Ubiquitous Computing Environments through Open Systems

A. Puder

Deutsche Telekom AG
400 South El Camino Real, Suite 1550
San Mateo, CA 94402
arno.puder@telekom.de

Abstract. With the growing trend of making information and services available online, one of the challenges for ubiquitous computing environments will be to provide access through handheld devices. The complexities involved can best be met by an *open system*. This paper provides a novel approach to interpret an open system. We then present an open platform based on CORBA which bridges the gap between handheld devices and online services. The architecture of a prototype is described where an open source implementation of CORBA has been ported to the PalmPilot.

Keywords: Open System, Open Source, CORBA, Handheld Devices.

1 Introduction

One trend of a modern information society is that information and services for everyday users is coming online. The World–Wide Web (WWW) serves as an enabling technology to provide easy access to the information and services. With the growing success of handheld devices, the demand to have access through these devices increases. Some of these devices, such as the Palm 7, already include a wireless modem which underlines this trend. This makes it increasingly attractive to provide online access for legacy applications such as SAP/R3 as well, e.g. to allow sales persons with immediate access to cooperate services independant from their current location.

The challenge for an ubiquitous computing environment will be to connect handheld devices with legacy applications (see Figure 1). Currently there is a gap which makes it difficult to access online information from handheld devices. First of all handheld devices are often still missing wireless capabilities which will change in the near future. But even with appropriate wireless add–ons, access to online information from handheld devices is limited to web content. This is usually done using a *micro–browser* approach which simply shrinks the well–known web browser interface to fit on a handheld device.

We believe that the user interface for handheld devices has to follow different paradigms than those for the desktop simply because the user behaviour patterns for handhelds and desktops are very different as are their I/O capabilities. Especially access to legacy applications demand native interfaces which follow the user interface paradigm of a handheld device. The web was also meant for

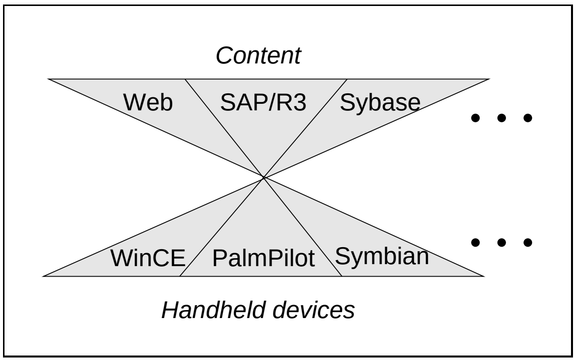

Fig. 1. Ubiquitous computing environment.

providing access to information, not for interaction with services. Operational interaction is already poorly supported from a desktop browser; from a handheld device using micro–browsers it will become infeasible.

Bridging the gap between handheld devices and online information to create an ubiquitous computing environment requires new approaches through what we call an *open system*. An open system manifests itself on different levels, ranging from standardization adoption processes to software development. It is important to acknowledge the diversity and complexity of existing technologies and legacy applications when seeking new solutions.

This paper is divided in two major sections. Section 2 provides a characterization of an open system from different perspectives. Based on the analysis, Section 3 describes an open platform to support ubiquitous computing environments including handheld devices. Finally, Section 4 concludes this paper with a brief summary and an outlook.

2 Characterization of Open Systems

Ubiquitous computing environments which embrace a diversity of technologies introduce new levels of complexity. No single vendor can solve the problems which have to be faced because the nature of ubiquitous computing demands cross platform solutions. The central claim of this paper is that this requires a paradigm shift from closed to open systems. This section establishes the term open system as used for the remainder of this paper. When reasoning about open systems, different perspectives have to be considered. In the following sections a discussion on technologies with respect to standards (Section 2.1) and software development (Section 2.2) is presented for the context of open systems.

2.1 Technological perspective

Coping with the complexities of ubiquitous computing at a technical level requires to accept the heterogeneous nature of todays systems. Legacy applications need to be integrated and made accessible through handheld devices employing different technologies. It is neither feasible nor possible to prescribe a specific technology and for that reason any attempt to build a homogeneous system is bound to fail. This implies the promotion of technology through open standards and not by pushing proprietary implementations.

