Mobility Support with REBECA

Andreas Zeidler Ludger Fiege
Databases and Distributed Systems Group
TU Darmstadt, Germany
{az,fiege}@dvs1.informatik.tu-darmstadt.de

Abstract

Publish/subscribe (pub/sub) proliferates loose coupling and is touted to facilitate mobility. The inherent loose coupling even allows existing applications to be transfered to mobile environments, if an appropriate infrastructure support is available. However, existing pub/sub middleware are mostly optimized for static systems where users as well as the underlying system structure is rather fixed. In this paper, we analyze the necessary steps to support mobile clients with publish/subscribe middleware. The REBECA *content-based pub/sub service is extended to accommodate to physically mobile clients, offering a location transparent access to the middleware without degrading the previously guaranteed quality of service. The transparent access allows existing applications to be seamlessly transferred from a static to a mobile scenario without having to adapt client applications.*

1 Introduction

Publish/Subscribe Systems. The 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 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.

Mobility support in pub/sub middleware. Unfortunately, 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 [3], JEDI [5], REBECA [11], to name a few) have optimized algorithms for information delivery in those settings. Support and optimizations for mobile clients are no built-in features of the infrastructure; it is left to the applications to adapt or reissue subscriptions.

Roaming clients. Obviously, the first step towards mobility is to make the pub/sub middleware mobility-aware. Therefore, we need to add support for roaming clients and their needs, e.g., buffering of notifications when disconnected, or rerouting and replay of messages to different locations. Making use of the inherent loose coupling, we should be able to continue to use existing, successfully deployed applications that are based on the pub/sub paradigm without having to rewrite them ("legacy" applications). As a first step and basis for future developments, applications should not need to be aware of mobility.

Location-awareness. A further step is making a pub/-sub system aware of the client's location, usually called *location-awareness*. An example could be a so-called *location-based service*, like a "free parking space service" or a restaurant guide. The basic idea is to express a generic interest in information depending on where the client is by using a special marker which is resolved to an appropriate set of valid information by the infrastructure itself. The advantage is that a client, while moving around, does not have to adapt its location-dependent subscriptions every time it changes location.

Related Work. We are not aware of many pub/sub systems offering mobility support. Huang and Garcia-Molina [8] provide a good overview of possible options for supporting mobility, but fail to investigate a concrete relocation protocol. In contrast to the work presented in this paper they sketch a solution relying on broker replication to meet similar quality of service requirements as in the non-mobile case. An extension of Elvin exists that allows for disconnectedness using a central caching proxy [14]. Obviously, this introduces a potential performance bottleneck and cannot exploit effects of locality as a distributed solution might do. CEA [1] and JEDI [5], too, tackle 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 can only react *after* having been moved. Interim notifications are stored in the meantime at the old location for later delivery. If the client reconnects elsewhere a valid address of the storage must be known to directly request the notifications from the new location. However, under certain circumstances notifications may get lost. Moreover, the fetched and newly arrived notifications must be merged raising problems of duplicate detection and ordering. The mobility extensions of SIENA [2] are very similar to the JEDI approach. Explicit sign-offs are required and stored notifications are directly requested from the old location, too. A rather complex leader election and group management protocol for dynamic dispatching trees is proposed in [4], which needs some further investigation of the inherent complexity.

2 Content-Based Publish/Subscribe

The following discussion is based on the REBECA notification service [11, 7], which we use as basis for the proposed mobility support.

2.1 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* 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. We assume that producers issue advertisements that describe the notifications they are about to publish in the same way as subscriptions do for receiving notifications; advertisements are used to optimize notification routing in the underlying infrastructure.

Filters are boolean 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 [10].

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. 1). The graph consists of brokers and clients. The edges are communication links that are point-to-point and assumed to be error-free for now, a common assumption that can be relieved later. 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.

