integrated.

This article introduces the "Java Border Service Architecture" (JBSA), a flexible infrastructure to support integration of arbitrary mobile devices into distributed system infrastructures. In contrast to other approches to the problem, the integration platform described here is based on the introduction of an abstract format used to analyze and describe an application's user interface in real time and its transformation into a concrete representation *on the fly*. As the platform uses runtime analysis rather than code modification, no changes to the application involved will be necessary.

2 Integration through abstraction

In accordance to a taxonomy proposed by M. Satyanarayanan ([13], see also [6]), one can classify mobile system integration strategies with respect to their level of adaptivity or "application awareness" in a spectrum ranging from "laissez faire" (integration without any system support to the application) to one extreme and "application–transparent" (full integration responsibility at the underlying system with no modification to the application) to the other. Adding another dimension to this taxonomy and measuring the level of "mobile device awareness", that is, the extent of modifications necessary to the mobile device, one will get the taxonomy as shown in fig. 1. Most existing inte-

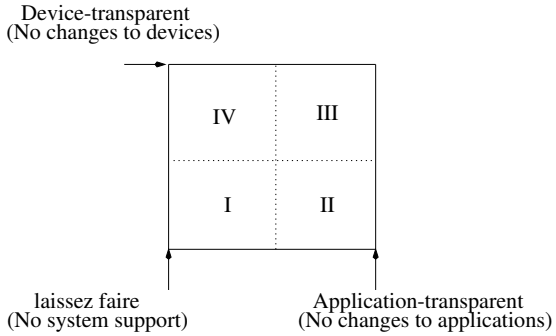

Figure 1. A taxonomy for mobile adaption and integration strategies

gration approaches have in common that they at least partly

rely on modifications to the mobile client. In some cases a special viewer has to be installed (e.g. a VNC viewer for the "Virtual Network Computing" environment [12]) to enable an interaction from the mobile device to be integrated. Other approaches are based on design time modification to the application environment and thus require changes to the application itself (e.g. UIML [1]). Accordingly, existing mobile device integration platforms can be assigned to quadrant *one* or *two* of fig. 1.

In contrast to that, one of the main aim of the integration principle underlying the JBSA was to completely avoid changes to the mobile device or the application. Consequently, the following aspects were identified as its primary design goals: *Platform indepencency:* no restriction to certain device classes but rather validity for arbitrary devices. *Device transparency:* no modifications to the mobile device but rather to network nodes and within the infrastructure. *Application transparency:* no modifications at design time and avoidance of changes to the application code. With respect to these goals, the integration principle can be assigned to quadrant *three* of fig. 1.

Applications can generally be subdivided into multiple layers, each one offering certain functionality to adjacent ones. Most of the times, a *presentation layer* will be responsible for user interaction (data presentation and visualisation, interception of user commands, etc.); an *application logic* will be responsible for data creation and processing and, where applicable, a *data storage layer* to hold all local data the application is handling (see [11]). Application logic and presentation are commonly grouped together to coordinate all access to stored or processed application data.

The principle underlying the JBSA is based on a runtime analysis of an application's presentation layer and its description in an abstract data representation, a specially designed XML dialect. By means of a set of device–dependent transformation rules expressed in XSL and processed by an XSLT processor, the abstract description can be transferred into concrete as required for a proper presentation on the target device (a technique sometimes described as "rendering" [9]). In a similar way, interaction on the device is transferred into an abstract representation, routed to the application and retranslated into events as if caused by local user interaction. All steps are performed in real time and without significant change from a user's point of view. As part of the analysis, all user interface primitives (push buttons, text fields etc.), are identified at an operating system level by means of GUI object hierarchy inspection. No *semantic* analysis is performed and thus no "valence" will be given to identified elements. Type and unique identifier of each user interface primitive is derived and registered to allow for later access to each GUI element. All results of the runtime analysis are assembled and form a "snapshot" in a well defined descriptive language.

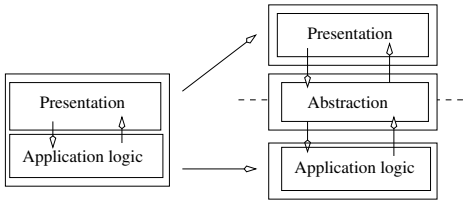

Figure 2. An abstraction layer between presentation and application logic

The snapshot together with a stylesheet corresponding to the target platform will then be routed to an XSLT processor. The processor in turn will translate all elements from the snapshot to a concrete representation as defined in the stylesheet. In addition to a basic transformation of user interface elements into a target language (e.g. WML), the stylesheet can contain rules on how to adapt (or even substitute) certain GUI elements to the target device where necessary [1]. New devices can be introduced at any time by providing a rule set for their respective device class and mapping each GUI element from the source platform to corresponding elements as indicated by their expressional power.

