Location-based Publish/Subscribe

Patrick Th. Eugster† Benoît Garbinato‡ Adrian Holzer‡

† *Chair of Software Engineering*
Swiss Federal Institute of Technology in Zürich
patrick.eugster@inf.ethz.ch

‡ *Distributed Object Programming Lab*
Université de Lausanne
{benoit.garbinato, adrian.holzer}@unil.ch

Abstract

This paper introduces the concept of location-based publish/subscribe (LPS), which allows mobile ad hoc applications to anonymously communicate with each other, depending on their locations. With this concept, publish/subscribe topics are typically expressed in a dynamic manner including proximity criteria, e.g., "I subscribe to all events on topic T *published within some range* R*". We advocate that location-based publish/subscribe is a key programming paradigm for building mobile ad hoc application, and sketch our current implementation, which is based on standard APIs of the Java 2 platform, Micro Edition.*

1. Introduction

In recent years, several researches have been studying the next generation of mobile communication and information infrastructures, based on self-organisation approaches, e.g., Terminodes [8], CarNet [11]. In this context, ad hoc networks are expected to play a key role in the future, although nobody knows which emerging technology will win (Bluetooth [10], Wi-Fi [2], etc.) and when it will be ready for prime time. When this will happen however, it seems reasonable to forecast that adequate programming frameworks will be necessary to leverage ad hoc networks and build the next generation of mobile applications.

Many authors have proposed *abstractions for programming in mobile ad hoc environments*, mostly relying on a form of tuple space combined with a notion of context (e.g., [3]). The tuple space provides *anonymous* communication between peers, while the context is used to improve location-*awareness*. To provide *asynchrony*, tuple spaces are extended by callback primitives on the consumer side, coming close to a publish/subscribe style. To further decouple peers, interacting entities are viewed as agents, which may roam across several tuple spaces representing transitory communities. One can however argue that it is somewhat misleading to promote tuple spaces in a mobile ad hoc environment, as the tuple space abstraction originally modelled one global, unique, shared space [6]. We advocate that the publish/subscribe style leads to a slimmer and more inherently asynchronous interaction. The key difference between our *location-based publish/subscribe* (LPS) and more classical topic- & content-based variants lies in the existence of an external context that impacts the matching of published events and subscriptions. This notion of context, which is used in addition to the dynamic content of events for matching them against subscriptions, makes LPS a generalization of the topic- & content-based publish/subscribe mixture, which has become the *de facto* standard in industrial settings. Just like static topic information can be used to efficiently route and filter events by pre-establishing connections between potential publishers and subscribers, dynamic context information can be used for improving efficiency when conveying location information in an ad hoc application.

The contributions of this paper are the following. In Section 2, we propose a definition of mobile ad hoc applications that is *independent of ad hoc networks* through *Pervaho*, a platform providing high-level support for programing mobile ad hoc applications. Section 3 presents the LPS paradigm, the cornerstone of the Pervaho platform, along with the specification of a corresponding service in Java. Section 4 then provides an overview of our current implementation of the LPS service in Pervaho. Section 5 concludes with some perspectives for future work. More details, including two example mobile ad-hoc applications built on LPS, and a thorough discussion of related work can be found in [4].

2. The Pervaho platform

Ad hoc networks exhibit an inherent *here & now* nature: at some point in time (now), the network is defined by all nodes that are both within a given range and online (here). This also characterizes ad hoc applications [5]. However, we believe that defining ad hoc applications simply as *pieces of software that run on ad hoc networks* does not adequately capture their essence. By coupling ad hoc

applications to a particular network architecture, one reduces the generality and effectiveness of the definition. In an ad hoc network for instance, the willingness to collaborate directly translates to going online or offline, which does not allow for finer control over the membership of ad hoc communities.

2.1. Mobile ad hoc applications

Thus, we define an ad hoc application as *a self-organizing application involving mobile autonomous devices, interacting as peers and whose relationships are meaningful because they are within some physical range defined by the application semantics*. That is, an ad hoc application must exhibit three features: *(1) a set of autonomous mobile devices, (2) a peer-to-peer communication model, and (3) a proximity-based semantics.* The peer-to-peer communication model is by no means limited to ad hoc networks. Peer-to-peer communication simply means that collaborating entities do not assume predefined client or server roles but rather interact directly, via some underlying communication layer. The fact that the communication layer relies on a genuine ad hoc network or not is irrelevant for ad hoc applications.

