Dealing with Uncertainty in Mobile Publish/Subscribe Middleware

Ludger Fiege Andreas Zeidler
Darmstadt University of Technology (TUD)
64283 Darmstadt, Germany
{fiege|az}@dvs1.informatik.tu-darmstadt.de

Felix C. Gärtner Sidath B. Handurukande
Swiss Federal Institute of Technology (EPFL)
1015 Lausanne, Switzerland
fcg@acm.org, sidath.handurukande@epfl.ch

Abstract

Because of its loose coupling between event producers and consumers, publish/subscribe (pub/sub) middleware has many advantages when implementing systems for spontaneous, ad-hoc, pervasive applications. One main aspect of such applications is device mobility, but unfortunately, most of the current pub/sub systems do not adequately support mobile clients. Mobility has two orthogonal aspects: *physical mobility* is concerned with location transparency (i.e., roaming clients) while *logical mobility* deals with location awareness (i.e., subscriptions are automatically adapted to a client's current location). To efficiently support mobility, it is necessary to adequately deal with the uncertainty introduced by client movement. This paper sketches how this is done in the existing pub/sub middleware REBECA and shows how to increase the efficiency of logical mobility by adapting the implementation of physical mobility. The paper closes with a list of open research issues related to the use of pub/sub middleware in the context of mobile and pervasive computing.

1 Introduction

Publish/subscribe systems for pervasive computing. The publish/subscribe (pub/sub) communication paradigm is increasingly used in many application domains and areas of computer science. It allows processes to exchange information based on message type or content rather than particular destination addresses. Information about some event is published via notifications, which are conveyed by the underlying pub/sub notification service. A consumer registers its interest in certain kinds of notifications by issuing subscriptions, and it gets notified by the notification service about any newly published notification that matches at least one of its subscriptions. The *loose coupling* of producers and consumers is the prime advantage of pub/sub systems and has many applications in the context of spontaneous, ad-hoc and pervasive environments.

Mobility support in pub/sub middleware. One major characteristic of pervasive applications is mobility. However, up to now research is mainly focused on using pub/sub middleware in rather static, non-mobile environments, i.e., systems where clients (producers and consumers) do not roam and the infrastructure itself stays rather fixed or is only changing slowly during the system's lifetime. Consequently, most pub/sub infrastructures (e.g., SIENA [1], JEDI [2], REBECA [3], to name a few) have optimized algorithms for information delivery in those settings. Support and optimizations for mobile clients are not built-in features of the infrastructure; it is left to the applications to adapt or reissue subscriptions.

Location transparency and physical mobility. A first step towards mobility is to enhance existing pub/sub middleware to allow for roaming clients so that existing applications can be used in mobile environments. This means that the interfaces for accessing the middleware and the applications on top are not required to change. More importantly, the quality of service offered by the middleware must not degrade substantially. Generally speaking, *location transparency* is what makes existing applications mobile, e.g., stock quote monitoring can be seamlessly transferred from PCs to PDAs. Location transparency is the main aspect of what is called *physical mobility*.

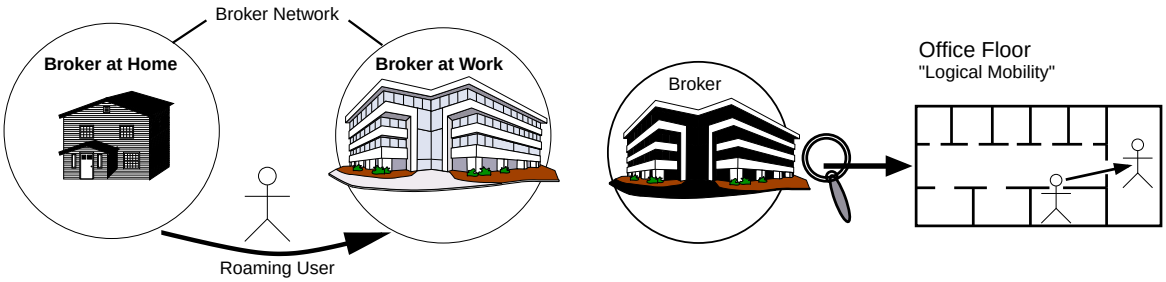

Figure 1: Physical mobility (left) and logical mobility (right).

