A Service Architecture for Mobile Teamwork*

Engin Kirda, Pascal Fenkam, Gerald Reif and Harald Gall
Technical University of Vienna
Distributed Systems Group
Argentinierstrasse 8/184-1, 1040 Vienna / Austria
{E.Kirda,P.Fenkam,G.Reif,H.Gall}@infosys.tuwien.ac.at

ABSTRACT

Mobile teamwork has become an emerging requirement in the daily business of large enterprises. Employees collaborate across locations and need support while they are on the move. Business documents (artifacts) and expertise need to be shared independent of the actual location or connectivity (e.g., access through a mobile phone, laptop, Personal Digital Assistant, etc.) of employees. Although many collaboration tools and systems exist, most do not deal with new requirements such as locating artifacts and experts through distributed searches, advanced information subscription and notification, and mobile information sharing and access. The MOTION service architecture that we have developed supports mobile teamwork by taking into account the different connectivity modes of users, provides access support for various devices such as laptop computers and mobile phones, and uses XML meta-data and the XML Query Language (XQL) for distributed searches and subscriptions. In this paper, we describe the architecture and the components of our generic MOTION service platform for building collaborative applications. The MOTION Teamwork Services Components are currently being evaluated in two industry case-studies.

Keywords

Mobile teamwork, collaborative systems, distributed searches, XML meta-data and XQL, architectures, components

1. INTRODUCTION

Most companies and enterprises use some form of collaborative system or tool in their organizations. People need to work together by sharing information and communicating. Hence, support for information storage, communication, sharing, and retrieval is provided by popular collaborative systems such as Lotus Notes and Microsoft Outlook.

Mobile teamwork[19] (i.e., nomadic working) is an emerging requirement for large enterprises. These global organizations have employees that are situated in many countries and need to collaborate across locations. Because of the distributed nature of these organizations, many employees are often on the move and they need to share expertise and business documents independent of their physical location and actual connectivity. When a consultant is at an airport, for example, she may only have access to a Web browser. Her WAP-enabled mobile phone will work in Europe, but chances are that it will not work in the US.

Thus, anytime, anywhere access to information has become interesting to many organizations. New working methods and technologies are needed for dealing with emerging requirements such as locating documents and experts through distributed searches, advanced information subscription and notification, and mobile information sharing and access. We are addressing such problems in the MObile Teamwork Infrastructure for Organizations Networking (MOTION) project that aims to create a flexible, open and scalable Information and Communication Technologies (ICT) architecture for mobile collaboration. The MOTION project has started in February 2000 and is near completion. The MOTION prototype has been implemented and the MOTION Teamwork Services Components are currently being evaluated in two industry case-studies.

Although many collaboration tools and systems have been built to date and much has been written on the topic, most do not provide support for mobile teamwork. In this paper, we describe the architecture and components of our generic MOTION service platform for building collaborative applications.

The paper is structured as follows: Section 2 gives an overview of the layered architecture. Section 3 discusses the main components of the architecture. Section 4 presents related work and Section 5 concludes the paper.

Terminology

We first define some basic terms that will be used throughout the paper.

- **Artifact**: Any document or file in the MOTION system (e.g., a text-processing document, a picture, a sound file, etc.)
- **Peer**: Any computing device connected to the MOTION system (e.g., notebook, Web browser, Personal Digital Assistant (PDA), etc.)
- **Community**: A collection of users in the MOTION system that are interested in a topic or that have a common property (e.g., “researchers”, “paper writing”,“review”,etc.)

*This project is supported by the European Commission in the Framework of the IST Programme, Key Action II on New Methods of Work and eCommerce. Project number: IST-1999-11400 MOTION (MObile Teamwork Infrastructure for Organizations Networking)

SEKE ’02, July 15-19, Ischia, Italy.