A popular solution which facilitates the development of distributed applications and which honours the de–facto heterogeneity is a *middleware architecture*. The term middleware derives from the fact that it is a software architecture which resides between the hardware and the application. A middleware spreads out like a table cloth within a heterogeneous environment, offering the same *Application Programming Interface* (API) at each location of the network, yet embracing different kinds of technologies.

A popular specification of a middleware architecture is CORBA (*Common Object Request Broker Architecture*, (see [1]). It is issued by the *Object Management Group* (OMG), an international, non–profit organization with over 800 information software vendors, software developers, and users. The goal of the OMG is the establishment of industry guidelines and object management specifications to provide a common framework for application development. There are several dozen vendors which provide CORBA technology for different market segments.

CORBA as a standard is derived through an open, technology driven adaptation process. If a need for a standard in a certain domain is recognized, a *Request For Information* (RFI) is issued which helps to determine the scope of the problem. Members and non–members of the OMG can contribute to a RFI. The next step is to issue a *Request For Proposals* (RFP). Only members can make contributions for RFPs and through discussions and a final vote the standard is derived. It is important to note that every member has exactly one vote. This prevents "political" decisions and guarantees that only good technical solutions are adapted as standards.

With respect to the characterization of an open system we propose to employ open standards which have been derived through a transparent, technology driven, open discussion. The general availability of the standard promotes a multiplicity of implementations, tailored for different technological domains.

2.2 Software development perspective

With the increasing popularity of Linux, the term *open source* has become a synonym for a new software development process. The origin of open source is attributed to Richard Stallman, who opposed in the early eighties the growing trend of making Unix proprietary. He started the GNU (**G**NU's **N**ot **U**nix) initiative with the intention to write a free Unix operating system, which eventually let into the Linux operating system.

The essense of open source is that the source code of a project is released to the public, allowing independant peer review. One often used definition of open source was given by Eric Raymond (see [6]):

> *"Open source promotes software reliability and quality by supporting independent peer review and rapid evolution of source code. To be certified as open source, the license of a program must guarantee the right to read, redistribute, modify, and use it freely."*

The essense of the open source model is that it allows independant peer review by other programmers throughout the Internet which greatly enhances the quality of the code. This has also coined the phrase *debugging is parallizable*. Open source projects have an impressive record when it comes to fixing problems. Programmers who report problems with an open source project often not only provide a problem description but also contribute a so–called software patch which solves the problem. The turnaround time from identification to correction of a problem is usually much shorter than with any commercial product (hours/days vs. months).

Open source projects typically have a much shorter release cycle than compared to commercial software. It is not unusual to have several new releases of a project per month. This can only work by using the web as a media to distribute the source code of the project. Commercial vendors have a rigorous and lengthy release cycle to assure the highest possible quality. But beta programs and quality assurance in the commercial world only expose a future product to a subset of all potential customers. The amount of testing is therefore limited. This distinction of beta test and final release is not made with open source projects where problems are usually fixed immediately once they are identified. Commercial product development leads to a slower innovation cycle than compared to an open source project.

It is also argued that open source increases security and reliability because the blue–print of a program is available through the source code. This allows independant inspection and decreases the possibility of "unwanted features" such as Trojan Horses. These unwanted features are much harder to detect in a compiled version of a program. For example, many users of Microsoft's Outlook are not aware that this program contains a hidden game (a complete flight simulator) which can be activated by a certain sequence of key strokes. Especially with the increasing need for security with the rise of ECommerce applications, supporters of open source argue that the availability of source code to the general public make it more difficult for governements to force vendors to include hidden trapdoors.

2.3 Comparison

The essense of an open system is that in order to cope with todays infrastructure problems one must employ a collaborate effort to achieve a common goal. As argued in the previous subsections, it seems unfeasable and probably impossible for one company to solve all problems. In summary, the following attributes are essential for an open system:

– decentralized organization: members are distributed over various physical locations
– distribution channel: in order to cope with the decentralization, an efficient communication media is required.
– open membership: membership is open to anyone who wishes to contribute, independant of their affiliation
– equality: each contribution is evaluated only for its content, not by the member making the contribution
– transparency: no information is kept proprietary; all information is freely available

With respect to this characterization we see similarities between the way the OMG is organized and open source projects. For the scope of this paper we consider both as examples of an open system.

