

# Network middleware and Mobility\*

P.H.F.M. Verhoeven, J. Huang, J.J. Lukkien  
Eindhoven Embedded Systems Institute  
Eindhoven University of Technology  
P.O. Box 513, 5600 MB Eindhoven, The Netherlands  
Tel. +31 40 2478207, Fax. +31 40 2451733  
Email: {p.h.f.m.verhoeven,j.huang,j.j.lukkien}@tue.nl

**Abstract** *Within PROGRESS project WebES we look at connecting embedded systems to the Internet. Part of the project concerns issues related to mobility and mobile communication. We look at small footprint implementations of Corba specifically destined for mobile terminals and compare several different implementations in that respect. In addition, we look at the issue of modelling systems containing wireless connections.*

*Keywords:* mobility, middleware, interface, protocol

## 1 Introduction

Within PROGRESS project WebES<sup>1</sup> (Web enabled Embedded Systems) at EESI we look at connecting embedded systems to networks, in particular, to the Internet. One specific issue concerns mobility. Mobility refers to the fact that devices can easily move to different physical locations. It is not the same as wireless although a wireless communication facilitates mobility. Examples of mobile systems within this context are notebook computers, handhelds (PDA's), mobile phones or just any device equipped with a Bluetooth connection.

With respect to networking, mobility introduces challenging issues in specification, design and realization. Moving to a different location implies that a new connection has to be established at the new location, preferably without user interven-

tion. For a notebook computer that gets suspended while moving it may be acceptable to go through the same stages as when starting it. Connections are rebuilt at the new location. However, when the moved system remains running ("roaming"), requirements are much stronger: first, the existing connections should remain intact; second, when the systems runs a real-time application like streaming audio or video, reconfiguring should be quick enough to sustain this. A third class of problems pops up when the system moves from one network technology to another, e.g., from a wireless LAN environment into the GSM network. In all three cases it is a relevant question at which level these issues should be dealt with.

In this paper we present current work at EESI in relation to mobility. In the next section we discuss network middleware in general and how the mobility issues are addressed. In section 3 we present our analysis and comparison of several CORBA implementations. In section 4 we discuss the work we did on our implementation of the Session Initialization Protocol (SIP). Finally, we address the work we are starting on *modelling* systems containing wireless connections.

## 2 Mobility in network middleware

Looking at networking from a software perspective, we distinguish four functional layers<sup>2</sup>: physical, transport, interoperability and application layers, see Figure 1. The first two layers together form the communication infrastructure, supporting information transfer between network nodes. On the

---

\*To be published in the proceedings of the PROGRESS workshop in October 2001

<sup>1</sup>The WebES project has been assigned funding in spring 2001 so it started officially only a few months ago. It is sponsored through PROGRESS by STW, Philips and TNO. The project has been running in a smaller setting for one and a half year.

<sup>2</sup>This is, of course, a simplification from the point of view of the technical realization of networks. However, for our discussion this separation is more clear.

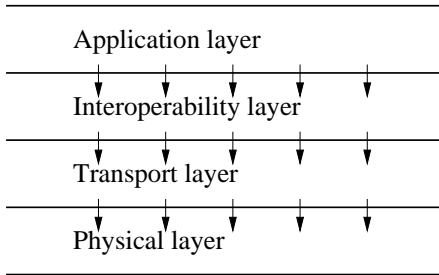


Figure 1: A simplified picture of the functional layers in a network. The physical and transport layer together form the communication infrastructure that admits transport of information. The interoperability layer represents the way applications use the transport layer. It provides services to the applications.

Internet it is represented by the TCP,UDP protocol layer. “Middleware” comes up within the interoperability layer. The term “middleware” is actually applied in two different ways. In one interpretation it concerns all manipulation of network traffic that changes the semantics of the transport layer. Examples include firewalls, proxy services and network address translation (NAT). For a discussion see e.g. [1]. In the second interpretation it is the software that resides between the transport layer and the application, see e.g. [2]. (Notice that these two references have the same name!) We will use the term middleware in the second interpretation: middleware implements the interoperability layer. For an application, the interoperability layer is the means to use the communication infrastructure; middleware provides services to applications.