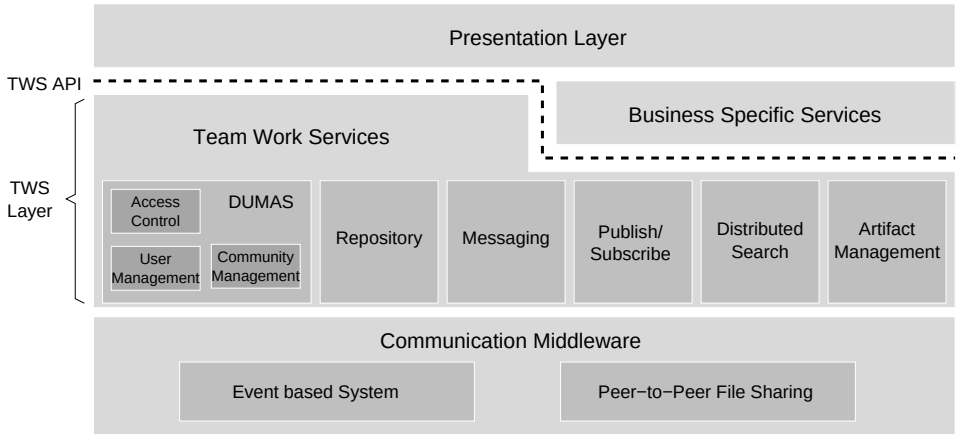

Figure 1: Overview of the MOTION Architecture

2. MOTION SERVICE ARCHITECTURE

In this section, we give a brief overview of the layered architecture of the MOTION system and provide details about the main components in the following sections. Figure 1 depicts the MOTION architecture.

The MOTION system is composed of peers. Some host services and some only act as clients. Any peer that is able to run the MOTION libraries can act as a service host. A typical MOTION configuration consists of desktop computers, laptops (i.e., notebooks and sub-notebooks) and PDAs that host services and clients such as Web browsers and WAP-enabled mobile phones that do not host services, but can only be used to remotely access them.

The lowest layer of the architecture is the communication middleware. It offers basic communication services such as peer-to-peer file sharing through distributed searches and publish/subscribe (i.e., event-based system) mechanisms to the layers above. In the prototype implementation of the MOTION platform, this functionality is provided by PeerWare[17]. The communication layer, however, can be replaced by any other suitable middleware that provides distributed search and publish/subscribe support (e.g., distributed searches with JTella[13] and publish/subscribe with JEDI[3]). We chose to use PeerWare in the prototype implementation because we had access to its source code and could experiment with it. Furthermore, PeerWare has support for *both* distributed searches and publish/subscribe, and thus covers all the requirements of the lowest layer in the architecture.

The Teamwork Services (TWS) layer is situated directly above the communication middleware. This layer integrates the basic system components such as the repository and DUMAS (see next section) and provides an Application Programming Interface (API) to the teamwork services. This is a Java API in our prototype.

The TWS API offers services such as (1) storing artifacts and their meta-data (*profile*) in the local repository, (2) managing *resources* (artifacts, users, and communities), (3) sharing artifacts with other users in communities, (4) subscription to specific events in the MOTION system, (5) sending and receiving messages from other users or from the system, (6) managing access rights on resources, (7) and searching for resources based on their profile information.

An application programmer can build business specific services (BSS) on top of the TWS API. By using the functionality provided by the API, the programmer can implement new functionality according to the end-users' business requirements. Hence, the basic set of services provided by the TWS API can be customized and extended by businesses and organizations. For example, a company might be interested in integrating workflow support for transistor design into the platform whereas another might be interested in having document versioning support for artifacts.

The top layer of the architecture is the presentation layer. It provides a user interface to the services provided by the MOTION system. The presentation layer is built using the TWS API. Because of the need for mobility, a typical configuration has a number of user interfaces for different devices such as desktop computers, laptops, Personal Digital Assistants (PDAs), Web Browsers and WAP. In the current prototype, we have a native Java user interface that provides full functionality and an experimental lightweight Java PDA interface.

3. TEAMWORK SERVICES COMPONENTS

In this section, we describe the components of the MOTION Teamwork services layer.

3.1 The Dynamic User Management and Access Control Component (DUMAS)

Confidentiality, security and privacy are important in many distributed multi-user applications. This has motivated the design and implementation of a number of access control models (e.g., [7, 20]). In most cases, the access control model is chosen by the software/security engineer and is hard-coded into the application. Hence, users of these applications have little or no support at all for customizing and adapting the security settings to requirements that may change over time.

