Towards Middleware Support for Mobile and Cellular Networks
Core Problems and Illustrated Approaches

Dem Fachbereich IV / Informatik
der Universität Trier
vorgelegte

Dissertation
zur Erlangung des akademischen Grades des
Doktors der Naturwissenschaften (Dr. rer. nat.)
eingereicht von
Dipl.-Inform. Steffen Rothkugel

Oktober 2001
Abstract

Mobile computing poses different requirements on middleware than more traditional desktop systems interconnected by fixed networks. Not only the characteristics of mobile network technologies as for example lower bandwidth and unreliability demand for customized support. Moreover, the devices employed in mobile settings usually are less powerful than their desktop counterparts. Slow processors, a fairly limited amount of memory, and smaller displays are typical properties of mobile equipment, again requiring special treatment. Furthermore, user mobility results in additional requirements on appropriate middleware support. As opposed to the quite static environments dominating the world of desktop computing, dynamic aspects gain more importance. Suitable strategies and techniques for exploring the environment e.g. in order to discover services available locally are only one example. Managing resources in a fault-tolerant manner, reducing the impact ill-behaved clients have on system stability define yet another exemplary prerequisite.

Most state of the art middleware has been designed for use in the realm of static, resource rich environments and hence is not immediately applicable in mobile settings as set forth above. The work described throughout this thesis aims at investigating the suitability of different middleware technologies with regard to application design, development, and deployment in the context of mobile networks. Mostly based upon prototypes, shortcomings of those technologies are identified and possible solutions are proposed and evaluated where appropriate.

Besides tailoring middleware to specific communication and device characteristics, the cellular structure of current mobile networks may and shall be exploited in favor of more scalable and robust systems. Hence, an additional topic considered within this thesis is to point out and investigate suitable approaches permitting to benefit from such cellular infrastructures. In particular, a system architecture for the development of applications in the context of mobile networks will be proposed. An evaluation of this architecture employing mobile agents as flexible, network-side representatives for mobile terminals is performed, again based upon a prototype application.

In summary, this thesis aims at providing several complementary approaches regarding middleware support tailored for mobile, cellular networks, a field considered to be of rising importance in a world where mobile communication and particularly data services emerge rapidly, augmenting the globally interconnecting, wired Internet.
## Contents

1 Introduction .............................................................................1  
   Structure of the Thesis .........................................................2  

2 Applications in Mobile, Cellular Networks .................5  
   Technological Background  
   2.1 GSM .............................................................................6  
       History .................................................................6  
       GSM System Architecture ........................................7  
       Mobile Stations ......................................................10  
       Handover ............................................................10  
       Security ...............................................................12  
       Mobile Services .....................................................14  
   2.2 UMTS .................................................................18  
       Extended General UMTS System Architecture ...........19  
   2.3 Mobile Equipment ....................................................20  
       Smartphones ..........................................................20  
       Wireless Operating Systems ......................................21  
       Smartphone Market Overview ...................................24  
   Middleware for Mobile, Cellular Networks  
   2.4 Challenges and General Approaches .......................25  
       Wireless Networking ..............................................25  
       User Mobility .......................................................28  
       End Device Restrictions .........................................29  
   2.5 Core Investigations .....................................................30
3 Middleware Support based on Web Standards ...... 33

State of the Art
3.1 Wireless Application Protocol ...................... 33
   WAP Architecture .................................. 34
   WAP Push Framework .............................. 37

Investigations
3.2 A Mobile Group Schedule based on WAP 1.2 ...... 39
3.3 Evaluation and Criticism of WAP ..................... 41
3.4 Summary and Conclusions .......................... 48

4 Middleware Support based on CORBA ............... 51

State of the Art
4.1 Message-Oriented Middleware ...................... 51
4.2 Distributed Object Middleware .................... 52
   Java RMI .......................................... 53
   COM+ ............................................. 53
   CORBA .......................................... 54
4.3 Introduction to CORBA ............................. 54
   Object Management Architecture .................. 55
   MinimumCORBA ................................. 56
4.4 Wireless CORBA .................................. 57
   Architectural Framework .......................... 58

The Application Layer
4.5 General Considerations ............................ 59
   CORBA Servants and Persistence .................. 60
   Two-Layer Architecture ........................... 61
   Contradictory Standards .......................... 62
4.6 Callback Framework ................................ 63
   CORBA Event Service .............................. 63
   Conceivable Optimizations ....................... 65
4.7 Deployment Issues ................................ 66
   Installation Options ............................... 66
   Code Size ........................................ 67
   Portability ....................................... 68

The Protocol Layer
4.8 Examining CORBA Communication Protocols ....... 69
   Common Data Representation ...................... 69
   GIOP Message Formats ............................ 71
   GIOP Transport Assumptions ...................... 72
   IIOP ............................................. 72
   ESIOPs ........................................... 73
## Contents

ORB Interoperability Architecture .................................. 74  
Communication Failures .............................................. 74  
Endpoint Identification ............................................... 75  
Discussion .............................................................. 76  

4.9 Evaluating WAP as Basis for an ESIOP ......................... 78  
Requirements .......................................................... 78  
Wireless Datagram Protocol ......................................... 79  
Wireless Transaction Protocol ..................................... 79  
Wireless Session Protocol .......................................... 82  

4.10 WAPIOP—A WAP-based Inter-ORB Protocol ................... 84  
Basic Considerations .................................................. 84  
Implementation Strategies ......................................... 86  
Wireless Data Representation ..................................... 87  
WSP/CORBA ............................................................. 93  

4.11 Evaluation .......................................................... 99  
WDR versus CDR ....................................................... 99  
Comparative Example ................................................ 103  

4.12 Summary and Conclusions ........................................ 106  

5 Middleware Support in Spontaneous Networks ................. 109  

State of the Art  

5.1 Introduction to Jini ................................................. 110  
  Services .............................................................. 110  
  Lookup Service ..................................................... 110  
  Discovery ............................................................ 111  
  Leases ................................................................. 111  
  Jini Surrogate Architecture .................................... 112  

Investigations  

5.2 Discovery and Lookup for Mobile Devices ..................... 113  
  Discovery Protocol .................................................. 113  
  Notion of Locality .................................................. 113  
  Service Inquiry ...................................................... 114  
  Notion of Locality Revisited ................................... 115  
  Lookup Service Federations ..................................... 115  
  Automated Service Announcement ............................... 116  

5.3 Integrating Jini and CORBA ...................................... 117  
  Experiences .......................................................... 118  

5.4 Scalability in Large Jini Communities ......................... 119  
  Wireless Smart Warehousing .................................... 120  
  Smart Labels as Jini Services ................................... 121  
  Agglomerations ...................................................... 121  
  Distributed Self-Organization .................................. 122  
  Self-Optimization .................................................. 123
Contents

Simulation Results .................................................. 124
5.5 Summary and Conclusions ................................. 125

6 Middleware Support based on Mobile Agents .... 127

State of the Art
6.1 Fixed Network Proxies ............................ 128
6.2 Mobile Agents ........................................ 129

Investigations
6.3 Mobile Agent-Based System Architecture ........... 130
   Exploiting the Cell-Based Network Structure ........ 130
   The Big Picture ........................................ 131
6.4 Prototype Implementation ............................. 132
   CarPAcities .......................................... 132
   Agent-Based Proxies .................................. 133
   User Tracking ......................................... 134
   Agent Ensembles ..................................... 135
   Agent Propagation .................................... 136
   Jini Services in CarPAcities .......................... 138
   Lease-Based Resource Management .................. 139
   Failure Resilience .................................... 139
6.5 Summary and Conclusions ............................. 142

7 Conclusions ...................................................... 145

Recapitulation

Emerging Standards
7.1 Global Multimedia Mobility ....................... 150
   Open Service Architecture .......................... 150
   Virtual Home Environment .......................... 152
7.2 MExE ..................................................... 152

Future Directions

8 Literature ......................................................... 159
Introduction

The popularity of the Internet is without doubt still growing. While only a few years ago access to the network of networks has been rather limited, it now has become almost natural even for private individuals to be connected. Though this revolution is still going on, another trend currently dominates the world of connectivity. People not only want to be able to access information from their homes, they additionally strive for ubiquity. Information anywhere at any time is the slogan, or put in other words, being always on is the goal.

Rapidly emerging technologies in the field of mobile communications shall put this dream into reality. I-mode and WAP—both technologies permit restricted access to the Internet—are two popular examples clearly documenting this trend, albeit with different background and diverse success. Over 15 million subscribers use i-mode on a daily basis. They enjoy sending and receiving e-mail, exchanging photographs, shopping and banking, downloading personalized melodies for their phones, and navigating thousands of specially formatted Web sites [Bou00].

Obviously, the services offered so far to mobile users highly parallel those they are accustomed with from their homes, albeit mostly in a stripped-down version. More sophisticated, location-dependent services are emerging, but still are rather infrequent today. Partly, this is due to the limitations imposed by the equipment on the one hand and—to a higher degree—by an insufficient application and network infrastructure on the other hand. This thesis is focused on discussing and solving some of the essential problems inherent to middleware support for applications in the context of mobile, cellular networks. For applications of this type, the industry coined the term M-Solutions, with the initial letter explicitly emphasizing their focus on mobility.
Current mobile devices, in particular cellular phones, are rather restricted in terms of processing power and display capabilities. However, there is a visible trend towards more sophisticated appliances. Referred to as smartphones, they are a kind of symbiosis between cellular phones and personal digital assistants. On top of such devices, it will be possible to create elaborate applications, probably to be downloadable dynamically.

Several current limitations stem from the network technology and an inadequate support for sophisticated applications in the context of mobile networks. The intrinsic characteristics of such wireless networks are foremost low bandwidth, high latency, and unreliability. Besides those communication related attributes, different requirements on the underlying application infrastructure arise due to the inherent mobility.

Location-dependency, coming in different flavors, is a distinct feature of such mobile applications, having to drive the design of a suitable infrastructure. For example, services available locally in the current surroundings have to be made aware to the mobile user. Proper techniques for service discovery, complemented by automated service announcement, have to be provided.

Moreover, services for mobile users can no longer be deemed to be stationary in nature. Rather, a single service may be multifaceted, again depending on local factors. Together with the obvious need for scalability—an important factor in the realm of popular mobile networks—centralized services are not a well-suited approach. Instead, the cell-based structure of current mobile networks shall be reflected in the service realization, fostering scalability in a natural way.

**Structure of the Thesis**

The following chapter introduces current wireless network technologies and mobile equipment, primarily discussing their special properties and characteristics contrasting them from traditional settings employing desktop computers connected via fixed networks. The requirements on sensible middleware support in this context will be discussed subsequently, identifying several challenges together with well-known general approaches and their limitations. The chapter concludes with a presentation of particular distinctive problems that will be faced within the remaining chapters of this thesis.

An examination of the Wireless Application Protocol as the mobile counterpart of the World Wide Web will be performed in the chapter that follows. Based upon a sample application realizing a group schedule, the advantages and shortcomings of WAP and its applicability as middleware platform for the application development in the context of mobile, cellular networks will be discussed.
In the subsequent chapter, a different programming paradigm employing distributed object middleware will be discussed. After introducing the CORBA object bus and current standardization efforts in terms of wireless access and terminal mobility, several remaining open questions regarding middleware support in the setting under scrutiny will be addressed. The investigations will be subdivided in application- and protocol-layer specific sections.

The following chapter will focus on spontaneous networks, foremost examining aspects regarding service discovery and location-dependency. In line with the preceding chapter, approaches for the integration of the proposed solutions with CORBA are investigated. Finally, scalability-related issues will be elaborated on.

Subsequently, the experiences gained from the work described in the preceding chapters will be assimilated, resulting in the presentation of a mobile agent-based system architecture explicitly tailored for cell-based networks. In the context of a specific prototype application scenario—particularly a distributed car pool agency—this system architecture will be evaluated further.

The final chapter will present several emerging technologies in the field of application development in the context of mobile, cellular networks. The results of the work will be summarized with a special focus on embracing those upcoming standards. Pinpointing future trends and possible extensions will conclude this thesis.
Distributed system technologies have become an integral part of many modern applications. A set of processes hosted on different, often heterogeneous machines, collaborating on a common task is a typical system architecture nowadays. Network technologies play a central role in this regard. Information may flow—in a controlled manner—between computers located all around the world over high-speed corporate Intranets mutually interconnected by the Internet backbone.

Besides the Internet, a second (nearly) world-wide network exists even longer—the telephone system. Designed initially for voice transmission, the importance of data communications increases. Most private individuals for instance use their telephone lines to connect to the Internet.

In recent years, a third type of network joined the picture—mobile, cellular phone networks. The same story that happened with respect to the traditional telephone system has been repeated. Used primarily for voice calls in its early days, data connections over-the-air gained more momentum. Initially, cellular phones have served as a kind of mobile connector to the Internet for portable computers only. The most current trend—enabled by comparatively sophisticated handsets—is to use the mobile device itself as a small, portable computer.

Distributed system technologies, albeit mature and elaborate with respect to the former fixed networks are not well-suited for the latter, wireless communication systems, particularly when employing resource constrained end-devices. The characteristics of both network types together with the technical environment are considerably diverse, resulting in different requirements on sensible middleware support.
This chapter will first provide an in-depth introduction to the technological background of mobile networks. Besides explaining the infrastructure of such wireless networks, special focus is given to characteristics relevant for the investigations, namely those of data communications. Subsequently, mobile equipment will be introduced, discussing its properties as well as hardware and software capabilities. Thereafter, the challenges posed on suitable middleware support for the application development in the context this particular environment will be set forth. Finally, a road map of the investigations performed throughout this thesis will be presented.

## Technological Background

### 2.1 GSM

Digital cellular networks are the most prominent counterpart of traditional public switched telephone networks (PSTN) in the wireless world. The Global System for Mobile Communications (GSM) is today's most popular and successful digital mobile communications system. Statistics dated from end 2000 published by EMC [EMC] as shown on the Web site of the GSM Association [GSM] report approximately 400 million subscribers located in over 160 countries.

**History**

In 1982, the development of a standard for a digital cellular mobile network was initiated by the Groupe Spécial Mobile, originally coining the acronym GSM. After the European Telecommunication Standards Institute (ETSI) has been founded in 1989, the GSM group became a technical committee within the ETSI. The broad acceptance of GSM networks lead to the renaming into the Global System for Mobile Communications.

The proposed standard had to meet certain requirements [Sco97]:

- good subjective speech quality
- low terminal and service cost
- support for international roaming
- support for new services and facilities
- spectral efficiency
- compatibility with ISDN

In 1990, the so-called phase 1 GSM specifications were published. They were composed of 130 documents with over 5000 pages. Commercial service started in mid-1991. By 1993, there have already been 36 GSM net-
works established in 22 countries. At end of the year 2000, there have been 376 operational networks in over 160 countries. The tremendous growth of the number of subscribers is shown in figure 1.

GSM currently accounts for 68.5% of the World’s digital market. The market penetration of GSM in European and Asian countries is far bigger than in other parts of the World (cf. figure 2).

Being a descendant of traditional telephone architectures, digital cellular networks are primarily designed for the transfer of voice. Consequently, the basic versions typically implement circuit switched services. Although even today the main traffic is created by voice calls, the usage of data services is growing continuously. For instance, the number of so-called short messages (SMS) sent grew from 4 billions in January 2000 to 12 billions in October 2000.

Being prepared to handle a vast amount of customers in a reliable way, and to incorporate scalability as an integral factor, GSM has a hierarchical design with many entities communicating over well-defined interfaces. A simplified overview of the GSM system architecture as specified in [ETSI91a] is shown in figure 3.
The GSM architecture is comprised of the following three subsystems:

- the radio subsystem (RSS),
- the network and switching subsystem (NSS), and
- the operation subsystem (OSS).

### Radio Subsystem

As the name implies, the radio subsystem is responsible for the management of over-the-air communications. The infrastructure is made up of radio cells, covering an area between approximately several 10 meters and a few 10 kilometers in diameter each [Schi00]. The cell size depends on geographical data, because of the physical properties of radio waves, as well as on the expected number of customers to be served simultaneously.

Each cell is managed by a base transceiver station (BTS). The BTS comprises all radio equipment like antennas and amplifiers that are needed for radio transmission. Base transceiver stations communicate with mobile stations (MS) via the so-called Um interface, covering all mechanisms needed for wireless transmission.

Multiple BTSs are managed by a base station controller (BSC). The communication utilizes the A_bis interface, consisting of 16 or 64 kbps connections. Among the responsibilities of the BSC are the reservation of radio frequencies and paging of the MS. The organizational unit which is comprised of a single BSC and all the BTSs it manages is called a base station subsystem (BSS).
One level above the base station subsystems are mobile switching centers (MSC), each serving multiple BSSs, communicating over the A interface. MSCs are high-performance digital ISDN switches, forming the backbone of the fixed network of a GSM system.

The network and switching subsystem essentially connects the wireless network with the traditional, wired telephone systems. For this purpose, gateway MSCs (GMSC) are employed. Furthermore, so-called interworking functions (IWF) realize connectivity to public data networks like X.25.

Another responsibility of the NSS is the localization of users. For this purpose, MSCs rely on two different kinds of database services, called home location register (HLR) and visitor location register (VLR). Although HLR and VLR are explicitly designed as standalone services, for reasons of resource savings in terms of hardware and signalling bandwidth, they typically are implemented as part of MSCs.

