Using a publish/subscribe middleware to support mobile computing

Gianpaolo Cugola and Elisabetta Di Nitto
{cugola, dinitto}@elet.polimi.it
Dipartimento di Elettronica e Informazione
Politecnico di Milano
P.za Leonardo da Vinci 32
20133 Milano, Italy.
Tel.: +39-2-23993666
Fax: +39-2-23993411

June 15, 2001

Abstract

In this paper we argue that the publish/subscribe model is well suited to address the requirements of mobile computing. Unfortunately, current middleware infrastructures implementing such model present some limitation mainly due to the fact that do not account for the possibility of dynamically reconfiguring the system. In this paper we briefly present an approach to partially overcome such limitations and identify some open issues.

1 Introduction

In the last few years, the advances in wireless networking and the availability of powerful mobile computing devices like laptops and personal digital assistants (PDAs) at a reasonable price, have fostered the diffusion of *mobile computing* [1] as a new paradigm of computation in which users are able to communicate and collaborate while they are moving without the need of being physically connected through cables.

Depending on the availability of a fixed network, mobile computing technologies are usually tailored to one between two scenarios: *base station* and *ad-hoc*. In a base station scenario, users exploit wireless technologies to connect to a fixed network (e.g., the enterprise-wide intranet, or even the Internet). In the more radical ad-hoc scenario, users can completely bypass the fixed network and exploit opportunistically formed network structures where communication is enabled at all levels uniquely by the mobile hosts. This allows people to communicate and collaborate even in environments where a conventional fixed network is not available (e.g., to support impromptu meetings).

Preliminary experiences in both scenarios have demonstrated that the development of new middleware technologies is crucial to help the diffusion of mobile computing and to ease the development of applications that operate in such dynamic environment. Indeed, it has become clear that the traditional client/server middleware (like RMI, CORBA, and DCOM, just to cite the best known) are no longer sufficient in a mobile setting. By imposing a tight coupling between clients and servers, and by relying on the permanent availability of the latter, client/server middleware do not accommodate well the requirements of flexibility posed by the new distributed scenarios and are impractical in a mobile scenario where the physical and logical structure of the network is made extremely fluid by connectivity changes.

In this paper, based on our experience [2], we argue that publish/subscribe middleware [3] is a good candidate for supporting mobile computing applications. Unfortunately, although several publish/subscribe middleware have been developed thus far, both in industry and in academia, they happen to be designed for fixed network environments. Thus, they do not take into account the problems introduced by wireless networks, re-

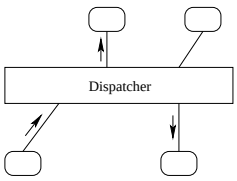

Figure 1: Publish/subscribe middleware: a general view.

lated to the dynamic reconfiguration of the network topology. As a consequence, the current goal of our research is to evaluate the impact of mobile computing on the design and implementation of a publish/subscribe middleware.

In this paper, we focus our attention on the base station scenario since it is the most common scenario in which mobility comes into place, and we describe our initial work in developing a publish/subscribe middleware for this scenario. In particular, the next section describes the publish/subscribe paradigm, identifies the inherent pros in using this kind of technology to develop mobile applications, and provides a preliminary discussion about how mobility affects it. Section 3 describes our approach to overcome the limitations of currently available publish/subscribe middleware when applied in a mobile environment. Finally, Section 4 describes the issues that remain open and present our research agenda on the topic.

2 Publish/subscribe middleware

Applications that exploit a publish/subscribe middleware are organized as a set of autonomous components, which interact through *event notifications*, often simply called *events*. Each component may notify a change in its state, or in the state of the environment it interacts with, by *publishing* an event E. Components may *subscribe* to one or more classes of events, thus expressing their interest in receiving them.

All the components that subscribed to a class of events that includes E receive a copy of such event. At the architectural level (see Figure 1), publish/subscribe middleware usually include an *event dispatcher*, a special component in charge of managing event distribution by collecting event subscriptions and distributing events to all the subscribers.

