Using Publish/Subscribe Middleware for Mobile Systems

Gianpaolo Cugola[a] *cugola@elet.polimi.it*
H.-Arno Jacobsen[b] *jacobsen@eecg.toronto.edu*
[a] Dip. di Elettronica e Informazione, Politecnico di Milano, Italy
[b] Dep. of Electrical and Computer Engineering and Dep. of Computer Science, University of Toronto, Canada

The range of mobile computing applications comprises location-based services, sensor networks, and ad hoc networking. Middleware for these applications must effectively support the interaction of a priori anonymous entities, support timely decoupled processing, and mediate between potentially millions of mobile clients. These requirements are hard to achieve with traditional client/server middleware systems. We argue that the publish/subscribe paradigm effectively addresses many of the challenges raised by emerging mobile applications. In this paper, we give an overview of the publish/subscribe paradigm, present a detailed analysis of mobile application requirements, and discuss two proven implementations of this paradigm that address these requirements. We thus show that the publish/subscribe paradigm may effectively address most of the challenges raised by emerging mobile applications.

I. Introduction

The proliferation of pervasive computing devices, the integration of network access technologies (mobile wireless, wireline, and Internet), and the network enablement of all kinds of devices has given rise to a new computing paradigm, namely *mobile computing*. Broadly speaking, this area can be further broken down through a simple taxonomy that distinguishes between the nature of device mobility and the nature of the network infrastructure deployed. On the one hand, either the computing devices can be assumed stationary while on-line, but join and leave the network at different, physically distributed access points, or the devices can move even while on-line. On the other hand, either a fixed network infrastructure is deployed, or no external network infrastructure is available at all. The different combinations of these two dimensions characterize different kinds of mobile applications.

Information dissemination applications include applications such as stock, sports and news tickers, tourist, travel and traffic information systems, and emergency notification systems. Location-based services further correlate users' interests with specific location information. While these applications may fall back on a fixed network infrastructure, either a cellular wireless network, or the Internet; sensor network applications, such as environmental monitoring, often cannot assume such an infrastructure. Similarly, there are situations, like networks set up for impromptu meetings and networks to support disaster recovery operations, in which we cannot rely on any fixed network infrastructure.

Common to all of these applications is the need to continuously collect and integrate data distributed among a large set of users, sites, and applications. The application must filter and deliver relevant data to interested users and components in a timely manner. More abstractly speaking, in these scenarios the interacting mobile devices constitute a priori unknown entities, there are potentially a very large number of devices exchanging data, and the information sources and sinks may not all be present in the network at the same time.

The traditional request/response paradigm, widely advocated as the distributed computing paradigm of choice, is not adequate to address these requirements. In this classical "pull-based" approach, a client, requiring instantaneous updates of information, would need to continuously poll the information provider (i.e., the server), thus leading to server resource contention and network overload and congestion. Moreover, in many mobile applications energy is a scarce resource and unnecessary information requests should be avoided. Finally, a pure pull-based solution does not support the high dynamicity of information sources which characterizes the above scenarios, since new sources can only be discovered by exhaustively searching the network. This may be very demanding when the network is large, and is impossible in mobile and wireless environments where a continuous network access may not always be possible.

These requirements are driving the need for an in-

formation dissemination model that offers its users highly pertinent information on an as-available basis through a "push-based" approach. This can be supported extremely well through the *publish/subscribe paradigm*. This paradigm is a simple to use interaction model that consists of information providers, who publish events to the system, and of information consumers, who subscribe to events of interest within the system. The publish/subscribe system ensures the timely notification of events to interested subscribers. An event can be seen as a special message sent by an information provider and implicitly addressed to the set of information consumers which issued a subscription that matches the event.

The information processing needs for these kind of applications are immense. The middleware platform operating them must filter information for potentially millions of users, given their continuously changing location, their changing profiles (i.e., the set of subscriptions they issued), and the static and dynamic information about the environment.

This paper is organized as follows. Section II characterizes publish/subscribe middleware technology. Section III analyzes the requirements of mobile computing as they pertain to the publish/subscribe paradigm. Section IV describes the ToPSS research prototype that addresses some of the requirements outlined in the previous section, namely the one related to the need of effectively filter data. Section V describes the JEDI research prototype, which addresses other requirements, in particular scalability through distributed processing and support to mobile components. Finally, Section VI sums up and outlines open research problems.

