Supporting Mobility in Content-Based Publish/Subscribe Middleware

Ludger Fiege[1], Felix C. Gärtner[2], Oliver Kasten[3], and Andreas Zeidler[1]

[1] Darmstadt University of Technology (TUD), Department of Computer Science, Databases and Distributed System Group, D-64283 Darmstadt, Germany, {fiege,az}@dvs1.informatik.tu-darmstadt.de
[2] Swiss Federal Institute of Technology (EPFL), School of Computer and Communication Sciences, Distributed Programming Laboratory, CH-1015 Lausanne, Switzerland, fcg@acm.org
[3] Swiss Federal Institute of Technology (ETH Zurich), Department of Computer Science, Distributed Systems Group, CH-8092 Zurich, Switzerland, oliver.kasten@inf.ethz.ch

Abstract. Publish/subscribe (pub/sub) is considered a valuable middleware architecture that proliferates loose coupling and leverages reconfigurability and evolution. Up to now, existing pub/sub middleware was optimized for static systems where users as well as the underlying system structure were rather fixed. We study the question whether existing pub/sub middleware can be extended to support *mobile* and *location-dependent applications*. We first analyze the requirements of such applications and distinguish two orthogonal forms of mobility: the system-centric physical mobility and an application-centric logical mobility (where users are aware that they are changing location). We introduce *location-dependent subscriptions* as a suitable means to exploit the power of the event-based paradigm in mobile applications. Briefly spoken, location-dependency refines a subscription to accept only events related to a mobile user's current location. Implementations for both forms of mobility are presented within the content-based pub/sub middleware Rebeca, drawing from its refined routing capabilities (namely, covering and merging).

1 Introduction

Location-based services. The emergence of mobile computing has opened up a whole new field of services provided for the benefit of the mobile user. Many such services can exploit the fact that the mobile device is aware of its current location. For example, car navigation systems use knowledge about current and past locations to aid drivers find their way through unknown cities. Location information can even be combined with other sources of data, e.g., the weather report, information on traffic jams or free parking spaces. In such cases, the system can propose routes that avoid places where traffic is high or weather conditions are unpleasant, or can direct the driver to the nearest free parking space. All these are examples for *location-based services*.

Publish/subscribe systems. A convenient way to construct location-based services is to build them using event infrastructures, such as those provided by *publish/subscribe systems*. Here, producers and consumers are enabled to exchange information based on message type or content rather than particular destination identifiers or addresses. This *loose coupling* of producers and consumers is the premier advantage of pub/sub systems, which facilitates mobile communication. Producers are relieved from managing interested consumers, and vice versa. In this paper we study how to exploit these advantages and what extensions are eligible in the context of mobile services.

Supporting mobility in pub/sub middleware. We argue that support for mobility should be an issue of the pub/sub middleware itself and not be delegated to the application layer. Three kinds of application scenarios have to be supported: i) existing applications in a static environment, ii) existing applications in a mobile environment, iii) mobility-aware applications. Since pub/sub systems and applications have been deployed very successfully, extending existing systems and models is preferred to creating new "mobile" middleware from scratch in order to facilitate the integration of the first two scenarios. As a consequence, the middleware must transparently handle some of the new mobility issues. This allows existing event-based applications to directly interact with and even to be deployed as mobile applications. On the other hand, the third scenario requires the middleware to support a (semi-)automated handling of location changes. If no such support is available, mobility is actually controlled by the application and not by the movement of the client.

We provide support for two different and orthogonal types of mobility. The first type of mobility is called *physical* mobility, where clients may temporarily disconnect from the pub/sub system (due to power-saving requirements or the network characteristics). This means that applications are not necessarily aware of the fact that the client is moving, allowing existing applications to be transferred to mobile environments. The second type of mobility is called *logical mobility*, where clients remains attached to the their broker and have an application-level notion of location, which is described by *location-dependent subscriptions* introduced in this paper. As an example, consider a car looking for a free parking space in the street it is currently driving along. In this situation it may subscribe to "New free parking space on Rebeca Drive". However, if Rebeca Drive is a very long street, the same driver will also receive notifications about free parking spaces very far down the road (or behind him), which are impossible to reach in good time. What the user would like to do is to specify a subscription such that he receives all notifications about "vacancies in the vicinity of his current location". We call these subscriptions *location-dependent*.

Related work. Work on middleware for mobile computing usually concentrated on classical synchronous middleware like CORBA, see [4] for a survey. Only recently, position papers have stated that pub/sub systems have an enormous potential to better accommodate the needs of large mobile communities [16, 6]. Research in pub/sub systems has mainly focused on *static* systems, where

clients do not move and the pub/sub infrastructure remains relatively stable throughout the system's lifetime, e.g., Elvin [23], Gryphon [15], REBECA [12], and Siena [5]. If present at all, mobility support is a concern of the application layer. Applications detect the need to change a subscription and have to react explicitly and manually to this detection.

Huang and Garcia-Molina [13, 14] provide a good overview of possible options for supporting mobility in pub/sub systems. They describe algorithms for a "new" middleware system tailored and optimized to mobile and ad hoc networks, not so much an extension of an existing system. CEA [1] and JEDI [7], too, address problems of mobility. JEDI uses explicit moveIn and moveOut operations to relocate clients. Hence, mobility is controlled by the application, which is not transparent and even unrealistic since clients usually only can react *after* having been moved. The mobility extensions of SIENA [3] are very similar. Explicit sign-offs are required and interim notifications stored during disconnectedness are directly forwarded to a new location upon request. Cugola et al. [6] proposes a leader election and group management protocol for dynamic dispatching trees to dynamically adapt the internals of the JEDI event system, their implementation model is based on multicast and it groups identical subscribers. An extension for Elvin allows for disconnectedness using a central caching proxy [24], which is a potential performance bottleneck. Jacobsen [16] presents some very interesting ideas on location-based services and the possible expressiveness of subscription languages. STEAM [18] is an event service designed for wireless ad hoc networks. Subscribers consume only events produced by geographically close-by publishers. It relies on proximity-based group communication.

