Evaluation of a Publish/Subscribe System for Collaborative and Mobile Working

Pascal Fenkam, Engin Kirda, Schahram Dustdar, Harald Gall and Gerald Reif

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

Abstract

The MObile Teamwork Infrastructure for Organizations Networking (MOTION) [1] *service platform that we have designed and implemented addresses an emerging requirement in the daily business of large, distributed enterprises: support for mobile teamwork. Employees are often on the move and use a wide range of computing devices such as WAP phones, PDAs, notebooks and desktop computers. The service architecture that we have developed supports mobile teamwork by providing multi-device service access, XML meta data for information sharing and locating, and the XML Query Language (XQL) for distributed searches and publish/subscribe. We present the solution that we adopted in our prototype, analyze the shortcomings of this approach and based on our evaluation experiences, list the requirements for a publish-subscribe middleware for collaborative mobile working.*

Keywords: MOTION, Mobile Teamworking, Evaluation, XML meta-data and XQL, Publish/Subscribe, Mobile and distributed collaboration

1 Introduction

Large, global organizations are usually distributed across sites that are located in different countries. Employees are often on the move and need to collaborate on documents. Advanced mechanisms are needed to enable employees to work together by exchanging, locating and sharing information and communicating with their co-workers. Hence, many organizations are increasingly faced with the problem of supporting nomadic workers [12, 15].

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

The MObile Teamwork Infrastructure for Organizations Networking (MOTION) system that we have designed and implemented addresses the mobile teamwork requirements of organizations in their daily business. It provides mobile teamwork support by covering requirements such as locating distributed business documents and expertise through peer to peer searches, advanced subscription and notification, community building, and mobile information sharing and access.

In this paper, we present an evaluation of the subscription/notification facility of our MOTION system. The purpose of this evaluation is to investigate how well our design decisions fulfill the functional and non-functional requirements of *context awareness* (i.e., information on what other people are doing, the current state of business documents, resources, etc.).

Two types of evaluation of awareness in collaborative systems are advocated in the literature [5]: evaluation of a system from user's perspective and evaluation of technical performance. The evaluation case study presented in this paper is classifiable in the second category.

We present the solution for awareness that we adopted in our prototype, analyze the shortcomings of this approach and based on our experiences with the system prototype, list the requirements for an effective publish-subscribe middleware for collaborative mobile working. The difficulties encountered during the design and implementation of the MOTION Teamwork Services are our principal motivation for early evaluation.

The remainder of the paper is structured as follows. The next section gives a brief introduction to the industrial case studies of the MOTION project. Section 3 gives an overview of the architecture of the MOTION platform. Section 4 presents the architecture of our subscription and notification components in MOTION. Section 5 discusses the important decisions that guided our design. Section 6 presents the requirements for publish/subscribe systems for collaborative and mobile working. Section 7 presents

Proceedings of the Eleventh IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'02)
1080-1383/02 $17.00 © 2002 **IEEE**

a possible candidate middleware solution for solving the problems we encountered in the implementation of the MOTION subscription and notification facility. Section 8 concludes the paper.

2 Case study companies

In this section, we give a brief overview of the case study global companies and discuss the main problems that the MOTION system attacks.

2.1 Case study: Mobile phone design

The first case study is a global company involved in the market of telecommunication equipment and systems. The company is well-known in the market of mobile phones and is globally distributed across sites in several countries.

The company would like its employees to communicate and collaborate on mobile-phone related artifacts (i.e., documents) and to locate "experts" whenever they face technical or organizational questions. Experts are employees that are highly specialized in a specific mobile-phone related domain.

An information infrastructure, hence, is needed that helps to arrange virtual and face-to-face meetings and provides subscription and notification mechanisms. For example, when an expert is available who is knowledgeable about the software of the phone version 3218A, all subscribed employees are notified.

2.2 Case study: Household appliances

The second industry case study is a global, well-known producer of white goods.

The company has manufacturing experts who travel around the world and need ubiquitous access to information. The main problem the company would like to solve is to support these experts in querying distributed knowledge repositories. Similarly, the company would like to enable its employees to locate the experts they need based on the contents of documents users read or edit.

2.3 Publish/subscribe context awareness

As can be seen in the case study descriptions, both case study organizations need advanced distributed querying as well as subscription and notification mechanisms. In both of the case studies, we identified the publish/subscribe paradigm (enforced with appropriate messaging and meta data mechanisms) as a suitable solution to increase the availability and awareness of documents and users.