3 Towards an Open Platform

In the remainder of this paper we make a proposal for an open platform which enables next generation ubiquitous computing. This proposal evolves around the issues of open systems as explained in the previous section. We explain how the middleware platform as specified by the CORBA standard (see [1]) can be used to embrace handheld devices. Deutsche Telekom AG is hosting the open source CORBA implementation called Mico (see [5]), which was founded by the author. Section 3.1 will first provide an overview of CORBA. Section 3.2 then explains how CORBA supports ubiquitous computing environments. In Section 3.3 some deficiencies of the current CORBA standard with respect to ubiquitous computing environments are discussed and Section 3.4 finally gives an overview of Mico and the current status quo of the port of Mico to 3Com's PalmPilot.

3.1 Overview of CORBA

One important aspect of CORBA is that it is a *specification* and not an *implementation*. CORBA just provides a framework allowing applications to interoperate in a distributed and heterogeneous environment. But it does not prescribe any specific technology for how to implement the CORBA standard and different vendors can target different markets. The following principles summarize the goals of CORBA:

object orientation: Objects are the basic building blocks of CORBA applications.
distribution transparency: A caller uses the same mechanisms to invoke an object whether it is located in the same address space, on the same machine, or on a remote machine.

hardware, operating system, and language independence: CORBA components can be implemented using different programming languages on different hardware architectures running different operating systems.

vendor independence: CORBA compliant implementations from different vendors interoperate.

The basic building blocks of a CORBA platform are depicted in Figure 2. The central piece of the architecture is the *Object Request Broker* (ORB) which serves as a "software bus" connecting different objects over the network. At the client side the ORB offers a generic API to send operation invocations called the *Dynamic Invocation Interface* (DII). The ORB's task is to locate an appropriate server and to deliver the operation invocation to the server via an *Object Adapter* (OA). The purpose of the OA is to support the life cycle of server objects (e.g., creation and deletion of objects). The OA delivers an operation invocation via the *Dynamic Skeleton Interface* (DSI) which mirrors the behaviour of the DII for the server side.

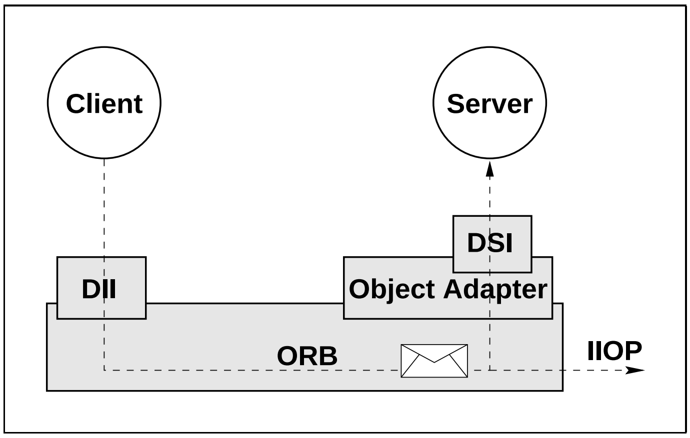

Fig. 2. Components of a CORBA platform.

The CORBA standard also includes an inter–ORB protocol called *Internet Inter–ORB Protocol* (IIOP) which describes the on–the–wire representations of basic and constructed IDL types as well as *Protocol Data Units* (PDU) needed for the protocol. The design of IIOP was driven by the goal to keep it simple, scalable, and general. IIOP uses TCP/IP for transporting operation invocations and their parameters between different ORBs.

3.2 CORBA and ubiquitous computing environments

In CORBA, the *Interface Definition Language* (IDL) describes the interface of an object. Given an IDL specification, a programmer knows how to implement the client or server side of an application based only on the interface specification. In fact, the programmer does not even need to know how operation invocations are handled by the underlying ORB or in which programming language client or server are implemented. For the scope of this paper this interface between application programs and an ORB is called the *horizontal interface* (see Figure 3). The corresponding *vertical interface* is defined through the IIOP protocol (see [4]).