Outline. This paper is structured as follows: We provide some basic background and terminology on content-based pub/sub and the REBECA system in Section 2. We then discuss in more detail the issues involved when supporting mobility using existing content-based pub/sub middleware in Section 3. We present a solution for physical mobility in Section 4 and a solution for logical mobility in Section 5. Section 6 concludes the paper.

2 Content-Based Publish/Subscribe

The following gives an introduction to publish/subscribe systems and the system model we used as basis for the proposed mobility support. It is based on the REBECA notification service [11, 20].

2.1 Publish/Subscribe Systems

Processes in pub/sub systems (also known as *event-based systems* [12]) are clients of an underlying notification service and can act both as producers and consumers of messages, called event notifications or notifications for short. The communication interface to the system consists of four primitives only: *pub*, *sub*, *unsub*, and *notify*. The latter is a function provided by consumers that the

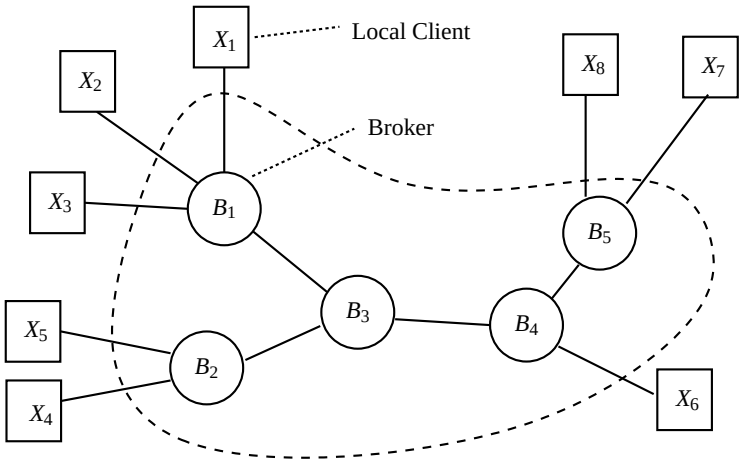

Fig. 1. The router network of REBECA.

event system calls to deliver notifications. A *notification* is a message that reifies and describes an occurred event. Notifications are injected into the event system rather than being published towards a specific receiver. They are conveyed by the underlying notification service to those consumers that have registered a matching subscription (*sub*). Subscriptions describe the kind of notifications consumers are interested in. In some systems producers are required to issue advertisements to describe the notifications they are about to publish.

The expressiveness of the notification service is determined by its language used for specifying subscriptions and the data model of the transmitted notifications. Subject- and type-based addressing exists [22, 2, 8], but the most flexible scheme is offered by content-based filtering [19]. Filters are boolean functions on the entire content of a notification and a common way to implement subscriptions. Together with the typically used name/value-pairs data model, subscriptions look like: $(\textit{service} =$ "parking"$), (\textit{location} =$ "100 Rebeca Drive"$), (\textit{cost} <$ "3 EURO"$), (\textit{car-type} \geq$ "compact"$)$. The Java Message Service (JMS), for example, uses a combination of subjects and content-based addressing.

The notification service for our scenario is distributed, of course, 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. 1). The graph consists of brokers and clients. The edges are point-to-point, FIFO order communication links, e.g., TCP connections, that are error-free, a common assumption that can be relieved later. This model simplifies the presentation and opens up implementation-dependent options, like using Multicast, to improve communication performance. 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.

The individual processes are assumed to have local real-time clocks that are synchronized using a standard protocol like NTP. While we postulate that there is no upper bound on the message delivery delay, we assume that the delays satisfy some probability distribution so that an expected delivery time can be computed statistically.

2.2 Content-Based Routing

Each broker maintains a routing table that determines the decision 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. 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 simplest form of routing is *simple routing*: active filters are simply added to the routing tables with the link they originated from. Obviously, this is not optimal with respect to routing table sizes, which grow with the number of subscriptions. A first improvement is to check and combine filters that are equal. More generally, the *covering* routing strategy [5] tests whether a filter F_1 accepts a superset of notifications of a second filter F_2, and in this case replaces all occurrences of F_2 assigned to the same link in the routing table, significantly decreasing the table size. In a second step, if no cover can be found in a given set of filters, *merging* can be used to create new filters that are covers of existing ones [19]. Only the resulting merged filter is forwarded to neighbor brokers, where it covers and replaces the base filters.

3 Publish/Subscribe Systems and Mobility

In this section we analyze and discuss the basic issues involved when adding mobility support to a pub/sub infrastructure. We identify and define two orthogonal forms of mobility (physical and logical mobility) and discuss the requirements of a system supporting both types of mobility.

3.1 Mobility Issues in Publish/Subscribe Middleware

Mobile clients have many characteristics, among them the need to disconnect from the network for different reasons. Be it for geographical, administrative, or power saving reasons, being connected to the same broker all the time is no longer

possible. Hence, we have to take into account that clients will disconnect from their border broker once in a while. The middleware has to deal with moving clients and the possibility that a disconnected client reconnects at the same or a different broker later.

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. The resulting location transparency is necessary to make existing applications mobile, e.g., stock quote monitoring seamlessly transferred from PCs to PDAs.

On the other hand, future applications do not want complete transparency, but rely on awareness of mobility. More specifically, mobility support should blend out unwanted phenomena, like disconnectedness, and enforce wanted behavior, like the location awareness in location-based services. Consequently, extending the interface of the pub/sub middleware to facilitate location awareness is a promising open issue, since most existing work concentrated on the transparency only.

When roaming, clients change (at least some portion of) the context they are operating in, and they might want to react to these changes, e.g., to adapt their subscriptions. However, an appropriate infrastructure support has to relieve the application from having to react "manually" to all changes. The middleware should rather offer an automated adaptation to context changes, i.e., facilitating location dependency. This leads to a different notion of mobility and we distinguish:

- *Physical mobility*: A client that is physically mobile disconnects for certain periods of time and has different border brokers along its itinerary through the infrastructure. The main concern of physical mobility is *location transparency*.
- *Logical mobility*: A client that is logically mobile is aware of its location changes. In order to relieve the client from adapting *manually* to new locations, the main concern of logical mobility is *automated* location awareness within the pub/sub middleware.