The results of the transformation are finally routed through a communication adapter and to the target device itself. As part of this process, necessary adaptions to the basic communication mechanism can take place (e.g. retranslation into SMS messages, or even speech synthesis) as required by the device.

In opposite direction, reactions on the target device will be intercepted by the communication adapter, translated into an internal representation and forwarded to an application adapter that will "replay" each event to the application itself. It is an important design feature that the application will *not* be able to distinguish between local user interaction and interaction "remote controlled" through the integration infrastructure.

3 The Java Border Service Architecture

An infrastructure to support mobile device integration has been designed that is based on the principle of introducing an abstraction layer between presentation and application logic as depicted in the previous section.

The JBSA realizes a supporting framework to provide remote access to running Java applications. The JBSA can continously inspect applications, generate "snapshot" information of their user interface in an abstract data representation and transform this abstract representation into a con-

[1] Substitution can be necessary with WAP devices, for example, where horizontal scrolling is not possible and tables sometimes can only be displayed when remapping columns of a row to a vertical orientation.

crete format as required by the target platform. Generated information can be presented to the device using any transport as indicated by its communication capabilities. In addition to mere inspection, analysis, transformation and transport, the JBSA framework will aid the mobile user in locating optimum applications for her needs, provide session control for multiple application access and offer user authentication and authorization for enhanced security.

The requirement to realize a complete set of core services (e.g. RPC or event queueing) on the mobile device itself, as necessary with many other integration architectures, can be avoided when using the JBSA. As the application itself is not relocated to the mobile device but rather accessed through a remote copy of its *presentation layer* (i.e. its user interface), the only requirement arising will be to provide enough functionality to view that copy — and using the JBSA´s flexible mechanism of generating an abstract representation first, the copy can be presented in a way that will best suite the viewing environment offered by the device. No functionality has to be provided for any aspects concerning *application logic*, as all parts dealing with data manipulation as well as the application itself remain within the connected network. And as no libraries, or supporting services have to run on mobile devices, the JBSA will be a preferrable solution for "closed" systems such as mobile phones, where modifications are hardly possible.

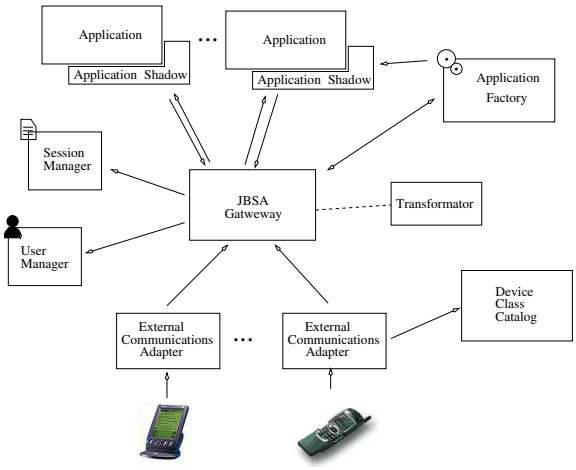

Figure 3. The JBSA and its components

A modular design was chosen for the JBSA, in which functional components are grouped in independent services. The following section will give an overview on each of the services currently realized.

Any interaction of a mobile device with the infrastructure will take place through one of the *External Communication Adapters*. ECAs are responsible for providing a communication channel as necessary to meet the capabilities of a device class and mapping internal JBSA communication to external communiction with the device (e.g. translation to a WAP transport protocol, etc.)

The *JBSA gateway* is responsible for routing events as intercepted by a certain ECA (e.g. with a button being pressed or a list item being selected) to a corresponding application. In addition to that, the gateway controls the various XSLT processors used to translate user interface "snapshots" into concrete representation formats.

Application Shadows are "chained" into Java's SWING user interface event queue mechanism of a running application and continously monitor its state. Each application shadow is responsible for exactly one application instance and can analyze its user interface to generate "snapshots" in the "Java Swing Markup Language" (JSML), an XML–dialect especially designed to describe graphical user interfaces[2], and "inject" user interaction back into the running application´s event queue.

The *Application Factory* is responsible for the management of applications registered with the JBSA infrastructure and available for remote control from mobile devices. It offers necessary functionality to add applications and their respective startup parameters in a central database and it is the factory´s duty to start applications and their corresponding application shadow when triggered by the JBSA gateway.

The *Session Manager* realizes a session control at the application layer within the JBSA infrastructure. It is responsible for temporary suspension of a running session, e.g. when changing the device used to access the application, and its subsequent reactivation after the user has reconnected to the infrastructure at a later moment.

The *User Manager* serves to personalize the JBSA infrastructure. It organizes the user accounts known to the system and is responsible for user authentication issues and will be used within the JBSA to store all user–specific information such as lately used applications, preferences, etc.