The Home Location Register stores all relevant information about each subscriber. For example, the HLR contains the customer’s mobile subscriber ISDN number (MSISDN) as well as other administrative information like services subscribed to. An important entry in the HLR is the current location area (LA) of a user’s MS, enabling its localization. The information about the LA is updated as the user moves between different MSCs and their associated location area.

In contrast to the mostly static HLR, the VLR stores highly dynamic data. The visitor location register contains copies of selected administrative information about subscribers currently residing in the geographical area managed by the VLR. This kind of caching mechanism avoids frequent communication with the HLR. Furthermore, the NSS is responsible for charging and accounting purposes, as well as to allow users to roam between several network providers, possibly located in different countries.

The third subsystem, the operation subsystem, realizes all functions necessary for network operation and maintenance. The operations and maintenance center (OMC) monitors and controls selected network entities like MSCs, BSCs and GMSCs via the O interface, collecting for instance traffic information for status reports. An important responsibility of the OMC is security management, handled in conjunction with the Authentication Center (AuC). The AuC helps to secure data transmissions through authentication and encryption algorithms. Information about the subscribers and their mobile station is held in the equipment identity register (EIR). For example, the EIR stores information about stolen mobile stations in form of their unique international mobile equipment identity (IMEI).
The user accesses a GSM network using a mobile phone in conjunction with a subscriber identity module (SIM) [ETSI91c], a chipcard holding subscriber related information. This information is comprised of e.g. the user’s unique international mobile subscribe identity (IMSI), a PIN needed to activate the SIM, and cipher keys used for secure data transmission. Furthermore, when the user is logged into a GSM network, the SIM stores dynamic information like the identification of the current location area and a temporary mobile subscribe identity (TMSI), as will be explained further in Subscriber Identity Confidentiality on page 13. Employing transient TMSIs instead of the static, well-known IMSI is favorable in terms of anonymity.

The SIM can be used to personalize a mobile station, essentially separating terminal from user mobility. Users are not forced to use a single phone only, while conversely one terminal can be used by different people.

The cellular infrastructure of mobile networks like GSM demands for a mechanism called handover. As users move, they traverse different cells, each covering a limited area only. In order to avoid disruption of service, also known as call drops, connections must transparently be handed over from one cell to another. The GSM standard identifies about forty possible reasons for handovers to take place. They can be roughly classified as follows [Schi00]:

- **Handover due to decreasing signal grade:**
  The quality of the radio link is influenced by a lot of factors, like the distance from the transmitter and interference with neighboring cells. Environmental factors, e.g. buildings and mountains, cause blocking, shadowing and reflection of radio waves. This eventually leads to multi-path propagation, where the signals emitted by the sender reach the receiver along multiple paths with different lengths and at various times. All those effects essentially may cause the error rate to exceed an upper limit.

- **Traffic density initiated handover:**
  When the number of subscribers served by a single cell gets too high, the network infrastructure—BSCs or MSCs—may decide to shift some connections to neighboring cells. Thus, handover may be a result of load balancing.

Handovers are only propagated as high in the network infrastructure as needed. There are four possible scenarios [Schi00]:

- **Intra-cell handover**
  Due to interference, a BSC may decide by itself to change the carrier frequency.

---

**MOBILE STATIONS**

**HANDOVER**
Inter-cell, intra-BSC handover

When the mobile station moves between cells managed by a single BSC, this BSC performs the handover on its own, assigning a new channel in the new cell and releasing the old one.

Inter-BSC, intra-MSC handover

Because BSCs cover a limited area only, a mobile station may cross the bounds of two neighboring BSCs. In this case, the corresponding MSC performs the handover.

Inter-MSC handover

Finally, the mobile station may move between cells belonging to different MSCs. Here, both MSCs are involved in handover processing.

In any case, after completion of a handover process, the MSC is informed, allowing to update the location information stored in the HLR.

Both the mobile station and the BTS periodically measure the quality of the downlink and the uplink, respectively. The link quality depends on the signal level and the bit error rate. The MS propagates information about the current link as well as selected channels of neighboring cells every half a second to its BTS.

An example of a handover decision is shown in figure 4 [Schi00][Wal00]. The user moves from one base transceiver station (BTS_{old}) to a different one (BTS_{new}). The corresponding BSC collects averages of the measured signal levels. A handover is theoretically possible only between locations a and c. However, the suggested time to perform the handover is at location b, when the difference between the two average signal levels exceeds the hysteresis value HO_MARGIN.

![Fig. 4. Handover Decision](image-url)
The reason behind including the threshold \texttt{HO\_MARGIN} is to avoid rapid successive handovers— the so-called ping-pong handover effect—due to temporary signal degradation [WL97]. However, this technique only alleviates the problem, but cannot avoid it entirely. For example, in structurally dense regions like downtown areas where cell sizes are small, the aforementioned shadowing, blocking and reflection effects result in a high variation of signal levels. It is not uncommon that a handover from one cell to another and back again occurs within less than 10 seconds [Wal00].

Mobile, wireless networks are doomed to multiple kinds of security threats. While mobile equipment is vulnerable to theft, wireless communication in particular is susceptible to eavesdropping. The GSM specifications comprise several security related techniques, classified as follows:

- subscriber identity confidentiality
- subscriber identity authentication
- confidentiality of signalling information and user data.

The GSM specifications define several algorithms and protocols to be used for authentication and encryption purposes. However, the documents give only some advice on concrete implementations, leaving room for variability. An architecture providing a high degree of security is shown in figure 5. In this realization, the subscriber’s individual key $K_i$ is only known by the authentication center (AuC) and the user’s SIM. The key is never exposed to other entities in the network.

For authentication and encryption purposes, the AuC generates and stores a set of tuples ($\texttt{RAND}$, $\texttt{SRES}$, $K_c$). $\texttt{RAND}$ is a 128-bit random value, used together with the secret key $K_i$ as input of two algorithms, A3 and A8. By means of A3, the expected answer of a challenge-response protocol, $\texttt{SRES}$, is calculated. Using A8, a stream cipher key $K_c$ is produced. A couple of those tuples is created in advance and forwarded to the HLR. This way, requests by the VLR at the time of connection establishment can be satisfied immediately, drastically reducing signalling overhead in the critical path.

Before users are given access to a GSM network, they have to be authenticated. The authentication process performed when logging into the network is shown in the upper right part of figure 5. First of all, the user must unlock his mobile phone, or essentially his SIM, by entering a secure PIN. Thereafter, a challenge-response protocol is applied, based on one ($\texttt{RAND}$, $\texttt{SRES}$, $K_c$)-tuple as requested from the HLR. The network sends the 128-bit random value $\texttt{RAND}$ to the MS. Analogous to the AuC, the mobile station calculates $\texttt{SRES}'$ and $K_c'$, using the individual key $K_i$ retrieved from the user’s SIM. This key shall be the same as the one stored in the AuC. The
calculated SRES’ is sent back to the network, which now checks on equality, finally authenticating the user.

After the session is authenticated, the communication between the mobile station and the network can be encrypted. An algorithm called A5 is used for this purpose. A5 is a symmetric stream cipher, applied in the mobile station and in the base transceiver station, respectively. Therefore, only the over-the-air communication channel between MS and BTS is encrypted. Ultimately, there is no stringent encryption within the fixed part of the network, and consequently no end-to-end encryption.

In general, it should be noted that encryption is not very strong. Initially, only A5 was publicly available, whereas A3 and A8 have been secret with public interfaces only. However, in 1998 both A3 and A8 were published on the Internet. It can be shown that in certain implementations, 10 of the 64 bits of the cipher key $K_c$ are always zero, essentially leaving a key with only 54 bits real length [Schi00].

The primary idea in order to retain the anonymity of users is to prevent well-known information identifying subscribers to be submitted over-the-air as far as possible. This is achieved mainly by employing a temporary mobile subscriber identity (TMSI) instead of using the unique IMSI directly. At latest on each change of the subscriber’s location area, the VLR assigns a new TMSI, which is transferred over-the-air in encrypted form. Therefore, only the TMSI in conjunction with the current location area identifier (LAI) uniquely identifies a single user.
However, in some cases, the IMSI must be sent in plain text over-the-air. For example, if VLR data is lost, corrupted, or incomplete, the GSM standard requires the re-authentication of the subscriber. The base station requests the IMSI from the mobile station, starting a new authentication procedure and generating a new TMSI.

**MOBILE SERVICES**

GSM offers a set of different voice and data services, comparable to the Integrated Services Digital Network (ISDN) [Boc97]. Those services are divided in three categories: bearer, tele, and supplementary services.

**Bearer Services**

Bearer services are responsible for the data transmission. The bandwidth provided varies between 300 bps and 9.6 kbps [ETSI91b]. GSM distinguishes between transparent and non-transparent modes.

Transparent bearer services are implemented at layer 1 of the ISO/OSI reference model [ISO-7498], using a circuit switched connection between the mobile station and the Interworking Function (IWF) entity (cf. figure 3). Such connections apply forward error correction to alleviate bit errors. However, no recovery of lost data is performed, e.g. in case of shadowing or interruption due to handovers. Transparent transmissions are characterized by a constant bit rate, constant delay, and variable bit error rate.

In non-transparent mode, the radio link protocol (RLP), an ISO/OSI layer 2 protocol, is applied. RLP triggers the retransmission of corrupt data. Ultimately, the achieved bit error rate is less than 10⁻⁷, but throughput and delay vary depending on transmission quality.

**Tele Services**

Based on the bearer services, GSM defines a couple of tele services, e.g. voice telephony, undoubtedly currently the most important tele service, group 3 facsimile, and the short message service (SMS).

**SMS.** The short message service allows the transfer of textual messages up to 160 characters in length. The transmission is based on a store-and-forward principle. Service centers in the network provider’s infrastructure are responsible for the transfer of the messages, using acknowledgements and retransmissions for reliable delivery. However, there is no feedback regarding if and when a message has been read by the receiver. One interesting feature of the SMS service is that it exploits unused capacity in the signalling channels. Therefore, sending and receiving short messages is possible even during an active call.

**Data Services**

Besides making voice calls, there is an increasing demand for the transmission of data between the mobile station and the wired network. In current phase 2 compliant GSM networks, communication with packet switched data networks is already possible. However, there are a couple of shortcomings. First of all, the over-the-air interface is still connection-oriented, occu-
pying a full traffic channel. Due to the nature of data communications, which is typically bursty, a lot of resources of the air link are wasted. Furthermore, the available bandwidth of 9.6 kbps (or up to 14.4 kbps with some providers) is rather low when compared to the requirements of increasingly popular applications for the mobile world like Web browsing and E-mail.

**HSCSD.** A straightforward approach at gaining higher throughput is to use multiple traffic channels in parallel. With *High Speed Circuit Switched Data* (HSCSD), up to four time slots of a single transmission frame can be occupied concurrently [ETSI97]. The resulting bandwidth is up to 38.4 kbps with 9.6 kbps per slot, or 57.6 kbps with 14.4 kbps per slot. In theory, when occupying all eight time slots, up to 115.2 kbps can be achieved [ETSI98a]. Slot allocation has to be negotiated in advance and can be asymmetrical, allowing more bandwidth available on the downlink (*to* the mobile station) than in the uplink (*from* the mobile station).

HSCSD can be implemented with relative low effort in the fixed network infrastructure. Channel bundling can be realized with software updates only. However, the requirements on mobile stations are higher, because data now has to be transmitted in several time slots using different frequencies concurrently.

Although data rates are significantly higher, HSCSD has major drawbacks. First of all, transmission remains circuit switched, now occupying even multiple traffic channels. Moreover, handover is significantly complicated, because each channel is handled separately, requiring allocation and deallocation of multiple channels simultaneously.

**GPRS.** The *General Packet Radio Service* (GPRS) is a true packet switched service, providing packet-oriented transmission even over the wireless link [ETSI98b]. This not only allows for an accounting model based on the real amount of data transferred instead as on pure connection time. Beyond, it enables the mobile user to be always-on by retaining open connections to service providers without inducing costs.

The primary design principle is to provide capacity on demand. Unlike all other techniques described so far, time slots are not allocated in advance and in a fixed way, but solely on demand. Depending on the underlying coding scheme, bandwidth of GPRS can be up to 21.4 kbps per slot, resulting in an overall theoretical maximum of 171.2 kbps. In this respect, GPRS is an evolutionary step, aimed at exploiting resources in a fairly efficient way.
In phase 1, GPRS offers point-to-point (PTP) transmissions [ETSI98c] only, while point-to-multipoint (PTM) packet transfer is left for phase 2 [ETSI98d]. PTP packet transfer can be connection-oriented, maintaining a virtual circuit regardless of cell changes. This service is well-suited for circuit switched protocols like X.25. A different option for PTP communication is the connectionless service, best suited to support corresponding protocols like IP. In PTM, packets are either delivered to all subscribers staying in a particular geographical area, or to all members of a specific PTM group.

However, GPRS support requires major changes in the network infrastructure. A new class of nodes supplements the GSM system architecture as shown in figure 6. The gateway GPRS support nodes (GGSN) are the connection points to external networks like the Internet or X.25. The GGSN bridges between internal address and packet formats and the externally used ones. The serving GPRS support node (SGSN) is responsible for the logical link management, requesting user addresses from the GPRS register (GR), as well as for the mobility management, keeping track of the mobile station’s location and state.

A mobile station can be in one of three states: idle, ready, and standby [ETSI98d]. In the idle state, the MS is disconnected and unreachable. Before any data can be transmitted, the mobile station first must attach to the GPRS network, changing its state to ready. Detaching takes the MS back to the idle state.
While in the ready state, the SGSN is notified of each cell change, allowing data to be routed to the mobile station without any further lookup overhead. However, when the mobile doesn’t transmit any data packets for a pre-defined time, the GPRS state changes to standby. In this state, the SGSN will only be notified whenever the MS changes its routing area (RA), consisting of a couple of neighboring cells. This way, signalling effort is reduced at the expense of a higher setup overhead at the time of the next packet transmission. If after another period of time no data has been transferred, the state will change back to idle automatically.

After a successful GPRS attach, networking sessions can be initiated by the mobile station. For this purpose, a packet data protocol (PDP) context has to be created. The PDP context contains an address from the address space of the destination network, e.g. IPv4. This address can be a fixed address selected by the MS or it can be left empty to request dynamic allocation. Additionally to the address, the mobile station specifies a desired quality-of-service profile (see below) and a GGSN serving as access point to the external public data network (PDN), e.g. the Internet. The PDP context is stored in the SGSN and the GGSN as well as in the mobile station itself.

After their creation, PDP contexts can be activated, at first leading to the authentication of the subscriber as described above. In case of the address contained in the PDP context being empty, the GGSN is responsible for dynamic address allocation. Through activation of the PDP context, the MS becomes a visible entity for the respective PDN. The GGSN now can route packets between the PDN and the MS, using the mapping of the mobile station’s IMSI onto the address contained in the PDP context.

GPRS defines four parameters for the quality-of-service specification: service precedence (or priority), reliability, delay, and throughput [ETSI98c]. One quadruple of concrete QoS parameter values is associated with every PDP context. Example reliability and delay classes are shown in table 1 and table 2. Obviously, both reliability and in particular delay values are orders of magnitude worse than those of fixed network technologies.

<table>
<thead>
<tr>
<th>Reliability class</th>
<th>Probability of lost packet</th>
<th>Probability of duplicate packet</th>
<th>Probability of out of sequence packet</th>
<th>Probability of corrupted packet</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$10^{-9}$</td>
<td>$10^{-9}$</td>
<td>$10^{-9}$</td>
<td>$10^{-9}$</td>
</tr>
<tr>
<td>2</td>
<td>$10^{-4}$</td>
<td>$10^{-5}$</td>
<td>$10^{-5}$</td>
<td>$10^{-6}$</td>
</tr>
<tr>
<td>3</td>
<td>$10^{-2}$</td>
<td>$10^{-5}$</td>
<td>$10^{-5}$</td>
<td>$10^{-2}$</td>
</tr>
</tbody>
</table>

TABLE 1. GPRS Reliability Classes
An interesting property of GPRS is its capability of supporting IP-based communication. Mobile units will be assigned their own IP address. This way, they become seamlessly integrated with the world of the Internet, without any bridging overhead needed. Especially applications in the fixed network—either legacy or newly developed—can benefit from a homogeneous view on fixed as well as mobile communication partners.

**EDGE.** In the context of the standardization of **enhanced data rates for GSM evolution** (EDGE) [ETSI98e][ETSI98f], **enhanced GPRS** (EGPRS) emerges. Employing modified modulation techniques, gross bit rates of up to 69.2 kbps are possible per time slot. Using all eight time slots theoretically results in a net bit rate of about 384 kbps. For the implementation of EDGE, only the transceiver units of the mobile station and the base stations have to be replaced. Except for software updates, the network infrastructure can remain unchanged. Eventually, EDGE is an intermediate step from second generation mobile telecommunications systems like GSM to third-generation systems like UMTS.

### 2.2 UMTS

In 1992, the **International Telecommunication Union** (ITU) began its work on IMT-2000, a framework for International Mobile Telecommunications. The basic idea was to establish a set of interoperable telecommunications systems with worldwide roaming capabilities. Through IMT-2000, diverse wireless networks, both terrestrial and satellite based, are linked together into a single, cohesive, global network infrastructure [Dor01].

