DEEDS - An event dissemination service for mobile and stationary systems

Sérgio M. Duarte, José Legatheaux Martins,

Henrique João Domingos e Nuno M. Preguiça

Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
{smd, jalm, hj, nmp}@di.fct.unl.pt

Abstract

Event-driven programming has emerged as one of the mainstream paradigms for the engineering of distributed applications assembled out of loosely related software components. In this paper, we present our on-going research in the development of a flexible event-dissemination framework targeted at mobile and stationary computing environments. We discuss a solution based on virtual multicast event channels, where protocol heterogeneity support and active routing over a network of servers are advocated as a flexible answer to the challenges posed by disconnection and variable connectivity constraints.

1 Introduction

Distributed event-driven programming is an established choice for structuring applications in many different areas. Well-known examples, among others, are distributed GUIs; surveillance applications; shared virtual reality environments; multi-player games; multimedia and CSCW applications. Scenarios involving mobility are no exception; the current ability to broadcast short messages to (disconnected) mobile devices strongly hints towards an event-based programming model [1].

A crucial aspect when dealing with mobility lies in the fact that disconnection and variable connectivity are the norm, as opposed to what happens in traditional distributed systems, where they are regarded as abnormal situations. Consequently, it is expectable that conventional event-dissemination models may prove themselves inadequate in coping with these traits of mobility. In particular, it is not very realistic to expect or impose a high-level of consistency to all the listeners of an event source. Moreover, it is unacceptable to ignore the problem and just consider periods of disconnection as some sort of communication blackouts. Therefore, it seems unreasonable and inappropriate to resort to the same protocols for event-dissemination among loosely connected mobile computers and well connected stationary computers. Yet, the use of a uniform programming model based on the same abstractions seems very desirable.

Closely related to the event-driven programming model are the notions of event objects and event propagation channels. The relevance of these paradigms is unquestionable and asserted by their widespread support in almost every standard for object oriented, component based systems (e.g. JavaBeans, DCOM, CORBA).

Essential to the ev*ent channel* paradigm is that *channels* reflect structure in the event flow, by capturing patterns that are persistent relatively to application life cycles. As such, they transcend particular instances of event sources and event consumers, and glue together seemingly unrelated events and applications. Thus, *event channels* have the potential to actually drive the design and shaping of distributed applications, particularly those based on loosely coupled components that exchange information in an asynchronous manner.

Important as they are as a structuring tool and binding mechanism, *event channels* possess an increased value for mobility applications. Again, the fact that disconnection is a normal mode of operation means that the common tight-coupled models of consumer and producer interaction are arguably inadequate. Mobility scenarios, therefore, require that event production and event consumption should be made into independent processes. In this regard, *event channels* can act as true intermediaries in the event dissemination, playing a part that goes beyond the logical existence usually seen.

In the remaining of this paper, we present the main design issues driving our investigation in the development of a framework where events and event channels are meant to be standard building blocks of distributed applications, including mobile ones. This framework (and the associated system) is globally named DEEDS, which stands for DAgora* [2] Event Dissemination Service.

2 Overview of DEEDS

DEEDS is a Java based, general-purpose notification system modeled around a de-coupled, publish-subscribe paradigm based on *active event channels,* a notion partially borrowed from *active networks* [3]. The architecture is driven mainly by scalability concerns and transparent support for arbitrary communication transports and various event delivery semantics.

The active event channel concept proposed by DEEDS provides applications with what can be viewed as “virtual multicast channels”, characterized by abstract quality of service guarantees that are realized by multiple network protocols. The novelty about these enhanced *(active) event channels* lies in the possibility of binding more than one network protocol to each virtual channel and allowing any of those bindings to be subjected to localized customization by applications or system administration.

DEEDS provisions for protocol heterogeneity within the same *event channel* may sound useless, at first, or look like a source of unnecessary complication. However, there are many

*The DAgora platform and its bibliography are available at http://dagora.di.fct.unl.pt. This platform aims to ease the construction of distributed, large-scale CSCW systems.

cases where it can be exploited to great advantage. For instance, a SMS based protocol could be used to relay high priority events to a disconnected mobile computer, which in a well-connected state would rather be reached by some conventional IP-based protocol. Furthermore, a third protocol could yet be used to buffer the remaining lower priority events until full connectivity is resumed. Elsewhere on the network, a different set of protocols may be more adequate in dealing with local environment conditions suggesting a different (local) *event channel* configuration.

