JBSA: An Infrastructure for Seamless Mobile Systems Integration

Stefan Müller-Wilken and Winfried Lamersdorf

University of Hamburg, Vogt-Kölln-Str. 30, 22527 Hamburg, Germany,
Phone: +49-40-42883-2327 Fax: +49-40-42883-2328
{smueller,lamersd}@informatik.uni-hamburg.de,
http://vsys-www.informatik.uni-hamburg.de

Abstract. While the desire to gain full access to stationary information sources (e.g. the company´s backoffice database) when in transit is only natural, inherent design limitations such as a lack of computational power, very limited resources and closed architectures, let mobile system integration still be a difficult issue. In contrast to other approaches to the field, the "Java Border Service Architecture" introduced in this article doesn't require modifications to the mobile device´s system software or the application environment to give access existing applications from the mobile system. It features a new way of generating user interface snapshots in an XML–based description format "on the fly", translating them to a light–weight format suitable for the device and sending UI snapshots to the device in real time. This article gives an overview of the architecture and its functionality and presents an application scenario that has been implemented on top of it.

1 Introduction

The demand for seamless mobile system integration and optimal access to existing information systems has been identified in its relevance to distributed systems research several years ago. Various publications have stated arising problems (e.g. [8],[9],[17]) and several approaches on how to integrate mobile devices into existing system infrastructures have been proposed over the time (e.g. [2], [11], [13], [16]). 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 due to the fact that mobile systems are very often of "closed" design and resist any modification to the device itself, such integration platforms will not suffice as an 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, it will not be acceptable for numerous application contexts to make use of specialized software libraries when realizing mobile system integration. Consequently, one does have to offer an integration platform that will not have such an impact on system design on the one hand and mobile devices to be integrated on the other.

This article introduces the “Java Border Service Architecture”, a flexible infrastructure to support integration of arbitrary mobile devices into distributed system environments. In contrast to other approaches 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 allows for 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. Using this technique, it is possible to support the ever increasing number of device types by just changing the set of transformation rules to apply to the abstract representation for a certain class. As the application code remains unchanged, it is also possible to integrate applications where a modification is impossible due to design aspects or because the source code is not available for modification.

The article is organized as follows: section 2 describes the principles underlying the JBSA. Section 3 gives an overview on the infrastructure itself and its core components. An example is introduced in section 4 and finally, section 5 concludes with a summary and outlook on future work.

2 Integration through Abstraction

Various approaches to the problem of finding better means to seamlessly integrate mobile systems into distributed system environments were proposed in recent years. In accordance to a taxonomy proposed by M. Satyanarayanan ([18], see also [10]), 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 shown in fig. 1. Most existing integration 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 [15], etc.) 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], XFDL [4], etc.). Accordingly, existing mobile device integration platforms can be assigned to quadrant I or II of the taxonomy depicted in fig. 1.

In contrast to that, one of the main aims of the integration principle underlying the JBSA is to completely avoid changes to both the mobile device and the application. Consequently, the following aspects were identified as its primary design goals. *Platform indepencency:* the integration platform is not to be restricted to certain device classes but should valid for arbitrary devices. *Device transparency:* modifications are not to be made to the mobile device but rather

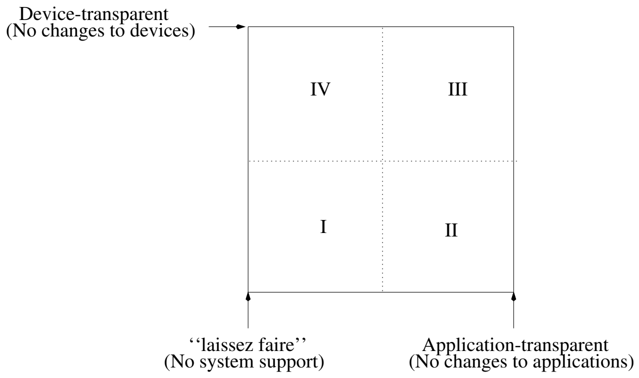


Fig. 1. A taxonomy for mobile adaption and integration strategies

to network nodes and within the infrastructure. *Application transparency:* modifications are not to be made at design time and changes to the application code are to be avoided. With respect to these goals, the integration principle can be assigned to quadrant III of fig. 1.