In order to streamline the standardization efforts, several organizations in the global telecommunications market have founded the **Third Generation Partnership Project** [3GPP]. Its members are the **European Telecommunication Standards Institute** [ETSI], the **Committee T1** for the United States of America [T1], the **Telecommunication Technology Committee** [TTC] and the **Association of Radio Industries and Businesses** [ARIB] from Japan, the **Korean Telecommunications Technology Association** [TTA], and the **China Wireless Telecommunication Standard Group** [CWTS]. The proposal from

<table>
<thead>
<tr>
<th>Delay class</th>
<th>128 byte packet</th>
<th>1024 byte packet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean</td>
<td>95% percentile</td>
</tr>
<tr>
<td>1</td>
<td>&lt; 0.5 s</td>
<td>&lt; 1.5 s</td>
</tr>
<tr>
<td>2</td>
<td>&lt; 5 s</td>
<td>&lt; 25 s</td>
</tr>
<tr>
<td>3</td>
<td>&lt; 50 s</td>
<td>&lt; 250 s</td>
</tr>
<tr>
<td>4</td>
<td>unspecified (best effort)</td>
<td>unspecified (best effort)</td>
</tr>
</tbody>
</table>

**TABLE 2. GPRS Delay Classes**
the Third Generation Partnership Project for IMT-2000 is called *Universal Mobile Telecommunications System* (UMTS). UMTS is a Third Generation mobile system, as opposed to the Second Generation GSM. However, UMTS inherits much of GSM, thereby providing a smooth evolutionary path from GSM to UMTS.

One of the most interesting aspects of UMTS is its explicit support for multiple diverse wireless networking technologies, together with the capability to roam transparently between different networks with minimal break in communication [UF1998]. People are supposed to move between indoor, outdoor congested (urban) and outdoor rural environments. This is reflected in the system architecture by a hierarchical cell structure, as depicted in figure 7. The smaller the cell size, the higher the potential user density and the available bandwidth will be. While at least 144 kbps shall be made available in macro cells for people moving with speeds up to 500 km/h, pico cells are supposed to offer data rates of up to 2 Mbps at a maximum speed of 10 km/h.

An important perspective taken by ETSI’s UMTS Task Force is to include fixed and wireless access to public as well as private networks in the extended general UMTS architecture [ITU-1701][ETSI99c], as shown in figure 8.

The *infrastructure domain*, which is the property of a network operator, is further subdivided in the *core network* and the *access network*.
domains [ETSI99d]. The core network encompasses all entities of the (fixed) backbone, comparable to the network and switching subsystem of a GSM network. The separate access network domain effectively decouples access related functionality from non-access related functionality. One example of a concrete access network technology, not shown in the figure, is the UMTS terrestrial radio access network (UTRAN) [ETSI99e], closely resembling the GSM radio subsystem.

In contrast to GSM, the customer equipment domain now explicitly includes a private network domain and a cordless base station domain, thus allowing to embed private networks in a global network infrastructure. Cordless base stations may use frequencies in the spectrum identified for non-public domestic and corporate use [ITU-1036]. Private networks may employ both fixed and wireless technologies like TCP/IP and Bluetooth or DECT, respectively.

It is assumed that in the near future, it will be common for devices to include adapters for multiple networking technologies, wired as well as wireless. Accordingly, third generation mobile systems will enable users not only to roam among countries which currently use different technologies but will also allow them to move seamlessly between multiple networks—fixed and mobile [ETSI99b].

Through the inclusion of private networks in the general UMTS infrastructure, mobile equipment as well as fixed terminals are in principle enabled to access both public and private services via a uniform interface. Like in GSM networks, those user equipment is personalized by a user services identity module (USIM) [ETSI99f]. The USIM authenticates a single user, thus allowing authorization control, especially with respect to private networks and services.

### 2.3 Mobile Equipment

Users access mobile telephone networks through more or less sophisticated mobile terminals. Simple mobile phones are rather low-cost devices, often delivered to subscribers for free. They mostly provide only voice call related functionality and the ability to send and receive SMS. Current standard mobile phones are comparatively limited in terms of processing power and display capabilities. However, the trend goes towards smartphones, a symbiosis between traditional mobile phone and personal digital assistant (PDA). Those appliances often incorporate personal information management (PIM) software like calendars, schedules, and contact
Applications in Mobile, Cellular Networks

databases. Moreover, E-mail readers and Web browsers are typical applications included in such devices.

Mobile terminals consist of the hardware, an operating system, and applications. Traditionally, manufacturers developed the whole product in-house. However, the development effort steadily increases, along with the complexity of all the single pieces. This is one of the reasons why especially the software parts often become outsourced. Another rationale behind this is to allow third-party development of applications and more interoperability due to open standards.

Symbian [Sym] is an evident example, founded by Nokia, Ericsson and Motorola together with Psion Software. Those corporations teamed up to build a software platform intended to become a standard for the next generation wireless smartphones [Buc00b].

The Symbian Platform 6.0 [Sym00a], as shown in figure 9, basically is a next evolutionary step, built on top of the EPOC operating system release 5 [Sym00b]. The EPOC multi-tasking kernel together with some generic components serve as a common basis. Above, there is a separate GUI layer, adapting to the capabilities of the end device used.

The aforementioned generic components encompass data management, communications, security, basic graphics functionality, a messaging engine, browser engines for WAP and HTML, and support for data synchronization with a desktop PC. Finally, the Symbian Platform contains a PersonalJava 1.1.1 [Sun99a] compliant runtime environment, supplemented by the JavaPhone 1.0 API [Sun00a]. JavaPhone offers convenient access to key wireless facilities like telephony, power monitoring, contacts, schedule, and more. Application development is possible either in Java or C++.

Symbian Platform 6.0 contains two GUI layer reference implementations, called Quartz and Crystal, respectively. Quartz is targeted at palmtop-like devices with pen-based input, with built-in handwriting recognition. The
reference implementation supports a 240x320 pixel color screen. In contrast, Crystal is designed for keyboard-based wireless devices with a 640x200 pixel screen in the reference implementation. Moreover, in November 2000, Symbian announced a third GUI layer reference implementation called Pearl, especially targeted at smartphones.

At the time of writing, the first Symbian-based smartphone became available commercially: the Ericsson R380s [Eri].

Microsoft’s Windows CE (Consumer Electronics) is an operating system belonging to the Windows product family targeted at embedded systems. Windows CE comes in different flavors for various device categories:

- palm-sized computers with pen-based input
- handhelds with pen- or keyboard-based input
- sub-notebooks
- industry terminals.

The basic idea behind Windows CE is to extend the success of the Windows operating system platforms to embedded devices. By providing a subset of the well-known Win32 API and retaining the Windows look-and-feel, customers shall be allowed a smooth transition between desktop and mobile device. Moreover, keeping a common API ought to permit a relatively easy migration of legacy applications. Besides being programmable in Microsoft eMbedded Visual Basic and C++, there is a PersonalJava implementation available.

Recently, Microsoft unveiled a project code-named Stinger, a development platform for smartphones, combining features of traditional mobile phones with those of personal digital assistants. Microsoft announced a cooperation with Samsung Electronics Co. Ltd. [Sam], developing a smartphone based on Stinger expected to be delivered during 2001.

Palm OS by Palm Inc. [Palm] is a third major competitor in the wireless operating system market. Originally being an OS for personal digital assistants, the scope of Palm OS now is extended to become an open platform to be licensed by third parties. At the time of writing, there are already some commercial smartphones employing the Palm OS platform, e.g. the Kyocera pdQ series [Kyo], and the VisorPhone by Handspring [Han].

The architecture of the Palm OS platform is shown in figure 10, emphasizing the openness for third party hard- and software. The Hardware Abstraction Layer shields the Palm OS kernel and system services from platform specifics. Applications for Palm OS can be written for example in C/C++, BASIC, and Forth. Moreover, there is a special port of the Java runtime
Applications in Mobile, Cellular Networks

Platform available. The KVM [Sun00c], which stands for *kilobyte virtual machine*, is a customized version of the Java 2 Micro Edition introduced in the following section.

Although strictly speaking Java as such is no operating system, it represents an important runtime environment with respect to (mobile) small devices. Especially targeted at consumer and embedded devices, Sun defined the Java 2 Micro Edition (J2ME) [Sun00b]. Basic building blocks of the J2ME platform are configurations and profiles [Sun00c]:

- **Configuration**
  A J2ME device configuration defines a minimum platform for a “horizontal” category or grouping of devices, each with similar requirements on total memory budget and processing power. A configuration defines the Java language and virtual machine features and minimum class libraries that a device manufacturer or a content provider can expect to be available on all devices of the same category.

- **Profile**
  A J2ME device profile is layered on top of (and thus extends) a configuration. A profile addresses the specific demands of a certain “vertical” market segment or device family. The main goal of a profile is to guarantee interoperability within a certain vertical device family or domain by defining a standard Java platform for that market. Profiles typically include class libraries that are far more domain-specific than the class libraries provided in a configuration.

With respect to smartphones and PDAs, Sun defined the *Connected Limited Device Configuration* (CLDC) [Sun00d] together with the complementary *Mobile Information Device Profile* (MIDP) [Sun00e]. The APIs defined in those specifications include—amongst other things—functionality related to persistent storage, timers, networking, and user interfaces tailored to small displays and limited input/output capabilities. At the time of writing, implementations of CLDC/MIDP permitting the execution so-called *midlets* exist for Palm OS and in the form of a mobile phone emulation called J2ME Wireless Toolkit.
Today’s smartphones still contain moderately powered processors and relatively small screens, when compared to notebooks. For example, the current high-end Nokia 9210 Communicator [Nok] shown here employs a 32-bit ARM9-based RISC CPU [ARM01] operating at clock speeds between 130 and 200 MHz. The display has a resolution of 640x200 pixels with 4096 colors. It offers non-volatile, exchangeable memory cards with 16 MB plus additional 8 MB of built-in SD-RAM. The 9210 employs the Symbian Platform, thus allowing for—or even encouraging—third-party application development. The suite of applications included encompasses an E-mail client, HTML and WAP browser, contacts, calendar, and many more. Moreover, a stripped-down version of Microsoft Office is available. Data communication speed is up to 43.2 kbps using HSCSD (cf. page 15). However, due to its size of 158x56x27 mm and its weight of 244 g, the Nokia 9210 is somewhat unwieldy.

A different smartphone example is the upcoming Trium Mondo [Tri] shown alongside. This device is an explicit combination of palm-sized PDA and mobile phone. It has an Intel 166 MHz processor and a grayscale display with 240x320 pixels. The Mondo uses the Microsoft Pocket PC operating system, a customized version of Windows CE specifically designed for PDAs. Stripped-down versions of Microsoft Outlook, Word and Excel are included, as well as the usual PIM applications. Web browsing is possible through a Pocket PC version of Microsoft Internet Explorer and a WAP-browser.

Future mobile equipment is expected to include much more functionality, mostly multimedia related (cf. figure 11). Due to the increased bandwidth to be expected at the latest with the introduction of UMTS, such applications like video conferencing are envisaged. Another trend goes towards integration of mobile communications technology into portable computers like notebooks. At the time of writing, Nokia already offers the Card Phone to be plugged into a standard PC Card slot.
Applications in Mobile, Cellular Networks

Middleware for Mobile, Cellular Networks

Smartphones and PDAs are well suited for running small- to medium-sized applications for everyday use. Mobile networks in turn generally provide a sound infrastructure for wireless communications, not only in terms of speech but particularly for the transmission of data. However, although the mere technical prerequisites for the deployment of such applications are met, or may well be expected to be met tomorrow, support from the software perspective currently is comparatively low. While sophisticated middleware for the development of distributed applications is available, existing technologies are almost entirely tailored for use on desktop computers connected to the (fixed) Internet.

Both the special characteristics of the end devices as well as those of wireless networks place distinctive requirements on the software layer, namely on middleware for applications in the context of mobile, cellular networks. The following sections will first give an overview of those special requirements and resulting challenges imposed by the specific properties and characteristics of the setting under scrutiny. Thereafter, core problems investigated further in the subsequent chapters of this thesis will be identified and introduced briefly.

2.4 Challenges and General Approaches

Mobile computing as envisaged in this work—mobile users employing small devices like smartphones in order to access location-based respectively location-sensitive services in the context of wireless, cellular networks—poses several challenges predominantly stemming from three system characteristics: the use of wireless networking technology, the mobility of users, and the utilization of restricted end devices [FZ94].

Wireless networking technologies, as opposed to their fixed network counterparts, suffer mostly from low throughput, high bandwidth variability, high latencies, and unreliability, respectively their susceptibility to sudden connection losses and temporary unavailability. An overall challenge is to strive for suitable techniques keeping applications working in face of all the diverse problems resulting.

According to [Raa+98], discussing the usability of GPRS as communication layer, it is reported that delay classes 2-4 and reliability class 3 will not offer acceptable GPRS service for mobile users. Under favorable radio conditions, however, GPRS links could be expected to provide a quality of service comparable to wired modem links. Hence, applications could still
work in a reasonable way, provided that they take the comparatively low throughput, high latency, bandwidth variability, and general vulnerability of wireless links into account.

Several well-known techniques addressing the deficiencies as set forth above exist. While most of them have a rather general background, however, the degree of their applicability may be application specific.

**Compression**

Data compression is a common strategy helping to save bandwidth. One possibility is to use generic compression techniques. The advantage of this approach is that it can be applied at lower protocol levels, fully transparent to the application layer. One prominent example for the use of generic compression are modems [ITU-42].

However, incorporating knowledge about the data to be compressed may lead to increased effectiveness and thus noticeable higher compression rates [LH87]. An example for such an approach is the binary coding of HTTP headers as used in the Wireless Application Protocol.

Especially with respect to the restricted computing power of mobile devices it shall be kept in mind, however, that software compression may induce a considerable processing overhead. Dedicated hardware support may be a suitable solution, as is done for example with respect to MPEG videos.

**Request Batching**

A general approach to cope with low bandwidth as well as high latencies is to batch multiple requests into a single data packet. This increases effective bandwidth usage due to smaller protocol overhead in terms of saved header information. The protocol employed by the X window system is a prominent example for this approach [Nye92]. Multiple client requests are buffered and sent in a single packet to the server. The problem with this approach is to determine proper points in time when to flush the request buffer. The more requests are combined, the more effective this scheme works, but the higher the risk for user perceived drop outs will be. Especially with respect to synchronous applications, this technique quickly reaches its limits.

**Balanced Communication**

Yet another general approach to cope with the problems at hand is to strive for balanced communications. Typical communication patterns include short peaks followed by longer periods of silence. During the peaks, bandwidth is overly exhausted, while it is spare over the rest of the time. Hence, communication shall be done in a more balanced way where possible, e.g. by postponing low priority requests.

**Caching and Pre-Fetching**

Caches are a prominent way of achieving this goal of balancing communications. On the one hand, the cache may store changes locally, deferring
Applications in Mobile, Cellular Networks

their propagation e.g. by means of lazy writebacks as done for instance by nearly all modern operating systems with respect to file system writes.

On the other hand, pre-fetching data into local caches may help in improving response times. The trouble with this approach, however, lies in the selection of appropriate heuristics. Generally, it is not possible to predict which data will be needed in the near future. If the wrong data will be pre-fetched, valuable bandwidth is wasted.

The Coda file system [KS92] is an evident example, designed to explicitly support users working in a disconnected mode as for instance portable computers connected to the network only temporarily. While being connected, files are hoarded on the particular machine, with the local file system acting as a kind of long-term cache. In the disconnected state, files are—whenever possible—served from this local cache. After reconnecting, changes are propagated back to the original servers, detecting and resolving conflicts when needed, probably with manual support from the user.

In the context of this work caching and, in parts, pre-fetching may well be applied in M-Solutions based on Web standards. However, with respect to distributed applications, the applicability of those techniques quickly reach their limits.

A general conflict intrinsic to mobile settings stems from the contradictory requirements imposed by wireless networks on the one hand and the resource limitations of the employed end devices on the other hand. Due to the unreliability of the communication technology, robust applications should continue to work even in case of connection losses. This suggests that most of the functionality should be executed on the client. However, this is in strong contrast to the limitations imposed by weak client devices, in turn implying to shift as most responsibilities as possible to the more powerful servers.

The extended client-server model introduced in [Sat95] accounts for this. Instead of cleanly separating the roles of client and server as done in the traditional client-server model, the distinction between client and server may be temporarily blurred. Due to resource limitations, functionality normally being performed by clients sometimes may be delegated to resource-rich servers. Conversely, in order to cope with unreliable networks, clients occasionally may take over the role of a server [JHE99].

Implementing the extended client-server model in a sensible way requires to dynamically (re)assign the roles of clients and servers, depending on the current situation. This process is called adaptation in [Sat95]. Two extreme approaches in this context are to leave the application itself full freedom in
deciding on its own, and on the contrary to let the runtime system determine the allocation of responsibilities. The spectrum of possibilities lying in between is collectively referred to as application-aware adaptation, described in depth in [Nob98]. The application thereby bases its decisions on information provided by the runtime systems.

Another central subject distinguishing M-Solutions from desktop computing is the ability of users to change their location, even while being connected to the network and accessing services.

A prevalent interpretation of user mobility is that of employing a portable computer changing network access points while on the move. With respect to traditional networks, this induces the need for dynamic reconfiguration at the network layer.

Mobile IP [Per96] is a well-known approach to this problem area. A general overview of how Mobile IP works is illustrated in figure 12 [Per97]. A mobile node is attached to the visited local network via a foreign agent and is given a temporary care-of-address. To the outside world, however, only the (static) IP address of the mobile node’s home agent is visible. Packets from foreign IP hosts are sent via standard IP to the home agent (1), intercepting and tunneling them to the care-of-address (2). The foreign agent detunnels the datagrams and forwards them to the mobile node (3). Data sent by the mobile node in turn use standard IP routing, delivering it directly to the destination IP host (4).