DUMAS[5, 6] is an access control component that is formally specified, verified, and implemented. Its goal was the creation of a generic, customizable component that satisfies different security requirements. This access control component provides support for managing users and roles (e.g., by creating, deleting, etc.) and assigning users to roles. Furthermore, generic permissions can be created, assigned to users and bound to specific operations (e.g., a user *X* has a permission *Invoke* on operation *SendMessage()*). The functionalities of DUMAS are grouped in three sub-components: a *user management* component, a *community management* component and an *authorization* component. These sub-components are strongly connected in the sense that each of them is necessary for the two other sub-components to operate.

Considering the number of external requests, DUMAS is one of

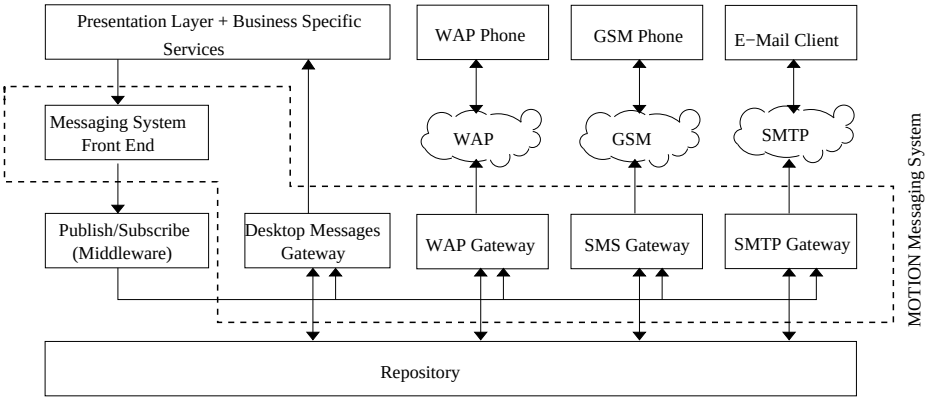

Figure 2: The MOTION Messaging Architecture

the most demanded components of the Teamwork Services Layer. For performing any security sensitive operation (e.g. creating a user, downloading an artifact, etc.) DUMAS must be consulted to find out whether the user is permitted to invoke this operation. In this sense, DUMAS is a service provider. DUMAS, however, is also a service consumer as it needs to store data and publish events. The publish/subscribe support of the underlying middleware is used for distributing events across peers.

DUMAS follows an architecture driven by (1) the the Peer-to-Peer model of the underlying file sharing system and (2) the requirement for mobility support.

In the first case, access control data (ACD) are divided in two types. First, ACD on artifacts are stored with these artifacts and are therefore distributed across peers. Second, ACD on other entities such as users and communities are stored on distributed servers. Each DUMAS component can be configured to behave as such a server.

As far as mobility is concerned, DUMAS can be customized to run on a variety of mobile devices (e.g., Java-enabled PDAs such as the Nokia Communicator and the Compaq iPAQ). μDUMAS is the lightweight implementation of DUMAS for such devices. This version is based on the remote invocation of implementations available on more powerful peers (i.e. desktop computers and laptops). The communication between a μDUMAS and a DUMAS instance is performed by publishing an XML event using the underlying middleware. The μDUMAS component does not need to know the actual instance or the location of the server it is communicating with. The XML event is simply published and the DUMAS component configured to address such requests executes the specified operation and publishes the result back. This technique allows us to explicitly deal with device and user mobility in the sense that the instance of DUMAS to which a μDUMAS instance connects can be anyone. For example, a user can move from Montreal to Vienna without the need for re-configuring her system. Further, if the peer on which the DUMAS instance is running is not available for some technical reason, or there is a connection problem as it is often the case in mobile computing, the publish/subscribe system will queue the XML event and deliver it whenever possible.

3.2 MOTION Messaging Component

MOTION Messaging is an integrated messaging service that enables users to communicate and exchange information. Notifications based on subscriptions are also delivered by this messaging service. MOTION messages are sent to users using technologies such as lightweight push[1], email (i.e., SMTP), GSM short messages (SMS), and wireless application protocol service indication (WAP SI)[8].

MOTION Messaging enables employees to stay in direct and constant contact no matter what devices they are using and where they are.

From the point of view of the originator and the recipient, there are *System-to-User*, *System-to-Community*, *User-to-User*, and *User-to-Community* messages. On the other hand, messages can also be categorized based on the delivery mechanism; we distinguish between SMTP, SMS, WAP SI, and desktop messages.

