Communication Abstractions in MundoCore

Erwin Aitenbichler, Jussi Kangasharju
Darmstadt University of Technology, Department of Computer Science, Telecooperation Group
{erwin,jussi}@tk.informatik.tu-darmstadt.de

Abstract

Ubiquitous computing with its many different devices, operating system platforms and network technologies, poses new challenges to communication middleware. Mobile devices providing vastly different capabilities need to integrate into heterogeneous environments. We have identified a key set of requirements for ubiquitous computing communication middleware which are modularity, small footprint, and ability to cope with handovers. In this paper we present MundoCore which provides a set of communication abstractions based on Publish/Subscribe for ubiquitous computing scenarios. We present the communication model of MundoCore and discuss our implementation, along with our experiences and observations from the implementation process.

1 Introduction

Internet is currently enjoying great success as a means for communication between people and between people and computers. Popular Internet services include email, the Web, and more recently mobile services, such as WAP. Advances in mobile networking, as demonstrated by GPRS and in the future UMTS, will bring a greater need of communications.

However, the biggest foreseeable growth of communication needs does not come from people-to-people or people-to-computer communications, but from computer-to-computer communications. Smart appliances and future pervasive computing architectures will mean billions and billions of devices, all with communication capabilities and needs. In the future, a large majority of traffic on the Internet will come from devices talking to other devices.

In this paper we present communication abstractions in MUNDO, our project on pervasive and ubiquitous computing infrastructures which we present briefly in Section 2. One of the main focus points of MUNDO is efficient communications between different devices that form different parts of the infrastructure. In this paper we present the core communications component of MUNDO which we call *MundoCore*. MundoCore is responsible for handling all the basic communication needs of all MUNDO entities and it provides all the communication abstractions needed by the different entities.

A fundamental way in which nomadic computing differs from desktop computing is the great variability of network connectivity. Current applications treat changes in bandwidth or latency as exceptions or errors, but these conditions must be treated as the normal case in nomadic environments. Network access takes place from different locations with a number of different devices by means of different network providers.

From our experiences in designing and implementing parts of MUNDO and MundoCore, we have identified a set of key requirements for pervasive computing communication middleware. These requirements are:

- **Modularity**. The middleware must have a minimal kernel and we need the ability to plug in services on-demand, according to our current needs. Pluggable services also offer us the possibility for determining the *service fidelity* on-demand. By service fidelity, we mean total quality of the service obtained by combining several pluggable service modules.

- **Small footprint**. Many pervasive computing devices (e.g., the Talking Assistant [1]) have only very limited resources in terms of memory and processing power, hence it is vital that the middleware has a small memory footprint. Being able to plug in services on-demand helps us achieve this goal.

- **High mobility**. The middleware must support

both horizontal and vertical handovers. Some devices are likely to be mobile and they will need efficient handovers. Currently, a handover often results in a broken TCP connection which most applications cannot handle well.

- **Heterogeneity**. The middleware must support many different hardware and operating system platforms, starting with small sensors up to server clusters.

From a software engineering point of view, abstracting all the communications of MUNDO into MundoCore provides us with several advantages. The main advantage is that it allows us to separate the communication layer and services. In effect, using MundoCore pulls out all communication related code from service code, making it easier to write new services, since now the service code does not need to handle as many exceptions, deal with broken connections, etc.

In the rest of the paper, we present how we have implemented these requirements in MundoCore. In our opinion, it is highly unlikely that a single platform will dominate in the pervasive computing world. Hence, it is vital that our implementation is also portable to different architectures and devices.

1.1 Related Work

Conventional middleware systems like CORBA are usually very resource intensive. They are not suitable for small devices like often found in pervasive computing and usually assume a mostly stable network environment.

Modular middleware approaches that specifically target pervasive computing applications include UIC [9] and BASE [2].

This paper is organized as follows. Section 2 gives an overview of MUNDO. Section 3 presents MundoCore. In Section 4 we discuss our experiences and observations learned from our implementation of MundoCore.

2 Mundo

In this section, we provide a very brief overview of the MUNDO project. Figure 1 shows an overview of our MUNDO architecture. A more complete description can be found in [5]; below we will briefly present the different entities in MUNDO.

ME (Minimal Entity)

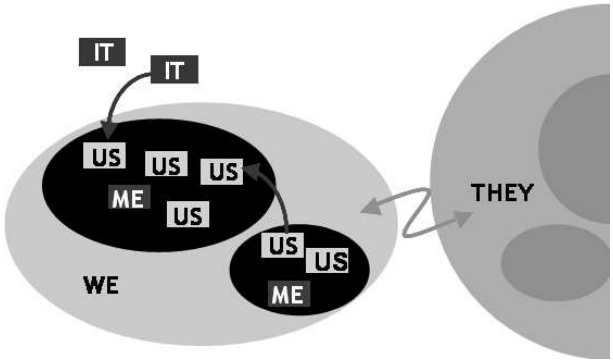

Figure 1: Mundo vertical architecture

We consider ME devices as the representation of their users in the digital world. The personal ME is the only entity always involved in the user's activities. Our decision for one individually owned device rather than a publicly available infrastructure comes from *trust establishment*. As an increasing amount of computer based activities have (potential) legal impact and involve privacy issues, it is vital that users can trust the device carrying out such transactions.