In the context of cellular networks, however, user mobility comes in a different flavor. Address migration as exercised by Mobile IP is not necessary. Rather, mobile devices are assigned an IP address that remains constant at least for the time of a single session [Ful00]. The provider’s network infrastructure is responsible for routing data packets to the mobile device in a fully transparent way without the need for dynamic reconfiguration at the network layer.
Mobile users not only want to be able to access static, location-independent services like for instance their e-mail account. Moreover, the need to retrieve location-sensitive information arises. For this kind of application, the term location-based services (LBS) has been coined. Prominent examples include things like finding the nearest vegetarian restaurant or identifying the particular store—within a limited region—offering a specific product at the lowest price.

Since a while there is a market hype regarding those location-based services\(^1\). However, even at the time of writing there are nearly no working LBS installed and in use. One of the primary reasons for the offers still missing is the lacking support for location determination [Wie00]. Although several different techniques are possible [Pra00], appropriate ones have to be determined and implemented by the network providers.

When developing applications to be used on top of smartphones or PDAs, the overall architecture has to take the restrictions imposed by those small devices into account.

Foremost, the limitations in terms of (relatively) slow processors and small memories mandate to reduce the level of functionality and its complexity to be executed on the mobile station. This aspect has already been considered briefly in the preceding section introducing the Extended Client-Server Model on page 27.

The size of typical displays is rather small when compared to traditional desktop computing. Wireless operating systems partly take this into account, for instance through dedicated, light-weight user-interface elements. Obviously, there is a wide range of devices with differing display capabilities. It would be advantageous to be able to tailor the content being delivered for each particular device employed. However, appropriate support for negotiation of device characteristics is only emerging as will be described in Capability Negotiation on page 153.

Finally, the battery life of mobile devices is severely limited, particularly when they are actively used and not in doze mode. Besides the aforementioned demand for relieving the mobile station from tasks that can be performed elsewhere, those devices have to be considered—in a sense—unreliable. Rather similar to sudden connection losses, mobile terminals may be powered down unexpectedly.

\(^1\) Several independent surveys state a steadily increasing global revenue reaching between “only” 18.5 billion dollars (as indicated by the UK market research firm Analysys) and about 40 billion dollars (according to Allied Business Intelligence) in the year 2006 [Ada01].
2.5 Core Investigations

There are a couple of different subjects to explore, both from the conceptual and from the technical point of view, at the application as well as at the protocol layer. The approach followed in this work has been to first identify several relevant topics, and to investigate them initially in isolation. In the course of exemplary, self-contained surveys and appropriate prototypes, each tailored to highlight particular aspects, context specific experiences have been gained and are reported on. However, because of the complexity of the overall problem area, and due to the broad spectrum of diverse technologies employed, together with the difficulties resulting from applying them jointly, it is not sensible for the time being to propose a single, coherent, sophisticated, and mature framework. In the ensuing discussion, the central issues driving and constituting this work will be introduced. Each topic thereby may refer to several of the aforementioned general challenges in parallel.

Wireless Web

When investigating different approaches for mobile computing, the Wireless Application Protocol (WAP) as the mobile counterpart of the ubiquitous World Wide Web may not be missing. A couple of questions arise in the context of WAP and its applicability as a technique for building M-Solutions. For instance, did the designers of WAP learn from the mistakes the WWW still suffers from today? To what extent are the (fixed) World Wide Web and its wireless counterpart compatible, both in terms of the markup language and the programming model employed? In the mobile world, there is a potentially high number of concurrent clients whose behavior must be expected to be comparatively volatile, making scalability an important issue. In this regard, it has to be answered whether there is suitable support for building scalable applications comprised of multiple cooperating servers covering larger areas.

Wireless CORBA

Besides hypertext-based computing, distributed object infrastructures represent a different yet distinguished option for the development and provision of services in the context of mobile networks. For reasons to be discussed in detail in the corresponding chapter, the Common Object Request Broker Architecture (CORBA) [OMG97b] has been chosen as a basis for the investigations. The applicability of CORBA with respect to wireless networks will be examined, concerning both the application as well as the protocol layer.

While in fixed networks an entirely synchronous programming model often is adequate, the unreliability of mobile networks manifesting itself e.g. in terms of sudden and temporary connection losses has to be taken into account. Hence, where appropriate, asynchronous communications shall take place, requiring suitable support.
As opposed to desktop computing where applications are typically installed statically, mobile settings tend to be more vigorous. Hence, deployment issues like options how to dynamically install client-side applications on mobile terminals are to be investigated. Due to the limitations of those devices and the costly communication links, the size of the code to be transferred plays a central role in this context.

Complementary, the underlying communication protocol, namely GIOP, has to be examined. Particularly with respect to bandwidth consumption, the design of the (native) protocol will have to be reconsidered. Moreover, an adaptation to the suite of protocols prevalent in today’s wireless networks, specifically those of the WAP protocol stack, will be performed.

In addition to those aspects regarding implementation strategies for single services and mobile clients, appropriate means for the publication of service offers are required. In this respect, techniques for service discovery are needed and will be discussed. As a special requirement, the current location of the user thereby has to be taken into account. Besides the ability to explicitly query for services, support for automated service announcements will be proposed.

Additionally, suitable support for managing server-side resources is crucial. As opposed to tightly coupled systems, where the administration of resources may become an integral part of the distributed application architecture—e.g. via dedicated interfaces and methods—this is not sensible in loosely coupled environments. Due to the intrinsic diversity and heterogeneity of a mobile setting, together with its vivid and unreliable nature, more liberal and fault-tolerant techniques for distributed resource management will have to be introduced.

As set forth above, scalability is important due to the potentially high number of concurrent clients. Thus, the issue of support for building a logically single service in terms of multiple cooperating instances will be reconsidered with respect to distributed object applications. In this regard, adopting the cellular design of today’s mobile networks will prove useful.

A specific property of mobile users naturally is their mobility, particularly becoming important when aiming at continuously using a single service over a longer time. There are several implications, one of them—as already mentioned above—manifesting itself in terms of sudden and temporary connection losses. Restraining on asynchronous notifications is sufficient and appropriate under certain conditions only. For instance when powering down the mobile device, additional measures have to be taken. In general, more flexible mechanisms are desirable and can be attained by placing a representative of the mobile client into the fixed network. Hence, an approach utilizing mobile agents as proxies will be introduced.
Another implication of user mobility in conjunction with compound service realizations exploiting cellular structures is the need to let the agent-based proxies follow their users. Suitable techniques supporting user tracking will be discussed. In addition to that, it will be shown that for different reasons it is sometimes advantageous to place multiple representatives into the fixed network, demanding for proper techniques to manage such agent ensembles.

The core issues described throughout the preceding section will drive the investigations and will be examined further on. Where appropriate, the remaining chapters will include specially tagged introductory sections elaborating on the current state of the art with respect to the topics at hand.
The World Wide Web in a sense represents the killer application of today’s Internet. Web browsers are almost pervasive tools, and hypertext-based computing is still gaining momentum. The big success of Web standards is tried to be carried forward to the mobile world in terms of the Wireless Application Protocol (WAP).

In this chapter, the Wireless Application Protocol and particular its applicability as a middleware for M-Solutions will be examined. After introducing WAP, a prototype application driving the investigations will be described. The experiences gained and relevant results are reported on.

3.1 Wireless Application Protocol

The Wireless Application Protocol (WAP) defines network protocols as well as an accompanying application framework, the Wireless Application Environment (WAE), targeted at making Internet services available over wireless communication links.

In 1997, a U.S. network operator called Omnipoint issued a tender for the supply of mobile information services [Buc00]. Four companies replied. The cell phone vendors Nokia [Nok], Ericsson [Eri], and Motorola [Mot] all suggested their own variants of messaging [Dor01]. Unwired Planet (thereafter Phone.com, now Openwave Systems Inc.) [OW] submitted the Handheld Device Markup Language (HDML) [KH97]. HDML is a deriva-
ative of the *Hypertext Markup Language* (HTML) [W3C99c], tailored to small, handheld devices.

Omnipoint did not accept proprietary solutions, encouraging the four bidders to team up and establish the WAP Forum [WAP]. At the time of writing, the WAP Forum has 250 full members and 389 associate members. The objectives of the WAP Forum are [WAP98]:

- To bring Internet content and advanced data services to digital cellular phones and other wireless terminals.
- To create a global wireless protocol specification that will work across differing wireless network technologies.
- To enable the creation of content and applications that scale across a very wide range of bearer networks and device types.
- To embrace and extend existing standards and technology wherever appropriate.

The WAP architecture, as depicted in figure 13, defines a layered protocol stack, aimed at providing a scalable and extensible environment for application development for mobile devices [WAP98]. The layered architecture enables other services and applications to utilize the features of the WAP stack through a set of well-defined interfaces.

The transport layer protocol of the WAP stack is called *Wireless Datagram Protocol* (WDP) [WAP00a]. It offers connectionless communications over different types of bearers, effectively providing a bearer independent interface for upper layers of the protocol stack. WDP supports circuit switched and packet switched transport protocols, as well as paging services like SMS. The specification explicitly aims at being open for the incorporation of even more kinds of bearers, e.g. Bluetooth [BT99].
Different kinds of bearer networks vary in the level of service they provide. The Wireless Datagram Protocol relies on the properties of a specific bearer, only augmenting functionality where needed. Sometimes the bearer may already provide higher guarantees than required by WDP itself. For example, circuit switched bearers preserve packet ordering (although not necessarily warranting reliability). In those cases, higher level WAP protocol layers may duplicate functionality, wasting network bandwidth and processing power. However, these situations are relatively rare, and tend to arise in networks where bandwidth is of lesser concern anyway [Sin+00]. One special case, with respect to the Internet, is the User Datagram Protocol (UDP) [Pos80]. UDP largely parallels WDP, allowing to use UDP/IP directly to achieve the service abstractions needed, as is the case with IP-supporting bearers, like GPRS, as specified in the GSM GPRS Profile in [WAP00a].

The Wireless Transport Layer Security (WTLS) [WAP00b] is a security protocol based upon the de facto industry standard Transport Layer Security (TLS) [DA99], which has formerly been known under the name Secure Socket Layer (SSL) [FKK96]. WTLS is optimized for use in narrow-band communication channels. It provides features for authentication, data integrity, and privacy.

The Wireless Transaction Protocol (WTP) [WAP00c] provides a lightweight, transaction-oriented protocol on top of WDP and, optionally, WTLS. WTP supports three kinds of transactions: unreliable one-way requests, reliable one-way requests, and reliable two-way request-reply transactions. Moreover, WTP allows for user-to-user reliability, forcing explicit triggering of confirmations of each message by the WTP user.

The Wireless Session Protocol (WSP) [WAP00d] provides the application layer with interfaces for session management. WSP allows connection-oriented sessions operating over WTP as well as a connectionless service both over secure or non-secure WDP datagrams.

The Wireless Session Protocol allows for long-lived session state and session suspend/resume in conjunction with session migration. At the beginning of each session, capability negotiation takes place. Essentially, this enables the session peer to cache relevant data as for instance constant header information for the lifetime of a single session.

Basically, WSP—or strictly speaking WSP/B—complements HTTP/1.1 on the wireless link. In contrast to the Hypertext Transfer Protocol, which uses a pure textual representation, WSP/B binary encodes well-known headers to reduce protocol overhead.
The Wireless Application Environment (WAE) is a general purpose environment, combining traditional WWW and mobile telephony services. WAE includes a so-called micro browser, a stripped-down version of ordinary Web browsers. Those browsers are capable of displaying content expressed through the Wireless Markup Language (WML) [WAP00e], a lightweight version of HTML, optimized for use in low-performance devices. WML is complemented by WMLScript [WAP00f], a restricted version of JavaScript [ECMA99] as known from desktop browsers.

WML introduces the notion of decks and cards. A single WML request delivers a complete deck, which is composed of one or more cards. The browser can display only one card at a time. Hyperlinks on each card can refer to other cards, either in the same deck or in a different one. This way, the content of a single document—being merged into an encompassing deck—can be presented to the user in small portions—split onto several cards—without the need for contacting the server multiple times.

The Wireless Application Environment differentiates between user agents and services/formats. WML, WMLScript, image formats, vCard [IMC96a], and vCalendar [IMC96b] are examples of the latter WAE component. The aforementioned micro-browser is a sample user agent, particularly the WML user agent. Concrete WAE implementations may choose to combine all the services into a single user agent, or may distribute the services among several user agents [WAP00g].

The Wireless Telephony Application (WTA) [WAP00h] framework further supplements the WAE. Through the Wireless Telephony Application Interface (WTAI) [WAP00i], entities in the WAE can access telephony related functions and the network telephony infrastructure. A WTAI public library implements this functionality, making it accessible through WML and WMLScript as well as for other applications running on the mobile device.

Figure 14. WAE Logical Model

The Wireless Application Environment adopts a model that borrows much from the classic World Wide Web. However, due to the nature of wireless communications and low-power mobile end devices, WAP strives at mini-
mizing the network bandwidth consumed as well as the processing power needed inside the mobile terminal. This is achieved mostly through employing a gateway sitting in between the client’s WAE user agent and the origin server, as shown in figure 14.

The gateway is responsible for decoding and encoding requests and replies, respectively. For example, when the client points its micro-browser to a specific URL, the WML user agent sets up a WSP link with the gateway. The gateway decodes the binary encoded WSP request, forwarding it to the origin server using plain text HTTP/1.1.

The reply, in turn, may be a WML document with embedded WMLScript, again sent back to the gateway in plain text via HTTP/1.1. The gateway encodes the document, translating well-known headers into a binary representation and compiling WMLScript into compressed bytecode. Documents may be pre-compiled as well, taking off workload from the gateway. The encoded response finally is forwarded to the client using WSP.

While older WAP versions allowed for client-initiated requests only, push services are now part of the specifications [WAP99b]. In the WAP push architecture, as depicted in figure 15, a Push Initiator (PI) transmits content, a push message [WAP99c], to the WAP client via a Push Proxy Gateway (PPG) [WAP99d]. The push message is transferred to the PPG via the Push Access Protocol (PAP) [WAP99e]. The Push Proxy Gateway, in turn, forwards the push message to the WAP client using the Push Over-The-Air Protocol (OTA) [WAP00j].

In the push framework, the Push Proxy Gateway plays a central role. The PPG first has to authenticate the Push Initiator. After checking whether the PI is allowed access to the push service, push messages have to be parsed, retrieving the address of the client, who has to be discovered prior to forwarding the message.

Basically, the push framework employs a store-and-forward technique. The Push Initiator performs a push submission by sending the push message,
containing besides delivery instructions a unique *push-id*, to the Push Proxy Gateway, buffering the message. The PI has the option of requesting confirmation of successful delivery, which is the case if and only if the target application has taken responsibility for the pushed content. Moreover, the Push Initiator may send *status query* and *push cancellation* requests, referring to a specific push message via its push-id.

The Push Proxy Gateway is responsible for trying to forward push messages to the client until a time-out expires, resulting in discarding the push message and sending a negative result notification back to the Push Initiator. Push delivery may be either connectionless or connection-oriented. In the latter case, the PPG relies on an active session with the client. In case there is no active session, the PPG ought to setup one. However, sessions can only be initiated by the client. To resolve this, the PPG sends a session initiation request to a special application on the client-side, the Session Initiation Application (SIA) [WAP00j]. The SIA checks whether the destination application of the push message, which is specified in the session initiation request, allows for push requests, initiating a session with the PPG or rejecting the request, as appropriate.

Push messages have to be addressed for a specific recipient or group of recipients, and for a particular client application. The latter can be determined by a dedicated header of the push message, *X-Wap-Application-id*. Recipient addresses are composed of a client specifier and a PPG specifier. The PPG specifier does not necessarily identify a physical PPG, and is not required to be the hostname of the PPG receiving the address from a Push Initiator.

Client specifiers utilize two alternative addressing schemes [WAP99d]: user-defined identifiers and device addresses. User-defined identifiers contain arbitrary values, mapped onto wireless network addresses in an unspecified manner. Push Proxy Gateway and Push Initiator have to agree on the semantics of user-defined identifiers. Device addresses use network addresses from well-known network address spaces, e.g. telephone numbers or IP addresses. The client specifier may as well lead to a point-to-multipoint delivery in the wireless network, e.g. a cell broadcast.

**Investigations**

After having introduced the Wireless Application Protocol, the following sections will investigate the applicability of WAP as a basis for the development of M-Solutions. Based on a prototype application, the advantages as well as shortcomings will be examined and discussed in detail.
3.2 A Mobile Group Schedule based on WAP 1.2

Within the context of a diploma thesis [Mau00], a prototype application based on WAP 1.2 has been developed. In particular, an asynchronous groupware realizing an appointment schedule has been the focus of the work. The motivation behind selecting this problem statement was its inherently asynchronous nature. It is very common for field workers—like for example sales representatives—having to meet other cooperating employees from time to time. However, interaction with the scheduling system should not be limited to times where the organization’s Intranet is directly accessible, e.g. from the office PC. The mobile phone as a common appliance is perfectly suited to act as a person’s interaction medium. The Wireless Application Protocol together with the push framework thus promises to suit the needs of applications of this kind.