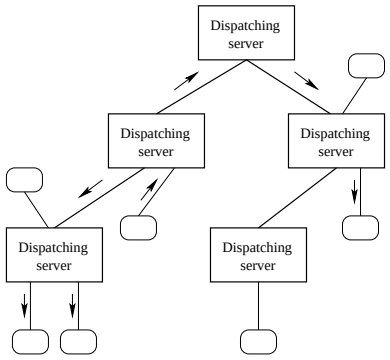

Figure 2: The architecture of JEDI.

In order to guarantee scalability, some approaches for distributing the dispatcher have been proposed [4, 5, 6]. As an example, in JEDI [4] the dispatcher is actually composed of a number of dispatching servers organized in a tree (see Figure 2). The distribution of dispatching servers is transparent to the other components. The system, in fact, guarantees that subscribers receive all matching event notifications regardless of the position of the corresponding publishers in the dispatching hierarchy. Dispatching servers coordinate among each other to minimize the network traffic. In particular, they implement a coordination protocol that is used to manage propagation of event notifications outside the boundaries of a single dispatching server.

2.1 Issues in applying publish/subscribe middleware to mobile computing

The relevant aspect of publish/subscribe middleware is that the propagation of event notifications is completely hidden to the component that has generated it. In practice, the dispatcher implements a multicasting mechanism that fully decouples event generators from receivers. This provides two important effects that makes publish/subscribe middleware a good candidate for mobile computing. First, a component can operate in the system without being aware of the existence of other components. The only knowledge required is the one about the structure of the event notifications that are of interest, in order for the component to issue the related subscriptions. For instance, a PDA could easily exploit a publish/subscribe approach to advertise its pres-

ence in a room and retrieve the services available there. Second, it is always possible to plug a component in and out of the architecture without affecting the other components directly. Moreover, the communication enabled by publish/subscribe middleware is inherently asynchronous, and thus better suited than synchronous communication to cope with the unannounced disconnection, which characterizes mobile networks.

Unfortunately, currently available publish/subscribe middleware have been designed for fixed network environments and do not take into account dynamic reconfiguration of the network topology introduced by wireless networks. In particular, most of the currently available publish/subscribe middleware assume (1) that publishers and subscribers are stationary and (2) that the pattern of communication is fixed.

A relevant exception to this situation is Jedi, which supports the disconnection and reconnection of components to the dispatching system. Jedi components can invoke the `moveOut` operation to disconnect from the dispatcher, change location, and then invoke the `moveIn` operation to reconnect again, possibly, through a different dispatching server. Dispatching servers manage temporary storage of messages for the duration of the disconnection and coordinate during the execution of `moveIn` to guarantee that messages are not duplicated and are received in a sequence that respects causal ordering. This partially solves the first issue above, but do not solve the second issue, i.e., the ability of adapting the routing strategy to changes in the pattern of communication. In particular, as mentioned above, Jedi publishing servers are organized in a tree and events are routed along this tree to move from publishers to subscribers. There is no possibility to adapt the routing strategy to changes in the pattern of communication. In other words, this approach offers good performance if the tree is organized in a proper way to minimize network traffic, but in presence of changes in the pattern of communication (very common in mobility applications) the tree should be build dynamically and rearranged when required. The next section shows an approach to extend Jedi to overcome this limitation.

3 Our approach

As we have mentioned in the previous section, our approach aims at defining the dispatching tree dynamically, in order to follow changes in the pattern of communication. The approach is based on the results of research in multicast routing [7] that has identified two main strategies:

- A routing tree (the minimal spanning tree) is computed for each pair of publisher and group of subscribers in the system.
- A single routing tree for each group of subscribers is computed, and different sources for the same class of events share the same tree.