In the current Internet the used middleware is often not a layer but rather a collection of protocols placed as “blocks” on top of the transport layer. The transport layer then is not hidden but remains accessible (see Figure 2). In fact, the protocol API is often not standardized. The used protocols are intimately related to the Internet. A more complete interoperability layer hides all details of the transport layer. Quite a number of techniques have been introduced for this purpose, like Remote Procedure Calls (RPC’s), Transaction and message oriented middleware, distributed DBMS’s and object oriented middleware of which the Object Request Brokers are most dominant (see e.g., [3]). Advantages of the use of middleware include the following.

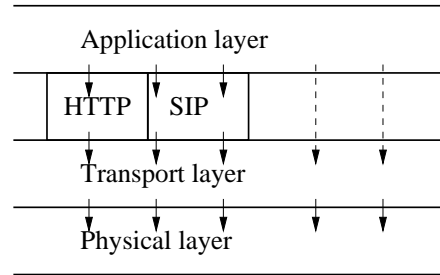


Figure 2: Applications on the Internet often communicate using fixed protocols of which HTTP is an example. The transport layer is not hidden but can be reached directly by the application (the dashed arrows). Instead of forming a layer, these protocols form separate blocks.

- A simplification of distributed programming is achieved by simplifying the logical structure.
- It is straightforward to use heterogeneous systems (e.g., combining different programming environments or different hardware architectures).
- Legacy software can be encapsulated.
- It gives support for separating a system into relatively independent components thus supporting the easy distribution of functionality. The client-server approach is a successful example of this. Recently, the peer-to-peer architecture has become popular as well.
- Abstraction from the transport layer is provided. In fact, no uniform transport layer is required.

Mobility is addressed within all four layers. Mobile IP, [4], aims at hiding mobility within the communication infrastructure. Traffic is re-routed to the new location of the device through agents at the home location and the new location of the device that set up a tunnel. The next version of IP, IPv6, has incorporated mobility even more effectively.

Solving the mobility problem within the communication infrastructure has as advantage that higher layers do not need modification. However, generic middleware like CORBA supports different infrastructures and does not rely on a complete

transport layer. Within the wireless CORBA implementation a software bridge is built around the wireless link which could be based on a protocol different from IP. This works across Bluetooth or infrared connections that solve the mobility problem internally. The mechanism supports, in fact, connecting two Object Request Brokers through an arbitrary tunnel.

Mobility also plays a role at the application level. A *user* of a network terminal may be moving to another terminal wishing to continue his work at the new location. A user may also wish to divert network traffic destined for herself to another place. An example is the “follow me” functionality of a telephone connection. This aspect of mobility is addressed by the Session Initiation Protocol (SIP, [5]).

These three cases show that on the one hand it is advantageous to hide the mobility aspect inside a layer while, on the other hand, the mobility information is needed at several layers. This is true in particular for any application that wishes to use mobility information, for example to optimize the use of resources or to behave differently as a result of movements.

### 3 Comparison of CORBA implementations

Mobile terminals, like mobile phones and PDAs interact at the interoperability level, leading to the use of distributed middleware. Several middleware solutions are readily available, like CORBA, DCOM, Java RMI and XML/SOAP. However, these solutions are often targeted at enterprise solutions and not at mobile terminals with limited resources. Furthermore, the middleware has to provide both mobility hiding as well as mobility awareness to the application layer, as the intention of the middleware is to hide the architecture and provide a common interface to the application. For example, for a ticket service, the mobility of the terminal is largely irrelevant, while for a guided tour in a museum, the mobility and location awareness is vitally important.