The overall application architecture is depicted in figure 16, showing the flow of suggesting a new appointment as described below. A central appointment server stores the whole schedule. This database is accessed and modified via various PHP scripts [Bak+00] which also generate the WML documents to be sent back to the clients’ mobile phones.

The WML deck transferred to the users contains several cards, for viewing the current schedule, suggesting new appointments, and confirming or rejecting proposed ones. This way, the number of requests having to be sent to the appointment server is attempted to be reduced.

The application makes heavy use of the WAP push framework. For example, when the user suggests a new appointment, the following steps are...
taken (cf. figure 17). Firstly, the user enters the data of the new appointment, namely date and time, together with the desired attendees. The proposal is sent to the central appointment server, creating an appropriate entry in the schedule database. Thereafter, the proposal is forwarded to all attendees via the push proxy gateway. Each attendee replies, either conforming the appointment, suggesting an alternative date, or rejecting participation entirely. As an additional service, people can optionally enroll for a reminder, which will be sent as a push message as well.

Push messages can be sent to clients in different ways. The most simple form is to transfer a complete WML deck immediately. However, this has the drawback of the mobile client having to store the full data until it can be presented to the user—who currently may be busy, performing other tasks like for example entering a SMS.

**Service Loading**

A different possibility of pushing messages called *service loading* is to send an URL referring to the WML deck only. This way, the amount of data to be stored on the mobile terminal may be reduced. However, at the time the WML deck is to be presented to the user, it first has to be downloaded explicitly, resulting in an additional message.

A push message utilizing service loading for example looks like the following snippet:

```xml
<?xml version="1.0"?>
<!DOCTYPE s1 PUBLIC "-//WAPFORUM/DTD SL 1.0//EN" "http://www.wapforum.org/DTP/sl.dtd">
<s1 href="http://www.acme.com/pushmsg.wml"/>
```

**Service Indication**

Yet another approach is known as *service indication*. Similar to service loading, only the URL of the push message is sent. However, instead of downloading the WML deck transparently, a short descriptive message accompanying the service indication will be shown first. The user then may decide whether to retrieve the push message’s content or not. Additionally, the service indication may contain a creation as well as an expiry date after which to discard the push message silently. Moreover, each service indica-
tion may contain an identifier that may be used later e.g. in order to replace previously sent, outdated messages.

A push message utilizing service indication, namely notifying about the number of newly arrived e-mails, is shown below. In this particular case, for example, the possibility of replacing service indications based upon their si-id is a rather useful feature.

```
<?xml version="1.0" ?>
<!DOCTYPE si PUBLIC "-//WAPFORUM//DTD SI 1.0 //EN"
"http://www.wapforum.org/DTD/si.dtd">
<si>
  <indication
    href="http://localhost/wap/readmail.php"
    created="2000-08-25T13.16.15Z"
    si-expires="2000-08-30T00.00.00Z">
    si-id="notification@hoyaa.com"
    You have 2 new e-mails
  </indication>
</si>
```

At the time of its development, the technical prerequisites for a genuine realization, utilizing off-the-shelf hardware and existing mobile network infrastructures, unfortunately have not been met. Therefore, the prototype has been setup and built in the context of a WAP test environment.

The Nokia WAP Toolkit, Version 2.0 [Nok00], has been employed, offering primarily a WML browser with debugging support and a WAP gateway server. According to its specifications, this toolkit is fully WAP 1.2 compliant. Alas, especially the push functionality new to WAP 1.2 has been quite buggy.

### 3.3 Evaluation and Criticism of WAP

The appointment schedule presented in the previous section obviously has not been developed for real-world use, but rather to act as a prototype application allowing to evaluate the WAP technology. However, most of the lessons learned apply to the Wireless Application Protocol in general. In addition to the results drawn from the prototype, the following discussion also includes common experiences from third parties, where noted explicitly.

WAP technology is rather new, with the specifications being a lot of time ahead of the installed base of devices. For example, the WAP 1.2 specifications have been released around end 1999. The first conformant browsers have been appearing about one and a half years later, in mid-2001.
Middleware Support based on Web Standards

Although this is not a WAP-specific problem—new technologies typically emerge more rapidly in their infancy, resulting in products to lack behind considerably—especially in mass-markets like mobile communications, backward compatibility plays a major role. Thus, applications shall consider this by providing smooth migration paths. For instance, in the groupware application described above, non-push-capable devices may fall back on an inbox-like feature allowing to pull for new appointments. For application developers, this quickly results in a significant overhead.

Device Characteristics

When designing applications to be used on off-the-shelf mobile phones, their intrinsic characteristics have to be taken into account. For example, the need for typing in text shall be kept to a minimum. Instead, reasonable defaults or single- as well as multiple-choice selection lists should be preferred.

Moreover, while in principle it is a good idea to separate mechanisms from policies, this sometimes may prove to be counterproductive. For instance, while some display elements may be in-place editable, others open a separate dialog window. However, this is a browser-specific feature. Thus, a single WML form may look radically different on several browsers, making it a good design decision for one type of browser and a bad one for another. The degree of freedom the specifications leave for individual vendors tends to be a bit too excessive in this regard.

A different influence device characteristics may have can be illustrated by considering the visualization of appointments. While on a small display a pure textual representation in form of a list may be most appropriate (as shown alongside), another device may benefit from its bigger display. A more sophisticated, calendar-like view probably would be the better choice in such cases.

With WAP 1.2, the User Agent Profile Specification [WAP99a] has been introduced, allowing for device-specific capability negotiation. Admittedly, this may increase the implementation effort considerably, as at least different representations have to be generated. When further emphasizing the influence of device characteristics, even different levels of functionality could be supported.

WML Deck Size

Limitations

WAP-based conversations over wireless links are characterized by high latencies and low bandwidth. Thus, communication shall be kept to a minimum. The basic idea of transmitting multiple pages, or WML cards, once in a single enclosing WML deck accounts for this. However, due to limitations of current browsers and gateways, the size of encoded WML documents shall not exceed 1.4 KB [Jun00]. This is a severe restriction of the generally useful deck approach.
The user interfaces of WML clients naturally are restricted to the elements and tags provided by the markup language. For interactive applications, WML includes controls like input fields and selection lists. However, the look-and-feel of such applications greatly reflects their desktop counterparts in being forms-based. More sophisticated input/-output-mechanisms like for example interactive, graphical frontends are not an option with plain WML and WMLScript.

Moreover, client-side processing is fairly limited with pure WML applications. Even though WMLScript allows the use of variables, the abilities to perform computations on the mobile device is limited to simple calculations only. With respect to the characteristics of mobile networks, this is rather disadvantageous, as sophisticated applications in a sense quickly become dialog-based. Because data that has been entered has to be processed by the application server, a continuous request/response-style communication will result. The net effects are high bandwidth consumption together with the induced costs and repeatedly occurring and annoying user-visible latencies.

Although WAP can be seen as the transition of the Web into the mobile world, there are strong differences with regard to the markup languages employed. The traditional Web utilizes the Hypertext Markup Language, while WAP uses WML. Because HTML and WML are considerably different, Web pages must be formatted separately in both languages for being viewable in the respective browsers.

XHTML 1.0 [W3C00b], the Extensible HyperText Markup Language, basically is the reformulation of HTML 4 in XML 1.0. The accompanying XHTML Basic document type [W3C00c] includes a minimal set of modules required to be an XHTML host language document type. It is designed for clients with limited capabilities that cannot support the full set of XHTML features.

With XHTML and XHTML Basic at hand, it shall be possible to have a single origin document only. While desktop Web browsers could display the full content, a stripped-down view could be delivered to clients supporting XHTML Basic only. This view would be generated via some kind of filtering from the original document.

When accessing a service managing personalized data, it is typically required to identify the user sending requests. The most common approach is to let users enter a login name together with a password. Only after successful authentication will the user be allowed access to the service.
Entering sufficiently long user names and passwords on mobile phones is rather clumsy. However, user identification has to be performed at least once, the first time a service is called from the mobile device. One may benefit from a special characteristic of cellular phones, namely that most often they are used by a single person only and can be secured via a PIN to be entered when powering up. Thus, in cases like the groupware application where data is not overly sensible, it is a reasonable approach to store identification information locally on the device. It has been decided to let the appointment server generate a unique MD5 hash identifying each user after logging in successfully. The hash value is appended as a parameter to the bookmarked URL referring to the service.

Unfortunately, currently there is no standardized way to get any information identifying the user on server-side [Inf00]. Only proprietary solutions exist. The UP.Link WAP gateway for example appends a—proprietary—HTTP header field when forwarding requests to the origin server [OW00]. This header, called \texttt{x-up-subno}, contains a subscriber ID being unique across all UP.Link servers throughout the world.

User identification is closely related to session support. Typically, after a user has been authenticated, a new session is established. Unfortunately, WAP-based applications thereby inherit most characteristics of their Web counterparts, as will be discussed below.

The original design of the HTTP/1.0 protocol mandated to setup a new connection for every request [Ber+96]. Besides being a rather wasteful approach inducing performance and scalability problems [Spe94], it inhibits Web servers to track all requests of a single user without taking additional measures. The stateless nature resulting from this is in strong contrast to the needs of server-side applications, which typically need to maintain state across multiple subsequent invocations.

HTTP/1.1 [W3C99b] introduced a \texttt{Connection: Keep-Alive} header field, permitting to setup and maintain a single connection for a complete session. However, even today there are a couple of current Web browsers as well as servers not supporting this feature. Together with the need to serve legacy browsers, server-side application programmers are still forced to track sessions via different means, e.g. user authorization, hidden form fields, URL rewriting—as applied in the groupware application—and persistent cookies [HC98]. Each of this techniques has its own advantages as well as drawbacks. However, they all share several disadvantages.

Firstly, the need for explicit session expiry arises. Because there is no clean way to force users to logout, the typical approach to cope with the missing end-of-session information is to let sessions time out after a predefined
period of time. Secondly, and more severe, is the possible security hazard of sessions being compromised by third parties. Only by using a secure communication channel, this risk can be reduced somewhat. Finally, each of the strategies mentioned consumes valuable bandwidth, because some information enabling the origin server to identify a particular session has to be sent repeatedly.

Although WAP, namely the WSP/B protocol as the wireless counterpart of HTTP/1.1, establishes long-lived sessions, those sessions encompass the connection between the mobile station and the WAP gateway only. A single session thereby may last for multiple requests to different content servers. The session may even survive across power cycles of the mobile device. However, for the remaining part of the communication path, namely between the WAP gateway and the content server(s), the general properties of HTTP as described above are still valid, and the need for explicit session tracking remains.

Likewise to the user identification aforementioned, only proprietary solutions to the problem of session management exist. Some WAP gateways include a session ID with each HTTP request forwarded, allowing the target content server to avoid the use of cookies, encoded URLs, or similar techniques. The Kannel WAP gateway [WM01] for example appends the proprietary $x$-wap-session-id header field for this purpose.

The general WAE communication model as described beforehand primarily restricts the application infrastructure to a single server only. This is due to the fact that a complete WML deck has to be delivered back to the mobile device in a single response. In a more sophisticated setting where multiple servers shall provide content in the same context, application programmers basically have two choices, each having its own drawbacks, respectively.

The first approach is to let each server generate its own WML deck, independently from the other servers. The data for example could be sent to the client via the push architecture. One annoying side effect is that the user would be confronted with multiple different WML pages, each containing information from a single server only. Thus, the user would be forced to switch between multiple WML pages, having to be saved and managed manually on the device. In terms of ease of use, this is a quite clumsy approach.

In the second solution, the group of servers would have to be linked together, exchanging data and letting one dedicated server generate a single, compound WML document. Hence, the client would see information from different servers in a much more appropriate and comfortable way. However, there are a couple of disadvantages with this approach as well.
The need for linking the group of servers would decrease their autonomy. Additionally, a single server generating the compound WML document would have to be selected, requiring some kind of election algorithm to be applied. If the server group varies dynamically, things get even more complicated. Moreover, when the information of a single server changes, submitting the changed data would force to generate a new compound document. Thus, although only parts of the information have been modified, the full document would have to be sent, probably consuming much more bandwidth than actually needed.

A general criticism of WAP is induced by the central role the gateway plays in all WAP-based communications. The WAP protocols cover the link between client and gateway only. The gateway in turn is responsible for forwarding requests to the origin servers.

Thus, with respect to encryption, WTLS secures the wireless link only \[\text{Alg01}\]. In order to protect the second section of the channel, the gateway is responsible to translate between WTLS and TLS, or SSL, respectively. Hence, there is no end-to-end security between the client and the target Web server. Moreover, the gateway must have full access to the unencrypted requests as well as responses for encoding and decoding purposes. Accordingly, the gateway is vulnerable to man-in-the-middle attacks. Lastly, because the second section of the encrypted channel ends at the target Web server instead of at the application server itself, there is a third section in the whole communication path. The application provider is fully responsible for securing this remaining channel, which is completely independent of a possible WTLS/TLS encryption of the link.

Due to the first two security gaps described—no end-to-end trust and man-in-the-middle vulnerability—application providers as well as users are doomed to fully trust the gateway. Consequently, especially in security sensitive scenarios, service providers are forced to maintain their own gateway, increasing the overhead for WAP application deployment.

A conceptually minor issue, but one having a notable impact on real-world WAP-based application deployment scenarios again concerns the WAP gateway. In common mobile stations the WAP gateway is determined via particular device settings in a comparatively static manner—changing the gateway requires manual intervention like entering its address in a configuration dialog. Thus, independently of the URL entered, communication typically traverses one and the same gateway. Although employing a WAP gateway on the path between mobile client and origin server is mandatory—basically being an essential requirement in order to setup a communication channel—it shall be possible to change the gateway in a dynamic
way, depending on preferences as determined by the origin server. However, with current WAP technology, this is not possible.

Most mobile phones available today are WAP-enabled. Nevertheless, the success of the WAP technology is rather limited and its market penetration is still low. The term Wapathy has been coined, illustrating “the frustrations that have mounted this year over the limitations of calling up the World Wide Web on cell phones” [NYT00].

The main reasons behind this disillusion are twofold. Firstly, WAP as being available today is slow. The mobile network infrastructure suffers from low bandwidth. This property is predicted to change as soon as higher bandwidth links, e.g. through HSCSD, GPRS, or even UMTS become commonplace. The second drawback of WAP is its expensiveness. Most providers today charge based on the time a user is active instead of the amount of data that is transferred. Thus, both downsides of WAP essentially correlate. As soon as packet switched data transmissions, presumably with higher bandwidth, become possible, charging per data volume probably will result in lower costs, depending on the policies of the network providers.

Multiple studies further tried to investigate the reasons behind the low acceptance of WAP. The WAP Usability Report [Nie00] published by the Nielsen Norman Group contains the results of a test run of twenty people that obtained a WAP-enabled device for one week. Most of the test persons “definitely didn’t like” WAP. The Meta Group [MG] reports in their survey that 85 percent of the respondents are dissatisfied with the handling of their WAP phone. A survey conducted by A.T. Kearney together with the Judge Institute of the Business School of the University of Cambridge [Kea01] showed that 26 percent of the people interviewed didn’t even have an idea of what WAP shall be good for at all. Albeit the overall number of WAP-enabled devices has grown drastically, only 12 percent of the respondents intend to shop via WAP. In the last year, the share of m-commerce willing people has been 32 percent.

The Japanese telecommunication organization NTT DoCoMo [NTT] invented i-mode, which is comparable to WAP in its intentions, but with several differences regarding its technical realization.

One crucial difference between i-mode and WAP, apparently being one important reason for its much higher market penetration and success, is the markup language applied [Bat00]. Instead of using two incompatible languages for the desktop and mobile world, as is the case with HTML and WML respectively, i-mode makes use of C-HTML [W3C98b]. C-HTML essentially is a subset of plain HTML, thus reducing the effort needed to
make Web sites accessible from i-mode devices considerably when compared to WML.

Another big advantage of i-mode is that it is backed by a packet switched network [FT00]. This not only allows for volume based charging, making the fees induced by mobile Web access both time independent and often cheaper. Transmitting 100 bytes over i-mode is reported to cost about one cent and a half only [Bat00]. In addition to that, packet switching permits users to be always on instead of logging off and on repeatedly in order to save money. The—noticeable—time needed to setup a (fresh) connection each time can be neglected.

However, the bandwidth of up to 9.6 kbps available for i-mode is not higher than for current WAP access. This fact clearly indicates that speed plays a minor role only—provided the applications in use, or the content offered, are tailored for wireless use. According to Keiji Tachikawa, the president of NTT DoCoMo, “the winners … will be the people who are creative enough to find out what kind of content is most desirable in a mobile [phone] environment” [FT00].

![Figure 18. Shares of Most Popular i-mode Content](image)

In June 2001, i-mode had over 24 million subscribers, still increasing for about 50,000 additional users per day [NTT]. The share of the most popular contents according to an unofficial survey is shown in figure 18. Anyway, the definite killer application for i-mode—likewise in the initial years of Internet growth—apparently is e-mail [Eur01][NTT]. Having in mind that the mobile e-mail inbox is always on, thereby avoiding the need for polling, this is no surprise and parallels the European success of the—comparable but rather restricted—short message service.