Physical mobility is also a concern in the context of disconnections. Clients often have to disconnect from the pub/sub system (due to power-saving requirements or because of geographical or administrative reasons while roaming, e.g., different network cells or changing responsibilities). Often, the *border broker* (i.e., the access point to the pub/sub network) has changed after reconnecting. For example, the border broker at home is (physically and administratively) not the same as the border broker at the office (see left side of Fig. 1). For the ease of use, change of location should be transparent to the application as long as possible.

Location awareness and logical mobility. Full location transparency can be counterproductive in some cases. Most strikingly, location-based services (like pervasive tourist guides) rely on an explicit knowledge about the current location. More specifically, a system supporting mobility should not only blend out unwanted phenomena, like disconnectedness, but should also facilitate *location awareness*. Location awareness is the main characteristic of what is called *logical mobility*, that is, mobility that transfers an application into a different context that requires to adapt subscriptions (note the difference to the logical mobility as defined in LIME [4] where it denotes code mobility).

A simple way to support logical mobility is to provide *location-dependent filters* [5]. A location-dependent filter is a subscription that refers to the current location of the subscriber, thus making *location* a first-class concept within the pub/sub system. More precisely, location-dependent subscriptions postulate a specific marker *myloc* to be used in a subscription. The marker stands for a specific set of locations that depends on the current location of the client.

For example, a client could express his interest in all temperature readings referring to his current location (i.e., the particular office) as follows: $(\mathit{service} = \text{“temperature”}), (\mathit{location} \in \mathit{myloc})$. The pub/sub system ensures that this subscription is always mapped to an appropriate set of locations which is application dependent (see right side of Fig. 1).

Logical and physical mobility in REBECA. Physical and logical mobility are two orthogonal aspects of mobility that can be treated completely separately. While the former deals with rerouting subscriptions to new locations, the latter automates the adaption of subscriptions. While the former is bound to the granularity of the broker network, the latter can refer to a totally different notion of “location” (physical, system dependent vs. logical, application dependent).

Logical and physical mobility have been studied in the context of the REBECA system [6, 7]. The main problem when implementing both types of mobility is to deal with the uncertainty in a client’s movement. When implementing physical mobility, a complex reconfiguration algorithm combined with a certain amount of buffering ensures that a relocated client receives a transparent, uninterrupted flow of notifications matching his subscriptions [8].

When implementing logical mobility, a mobile client moving from one location to another, e.g., from one room to another on an office floor, will also expect a frictionless change of location without a notable setup time. The adaptation of location-dependent subscriptions should take place “instantaneously.” Intuitively, we would like to experience the notion of being subscribed to “everything, everywhere, all the time” and increase the reactivity of the system to clients moving. In the algorithm for logical mobility [5] this is achieved by using a *movement graph* that reflects the possible locations a client might move to. This graph can be regarded as a formalization of movement constraints, thereby making the *uncertainty* exploitable by the mechanism of event routing.

Aim of this paper. We have observed that the notion of a location is not completely different in logical and physical mobility. In general, the movement graph in logical mobility is a refinement of the graph of possible border brokers, i.e., logical mobility allows for mobility within the scope of a single broker (as shown in Fig. 1). In this paper, we sketch how to exploit this observation to implement an efficient handling of logical mobility in cases where a change in logical location coincides with a change in physical location.

Approach: Pre-subscriptions and virtual clients. In the current implementation [5], location-awareness is only efficiently supported if client movements remain within the boundaries of a single border broker. Whenever a client leaves this range, the location-dependent subscriptions have to be re-issued at the next broker the client connects to causing a non-negligible overhead. This seems to be acceptable whenever a change of broker also implies a change of "view" or applications. For example, location-aware applications typically found in an office environment usually are different from those found at home. But on the other hand, some applications in the context of pervasive computing do not draw from those "logical localities" but rely on a longer lasting and more general notion of location-awareness, e.g., the weather at the region someone is currently located in or the menus of restaurants along the route of a car. Here, another notion of quality-of-service is needed: the client wants to be informed about the situation at a new location immediately.