Physical and logical mobility are two orthogonal aspects of mobility. Since the physical layout of a pub/sub system is usually fixed and its layout does usually not correspond to geographical realities, it seems reasonable to separate the two notions of mobility. In this paper, we assume logical mobility to be a refinement of physical mobility in that a client remains connected to the same broker when roaming logically. The two notions have different quality of service requirements and therefore different solutions are developed to match both.

3.2 Physical Mobility

Physical mobility is similar to what in the area of mobile computing is called *terminal mobility* or *roaming*. A client accesses the system through a certain

number of *access points* (GSM base stations, WLAN access points, or border brokers). When moving physically, the client may get out of reach of one access point and move into the reach of a second access point which are not necessarily overlapping. In general we cannot expect to have seamless access to the broker network but more a sequence of phases of connectedness, e.g., on the daily route between home and office. In this setting we analyze the quality of service requirements from the viewpoint of roaming clients:

- Interface. Obviously, the interface to the pub/sub system must not change as legacy applications are not aware of mobility.
- Completeness. Despite intermittent disconnects, the pub/sub middleware delivers all notifications for a client eventually. This is the core requirement for transparency.
- Ordering. Sender FIFO ordering was guaranteed in Section 2 and it is an eligible feature in the mobile case, too.
- Responsiveness. The delay of relocating a roaming client should be minimal to maximize the responsiveness of the system. This has to be taken into account when designing a relocation protocol.

Possible Solutions. One solution would be to rely on Mobile IP [17] for connecting clients to border brokers, hiding physically mobility in the network layer. The drawback, however, is that the communication is also hidden from the pub/sub middleware, which is then not able to draw from any notification delivery localities or routing optimizations, thereby possibly violating the requirement of responsiveness. Such an approach might only be feasible if the physical and logical layout of a given system is completely orthogonal.

A different, naïve solution to implement physical mobility would be to use sequences of *sub-unsub-sub* calls to register a client at a new broker. When a client moves from border broker B_1 to B_2, it simply unsubscribes at B_1 and (re-) subscribes at B_2, without any support in the middleware. But a client may not detect leaving the range of a broker and is in this case not able to unsubscribe at its old location. Even more severely, during its time of disconnectedness, the client might miss several notifications or get duplicates, even if notifications are flooded in the network and the location change is instantaneous. This problem is depicted in Figure 2. Hence, this solution is not complete and we outline an algorithm in Section 4 that takes into account all requirements stated above. The complete algorithm is detailed in [25].

3.3 Logical Mobility

While physical mobility is a rather technical issue invisible to the application, logical mobility involves location awareness. An example for logical mobility is when clients move around a house or building that is served by only one border broker. In this case, the user might be interested to receive just those notifications that refer to the room he is currently located in. Note that a client can be both logically and physically mobile at the same time.

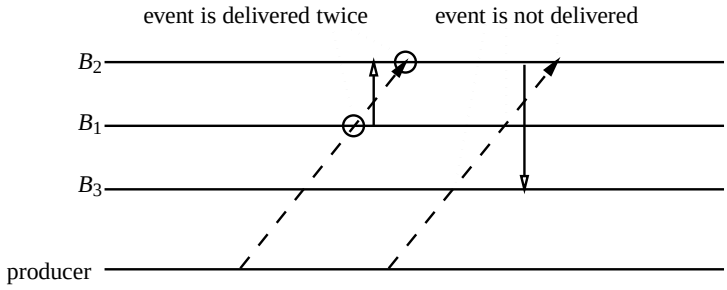

Fig. 2. Missing notifications in a flooding scenario.

A logically mobile client moving from one location to another, e.g., from one room to the other in a company building, will expect a frictionless change of location explicitly without a notable setup time after having changed from its own office to the conference room next door. The adaptation of some location-dependent subscription 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 moving clients.

Location-Dependent Filters. A pub/sub system that offers location-dependent filters has the same interface as a regular pub/sub system (i.e., it offers the *pub*, *sub*, *unsub*, *notify* primitives). However, in specifying subscription filters for name/value pairs referring to "*location*" it supports a new primitive to specify things like "all notifications where the attribute *location* equals my current location". More precisely, we postulate a specific marker *myloc* that can be used in a subscription. The marker stands for a specific set of locations that depend on the current location of the client. For example, a client could issue a subscription for all free parking spaces in the vicinity of his current location as follows: (*service* = "parking"), (*location* ∈ *myloc*), (*car-type* ≥ "compact").

The set of locations associated with the marker is taken from a particular range L of locations. This set is application dependent and can, for instance, contain all the different rooms of a building, all the streets of a town, or all the geographical coordinates given by a GPS system up to a certain granularity. Given a notification with the attribute *location*, the subscription (*location* ∈ *myloc*) will evaluate to true for a particular client at location y if and only if $x \in myloc(y)$ where $myloc(y)$ is the specific set of locations associated with y. In this case we say that the notification matches the location-dependent filter.

The simplest form of $myloc(y)$ is simply the set $\{y\}$. In this case a notification matches the subscription if $x = y$. But in the car example, the car driver looking for a parking space might want to specify:

(*location* = "at most two blocks away from *myloc*")

In this case, *myloc* corresponds to all elements of L that satisfy this requirement.