### 3.4 Summary and Conclusions

The Wireless Application Protocol basically is an attempt to bring the World Wide Web—and its tremendous success—to the mobile world. However, WAP suffers for several reasons, and its market penetration is comparatively low.
One of the drawbacks of WAP stems from its intrinsic content related incompatibility with today’s Web, mostly due to the diversity of the markup languages used. Unfortunately, compatibility with other Web standards moreover hampers the implementation of sophisticated WAP-based services more than needed. One example of such a “legacy handicap” is the awkward session management. Although the WAP protocols explicitly support the notion of sessions, the origin servers have no standardized way to benefit from this. Only proprietary approaches exist. It is not obvious why compatibility with the WWW in this respect is retained, making it transparent for Web servers whether requests have been sent by (stationary) WWW or (mobile) WAP browsers. The content that has to be delivered is different anyway—HTML for the former and WML for the latter clients. Thus, the usefulness of WSP/B sessions is limited to the WAP gateway in terms of the ability to cache session-specific state like for instance relevant header information.

The central role of the gateway and the static way it is specified are problematic aspects of the Wireless Application Protocol as well. Especially with respect to security-sensitive applications, this must be considered as a major drawback. The static determination of the gateway, independently of the URL visited, moreover is one reason hindering the provision of scalable and secure services in the context of cellular networks.

The overall applicability of WAP as a general middleware for M-Solutions thus is fairly limited. This statement, however, applies to the hypertext-based Wireless Application Environment only. The WAP protocol stack itself still can be considered to be a viable approach, well-suited to be employed as a basis for application development in the context of mobile, cellular networks as will be shown in the subsequent chapter.
Middleware Support based on CORBA

Although the effort needed to develop M-Solutions based on Web standards is relatively low—as only the content has to be generated while the browser represents a generic runtime environment—distributed object applications are a promising alternative. Different approaches, respectively middleware platforms, exist today, each having its own strengths as well as weaknesses. All those technologies, however, are primarily targeted at the use in the context of fixed networks. This chapter will focus on investigating the applicability of state of the art middleware for building M-Solutions, identifying shortcomings, and proposing appropriate solutions, both at the application as well as at the protocol layer.

State of the Art

Middleware supporting the development of distributed applications comes in several different flavors. From a communication and programming model point of view, those technologies may be categorized—roughly speaking—into two diverse classes: Message-Oriented and Distributed Object Middleware.

4.1 Message-Oriented Middleware

Message-Oriented Middleware (MOM) may further be subdivided into two broad types: point-to-point and publish/subscribe. In the former, two peers communicate via a message queue, the central abstraction of MOM systems. Both peers are connected to the message queue. Single messages are put into the queue and it is the responsibility of the Message-Oriented Mid-
Middleware Support based on CORBA

dleware to reliably forward this message to the other peer. This way, the message queue provides a way for establishing an asynchronous communication channel, essentially decoupling the two peers.

In the publish/subscribe model, the message queue can be interpreted as a kind of bulletin board. Publishers post messages into the queue, and the Message-Oriented Middleware distributes those messages to all subscribers in a reliable way.

Explicitly isolating the peers is a distinctive characteristic of MOM systems. They require an unilateral application-level participation in message exchange only, resulting in highly robust systems insusceptible to failures. Message-Oriented Middleware thus is a very good choice for the implementation of applications that are inherently loosely coupled, as for instance information systems or other notification-driven applications.

However, Message-Oriented Middleware has its drawbacks. To begin with, MOM systems lack from well-defined standards, hampering interoperability and portability significantly. More importantly, Message-Oriented Middleware is often criticized for exposing a low level of abstraction that does not map directly to any programming model [RHK00]. Typical programming models adhere more to a synchronous invocation style. In case tighter integration of the communication peers is required, the (exclusively asynchronous) abstractions provided by MOM are not well suited.

4.2 Distributed Object Middleware

A different way to develop distributed applications is by means of Distributed Object Middleware (DOM). Layered between the application and the operating system, DOM provides—amongst other things—abstractions for the invocation of methods on objects residing on remote hosts. This way, the synchronous programming model commonly used today is augmented, enabling the collaboration across process and machine boundaries. Additionally, several distributed object middleware systems permit asynchronous remote calls. Thus, the advantages inherent in Message-Oriented Middleware may be retained.

In addition to the ability of easy interprocess communication, popular DOM implementations often provide a set of different, commonly used (generic) services. Naming, trading, and transaction services are typical examples. Developing distributed applications may benefit from those services in terms of reduced implementation effort. Furthermore, off-the-shelf
service implementations may well be assumed to be highly reliable and mature.

Recalling the *Extended Client-Server Model* introduced on page 27, distributed object middleware provides proper abstractions for their realization. As opposed to browser-based M-Solutions, with DOM it becomes possible to partition the responsibilities. Functionality may be implemented both on servers located inside the fixed network as well as on mobile devices, to a variable degree.

Different DOM products are available today, each with its own strengths as well as weaknesses. Three of the most popular distributed object middleware solutions will be discussed throughout the following sections.

*Remote Method Invocation* (RMI) [Sun98] basically provides a way to setup method calls to objects located on foreign machines. RMI is tightly integrated with the Java programming language. Classes that shall be exported must declare an abstract interface which extends the marker interface `java.rmi.Remote`. Concrete implementations in turn implement this interface, extending a particular standard base class from which they inherit the functionality needed to be callable remotely.

Clients refer to such concrete server objects via the exported interface only. The essential proxy code needed on client-side for purposes of parameter marshaling and remote method invocation will be downloaded dynamically when binding to a specific server. This ability of extending the code repository at runtime and in a secure way is a favorable property of the Java programming environment, and is a desirable attribute regarding the development and particularly the deployment of M-Solutions.

However, RMI has several drawbacks as well. One of the downsides is its restriction as a Java-only solution\(^1\). Another disadvantage of Java Remote Method Invocation is its limitation to synchronous calls. At the interface level, it is not possible to declare a method as asynchronous. Basically, asynchronous communication is possible, but requires the use of special APIs, namely the *Java Message Service* (JMS) [Sun99d], together with a JMS-compliant service implementation.

Microsoft’s distributed object middleware has evolved over time, being renamed rather frequently. Today, the technology is called COM+, an acronym for *Component Object Model*.

---

\(^1\) Although the communication protocol—the Java Remote Messaging Protocol—may be replaced by RMI-IIOP, thus allowing interoperability with CORBA, this in fact is an evidence that Sun identified the need for more openness and interoperability, basically confirming the decision of focusing this work on CORBA.
Basically, as the name implies, COM is designed for the development of self-contained components that may be reused in binary form. Components thereby may be synthesized in a hierarchical way, building higher level abstractions or components with more sophisticated functionality as a composition of several other sub-components. The interface of a single component may be described via the so-called Microsoft Interface Definition Language (MIDL).

COM+ explicitly allows components to interact remotely, a technique formerly known as Distributed Component Object Model (DCOM), now an integral part of COM+. Clients may perform method invocations on components being located on foreign hosts, provided they are exported properly.

Remarkably, the COM specification published on the Microsoft Web site has a version number of 0.9, dated October 24, 1995, and is explicitly tagged as having draft status only [MS95]. For COM+ in turn, there are a couple of white papers, surveys, presentations, and books available, but a specification is missing.

One downside of COM+ is that of being a proprietary solution only. COM+ is under full control of Microsoft and its platform support is in fact limited to the Windows operating system family.

The last distributed object middleware to be discussed is the Common Object Request Broker Architecture (CORBA). In contrast to Java RMI and COM+, CORBA is an open, vendor-independent architecture and infrastructure that computer applications can use to interoperate over networks [OMG97b]. The Common Object Request Broker Architecture permits the interaction of distributed components independent of their location, implementation language, operating system, communication protocol, and hardware platform. Moreover, CORBA includes explicit support for asynchronous communications, or messaging. Thus, CORBA has been chosen as the distributed object middleware best suited for the investigations regarding DOM-based M-Solutions.

4.3 Introduction to CORBA

The CORBA specifications are under control of the Object Management Group, a non-profit consortium with currently over 800 members, including virtually every large company in the computer industry [OMG]. Whereas the OMG is responsible for the specifications only, the industry has to produce CORBA-compliant products. While this approach is benefi-
cial in terms of openness and interoperability, the standardization process suffers from comparatively long round-trip times.

The OMG was founded in April 1989, publishing the first specification, CORBA 1.0, in 1991. At that time, only a C-based language binding has been available. Support for the C++ programming language has been developed by the OMG between spring 1993 and fall 1994 in about eighteen months. This C++ mapping has been published first in CORBA 2.0. Although there have been several revisions to fix minor flaws, the language binding itself has remained surprisingly stable [HV99]. By now, language mappings for Ada, C, C++, COBOL, Java, Lisp, Python, and Smalltalk have been defined. Whilst the core CORBA specifications are mostly stable, the overall standardization process is still in flux, constantly adding features.

The Object Management Architecture (OMA), shown in figure 19, defines the conceptual models upon which all OMG standards are based. The components identified by the OMA Reference Model include [OMG97a]:

- The Object Request Broker (ORB), enabling clients and objects to communicate in a distributed environment.
- Object Services are interfaces for general services that are likely to be used in any program based on distributed objects.
- Common Facilities are interfaces for horizontal end-user oriented facilities applicable to most application domains.
- Domain Interfaces are application domain-specific interfaces.
- Application Interfaces are non-standardized application-specific interfaces.
The general infrastructure of CORBA-based applications is shown in figure 20. Server applications may export objects to be callable over the network by clients. The interface of those server objects, or servants, must be declared in the standardized CORBA Interface Definition Language (IDL). Those IDL interface definitions are mapped onto a native language, using an IDL compiler provided by a concrete ORB product.

The IDL compiler also generates the (static) client-side stub and server-side skeleton code. Client applications then can make local invocations to the stub, synthesizing a request message through encoding the operation called and marshaling the parameters. The client-side ORB core routes the request message to the server-side ORB over the network. The skeleton in turn unmarshals the parameters and makes an upcall, invoking the appropriate operation on the servant. Correspondingly, results of the call take the opposite way back to the client.

Besides the ability to invoke operations on interfaces known at compile-time, clients may also call servant methods dynamically, via the Dynamic Invocation Interface (DII) and the Dynamic Skeleton Interface (DSI), respectively. Dynamic calls are always possible, without further ado. Interface definitions thereby can be requested from the Interface Repository (IR), a standard CORBA facility, provided that the server application has fed the IR beforehand.

The Object Adapter provides the glue between the ORB and the servant. The former Basic Object Adapter (BOA) has been replaced nowadays with the—much more sophisticated—Portable Object Adapter (POA). The core responsibilities of object adapters in general are generating identifiers for objects that are exported to clients and mapping subsequent client requests to the appropriate servants.

CORBA is a fairly sophisticated middleware, providing a broad spectrum of functionality based upon a substantial amount of specifications. Because CORBA implementations tend to be comparatively resource intensive, they are primarily not well suited for constrained environments. In response to
that, the OMG defined *minimum CORBA* [OMG01a], becoming an integral part of the specifications as of version 2.4.2.

The minimum CORBA profile explicitly strives for full interoperability with applications based on (original) CORBA. For instance, IDL should be supported without any restrictions. Several omissions, however, are proposed in order to permit finding a trade-off between usability on the one hand and conserving resources on the other hand. Minimum CORBA compliant products may choose which of those omissions to follow, probably with appropriate configuration options for the developers.

Among the features suggested to be omitted are for example support for exception handling and multiple inheritance as well as several seldomly used ORB and POA functionalities. The selection is based mostly on the influence those features have in terms of static ORB and stub code size.

### 4.4 Wireless CORBA

In 1998, the OMG—strictly speaking the OMG Telecom Domain Task Force—issued a Request for Information entitled *Supporting Wireless Access and Mobility in CORBA* [OMG98]. The objective of this RFI was, as the name suggests, to “seek information to help the OMG make useful and effective decisions in the technology adoption process to support CORBA-based applications in wireless and mobile networking environments as well as in the case of personal mobility”.

Several responses have been submitted to the OMG, e.g. a short discussion by Bellcore [Bel98], and a white paper from Alcatel, Ericsson, et al. in cooperation with several universities, edited by Kimmo Raatikainen from the University of Helsinki [Raa+98]. After the OMG made a Request for Proposal in mid-1999 [OMG99b], responses have been sent by Nokia, Vertel, et al. [NV+00], as well as from Inprise, Highlander, et al. [IH+00].

At the time of writing, the process has reached the status of finalization, and a draft adopted specification exists [OMG01c]. This draft specification includes mandatory as well as optional requirements. The former ones are addressed in the architectural framework described throughout the draft specification itself and are introduced below. In the course of the DOLMEN project [TB99], a prototypical implementation of this architecture has been developed, serving the purpose of a proof of concept.

The optional requirements in turn include mappings of the CORBA communication protocols onto wireless transport protocols, particularly WAP,
and the definition of protocol optimizations. Those optional requirements, however, are not being addressed directly in the draft specification and will be investigated in detail in the sections constituting the protocol layer part of this chapter, essentially complementing the work of the OMG.

**ARCHITECTURAL FRAMEWORK**

The architectural framework proposed in [OMG01c] and shown in figure 21 primarily focuses on the mobility aspect, with the wireless facet playing a secondary role only. CORBA clients as well as servers may be mobile. Such a mobile CORBA entity originates from a *home domain* to which it belongs from an administrative point of view. While on the move, it lives in the (wireless) *terminal domain* and accesses the network via a connection point located in the *visited domain*.

**Fig. 21. Architecture for Terminal Mobility**

Mobile CORBA servants are identified by special relocatable object references—so-called Mobile IORs—containing the address of the *access bridge* the hosting terminal currently is attached to. An access bridge is the (fixed) network side endpoint of a *GIOP tunnel* through which messages are exchanged with a *terminal bridge*. The *home location agent* is responsible for keeping track of the terminal’s current location. Sophisticated hand-off procedures are applied when changing access bridges. In order to be able to setup and terminate a connection with an access bridge and thus establishing or releasing a GIOP tunnel, dedicated messages are required. Those messages are part of the so-called *GIOP Tunneling Protocol* (GTP).

**The Application Layer**

The investigations regarding the applicability of distributed object middleware, particularly CORBA, for the development and deployment of M-Solutions are reported on in the subsequent sections. They can be subdivided in two parts, concerning the application and protocol layers, respectively.

---

1. GIOP is one of the CORBA communication protocols and will be discussed in detail in *Examining CORBA Communication Protocols* on page 69 ff.
Unfortunately, both the hardware as well as software base needed for sensible examinations in the setting envisioned have been either insufficient or even missing. However, with respect to the goal aspired, it has not been necessary to perform the investigations under real-world conditions. Although the studies are mostly based on prototypes being developed in the realm of fixed networks, the experiences gained from designing, implementing, and deploying those applications apply to mobile environments as well.

4.5 General Considerations

The purpose of the first prototype application described in full detail in [RS99] has been to investigate several rather general aspects regarding M-Solutions based on CORBA. Its functionality is very similar to the mobile group schedule described on page 39. The application realizes a simple calendar for resource allocation, particularly allowing to schedule the usage of a shared beamer and notebook, as required for online presentations. Basic functionality like looking up the calendar and entering new reservations is provided. Because neither functional completeness nor a sophisticated GUI would have led to additional results, all irrelevant aspects have been deliberately omitted. Although the importance of security issues like authentication and encryption is not underestimated, they were not incorporated in the initial prototype implementation.

The application’s design is based on a two-tier client/server architecture (see figure 22). There is a central server offering access to the schedule. On the client-side, the user accesses the calendar via an applet running in a Web browser. The communication between client and server uses the CORBA object bus. A callback mechanism is used to asynchronously...

1. At the time the prototype has been developed, midlets as introduced in Java 2 Micro Edition on page 23 were not available. However, regarding the investigations performed, the concrete realization variant is basically irrelevant.
inform the clients of relevant changes to the schedule induced by other concurrent users.

The server is implemented in C++. Since the server needs persistent storage mechanisms, a database—either relational, object-relational or fully object-oriented—has to constitute the backend. The latter has been chosen, namely the ObjectStore database system, as staying in the object-oriented paradigm lowers the impedance mismatch between the implementation language and the interface to the database itself [Kim90]. Furthermore, it allowed to reveal the problems of integrating CORBA and an ODBMS.

A subset of the entities managed on server-side, for example Person as well as Appointment objects, both are CORBA servants visible to clients and must essentially be persistent-capable. The latter is a term defined in the Object Database Standard [CB97], denoting classes that can have persistent instances. At the time the prototype application has been developed, only the now outdated Basic Object Adaptor (BOA) has been available. With the BOA, it has been comparatively difficult to manage persistency of CORBA servants.

The more recent Portable Object Adaptor (POA), which replaced the BOA, provides much more appropriate mechanisms for this purpose. In particular, when using the POA together with the complementary ServantManager, it now has become possible to activate CORBA servants directly out of the database.

An important decision regarding servants with a persistent state is the point in time when activation shall take place—and, correspondingly, when servants can be deactivated. The ServantActivator, one concrete type of ServantManager, provides two upcalls for this purpose, namely incarnate() and etherealize(). The first one is called whenever the object is currently not activated and a request arrives.

A suitable time when to call the second method, deactivating the servant, is much harder to determine. ORB implementations may base their decision on different criteria, for example after some period of quiescence or when the number of active objects reaches some limit. However, the application can give hints to the ORB via some non-standardized policies only, if any. Nevertheless, single servants can be deactivated explicitly by calling the deactivate_object() function on the managing POA. But the latter method, whereas being the only portable solution, forces the programmer to fully implement the—typically application specific—heuristics on its own. Unfortunately, there is no support whatsoever provided by the ORB. For example, it would be advantageous if statistical information about method invocations on single servants would be available.
A different problem area regarding persistence stems from the need to differentiate between the object-specific state to be made persistent on the one hand and the transient, CORBA-specific state on the other hand. Fortunately, CORBA provides a standard implementation strategy, the so-called tie approach, suitable for this purpose. The servant implementation thereby is delegation-based, factoring out the CORBA-specific functionality and state in a wrapper class automatically generated. This wrapper forwards all method invocations to its tie object, which now has to contain the object-specific logic and state only.