An employee might issue a query, for example, to locate experts on "valve design." If she does not find any, she can subscribe to such experts and is notified whenever the system discovers an expert in this area. An employee, hence, must be able to receive notifications on the availability of any information she is able to search for.

3 Brief overview of the MOTION architecture

We refer to every computing device that is connected to the MOTION system as a *peer*. The MOTION system typically consists of desktop computers, notebooks and PDAs.

Usually, every peer has a local repository that the user can use to store business documents (i.e., *artifacts* in MOTION) that she can share in virtual communities with other connected users. XML Meta-data (i.e., *profiles* in MOTION) is stored about each artifact, hence, enabling searches by other users. Some clients such as WAP-enabled mobile phones and Web browsers are thin clients that do not host a repository or any services, but only access resources remotely.

The MOTION system has a three layer architecture. The bottom layer, the communication middleware, offers basic communication services such as publish/subscribe mechanisms, peer to peer file sharing and distributed searches. We use the PeerWare [10] system as the communication middleware infrastructure in the current prototype. PeerWare provides publish/subscribe and distributed search propagation functionality.

The middle layer in the architecture, the Teamwork Services (TWS) layer, is built on top of the communication middleware. This layer is responsible for the integration of the main components of the system (e.g., access control, user and community management, repository).

The TWS layer provides an Application Programming Interface (API) to generic services such as storing and retrieving artifacts in the local repository and from remote repositories on other peers, creating and managing virtual communities, sending and receiving messages from other users and distributed search specification and invocation.

The presentation layer, as the top layer in the architecture, provides the user interface to the MOTION services and is built using the TWS API.

4 Publish/subscribe in MOTION

The publish/subscribe paradigm has been identified as an architectural style that fosters mobility and high decoupling of components (e.g., [4, 16]).

The publish/subscribe component in the Teamwork Services Layer of the MOTION system wraps the underlying middleware and gives a uniform and consistent view of the *event* concept to the application layer.

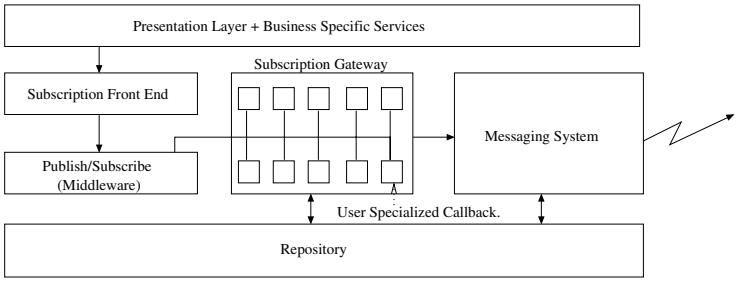

Figure 1. The TWS Layer Publish/Subscribe Architecture

Typically, publish/subscribe systems enable components to subscribe and react to events by specifying user written callback methods that are invoked by the system whenever an event occurs that matches a subscription. A system-specific query language is often used to create subscription criteria.

Having only a publish/subscribe middleware, however, is not enough to easily construct collaborative distributed applications. This is because there is no direct mapping between component-level (system) subscriptions and user-level (application) subscriptions. The repository component, for example, may use the publish/subscribe middleware to subscribe to all component-level events generated by the access control component. Enabling a user *Tom* to subscribe to artifacts created by *Joe*, in contrast, is a higher-level event and needs a mapping of component-events to user-level events in the system.

We use so called *subscription gateways* in our system and utilize *user specialized callbacks* (see Figure 1). 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 with subscription criteria. This callback mediates between the underlying publish/subscribe system and the user and subscribes on her behalf. Hence, when an event occurs that satisfies one of the user's subscription criteria, the user specialized callback is invoked and it transforms the received event into a message. The message is sent to the user and contains the details about the event (e.g., URLs pointing to documents, notification details, etc.).

The subscription gateway denotes the set of callback components running on a particular peer. A subscription gateway has to be configured for each user. Note that choosing a subscription gateway is a configuration issue because every peer can be used as a subscription gateway. It is important, of course, that the peer stays connected to the system (e.g., a PDA peer would not be a suitable gateway because the user is not always connected to the network).

Different middleware implementations provide different subscription languages. To create a flexible system that is not dependent on a specific publish/subscribe system, it is necessary to hide the differences.