System-to- User and System-to- Community messages are mainly sent by the MOTION system as notifications of subscriptions (see Section 3.3). Whenever a document on transistor design, for example, is available in the system, a community of users interested in transistor design will receive a notification.

User-to-User messages are messages sent by users to other users. In order to set up a meeting, for example, two users may communicate using this type of messages.

User-to-Community messages are sent by one user to a community (e.g., community of users interested in transistor design). In this sense, User-to-Community messages are similar to mailing lists. In MOTION, however, User-to-Community messages provide some added value such as the ease of address management and the support for mobility. In common mailing lists, managing addresses is not always an easy task. For instance, an address might expire, but may still be referred to in the mailing list. This means that the user has to inform the mailing list administrators that the address is no longer available, or the administrator has to regularly check to see if the addresses in the list are valid. Such problems do not exist in the MOTION system. Users are referenced with their login names (i.e., user names). This identifier is used in the MOTION system as the address of the user (or community). Each user can also specify the medium under which she is available. For example, a user Hakkinen might specify that he wishes to receive all messages where the sender's name contains the string "*speed*" via SMS, whereas messages that contain the subject string "*Tyres*" should be sent to the address "*hakkinen@infosys.tuwien.ac.at*".

In order to deal with memory and message size limitations, messages that are addressed to mobile phones are automatically split into a suitable number of SMSs or WAP SIs (e.g., SMS messages can only be about 160 characters in size). In our prototype, this is

[1] Lightweight push differs from normal push systems in that only the locations (URLs) of artifacts are sent to users, and not the entire artifact contents.

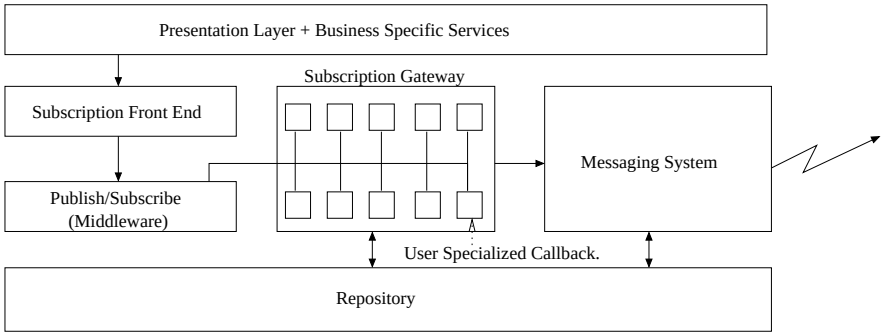

Figure 3: The TWS Layer Publish/Subscribe Architecture

done by the gateway that provides access to SMS or WAP SI[2].

SMS and WAP SI provide the ability to send notifications to users in an asynchronous manner. The advantage of WAP SI is the capability to include a Universal Resource Identifier (URI) indicating a service that the user can start by following the link. Messages, of course, can also be delivered to users using SMTP e-mail and depending on the settings of the particular user, MOTION messages can be made to directly pop up in windows on users' desktops (i.e., this is similar to messaging services provided by popular applications such as ICQ and Yahoo Messenger).

The MOTION Messaging component in our prototype consists of five main components. These components are the SMS gateway, the SMTP (email) gateway, the standard messages gateway, the WAP gateway, and the MOTION front end component (see Figure 2). The MOTION front end component is the interface between the business specific services and the MOTION Messaging component. It provides transparency to the business specific services by simple primitives for sending messages. These messages are transformed into XML events that are published through the underlying publish/subscribe component. Once a message is sent to a specific user, the configured gateway receives the corresponding XML event, transforms it and forwards it using the appropriate protocol (e.g., WAP gateways transform XML events to WAP SIs and SMS gateways to SMS messages). In case a MOTION gateway is unable to send a message for some reason, it can queue it in the repository that is available on the peer (host) the gateway is running on.

3.3 The Teamwork Services Layer Publish/ Subscribe Component

Some have identified Publish/Subscribe as an architectural style that enables high decoupling among components and fosters mobility (e.g., [2, 11]). The teamwork services layer's publish/subscribe component bridges the gap between the underlying middleware and the business-specific services and gives a uniform and consistent view of the *event* concept to the application layer.