Separating User Interfaces

Applications can generally be subdivided into multiple layers, each one offering a certain functionality to adjacent ones. Most of the time, 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* is used to hold all local data the application is handling (see [14]). Application logic and presentation are commonly grouped together to coordinate all access to stored or processed application data.

The solution chosen by 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 [5] dialect. By means of a set of device–dependent transformation rules expressed in XSL [3] and processed by an XSLT processor [6], the abstract description can be transferred into a concrete format as required for a proper presentation on the target device (a technique sometimes described as "rendering" [12]). 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 such as text fields, lists, 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 added to identified elements. Type and a unique identifier of each user interface primitive is derived and registered to allow for later access to each GUI element. All results

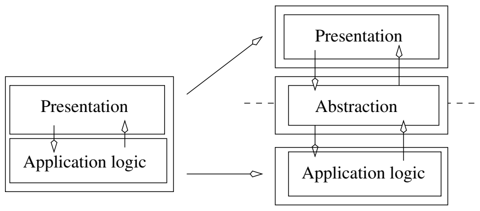


Fig. 2. An abstraction layer between presentation and application logic

of the runtime analysis are assembled and form a "snapshot" in a well defined descriptive language.

The snapshot together with a stylesheet corresponding to the target platform will then be routed to a 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 expressive 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 the 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

The *Java Border Service Architecture* (JBSA) realizes a supporting framework to provide remote access to running Java applications. The JBSA can continuously inspect applications, generate "snapshot" information of their user interface in an abstract data representation and transform this abstract representation into a concrete format as required by the target platform. Generated information can be presented to the device using any transport mechanism 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

[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.

optimal 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 suit the viewing environment offered by the device. No functionality has to be provided for any aspects concerning *application logic*, since 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 preferable solution for “closed” systems such as mobile phones, where modifications are hardly possible.

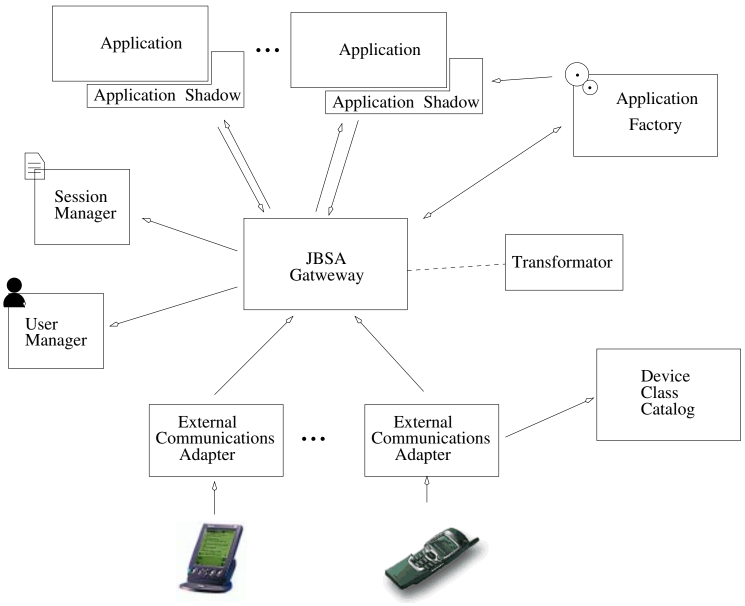


Fig. 3. The *Java Border Service Architecture* and its components

3.1 The JBSA and Its Components

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

External Communications Adapter. Any interaction of a mobile device with the infrastructure will take place through one of the *External Communications 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 communication with the device. Possible examples: translation to a WAP transport protocol, translation to SMS messages, etc. ECAs will additionally provide basic session control capability to the mobile device. They will decide which ECA instance to route certain events to, when to declare a particular communication channel as interrupted, and the like.