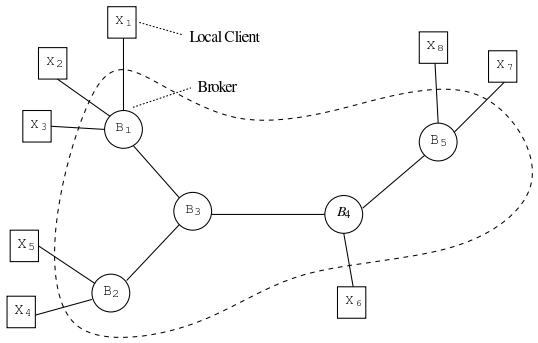


Figure 1. The router network of REBECA.

2.2 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 [3, 5]. 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 table according to the link they belong to. Obviously, this is not optimal with respect to the routing table size. A first improvement is to check and combine two filters if they are equal. More generally, the *covering* routing strategy [3] tests whether a filter F_1 accepts a superset of notifications of a second filter F_2, and in this case replaces F_2 in the routing table and is forwarded instead, 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 [10]. Only the resulting merged filter is forwarded to neighbor brokers.

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 like REBECA. The analysis leads to the identification of two orthogonal forms of mobility: physical and logical mobility; the former is suited to blend out some unwanted effects of mobility for existing applications, the latter facilitates location-aware applications.

3.1 Mobility Issues in Pub/Sub Middleware

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 and mobility-aware applications are relieved from dealing with mobility on a lower level. 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. Naïvely, location transparency is what makes existing applications mobile, e.g., stock quote monitoring seamlessly transferred from PCs to PDAs, and mobility-aware applications easier to implement. A prominent example for mobility related problems is *disconnectedness*: a mobile client usually gets disconnected from the network sometimes, reconnecting to it again later. This might be to save energy or because of geographical or administrative reasons while roaming, e.g., different network cells or changing responsibilities. Often, the access point, that is the border broker of the broker network, has changed after reconnection. For example, the border broker at home is (physically and administratively) not the same as the border broker at the office (cf. Fig.2(a)). For the stock quote monitoring application the change of location should be transparent.

However, future applications might not want complete transparency, but rely on awareness of mobility. More specifically, mobility support should not only blend out unwanted phenomena, like disconnectedness, but should also facilitate wanted behavior, like handling of the location-awareness in location-based services. Announcements of location changes are conveniently conveyed by the notification service. Yet if they have to be taken up by the application to adapt the active subscriptions manually, we miss an appropriate support. The notification service is required to support location-dependent subscriptions, enabling the infrastructure to adapt to location changes automatically and to draw from delivery localities it would otherwise have no knowledge about. Consequently, extending the interface of the pub/sub middleware to facilitate location-awareness (or even context-awareness [13]) is a promising open issue. At this stage we distinguish between:

Physical mobility. A client that is physically mobile is disconnected for certain periods of time and has different border brokers along his itinerary through the infrastructure (cf. Fig. 2(a)). The main concern of physical mobility is *location transparency*.

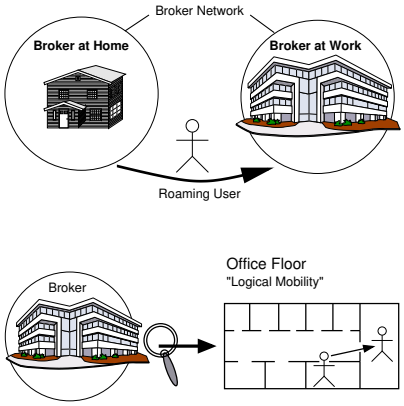


Figure 2. (a)Physical and (b)Logical Mobility.

Logical mobility. While a client that is logically mobile stays connected to the same broker (cf. Fig. 2(b)), its movement is reflected in and mapped to changing subscriptions only[1]. The main concern of logical mobility is automated *location awareness* within a defined environment. Due to space limitations we refer to [6] for further details.

Physical and logical mobility are two orthogonal aspects of mobility: 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 be used, e.g., as refinement to allow for mobility within the scope of a single broker, as shown in Fig.2(b).

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, WaveLan 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. In general, we cannot expect to have ubiquitous access to the broker network. More likely is that the broker network is accessible only from certain locations on the daily route of a user, e.g., at home or at office. Obviously, from the viewpoint of "legacy" applications the phases of disconnectedness must be made transparent by the pub/sub infrastructure without loosing expressiveness. They rely on the *impression* to be non-mobile. The required quality of service of physical mobility as experienced by the client can be summarized as follows:

[1] Note the difference to the logical mobility as defined in LIME [12] where it denotes code mobility.