Additionally, the client cannot rely on the fact that notifications, for example, of the type "restaurant menu," happen to be published just as the client enters the new broker's range. The client may miss important notifications by a fraction of a second. In general this is not avoidable without explicit support of *event histories*, introducing a complete new set of open research issues, e.g., expiration and relevance of information. To circumvent such problems we propose a more hands-on approach and introduce *pre-subscriptions* and *virtual clients*: location-dependent subscriptions are issued at probable new locations the client will emerge at *in the future* while moving, before it actually gets there. In a sense pre-subscriptions are "virtual clients" or "information shadows" cast at possible new locations, subscribing to information related to the new location with only a proxy being there. This approach exploits the ideas in the REBECA implementation of logical-mobility [5]. The basic idea is to emulate the exact behavior of the real client, with respect to location-dependent information, without processing the information but buffering them instead. This is exactly the "listen for a while" semantics we intended: once a client actually arrives, all buffered messages are delivered as if the client has been there some time. In a sense, for the client this is equivalent to a *subscription in the past*.

Related work. We are only aware of some pub/sub systems offering support for mobile clients to some extent. Huang and Garcia-Molina [9] provide a good overview of possible options for supporting mobility. An extension to Elvin exists that allows for disconnectedness using a central caching proxy [10], but it is not used for location-dependent subscriptions. CEA [11] and JEDI [2], too, tackle problems of mobility. JEDI uses explicit moveIn and moveOut operations to relocate clients but also has no explicit notion of location as a first class concept for pub/sub systems. The mobility extensions of SIENA [12] are very similar to the JEDI approach. Probably the most related work is STEAM [13], a middleware service designed for wireless local area networks using the ad-hoc network model where there are no access points and system wide services. Subscribers only consume notifications produced by geographically close-by publishers. For this it relies on proximity-based group communication. As a result it is not clear how this approach can be applied in an application like weather forecast for a particular remote location produced by a forecasting service in a given region. Other communication paradigms facilitating loose coupling, like Linda tuple spaces [14], were also investigated for their potential support of mobility (e.g., in LIME [4]). However, for this paper we concentrate on pub/sub systems and mobility.

Paper outline. We first give a very brief background of basic pub/sub terminology in Sect. 2. The extension of current mobility approaches are presented in Sect. 3. We conclude this paper with a list of open research questions which we think are worthwhile to pursue.

2 Content-Based Publish/Subscribe

The following discussion is based on the REBECA notification service [3, 7], which we use as basis for the proposed mobility support.

Architecture. Processes of a system based on pub/sub communication, which is also called an *event-based system*, can act both as producers and consumers, they are

clients of the underlying notification service. The communication interface to the service is rather simple and consists of *pub*, *sub*, *unsub*, and *notify* calls only; the last one is an output function called on the registered client to deliver a notification. A *notification* is a message that reifies and describes an occurred event. Notifications are not published towards a specific receiver, but conveyed by the underlying notification service to those consumers that have registered a matching subscription.

Filters are boolean-valued functions over notifications and a common way of implementing subscriptions. The most flexible scheme for specifying these filters is content-based filtering, which utilizes predicates on the entire content of a notification [15].

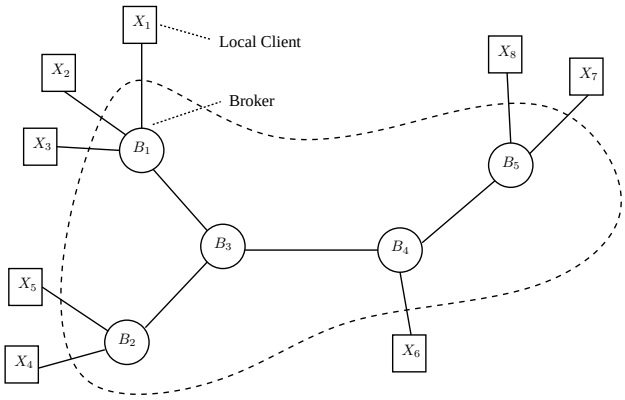

Figure 2: The router network of REBECA.