By abstracting the network level, we can now imagine mobile ad hoc applications defining proximity criteria within the range of a Metropolitan Area Network (MAN), i.e., several kilometers, and communicating in a peer-to-peer manner across such distances. This is simply impossible with genuine ad hoc network technologies currently available, yet can be achieved today if we set up mobile phones to access Internet via some GPRS/UMTS gateway and to build a peer-to-peer overlay network. The responsibility of building such an overlay network would typically be that of the LPS service.

2.2. Location-aware communication

Having decoupled mobile ad hoc applications from the underlying network technology, we still need a way to express proximity-based semantics. Here, we advocate that mobile ad hoc applications require some kind of *location-aware communication, supporting anonymous and asynchronous interactions*, in addition to the obvious location service. Furthermore, we strongly believe that location-aware communication should be provided in the form of a location-based publish/subscribe service. The Pervaho platform precisely aims at offering such programming support for mobile ad hoc applications. Its architecture consists of two main layers (see Figure 1): a high-level programming model and an underlying location-aware communication service.

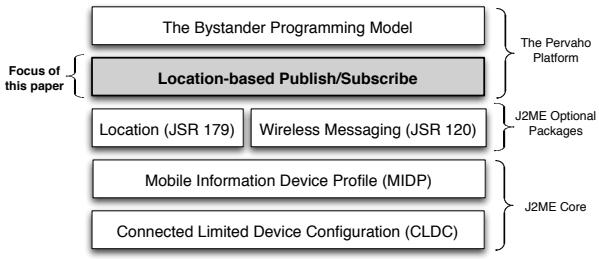

Figure 1. The Pervaho platform architecture

At the top level, the *Bystander programming model* allows programmers to express mobile ad hoc applications via a *declarative extension* of the Java 2 platform, Micro Edition (J2ME). The Bystander model reflects the inherent "here & now" semantics of ad hoc applications, and is implemented as a lightweight Java language extension offering the notion of condition-driven methods: a mobile ad hoc application consists of a set of condition/method pairs associated declaratively, where the condition expresses what the context should offer "right here & right now" for the corresponding method to be executed.

3. Location-based publish/subscribe (LPS)

The core concept underlying publish/subscribe is to view interacting entities in two roles: a first role, that of *publisher*, consists in generating events, and a second one, that of *subscriber*, consists in advertising interests in particular kinds of events. The goal of the publish/subscribe service is to, upon occurrence of an event, trigger notifications on the subscribers whose interests match the given event.

LPS starts from a generalization of the combination topic & content-based publish/subscribe. That is, an event is published in a particular *geographical context* and the matching process is performed dynamically on this context, as well as on the *content* of events. The explicit distinction of content and geographical context is motivated by the nature of mobile ad hoc applications, and *location information* must clearly be treated as first class object (just like topics), for the LPS engine to efficiently handle this information.

Location is of course not the only context information that could be relevant to mobile ad hoc applications. Other physical values (e.g., speed, temperature), as well as resource availability (e.g., battery power, computing resources) could be useful as context information, depending on the actual application, and on the sensing and monitoring capabilities of the mobile devices being used.[1]

[1] In Pervaho, any context information is made available to the Bystander model (including location) via predicates controlling method executions. The corresponding context events might however originate from different underlying services (one of them being LPS), depending on their types.