We chose to use an XML query language (XQL) [14]. In general, XML query languages give the business-specific services the capability to query complex XML events and data. In particular, we chose XQL because it was one of the currently available languages at the time the project started. The different XML query languages have now been merged into XQuery [2] which is still a moving target.

A user, for example, that wishes to be notified whenever an expert in "valve design" goes online could formulate the following XQL query with a client:

```
/UserProfile[(User/Expertise/* $contains$
'valve design') $and$ System/Access/Status
$equals$ 'online']/ID
```

(e.g., see Figure 2). Note that the XQL query defines criteria based on the contents of XML profiles in the system (e.g., The tag *Expertise* defines a string description of the expertise of each user in this case).

Suppose a user `Joe` is registered in the MOTION system. Whenever he goes online, his profile is included in an event that is published. All subscribed users are notified. The XML description of such an event may look as shown in Figure **??**

Two classes of subscriptions exist in the MOTION system: *system subscriptions* and *user subscriptions*. User subscriptions are subscriptions initiated by users. Resulting messages are sent using different communication means (e.g., SMS, specific MOTION Messages, e-mail, etc.).

System subscriptions generate events that inform components of system-specific activity. When, for example, a new MOTION user is created, an event is published for two reasons: First, interested users (i.e., those that have subscribed to the new user details) are notified. Second, the repositories of other peers are updated (events are queued for disconnected peers).

5 Related approaches and design decisions

In this section, we give an overview of existing awareness solutions in collaborative systems and discuss design

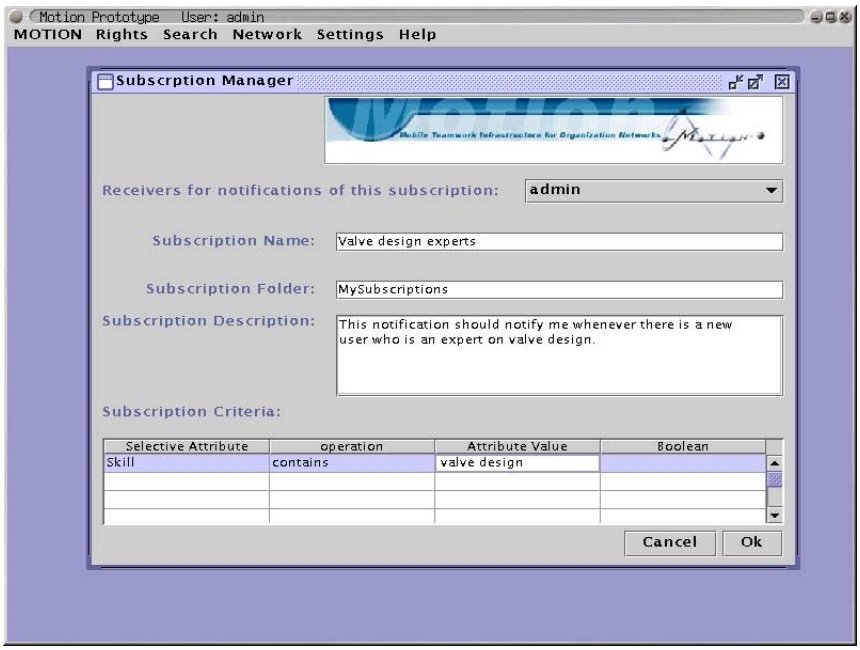

Figure 2. Creating a subscription with the native Java MOTION client

```
<UserProfile ID="smith">
   <System>
      <TimeZone>GMT</TimeZone>
      <Location>Vienna</Location>
      <Access Status="online">
          <Source>
           amd16.infosys.tuwien.ac.at
           </Source>
      </Access>
      <LastProfileUpdate>
           23.05.2001
       </LastProfileUpdate>
   </System>
   <User>
       <Name>
            <FirstName>Joe</FirstName>
            <LastName>Smith</LastName>
        </Name>
        <Expertise>
            <Role>Research Assistant</Role>
            <Skill>Walve design, java,
             mobile  computing</Skill>
            <Group>DSG</Group>
            <Language>English</Language>
            <Language>German</Language>
        </Expertise>
        <Address>
            <Smail>
               <Street>Argentinierstraße
                   8/184-1</Street>
               <ZIP>A-1040</ZIP>
               <City>Vienna</City>
                <State>Austria</State>
            </Smail>
            <Email>smith@gmx.net</Email>
            <Phone>555 123 123</Phone>
            <Mobile>555 123 123</Mobile>
        </Address>
   </User>
</UserProfile>
```

Figure 3. XML description of an event

decisions that differentiate our platform from existing systems.