II. Publish/Subscribe Middleware: A characterization

Applications that exploit a publish/subscribe middleware are organized as a collection of autonomous components, the *clients*, which interact by *publishing event notifications* (often simply *events*) and by *subscribing* to the classes of events they are interested in. A component of the architecture, the *event dispatcher* (or *event broker*), is responsible for collecting subscriptions and forwarding events to subscribers.

Given the potential of this paradigm, the last few years have seen the development of a large number of publish/subscribe middleware which differ along several dimensions[1]. Among many others, two of these dimensions are usually considered fundamental: the expressiveness of the subscription language and the architecture of the event dispatcher.

The expressiveness of the subscription language draws a line between *subject-based* systems, where subscriptions identify only classes of events belonging to a given channel or subject, and *content-based* systems, where subscriptions contain expressions (called *event patterns*) that allow sophisticated matching on the event content. Subject-based systems are simpler to implement, but are less flexible and expressive. In this paper we focus on publish/subscribe middleware providing a content-based subscription language since they are the only ones that can provide the required expressive power to filter millions of events for million of users with different interests.

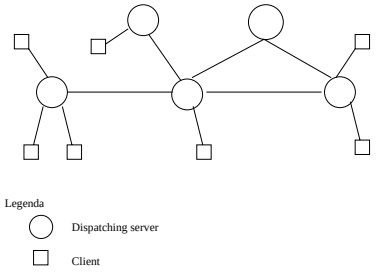

Figure 1: Distributed publish/subscribe middleware.

The architecture of the event dispatcher can be either *centralized* or *distributed*. In the first case a single component act as event dispatcher, which potentially reduces system scalability and introduces a single point of failure. In the case of publish/subscribe middleware (see Figure 1) which provide a distributed event dispatcher a set of interconnected *dispatching servers*[2] cooperate in collecting subscriptions coming from clients and in routing events, with the goals of reducing network load and increasing scalability. The overall topology of the distributed dispatcher and the strategies adopted to route subscriptions and events change from system to system. The ToPSS system presented in Section IV focuses on building and experimenting with a high-throughput oriented single, centralized dispatcher, while the JEDI system presented in Section V addresses research challenges in building and experimenting with a distributed publish/subscribe architecture.

[1] For more detailed comparisons see [1, 2, 3, 4].

[2] Unless otherwise stated, in the following we will refer to *dispatching servers* simply as *dispatchers*, although the latter term refers more precisely to the whole distributed component in charge of dispatching events, rather than to a specific server that is part of it.

III. Publish/subscribe middleware for mobility

The publish/subscribe communication and coordination model is inherently *asynchronous*, because information providers and information consumers operate asynchronously through the mediation of the dispatcher; *multi-point*, because events are sent to all the interested components; *anonymous*, because the publisher need not know the identity of subscribers, and vice versa; *implicit*, because the set of event recipients is determined by the subscriptions, rather than being explicitly chosen by the sender; and *stateless*, because events do not persist in the system, rather they are sent only to those components that have subscribed before the event is published[3].

In practice, the dispatcher implements a multicasting mechanism that fully decouples information providers from consumers. This provides three 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 presence 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. Third, the publish/subscribe communication and coordination model is better suited than traditional client/server model to cope with unannounced disconnection of components, which characterizes mobile networks.

As for the number of potential components that could be involved in a widely distributed publish/subscribe application we may consider that, to date, there exist millions of mobile wireless telecommunication service subscribers. In many countries this number is larger than the number of Internet/PC users [6]. A US mobile carrier, for example, is reported to have had 27 million subscribers in April 2000 [6] and on August 6, 2000, 18 months after the introduction of i-mode in Japan, the number of subscribers to NTT DoCoMo's i-mode service topped 10 million [7]. The same rate of adoption is occurring with IEEE 802.11 devices, which result in a fastly growing adoption of wireless LANs by several companies. All of these users could easily become clients of a pervasive publish/subscribe service.