3.1. A location-based publish/subscribe service

The LPS service allows mobile ad hoc applications to transparently and anonymously communicate with each others via a subscription and publication system, *based on their location.* As illustrated in Figure 2, the LPS service interface proposes four methods, which will be described in the following.

```
public interface LPSS extends Connection {
    void publish(Event e, long range);
    Publication publish(Event e, long range,
                        long timeToLive);
    void unpublish(Publication pub);
    Subscription subscribe(Event e, EventHandler hd,
                           long range);
    void unsubscribe(Subscription sub);
}
```

Figure 2. API of the LPS service in Java

A *publication* represents an event distributed within a determined geographical range around the publisher, called the *publication space*. To create a publication, publishers use the `publish()` method. A publication can be *persistent* or *non-persistent*. With a non-persistent publication, the event is offered for distribution to all subscribers located in the publication space *at the time of the publication*. A non-persistent publication is performed thanks to the `publish()` variant taking two arguments, i.e., an event and a publication range (expressed in meters). With a persistent publication, the event is offered for distribution to all interested subscribers located or entering the publication space *before the event is unpublished*. A persistent event can be unpublished either by the publisher or by the service after a determined period known as its *time-to-live*. The publication space is centered around the publisher, implying that the publication space moves along with the publisher between the time the event is published and the time the event is unpublished. A persistent publication is performed thanks to the `publish()` variant taking three arguments, i.e., an event, a publication range and a time-to-live (expressed in milliseconds). This method also returns a publication object, which can be used to explicitly unpublish the persistent event before its time-to-live expires, using the `unpublish()` method.

A *subscription* is a request to receive events published by producers located within a determined geographical range around the subscriber – the *subscription space*. In addition, content-based selection of events can be achieved via an *event template*. A subscription is performed thanks to the `subscribe()` method, which takes an event template, a subscription range (in meters) and an event handler as arguments. The event template is an event object used as filter: this object contains all attributes a published event is required to have in order to be considered a match. The event handler is an object implementing the `standUp()` callback, which is triggered *asynchronously* when a matching event is found (the event being passed as argument). This method is responsible for handling the matching event. The `subscribe()` method also returns a subscription object, which can be used to cancel the corresponding subscription via the `unsubscribe()` method.

In order to be called a *match*, a publication/subscription couple has to meet two conditions: (1) the location match and (2) the content match. The first condition is met when the publisher and the subscriber are both located in the intersection of the publication and the subscription spaces, as illustrated in Figure 3. The content match condition is met when the published event contains at least all attributes specified in the subscribed event template. This is similar to what most content-based publish/subscribe platforms are offering.

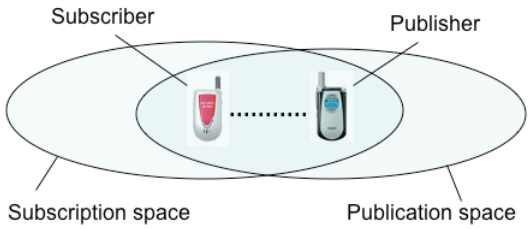

Figure 3. Location-based event matching

4. Implementation overview

Our current implementation of LPS is based on the *Mobile Information Device Profile* (MIDP) of the Java 2 platform, Micro Edition (J2ME) [7, 1]. The MIDP specification extends the *Connected Limited Device Configuration* (CLDC). The CLDC specification defines the base set of APIs along with the KVM for resource-constrained devices like mobile phones, pagers, and mainstream personal digital assistants. When coupled with a profile such as MIDP, it provides a solid Java platform for developing applications to run on devices with limited memory, processing power, and graphical capabilities.

From the communication viewpoint, LPS relies on the *Generic Connection Framework* (GCF), an essential part of CLDC that provides an extensible, generic I/O framework for resource constrained devices. A total of seven interfaces are defined in GCF, with `Connection` at the root. While many device vendors only support HTTP, the GCF foresees support for various protocols. The `Connector` class acts as a factory for instantiating such protocols. As illustrated by the `LPSS` interface in Figure 2, our LPS implementation integrates in the GCF infrastructure, which supports URL-

based protocol selection, e.g., `http://java.sun.com`, `lpss://trafficJam`, etc. As far as location-oriented support is concerned, LPS relies on the Java Specification Request (JSR) 179 [9], an APIs for managing location information in J2ME. More precisely, we have been using the official reference implementation provided by Nokia to test our approach.