However, the separation of the upper CORBA servant layer and the lower persistence layer poses a couple of problems. One problem arises from differences in the datatypes to be handled in the different layers, while the other results from the strong interconnection between the two layers. The declaration and implementation aspects of the management of a person’s appointments are considered as an example covering both problems.

To store a person’s appointments, a corresponding type AppointmentSequence has to be declared via a typedef, referring to the OMG IDL-provided sequence type (cf. figure 23). Using a typedef is mandatory, as it decouples the name AppointmentSequence from its concrete implementation. The IDL compiler employs the type AppointmentSequence when generating the implementation skeleton class Person_skel from the IDL interface Person. The upper layer CORBA servant class PersonImpl, using the standardized inheritance based implementation approach, directly inherits from Person_skel and therefore uses exactly the same member signatures. The lower layer class PersonPersist on the other hand has to use a persistence-capable imple-
mentation for the set of Appointments. The type CPSEPtrArray has been
selected for this purpose, an ObjectStore-specific variant of the generic
container class CPtrArray. This class is provided by the Microsoft Founda-
tion Classes (MFC) application framework used in the prototype.

The implementation of the IDL interface Appointment uses the aforemen-
tioned standardized tie approach. Additionally to the skeleton class
Appointment_skel, the IDL compiler generates a template class
Appointment_skel_tie. An instance of this tie class forwards all requests to an
instance of the concrete implementation class given as the template argu-
ment type. The upper layer CORBA servant class AppointmentImpl simply is
a typedef, instantiating the tie template using AppointmentPersist as template
argument. No application specific code is needed to implement the upper
layer CORBA classes. Since the concrete implementation class AppointmentPersist is completely decoupled from the CORBA layer, instances may
be made persistent without incurring the overhead of storing any CORBA
specific state.

Examining the implementation to retrieve a person’s appointments clearly
shows the strong interconnection between upper and lower layer as well as
the necessary conversion and bridging functionality. From the client’s point
of view, only the declarations of Person, Appointment and AppointmentSe-
quence are visible. The client accesses a person’s appointments via a call to
the method appointmentSeq() automatically generated by the IDL compiler.
The implementation of this method in the CORBA servant class PersonImpl
has to return a sequence of references to Appointment objects. Therefore,
PersonImpl has to navigate the whole path through the persistence layer to
the CORBA implementation class AppointmentImpl, converting between the
datatypes described above as needed.

**CONTRADICTORY STANDARDS**

When combining CORBA, ODBMS and C++, three competing standards
have to be considered, each employing different data types. Firstly, the
CORBA C++ language binding [OMG99a] specifies the mapping of OMG
IDL types to C++ types. These may either be built-in types or data struc-
tures to be provided by the CORBA implementation. Secondly, the ODMG
standard [CB97] defines the language bindings for persistence-capable
classes. Finally, the ANSI C++ standard [ANSI98] specifies, amongst oth-
ers, the Standard Template Library (STL).

Although the integration of the three standards is a declared objective, it is
far from being perfect. Besides the example given in the preceding section,
the ubiquitous string class may serve as another sample. CORBA maps the
IDL string type to a char *, encouraging the use of CORBA::String_var for re-
ference counting purposes. ODMG 2.0 defines the class d_String to be used
for string attributes. Application frameworks widely used in the develop-
middleware process most often provide their own proprietary datatypes. In the case of the Microsoft Foundation Classes employed in the prototype, the corresponding class is called C<String> and is used extensively inside the MFC application framework, essentially making its use mandatory. Lastly, the Standard Template Library provides its own implementation, namely the template class basic_string<> together with its standard instantiations string and wstring. The latter classes are used for example in third-party libraries often employed in industrial-strength applications.

When using the tie approach mentioned above, the tie implementation has to obey exactly the same signature as the generated wrapper class, and thus the IDL types. In case the database does not allow instances of those types to be made persistent, the tie approach as a whole basically becomes useless. In addition to that, because of the other competing implementation variants, it is highly likely that error-prone user-defined conversion and bridging code has to be introduced.

It shall be noted explicitly that this problem has its roots foremost in the fact that the IDL data types are not part of the language standard—in this particular case C++. With programming languages like Java, where the standard libraries contain definitions for all IDL types, this problem supposedly vanishes as the same standard classes are most likely employed by the IDL language mapping, the object database, the application framework, and third-party libraries.

### 4.6 Callback Framework

In many application domains, it is desirable to be able to asynchronously inform clients about relevant changes. Examples for such reactive applications are a couple of prominent e-business applications like stock market, business opportunities, sales alerts and much more [EA00], as well as the beamer calendar and the group schedule aforementioned. In the latter, for instance in case a re-scheduling has to be performed, all participants concerned shall be notified.

CORBA offers the Event Service [OMG01b] for this purpose. The specification defines the notion of an event channel, providing a means to push or pull events from any number of suppliers to any number of consumers connected to the channel (cf. figure 24). The standard contains various interfaces needed to connect to and disconnect from channels as well as to distributed events. However, for example the creation of event channels is not covered by those interfaces. Furthermore, the standard provides no means to specify quality-of-service requirements, thus leading to propri-
etary solutions, depending on the ORB implementor. Although this does not prohibit the usage of the Event Service in general, it leads to undesirable proprietary realizations.

Decoupling

One advantage of using event channels instead of explicitly registered callback objects managed by the event originator is the higher degree of decoupling between supplier and consumer. Suppliers may push an event once via a single method invocation to the channel, basically delegating the responsibility of distributing the event to all consumers to the Event Service implementation. The overall distribution process has an asynchronous nature. However, each single interaction—between the supplier and the channel as well as between the channel and each consumer, respectively—essentially is a synchronous method invocation.

Quality-of-Service Control

For important events that eventually must reach all consumers, this is exactly the desired behavior. In CORBA, synchronous method invocations are the only reliable way to assert successful processing. Because failed synchronous calls throw an appropriate exception, after an event message has been (reliably) delivered to the event channel, the channel itself is able to track event delivery regarding each single consumer.

Unfortunately, the supplier has only limited—standardized—ways to control the delivery process, both in terms of reliability and ordering. In particular, using the standard interfaces, the supplier has no way to control whether an event has reached its destination on a per consumer basis. Furthermore, no guarantees whatsoever regarding atomicity are given—it cannot be ensured that either all or none consumer will receive an event. Additionally, the ordering of event delivery cannot be controlled—neither for events from a single nor from multiple suppliers.

Altogether, quality-of-service parameters have to be specified in proprietary ways, with a varying degree of control. For example, off-the-shelf Event Service implementations typically permit the event channel creator to specify properties like maximum number of retries and retry intervals only. These values apply in general and may not be adjusted on a per event basis. If a higher degree of control is needed, one has to rely on more sophisti-
Middleware Support based on CORBA

cated, proprietary Event Service implementations, providing functionality beyond that offered by the standardized interfaces.

Another interesting property regarding event channels is that both suppliers as well as consumers can decide independently of one another whether they want to push or to pull events. With respect to resource constrained consumers, this is a particularly useful feature. In order to use push-style semantics, the consumer has to register a callback object with the event channel. This callback object necessarily must be a—comparatively resource intensive—CORBA servant. In contrast to this, for pull-style interaction the consumer can act—in a more resource economical way—as a client only. However, the drawback is the need to explicitly poll for events, thus increasing the communication overhead. Generally, this trade-off has to be decided upon on an application and context specific basis.

It is not an uncommon application scenario to have a single supplier for an event channel only. Even in a setting with a single supplier and a single consumer—as for instance in case of a mobile station and a dedicated fixed-network proxy—the Event Service still is a sensible approach for asynchronous event delivery. In contrast to the conceptually overlapping oneway operations which have best-effort semantics only, event channels may provide a higher degree of delivery guarantees.

The design of the Event Service does not mandate whether event channels are realized as standalone processes or as local objects, e.g. sharing the address space with its creator, most probably being a supplier. From the point of view of resource consumption, in-process event channels are the much more attractive and scalable approach. Certainly, with an in-process Event Service implementation, the degree of decoupling between supplier and event channel is reduced, introducing a single point of failure. However, specifically in a setting with a single supplier and especially in case there is a single consumer only, the advantage of reduced resource consumption outweighs the drawback of suppliers crashing together with their event channel.

Particularly with respect to mobile networks, some distinct optimizations of event channel communication behavior would be possible. The topics presented below have suggestive character only. A reasonable implementation and ensuing evaluation would incur considerable overhead and rely on the cooperation with a network provider.

One possible improvement concerns the prevention of apparently unsuccessful event delivery retries. The network provider typically knows whether a mobile station currently is reachable or not. In particular, the provider can detect whenever a target re-establishes a lost connection. Thus, from a mere technical point of view and aside from legal aspects, it would
be possible to forward such notifications to Event Service implementations. This way, event propagation could be delayed either until the consumer has become reachable again, or the event timed out.

Another conceivable optimization would be to exploit the intrinsic multicast ability of mobile networks as described shortly in the section introducing GPRS on page 15. Especially in applications where a high number of users appear in a condensed fashion in a few cells only, like for example on exhibitions, sporting events etc., notifications could—at least in principle—be propagated to multiple consumers in a single over-the-air transmission only.

While the idea sounds rather simple, its technical realization is non-trivial. Among the noteworthy problems is the high degree of coupling needed between the Event Service implementation and the network itself. Information about the—probably frequently changing—location, namely the current cell serving a particular user, would have to be propagated to the Event Service. This is not only problematic from a legal point of view. Propagation delays quickly could result in hazardous behavior. A reasonable implementation approach thus would imply to make the Event Service implementation an integral part of the network infrastructure.

4.7 Deployment Issues

Besides the topics regarding the development of M-Solutions as described so far, appropriate deployment schemes have to be determined. Obviously, this type of applications is very dynamic in nature, opening the need for flexible as well as secure approaches. This section discusses a couple of related issues and problems, proposing possible solutions and evaluating their usefulness.

Installation Options

Due to the vivid nature of M-Solutions and the variability of user preferences, suitable mechanisms for dynamically downloading and installing the client-side code are needed.

Wired Installation

The approach taken by Nextel [Nex], a manufacturer of Java-enabled mobile phones, clearly would be inadequate for a highly dynamic environment as being envisaged. Java applications to be run on their mobile equipment first has to be downloaded via the traditional Web onto a personal computer. Thereafter, the code is transferred to the phone using a special adapter cable connected to the PC’s serial port. It is not possible to download and install applications while on the move using a wireless link. Moreover, checking for updates remains a responsibility of the user. Support for automatic upgrades is not an option.
A different and more dynamic approach using Java applets has been employed in the prototype application described in [RS99]. Although the target environment has been a traditional Web browser, the technical background and principles are comparable to a mobile setting. The Web browser is pointed to a URL containing a special APPLET tag. This tag specifies a location where client code can be downloaded from using the HTTP protocol.

However, this approach has a major drawback. Namely, the code will be discarded after use. The next time the applet is started—aside from possible browser-local caching—the code has to be downloaded again. Interestingly, the Java plug-in—optionally replacing the often buggy and non-conformant browser built-in Java virtual machine—introduced in its version 1.3 a special caching feature [Sun00f]. An additional tag called cache_option can be applied in the HTML code referring to the applet, instructing the plug-in to make the code sticky.

Although the plug-in provided mechanism is a slight improvement, it still is no fully satisfying solution, especially with respect to caching on small devices. One problem is that it is the designer of the Web page who determines whether to include caching instructions in the HTML code. Rather, the decision which code to cache locally shall be made by the end-user as opposed to the originator. A second problem is that the cache_option gives a hint only. The cache still remains under control of the plug-in. Instead, decisions on which code shall be installed and discarded in case of cache space running low shall be made by the end-user only, possibly with hints from the runtime-platform regarding statistical information.

Java Web Start [Sun01a] is a relatively new technology approaching this problem, and supposedly becomes an integral part of the upcoming Java 1.4 release. Full Java applications—as opposed to applets—may be launched directly from Web pages. However, the code first will be downloaded and installed locally on the client machine and afterwards executed on top of a standard Java 2 virtual machine. In subsequent invocations of the same application, Java Web Start first checks for updates, automatically downloading new code when necessary. However, currently there are no visible plans indicating that Java Web Start will be integrated into the Java 2 Micro Edition targeted at small devices.

The size of the code to be downloaded and installed is of great importance, especially when considering the limited storage capacities of small devices and the costly bandwidth. In the prototype application described in [RS99], the mere application code accounted for a very small portion of the overall size only. The applet, basically containing the application logic and GUI code, used about 50 KB only.
ORB Code

The vast amount of code has been library code, particularly the ORB of approximately 1 MB in size. However, it shall be noted that this is the size of a full ORB intended to be used on desktop workstations.

ORBs explicitly designed for use in limited devices are much smaller. For instance, a complete “Hello world” application written in Java and employing ORBacus/E [OOC01] is reported to have a size of 63 KB for the client and 92 KB for the server, respectively, including the ORB code. K-ORB, an implementation of the minimumCORBA specification, states an ORB size of about 40 KB for the client and 50 KB for the server.

Although these numbers are much lower than for the non-optimized desktop implementation, the overhead of dynamically downloading an ORB onto the mobile client each time a new application is requested shall be avoided.

PORTABILITY

In order to obviate the need of dynamically downloading an ORB onto the mobile client, portability has to be guaranteed. In particular, the code being developed on top of one specific ORB is required to run without modification on implementations of different vendors.

Java Clients

With respect to Java, the portability of pure CORBA clients depends mostly on the portability of the ORB interface and the stub code. The standardization efforts of the OMG pay off here. The stubs needed to get access to server objects are fully generated by an IDL-compiler, and have to refer to well standardized interfaces only. Namely, these are definitions included in the org.omg.* packages.

The concrete implementation classes backing those interfaces are instantiated on a dynamic basis, depending on the ORB instantiation. The decision which ORB to instantiate is driven by Java properties—either via system properties or through explicit specification inside the client code.

Thus, in order to retain portability as much as possible, the application developer has to follow two rules. Firstly, using proprietary ORB extensions has to be avoided—which is not a limitation particularly for pure CORBA clients. Secondly, the selection of the concrete ORB implementation has to be left a client-side decision—one simply has to leave the according Java properties unset.

Java Servers

For the server-side, things are a bit more complicated. The degree of interaction and thus of coupling between the application and the underlying ORB is much higher in case of CORBA servers. It shall be remembered that callback objects—as needed for example for push-style event channel utilization—already open the need to implement server-side functionality.
However, extensive tests have shown that even Java-based CORBA server implementations are (binary) portable across different ORB implementations—provided, of course, that only the standardized interfaces are used. Since inception of the Portable Object Adaptor, and in contrast to the outdated Basic Object Adapter, portability has become much easier to achieve and to retain.

The Protocol Layer

Besides the investigations regarding the design, implementation, and deployment of CORBA-based M-Solutions, the communication plays an important role, especially with respect to mobile networks and their intrinsic characteristics. In the remaining part of this chapter, the CORBA communication protocols will be examined and proposals for their adaptation, respectively their integration with current WAP standards will be described and evaluated.

4.8 Examining CORBA Communication Protocols

The Object Management Group specified a protocol to be used between communicating CORBA objects. In essence, the General Inter-ORB Protocol (GIOP) provides a general framework for a protocol suite, defining the basic the protocol structure only, particularly GIOP message formats and the Common Data Representation (CDR).

When information is exchanged between two communicating parties, both have to agree on a common data representation. CDR is a transfer syntax, mapping OMG IDL data types onto a bicanonical, low-level representation for on-the-wire transfer between ORBs. The features of the CDR as defined by the OMG are as follows:

- Variable byte ordering
  Sender and receiver may use different byte ordering schemes. The sender transmits the data using his own representation, indicating the byte order in the messages sent. The receiver, if using a different byte order, is responsible for byte swapping.

- Data alignment
  Within a single GIOP message, all data is aligned on its natural boundaries. For all primitive types, the CDR defines alignment policies, inducing alignment for composed, complex types.
Complete IDL mapping

The CDR provides representations for all data types available in OMG IDL, relieving application developers completely from the burden of data marshaling.

It should be noted explicitly that by default the data exchanged is not self-describing, so the receiver must have compile-time knowledge of the data types contained in a message. However, so-called *type codes* can be used to augment messages with descriptions of arbitrary data structures that are known at run time only. A *TypeCode* fully describes such potentially compound, complex data structures, containing both the data type of each single element as well as its identifier, e.g. the name of a structure or an individual member.

Besides primitive and compound data types that are known at compile-time, CORBA provides the ability to encode arbitrary data in so-called *any* values. Standard CORBA services like the Event Service [OMG01b] or the Property Service [OMG00a] for instance make extensive use of the *any* type, because their design has to be generic. Type codes are used when transmitting a value of type *any*, thereby opening the potential to introspect the data in question.

When invoking methods on a CORBA servant, plain data parameters are passed by value. However, the traditional semantics of object parameters is pass by reference only. Since the adoption of so-called *value types* in CORBA 2.3, objects can be passed by value, too. This way, full objects encompassing both state and behavior can be migrated to the servant, creating a local copy. Method invocations by the servant on the object passed are executed locally in the server, retaining the state of the original client-side object.