From these considerations it is possible to derive the expected requirements of a publish/subscribe middleware for large mobile systems:

- manage mobility of application components. In a mobile setting users move while working, the middleware must take into account this behavior;
- manage changes in the underlying network topology that may occur in very dynamic settings like ad hoc networking;
- manage millions of information consumers and consequently millions of subscriptions;
- manage a potentially very large number of information providers (in some cases comparable with the numbers of information consumers);
- propagate notifications for thousands of information consumers simultaneously, which results in managing large amounts of content sent to the system for filtering, e.g., prepare for filtering, transform, type-check etc.;
- manage high volatility of users' interests (subscription update, insertion, deletion);
- process diverse content formats, ranging from topic tagged blobs and attribute-value pairs to HTML and XML marked-up data;
- support heterogeneous notification channels (e-mail, Internet-protocols, fax, phone, WAP, i-mode, ICQ etc.);
- support "approximate subscriptions" and "approximate events" to enhance system flexibility by increasing the expressiveness of the filtering language and the publication language;
- support high availability despite node failures
- perform accounting functions, i.e., publisher accounting, subscriber accounting, and possibly incentive-driven reimbursement schemes and pricing models, e.g., for advertisement and for e-coupon distribution;

[3]The publish/subscribe paradigm is conceptually stateless and this differentiates it from other interaction paradigms like Linda [5] that are based on a shared data-space. On the other hand some state has to be maintained by the infrastructure to support special features like disconnected operations. Moreover, recent advances in the development of publish/subscribe models permit to explicitly manage subscription and publication state within the publish/subscribe system. The subject-spaces model constitutes a novel data model and a research prototype that extends the publish/subscribe model with such features [4].

- perform security functions, like subscriber and publisher authentication, secure content distribution, e.g., not all subscribers may be allowed to receive all publications that match their subscriptions;

Unfortunately, currently available publish/subscribe middleware have been designed for medium size, fixed network environments and they 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) medium size networks (hundreds, not millions of components) and (2) that publishers and subscribers are stationary.

As for the first issue (i.e., scalability to million of components) the Toronto Publish/Subscribe System (ToPSS) [8] (see Section IV) experiments with publish/subscribe middleware for mobile environments that can manage millions of subscriptions and process high events rates. Similarly, the LeSubscribe publish/subscribe system [9] caters toward selective information dissemination applications on the Internet and supports a user base approximating Internet-scale. Both systems are very efficient, stationary, centralized, publish/subscribe middleware brokers. Other publish/subscribe middleware, such as TIBCO's TIB/Rendezvous, Siena [1], READY [10], Keryx [11], Gryphon [12], Elvin4 [13] in its federated incarnation and even JEDI [2] (see below) try to increase system scalability by adopting a distributed dispatcher.

As for the second point (i.e., support of mobile clients), two exceptions can be mentioned: JEDI [2], which will be described in Section V and Elvin [13].

The current Elvin implementation (4.0.3) does not support mobility yet, but some work has been done to support it [14]. In particular, a "proxy" to the Elvin server is being developed to support persistence of events to mobile clients that periodically disconnects. The most relevant limitation of this approach is that each client has to reconnect to the same proxy it has disconnected from. This means that if a clients moves from a location to another "far" from the first, it cannot choose the most closer proxy to reconnect, but it has to reconnect to the original one, thus greatly increasing network load and reducing system performance.

A discussion of how ToPSS and JEDI address the other requirements described above is deferred to their respective sections; requirements posing open research problems are summarized in the conclusion.

IV. Toronto Publish/Subscribe System

The Toronto Publish/Subscribe System (ToPSS) project focuses on developing a high-throughput content-based publish/subscribe system for high speed event notification in networked environments, targeting mobile wireless, intranet, and Internet applications. More generally speaking, ToPSS investigates questions related to the design of middleware systems targeting distributed event-driven applications, like location-based services, sensor networks, and network management. In this sense, event-driven applications are applications that must manage and respond to sudden changes of state in their environment of operation.

The project designs and implements various components to support these applications with mechanisms to efficiently and effectively support event-based computing in a distributed context. The ToPSS matching engine, for example, has been designed to manage millions of subscriptions, offer various degrees of subscription language and publication model expressiveness, and process high rates of events. While various event-management architectures and publish/subscribe system models have been defined in the past (cf. Section II and Section III), no one model clearly addresses all requirements of event-based systems. In Leung and Jacobsen [4] we discuss limitations of existing approaches and develop a novel data model for publish/subscribe systems to more effectively support event-based applications.