- *Interface.* No change of interface between pub/sub middleware and application for mobile or non-mobile uses.
- *Completeness.* Over phases of disconnectedness, eventually the pub/sub middleware delivers all notifications for a client eventually[2].
- *Ordering.* As we guaranteed FIFO ordering with respect to the sender in Section 2 we have to guarantee this for a mobile application as well.
- *Responsiveness.* It should be clear that a pub/sub middleware is obliged to deliver notifications for a certain client as soon as possible in the mobile as well as the non-mobile case. However, as the broker network takes some time to adapt to a client at a new border broker a lag in notification delivery might be experienced due to reconfiguration.[3]

3.2.1 Possible Solutions:

One solution would be to rely on Mobile IP [9] for connecting clients to border brokers, hiding physical 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. Such an approach might be desirable if the physical and logical layout of a given system is completely orthogonal. Otherwise, a solution that can by design draw advantages from localities, i.e., physical locations which are close by in the real world have brokers "close" by in the broker network, is highly superior in terms of responsiveness and number of messages sent. The same arguments can be applied to a solution using a central caching proxy (e.g., [14]).

A different, naïve solution to implement physical mobility on top of a pub/sub system would be to use sequences of "sub-unsub-sub" calls, simply registering 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. Please note that this is a rather complicated situation. Since a mobile client usually cannot predict a change of broker before leaving its range, e.g., because it is just leaving a wireless network cell, it can only react to the new situation. So, even a "roaming-aware" client cannot unsubscribe to a producer at the old border broker, and hence, mobility in a pub/sub system is more likely a sequence of subscribe operations than a sequence of subscribe-unsubscribe-subscribe operations.

[2]Note that in general it is not possible to fulfill this requirement in all cases due to limited buffers (cf. next section).

[3]Intuitively, physical and logical mobility are comparable to mobile telephony: on the one hand, after power-on the network takes some time until messages are forwarded to your new location, while on the other hand roaming within a single cell should not involve time consuming hand-overs or disconnected calls. This is in a sense the quality of service physical and logical mobility try to approximate.

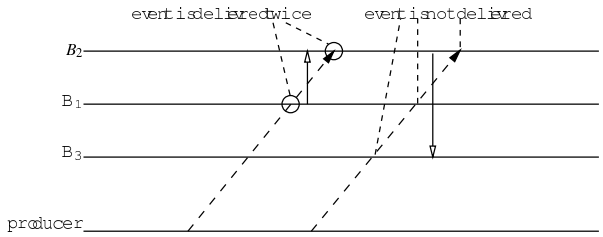


Figure 3. Missing notifications in a flooding scenario.

Without proper support relocating clients might miss several notifications or get duplicates, even if notifications are flooded in the network and location changes are instantaneous. The problem is that the time notifications are in transit through the network depends on the number of hops between issuer and receiver of a notification. As we cannot assume that location changes a user makes correspond to the delivery path of a notification in the broker network, named phenomena might occur (see Fig. 3): (i) a user moving from B_1 to B_2 after a notification is delivered at location B_1 but not yet at B_2 receives a duplicate, (ii) a user changing from B_2 to B_3 before delivery at B_2 and after delivery at B_3 misses a notification. Hence, this solution violates the requirement of completeness and possibly even FIFO ordering.

4 Notification Delivery with Roaming Clients

In this section we introduce an algorithm for extending standard REBECA brokers to cope with mobile clients, maintaining their subscriptions as well as guaranteeing the required quality of service that was described in the previous section. Apart from guaranteeing uninterrupted notification delivery, our algorithm also ensures that the old border broker will eventually receive an equivalent to an explicit *sign-off* from the client even if an explicit unsubscribe was not possible.

By design, the mechanism we use introduces a natural way of distributed caching, which seems in general preferable to a potentially problematic central caching proxy.