JBSA Gateway. 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. Additionally, the gateway controls the various XSLT processors used to translate user interface "snapshots" into concrete representation formats. The gateway will finally realize a higher session semantics. Supported by a session manager (see below), it will assign running applications to certain users on the one side and certain ECA instances (and hereby certain devices) on the other.

Application Shadow. *Application Shadows* are "chained" into the Java/SWING user interface event queue mechanism of a running application and continuously 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 specially designed to describe graphical user interfaces. Each application shadow instance offers a network interface used by the JBSA gateway to trigger snapshot creation, to fetch generated JSML documents and to "inject" user interaction back into the running application´s event queue.

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

Session Manager. The *Session Manager* realizes a session control at the application layer within the JBSA infrastructure. While *External Communications Adapters* form a *device session* that assigns a communication channel to a certain device, the *Session Manager* realizes an *application session* to assign an application context to a certain user. The *Session Manager* 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.

User Manager. The *User Manager* serves to personalize the JBSA infrastructure. The User Manager organizes the user accounts known to the system and is responsible for user authentication issues. In conjunction with the Session Manager described above, the User Manager serves to provide unique reference keys for running application sessions as required to identify applications started by a certain user and for a certain communication channel. In addition to that, the User Manager will be used within the JBSA to store all user–specific information such as lately used applications, preferences, etc.

Device Class Catalog. The *Device Class Catalog* (DCC) realizes a type management system for all device classes known to the JBSA infrastructure. For each device class, the DCC holds a description on its unique characteristics and the stylesheet necessary to create contents to be displayed by a device of that class. Relations between arbitrary device classes can be stored in nets and will be utilized whenever a stylesheet for a certain device class could not be found but similarities to a known class were noticed. The DCC is used by both ECAs and XSLT processors to find optimum stylesheets for devices they are generating output for.

3.2 Functionality of the JBSA

This section gives a detailed description of a typical access cycle between application and a mobile device integrated through the JBSA. Going through the various steps that take place as part of the cycle, participating components of the infrastructure are identified in their responsibilities and illustrated in their interaction with other components.

Making Contact — Step 1

– The device accesses an external communication adapter through a suitable access point (e.g. an HTTP–address or a dialup phone line). The ECA sends a request to the Device Class Catalog to determine the class the target device belongs, initializes a channel to the JBSA gateway and stores both in a freshly generated device session.
– In a second substep, the ECA sends a request to the JBSA gateway to create an application session for the device session generated in the previous step.

Initialization of an Application Session — Step 2

– The JBSA gateway creates a login dialog, translates it to a format indicated by the device type reported from the external communcation adapter, and sends the dialog to the ECA. The ECA forwards the request to the device to query the necessary authentication information and retransmits its result back to the gateway for further processing.

- The gateway sends a call to the user manager to check the validity of the user identity. As part of a positive response the user manager will send all information relevant to the starting phase (e.g. the user´s default application, personal default parameters for the device class, etc.) back to the JBSA gateway.
- The gateway makes a call to the session manager to check for the availability of application sessions suspended on behalf of the user. The session manager will look through its local database and report all application sessions carrying th users identifier back to the JBSA gateway.
- If suspended sessions belonging to the user could be identified, the JBSA gateway will generate an appropriate selection dialog, asking the user if to resume, cancel a suspended session or create a new session. The dialog data will be sent to the ECA and on to the target device and the user choice back to the gateway.
- If no session belonging to the user could be identified or if the user chose not to pick up an existing one, the identifier for the newly created application session in combination with the device session identifier reported from the ECA in step one will be registered at the session manager.

Starting the Application — Step 3. Now, with a session between the target device and the Border Service Architecture instantiated, the application has to be started.