Having applications explicitly coded for special circumstances, requiring multiple protocols, is an alternate possibility but not really a solution. That leads, necessarily, to ad-hoc solutions that demand code modification whenever execution conditions change. This problem is addressed by DEEDS *event channel* framework by exposing only the abstract characteristics of the communication protocols in use, such as latency, or persistency, to name a few. This way, it is possible to create applications that are programmed with those characteristics in mind, not the actual protocols. In this manner, applications are bound to specific *event channel* semantics. If care is taken to preserve those channel characteristics across transport customizations, then no code modification is necessary and applications are still apt to run under the new conditions. The model promises great flexibility and transparency but relies on the guarantee that sound event channel configurations are maintained.

In accordance with their denomination, DEEDS *active event channels* can perform application specific computations on the events they route. The *active* capabilities of the channels can be used to influence the routing of events in at least two useful ways. First, it allows applications to fully explore DEEDS' protocol heterogeneity support, by allowing them to program the DEEDS' infrastructure to find the best match between the available network transports and the desired *event channel* semantics. Second, it allows applications to perform event filtering on a per event basis. This is valuable, especially for mobile applications, because it establishes a general framework for adapting, degrading or prioritize the event flow in response to network bandwidth changes [4]. Active routing is an essential part of the DEEDS system and is explained in more detail in the next sections.

DEEDS is being designed on top of some of the services and interfaces offered by Jini [5]. This choice is justified by both technological reasons and practical ones. In our mind, Jini presents a very sound design and, in particular, offers several core-services that agree very well with DEEDS requirements. Therefore, it makes good sense to use it to make our system compatible with mainstream technology and, in the process, avoid reinventing the wheel.

3 Programming Model

DEEDS events are one-way, generally small and self-contained notifications, exchanged by the processes using the system. Events are object instances of any class that extends the RemoteEvent class of Jini. This class includes the bare minimum set of event attributes (fields) and methods necessary to the propagation of events within a distributed system. On the other hand, the remaining event data is application-specific and arbitrary in both type and structure.

DEEDS top-level programming interface is purely for publish-subscribe purposes. At this level, applications interface with a particular *event channel* to send or request events. The programming model is purposely simple and focused, destined to enforce a standard programming pattern that is likely to be repeated across most, if not all, application scenarios.

DEEDS other programming interface deals with *event channel* lookup, creation, and management. User-level applications will typically use this interface to locate and bind themselves to an existing channel or create their own. Channel management, on the other hand, is directed towards service-level or administrative applications. It provides the mechanisms to select the network protocols and other software components required to implement the particular working semantics of an event channel, as denoted by the abstract properties provided in its creation. In order to remain protocol-transparent, as mentioned previously, user-level applications are discouraged to use this part of the interface.

The *active-routing* capabilities of the channel dissemination infrastructure are explored using special application-supplied objects named *routing assistants*, as explained in the following section. From the software-engineering standpoint, *routing assistants* are to be regarded as another application structuring mechanism, again, intended to impose a common programming pattern when dealing with filtering, quality of service and other transport related issues. *Routing assistants* are a tool in a framework that seeks clarity of design, by allowing the core of event-aware applications to focus on the meaning of events and worry less on how to receive them or get them through.

4 Active Routing

As stated previously, DEEDS allows multiple network protocols to be used in the propagation of events associated with a particular *event channel*. The legitimacy of this choice is based on the absence of a universal network protocol, suitable to all connectivity scenarios. Any attempts to develop a such all-powerful protocol are unrealistic. Instead, a more cost-effective alternative is to combine existing protocols and explore their relative strengths. The price one pays for that path is a more complicated event routing problem.

In the presence of multiple routes to send or receive events, it is not always possible to determine which is the best one or, given the choice, which one an application would prefer. Variable connectivity conditions conspire to make those decisions even more difficult, so that one is bound to realize that there is no solution to this problem that is both general and static. Application-assisted routing (*active routing*) helps in this regard by letting applications directly influence the routing of events.

The form of *active event routing* offered by DEEDS allows event publishers and event consumers to provide Java objects (*routing assistants*) that will assist in the routing process. Given an event and a choice of network protocols (classified by their properties), the function of a *routing assistant* is to select which protocols will be used to forward that event. *Active routing* is optional; the system uses a default *routing assistant* when one is not provided.