4.1 Prerequisites

The solution sketched in this section can be used in every environment that meets the following requirements. First, border brokers have to install and maintain a buffer for all notifications that are not yet delivered for a certain period of time in order to deal with disconnects. Second, the underlying routing infrastructure uses advertisements. Although not strictly necessary, the relocation effort is reduced substantially in that they guide the search for the old delivery path. Simple routing is assumed as routing strategy for now and extended later. Finally, border brokers or clients must

have some means of detecting the new configuration that a client has entered the range of the broker. Some form of beacon or heartbeat is presupposed and we do not go into the details here.

4.2 Algorithm Outline

We use a stepwise refinement of traditional subscription processing as described in Section 2 to devise the algorithm:

1. When reconnecting to a broker, subscriptions are automatically reissued so that clients do not need to resubscribe manually.

2. The broker network configuration is updated to accommodate to client relocation rather than handling an independent new (re-)subscription from a new location.

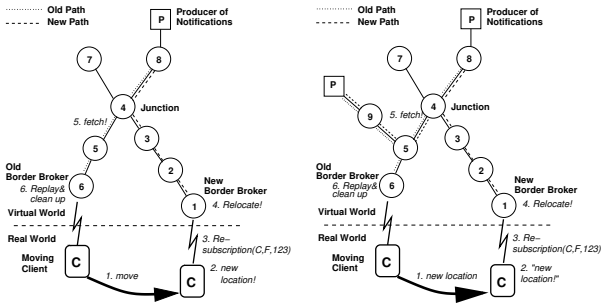
3. Notifications forwarded to the old location have to be replayed to the new one in order to bridge disconnectedness.

4. Delivery of new notifications has to be postponed until the replay is replay is finished. In this way, moving does not influence the per-sender order of notifications, fulfilling the ordering requirement.

Consider the scenario of Fig. 4(a). Client C is moving from broker B_6 to broker B_1 (step 1 in the figure). The local broker, which resides on the client, e.g., in form of libraries, is informed by the new border broker about its relocation, according to the prerequisites. It then reissues active subscriptions, which were previously forwarded through and recorded in the local broker anyway. By avoiding manual re-subscriptions of the client application, the first requirement of mobility transparency (cf. Sect. 3.2) is achieved, i.e., the interface to the middleware is not changed.

In the second step, we enable the pub/sub middleware to relocate the client. The goal of the relocation process is to update the routing configuration and redirect the old delivery paths to C to the new destination. During this process, the reissued subscription is propagated as usual in the direction of any received advertisement through B_2 and B_3 to broker B_4, setting up their routing tables. At B_4 the old and new path from producer P to client C meet (dotted and dashed line, respectively). Broker B_4 is aware of the junction because an entry of the old path of this subscription/-client is already in its routing table.[4] When the routing table in the junction is updated, new published notifications will be delivered to the relocated client. Without assuming any knowledge about the old location of the moving client, the system is able to draw from localities in that only a portion of the delivery path is changed. Changes are limited to the smallest subgraph necessary for diverting routing paths,

[4]Subscriptions can be identified if simple routing is used. For covering and merging cf. Section 4.4.



(a) Single Producer (b) Multiple Producer

Figure 4. Moving client scenarios with one and multiple producers

facilitating the timeliness/efficiency requirement which is only available with inherent middleware support.

The third step ensures completeness over phases of disconnectedness during movement. The junction broker B_4 sends a fetch request along the old path to B_6 following the routing table entries for the given subscription. All brokers along this path update their routing tables such that they are pointing into the direction the fetch originates from, i.e., B_4. Border broker B_6 as last recipient replays all buffered notifications. If delivered notifications are annotated with sequence numbers by the border broker, reissued subscriptions can in turn carry the last received number to qualify the replay. Note that replays are forwarded only in the direction of a specific subscription and do not mingle with other clients' data. After replaying the path from the old broker to the junction can be shut down by deleting the subscription's routing table entries as long as advertisement and routing entry point into the same direction; thereby excluding and stopping at the junction. In this way the notifications that passed the junction broker before its update are collected and sent towards the new location, ensuring the required completeness.