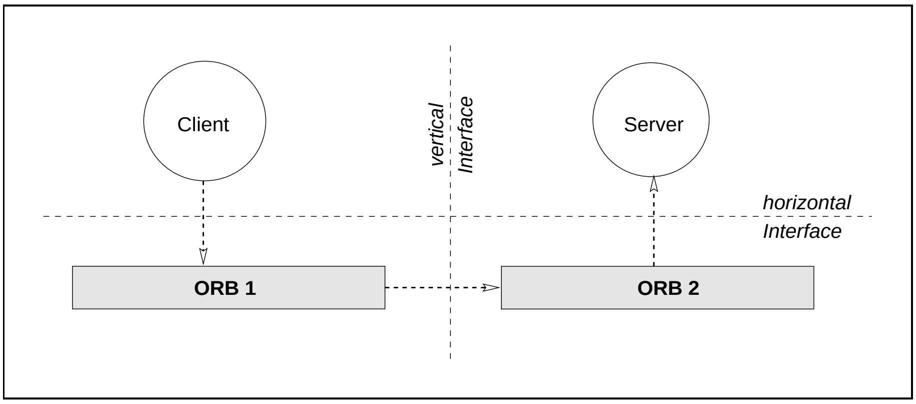

Fig. 3. Horizontal and vertical interfaces in a CORBA environment.

Whereas the horizontal interface guarantees *portability* (i.e., ease portability of applications), the vertical interface guarantees *interoperability* among applications. A compliant ORB must conform to both the horizontal and vertical interface. Under the premise of compliance, it should be possible to distribute an application across different ORBs, exploiting specific features of each ORB implementation. This independence in the choice of implementation is a necessary prerequisit of an open platform. The separation of horizontal and vertical interfaces in a distribution platform supports ubiquitous computing by allowing different vendors to focus on different market segments while guaranteeing interoperability between these segments.

With respect to Figure 3 it is important to point out that ORB_1 and ORB_2 can be from different vendors and can run on completely different kinds of hardware platforms. In one of the next sections this will be shown for the case that ORB_1 runs on a handheld device and ORB_2 on a fixed network. If the domain of ORB_2 is identical with an operator network, one can draw a comparison to the electrical power system. The power outlet defines a standard with respect to physical dimensions and electrical characteristics. This corresponds to a vertical interface. Through plug–and–play different kinds of applicances can be connected to the "information outlet" in the same way IIOP serves as the glue between an operator, retailer and end user. An operator can even offer value added services without having to worry which kind of technology is used on the client side (e.g., desktop running a web browser or a handheld device with a native user interface).

3.3 Deficiencies of CORBA

This section discusses some shortcomings of the current CORBA 2.3 standard with respect to ubiquitous computing environments. While providing an open platform for heterogeneous systems there are some issues which make it difficult to support CORBA on handheld devices. The first issue relates to the limited resources of a handheld device. CORBA in is entirety does not make any assumptions concerning resource restrictions. This clearly is not the case of handheld devices where memory, computing power, external communication bandwith are usually scarce resources. These problems have been recognized by the OMG and a future CORBA standard will include a MinimumCORBA specification (see [2]).

The current draft of MinimumCORBA defines a true subset of CORBA 2.3. It removes some components from the architecture which are not needed for embedded systems, e.g., such as the dynamic components (DII, DSI). There are no restrictions to IDL or IIOP in order to guarantee interoperability with existing CORBA applications. One drawback of the current MinimumCORBA standard is that it does not have separate compliance points for client and server side functionality. For many applications having only client side functionality on a handheld device is sufficient.

Another deficiency of the CORBA 2.3 standard is that is has no support for asynchronous operation invocations. IIOP as defined by CORBA makes the assumption of a reliable, end–to–end connection between client and server object. This is clearly not the case for handheld devices. Frequent disconnects are common because of wireless network disruptions or power shutdowns of the handheld device. Especially the latter is common when a user for example decides to turn off his or her handheld device in the middle of a long running operation and to view the results at a later time.

A proposed standard which could help here is the messaging service (see [3]). The basic idea of the messaging service is to get rid of the constraining assumption of having a reliable end–to–end connection. This is achieved by introducing a store and forward mechanism on the IIOP level. *Message routers* located on the handheld device as well as the network can temporarily buffer operation invocations and their actual parameters. Once a network connection is established, the previously buffered operations are forwarded to the next message router.