It is important to stress that *active routing* is primarily intended to add flexibility in the dissemination of events between end-points that experience variable degrees of connectivity, such as those involving mobile computers and a host network. In this context, mobile applications are encouraged to use *routing assistants* to manage the outgoing flow of (published) events to the host network. Likewise, events destined to a mobile computer will be forwarded according to the particular desires of the receiving applications, as encoded in the supplied *routing assistants*.

DEEDS provides a framework and a support system for the execution of routing assistants. These objects can access event fields and network monitoring variables that not only provide data for routing decisions, but also for other computations such as event filtering.

5 Architecture Overview

The DAgora Event Dissemination Service architecture consists of a network of cooperating (logical) servers; each structured as a federation of several Jini services. Jini is a Java based architecture, developed by Sun Microsystems, with the goals of providing standard mechanisms to enable users, devices and services locate, advertise, use or manage network resources. Jini views the network as a dynamic place, where services come and go with minimal outside intervention. Its design is particularly biased towards federation, so that simple, lightweight services can cooperate for a higher purpose. Like most of the extensive collection of Java standard extensions, Jini is readily available and well documented, and is distributed with an open-source reference implementation.

DEEDS (logical) servers fall, nicely, into the Jini federation model. The servers perform several functions that are very different in nature but are still inter-dependent. Instead of a complex monolithic design, it is much simpler and flexible to conceive them as smaller cooperating services. Discovery, service lookup, naming, binding, and service administration are some of the facilities provided by Jini that DEEDS will use to achieve the overall functionality of a DEEDS server.

The servers are found wherever event-aware applications are expected to execute. Proxy-servers are otherwise used in hosts with limited computing resources. To provide wide-area support, heavy-duty servers form the basis of a backbone, interconnecting far apart local area domains.

The main function of DEEDS servers is, as expected, to provide support to the actual propagation of events. For that purpose, they give shape to a logical or virtual network (Figure-1), whose "physical" links correspond to the various transport-connections established between them. Keeping with the analogy, DEEDS servers exert functions akin to those performed by the routers of a physical network. In particular, alongside with the forwarding of events, they are required to exchange state and topology information about the links of the virtual network. Additionally, on a higher level of operation, they provide the execution environment for *routing assistant* objects.

Server allocation and involvement in the event-routing process is dynamic. Servers are added or dropped automatically in response to the accounted subscriber and publisher membership lists for each *event channel*, so that routing of events uses the minimum number of servers.