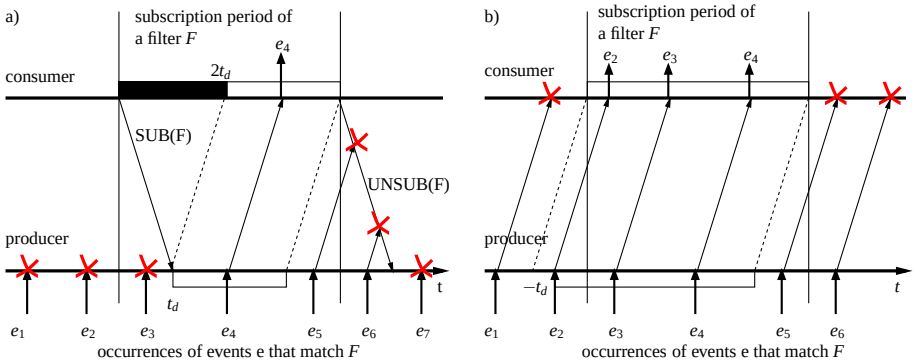

Fig. 3. Blackout period after subscribing with simple routing a) and flooding with client-side filtering b).

A tentative but incomplete solution for logical mobility. While location-dependent filters are not directly supported by current pub/sub middleware, one might argue that it is not very difficult to emulate them on top of currently available systems in this case. The idea would be to build a wrapper around an existing system that follows the location changes of the users and transparently unsubscribes to the old location and subscribes to the new one when the user moves. However, depending on the internal routing strategy of the event system, it may lead to unexpected results. The routing strategies deployed in many existing content-based event systems such as Siena [5], Elvin [23], and REBECA [10] lead to blackout periods where no notifications are delivered. The problem is that it usually takes an unnegligible time delay to process a new subscription. After subscribing to a filter, it takes some time t_d until the subscription is propagated to a potential source. Then it takes at least another t_d time until a notification reaches the subscriber. This phenomenon is depicted in Figure 3a. (Note that the delay t_d may be different for different notification sources and may change over time.) If the client remains at any new location less than $2t_d$ time, then the subscriber will "starve", i.e., it will receive little or no notifications.

An intuitive but inefficient solution. Another basic solution that can be immediately built using existing technology is again based on flooding. The local broker can then decide to deliver a notification to a client depending on the client's current location (see Figure 3b). Obviously, flooding prevents the blackout periods, which were present in the previous solution, but it should be equally clear that flooding is a very expensive routing strategy especially for large pub/sub systems [21].

Quality of service of logical mobility. Interestingly, while flooding is very expensive and therefore not desirable, it comes very close to the quality of service that we would like to achieve for logical mobility, namely to the notion of being subscribed to "everything, everywhere, all the time". The problem is that

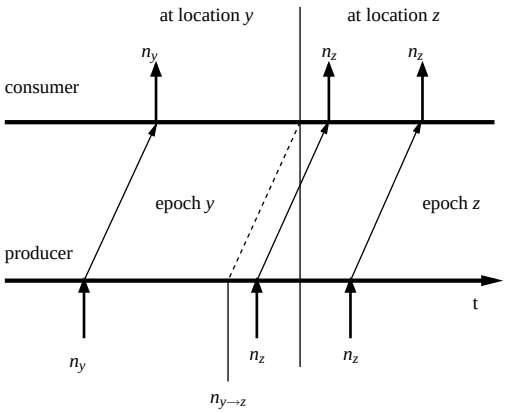

Fig. 4. Defining the quality of service for logical mobility using virtual notifications $n_{y\to z}$ that arrives at the consumer just at the time of the location change from y to z.

it is hard to precisely define the behavior of flooding without reverting to some unpleasantly theoretical constructions of operational semantics.

With logical mobility there is, however, no danger of receiving a notification twice because the consumer remains attached to the same "delivery path". The quality of service we require for logical mobility therefore is simply stated as follows: On change of location from x to y, all notifications should be delivered to the consumer "as if" flooding were used as underlying routing strategy. This statement is made a little more concrete in Figure 4 where the sequence of notifications generated by any consumer is divided into epochs that correspond to when the notification actually arrives at the consumer (the epoch borders between location y and z are drawn as a virtual notification $n_{y\to z}$). We require that all notifications matching the current location-dependent subscription from every such epoch must be delivered. Intuitively, the epochs define the semantics of flooding.

4 Notification Delivery with Roaming Clients

In this section we sketch an algorithm for extending standard Rebeca brokers to cope with roaming mobile clients, maintaining their subscriptions as well as guaranteeing the required quality of service described in the previous section.

Apart from guaranteeing uninterrupted notification delivery together with transparency of mobility, our algorithm also guarantees that the "old" border broker (i.e., the broker to which the roaming client was formaly attached) will eventually receive an equivalent to an explicit *sign-off* from the client, so that it can garbage collect all resources allocated to this specific client. In this process the algorithm also guarantees that any routing path to the old location related to the client will be deleted.

4.1 Main Idea

The basic idea is to maintain a "virtual counterpart" of a roaming client at the last known location until some broker at a new location is claiming responsibility and then merge "actual" and "virtual" client in such a way that no notification is lost or delivered twice.

In the light of the quality of service requirements from the previous section, a realistic choice to devise such an algorithm has to employ the following features:

- Reactive model. The relocation algorithm has to be *reactive*, i.e., no explicit MoveOut or un-subscribe at the old location should be needed.
- Distribution. To enhance responsiveness, the algorithm adheres to strict locality; the approach is completely distributed, buffers notifications wherever necessary, and restricts reconfiguration of the broker network to the smallest possible subgraph.
- Completeness. By introducing distributed buffers within the border brokers the algorithm guarantees completeness within the boundaries of time and/or space limitations of buffering approaches.
- Pub/sub adherence. All communication related to the relocation protocol is done within and based on the broker network. Other approaches using some sort of direct, out-of-band communication between old and new broker might introduce problems of ordering, duplicate detection, or even message loss. This can be avoided by only using communication mechanisms offered by the pub/sub middleware.

Example. We illustrate the relocation process using the simple example scenario on the left of Fig. 5 for a single producer and consumer; the generalization for multiple producers is indicated on the right of Fig. 5 and a more detailed description can be found in [25].