US (Ubiquitous aSsociable object)

Minimalization pressure will not permit feature-rich MEs in the near future. Hence, they must be able to connect to local devices such as memory, processors, displays, and network devices in an ad hoc manner. We call this process association and we call such devices *ubiquitous associable objects* (US). Through association, the ME can personalize the US to suit the user's preferences and needs. For privacy reasons, any personalization of an US must become unavailable if it is out of range of the user's ME device.

IT (smart ITem)

There are also numerous smart items that do not support association that would turn them into an US. Vending machines, goods equipped with radio frequency IDs, and landmarks with "what is" functionality are just a few examples. We define such *smart items* as ITs. An IT is any digital or real entity that has an identity and can communicate with the ME. Communication may be active or passive; memory and computation capabilities are optional.

WE (Wireless group Environment)

We expect ad hoc networking to be restricted to an area near to the user of a ME device, as connections with remote services will involve a non-ad hoc network infrastructure. The functionality of a WE is to bring to-

gether two or more personal environments consisting of a ME and arbitrary US entities each. A WE makes connections between the devices possible and also allows for sharing and transferring hardware (e.g., US devices) and software or data between WE users.

THEY (Telecooperative Hierarchical ovErlaY)

We regard overlay cells as the backbone infrastructure of MUNDO. These *telecooperative hierarchical overlay cells* (THEY) connect users to the (non-local) world, deliver services, and data to the user. THEY support transparent data access, for example, frequently used data may be cached on US devices. THEY offer cooperation between different physical networks, transparent to the users.

Communication in MUNDO

Even though users and their activities are the focus of MUNDO, through their MEs, the majority of the communications in MUNDO will happen between the different devices. Examples of this are an US associating automatically with a ME, a WE forming for sharing information, an US communicating with another US, and so on. It is therefore vital that we have a strong basic communication layer which will allow us to connect all the components of Mundo in an efficient manner.

3 MundoCore

MundoCore can be considered as a peer-to-peer technology. It features automatic discovery of other nodes, autoconfiguration and routing. The graph of connected MUNDO-aware nodes forms an overlay network over the existing Internet infrastructure and private network links.

3.1 Concepts

The basic communication layer is used for communication between local services and applications. It interconnects components residing in the same address space and operating system process. The communication to remote Mundo nodes on the same machine or on another machine that is reachable by IP or over a tunnel connection, relies on a number of transport and routing services. Nodes on the same physical machine can communicate via shared memory and remote nodes, for example, via IP connections or bluetooth RFCOMM tunnels.

In the following, we use the terminology introduced in [4] for Publish/Subscribe systems. The single programming abstraction on the basic level is that of a channel-based Publish/Subscribe system. Even remote method invocations build on top of this concept. It has been shown in many earlier projects [7] that event routing systems scale well to very large networks.

MundoCore consists of a basic communication layer, a few base services and a number of interface definitions with associated semantics. Specific functionality is implemented by pluggable services.

The current implementation is based on C++ and supports several platforms, including desktop/server Windows, Windows CE, Mac OS/X and Linux/uClinux. An installation with a minimum set of services and full XML processing requires only 88 KB. We are also working on a Java implementation of the system that is compatible on the protocol level.

3.2 Services

Services in MUNDO are components that implement a certain functionality by processing messages from channels or their methods can be invoked by means of remote method calls.

A service is the **migration unit** and manages objects that are closely related to each other. References to local and remote objects are explicitly distinguished by different types of reference variables. Services are not bound to a particular physical location; they can be migrated from one address space to another. The communication abstractions provided by MundoCore enable services to maintain all communication links to other services even if they are moved to other physical locations.

The service **manages execution**. By means of *sessions*, incoming remote method calls are serialized to control concurrency. A session is a single-threaded context for producing and consuming messages, like in JMS [6]. The service can also be thought of as a layer between the runtime environment and the objects that controls execution. The last two aspects are inspired from the virtual processor concept described in [3].

Multiple services can offer the same interface and realize the same functionality, but rely on services in the network to different grades. We call this concept **service fidelity**. On devices with very limited processing power, simple services run on the device that mainly delegate all tasks to more powerful services in the network. If no network infrastructure is present, the quality of a service may get limited to a basic set of functions. For example, a voice recognition system running on the

ME-device may be limited to understanding a dozen of core commands; once the device has a network connection, more complicated commands are recognized by a network-hosted full fledged voice recognition system and the device may also be used for dictating letters, mails, etc.