Publish/subscribe systems such as Peerware[17] and JEDI[3] allow components to subscribe and react to events by specifying a method that is invoked once an event occurs that matches a query. There are different realizations of this concept. In PeerWare, for example, the subscriber specifies a callback. The callback is an object of a class implementing the interface `peerware.EventCallback`. In JEDI, on the other hand, the subscriber directly specifies the name of the method to be invoked. In both cases, however, there is essentially no direct mapping between component-level (system) subscriptions and user-level (application) subscriptions.

[2]We use a commercial SMS gateway that provides an SMTP interface.

To bridge this gap, we use *subscription gateways* and *user specialized callbacks* (see Figure 3). A user specialized callback is a component that handles subscriptions of a specific user. Whenever the user wishes to perform a subscription, she informs her specialized callback. This callback mediates between the underlying publish/subscribe system and the user. It receives the user's subscriptions and subscribes on her behalf. Once an event occurs that satisfies one of the user's subscription criteria, the corresponding callback is informed and it transforms the received event into a message. This message is sent to the messaging system and the user is informed based on her availability criteria.

The set of callback components running on a particular peer is referred to as the *subscription gateway*. A subscription gateway has to be configured for each user. Choosing a peer that can function as a subscription gateway is a configuration issue. Every peer in the MOTION system can be used as a subscription gateway and the configuration can be decided by organizations depending on deployment policies.

The expressiveness of the subscription language also constitutes a reason for constructing a bridge between the business specific services and the middleware. Different middleware implementations provide different languages for this purpose. To hide all these differences, we use the XML Query Language (XQL). This gives the business specific services the capability to query complex XML events. The TWS Layer's publish/subscribe system provides the business specific services with the capability of subscribing to users, artifacts, and communities in the system.

3.4 The MOTION Repository

Every peer in the MOTION system that directly runs MOTION services contains a repository that is used to store artifacts and profile information about users, communities and artifacts. This repository component is composed of two parts: an *XML* and an *artifact* repository. The XML repository is used to store XML profile information. Some of this information is used by the system and some of it (such as artifact profiles) is entered by the user. The artifact repository is used to store artifacts that belong to a user. For example, when a user *Dr. Jaza* wishes to enter a paper he is writing for the SEKE conference into the MOTION system, he would first enter meta-data about it such as the description of the document and its purpose. The meta-data would then be inserted into the XML repository and the document would be physically copied into the artifact repository. The repository component provides method calls for inserting, deleting, editing and querying meta-data that it manages.

In the prototype we have built, artifacts are stored in the local file system of the peer and a part of the file system is designated as

the MOTION artifact repository. Artifacts are queried, inserted and deleted using API calls that provide an abstraction to the underlying storage system. Any repository that implements the repository API interfaces can be used (e.g., OO Database, RDBMS, etc.).

Inserting, deleting and editing artifacts in the MOTION repository means that the user is invoking these operations in her own *resource space*. In order to share an artifact with others in a community, the artifact is *tagged* as belonging to a specific community and is visible to other users as long as the user that owns the artifact is online. If a user wishes to make an artifact persistent so that it is available to others even when she is not online, she can copy it into the so called *community cabinet*. The community cabinet denotes the repository of a host that acts as a server and is always connected.

In our prototype, we use the GMD XQL engine and the GMD IPSI XML repository V1.0.2[9] as the XML repository.

3.5 Artifact Manager

The artifact manager component is composed of the MOTION repository component and the repository manages.

The repository manager component is responsible for mapping remote transfer requests to commands in the repository. It retrieves, inserts, deletes or queries the information in the repository and provides the communication infrastructure between artifact exchanging peers. The artifact transfer protocol is HTTP. For example, when *Dr. Jaza* issues a distributed XQL request and sees that *Dr. Marco* has related work on information sharing, he can download that article from *Dr. Marco*'s repository. The repository manager component takes care of transferring the article (i.e., artifact) from the remote repository into *Dr. Jaza*'s local repository.

The Artifact Manager component acts as a wrapper to the MOTION Repository (i.e., XML and artifact repositories) and the Repository Manager. It provides artifact management API calls in the TWS API (e.g., insert an artifact, download an artifact, etc.).