In ITEA project “Vivian” [6], a common platform for mobile terminals is being specified by Nokia, Philips and several research institutes. The selected Operating System platform is EPOC [7]. As a middleware solution, CORBA [8] is selected

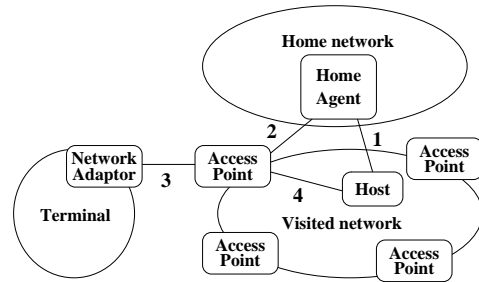


Figure 3: *Network mobility. Mobile IP uses route 1-2-3-3-4, wireless CORBA uses route 4-3-3-4.*

to handle the issues of distributed computing and service discovery. To handle the mobility related issues, the Telecom task force of the Object Management Group has recently defined an extension to CORBA to handle wireless access and terminal mobility [9]. This extension also describes how events related to mobility are reported to interested applications.

Although the CORBA specification is largely independent of the transport protocol that is used, in practice, it is mainly based on TCP/IP. In particular, it requires the network endpoints of the connection be fixed during a CORBA session. As such, the mobile IP solution could solve the mobility issues, as it transparently routes all IP traffic to the mobile terminal. However, there are some issues which make mobile IP less useful. In mobile IP, all IP traffic is routed through the home location agent, which might be located in a remote network, as shown in figure 3. As a result, the network route can become very long, even though the mobile terminal is within the same network. For a distributed application based on CORBA, this can result in unacceptable delays. Furthermore, the mobile IP specification states that it solves the macro mobility (that is, the mobility between home and office) and is not well suited for micro mobility with quick handovers between devices with short ranges, like BlueTooth.

The wireless CORBA specification provides a different solution to the mobility issues, which is transparent for existing CORBA implementations. Instead of sending all IP traffic through the home location agent, the home location agent is used to retrieve the current access point of the mobile terminal, which is then contacted directly. This reduces the length of the network route, but it creates a problem when the mobile terminal changes to a

different access point while a CORBA connection still exists. This problem is solved by requiring that an old access point forwards the relevant traffic to the current access point.

Although the wireless CORBA solution differs from mobile IP and requires additional resources in a mobile device, the solution is justifiable for the applications in mind. The target applications are mainly data services in the visited network, which should also work if the mobile terminal is homeless, like a PDA. Typical scenarios are shops with special offers, airport terminals with up-to-date information, ticket selling in theatres and guided tours in museums. None of these require external network connectivity and could in fact be handled over Bluetooth, InfraRed or Wireless LAN. For such local applications, the CORBA solution is preferable to mobile IP. However, the CORBA solution is very specific to CORBA itself, although it can be used as an example on how such a protocol could work. It should not be the case that every application develops its own protocol for handling mobility, as it would require too many resources in a mobile device.

When a handover or access recovery occurs, the standard CORBA services can be used to inform the applications that are interested in the mobile behavior of the terminal. Examples of such applications are the virtual tour in a museum, that presents additional information on the pieces in the current room, or an airport assistant that gives directions to the correct gate.

The CORBA implementation depends on the transport layer to provide sufficient information that it can determine that such an event took place, which could be achieved with interrupts, monitors or polling. Due to the information hiding by the different layers in the communication stack, the transport layer provides no standard interface to get such events easily.