Services usually realize their functionality by delegating subtasks to other services. In most cases, the Publish/Subscribe metaphor provides good decoupling between such services. If services explicitly use directed connections to other services, a Lease concept [6] ensures that the services are aware of broken links and that allocated resources on remote nodes are freed.

3.3 Extensions to Publish/Subscribe

A Publish/Subscribe system offers a clear advantage for point-to-multipoint communication patterns. We have optimized this mechanism to allow us to use the same abstraction for point-to-point connections. Specific communication patterns are expressed in attributes of the channel objects. The messaging middleware makes use of this information and can improve routing efficiency. Further extensions include zones, message queuing, streaming of multimedia data and remote method calls (RMC) built on top of the Publish/Subscribe system.

The scopes of channels are limited and organized into zones. Scoping ensures that subscribers do not receive event notifications even though they may be subscribed to relevant channels, unless the event has been propagated to a zone of which they are a member. Zones may overlap allowing publishers and subscribers to be a member of two or more zones. Typically one zone hierarchy is not enough. An organizational hierarchy is well suited for security aspects, but a hierarchy based on geographical location is better suited for distributing context information. The concept of zones first appeared in the ECO system [8].

In subscription-based Publish/Subscribe, a subscription has to be flooded to all network nodes in the respective zone. But in the case of a directed connection, like between an RMC client and server stub, the used channel has exactly one publisher and one subscriber. If this constraint is known to the messaging middleware, routing can be significantly optimized. For directed connections, the advantage of the Publish/Subscribe system comes into play, when a service is moved to another location:

- First, all subscriptions are closed at the source location. Like with Durable Subscribers [6], further messages will be queued.
- Then, all objects managed by the service are serialized and sent to the target node.
- The service is now started on the target node and initialized with the serialized object graph.
- Finally, all subscriptions are opened again at the target location.

3.4 Basic Services

Figure 2 shows the basic communication services arranged into a layer model.

Network Access Services communicate with access points, configure the network device settings of the underlying operating system and establish connections. For example, such a service performs dial-in to an ISP via a cellphone or maps a bluetooth serial connection between the client and access point so that a transport connection can be created on top of this serial link.

Transport Services connect adjacent nodes via a tunnel or use an underlying routing-capable transport system like IP to connect endpoints. Besides point-to-point connections, transport services can directly provide point-to-multipoint communication via broadcasts and multicasts.

The bandwidth of wireless networks is scarce, but because it is a shared medium, it is possible to send broadcast messages to all nodes in the vicinity. Therefore, it is highly desirable to have the multicast functionality in lower transport layers and aspects like encryption in higher layers. In contrast, the Bluetooth LAN access profile makes use of encrypted point-to-point tunnel connections to the access point and makes the low level multicast functionalities completely unavailable. This means that IP broadcast packets have to be replicated for each client and individually transmitted on the shared medium.

The transport service allows us to insert additional transformation steps like encryption or data compression for certain connections. A transport service may also feature automatic discovery of neighboring nodes. If neighbors are detected, this information is reported to the Routing Service.

The **Routing Service** uses Transport Services to create connections to remote nodes when needed and picks the most suitable Transport Service to route a message

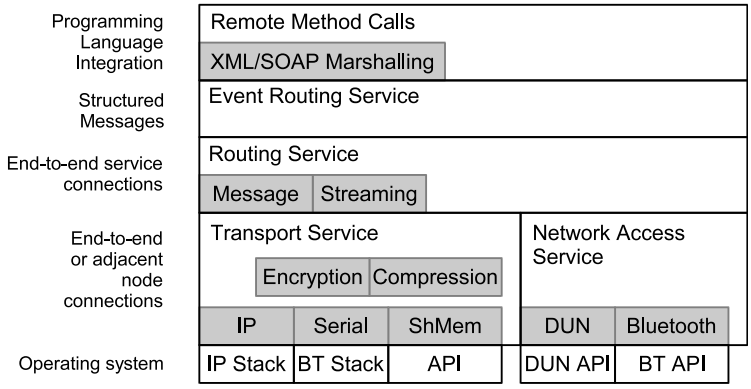

Figure 2: Layer model

towards its destination. Compared to solely IP based routing, private network links can be opened on demand, based on cost models and Transport Services that provide circuit-switched connections can be integrated to provide enhanced QoS.

The **Event Routing Service** (ERS) handles all subscriptions and advertisements of local services that are not restricted to the local runtime zone. It discovers Event Routing Services on neighboring nodes and performs message routing. A minimal ERS on a device with limited resources simply forwards all messages to the ERS of an access point. More sophisticated ERS in the MUNDO overlay network can perform filter merging and content-based Publish/Subscribe with complex XML-based filtering.