Figure 2 shows the architecture of a selective information dissemination system build with ToPSS components. In this application case study, user profiles and subscriber interests are modeled as subscriptions and content to be disseminated is modeled as publications. The overall system is comprised of an extensible publish/subscribe system matching engine kernel, a pervasive notification engine, a standard web server, and a web application server. The system, as depicted in Figure 2 is fully implemented and available as software demonstration.

The *matching engine kernel* is the core of the system, it implements the matching algorithm for the subscription language and publication model deployed. The ToPSS kernel implements a minimal predicate language on top of which various higher-level subscription languages are modeled. The design and evaluation of the currently implemented ToPSS matching kernel can be found in Ashayer *et al.* [8], where it has been experimentally tested on workloads comprising up to 10 million subscriptions. The kernel is im-

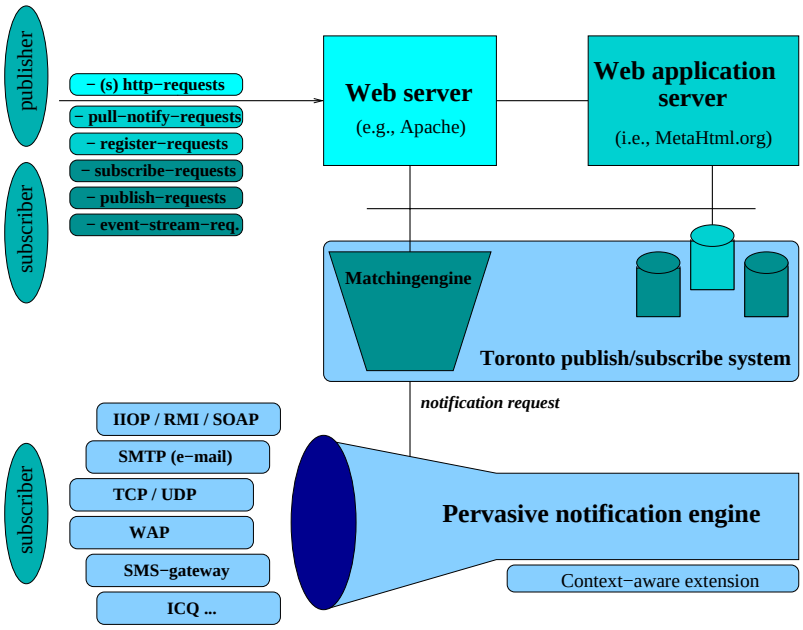

Figure 2: Deployment of ToPSS components in an architecture for selective information dissemination.

plemented in the C programming language and can be deployed as a stand-alone component, usable from any system that can link to C. The design of the minimal predicate language is active research. The objective is to map higher-level languages, such as Xpath-expressions (i.e., subscriptions) and XML-documents (i.e., publications) onto the predicate language supported by this minimal kernel. An XML-based extension of the system, X-ToPSS, is work in progress.

A-ToPSS is the approximate matching-based ToPSS research prototype that extends the kernel's predicate language with a notion of approximate matching [15], In conventional publish/subscribe systems subscriptions are either matched or not matched by a given event. This matching semantic is often too restrictive, because in may situations only imprecise knowledge about the exact value of a state variable is available, or because subscribers may often be satisfied with an event that matches a subscription *partially* or only to *a certain degree*. An event matches a subscription partially, if n of m predicates of the subscription are matched. This can be further restricted by defining which of the m predicates must not necessarily match. An event matches a subscription to a certain degree, if the evaluation of its predicates' and the subscription's truth value result in a value less than one (i.e., in a logic where *one* would represent a match.) Probability theory, fuzzy set theory, or possibility theory can be used to achieve this. Especially in mobile computing, where exact location information is not always given, devices remain disconnected for longer periods of time, and subscribers move about in unpredictable ways, is it important to be able to process imprecise data. The following depicts a sample subscription for a location-based information dissemination service that can be processed by A-ToPSS:

S_i: (*close to* downtown Toronto) `AND`
(*about* 75 square meters in size) `AND`
(*no more than* $1000) `AND`
(*close to* major grocery shopping)

The modeling of imprecise sensor readings, approximate location information, and stochastic environmental conditions are further examples that require an approximate matching-based approach for processing with a publish/subscribe system. The following depicts an approximate event, as pendant to the above subscription.