The service implementation is distributed to meet the mobility scenario and scalability considerations. The communication topology of the pub/sub system is given by a graph, which is assumed to be acyclic and connected (Fig. 2). The graph consists of brokers and clients. The edges are communication links that are point-to-point. Furthermore, messages are required to be delivered in FIFO order on each link. Brokers are processes that route the notifications along multiple hops to the appropriate clients. Three types of brokers are distinguished: *Local brokers* constitute the clients' access point to the middleware and are part of the communication library loaded into the clients; they are not represented in the graph, but only used for implementation issues. A local broker is connected to at most one border broker. *Border brokers* form the boundary of the distributed communication middleware and maintain connections to local brokers, i.e., the clients. *Inner brokers* are connected to other inner or border brokers and do not maintain any connections to clients.

Possible routing strategies. Each broker maintains a routing table that determines in which directions a notification is forwarded. Each table entry is a pair (F, L) containing a filter and the link from which it was received, denoting that a matching subscription is to be forwarded along L; this is a widely used data structure [1, 2]. The routing decision is assumed to be an atomic operation so that the end-to-end sender FIFO characteristic holds. The routing tables are maintained to correspond to the available information about active consumers and their subscriptions. Each broker forwards these information according to the routing algorithm used.

The basic form of routing is *simple routing*: active filters are simply added to the routing table according to the link they belong to. Although improvements to this strategy (e.g., *covering* and *merging*) are available in REBECA, for the sake of simplicity we assume simple routing throughout this paper.

Mobile REBECA. In a pub/sub system implemented using REBECA, it is also possible to support applications running on small devices like smart cards, sensors, smart labels that are connected to the immobile infrastructure through a wireless link. Generally, these small devices run some sort of application that should participate in the event system, i.e., produce and consume notifications. Two types of devices are distinguished: (i) devices powerful enough to host a local broker, then it is no problem to connect the device to the infrastructure. As long as it is possible to establish and maintain a (TCP) connection between the device and some event broker in the REBECA backbone; and (ii) devices not powerful enough to host a local broker. In this case it is possible to provide a *proxy* with the same interface as that of the event broker through which the device transparently connects to a virtual counterpart running at the border broker to which it is connected (see Fig. 3).

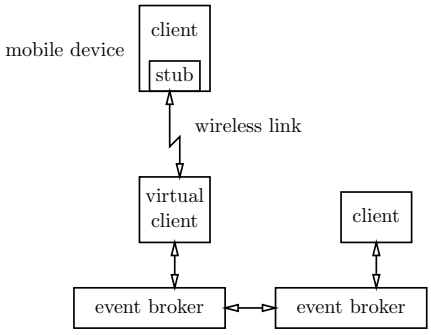

Figure 3: An architecture for mobile REBECA.

The translation between the operations is performed over the wireless link using some existing technology (WLAN, IrDA, Bluetooth) and hardware devices. We assume that this hardware allows both the client and the virtual counterpart to be "connection aware", i.e., they have means to check whether there is or is not an existing connection, and if there is no connection, whether some border broker is in reachable distance. In the latter case, it is possible to establish a connection to that broker, startup a virtual client and participate in the pub/sub system.

3 Transparent Implementation of Extended Logical Mobility

In this section we describe how extended logical mobility can be added to REBECA with physical and logical mobility extensions using a clean layering approach (i.e., without having to change the internals of the underlying routing framework). As we will argue later, this allows to add the intended notion of "listening" to a pub/sub system implementing logical and physical mobility.

3.1 Main Idea

The main idea of implementing extended logical mobility is to replicate the virtual client at several places in order to set up notification flow for location-dependent information well before the client reaches a particular broker. Generally speaking, a virtual client is started on every broker to which the client may connect in the "near" future. As we cannot predict where a client will move to in the future we do so to cope with the inherent *uncertainty* of movement in mobile systems. This means, a client (and its associated virtual clients) is always surrounded by a set of identical virtual clients that are ready to serve the mobile device as soon as it connects to that particular broker. Of course, this set of virtual clients must change when the client moves. New virtual clients must be created at those brokers which come into "range" of the client. Old virtual clients (which leave the range) must be garbage collected.