Assume client C is moving from the location at broker B_6 to another location at broker B_1. This refers to step 1 in the figure. After the client has detected the change of location and broker it automatically re-issues a subscription together with the last received sequence number for this subscription (e.g., $(C, F, 123)$, with 123 being the last known sequence number annotated by the former border broker; step 3). Broker B_1 will detect that this client has moved and must be relocated. Note that neither client nor broker need to have any knowledge about the old location B_6. Broker B_1 then starts the relocation process by sending a special message to its neigboring brokers.

The goal of the relocation process is to divert the delivery paths from producer P to C to the new location. During this process, the brokers propagate the subscription through B_2 and B_3 to broker B_4. Here the old and new path from producer P to client C meet (dotted and dashed line, respectively). Broker B_4 is aware of this by inspecting its routing table and its list of received advertisements, and comparing it to the subscription received. As B_4 has an old entry for this subscription, B_4 sends a fetch request $(C, F, 123, B_4)$ along the old path to B_6 and already starts routing all newly received notifications from P along the *new* path.

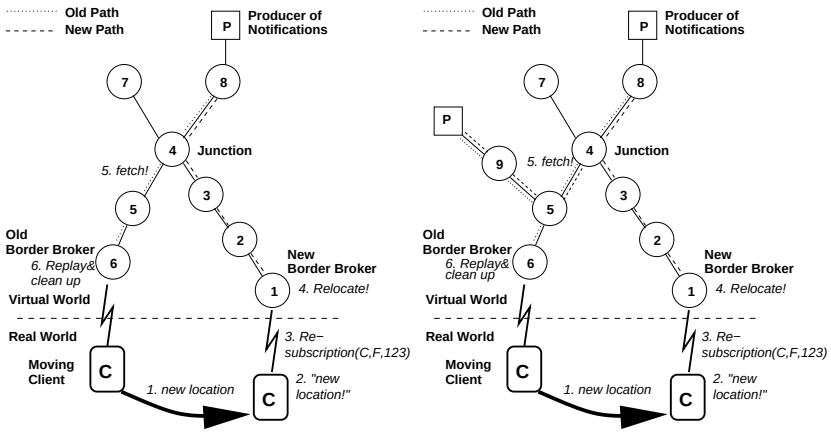

Fig. 5. A moving client scenario with one producer (left) and more than one producer (right).

When receiving the fetch request along the path to the old broker B_6, all brokers along this path update their routing tables such that they are pointing into the direction of B_4. B_6 as last recipient replays all events buffered in the virtual counterpart of (C, F) beginning with the sequence number initially given by C to B_1 (here 124). The counterpart sends a message with a replay of all notifications received in the meantime along the path into the direction of B_4. As all intermediate brokers have already updated their routing tables, the replay eventually reaches B_1 via B_4 and is delivered to C. In the meantime, B_1 has buffered all notifications that have arrived for C and delivers the old messages from B_6 first before delivering the "new" messages from its own buffer to guarantee the correct delivery order.

Broker B_6 at the old location can garbage collect all resources formerly associated with C, and so can B_5, resulting in the new routing path between P and C as shown in Fig. 5.

4.2 Discussion

This example should give a feeling of how relocation and adaptation of the delivery paths is performed in a fully distributed fashion. Through the use of administrative control messages, and buffering and replay mechanisms the algoritm makes good use of the already builtin features of Rebeca. Covering and merging can be exploited, too, if the fetch request sent by B_4 is directed towards both matching advertisments and covering filters.

5 Location-Dependent Filters for Logical Mobility

We now describe the algorithmic solution to the scenario where clients are only logically mobile, i.e., they remain attached to a single border broker.

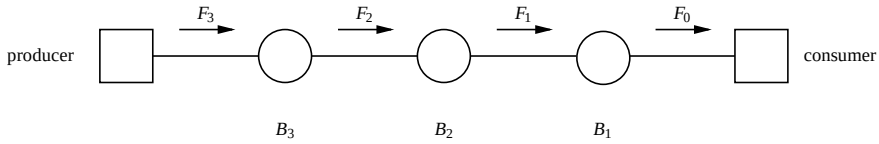

Fig. 6. Network setting for the example.

5.1 Main Idea

Consider an arbitrary routing path between a producer (publisher) and a consumer (subscriber). This path consists of a sequence of brokers $B_1, B_2, \ldots, B_{k-1}$, B_k where B_1 is the local broker of the consumer and B_k is the local broker of the producer (Figure 6 shows the setup for $k = 3$). Assume the consumer has issued a location-dependent subscription F. Using the "usual" content-based routing algorithms, the current value $\tilde{F}$ of F, which instantiates the marker variable with the current location, would permeate the network in such a way that the filters along the routing path allow a matching subscription published by the producer to reach the consumer. Formally, the filters $F_1, F_2, \ldots, F_k$ along the links between the brokers should maintain a set-inclusion property

$$F_k \supseteq F_{k-1} \supseteq \ldots \supseteq F_2 \supseteq F_1 \supseteq F_0 = \tilde{F}.$$

If F is the only active subscription in the network and if the subscription has permeated the network, the above formula can be simplified to

$$F_k = F_{k-1} = \ldots = F_2 = F_1 = F_0 = \tilde{F}.$$

Obviously, if for any new value $\tilde{F}$ of F a new subscription must flow through the network towards the producers, notifications published in the meantime might go unnoticed. The idea of the proposed scheme is to always have the local broker of the consumer do perfect client-side filtering (i.e., set $F_0 = \tilde{F}$), but to let possible future notifications reach brokers that are nearer to the consumer so that their delay to reach the consumer is lower once the consumer switches to a new location.

Let T denote the set of time values, which for simplicity we will assume to be the set of natural numbers $\mathbb{N}$. Let L denote the set of all consumer locations. Then we define a function $loc : T \rightarrow L$ that describes the movement of the consumer over time. For example, for a location set $L = \{a, b, c, d\}$ a possible value of loc is $\{(1, a), (2, b), (3, d), \ldots\}$ meaning that at time 1, the consumer's location is a, at time 2 it is b and so on.