User awareness has been recognized as an important value for collaborative environments. Khronika [6], Interlocus [9], Awareness@work [13], Nessie [11] are examples of collaborative systems that enable various levels of awareness. Important features provided by each of these systems range from security and access control to persistency. The majority of these systems are not concerned with device, and user mobility and do not provide support for distributed searching. Further, most systems are client-server based and fail to scale for widely distributed organizations.

One of the motivations for choosing PeerWare, a Peer-to-Peer (P2P) system, as middleware in MOTION was because the P2P architectural style has been increasingly gaining attention. The P2P style provides high flexibility and system configurability. Considering that our system has to be scalable for widely distributed organizations with more than 60.000 employees, we could not rely on the client-server paradigm proposed by most collaborative systems.

One other reason was because PeerWare provides *both* distributed searching/P2P file sharing and publish/subscribe support. There are middleware solutions that provide either publish/subscribe (e.g., [16, 1, 7]), or P2P file sharing (e.g., [8]), but an integrated solution such as PeerWare is uncommon.

We also had access to PeerWare's source code and could make modifications and extensions as necessary.

6 Requirements for an effective publish/subscribe middleware

In this section, we analyze our system and based on our experiences with the system prototype, list the requirements we identified for an effective publish-subscribe middleware for collaborative mobile working.

Requirement 1: An efficient filtering mechanism

This requirement seems to be the problem of most publish/subscribe systems. In user-oriented publish/subscribe systems, the number of subscriptions can easily grow.

In our case, it was not possible to optimize the filtering mechanism of PeerWare based on the semantics of filters and events. The filtering process is performed by Peerware while the semantics of this operation is the responsibility of application developers. If m is the number of subscriptions on a peer and n is the number of events to be matched, the filtering is performed with $\theta(mn)$ time complexity. With 10 millions of subscriptions, if processing one subscription requires t seconds, the system will need $5t$ billion seconds to match them to 500 incoming events. This is an alarming situation that often reflects reality.

Two existing technologies that could help solve this problem at the middleware level are parallel search trees (used in Gryphon [1]), and BDDs (used in SPEAR [3]).

Requirement 2: The capability to automatically destroy a subscription after a given delay

We are not aware of any publish/subscribe middleware that provides the simple functionality of automatically deleting subscriptions after a specified timeout. Proxy Elvin [16] and Khronika [6] provide a time-to-live for subscriptions. However, this time-to-live does not determine how long the subscription must live, but how long notifications matching a subscription must be stored.

This capability can easily be built by constructing a layer on top of any publish/subscribe system. Integrating this functionality into the middleware, however, is better because it saves development effort of the application developer.

This requirement is important because of organizational reasons. If users are able to create subscriptions at will, they may create and forget many subscriptions. These subscriptions will affect the performance and the scalability of the system. Hence, having a default policy of clearing subscriptions after some time can help solve problems. The mechanism we describe is similar to that applied by Web-based e-mail providers who delete unread emails after about two-three weeks.

Based on empirical results, Khronika's [6] evaluators also state the capability to destroy subscriptions as an important requirement. Unfortunately, the requirement is not yet fulfilled by their system.

Requirements 3: Authorization mechanisms for subscriptions

In collaborative applications, access restrictions and encryption mechanisms are needed when sharing information. It may not be desirable, for example, to allow every employee to subscribe to a specific document or be informed that a new mobile phone is being designed.

Most of the existing publish/subscribe middleware solutions do not have any integrated security and authorization mechanisms. There have been some attempts to integrate security considerations into publish/subscribe systems (e.g., Gryphon [1] and KERYX [7]), but there is no general consensus on what the best way of doing this.

More work needs to be done in identifying what the security threats in publish/subscribe systems are, and how they can be avoided without creating the publish/subscribe bottleneck.

Requirement 4: An expressive query language for defining subscriptions

The subscription language that is provided by most publish/subscribe middleware solutions is often too simple for user and application-level subscriptions. In Section 4, we discussed how we used XQL to create user and application-level subscriptions on top of a publish/subscribe middleware.

The disadvantages of this approach are that there was an increased implementation overhead and our system became less flexible because it was more difficult to switch to another publish/subscribe system. Subscription language standards for publish/subscribe systems, hence, need to be defined that enable application developers to build more flexible and complex applications.