At any time, only at most one of the virtual clients is in fact associated with (and connected to) the "real" client running on the mobile device. All other clients should mimic the behavior of the real client, i.e., they should subscribe and unsubscibe to the same location-dependent filters as the client. However, only the virtual client which is in fact connected to the mobile device publishes notifications and delivers notifications to the mobile device. Unconnected virtual clients do not publish notifications at all. However, they do buffer all delivered notifications according to some application-specific buffering policy. Note that, due to the nature of location-dependent subscriptions, they only receive and buffer information related to their own location (i.e., those subscriptions a client arriving at that location would have). Also note that the replication strategy need not be applied to any subscription which is not location-dependent. Those subscriptions are catered for by the standard relocation algorithm for physical mobility as proposed in [8]. Consequently, subscriptions for location-dependent information at the current broker are not touched by the algorithm proposed here, for those subscriptions are covered by standard logical mobility as proposed in [5]. This is reasonable in order to keep different aspects of mobility cleanly separated.

3.2 Algorithm

We have to assume that the mobile client obeys some movement restriction. We formalize this restriction as a *movement graph* with brokers as vertices. In this graph, an edge exists between broker b_1 and b_2 if and only if the client may connect to b_2 after disconnecting from b_1. In many system settings it is feasible to define such a movement graph. For example, if base stations in a GSM network contain a local broker each, the neighborhood relationship between them defines the movement graph for the system (we will discuss the effects of different movement graphs later).

Within the algorithm, the movement graph is formalized as a function *nlb* : $B \rightarrow 2^B$, where B denotes the set of all local brokers and 2^B denotes the powerset of B, i.e., the set of all subsets of B. The name "nlb" stands for "next local broker". For some $b \in B$, *nlb*(b) yields the set of all brokers which are reachable from b using exactly one edge in the movement graph. This set can be seen as the "neighborhood" of b (excluding b itself).

The algorithm is implemented within a horizontal layer between virtual clients and local border brokers (see Figure 4). With every broker process we associate an additional *replicator* process. This process offers the same interface as the actual broker. Instead of invoking methods of the broker interface itself, virtual clients interact with the replicator process of the broker.

The replicator process is transparent to virtual clients. The operations *publish*, *subscribe*, and *unsubscribe* are passed (downwards) through to the local border broker. Similarly, invocations of *notify* are passed (upwards) to the virtual client. Additionally, the replicator process

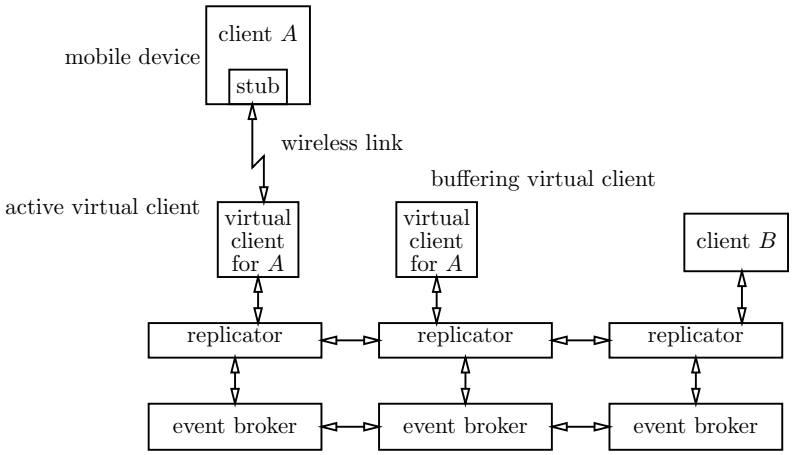

Figure 4: Transparent implementation of extended logical mobility on top of a mobile pub/sub system.

can interact autonomously with the replicator processes at neighboring event brokers through direct TCP connections (see Figure 4). The replicator knows the *nlb* function and is aware of which virtual client is currently connected to a mobile device and which is not.

3.2.1 Client Setup

Whenever an application is started on a mobile device which is connected to some local border broker b, a virtual client is started for the application at the broker. To implement extended logical mobility, the replicator inspects the set *nlb*(b) and instructs all border brokers in this set to create the same virtual client together with the same set of location-dependent subscriptions. In this way, whenever the client moves to a neighboring broker, a virtual client will already exist there and provide an initialized stream of (buffered) notifications relevant for the new location.