We assume that loc is subject to some movement restrictions, which in effect define a maximum speed of movement for the consumer. We assume that such a restriction is given by a *movement graph* such as the one depicted in Figure 7. The graph formalizes which locations can be reached from which locations in one movement step of the consumer. One movement step has some application-defined correspondence to one time step.

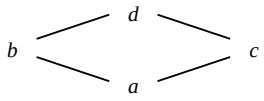

Fig. 7. Movement graph defining movement restrictions of a consumer.

Given the function *loc* and a movement graph, it is possible to define a function $ploc : L \times \mathbb{N} \rightarrow 2^L$ of possible (future) locations (the notation 2^L denotes the powerset of L, i.e., the set of all subsets of L). The function takes a current location x and a number of consumer steps $q \geq 0$ and returns the set of possible locations, which the consumer could be in starting from x after q steps in the movement graph.

Since a possible move of the consumer always is to remain at the same location, for all locations $x \in L$ and all $q \in \mathbb{N}$ we should require that

$$ploc(x, q) \subseteq ploc(x, q + 1). \tag{1}$$

Taking the example values from above, possible values for *ploc* are as follows:

$$ploc(a, 0) = \{a\} \qquad ploc(a, 1) = \{a, b, c\} \qquad ploc(a, 2) = \{a, b, c, d\}$$

Now, if the consumer is at location a, for example, every broker B_i along the path towards a producer should subscribe for $ploc(a, q)$ for some q, which is an increasing sequence of natural numbers depending on i and the network characteristics. If the time it takes for a broker to process a new subscription is in the order of the time a client remains at one particular location, then the individual filters F_i along the sample network setting in Figure 6 should be set as $F_i = ploc(a, i)$, e.g., $F_0 = ploc(a, 0) = \{a\}$, $F_1 = ploc(a, 1) = \{a, b, c\}$ and so on. This requirement should be maintained throughout location changes by the consumer. For example, whenever a consumer moves from an old location x to a new location y, the corresponding client node must declare the new location by sending a message to its broker B_1. This will cause B_1 to change the location-dependent part of filter F_0 for client-side filtering from the old to the new location. Broker B_1 updates its routing table appropriately.

In general, broker B_i sends a message with the new location to B_{i+1} instructing it to change F_i from $ploc(x, i)$ to $ploc(y, i)$ and consequently to update the routing table by removing certain locations and adding new locations. Removing and adding new locations corresponds to unsubscribing and subscribing to the corresponding filters. The normal REBECA administration messages can be used to do this. Note that Equation 1 guarantees the subset relationship, which should always hold on every path between producer and consumer.

5.2 Example

As an example, consider the value of *loc* where at time 1 the client is in location a, at time 2 at b and at time 3 at d in the movement graph depicted in Figure 7.

Table 1 gives the values of *ploc* for all locations and the first four time instances. For $t = 0$ the value of *ploc* is equal to the current location. For $t = 1$ it returns all locations reachable in one time step in the movement graph, etc.

Table 1. Values of $ploc(x, t)$ for the example setting.

t	$x = a$	$x = b$	$x = c$	$x = d$
0	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$
1	$\{a, b, c\}$	$\{a, b, d\}$	$\{a, c, d\}$	$\{b, c, d\}$
2	$\{a, b, c, d\}$	$\{a, b, c, d\}$	$\{a, b, c, d\}$	$\{a, b, c, d\}$
3	$\{a, b, c, d\}$	$\{a, b, c, d\}$	$\{a, b, c, d\}$	$\{a, b, c, d\}$

Now assume again the setting depicted in Figure 6. The values of Table 1 directly determine the filter settings for $F_0, \ldots, F_3$ as shown in Table 2. At time $t = 1$ the client moves to location b. This means that F_0 changes from $\{a\}$ to $\{b\}$ and that F_1 must unsubscribe to c and subscribe to d, yielding $F_1 = \{a, b, d\}$. At time $t = 2$ the client moves to d, causing F_0 to change to $\{d\}$ and F_1 to unsubscribe to a and subscribe to c. All other filters remain unchanged.

Table 2. Values of filters in example setting.

time t	F_3	F_2	F_1	F_0
0	$\{a, b, c, d\}$	$\{a, b, c, d\}$	$\{a, b, c\}$	$\{a\}$
1	$\{a, b, c, d\}$	$\{a, b, c, d\}$	$\{a, b, d\}$	$\{b\}$
2	$\{a, b, c, d\}$	$\{a, b, c, d\}$	$\{b, c, d\}$	$\{d\}$

The example nicely shows that the method does some sort of "restricted flooding", i.e, all notifications reach broker B_2 but from there the uncertainty is restricted and so is the flow of notifications forwarded by B_2. In fact, the method described above using the *ploc* function can be regarded as an abstraction of both "trivial" implementations discussed in Section 3 (i.e., both implementations are instantiations of our scheme), as we explain in the following section.

5.3 Adaptivity

The example setting above assumes that processing a new subscription by a broker takes about as long as a consumer stays at one particular location. Obviously, it will usually take much less time to process a subscription even if slow or wireless network connections are used (user movement will be in the order of seconds while network delay will be in the order of milliseconds). We now present a scheme that adapts the level of "buffering" in the network to the average movement time of the client. Our algorithm, which for lack of space is detailed in [9], satisfies this form of adaptivity.

Table 3. Values of $ploc(x,t)$ for trivial *sub*/*unsub* implementation (top) and flooding with client-side filtering (bottom).