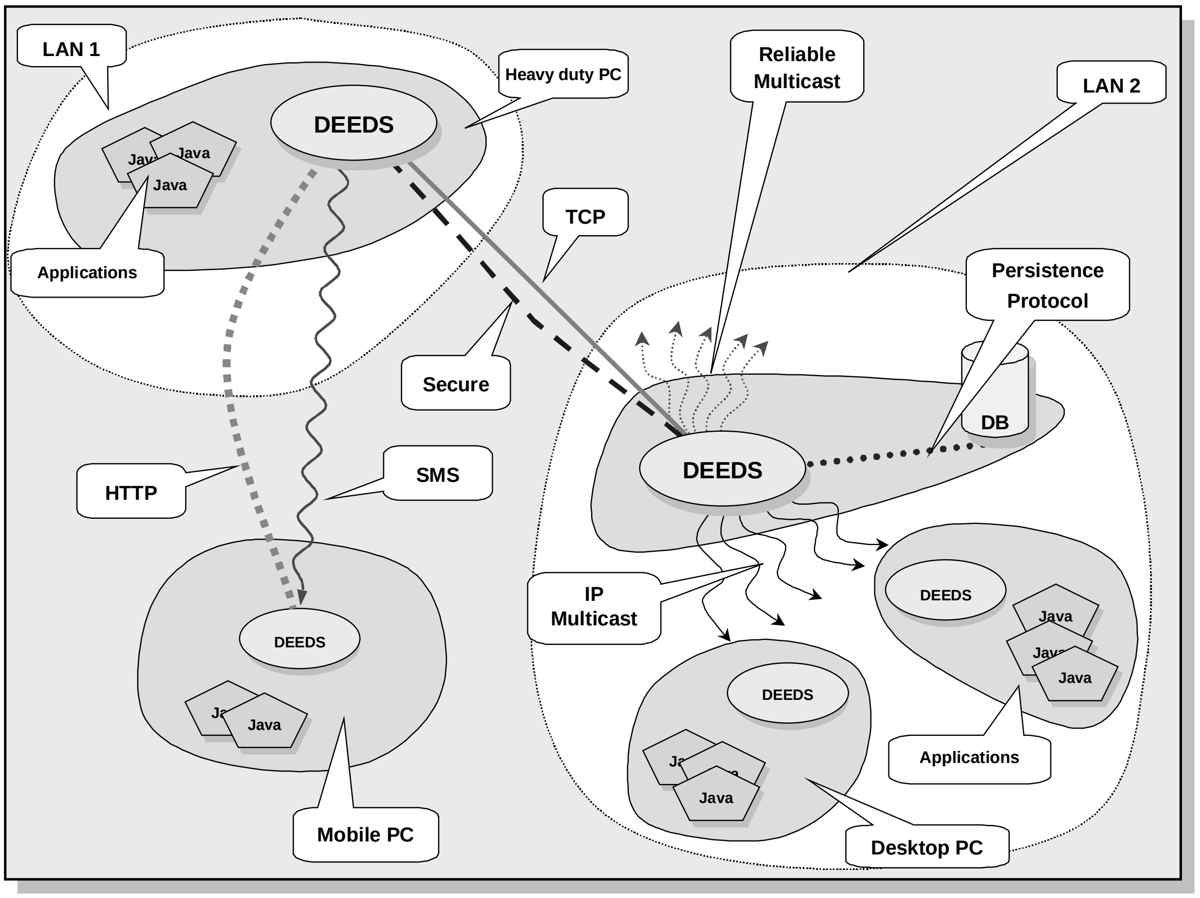

Figure 1 - An example of a virtual network of DEEDS servers.

6 Related Work

The recent shift of application development towards an increasing network-centric focus has resulted in a growing interest in the engineering of distributed applications using the familiar event-programming paradigm. This trend is reflected in the emergence, in recent years, of several event and data dissemination platforms [6, 7, 8, 9, 10, 11, 12, 13]. Most of these systems share among them many concepts and goals, and DEEDS is no exception. However, given the wide range of target environments and support technologies, emphasis on most aspects varies greatly, making comparisons difficult. Some stress reliability and high quality of service but require specific and expensive communication protocols, while a few are primarily geared toward near real-time event propagation with low quality of service. Others concentrate more on providing sophisticated event handling based on pattern matching and worry less on establishing a fully distributed solution. Some are exclusively for local-area network usage, in contrast with others that target wide-area environments. The following few are good examples of the available work and representative of their respective classes, so we find it useful to explain them in some detail and provide short comparisons with our own system.

iBus/MessageBus

[10] This is a communication middleware designed to allow Java applications to interact over a multicast channel abstraction, using a publish/subscribe model that is based on the JMS API standard by Sun Microsystems [11]. iBus supports protocol composition, by means of protocol stacks, offering various qualities of service such as reliable multicast, virtual synchrony, encryption, etc. The native set of protocols can be further extended by the developer to add new types of quality of service.

In iBus, the preferred communication model is peer to peer, in which messages are volatile and flow directly from application to application, without intervention of support servers. A recent update of this system added support for persistent messages but require funneling to a dedicated centralized hub.

iBus and DEEDS share many objectives but use radically different approaches. First, iBus is essentially a communication middleware that extends application code. DEEDS, on the other hand, is structured as an independent platform service that applications interface with. iBus communication is normally peer-oriented, while DEEDS subscriber/publisher interactions are completely de-coupled, based on events routed through a federation of servers. iBus addresses quality of service by extending or adapting the application protocol stack. Instead, DEEDS relies on active routing over a dissemination network consisting of multiple redundant paths, supported by alternative protocols.

Castanet

[13] This system is targeted at data replication and software updating through the Internet using the TCP protocol within a Java environment. The Castanet architecture is based on *channels*, *tuners* and *transmitters*. *Channels* correspond to content published by server-side provider applications (*transmitters*) that are subscribed and downloaded by the *tuners (client-side applications)*. Castanet *channels* offer data persistency on the client-side (*tuner*) and follow a synchronization policy based on polling. The system is an example of an essentially user-centric class of platforms, which, in this case, features a mechanism for uploading end-user channel preferences, as a form of feedback. This platform bears little or no resemblance with DEEDS.

Cobea

[6] This is one of the CORBA based event architectures. Cobea, in particular, uses a publish/register/notify model that supports server side event filtering based on parameter templates. In Cobea, event sources first *publish* in a *trader* the types of events they *notify* and their respective interfaces. Clients are notified when events match the parameter template that they provided when they registered their interest. Cobea follows a client/server model and relies on filtering to improve scalability. DEEDS and Cobea are different in both objective and overall architecture. However, another recently available CORBA based architecture [12] is somewhat closer in concept, supporting the event channel model, but having nothing resembling DEEDS' transport heterogeneity or active routing capabilities.