While the first strategy guarantees that event notifications follow the shortest path from the source to each destination, it requires a relevant effort to compute the minimal spanning tree and, in the case of multiple sources, a relevant quantity of memory on each dispatching server (router) to keep track of all existing trees. Based on the above considerations, we take the second approach and particularly the Core Based Tree strategy [8].

We assume that dispatchers are connected in a (possibly cyclic) graph and that each dispatcher knows its neighbors and is able to communicate with them. We also assume that dispatchers are built on top of an IP network. This means that each dispatcher can communicate with any other it can reference through an IP address. Such assumption will be relevant in the following of the presentation.

When a component issues a subscription to its dispatcher, let us call it A, this last one checks if such subscription has been already issued. If not, it updates its internal tables and broadcasts the subscription to all the other dispatchers. Broadcasting is obtained by recursively propagating the message to neighbors until all dispatchers have received it (cycles can be avoided by checking if a message has not passed twice for some dispatcher [5], or by exploiting a minimal spanning tree that includes all dispatchers). Each dispatcher receiving the subscription updates its tables by storing the subscription information and a reference to A. Any further subscription equivalent to the one propagated by A and issued in any point of the graph of dispatchers, is not broadcasted. Instead, it is sent only to A. A has implicitly become the leader of a group of subscribers. It manages the access of other subscribers to the group and the distribution of group

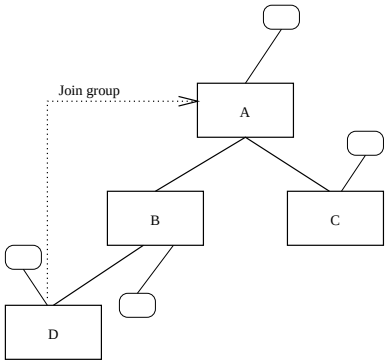

Figure 3: Creation of a dispatching tree.

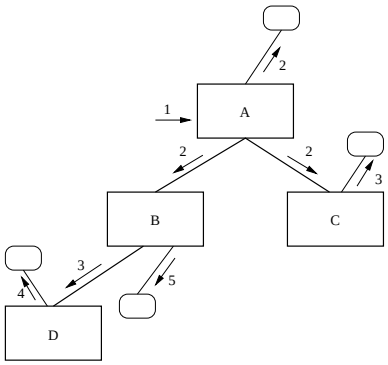

Figure 4: Propagation of notifications initiated by the leader.

members in a tree. The tree structure is used by the leader to distribute responsibilities and balance load. Figure 3 shows the case of a dispatcher D that requests to A access to the group and is redirected to B. Subscriptions originated at a dispatcher that is already part of the corresponding group, are not propagated outside that dispatcher.

When an event notification is published, if the originator is not part of the corresponding group of subscribers, the notification is directly sent to the group leader (as pointed out above, any dispatcher knows the group leaders for all subscriptions). In turn, the group leader propagates it to all its neighbors in the dispatching tree (see Figure 4). If the originator belongs to the group of subscribers, its dispatcher can immediately initiate the propagation of the notification along the dispatching tree (see Figure 5).

When a component issues the unsubscribe operation, the corresponding dispatcher starts the procedure to leave the group, unless it manages some other component that is interested in the same class of event. Three cases are possible:

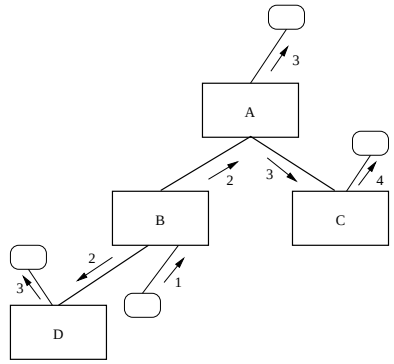

Figure 5: Propagation of notifications initiated by an intermediate node.

- The dispatcher is a leaf of the dispatching tree. In this case it simply communicates to its father in the dispatching tree that it is leaving. This information is propagated up to the group leader.