- The JBSA gateway sends the factory–identifier of the default application assigned to the user profile, the device class or the application scenario together with a unique startup identifier to the Application Factory.
- The Application Factory scans its database for the factory–identifier coming from the gateway to find the application path and startup parameters registered with it. Both, together with a unique startup identifier, are forwared to a new Application Shadow which the factory has initialized for the application session.
- The Application Shadow starts the selected application within its runtime environment, "hooks" up with the application´s event queue and initializes a communication channel to allow for further access through the JBSA gateway. Information about this channel together with the startup identifier is sent back to the JBSA gateway.
- The JBSA gateway uses the startup identifier reported back by the freshly created Application Shadow togeher with the Shadows´s communication channel to assign the latter to the application session as generated in step two.

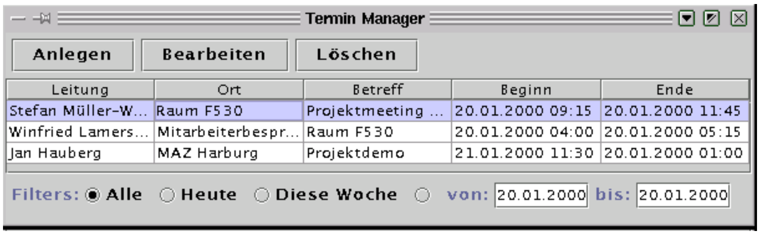
Using the Application — Step 4. As the application is running and the Application Shadow is ready to generate user interface snapshots in JSML, the actual interaction between target device and application instance sets in.

- After registering the application as part of the application session with the Session Manager, a request for a first user interface status is sent from the JBSA gateway to the Application Shadow. To accomplish this, a connection between the gateway and the Shadow registered with the application session is opened and a JSML snapshot is fetched from it.
- The JSML snapshot is sent to the XSLT processor initialized with the stylesheet identified by the Device Class Catalog for the target device. The processor translates it into an appropriate representation and hands it back to the gateway.
- The user interface snapshot, now in a representation suitable for a presentation on the target device is sent forward to the External Communication Adapter responsible for the target device.
- The user interface snapshot is presented to the user through some browser installed on the device. The user can begin her interaction with the application, e.g click a button of edit a text field.
- The browser reports any user interaction back to the External Communication Adapter. Considering the channel the interaction was reported through (e.g. the phone line used for the interaction), the ECA assigns the incoming interaction to a certain device session. The ECA transforms the reported interaction event (e.g. pressing a button) into a form useable within the JBSA infrastructure, combines it with the device session identifier and forwards both to the JBSA gateway.
- Using the device session identifier, the JBSA gateway identifies the corresponding application session. It extracts the Application Shadow assigned to the session from the session record and forwards the event on to the Application Shadow.
- The Application Shadow generates valid Java SWING events from the request coming from the JBSA gateway and "injects" them into the application´s event queue for further processing. The application can not tell the difference of an event generated by the Shadow to an event caused by local interaction.
- The application reacts to the event as it would have done to a local user interaction and changes its user interface according to changed status within its application logic.
- After a short break (but still not noticeable to the user), the whole process is reinitiated by the JBSA gateway and starts with another update request to the Application Shadow.

4 Application Example

An office communication context was chosen as one application scenario for the JBSA[2]. In this scenario, field workers need access to a central group calendar held at a company´s back office. Modifications should be possible from *any* location using a desktop PC as well as a PDAs or a mobile phone. The application

[2] The application is limited in its functionality and thus meant as an example and not a real time management software.

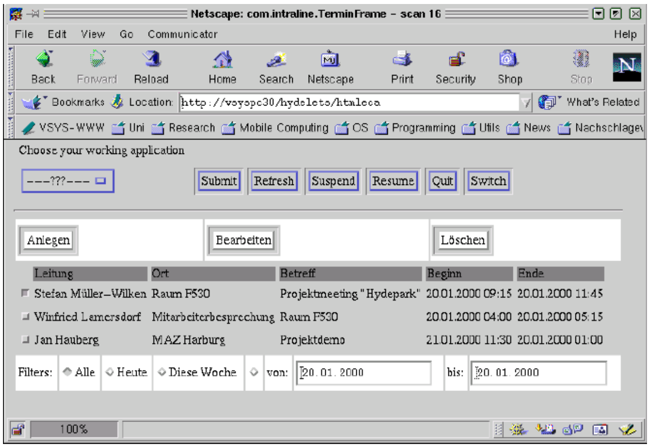


Leitung	Ort	Betreff	Beginn	Ende
Stefan Müller-W...	Raum F530	Projektmeeting ...	20.01.2000 09:15	20.01.2000 11:45
Winfried Lamers...	Mitarbeiterbespr...	Raum F530	20.01.2000 04:00	20.01.2000 05:15
Jan Hauberg	MAZ Harburg	Projektdemo	21.01.2000 11:30	20.01.2000 01:00