$ploc(x,t)$ for global *sub*/*unsub*				
t	$x=a$	$x=b$	$x=c$	$x=d$
0	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$
1	$\{a,b,c\}$	$\{a,b,d\}$	$\{a,c,d\}$	$\{b,c,d\}$
2	$\{a,b,c\}$	$\{a,b,d\}$	$\{a,c,d\}$	$\{b,c,d\}$
3	$\{a,b,c\}$	$\{a,b,d\}$	$\{a,c,d\}$	$\{b,c,d\}$

$ploc(x,t)$ for flooding				
t	$x=a$	$x=b$	$x=c$	$x=d$
0	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$
1	$\{a,b,c,d\}$	$\{a,b,c,d\}$	$\{a,b,c,d\}$	$\{a,b,c,d\}$
2	$\{a,b,c,d\}$	$\{a,b,c,d\}$	$\{a,b,c,d\}$	$\{a,b,c,d\}$
3	$\{a,b,c,d\}$	$\{a,b,c,d\}$	$\{a,b,c,d\}$	$\{a,b,c,d\}$

In the following, we denote the average time a client remains at one location by Δ and the time it takes to process a sufficiently large batch of *sub*/*unsub* messages between brokers B_i and B_{i+1} by δ_i. If the client moves very slowly, meaning that the sum of all δ_i is still less than Δ, we would like the scheme to behave like the trivial *sub*/*unsub* solution. For the example setting from the previous section this would mean that *ploc* has values like in the top part of Table 3 (note that the algorithm always has to provide information for "the next" user location to maintain the semantics of flooding). On the other hand, if the client moves very fast and Δ is much smaller than δ_1, the method should revert to flooding (i.e., *ploc* values like in the bottom part of Table 3).

If Δ is neither very large nor very small, what values should *ploc* acquire? The idea is to relate multiples of Δ to the increasing sum of the δ_i as follows: Whenever the sum of δ_i results in a value larger than the next multiple of Δ then the value of *ploc* must "take a step". As an example, assume the following values (all in milliseconds): $\Delta = 100$, $\delta_1 = 120$, $\delta_2 = 50$, $\delta_3 = 50$, $\delta_4 = 20$. Now consider Figure 8 where the sums of these values have been put on a single scale. The *ploc* value for client-side filtering (F_0) is fixed to the current location of the client. Since it takes longer for the brokers B_1 and B_2 to process a location change than the client moves, the system must insert a level of buffering at this point, i.e., *ploc* must cater for one additional step of uncertainty at this stage.

Considering that $\delta_1 + \delta_2 < 2 \cdot \Delta$, a location change can be processed fast enough between B_2 and B_3 so that no additional buffering is necessary at this point. However, the sum $\delta_1+\delta_2+\delta_3 > 2\cdot\Delta$, and so *ploc* must have one additional step between B_3 and B_4. The resulting values in the example setting for *ploc* are shown in Table 4.

Table 4. Values of $ploc(x,t)$ for the example setting with concrete timing values.

t	$x=a$	$x=b$	$x=c$	$x=d$
0	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$
1	$\{a,b,c\}$	$\{a,b,d\}$	$\{a,c,d\}$	$\{b,c,d\}$
2	$\{a,b,c\}$	$\{a,b,d\}$	$\{a,c,d\}$	$\{b,c,d\}$
3	$\{a,b,c,d\}$	$\{a,b,c,d\}$	$\{a,b,c,d\}$	$\{a,b,c,d\}$

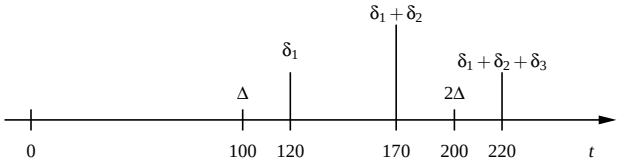

Fig. 8. Estimating *ploc* steps with respect to concrete timing bounds.

5.4 Discussion

The operations "subscribe" and "unsubscribe" in the algorithm refer to operations performed on the original routing table of the corresponding broker. In REBECA, these operations exploit the optimizations of the underlying routing strategy. For example, in covering-based routing, subscribing to a filter $\tilde{F}$ may have no effect on the routing table if there already exists a filter F' that covers $\tilde{F}$. The messages about location changes replace the administrative messages that are sent to spread the information about new subscriptions.

We have informally analyzed the total number of messages (notifications and administrative messages) generated by our new algorithm for an arguably realistic network setting, exactly one consumer and two different speeds of consumer movement: fast movement ($\Delta = 1s$) and slow ($\Delta = 10s$). We compare the results of these calculations with the total number of messages generated by flooding in Figure 9 (see [9] for a detailed description of the system assumptions and the derivation of these numbers). It is interesting to see that although our algorithm generates administrative messages on all network links for every location change of the consumer, the fraction of messages saved is still considerable. We also note that many of the assumptions made in calculating these figures have been very conservative. For example, we assume that there is only one consumer in the network and that notifications are generated by the producers according to a uniform distribution over set of locations. Both assumptions prevent the routing strategy optimizations of REBECA to play to their strengths.

6 Conclusions

This paper has presented an approach to support mobility in existing publish/subscribe middleware. We have analyzed the problem of mobility from the viewpoint of the event-based paradigm and have identified two separate flavors of

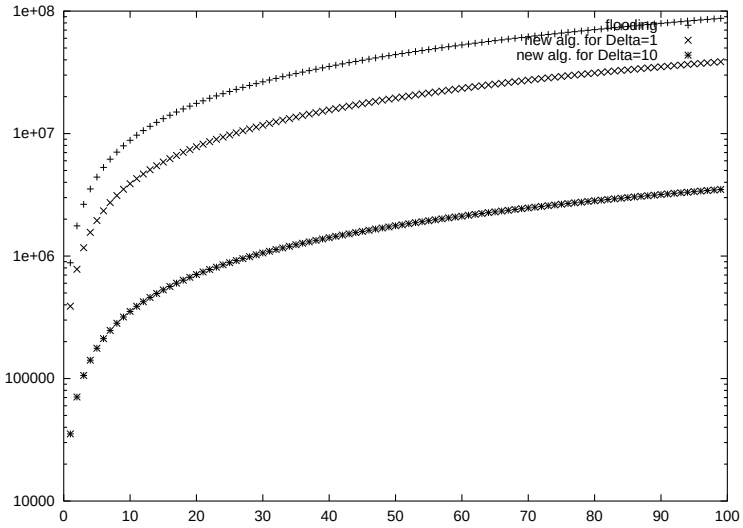

Fig. 9. Total number of messages generated for flooding and two scenarios of the new algorithm ($\Delta = 1s$ and $\Delta = 10s$). Note that the y axis has a logarithmic scale. The x axis denotes time in seconds.

mobility. While physical mobility is tied to the notion of rebinding a client to different brokers and can be implemented transparently, logical mobility refers to a certain form of location awareness offering a client a fine-grained control over notification delivery in the form of location-dependent filters. We have sketched how both notions can be implemented within the existing REBECA event system to exploit its refined routing strategies.