- The dispatcher is an intermediate node in the dispatching tree. In this case, it continues to be part of the dispatching tree (i.e, it continues to route event notifications) until it becomes a leaf (all its children leave the group), but communicates to the leader its intention to leave the group, so that the leader knows that it will not accept any new dispatcher.

- The dispatcher is the leader of the group. In this case the dispatcher elects a new leader and broadcasts a message to notify the entire graph of dispatchers that the group has a new leader. At this point, the dispatcher acts either as a leaf or as an intermediate node depending on its position in the dispatching tree.

During group startup, leadership clashes may occur when two (or more) equivalent subscriptions are independently broadcasted by different dispatchers in the graph. This problem is solved by ensuring that the two or more involved dispatchers manage to coordinate their dispatching trees. For the publishing nodes, all leaders are equivalent and can be contacted for propagating an event notification.

4 Conclusions

In this paper we have argued that the publish/subscribe communication model is well suited for the mobile context since it enables a good decoupling between communicating parties. We have also highlighted some weaknesses of currently available publish/subscribe middleware with respect to their usage in a mobile environment. In particular, we have identified two main limitations. First, this kind of middleware does not support temporary disconnection of components. Second, all existing infrastructures have a fixed dispatching structure and therefore are not able to adapt themselves to evolving patterns of communication.

We have already proposed a solution to the first problem in previous works. In this paper we have briefly presented an approach that enriches our prototype of middleware by building dispatching trees dynamically. Each dispatching tree handles a class of subscription and can be evolved depending on transformations in communication patterns. Also, dispatchers are not usually involved in propagating event notifications that are not interesting for them, thus resulting in an improvement in the overall performance of the system.

Still some open issues remain when applying the approach to mobile contexts. First, the frequency of disconnection and reconnection can be so high that the system may not been able to properly follow such changes by updating the dispatching trees accordingly. Second, while we have assumed so far that components always disconnect in an announced way, this is not the most common case in mobile networks, where disconnection is usually due to the fact that base stations become gradually unreachable while components are moving.

Both the above aspects need further investigation. While we plan to deal with the first problem by carefully analyzing and possibly optimizing the algorithms devoted to the management of dispatching trees, the last problem poses more general questions on the relationship between network level mechanisms and middleware. Do we need to care at the level of middleware about all problems related to unannounced disconnection? Shouldn't we rely on proper lower level mechanisms that are able to announce any disconnection to the upper levels? What kinds of abstractions should we expect to rely on? Would it be reasonable to provide in a mobile environment the same high level abstractions provided by IP for stationary networks? Wouldn't this choice limit the possibility of building context-aware systems that exploit the location information at the application level?

Answers to these questions are difficult to be found since they largely require an interdisciplinary knowledge that encompasses all aspects of mobility, from network to applications.

References

[1] G.-C. Roman, G. Picco, and A. Murphy, "Software Engineering for Mobility: A Roadmap," in *The Future of Software Engineering* (A. Finkelstein, ed.), pp. 241–258, ACM Press, 2000.

[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*, To appear.

[3] D. Rosenblum and A. Wolf, "A design framework for internet-scale event observation and notification," in *Proc. Sixth European Software Engineering Conf./ACM SIGSOFT Fifth Symposium on the Foundations of Software Engineering*, (Zurich, Switzerland), Sep 1997.

[4] G. Cugola, E. Di Nitto, and A. Fuggetta, "Exploiting an event-based infrastructure to develop complex distributed systems," in *Proc. of the 19th Int. Conf. on Software Engineering (ICSE98)*, 1998.

[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*, to appear.

[6] http://java.sun.com/products/jms.

[7] M. Ramalho, "Intra and inter-domain multicast routing protocols: A survey and taxonomy," *IEEE Communication Surveys and Tutorials*, vol. 3, First quarter 2000.

[8] T. Ballardie, P. Francis, and J. Crowcroft, "Core based trees (cbt)," in *Proceeding of ACM SIGCOMM'93*, (San Francisco, CA), 1993.