The last extension finally reorders the notifications so that the sender FIFO condition remains valid after relocation. The new border broker has to block and cache all incoming notifications that are to be delivered to the given client (not impeding communication of other clients) until the replay is finished. Of course, additional mechanisms like timeouts have to ensure that delivery is not delayed indefinitely. As with all buffering, consistency can always only be guaranteed for a predefined, finite amount of time or space.

4.3 Algorithm for Roaming Clients

We now give the details of the algorithm for roaming clients. The algorithm itself is given in two figures: Fig. 5 gives the algorithm for a border broker to which moving clients are connected. Fig. 6 gives the algorithm for an

```
{ upon recv sub (C, F, num) via link L_C do }
  if ((C, F) not in rTable) then {
    allocate new entry in rTable
    initialize entry with (F, C, L_C)
    initialize cache(C, F)
    if (num > 0) then {
       set blocking flag on entry (F, C, L_C) in rTable
    }
    send (C, F, num) to all neighboring brokers we
      have received a matching advertisement from
  } else {
    send cache(C,F) to C
  }
{ upon recv fetch(C, F, num, B_prod) from B_j do }
  if (#Advertisements matching Filter F > 1) then {
     update entry in rTable to (F, C, B_j)
     // third parameter B_j also denotes the link L_Bj
  } else {
     delete entry in rTable with (C, F, *)
  }
  send replay(fetch(C, F, num, B_prod), [e_1, ..., e_n]) to B_j
{ upon recv replay(fetch(C, F, num, nil), [e_1, ..., e_n])
       from B_j do }
  prepend notifications [e_1, ..., e_n] to cache(C,F)
  unset blocking flag on entry (F, C, B_j) in rTable
  send all notifications to client C
```

Figure 5. Actions of a border broker receiving a message from a client C

inner broker made aware of moving clients and receiving messages from neighboring brokers.

A client C connecting to a border broker sends a subscription (C, F, num) where num is the sequence number of the last notification it got (see first block of statements in Fig. 5). It is zero if the client just started and the broker has to update its local routing table `rTable`, and in order to adhere to the requirement of completeness, it has to instantiate a new *cache* for the client (we use a ring buffer data structure to limit the maximum number of cached notifications together with a Time-to-Live (TTL) mechanism for optimizing the utilization of buffer space). If the last received sequence number conveyed in the subscription is not zero the corresponding entry is marked as blocked to postpone new notifications until the replay is finished (third block of statements in Fig. 5). If the client re-appears at its previous location due to temporary communication break-downs, it simply gets the notifications cached in the meantime.

If the subscription is new, normal processing of the underlying pub/sub infrastructure is applied by forwarding it to the next brokers with matching advertisements to set up the new delivery path (block 1 of Fig. 6). The junction is reached if the received subscription of client C is already registered in the routing table, then the fetch phase starts by sending out a fetch request to the old destination. While forwarding this request towards the old location, the routing table is updated to point into the new direction so that the replay can be routed through the pub/sub network later on, without requiring a communication link between otherwise not directly connected border brokers. Moreover, the nearest branch is recorded in the request from where a different producer sends notifications (block 2 of Fig. 6). When replaying the old delivery path/routing table entries are discarded until the recorded branch is reached (block 3).

```
{ upon recv of subscription (C, F, num) from B_j do }
  if ((C, F) not in rTable) then {
    allocate new entry in rTable
    initialize entry with (F, C, B_j)
    send (C, F, num) to all neighboring brokers we
         have received matching subscriptions from
  } else {
    broker_oldnext = get(rTable, C, F)
    update entry in rTable to (F, C, B_j)
    send fetch(C, F, num, B_current) to
    broker_oldnext
  }
{ upon recv of fetch(C, F, num, B_prod) from B_j do }
  broker_oldnext = get(rTable, C, F)
  update entry in rTable to (F, C, B_j)
  if (#Advertisements matching Filter F > 1) then {
     send fetch(C, F, num, B_current) to
    broker_oldnext
  } else {
     send fetch(C, F, num, B_prod) to
    broker_oldnext
  }
{ upon recv of replay(fetch(C, F, num, B_prod), [e_1, ..., e_n])
       from B_j do }
  broker_next = get(rTable, C, F)
  if (B_prod != nil) then {
     if (B_prod == B_current) then {
       send replay(fetch(C, F, num, nil), [e_1, ..., e_n])
            to broker_next
     } else {
       delete entry in rTable
    with (C, F, *)
       send replay(fetch(C, F, num, B_prod), [e_1, ..., e_n])
            to broker_next
     }
  }
```

Figure 6. The algorithm for an inner-network broker receiving a message from broker B_j.