Salamander

[7] This is a feature-rich wide-area data dissemination middleware, designed to support *push*-based applications. The publish/subscribe paradigm offered presents several important characteristics such as persistent database queries, resource announcement and discovery, adaptable quality of service, data persistency and support for client heterogeneity. The Salamander platform allows applications to interface to virtual distribution channels in an attribute data space supported by a tree of servers. Client applications connect to points of service (nodes) in the server tree to publish or consume data. Furthermore, application plug-in modules can be added along the distribution path to allow data modification (degradation) and filtering.

DEEDS and Salamander are similar in several ways, but Salamander is considerably biased towards data storage/caching/digestion along a hierarchical dissemination path, while DEEDS favors a simpler flat multicast architecture with support for multiple alternative (less conventional) transport protocols.

7 Concluding Remarks

We conclude this document with a few final thoughts and open issues regarding the event-dissemination system we are developing.

First, we advocate that an event-dissemination platform aiming at general usefulness must be backed by a particular programming model. We believe that such model should envisage that the core of an event-based application should be concerned, solely, with the meaning of events, on the assumption that they are delivered according to some desired and *documented* fashion. It is up to the remaining of the system to meet such *expected* quality of service.

Meeting the above requirements for both stationary and mobile systems is difficult, to say the least, especially if simplicity and flexibility are also to be achieved. We propose that a rich, independent infrastructure of communication resources, tapped by separate and specific application-code, according to similar principles of active networking, will be able to provide an adequate answer for a wide range of application scenarios. Namely, it will be able to address the specific challenges posed by disconnection and changing connectivity quality introduced by mobility.

The above model is a significant departure from the more conventional approaches to the event dissemination problem. Its validity remains largely unproven and requires further investigation. Work is in progress in the construction of a demonstration prototype and closer analysis of model application scenarios.

8 Bibliography

[1] T. Imielinski, H. Korth. Introduction to Mobile Computing. Mobile Computing, ed. T. Imielinski and H. Korth, Kluwer Academic Publisher, 1996.

[2] H. J. Domingos, José A. Legatheaux Martins, Jorge F. Simão, J., "A Generic Platform and Flexible Object-Group-Oriented Framework to Support Large Scale Collaborative Applications," in Proceedings of the HICSS-30, 30th International Conference on System Sciences - Vol.4, IEEE Computer Society Press, January 1997, pp.s 82 - 91

[3] Jonathan M. Smith, et all, Activating Networks: A Progress Report, IEEE Computer, April 1999, Vol. 32, No. 4, pp.s 32-41

[4] B. Noble, M. Satyanarayanan, D. Narayanan, J. Tilton, J. Flinn, K. Walker. Agile Application-Aware Adaptation for Mobility. In Proceedings of the 16th ACM Symposium on Operating Systems Principles, 1997.

[5] Jim Waldo, The Jini Architecture for Network-centric Computing, CACM, July 1999, pp.s 76-82 and http://www.sun.com/jini

[6] Chaoying Ma and Jean Bacon, "COBEA: A CORBA-Based Event Architecture," in Proceedings of the 4th Conference of Object-Oriented Technologies and Systems, Santa Fe, USA, April 1998, pps 117 - 131

[7] G. Robert Malan, F. Jahanian and S. Subramanian,"Salamander: A Push-Subscribe Distribution Substrate for Internet Applications," In Proceedings of the USENIX Symposium on Internet Technologies and Systems, Monterey, USA, December 1998, ppg.s 171 - 181

[8] Patterson, J. F., Day, M., and Kucan, J., "Notification Servers for Synchronous Groupware", in Proceedings of the 6th ACM Conference on Computer- Supported Cooperative Work, ACM Press, Nov. 1996, pp. 122-129.

[9] Object Management Group. Common Object Services Specification Volume 1

[10] S. Maffeis, "iBus/MessageBus - The Java Intranet Software Bus", http://www.softwired.ch

[11] Sun Microsystems, “Java Message Service”, http:/java.sun.com/jms

[12] Object Management Group, “Notification Service - TC Document, telecom/98-06-15”

[13] Marimba Inc. “Castanet”, http://www.marimba.com