Filters: [✓ Alle] Heute Diese Woche von: 20.01.2000 bis: 20.01.2000

Fig. 4. A simple datebook application

offers basic functionality such as adding, modifying or removing dates (see fig. 4). It is implemented in a client/server design with a front end Java/SWING user interface accessing a backend database via RMI. Multiple users interfaces can access a single backend at a time and changes to the database will be propagated to all registered user interfaces at once. The application backend will serialize incoming requests and keep the database consistent when processing concurrent tasks. Stylesheets were designed for various target device classes such



Location: http://vsyspc30/hydeleto/htmleca

Choose your working application

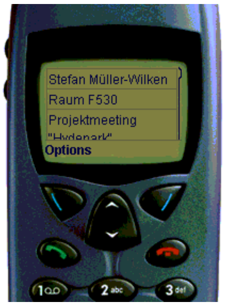
---???---

Submit Refresh Suspend Resume Quit Switch

Anlegen Bearbeiten Löschen

Leitung	Ort	Betreff	Beginn	Ende
Stefan Müller–Wilken	Raum F530	Projektmeeting "Hydepark"	20.01.2000 09:15	20.01.2000 11:45
Winfried Lamersdorf	Mitarbeiterbesprechung	Raum F530	20.01.2000 04:00	20.01.2000 05:15
Jan Hauberg	MAZ Harburg	Projektdemo	21.01.2000 11:30	20.01.2000 01:00

Filters: [✓ Alle] Heute Diese Woche von: 20. 01. 2000 bis: 20. 01. 2000



Stefan Müller-Wilken
Raum F530
Projektmeeting
"Hydepark"

Fig. 5. The datebook shown in different target representation formats

as 3Com PDAs, WAP phones (espec. Nokia 7110, Ericsson R380) and HTML browsers (Netscape Communicator, Internet Explorer). Fig. 5 shows access to the application from a WAP phone and a terminal using Netscape Communicatior. Even though actual *handling* differs sometimes significantly between two device classes (and even among 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.

5 Conclusions and Future Work

Integrating mobile devices into existing system infrastructures represents one important issue in distributed systems research today. Various approaches exist but most of them show the drawback of being based on modifications to the mobile device, the application to be accessed while on transit, or both.

This article proposes a new approach to the field based on the introduction of an abstract layer between presentation layer 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 for mobile devices, where a "closed" design renders modifications on the device itself hardly possible most of the times, and resource limitations make porting of application 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 integration framework for mobile devices. This article gave an overview on the architecture, its main components and their functionality. The actual operation was described and, finally, a short application example was given.

Early experiences with the prototype implementation were very promising, and a proof–of–concept could be derived from our tests. Access to Java client/server applications could be realized for a number of target device classes including first WAP phones on the market today. The development cycle to integrate new devices could be drastically reduced as changes are only necessary to a stylesheet while device and application can remain untouched.

We are currently implementing various enhancements to the "Java Border Service Architecture". 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 identifiers (such as GSM SIMs) for authentication purposes. And we are invesigating if JBSA support can also be realized for non-Java programming platforms such as native Win32 or Unix (Motif, Qt or similar) applications.