3.6 Distributed searches

One of the distinguishing features of the services provided by the MOTION platform is its support for distributed searches. A key requirement in the MOTION industrial case-studies was the ability to locate information in a loosely coupled, distributed setting. Large organizations often have employees that do not personally know each other and in most cases cannot benefit from the work others are doing. For example, a group working on transistor design in Austria might have a problem that a group in the company located in South Africa branch has already solved. The ability to query artifacts, hence, is beneficial and in some cases success critical.

The TWS API provides querying mechanisms to search the artifacts that are in the MOTION system. The user can define XQL queries that are propagated through the system. The concept of distributed searches in the system is similar to searching provided by peer to peer systems such as Gnutella[10], Morpheus[15] and Napster[16]. Whereas these systems only provide search support for document file names, searching in the TWS API is more advanced and has a finer granularity. For example, *Dr. Jaza* might search for related work by specifying a query that identifies all artifacts that containing the keywords "WICSA, Mobility", the string "mobile working" in the description and that was authored by a user that is working on collaborative systems.

The XQL query in the prototype is composed with the help of user interface elements such as combo boxes and tables. The user, hence, does not need to know XQL.

4. RELATED WORK

Much has been written to date on collaborative tools and systems and many research prototypes and commercial systems have been built. The majority of these systems, however, are not concerned with device and user mobility.

One of the first projects to tackle mobility issues in collaborative applications is StudySpace[21]. Although StudySpace attacks problems such as determining network, hardware and display capabilities before fetching a document, it does not address subscription, distributed searching and community support issues.

Several authors have identified the need to develop groupware systems that support user mobility.

DACIA[12] is a system that provides mechanisms for building groupware applications that adapt to available resources. Using DACIA, components of a groupware application can be moved to different hosts during execution, while maintaining communication connectivity with groupware services and other users. The system, however, does not provide higher-level service support for requirements that MOTION deals with such as notification and information sharing. Important mobile teamwork features such as file sharing, user awareness (i.e., notification that user X is online) and access control are not addressed by DACIA.

MOST[1, 4] provides five components: A group coordination module, a shared graphical editor module, a remote database module, a collaborative viewing module, a collaborative image viewing module and a job dispatching module. The functionality provided by the group coordination module is comparable in some respect to the community support provided by MOTION. The aim of MOTION, however, is not to provide a collaborative application, but a framework and platform for the development of business-specific applications. Further, MOST does not support emerging mobile teamwork service requirements that we have mentioned.

Sync[14] is a Java-based framework for developing collaborative applications for wireless mobile systems. Sync is based on object-oriented replication and offers high-level synchronization-aware classes based on existing Java classes. MOTION mainly differs from Sync because it provides an API to *higher-level* teamwork services.

A few commercially available collaborative systems (e.g.,[18]) are starting to support some form of mobility (e.g., WAP access), but they are usually not easily customizable. Further, they were reported by our case-study global organizations as having insufficient teamwork support and not covering the requirements we address in this paper.

One system that supports the querying of XML meta-data on artifacts is Roma[22]. This system, though, is not a collaborative system and only supports the querying of a central repository by mobile users and mobile-aware applications. Furthermore, it uses a lightweight XML query language and does not support powerful XQL queries as we do.

5. CONCLUSION

Mobile computing devices are becoming more powerful every day and restrictions such as memory limitations and low CPU power are disappearing. For example, PDAs such as the Compaq iPAQ are able to run Java 1.3 programs with Swing user interfaces. Thus, anytime, anywhere access to information has become interesting to many organizations and there is a need for new working ways and technologies for distributed, mobile teamwork.

The MOTION prototype we have built is currently being evaluated in two industry case-studies and business-specific user interfaces are under construction for Web browser and PDA access to MOTION services.

The MOTION basic system libraries consist of about 30,000

lines of Java code (without the off-the-shelf components) and have been tested under the Windows 2000 and Linux Red Hat 7.1 operating systems on notebooks and desktops, and under Familiar Linux on the iPAQ.

Many existing collaborative tools and systems are not customizable, are heavyweight, and do not have support for emerging requirements such as the locating of artifacts and experts through distributed searches, information subscription and notification, and mobile information sharing and access.