4.1. LPS architecture

Our current prototype of LPS relies on a web service accessed via HTTP by clients, i.e., mobile devices; this web service plays the role of communication backbone and performs the matching between published events and subscribers. In addition to the web sercice, our architecture is based on a client module running on each mobile device; this module is in charge of relaying information to the web service. This hybrid approach (static backbone/dynamic clients) is simple to implement and offers a high degree of availability and reliability. Furthermore, given the current uncertainty regarding the future winning ad hoc network technology, this architecture is well suited for a realistic and present implementation of LPS, relying on the mobile telephony infrastructures of GSM operators. It is important to note that this implementation does not contradict our definition of mobile ad hoc applications (Section 2). On the contrary, it illustrates the freedom one obtains when decoupling mobile ad hoc applications from the notion of ad hoc network.

The *client module* is responsible for handling all five incoming requests from the ad hoc application, e.g., publish (persistent and non-persistent), subscribe, unsubscribe and unpublish. For each request it creates a specific message containing (1) the client ID, (2) the request type (publish, subscribe, etc.) and (3) a publication or subscription ID. In the case of new subscriptions or publications, the message also contains the publication/subscription information (e.g., event, range) and the device's current geographic location, which is obtained thanks to the JSR-179 location service. The client module also sends a second type of messages: location update messages. These messages are sent to the server whenever the module detects a location change. More precisely, this is only necessary if the corresponding mobile ad hoc application has an active subscription or an active *persistent* publication. The stack layering of the client module is pictured in Figure 1. Contrary to what this figure suggests, however, the client stack is not strictly vertically layered. For example, some parts of the LPS layer directly access the MIDP or the even CLDC APIs.

The *web service* is responsible for checking potential event matches each time it receives a relevant update from some client, i.e., a new publication, a new subscription or a location change. To achieve this, it keeps track of (1) all the persistent publications and the location of their publishers and (2) all subscriptions, the location and the address of their subscribers. When a new match is found, it sends the matching publication/subscription couple to the concerned subscriber's client module, using its address. Upon reception of this message, the client module triggers the `standUp()` method on the subscription's event handler.

5. Concluding remarks

In this paper, we first made an attempt of precisely characterizing applications that are typical of ad hoc networks. We presented a paradigm for programming such applications, called location-based publish/subscribe, as well as a J2ME-based implementation of it. LPS leverages on the blend of topic- & content-based publish/subscribe, by generalizing the notion of topic to a that of a context expressing proximity criteria. Regarding ongoing work, we are currently investigating a fully decentralized implementation of LPS, as well as a more flexible way than mere templates to perform content-based selection of events [**?**].

References

[1] MIDP APIs for wireless applications (white paper). Sun Microsystems, 2001.

[2] AirPort Extreme, Technology Overview. Apple Computer, April 2004.

[3] G. Cabri, L. Leonardi, and F. Zambonelli. MARS: A Programmable Coordination Architecture for Mobile Agents. *IEEE Internet Computing*, 4(4):26–35, 2000.

[4] P. Eugster, B. Garbinato, and A. Holzer. Location-based publish/subscribe. Technical Report DOP-20050124, University of Lausanne, DOP Lab, January 2005.

[5] B. Garbinato and P. Rupp. From ad hoc networks to ad hoc applications. In *Proceedings of the 7th International Conference on Telecommunications*, pages 145–149, Zagreb (Croatia), June 2003.

[6] D. Gelernter. Generative Communication in Linda. *ACM Trans. Prog. Lang. Syst.*, 7(1):80–112, Jan. 1985.

[7] http://java.sun.com/j2me. Java 2 Platform, Micro Edition (J2ME). Sun Microsystems.

[8] J. P. Hubaux, T. Gross, J. Y. L. Boudec, and M. Vetterli. Towards self-organized mobile ad hoc networks: the Terminodes project. *IEEE Communications Magazine*, 31(1):118–124, 2001.

[9] JSR-179 Expert Group. *Location API for Java 2 Micro Edition, Version 1.0*, September 2003.

[10] J. Kardash. Bluetooth architecture overview. *Intel Technology Journal*, 2000.

[11] R. Morris, J. Jannotti, F. Kaashoek, J. Li, and D. De Couto. CarNet: A scalable ad hoc wireless network system. *IEEE Communications Magazine*, 31(1):118–124, September 2000.