Both the MinimumCORBA specification and the messaging service are still in draft mode and are subject to revisions. It is expected that they will be included into the version 3.0 of the CORBA standard. With these two additional specifications CORBA from our point of view is a viable solution as an open platform for ubiquitous computing environments.

3.4 Prototype

In early 1997 a free implementation of the CORBA standard called MICO (**M**ico **I**s **CO**rba) was started at the University of Frankfurt, Germany. Although the implementation was done for different reasons, the authors decided nevertheless to release the source code under the GNU general public license. Soon afterwards the first people interested in CORBA downloaded MICO and started using it. Bug reports as well as new contributions were made which lead to a thorough re–design of the distribution process as well as lead to a mailing list to support the increasing number of users.

Individuals and organization who were using MICO were pushing the implementation when at the middle of 1998 MICO was fully compliant to the CORBA 2.0 standard. The development still continues and the developers were able to catch up with changes made to the CORBA standard. The MICO developer mailing list counts more than 2000 subscribers world–wide who use MICO for both research projects and commercial products. In recognition of the open source movement, The Open Group branded MICO as CORBA compliant.

MICO's architecture is based on a micro kernel approach. Its ORB is designed to be small and efficient. Any extra functionality is moved outside the ORB. MICO's ORB can only transport operation invocations within one address space. Any Inter–Process Communication (IPC) support resides outside the ORB. Special purpose adapters which can register with the ORB give it its full functionality. The IIOP protocol for example is implemented as a pair of special purpose adapters.

Figure 4 depicts the architecture of MICO in a scenario, where the client runs on a PalmPilot and the server runs in the fixed network. Due to resource restrictions only client side functionality is available on the PalmPilot. This accounts for the fact that there is no Object Adapter (OA) inside the PalmPilot. The IIOP protocol is implemented through a pair of special purpose adapters which are depicted in gray in Figure 4.

An operation invocation takes the following path: originating from an application on the PalmPilot, the client invokes the operation via the Dynamic Invocation Interface (DII). The ORB, unaware of objects residing outside its own address space, forwards the operation to the special purpose IIOP client adapter. This adapter, which implements the client side of the IIOP protocol, establishes a TCP connection to its counter part: the IIOP server adapter. After the operation invocation has been received by the IIOP server, it is again send to the remote ORB via the DII. While the IIOP client is perceived as the "receiving end" (i.e. a server object), the IIOP server acts as an ordinary client object towards its ORB. As the final step, the operation is delivered to the target object by the remote ORB via the object adapter.

The prototype was developed in the spirit of the open source movement in a distributed manner, exploiting expertise of various people on the Internet. The

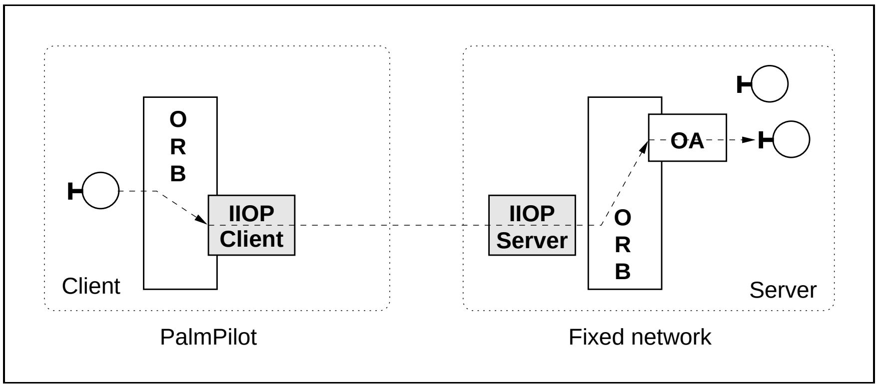

Fig. 4. Extending the CORBA platform to handheld devices.

Mico group was the first to deliver an ORB for a PDA, demonstrating the moving force behind open source projects in general. The result is an early prototype that can be downloaded from the web at `http://www.mico.org/pilot`. The prototype was implemented using the commercial development environment CodeWarrior from MetroWerks. One interesting conclusion of the project is that reducing the complexity of a C++ ORB to make it fit on an resource limited device does not necessarily mean to reduce its size. More often problems arose with the C++ compilers and reducing the complexity on a pragmatic level meant to restrict the kind of C++ language features used in the implementation.