The *Device Class Catalog* (DCC) realizes a type management system for all device classes known to the JBSA infrastructure. To each device class, the DCC holds a description on its unique characteristics and the stylesheet necessary to create content to be displayed by a device of that class.

4 Workflow as an Application Area

As motivated in the introduction, workflow is an application area which fits integration of mobile devices very well.

The primary task of a workflow management system is to enact case-driven business processes. The architecture of a workflow management system according to the reference model of the WfMC ([4]) consists of interacting modules, which are the process definition, the workflow engine(s),

[2] See *http://www.jbsa.de* for details on the JSML markup language.

the administration and monitoring tools, the invoked applications, and the workflow client application. For specifying the process definition, tools are used which translate graphical models into some process definition language. The workflow engine is the system's kernel which actually executes a workflow instance. Administration and monitoring tools are used for example to influence and visualise workflow execution at runtime. Normally, the workflow engine assigns to–do tasks to some user in the system (push model) and stores the resulting list of work items per user. The workflow client application is the means for the user to access his work item list and to tell the workflow engine which tasks are in work or completed. In more elaborated approaches, the server stores an overall list of tasks for the execution of which a certain skill or role is required. Any user who fulfills the requirements can commit himself to perform the task (pull model) and then later report its completion. The workflow engine then has to take care of each work item to being assigned to at most one user.

Of these modules, the most important for our application is the workflow client application. A workflow client is a stand-alone application which communicates with the workflow engine through an interface (called Interface 2 by the WfMC). Since this is a client/server scenario, this communication is usually a remote one. Many workflow systems use a normal web browser as workflow client, so that the user interface is also generated by the server. The workflow client is the part of the system for which remote and mobile access is most important, although for administration tools this might also be useful. In this paper, we focus on the workflow client.

5 Application Example

The application area of workflow management introduced in the previous section has been chosen for a small case study to evaluate the JBSA. For a prototype system, we chose to utilise a workflow engine and client developed at our University. Technical detail about this project can be found in [8]. We briefly describe the high-level architecture of this workflow management system and, slightly more detailed, the workflow client.

The process description language that has been used is based on a special flavour of high-level Petri nets called reference nets. Besides Java objects as tokens within the nets, this formalism allows multiple instances of a Petri net, which is very convenient for many case instances of the same business process.

The workflow engine uses the approach of a pull model described in the previous section. An existing reference net execution engine (see `www.renew.de`) has been extended so that the executing nets can be observed for activated transitions. These again correspond to tasks that can be executed. The workflow client interface provides event methods for telling a workflow client that a new task is available or no longer available.

Like the workflow engine, the workflow client application has been written in Java. Accessing the workflow server works via RMI. Multiple clients can access the same workflow server at a time and changes that affect availability of tasks will be propagated to all registered workflow clients at once. The workflow server serializes incoming requests and keeps the workflow instances consistent when processing concurrent tasks.

A Swing GUI is used to display and let the user choose available tasks (Figure 4, in the lower part of the window). The list is automatically updated whenever the availability of a task changes. A second list in the upper part of the window names the tasks which the user already commited himself to. These can be selected to inform the workflow engine of their completion. In the workflow project, the

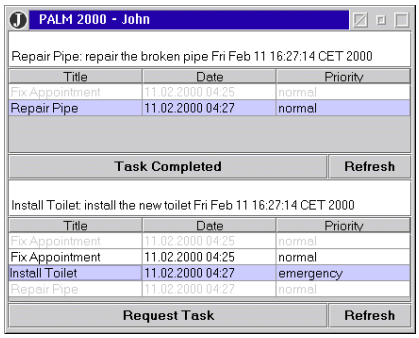

Figure 4. A simple workflow client application

scenario of a plumbing company was used and several business processes were modelled using the reference net editor. Please note that the application is limited in its functionality and thus meant as an example and not a real management software.

The specialty of the concrete application is that plumbers are in transit during the day, but jobs should be handled on request (e.g. emergency cases). Another reason for needing flexibility is that the duration of a job cannot always be estimated correctly. Thus, plumbers should be able to accept jobs and report completion of jobs using mobile devices. One can easily imagine different categories of such devices: The staff car of a plumber might be equipped with a laptop, while when at a customer's site, a mobile phone might be his only communication means. As as third case, there are employees of the company (like e.g. secretaries) who usually accept and manage jobs from their "static" working place.

The solution developed in the workflow project had the disadvantage that only Java-aware clients could be used as

workflow clients. Using the JBSA, it was possible to use a Web browser, a PDA, and even a WAP phone as the workflow client without changing the Java client application.