E_i: { (location, close to downtown)
(size, big)
(price, expensive)
(shopping, close-by) }

A-ToPSS is fully implemented and available as software demonstration for the processing of approximate subscriptions [15]; the processing of approximate events is work in progress.

The *pervasive notification engine* is a stand-alone component that supports a number of common transport protocols for subscriber notification. New protocols can be easily added. The engine supports a push-style notification semantic, while pull-style semantic is supported by the web application server. The notification engine receives as input a notification request,

that is comprised of a subscription ID, an optional message (i.e., could be the matching event), a protocol selector, and a subscriber address.

The research challenges in building this engine consists in developing a reliable high-performance notification protocol that, depending on the application at hand, supports different delivery semantic (e.g., guaranteed delivery despite failure). For application scenarios with millions of subscribers and frequent notifications, the overhead imposed by TCP is considerable, but the unreliability of UDP may not be tolerable, such that alternative protocols, which exploit different application characteristics (e.g., constraint, short notification message size) can be considered for designing protocol optimizations.

Our current implementation mostly targets Internet-based subscriber notification, since this is the more accessible testbed for us at the present. The engine is fully implemented in Java and supports most of the transports depicted (i.e., TCP, UDP, SMTP, ICQ, and SMS.). The *context aware extension* of the notification engine will support notification store-and-forward, scheduled notification, and content-aware addressing.

Notification store-and-forward stores and later delivers messages to mobile clients in case they are temporarily disconnected. *Scheduled notification* delivers messages to receivers on a pre-defined schedule, that may include various receiver device choices and temporal delivery preferences. *Content-aware addressing* identifies a set of receivers for an event based on the event's content, the receiver profile, and constraints expressed in the original event submitted for publication. This feature is useful to constraint the spread of notifications and to define security policies. The context aware extension constitutes work in progress.

The *web application server* ties the individual components together. In the depicted application scenario it receives http-requests from the web server and assembles html-pages for the application. The web application server manages subscriber and publisher registration, keeps track of publication and subscription types, maps between the basic type system of the matching engine and the higher level type system offered to the subscriber and publisher, and orchestrates subscriber notification once a match has been established. Both, push-style and pull-style notification semantic are supported. Pull-style notifications are implemented by the web application server. The application server is based on MetaHtml – its functionality is fully implemented. As database management system, MySql are used.

While the primary focus of ToPSS currently lies in supporting applications that need to centrally manage a large number of subscriptions and process very high event rates, a use of it as a broker component in a distributed publish/subscribe system is not precluded. Issues and challenges for this distributed scenario are addressed in the next section.

V. JEDI and some possible extensions

Developed at Politecnico di Milano and Cefriel, JEDI [2] (Java Event-based Distributed Infrastructure) focuses on both the aspects of increasing system's scalability and managing mobile users/clients.

To support the number of potential users that a pervasive publish/subscribe service could involve, JEDI provides a distributed dispatcher (see Figure 3). A set of dispatching servers, organized in a tree to simplify message routing, cooperate by collecting subscriptions and by routing events based on these subscriptions. An application component (i.e., a client of the publish/subscribe middleware) that has to join the system is free of choosing one of these dispatching servers, joining it, and using it to subscribe and receive event notifications. In particular, each dispatching server holds a routing table that tracks the interests, in terms of event patterns, of its clients and of the dispatching servers directly connected to it. When a client connected to a dispatching server S issues a new subscription, routing tables are updated (if needed) such that any of the other dispatching servers could route new events matching the subscription to S, which in turns routes them to the client.

To support mobility, JEDI natively supports the disconnection and reconnection of application components to the dispatching system. Clients of the distributed JEDI dispatcher can invoke the `moveOut` operation to disconnect from the dispatching server they are connected to, 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 approach solves the problem of managing mobility of clients but does not address neither the problem of managing mobility of servers (which could be fundamental in a very dynamic setting like ad hoc networking in which even servers are mobile components and the underly-

ing network topology may change) nor the problem of adapting the routing strategy to changes in the pattern of communication which results from mobility. In particular, JEDI dispatching servers are organized in a rather fixed tree (new dispatching servers can only be added as leaves of this tree) and events are routed along this tree to move from publishers to subscribers. It is not possible to dynamically change the topology of the dispatching tree to cope with changes in the networking environment (e.g., link breaks) or to changes in the overall workload, e.g., by adding or removing dispatching servers at runtime.