3.5 Remote Method Calls

The RMC functionality is also built on top of the Publish/Subscribe system of the basic communication layer. To export an object, a server stub is created and subscribed to a channel.

Server and client stubs are created by our own compiler that parses the class definitions found in C++ header files. Additional processing instructions can be passed to the compiler by means of special comment tags in the header file. Remote methods can be called in a synchronous, asynchronous, or oneway manner. Oneway calls are also suitable for point-to-multipoint communication patterns.

Calls to remote services within the same address space can be performed with a minimum of glue code. A serialization of the call parameters and return values is not necessary. Our RMC implementation is currently based on XML/SOAP, but is generally not fixed to a single Inter-ORB protocol. Currently there are two ways to extend the system. Firstly, the RMC compiler can be extended to create different client and server stubs. Secondly, a new service can be instantiated that translates from the XML/SOAP to another protocol.

4 Experiences and Observations

From our experiences in implementing MundoCore, we have made several observations regarding practical matters.

Even though MundoCore is aimed at communication between devices, we still need some kind of a user interface for debugging and verifying correct behavior. In case of embedded devices, this may not always be easy to achieve.

However, the basic abstraction of a Publish/Subscribe system helps us develop GUI tools for administration and connecting services. This is because all the services have well-defined and clear interfaces and the channels of the Publish/Subscribe system form a connecting network that can be mapped to a graphical representation.

Current wireless networks still have several problems that prevent large-scale deployment of pervasive computing communication infrastructures. MundoCore requires a robust and reliable wider area wireless multicast functionality which is not available yet. Our initial experiments with existing reliable multicast libraries have not been fully satisfactory. Most systems do not

deal well with a dynamically changing network. We are currently working on developing efficient methods for multicast using MundoCore. In addition, wide area wireless bandwidth (e.g., GPRS) is currently relatively scarce and very expensive. For these reasons, we plan to investigate implementing more size-efficient binary protocols beside XML/SOAP.

Java runtime environments on embedded devices are also severely lacking in capabilities, making it impossible to write code that would work efficiently in all environments. This is the main reason why our current implementation is done in C++. In our experience, porting Java code to embedded devices takes as much work as porting C++, and performance of Java is far from satisfactory. Some services like the Network Access Services require access to special operating system APIs that are not accessible from Java. This means, that a Java implementation will also have small C++ parts using the Java Native Interface.

We have already implemented parts of MundoCore in Java that is compatible to the C++ implementation on the protocol level. Because of the higher resource requirements, the Java version mainly targets desktop and server applications. Programming Java is more convenient than programming C++, because the language features a garbage collector. On the other hand, freeing operating system related resources needs special attention in Java.

References

[1] Erwin Aitenbichler and Max Mühlhäuser. The Talking Assistant headset: A novel terminal for ubiquitous computing. Technical report, TK-02/02, Telecooperation Group, Department of Computer Science, Darmstadt University of Technology, 2002.

[2] Christian Becker, Gregor Schiele, Holger Gubbels, and Kurt Rothermel. BASE - A Micro-Broker-Based Middleware for Pervasive Computing. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom'03)*, pages 443–451, 2003.

[3] M. Boger, F. Wienberg, and W. Lamersdorf. Dejay: Unifying concurrency and distribution to achieve a distributed Java. In *Proceedings of TOOLS Europe '99, Nancy, France*. Prentice Hall, June 1999.

[4] A. Carzaniga. *Architectures for an Event Notification Service Scalable to Wide-area Networks*. PhD thesis, Politecnico di Milano, Milano, Italy, December 1998.

[5] Andreas Hartl, Erwin Aitenbichler, Gerhard Austaller, Andreas Heinemann, Tobias Limberger, Elmar Braun, and Max Mühlhäuser. Engineering multimedia-aware personalized ubiquitous services. In *IEEE Fourth International Symposium on Multimedia Software Engineering (MSE 2002). Newport Beach, California*, pages 344–351, 2002.

[6] Sun Microsystems Inc. The Java Message Service 1.0.2b Specification. Available for download at http://java.sun.com/products/jms/, October 2001.

[7] René Meier. State of the Art Review of Distributed Event Models. University of Dublin, Trinity College, March 2000.

[8] Karl O'Connell, Tom Dinneen, Stephen Collins, Brendan Tangney, Neville Harris, and Vinny Cahill. Techniques for Handling Scale and Distribution in Virtual Worlds. In *Proceedings of the* 7^{th} *ACM SIGOPS European Workshop*, pages 17–24, 1996.

[9] Manuel Román, Fabio Kon, and Roy H. Campbell. Reflective Middleware: From Your Desk to Your Hand. IEEE Distributed Systems Online Journal, Special Issue on Reflective Middleware, 2001.