3.5 Benchmark

A simple IDL specification was chosen for the benchmark test. The IDL specification consists of one interface with four operations. Beside an operation with no input and return parameters, there are various permutations of one argument input/output parameters. The complete IDL specification looks as follows:

```
interface Benchmark {
  void m1 ();
  void m2 (in long x);
  long m3 ();
  long m4 (in long x);
};
```

The benchmark was conducted in three different scenarios. In each scenario, the above IDL specification was implemented on the server side using the *Portable Object Adapter* (POA) and running on an Intel Pentium 266 MHz PC with Linux 2.0.35 as the operating system. The three scenarios differ in the way the client is implemented. For the first scenario, the client is colocated with the server (i.e. client and server reside in the same address space). In this case, a CORBA operation invocation decays to a normal C++ method call.

For the second scenario the client is moved to a different process, but still on the same machine. Here, IIOP is used to connect the client and the server over the loopback device. The third scenario finally moves the client to the PalmPilot. In this configuration, the client (memory footprint 48 kBytes) runs on a Palm III and is connected via a cradle to the Linux PC. The connection between the Pilot and the PC uses a 19.200 baud serial link, running PPP on top. As with the second scenario, the client and server use standard IIOP for their interactions.

	Request	Reply	Colocated	TCP	PDA
`m1`	68	12	0.0059	0.9683	180.1
`m2`	72	12	0.0059	1.0034	191.0
`m3`	68	16	0.006	0.9981	184.2
`m4`	72	16	0.006	1.0030	194.7

Table 1. Performance of operation invocation in different scenarios (average time in milli–seconds).

Note the server is exactly the same in all three scenarios. From the server's point of view it is transparent from where the client is located, under which operating system the client runs and in what language it is implemented. Table 1 gives the result of the benchmark. For each of the four operations, the size of the IIOP request and reply packets in bytes is given (one operation invocation first sends a request from the client to the server; the server then returns the results via a reply in the other direction). Since IIOP is used when client and server are not colocated, the amount of data transmitted between the client and the server is the same, independant of the location of the client.

4 Conclusion and Outlook

The claim of this paper is that ubiquitous computing requires an open system. Open systems manifest themselfs in different ways and are a necessary prerequisit to achieve a consensus in a heterogeneous environment. With respect to standardization, an open system is characterized by a transparent, technology driven adaption process where all members of the standardization body have equal rights. With respect to the software development process, we believe that open source models have benefits regarding the quality and reliability of software.

CORBA seems to be a good candidate for a middleware architecture because it is an open standard with many different implementations, both in the commercial and open source domain. The IIOP protocol separates the concerns between provider, retailer and end users, thus enabling interoperability. A telecommunication company can for example focus on integrating value added services into their networks which can be used through IIOP. Other vendors on the other hand can focus on domain specific applications which can make use of the generic services offered by an operator.

A prototype demonstrating the feasibility to have CORBA on handheld devices is available based on the open source project MICO. The prototype runs on a PalmPilot and currently relies on a fixed connection to the network due to restrictions of IIOP. It is planned to evaluate the messaging service for its suitability of supporting the offline characteristics of handheld devices.

References

1. Object Management Group (OMG), The Common Object Request Broker: Architecure and Specification, Revision 2.3, June 1999.
2. Object Management Group. Draft Specification of MinimumCORBA. ftp://ftp.-omg.org/pub/docs/orbos/98-08-04.ps, 1998.
3. Object Management Group. Draft Specification of the Messaging Service. ftp://-ftp.omg.org/pub/docs/orbos/98-05-06.ps, 1998.
4. A. Puder and M. Moscarda. Native ATM Support for CORBA Platforms. In *International Conference on ATM (ICATM'98)*, Colmar, France, 1998. Institute for Electrical and Electronics Engineers.
5. A. Puder and K. Römer. *MICO — a free CORBA compliant implementation.* dpunkt Verlag, 1998.
6. E. Raymond. The Principles of OpenSource Projects. http://www.opensource.-org/, 1998.