3.2.2 Client Operation

Operations like *publish* and *notify* are handled transparently by the replicator layer. However, if the client subscribes or unsubscribes to a particular location-dependent filter, the replicator must also instruct all brokers in *nlb*(b) to (un)subscribe to the same filter.

3.2.3 Client Handover

Whenever the mobile device on which the client runs leaves the range of broker b_1, the virtual client at b_1 notices this and starts to buffer notifications instead of delivering them to the client. When the device enters the range of broker b_2, the virtual client at b_2 notices this and starts to deliver notifications to the client whenever they arrive. The delivery starts with a replay of all buffered notifications with information relevant to the new location. The replicator at b_2 must now inspect the following two sets

$$\textit{oldset} = \textit{nlb}(b_1) \qquad \textit{newset} = \textit{nlb}(b_2)$$

which resemble the neighborhood of brokers b_1 and b_2 respectively. Now, the replicator must ensure that new virtual clients for the application must be created on all brokers in *newset* $\setminus$ *oldset* and the virtual clients can be deleted on all brokers in *oldset* $\setminus$ *newset*.

3.2.4 Client Removal

Whenever an application is turned off, the system deletes the associated virtual client on the local broker b. Additionally, the replicator at b has to garbage collect all virtual clients which are running on the neighboring brokers, i.e., all brokers in the set *nlb*(b).

4 Open Issues and Research Agenda

Mobility support in existing pub/sub systems is far from perfect. Taking up some open points from the previous section, we conclude the paper by stating some of the new research questions which we consider worthwhile pursuing.

Scalability and dynamic environments

Scalability of efficient content-based filtering algorithms has been investigated for the standard, i.e., non-mobile, environments [16]. Pervasive environments with their complex issues of different forms of mobility pose greater challenges both in the number of clients to support as well as in the dynamics of their behavior. How scalable are implementations of logical and physical mobility? How can the efficiency of the underlying routing framework best be exploited?

Dealing with further uncertainty in user movement

The job of the replicator layer is to ensure that a virtual client is running on every local broker which is "around" the current physical location of the client. This ensures that when a client moves to a new local broker, he can immediately start to receive a stream of notifications without having to subscribe to his interests anew. With an application-dependent buffering scheme it is also possible to cater for longer periods of disconnection, as long as the next broker to which the client connects is one of the neighboring brokers on which a virtual client is already running.

One reason for disconnecting from the network is to save power by shutting down the device. Combined with client movement, this implies that a client may always "pop up" at *any place* in the broker network, i.e., places which are not covered by *nlb* and hence where no virtual client is running. Basically it would be possible to change the definition of *nlb* to cover (almost) every broker. However, this would mean that a virtual client is running (almost) everywhere in the system. In this case the scheme would degenerate to flooding, a very unpleasant situation.

In practice it will be a challenge to devise instances of *nlb* that are as "large" as necessary (to cater for a lot of different forms of user movement) but are as "small" as possible (to not waste too much bandwidth). In is sensible to offer an "exception mode," meaning that if a client connects to a local broker where no appropriate virtual client is running, it may be possible to start such a virtual client on the fly and retrieve buffered notifications from some other virtual client of the application. In general, it will probably be acceptable for users to expect some form of degraded service after long periods of disconnection.

Embedding event histories

Event histories can provide a certain form of buffering of notifications at virtual clients. Event histories can also be introduced at well-chosen points within the implementation of logical mobility. In general, such a history is specified by a garbage collection policy.

Garbage collection can be time-based, history-based or semantic-based. In a time-based scheme, all notifications published more than t seconds ago are deleted from the buffer. In a history-based scheme, the buffer always keeps the last n notifications. Both schemes can be combined. In semantic-based scheme new events can nullify old events (similar to [17]). In general, the buffering scheme is application dependent. What are the best buffering schemes for certain applications and where should buffering take place exactly?

If virtual clients buffer notifications individually, they may consume memory redundantly by keeping the same data. A shared buffer at the border broker can be used and virtual clients can keep only the digest (e.g., IDs or hash) of the events. Then the digest can be used to fetch events from this buffer and the events can be garbage collected according to a chosen policy when none of the virtual clients need them. Such schemes that reduce the resources at virtual clients need more exploration.