In this paper, we described the architecture and the components of our generic MOTION service platform for building collaborative applications. As more progress is made in mobile computing, we expect the demand for mobile teamwork architectures and collaborative platforms to grow.

6. REFERENCES

[1] K. Cheverst, G. Blair, N. Davies, and A. Friday. The support of mobile-awareness in collaborative groupware. *Personal Technologies*, 3(1-2):33 42, 1999.

[2] G. Cugola and E. D. Nitto. Using a Publish/Subscribe Middleware to Support Mobile Computing. In *Proceedings of the Workshop on Middleware for Mobile Computing, in association with IFIP/ACM Middleware 2001 Conference, Heidelberg, Germany*, November 2001.

[3] G. Cugola, E. D. Nitto, and A. Fuggetta. The JEDI Event-Based Infrastructure and its Application to the Development of the OPSS WFMS. *Transaction of Software Engineering (TSE)*, 27(9), September 2001.

[4] N. Davies, G. Blair, K. Cheverst, and A. Friday. Supporting collaborative applications in a heterogeneous mobile environment. In *Computer Communications Special Issue on Mobile Computing, Internal report number MPG-94-18.*, 1996.

[5] P. Fenkam, H. Gall, G. Reif, and E. Kirda. A Dynamic and Customizable Access control System for Distributed Applications. Technical report, Distributed System Group, Technical University of Vienna, December 2001.

[6] P. C. Fenkam. Dynamic User management System for Web Sites. Master's thesis, Graz University of Technology and Vienna University of Technology, September 2000. Available from http//www.ist.tu-graz.ac.at/publications.

[7] D. Ferraiolo and R. Kuhn. Role-Based Access Controls. In *Proceedings of 15th NIST-NCSC National Computer Security Conference*, pages 554–563, October 1992.

[8] W. A. P. Forum. Wireless Application Protocol Service Indication Specification. Technical report, Wireless Application Protocol Forum, November 1999. Available at http://www.wapforum.org/.

[9] GMD. XQL IPSI, http://xml.darmstadt.gmd.de/xql/, 2002.

[10] gnutella.com. Gnutella, http://www.gnutella.com, 2002.

[11] M. Hauswirth. *Internet-Scale Push Systems for Information Distribution—Architecture, Components, and Communication, Distributed Systems Group, Technical University of Vienna*. PhD thesis, October 1999.

[12] R. Litiu and A. Prakash. Developing adaptive groupware applications using a mobile component framework. In *ACM 2000 Conference on Computer Supported Cooperative Work. ACM, New York, NY, USA*, page 107 116. ACM Press, 2000.

[13] K. McCrary. JTella Homepage, http://www.kenmccrary.com/jtella/, 2002.

[14] J. Munson and P. Dewan. Sync: a Java framework for mobile collaborative applications. *IEEE Computer*, 30(6):59 66, June 1997.

[15] MusicCity. Morpheus, http://www.musiccity.com, 2002.

[16] Napster. Napster homepage, http://www.napster.com, 2002.

[17] G. P. Picco and G. Cugola. PeerWare: Core Middleware Support Peer-To-Peer and Mobile Systems. Technical report, Dipartimento di Electronica e Informazione, Politecnico di Milano, 2001.

[18] Pragmatyxs. Pragmatyxs, http://www.pragmatyxs.com, 2002.

[19] G. Reif, E. Kirda, H. Gall, G. P. Picco, G. Cugola, and P. Fenkam. A Web-based peer-to-peer architecture for collaborative nomadic working. In *10th IEEE Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), Boston, MA, USA*. IEEE Computer Society Press, June 2001.

[20] R. S. Sandhu. The Schematic Protection Model: Its Definition and Analysis for Acyclic Attenuating Schemes. *Journal of the ACM, 35(2):404-432*, April 1988.

[21] J. L. Schnase, E. L. Cunnius, and S. B. Dowton. The StudySpace Project: Collaborative Hypermedia in Nomadic Computing Environments. *Communications of the ACM*, 38(8):72–3, August 1995.

[22] E. Swierk, E. Kiciman, V. Laviano, and M. Baker. The Roma Personal Metadata Service. In *Proceedings of the Third IEEE Workshop on Mobile Computing Systems and Applications*, page 107 116. IEEE Computer Society, Los Alamitos, CA, USA, 2000.