XML seems to be a good candidates for such a language. Subset of XML query languages can be used for subscriptions. Although we used a subset of XQL for this purpose, any other XML query language such as XQuery can be used. An empirical evaluation of the Khronika system also identifies the lack of an event/subscription description language as a handicap.

7 A promising direction: SPEAR

SPEAR[3] (Scalable Publish subscribe Architectures for Efficiency and Robustness) is a publish/subscribe system designed with the goal of efficiency. Its filtering mechanism

is based upon BDDs (Binary Decision Diagrams). In Spear, events are described using a subset of XML. Correspondingly, the subscription language is a subset of XQL. Experimental results show that Spear is capable of matching 1000 events against 500,000 subscriptions in 15.57 seconds. A detailed comparison of SPEAR to other publish/subscribe systems (e.g., Gryphon, SIENA, ELVIN, KERYX, JMS) can be found in [3].

According to the description in [3], SPEAR solves two requirements we listed for building publish/subscribe-based collaborative applications: the support for an expressive enough query language (e.g., XML/XQL) and an efficient filtering mechanism.

We are planning to deploy other publish/subscribe systems in our prototype to identify the most suitable middleware approach for collaborative working applications.

8 Conclusion

The MOTION mobile teamwork service platform that we have designed and implemented addresses the emerging requirement of mobile teamwork support in distributed enterprises.

There has been a growing interest in publish/subscribe systems to engineer distributed systems. In this paper, we presented an early evaluation of the subscription and notification mechanism of our MOTION system.

We described the solution we adopted in our prototype implementation and discussed the shortcomings of our approach. We listed four important requirements for an efficient publish/subscribe middleware for collaborative mobile working. We plan to complement and extend the requirements list with empirical evaluations of the case studies.

Enterprise-scale publish/subscribe middleware solutions need to address scalability issues by providing more expressive subscription languages, subscriptions with timeouts and authorization and filtering mechanisms.

Acknowledgements

We would like to thank the MOTION team and partners for their contribution in the project.

References

[1] M. Aguilera, R. Strom, D. Sturman, M. Astley, and T. Chandra. Matching events in a content-based subscription system. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC 99)*, 1999.

[2] S. Boag, D. Chamberlin, M. Fernadez, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu. XQuery 1.0: An XML Query Language (XQL). Technical report, World Wide Web Consortium, April 2002. Available from http://www.w3.org/TR/xquery.

[3] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith. Efficient Filtering in Publish/Subscribe Systems using Binary Decision Diagrams. In *Proceedings of the 21st International Software Engineering Conference (ICSE), Toronto, Canada*, May 2001.

[4] 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.

[5] T. Hall. Practitioner's guide to evaluating collaborative systems. In *Proceedings of the 10th IEEE International Workshop on Enabling Technologies (WET ICE 2001) Infrastructure for Collaborative Enterprises, Cambridge, MA, USA*, July 2001.

[6] L. Lövstrand. Being selectively aware with the khronika system. In *Proceedings of the 6th European Conference on Computer Supported Cooperative Work-ECSCW'91*, September 1991.

[7] C. Low. Integrating communication services. *IEEE Communications Magazine*, 35(6), 1997.

[8] K. McCrary. Jtella homepage, http://www.kenmccrary.com/jtella/, 2002.

[9] T. Nomura, K. Hayashi, T. Hazama, and S. Gudmundson. Interlocus: Workspace configuration mechanisms for activity awareness. In *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work, Seattle*, pages 19–28, November 1998.

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

[11] W. Prinz. Nessie: An awareness environment for collaborative settings. In *Proceedings of the 6th European Conference on Computer Supported Cooperative Work-ECSCW'99*, pages 391–410, September 1999.

[12] G. Reif, E. Kirda, H. Gall, G. P. Picco, G. Cugola, and P. Fenkam. 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.

[13] K. O. Sandor and A. Schmer. Supporting social awareness @ work, design and experience. In *Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work, Boston*, 1996.

[14] D. Schach, J. Lapp, and J. Robie. XML Query Language(XQL). Technical report, World Wide Web Consortium, September 1998. Available from http://www.w3.org/TandS/QL/QL98/pp/qxql.html.

[15] J. Schiller. *Mobile Communications*. Addison-Wesley, Reading, Mass. and London, 2000.

[16] P. Sutton, R. Arkins, and B. Segall. Supporting disconnectedness-transparent information delivery for mobile and invisible computing. In *Proceedings of 2001 IEEE International symposium on Cluster Computing and the Grid (CCGrid'01)*, May 2001.