The JBSA is being developed as part of the Hydepark project, in which the University of Hamburg's Distributed Systems Group is developing and evaluating new methods on how to integrate mobile devices into distributed system infrastructures. Another subproject within the Hydepark context is currently working on ways to integrate non–Java devices into SUN´s "Java Intelligent Network Infrastructure" (JINI) [7] using the JBSA. JINI itself features an approach comparable to JBSA with so called "service proxies" acting as loosely coupled front end interfaces to corresponding service instances. The JINI framework is thoroughly done in Java and consequently not accessible from non–Java devices at this time. A combination with the JBSA will eliminate this drawback and give access to JINI–enabled services from any location and with arbitrary devices, making JINI even more appealing as a platform for distributed systems design.

References

1. Abrams, M., Phanouriou, C., Batongbacal, A., Williams, S., and Shuster, J. UIML: An Appliance Independant XML User Interface Language. In *Proc. 8th International World Wide Web Conference* (Toronto, Canada, May 1999).
2. Adams, N., Gold, R., Schilit, W., Tso, M., and Want, R. An infrared network for mobile computers. In *Proc. of USENIX Symposium on Mobile Location–Independant Computing* (Cambridge, Mass., 1993), pp. 41–52.
3. Adler, S., Berglund, A., Caruso, J., Deach, S., Grosso, P., Gutentag, E., Milowski, A., Parnell, S., Richman, J., and Zilles, S. Extensible Stylesheet Language (XSL) 1.0. Tech. Rep. WD-xsl-20000327, W3C, March 2000.
4. Boyer, J., Bray, T., and Gordon, M. Extensible Forms Description Language (XFDL) 4.0. Draft Specification NOTE-XFDL-19980902, W3C, September 1998.
5. Bray, T., Paoli, J., and Sperberg-McQueen, C. Extensible Markup Language (XML) 1.0. W3C Recommendation REC-xml-19980210, W3C, Feb 1998.
6. Clark, J. XSL Transformations (XSLT) 1.0. W3C Recommendation REC-xslt-19991116, W3C, November 1999.
7. Edwards, W. K. *Core JINI.* The SUN Microsystems Press Java Series. Prentice Hall PTR, Upper Saddle River, NJ, 1999.
8. Forman, G., and Zahorjan, J. The challenges of mobile computing. *IEEE Computer* (April 1994), pp. 39–47.
9. Imielinsky, T., and Korth, H. F., Eds. *Mobile Computing.* Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, 1996.
10. Jing, J., Helal, S., and Elmagarmid, A. Client–server computing in mobile environments. *ACM Transactions On Database Systems* (Prelim.Version, 1999).
11. Joshi, A., Weerawarana, S., Drashansky, T., and Houstis, E. Sciencepad: An intelligent electronic notepad for ubiquitous scientific computing. In *Proc.International Conference on Intelligent Information Management Systems (IASTED'95)* (Washington, USA, December 1995), Purdue University, Dept. Of Computer Science.
12. Lei, H., Blount, M., and Tait, C. 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).
13. McDirmid, S. A distributed virtual machine architecture for mobile java applications. *Handheld Systems 6*, 5 (Sep./Oct. 1998), 37–42.
14. Murer, S., Schnorf, P., Gamma, E., and Weinand, A. *Eine realistische Applikationsarchitektur f r Multi–Tier Java–basierte Clients in der Praxis.* in: Java in der Praxis. d.Punkt Verlag, Heidelberg, Germany, 1998, ch. 9.
15. Richardson, T., Stafford-Fraser, Q., Wood, K. R., and Hopper, A. Virtual network computing. *IEEE Internet Computin 2*, 1 (Jan/Feb 1998), 33–38.
16. Roman, M., Singhai, A., Carvalho, D., Hess, C., and Campbell, R. H. Integrating PDAs into distributed systems: 2k and PalmORB. In *Handheld and Ubiquitous Computing – First International Symposium on Handheld and Ubiquitous Computing (HUC 99)* (Heidelberg, Germany, September 1999), H.-W. Gellersen, Ed., no. 1707 in Lecture Notes in Computer Science, Springer–Verlag.
17. Satyanarayanan, M. Fundamental challenges in mobile computing. In *Proc. 14th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)* (Ottawa, Canada, August 1995), ACM.
18. Satyanarayanan, M. Mobile information access. *IEEE Personal Communications 3*, 1 (February 1996).