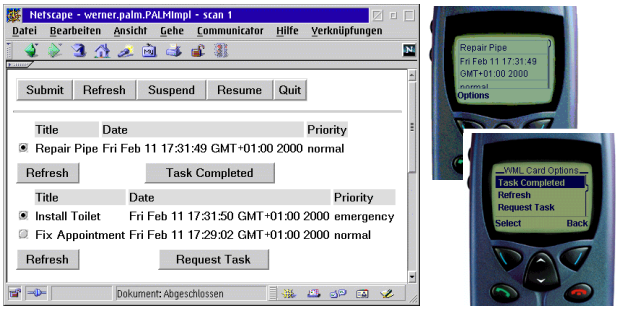

Figure 5. The workflow client shown in different target representation formats

Fig. 5 shows access to the application from a WAP phone and a terminal using a web browser. Even though actual *handling* differs sometimes significantly between two device classes (and even within a class, e.g. among the various WAP phones), all application functionality remains available for remote access. Changing from one device to another is possible anytime and without losses at runtime using the JBSA. Support for new device classes can be integrated even with applications being actively accessed.

6 Conclusions and future work

This article proposed a new approach to the field based on the introduction of an abstract layer between presentation and application logic of an application. By means of a mechanism of generating user interface "snapshots" on the fly and transforming generated descriptions through a flexible stylesheet processor, all integration related problems can be hidden away from both the application and the target device. This is of particular importance with mobile devices, where a "closed" design renders modifications on the device itself hardly possible most of the times, and resource limitations make porting of appliction clients a difficult task even where third party code *can* be installed on the device (e.g. for Java applications to be used from an average PDA). The principles described were implemented in the "Java Border Service Architecture", a modular framework integration framework for mobile devices.

The workflow management scenario has proven to be an ideal target for mobile system integration using the JBSA. Field workers can now access the central WFM engine from anywhere. With the integration environment adapting to the device in use at a time they are free to take the device with them that best suites their needs on a work day. Still, no special treatment for mobile devices was necessary within the existing workflow engine to accomplish this.

We are currently implementing various enhancements to the JBSA. Additions to the session management will allow for a user transparent reactivation of multiple suspended application sessions. Improvements to the user management will make it possible to use unique mobile device IDs for authentication purposes. And we are invesigating if JBSA support can also be realized for non-Java programming platforms such as native Win32, Motif or similar.

References

[1] M. Abrams, C. Phanouriou, A. Batongbacal, S. Williams, and J. Shuster. UIML: An appliance independant XML user interface language. In *Proc. 8th Intl. World Wide Web Conference*, Toronto, Canada, May 1999.

[2] N. Adams, R. Gold, W. Schilit, M. Tso, and R. Want. An infrared network for mobile computers. In *Proc. of USENIX Symposium on Mobile Location–Independant Computing*, pages 41–52, Cambridge, Mass., 1993.

[3] G. Forman and J. Zahorjan. The challenges of mobile computing. *IEEE Computer*, pages pp. 39–47, Apr. 1994.

[4] D. Hollingsworth. The workflow reference model. Specification TC00-1003, Issue 1.1, Workflow Management Coalition, Jan. 1995.

[5] T. Imielinsky and H. F. Korth, editors. *Mobile Computing*. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston / Dordrecht / London, 1996.

[6] J. Jing, S. Helal, and A. Elmagarmid. Client–server computing in mobile environments. *ACM Transactions On Database Systems*, Preliminary Version, 1999.

[7] A. Joshi, S. Weerawarana, T. Drashansky, and E. Houstis. SciencePad: An intelligent electronic notepad for ubiquitous scientific computing. In *Proc. IASTED'95*, Washington, USA, Dec. 1995. Purdue University, Dept. Of Computer Science.

[8] O. Kummer and D. Moldt. Workflow-Konzepte und Workflow-Ausführung für High-Level-Petrinetze. Technical report, University of Hamburg, Germany, 2000. To be published.

[9] H. Lei, M. Blount, and C. Tait. DataX: an approach to ubiquitous database access. In *Proc. 2nd IEEE Workshop on Mobile Computing Systems and Applications, WMCSA'99*, New Orleans, USA, Feb. 1999.

[10] S. McDirmid. A distributed virtual machine architecture for mobile java applications. *Handheld Systems*, 6(5):37–42, Sep./Oct. 1998.

[11] S. Murer, P. Schnorf, E. Gamma, and A. Weinand. *Eine realistische Applikationsarchitektur für Multi–Tier Java–basierte Clients in der Praxis*, chapter 9. in: Java in der Praxis. d.Punkt Verlag, Heidelberg, Germany, 1998.

[12] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, Jan/Feb 1998.

[13] M. Satyanarayanan. Mobile information access. *IEEE Personal Communications*, 3(1), Feb. 1996.