For the Vivian project, hands-on experience with wireless CORBA and EPOC is required, in the light of the resource restrictions of mobile terminals. Therefore, a survey of several freely available CORBA implementations is made to compare the source code, the resource requirements, the performance and the interoperability. The survey includes four Java ORBs (JacORB, ORBacus, OpenORB and Engine Room), four C++ ORBs (MICO, ORBacus, omniORBA and TAO), one C ORB (ORBit) and one C++ ORB that is optimized for em-

bedded systems (ORBacus/E). As expected, the survey showed that huge differences exist between different ORB implementations. Some of the differences are due to the different features that are implemented by the different ORBs. Some ORBs don't support portable object adapters, some don't support the interface repository, some don't support interceptors. All of these features require resources and additional processing time. However, in a minimal setting, it is possible to be interoperable without these features, as specified in the latest CORBA specification. Figure 4 shows the performance analysis of the ORBs that were tested, while figure 5 shows the sizes of different ORBs.

Some of the preliminary results:

- The source code size for the standard ORB varies between 17.000 and 153.000 lines of code. The smallest ORB does not include any features, while the largest ORB includes real-time support.
- The library resource size varies between 562 kilobyte and 11.3 megabyte. Again, the one that is smaller doesn't provide any special features.
- The performance varies between 7500 calls per second and 260 calls per second, depending on the combination of ORBs that is used for the client and the server.
- The fastest Java combination is considerably slower than the fastest C++ combination.

The performance measurements were done on a Linux system with sufficient resources. As Vivian selected the EPOC operating system for the mobile terminal platform, a similar performance measurement under EPOC still has to be done for comparison, where the Java virtual machine for EPOC helps in evaluating some of the Java ORBs, to get an impression on the performance impact of the limited resources and reduced processor speed. In a later stage, a C++ ORB will be ported to EPOC to get more accurate numbers on the possible performance.

In the future, the wireless CORBA specification will be implemented and tested in scenarios available from the Vivian project. A demonstrator will be build to show the possibilities of distributed computing with mobile terminals.

Client	Server	JacORB	MICO	ORBit	ORBacus C++	ORBacus Java	ORBacus/E	omniORB	OpenORB	Engine Room	TAO	Average
JacORB		37.4	30.0	11.8	33.2	35.7	11.1	12.7	71.7	22.2	25.7	29.2
MICO		36.6	21.5	11.4	—	—	11.5	12.7	70.7	20.1	21.9	25.8
ORBit		30.4	15.9	4.4	20.1	28.0	5.0	5.9	66.7	12.3	13.0	20.2
ORBacus C++		35.0	24.5	10.7	26.4	32.8	9.9	11.0	69.6	18.5	21.6	26.0
ORBacus Java		38.1	30.1	16.2	34.2	35.9	15.5	16.5	75.1	20.9	27.8	31.0
ORBacus/E		27.5	16.4	4.7	19.7	24.7	4.0	4.8	62.4	10.5	13.2	18.8
omniORB		28.8	17.6	5.7	20.8	25.1	4.9	5.1	64.6	11.6	14.9	19.9
OpenORB		—	70.9	63.1	105.3	77.4	62.1	63.2	115.6	71.2	88.5	79.7
Engine Room		35.4	27.2	13.7	—	—	13.2	14.1	—	18.5	—	20.4
TAO		39.9	29.4	15.2	34.0	39.0	15.5	16.5	75.1	23.8	23.1	31.1
Average		34.3	28.3	15.7	36.7	37.3	15.3	16.2	74.6	22.9	27.7	30.2

Figure 4: *The interoperability performance test results, in number of seconds for 30.000 inter-ORB calls.*

ORB	language	CORBA version	lines of code	compressed binary	uncompressed binary	account client	appl. server
JacORB 1.3.21	Java	2.3	50K	4034 KB	8697 KB	9 KB	10 KB
MICO 2.3.5	C++	2.3	62K	1235 KB	4715 KB	51 KB	55 KB
ORBit 0.5.7	C	2.2	21K	175 KB	562 KB	6 KB	6 KB
ORBacus 4.0.5	C++	2.3	89K	2188 KB	9574 KB	64 KB	81 KB
ORBacus 4.0.5	Java	2.3	58K	1704 KB	3355 KB	10 KB	10 KB
ORBacus/E 1.0.1	C++	2.3	17K	422 KB	1735 KB	50 KB	
server				593 KB	2458 KB		61 KB
omniORB 3.0.3	C++	2.3	30K	310 KB	1003 KB	19 KB	21 KB
OpenORB 1.1.0	Java	2.4	40K	1796 KB	3713 KB	9 KB	10 KB
Engine Room 1.4-1	Java	2.2	7K	181 KB	346 KB	8 KB	9 KB
TAO 1.1	C++	2.3	220K	2863 KB	11299 KB	37 KB	61 KB