Replay is started when the fetch request reaches the old border broker (block 2 of Fig. 5). The size of the replay is constrained by the sequence number contained in the fetch request, if any. Since the client is typically not able to unsubscribe at its old location, receiving the fetch request is the explicit sign-off.

4.4 Extensions

Covering. If covering instead of simple routing is used to establish the routing tables, the fetch phase of the algorithm has to be extended. Now, the junction is reached if an entry with a covering subscription $F' \supset F$ is already registered. At this point the delivery path to the new location is correctly built up, but we do not know whether the old location lies in the direction of F' or in the direction of the advertisements. The fetch phase is extended in that fetch requests are sent towards all advertisements and all covering subscriptions; it is a kind of flooding in the overlay network of matching producers and consumers of similar interests. Only one of the fetch requests will not get dropped and finally reach the old border broker. The replay has to be flooded in the same overlay network if no tunneling mechanisms, internal or external, are used.

Merging. The previously denoted extensions can also cope with a broker network that is based on merging. Only the number of potential covers increases, and hence the size of the flooded overlay network. Both covering and merging promise to increase routing efficiency, but on the other hand aggravate relocation management.

Movement Speed. For simplicity reasons we assume that the client's movement speed is not too fast for the relocation process to terminate before the client moves again, i.e., the process always terminates at the correct broker. However, if re-subscriptions of the local broker are annotated with a relocation counter, which is reset after a successful replay, concurrent relocation processes can be identified and controlled in the middleware, avoiding the speed limit.

Cache Management. Even if storage constraints in the border brokers are not of concern, mobile clients may be disconnected for a long period of time in which more missed notifications are cached than the client can handle during replay. The possibly limited resources of mobile clients must be taken into account when designing cache sizes or limiting the replay by semantic filtering [8].

5 Conclusion

In this paper we 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 mobility: physical and logical mobility, the latter being considered a refinement of the former.

A relocation algorithm is presented that facilitates physical mobility with location transparency, offering the possibility to transfer existing event-based applications to mobile scenarios as well as supporting mobility-aware applications. The algorithm seamlessly extends an existing content-based routing infrastructure, the REBECAnotification service, to support non-interrupted, sender-FIFO ordered delivery of notifications to moving clients, which need not be aware of this extension. No central repository or control nor any communication outside of the pub/sub infrastructure is needed. On the other hand, applications can still benefit from the service' inherent benefits, like advanced routing algorithms. To our impression, the presented solution for mobile clients in pub/sub systems transfer the characteristics of the pub/sub, or event-based, paradigm to mobile scenarios in an appropriate way. Loose coupling and drawing from notification delivery localities is explicitly supported.

Acknowledgments

We gratefully acknowledge the contributions of Oliver Kasten and Felix Gärtner to the mobility discussions, and Gero Mühl's essential work on the REBECA system.

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] 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.

[3] 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.

[4] 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.

[5] 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.

[6] L. Fiege, F. C. Gärtner, O. Kasten, and A. Zeidler. Supporting mobility in content-based publish/subscribe middleware. In *Middleware 2003*, 2003.

[7] 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.

[8] 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.

[9] D. Johnson. Scalable support for transparent mobile host internetworking. *Wireless Networks*, 1:311–321, Oct. 1995.

[10] 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.

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

[12] A. L. Murphy, G. P. Picco, and G.-C. Roman. LIME: A Middleware for Physical and Logical Mobility. In F. Golshani, P. Dasgupta, and W. Zhao, editors, *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21)*, pages 524–533, May 2001.

[13] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US, 1994.

[14] 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.