It is quite obvious that even both of our solutions together cannot claim to solve *all* problems related to mobility or together with REBECA constitute a *complete* mobile computing middleware. In some worst case scenarios both algorithms may lead to undesirable behavior like missing notifications or even starvation of a client, i.e., where a client does not receive notifications due to the latency of the event middleware. For example, this is the case if a client is just too fast for the infrastructure to adapt or if some network links within the broker network are too slow. We have attempted to alleviate these problems by designing adaptive solutions that should work in and can be tuned to most real world scenarios. A detailed analysis of the behavior of the solutions in more extreme and dynamic network settings is a point for future research.

Many other interesting problems concerning the combination of mobility and pub/sub infrastructures remain. For example, location-dependent filters may be generalized to "dynamic filters" that depend on a function of the local state of the client (not only its current location), like a client interested in receiving notifications for sales that he still can afford. Currently, we are investigating how logical and physical mobility can be integrated to allow for logically mobile

clients roaming beyond the boundaries of a single broker. First results using the *idea* of logical mobility to deal with the uncertainty of roaming with mobile clients and "pre-subscribe" to information at brokers at *possible* next locations seem promising but need further investigation.

Acknowledgments

We thank Gero Mühl for his cooperation in the REBECA project, and Alejandro Buchmann and Sidath Bandara Handurukande for helpful comments on an earlier version of this paper. Also we would like to thank the anonymous referees for their suggestions.

References

1. J. Bacon, K. Moody, J. Bates, R. Hayton, C. Ma, A. McNeil, O. Seidel, and M. Spiteri. Generic support for distributed applications. *Computer*, 33(3):68–76, 2000.
2. J. Bates, J. Bacon, K. Moody, and M. Spiteri. Using events for the scalable federation of heterogeneous components. In P. Guedes and J. Bacon, editors, *Proceedings of the 8th ACM SIGOPS European Workshop: Support for Composing Distributed Applications*, Sintra, Portugal, Sept. 1998.
3. M. Caporuscio, P. Inverardi, and P. Pelliccione. Formal analysis of clients mobility in the Siena publish/subscribe middleware. Technical report, Department of Computer Science, University of L'Aquila, Oct. 2002.
4. L. Capra, W. Emmerich, and C. Mascolo. Middleware for mobile computing (a survey). Research Note RN/30/01, University College London, July 2001.
5. A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.
6. G. Cugola and E. Di Nitto. Using a publish/subscribe middleware to support mobile computing. In *Proceedings of the Workshop on Middleware for Mobile Computing*, Heidelberg, Germany, Nov. 2001.
7. 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.
8. P. T. Eugster, R. Guerraoui, and C. H. Damm. On objects and events. In L. Northrop and J. Vlissides, editors, *Proceedings of the OOPSLA '01 Conference on Object Oriented Programming Systems Languages and Applications*, pages 254–269, Tampa Bay, FL, USA, 2001. ACM Press.
9. L. Fiege, F. C. Gärtner, O. Kasten, and A. Zeidler. Supporting mobility in content-based publish/subscribe middleware. Technical Report IC/2003/11, Swiss Federal Institute of Technology (EPFL), School of Computer and Communication Sciences, Lausanne, Switzerland, Mar. 2002.
10. L. Fiege and G. Mühl. Rebeca Event-Based Electronic Commerce Architecture, 2000. http://www.gkec.informatik.tu-darmstadt.de/rebeca.
11. L. Fiege, G. Mühl, and F. 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.

12. L. Fiege, G. Mühl, and F. C. Gärtner. Modular event-based systems. *The Knowledge Engineering Review*, 17(4), 2003. to appear.
13. Y. Huang and H. 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.
14. Y. Huang and H. Garcia-Molina. Publish/subscribe tree construction in wireless ad-hoc networks. In M.-S. Chen, P. Chrysanthis, M. Sloman, and A. Zaslavsky, editors, *4th International Conference on Mobile Data Management (MDM 2003)*, volume 2574 of *LNCS*, pages 122–140, Melbourne, Australia, 2003. Springer-Verlag.
15. IBM. Gryphon: Publish/subscribe over public networks. Technical report, IBM T. J. Watson Research Center, 2001.
16. H.-A. Jacobsen. Middleware services for selective and location-based information dissemination in mobile wireless networks. In *Proceedings of the Workshop on Middleware for Mobile Computing*, Heidelberg, Germany, Nov. 2001.
17. D. Johnson. Scalable support for transparent mobile host internetworking. *Wireless Networks*, 1:311–321, Oct. 1995.
18. 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.
19. G. 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.
20. G. Mühl. *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, Darmstadt University of Technology, 2002.
21. G. Mühl, L. Fiege, F. C. Gärtner, and A. 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.
22. B. Oki, M. Pfluegl, A. Siegel, and D. Skeen. The information bus—an architecture for extensible distributed systems. In B. Liskov, editor, *Proceedings of the 14th Symposium on Operating Systems Principles*, pages 58–68, Asheville, NC, USA, Dec. 1993. ACM Press.
23. W. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of the 1997 Australian UNIX Users Group*, Brisbane, Australia, Sept. 1997.
24. P. Sutton, R. Arkins, and B. 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.
25. A. Zeidler and L. Fiege. Mobility support with REBECA. In *Proceedings of the 23rd International Conference on Distributed Computing Systems Workshop on Mobile Computing Middleware*, 2003.