Figure 5: *The object sizes of the different ORBs, with compressed and uncompressed sizes for Java JARs and C++ libraries.*

## 4 The Session Initiation Protocol

SIP is defined by the IETF as an application-layer control protocol that can establish, modify and terminate multimedia sessions or Internet telephony calls. It transparently supports name mapping and redirection services, allowing the implementation of ISDN and Intelligent Network telephony subscriber services. These facilities also enable personal mobility, which is defined as the ability of end users to originate and receive calls and access subscribed telecommunication services on any terminal in any location, and the ability of the network to identify end users as they move. Personal mobility complements terminal mobility.

SIP is a text based protocol, with messages similar to HTTP. With the requests and replies, it is possible to handle all the scenarios that are common in telephony call handling. In a normal situation, the caller send an INVITE request to the callee with a description of the supported multimedia sessions. The callee either accepts the invitation and send a reply with the selected multimedia session or rejects the invitation and send a reply with the reason. Once the session is established, both the caller and the callee are able to change or terminate the session.

Unlike the H.323 protocol defined by ITU, SIP addresses users at terminals instead of just terminals, which allows multiple users at the same terminal. For example, a secretary might react differently depending on whether the call is private or business.

Similar to HTTP, the terminal can reply with the redirection to indicate that the requested user has moved. For example, when you go to work, you can redirect your calls to the terminal at work, without redirecting calls for other family members.

With the REGISTER request, it is possible to modify the redirections that the terminal will use. This is useful if you forgot to redirect your calls before you went to work. Depending on the exact interpretation, it is also possible to redirect the call while the terminal is ringing, thereby allowing remote call pick-up.

Unlike HTTP, SIP supports preliminary responses, which are used to indicate progress. While the terminal is trying to contact the user, the caller has to have some feedback on the progress, for example whether the phone is working. It is also possible that the terminal supports queuing of incoming

calls, where the caller has to be informed of the position in the queue and possibly the expected time to wait.

In an investigation into Internet Telephony, an example implementation of SIP was build to get familiar with the issues and to demonstrate the features that are possible with it. In the demonstrator, most of the features mentioned above were implemented, although the actual transportation of multimedia content was not added.

## 5 Modelling wireless systems

A (digital) physical system can be described as a collection of communicating components. Such a description is usually hierarchical. The purpose of this *system level* description is to support architectural exploration and analysis. This comprises evaluation of alternatives in terms of hardware and software realizations, choices of building blocks and connections as well as verification of correctness and performance criteria.

A system level design is described in some modelling language with a precise mathematical semantics in order to support mathematical reasoning. Such a semantics can be in terms of CCS ([10]), CSP ([11]) or some other process algebra. At Eindhoven University the language POOSL has been developed[12]. POOSL integrates a process part with a data part, which are both based on a formal semantics. The semantics of process part is based on real-time CCS and that of the data part is like traditional object-oriented languages (e.g., Smalltalk or Java). This supports a rather straightforward modelling of systems of communicating components. Some examples can be found in [13, 14].

The advantage of a formal system-level specification is that it is unambiguous and consistent. Analysis at this level is helpful for obtaining insight in system behavior and for finding design flaws already in an early stage. Crucial for the succes of a method is that it equips the designer with an easy to read and understand modelling language such that a specification becomes clear and concise.