During the last two years some effort has been put to overcome these limitations. As a first result, an algorithm [16] has been developed to manage link breaks in a tree of dispatching servers. By implementing this algorithm, the topology of JEDI dispatching servers could be easily adapted to changes in the networking environment or to changes in the workload. Links among existing dispatching servers could be removed and added as needed (as soon as the tree structure is kept) and even new dispatching servers could be easily added or existing dispatching servers removed at runtime. Moreover, since this algorithm does not make any assumption about the way link breaks, thus taking into consideration even unanticipated link breaks, the same approach could be used to exploit very extreme mobile scenarios in which the dispatching system itself exploit wireless links, thus allowing dispatching servers to move.

A different approach has been described in [17] to adapt the routing strategy to the changes in workload that comes from mobility of clients. The basic idea is to use a different, dynamically built dispatching tree for each event pattern. This idea is not new and has been already adopted by some multicast IP algorithms. In particular, we adapt the Core Based Tree strategy [18] to the publish/subscribe paradigm. We assume that dispatching servers are connected in a (possibly cyclic) graph and that each dispatching server knows its neighbors and is able to communicate with them. As usual in JEDI, we also assume that dispatching servers are built on top of an IP network. When a client issue a subscription to its dispatching server (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 dispatching servers (cycles can be avoided by checking if a message has not passed twice for some dispatching server [1], or by exploiting a minimal spanning tree that includes all dispatching servers). Each dispatching server 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 dispatching servers, 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 members in a tree. 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 dispatching server knows the group leaders for all subscriptions). In turn, the group leader propagates it to all its neighbors in the dispatching tree. If the originator belongs to the group of subscribers, its dispatching server can immediately initiate the propagation of the notification along the dispatching tree.

When a dispatching server A wants to leave one of the groups it belongs to, three cases are possible:

- A 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.
- A 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 dispatching server.
- A is the leader of the group. In this case it elects a new leader and broadcasts a message to notify the entire graph of dispatching servers that the group has a new leader. At this point, A 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 dispatching servers in the graph. This problem is solved by ensuring that the two or more involved dispatching servers manage to coordinate their dispatching trees. For the publishing nodes, all leaders are equivalent and can be contacted for propagating an event notification.

The two approaches described above could be combined to dynamically adapt the dispatching network to changes in the workload that results from movement of clients and to cope with mobile dispatching servers.

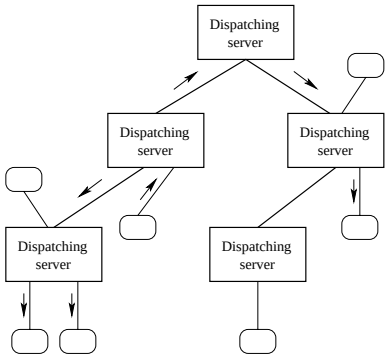

Figure 3: The architecture of JEDI.

VI. Conclusion

Mobile computing applications raise a number of challenges for the middleware designer. We have demonstrated that most of these requirements can be effectively addressed by the publish/subscribe paradigm. In particular, the decoupling among information providers and information consumers introduced by the presence of the event dispatcher matches the need of time and space decoupling introduced by mobility.

In this paper we identified two key problem areas for publish/subscribe middleware for mobile applications: scalability (i.e. support millions of largely distributed clients) and the ability to support changes in the application topology which result from mobility of components and from changes in the underlying network topology (e.g., when ad hoc networking is adopted).

To address the first problem, a publish/subscribe broker has to efficiently filter messages on behalf of its subscribers. An example of such a broker is the one adopted by the TOPSS system described in Section IV. Message filtering is constrained by the expressiveness of the subscription language. More expressive subscription languages and more intricate publication schemata and data formats are more costly to process. Scalability issue may also be addressed by distributing the publish/subscribe broker throughout the network like in JEDI (see Section V). The design of efficient routing protocols and message dissemination schemes is of primary concern in this case.

As for the second issue, a publish/subscribe middleware has to be able of supporting mobile users and it has also to be able to support changes in the underlying network (i.e., link failing and new links appearing due to mobility of devices). To solve these issues adequate protocols to adapt the routing schemata and to support disconnected operations have to be put in place.