From location-awareness to context-awareness

Another important building block for the use in pervasive environments is to generalize the concept of location-dependent subscriptions to "state-dependent" subscriptions, opening the whole area of context-awareness [18] to the domain of pub/sub middleware systems. How can systems implement or make use of such dynamic filters, which depend on a function of the local state of the client (not only its current location)?

Acknowledgments

We thank Gero Mühl for his cooperation in the REBECA project and Oliver Kasten for many helpful discussions on the topic of location-dependent subscriptions. Work by Ludger Fiege and Felix Gärtner was supported by Deutsche Forschungsgemeinschaft (DFG) as part of PhD program "Enabling Technologies for Electronic Commerce" and the Emmy Noether program, respectively.

References

[1] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[2] G. Cugola, E. Di Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 27(9), 2001.

[3] Gero Mühl. *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, Darmstadt University of Technology, 2002.

[4] Amy L. Murphy, Gian Pietro Picco, and Gruia-Catalin Roman. LIME: A Middleware for Physical and Logical Mobility. In Forouzan Golshani, Partha Dasgupta, and Wei Zhao, editors, *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21)*, pages 524–533, May 2001.

[5] Ludger Fiege, Felix C. Gärtner, Oliver Kasten, and Andreas Zeidler. Supporting mobility in content-based publish/subscribe middleware. In *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distribued Processing (Middleware 2003)*, 2003.

[6] Ludger Fiege and Gero Mühl. Rebeca Event-Based Electronic Commerce Architecture, 2000. `http://www.gkec.informatik.tu-darmstadt.de/rebeca`

[7] Ludger Fiege, Gero Mühl, and Felix C. Gärtner. A modular approach to build structured event-based systems. In *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC'02)*, pages 385–392, Madrid, Spain, 2002. ACM Press.

[8] Andreas Zeidler and Ludger Fiege. Mobility support with REBECA. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS) Workshop on Mobile Computing Middleware (to appear)*, 2003.

[9] Yongqiang Huang and Hector Garcia-Molina. Publish/subscribe in a mobile environment. In *Proceedings of the 2nd ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE01)*, Santa Barbara, CA, May 2001.

[10] Peter Sutton, Rhys Arkins, and Bill Segall. Supporting disconnectedness – transparent information delivery for mobile and invisible computing. In *First International Symposium on Cluster Computing and the Grid*, pages 277–287, Brisbane, Australia, May 2001. IEEE/ACM.

[11] Jean Bacon, Ken Moody, John Bates, Richard Hayton, Chaoying Ma, Andrew McNeil, Oliver Seidel, and Mark Spiteri. Generic support for distributed applications. *IEEE Computer*, 33(3):68–76, 2000.

[12] Mauro Caporuscio, Paola Inverardi, and Patrizio Pelliccione. Formal analysis of clients mobility in the Siena publish/subscribe middleware. Technical report, Department of Computer Science, University of L'Aquila, October 2002.

[13] R. Meier and V. Cahill. STEAM: Event-based middleware for wireless ad hoc networks. In *Proceedings of the International Workshop on Distributed Event-Based Systems (ICDCS/DEBS'02)*, pages 639–644, 2002.

[14] N. Carriero and D. Gelernter. Linda in context. *Communication of the ACM*, 32(4):444–458, April 1989.

[15] Gero Mühl. Generic constraints for content-based publish/subscribe systems. In C. Batini, F. Giunchiglia, P. Giorgini, and M. Mecella, editors, *Proceedings of the 6th International Conference on Cooperative Information Systems (CoopIS '01)*, volume 2172 of *LNCS*, pages 211–225, Trento, Italy, 2001. Springer-Verlag.

[16] Gero Mühl, Ludger Fiege, Felix C. Gärtner, and Alejandro P. Buchmann. Evaluating advanced routing algorithms for content-based publish/subscribe systems. In A. Boukerche and S. Majumdar, editors, *The Tenth IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2002)*, Fort Worth, TX, USA, October 2002. IEEE Press.

[17] J. Orlando, L. Rodrigues, and R. Oliveira. Semantically reliable multicast protocols. In *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS 2000)*, October 2000.

[18] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, USA, 1994.