The current way of modelling supports the description of static systems, i.e., systems of which interconnections are fixed. We want to extend this in order to describe dynamic systems of which the connectivity changes over time. The current modelling language allows us to specify dynamic change

but with a rather large notational overhead which makes the specification all but intuitive. Rather than trying to “bend” the language we want to introduce new primitives that describe dynamic connections more satisfactorily. Not only the physical location is taken into account but rather a more general state-dependence approach admitting, for example, to include signal strength and interference.

The new primitives also call for an extension of the semantics. Over the last decade several theories have been proposed. Examples are the  $\pi$ -calculus[10]; the ambient calculus[15] and Mobile and Dynamic Petri-nets. These theories give mathematical frameworks to reason about mobile system and have been applied, for example, to verify the correctness of mobile agent systems. However, it is still a challenging research issue to fill the gap between these theories and industry practices, in particular when it comes to system analysis (e.g. performance questions). Current research is focussed on additions of the  $\pi$ -calculus to accommodate this.

## 6 Conclusion

In this paper we have summarized work at EESI with respect to mobility. Mobility appears to play a role at several levels in the communication. On the one hand, mobility is hidden from the higher levels in order to support transparency. On the other hand, knowledge about the current location is required for correct operation. We have been looking at the interoperability layer and, more specifically, at the penalty of several Corba implementations in terms of memory requirements and performance. At the application layer we have been looking at the Session Initiation Protocol that allows the user to redirect network traffic explicitly. We intend to use both as part of a demonstrator.

Finally, for supporting the *design* of systems with dynamic connections we are developing the necessary theory and tooling support.

## References

[1] G. Huston. The middleware muddle. [www.cisco.com/warp/public/759/ipj\\_4-2/ipj\\_4-2\\_middle.html](http://www.cisco.com/warp/public/759/ipj_4-2/ipj_4-2_middle.html).

[2] D. Ritter. The middleware muddle, May 1998. [www.dbmsmag.com/9805d14.html](http://www.dbmsmag.com/9805d14.html).

[3] P.A. Bernstein. Middleware, a model for distributed system services. *Communications of the ACM*, 39(2):86–98, February 1996.

[4] C.E. Perkins. *Mobile IP, design principle and practices*. Addison-Wesley, 1998.

[5] Rfc 2543.

[6] [www.nrc-nokia.com/Vivian](http://www.nrc-nokia.com/Vivian).

[7] M. Tasker. *Professional Symbian Programming: mobile solutions on the EPOC platform*. Wrox press ltd., 2000.

[8] The Common Object Request Broker Architecture and specification, revision 2.0, July 1995. [www.omg.org/corba.corriop.htm](http://www.omg.org/corba.corriop.htm).

[9] Borland, Highlander, Nokia, and Vertel. Wireless access and terminal mobility in CORBA. Technical Report dtc/2001-06-02, OMG, May 2001.

[10] R. Milner. *Communicating and mobile systems: the  $\pi$ -calculus*. Cambridge University Press, 1999.

[11] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[12] P.H.A. van der Putten J.P.M. Voeten. *Specification of reactive Hardware/Software systems*. PhD thesis, Eindhoven University of Technology, Eindhoven, the Netherlands, 1997.

[13] M.P.J. Stevens B.D. Theelen J.P.M. Voeten P.H.A. van der Putten H.J.S. Dorren. Modelling Optical WDM Networks using POOSL. In *Proceedings of ProRISC'99*, pages 503–508. STW Technology Foundation, 1999.

[14] B.D. Theelen J.P.M. Voeten P.H.A. van der Putten A.M.M.Niemegeers G.G.de Jong. Performance modeling in the large: A case study. In *Proceedings of the European Simulation Symposium (ESS) (to be published)*, 2001.

[15] L. Cardelli A.D. Gordon. *Mobile Ambients*, pages 140–155. Cambridge University Press, 1998.