In this paper we described possible approaches to address these issues but a lot of work has still to be done to further refine these approaches and to introduce them in commercial products.

Further requirements identified in Section III and not addressed here are the ability of processing diverse content formats, the ability of supporting heterogeneous notification channels, and the ability of performing accounting functions. Achieving of high availability in spite of node failures in distributed publish/subscribe systems is also of concern, especially in mobile environments. Security is also an open issue for publish/subscribe middleware. In particular, it is critical to support confidentiality of events given that the content of events has to be available in some form to the event dispatcher to route them.

References

[1] A. Carzaniga, D. Rosenblum, and A. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Trans. on Computer Systems*, vol. 19, pp. 332–383, Aug. 2001.

[2] G. Cugola, E. Di Nitto, and A. Fuggetta, "The JEDI event-based infrastructure and its application to the development of the OPSS WFMS," *IEEE Trans. on Software Engineering*, vol. 27, pp. 827–850, Sept. 2001.

[3] D. Rosenblum and A. Wolf, "A Design Framework for Internet-Scale Event Observation and Notification," in *Proc. of the 6^{th} European Software Engineering Conf. held jointly with the 5^{th} Symp. on the Foundations of Software Engineering (ESEC/FSE97)*, LNCS 1301, (Zurich (Switzerland)), Springer, Sept. 1997.

[4] H. Leung and H.-A. Jacobsen, "Subject-spaces: A state persistent programming model for publish/subscribe systems," tech. rep., CSRG, University of Toronto, September 2002.

[5] D. Gelernter, "Generative Communication in Linda," *ACM Computing Surveys*, vol. 7, pp. 80–112, Jan. 1985.

[6] D. Sims, "Who'll win the cell phone battle?." TheStandard.com - `http://biz.yahoo.com/st/-010524/26680.html`, Thursday May, 24 2002.

[7] Mobile Media Japan, "More than 10 million i-mode subscribers." http://www.mobilemediajapan.com/-newsdesk/imode10million/, August, 6 2000.

[8] G. Ashayer, H. K. Y. Leung, and H.-A. Jacobsen, "Predicate matching and subscription matching in publish/subscribe systems," in *Proceedings of the Workshop on Distributed Event-based Systems, 22nd International Conference on Distributed Computing Systems*, (Vienna, Austria), IEEE Computer Society Press, July 2002.

[9] J. Pereira, F. Fabret, H.-A. Jacobsen, F. Llirbat, R. Preotiuc-Prieto, K. Ross, and D. Shasha, "Le subscribe: Publish and subscribe on the web at extreme speed," in *SIGMOD digital library*, ACM Press, 2001. http://caravel.inria.fr/LeSubscribe/LeSubscribe.html.

[10] R. Gruber, B. Krishnamurthy, and E. Panagos, "The architecture of the READY event notification service," in *Proc. of the 19^{th} IEEE Int. Conf. on Distributed Computing Systems—Middleware Workshop*, 1999.

[11] M. Wray and R. Hawkes, "Distributed Virtual Environments and VRML: an Event-based Architecture," in *Proc. of the 7^{th} Int. WWW Conf.*, (Brisbane, Australia), 1998.

[12] G. Banavar *et al.*, "An Efficient Multicast Protocol for Content-based Publish-Subscribe Systems," in *Proc. of the 19^{th} Int. Conf. on Distributed Computing Systems*, 1999.

[13] B. Segall *et al.*, "Content Based Routing with Elvin4," in *Proc. of AUUG2K*, (Canberra, Australia), June 2000.

[14] P. Sutton, R. Arkins, and B. Segall, "Supporting Disconnectedness—Transparent Information Delivery for Mobile and Invisible Computing," in *Proc. of the IEEE Int. Symp. on Cluster Computing and the Grid*, May 2001.

[15] H. Liu and H.-A. Jacobsen, "A-ToPSS - a publish/subscribe system supporting approximate matching," in *28th International Conference Very Large Databases*, (Hongkong), August 2002.

[16] G. Cugola, G. Picco, and A. Murphy, "Towards dynamic reconfiguration of distributed publish-subscribe middleware," in *3rd International Workshop on Software Engineering and Middleware (SEM 2002)*, (Orlando, Florida), May 2002.

[17] 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, in association with the IFIP/ACM Middleware 2001 Conference*, (Heidelberg, Germany), November 2001.

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