GIOP messages are predominantly composed on-the-fly, as most data to be sent is calculated dynamically. However, there are some exceptions to this rule, manifesting themselves in the form of so-called encapsulations. Basically, encapsulations are octet streams encompassing pre-marshaled data. For example, *TypeCodes* are perfect candidates for encapsulations, because once generated they remain constant throughout their lifetime. Thus, to lower the overall marshaling overhead, such data can be generated once and re-used in multiple GIOP messages.

Encapsulations can be interpreted as self-contained GIOP octet streams, obeying the same CDR formatting rules as any other GIOP data. Firstly, this means that data alignment instructions are strictly adhered to, taking the first byte of the encapsulation as offset zero.
A second implication of the CDR formatted encapsulations is related to byte ordering. Each encapsulation contains a flag indicating its byte order. This feature allows encapsulations to be transferred as-is across ORB domains, without the need for their unmarshaling and subsequent re-marshaling. Thus, a single GIOP message may well be composed of multiple parts with different byte orders.

The General Inter-ORB Protocol defines exactly eight distinct messages, falling in two categories: administrative and object invocation messages. Each message is composed of a (general) GIOP message header, followed by a (specific) message body.

The GIOP message header contains, among other administrative information, the GIOP protocol version in use, the byte order used, and the type of the message body.

The eight messages types contained in the body are as follows:

- **Request**
  When a client invokes an operation on a server object, a Request message is sent, consisting of a request header and a request body. The request header contains, among other information, a request_id uniquely identifying each outstanding request, the address of the target object, and the operation name, encoded as a string. The request body in turn encompasses the operation parameters to be sent to the server.

- **Reply**
  If an operation invoked through a Request message has to return any information, the server sends a Reply containing in its reply header, amongst other data, the request_id of the corresponding invocation and a reply_status. If the reply_status indicates success, potential data to be returned is sent as part of the reply body. In case of a failure, the reply body contains an exception.

- **LocateRequest**
  LocateRequest messages are used by clients to determine whether a server is capable of directly receiving requests for an object reference specified or, if not, to what address requests should be sent.

- **LocateReply**
  In response to LocateRequest messages, a server sends a LocateReply, designating the address of the server where to reach the object in question.

- **CancelRequest**
  Messages of this type are sent by clients, notifying the server that a response to a Request or LocateRequest message corresponding to a specified request_id is no longer expected.
CloseConnection
Through CloseConnection messages, an orderly termination of the communication is requested, e.g. in case of a server shutdown.

MessageError
Messages of this type are sent in response to any GIOP message that cannot be processed, e.g. because of a faulty message format.

Fragment
Sometimes it is impractical or undesirable to determine the size of a GIOP message to be sent in advance. In such cases, the whole transmission can be split, with the first snippet indicating the concrete message type, followed by one or more Fragment messages.

GIOP has to be mapped onto specific transport protocols. The baseline transport specified for GIOP is TCP/IP [Pos81]. However, in order to allow for a wide range of other GIOP mappings, the requirements placed by the General Inter-ORB Protocol on the underlying transport have to be as minimal as possible. In fact, GIOP is designed to run on top of any transport protocol conformant to a couple of assumptions [OMG01a]:

- The transport is connection-oriented. GIOP uses connections to define the scope and extent of request IDs.
- The transport is reliable. Specifically, the transport guarantees that bytes are delivered in the order they are sent, at most once, and that some positive acknowledgment of delivery is available.¹
- The transport can be viewed as a byte stream. No arbitrary message size limitations, fragmentation, or alignments are enforced.
- The transport provides some reasonable notification of disorderly connection loss. If the peer process aborts, the peer host crashes, or network connectivity is lost, a connection owner should receive some notification of this condition.
- The transport’s model for initiating connections can be mapped onto the general connection model of TCP/IP. Specifically, an agent (described herein as a server) publishes a known network address in an object reference, which is used by the client when initiating a connection.

The probably most-widely used concrete GIOP implementation is called the Internet Inter-ORB Protocol (IIOP), mapping GIOP onto the TCP/IP protocol suite.

According to [OMG01a], the design of GIOP and IIOP pursue vigorously the following goals:

¹ It shall be noted explicitly that only byte order within single messages has to be preserved. The standard does not require the ordering of multiple—autonomous—messages to be maintained.
Middleware Support based on CORBA

- Widest possible availability
  Both GIOP and IIOP are based on the most widely-used and flexible communications transport mechanism available (TCP/IP), and define the minimum additional protocol layers necessary to transfer CORBA requests between ORBs.

- Simplicity
  The GIOP is intended to be as simple as possible, while meeting other design goals. Simplicity is deemed the best approach to ensure a variety of independent, compatible implementations.

- Scalability
  The GIOP/IIOP protocol should support ORBs, and networks of bridged ORBs, to the size of today’s Internet, and beyond.

- Low cost
  Adding support for GIOP/IIOP to an existing or new ORB design should require small engineering investment. Moreover, the run-time costs required to support IIOP in deployed ORBs should be minimal.

- Generality
  While the IIOP is initially defined for TCP/IP, GIOP message formats are designed to be used with any transport layer that meets a minimal set of assumptions; specifically, the GIOP is designed to be implemented on other connection-oriented transport protocols.

- Architectural neutrality
  The GIOP specification makes minimal assumptions about the architecture of agents that will support it. The GIOP specification treats ORBs as opaque entities with unknown architectures.

However, the OMG also makes provisions for a potentially open-ended set of so-called Environment-Specific Inter-ORB Protocols (ESIOPs) (cf. figure 25). An ESIOP allows to leverage the capabilities of particular environments like for example specialized networking or distributed computing infrastructures, e.g. related to security. For instance for use in the context of the Distributed Computing Environment (DCE) [OSF-DCE] the (standard-
ized) DCE Common Inter-ORB Protocol (DCE-CIOP) has been defined. While ESIOPs may be optimized for use in particular environments, the interoperability between ORBs using different protocols still has to be guaranteed.

The ORB Interoperability Architecture defines a conceptual framework permitting ORBs of separate vendors—probably employing distinct protocols—to be made interoperable.

Logical groupings of clients, object implementations, and ORBs—based on security, product, application, administration, or any other meaningful characteristic—form so-called CORBA domains [RHK00] (cf. figure 26). Within a single domain, most often using ORBs of a single vendor, interoperability is handled transparently. Communication between different domains uses the concept of bridging. Through bridging, ORBs can interoperate without knowing any details of specific ORB implementations, such as what particular interprocess communication techniques or protocols are used to implement the CORBA specification.

A prominent way of bridging between ORBs is through IIOP. On either side connecting the domain to the outside world, each ORB uses an IIOP half-bridge. Being fully IIOP-compliant guarantees interoperability between ORBs of different vendors.

With respect to wireless communications as playing a central role in this work, the influence of unreliable and broken communication links is essential. By definition, as stated in the preceding section, the channel between client and server is assumed to be reliable. Consequently, no measures are included in the CORBA specification regarding corrupt messages.
However, any distributed system—whether fixed or wireless—suffers the possibility of tentatively or even permanently broken links. In order to cope with this, CORBA defines a couple of system exceptions, each including a completion_status code, taking one of the following values [OMG01a]:

- **COMPLETED_YES**
  The object implementation has completed processing prior to the exception being raised.

- **COMPLETED_NO**
  The object implementation was never initiated prior to the exception being raised.

- **COMPLETED_MAYBE**
  The status of implementation completion is indeterminate.

Additionally, one of the standard system exceptions, **COMM_FAILURE**, is raised if communication is lost while an operation is in progress, after the request has been sent by the client, but before the reply from the server has been returned to the client.

Client applications must be prepared to handle CORBA system exceptions. Consequently, communication failures propagate up to the application layer, becoming apparent as language level exceptions. The decision how to handle such exceptional cases is left fully to the application developer.

After having discussed the communication protocols as well as failure handling, this paragraph illustrates how calls from a client can be routed to a particular server object. For this purpose, CORBA defines so-called Interoperable Object References (IOR). IORs essentially are opaque byte sequences, used only to allow addressing information on server objects to be exchanged between different applications. Two general ORB operations allow to generate object references, represented externally as a plain string, as well as the possibility to let the ORB generate a local proxy from such a stringified IOR.

In order to allow an ORB to correctly manage object references, an IOR has to contain several information. First of all, the type of the object—basically, its IDL interface definition—is specified in the type_id field. The object type is followed by one or more server object addresses, each encoded as a TaggedProfile. This data structure contains a ProfileId, a tag identifying a particular protocol. For example, **TAG_INTERNET_IOP** is the predefined constant tagging IIOP related endpoint information. The second component of a TaggedProfile is the protocol specific body called profile_data. For IIOP, this profile body mainly contains the hostname and port of the server process plus an opaque object_key. The object key is an ORB specific chunk of
bytes, uniquely identifying a single server object within the context of the
object adapter managing the servant.

**DISCUSSION**

Beginning with CORBA major version 2, the OMG spent much effort to
pave the way for ORBs of different vendors to be fully interoperable. The
goals driving the design of GIOP obviously reflect this.

**CDR**

The Common Data Representation—which is mandatory for any GIOP-
compliant protocol whereas each ESIOP may define its own data representa-
sion scheme—for instance is not well suited for low bandwidth networks.
For example, the encodings of numerical types are of constant size. With
respect to fixed networks, this is a sensible design decision. However,
packed representations could save space while consuming comparatively
low additional CPU resources.

**Alignment**

Moreover, the alignment policies mandated by CDR are disadvantageous
for networks where bandwidth is scarce. Although there is a processing
overhead resulting from the need to interpret misaligned data, communica-
tion over costly and slow wireless links certainly should strive to reduce the
amount of data to be transmitted.

**Type Codes**

Minimizing bandwidth consumption clearly has not been an objective in
the design of GIOP. One example for this is the fact that any values are
always accompanied with their TypeCode information, incurring a notice-
able protocol overhead. This not only provides an introspection capability,
it additionally enforces type safety on the receiver side. However, the
importance of both features is questionable. With respect to type safety, it
shall be kept in mind that for example primitive parameters of an operation
cannot be type checked anyway, because their type code is not included in
GIOP messages. Moreover, introspection supposedly is a capability used
rarely. However, in standard CORBA, the application programmer has no
control of whether to include type codes or not.

**Encapsulations**

Another consideration regards encapsulations. Remarkably, although the
data is aligned correctly inside the encapsulation, alignment rules may be
violated when being viewed from the outside, at the level of the surround-
ing full GIOP message. The reason behind this is that the encapsulation
itself—since being represented as a sequence<octet>—is forced to begin at a
four byte boundary only. To guarantee correct alignment of the data con-
tained in the encapsulation as well, the encapsulation would have to start at
an eight byte boundary instead. Thus, the rationale behind data alignment
rules is not followed consistently.

In contrast to the statement given above, it shall be kept in mind that encaps-
sulations take advantage of the fact that the Common Data Representation
contains standardized data type encoding rules. Without them, the need for
Middleware Support based on CORBA

differently encoded encapsulations arises, thereby lessening the benefits gained from pre-marshaling.

Another shortcoming of GIOP with respect to bandwidth consumption stems from the header information repeatedly transferred in each message, foremost the four bytes magic header—containing the string “GIOP”—and the GIOP protocol version. While the inclusion of the magic header allows for simple detection of non-GIOP messages, this is deemed to be unnecessary, because GIOP per definition is connection-oriented. Thus, each connection could well be associated with its own session, defining the communication parameters—in particular, the GIOP protocol version and byte ordering—in a setup phase. While only a few bytes per message could be saved when abandoning those header fields, each eligible opportunity should be taken to save bandwidth when communicating e.g. over wireless links.

Furthermore, GIOP has a gossipy nature, which has its advantages as well as drawbacks. One example are operation names contained in GIOP request messages, being transferred as plain strings. In terms of bandwidth consumption, this is a rather wasteful approach. As both the client and the server know the interface definition of the object being called, the IDL compiler would be able to generate a unique numerical identifier for each method, even in case of inheritance being applied. While such numerical identifiers would be in favor of bandwidth savings, they have a significant drawback in terms of interface evolution. As long as the signature of the method in question does not change, the server interface may evolve, e.g. by adding new operations. Because the method being called is identified by the operation name, the call may be processed correctly. Albeit from a pure theoretical point of view changing an interface is bad practice, and e.g. inheritance—leading to a completely new interface—shall be applied instead, the behavior as is allows for more silent interface evolution.

The endpoint information encapsulated in Interoperable Object References is explicitly designed to enable the inclusion of multiple endpoints, each employing its particular protocol. This way, a single server object can potentially be reached via multiple protocols. Environment specific protocols may define their own addressing schemes via appropriate profile bodies. In order to preserve interoperability, the OMG has to adopt the IDL definition of the profile body, assigning it a uniquely identifying tag. The binary representation of a single IOR is rather big, varying in some degree between different ORBs. The variation in size stems from the fact that object keys, being part of each IOR, are ORB-specific entities. Moreover, the IIOP profile_data optionally may contain other ORB-specific information, encapsulated in so-called TaggedComponents. Such components
may be included to support ORB services such as security, and to embed additional information that may be used in making invocations on the object described by this profile.

For example, the size of the IOR of an exemplary HelloWorld servant is 152 bytes when using ORBacus for Java [OOC00], while Visibroker for Java [Inp00] encodes the IOR in 100 bytes. The mere endpoint information could be reduced to a few bytes only, as the IP address and port of the server process, together with identifiers for the managing object adapter and an index of the object itself would be sufficient. Thus, ORBs targeted for use in environments with scarce bandwidth should generate more concise Interoperable Object References.

Through the possibility for third parties to define their own ESIOP, ORB technology may be used in network domains outside the TCP/IP-dominated world of the Internet. Only through definition of an ESIOP it becomes possible to specify a protocol optimized for use in the context of wireless protocols, taking their special properties into consideration.

4.9 Evaluating WAP as Basis for an ESIOP

While CORBA as such has become an accepted and widely used standard in the field of distributed object middleware, the communication protocols—as has been described in the previous section—are not well-suited for wireless networks. The WAP stack yet defines a protocol suite designed explicitly to cope with the special characteristics of mobile networks. This chapter examines the WAP protocol stack, revealing its strengths and weaknesses with respect to CORBA communications.

In order to fulfill the requirements CORBA places on the underlying transport—along the lines of the GIOP Transport Assumptions introduced on page 72—the Wireless Datagram Protocol (WDP), the Wireless Transaction Protocol (WTP), and the Wireless Session Protocol (WSP) have to be utilized in conjunction. Optionally, links can be transparently secured by incorporation of the Wireless Transport Layer Security (WTLS) protocol entity.

WDP only guarantees delivery of single messages while preserving the byte order. WTP adds reliability, and the basic WSP provides the possibility to manage connection-mode sessions. In the following paragraphs, the protocol layers are described in more depth, with special focus put on their integration with CORBA communications.
WDP provides upper protocol layers with a consistent datagram service, adapting to a specific underlying bearer network where needed. As has already been mentioned shortly in section WDP on page 34, UDP is adopted directly to achieve the abstractions required for any wireless bearer network where IP is used as a routing protocol [WAP00a]. WDP supports simultaneous communication instances from a higher layer over a single underlying WDP bearer service. Port numbers are used to identify the higher layer entity above WDP where to forward packets to.

The Wireless Datagram Protocol—when used with an IP-based bearer like GPRS—inherits all features of UDP/IP. For higher layers, this is mostly interesting with respect to addressing and routing purposes. Basically, applications on mobile units may use regular IP addresses and UDP port numbers. This applies both to entities located on the mobile unit as well as to objects located in the fixed network. Regarding the CORBA endpoint information—respectively the profile body contained in an Interoperable Object Reference—this means that basically the IIOP profile can be reused. However, while the port specified in the IIOP profile designates a TCP endpoint, WAPIOP could utilize the similar UDP port.

If WDP is used with a non-IP-capable bearer, things become more complicated. In principle, endpoint information for server objects located in an IP-based world can remain unchanged, still opening the need for protocol bridging. However, server objects located on mobile units would demand for their own addressing scheme tailored to the bearer network in use. For instance, the mobile subscriber ISDN number (MSISDN) uniquely identifying each user could be used in place of an IP address. Consequently, bearer-specific IOR profile bodies would be required.

The Wireless Transaction Protocol, to be used on top of WDP, adds reliability through its support for transactions. Messages sent via WTP can be categorized into three classes [WAP00c]:

- **Class 0**: unreliable invoke message with no result message
  Messages of this class are essentially simple datagrams, as those sent by the underlying WDP layer. No functionality is added by the WTP protocol layer. The purpose of this class is to permit users to send datagrams—e.g. unreliable push messages—within the context of an existing session.

- **Class 1**: reliable invoke message with no result message
  Messages sent as WTP class 1 datagrams can be considered as “reliable push” datagrams. The protocol uses acknowledgements and re-transmissions to make the communication reliable. A single Invoke message is sent from the Initiator (sender) to the Responder (receiver), who sends back an acknowledgement, maintaining state information for some time
about the message to enable handling of possible re-transmissions. If the Initiator doesn’t receive an acknowledgement after a timer expires, the Invoke message is re-sent. The state information in conjunction with unique sequence numbers—so-called transaction identifiers (TID)—is used to detect duplicates and lost messages. Transactions can be aborted explicitly at any time.

Class 2: reliable invoke message with one reliable result message

Messages of class 2 provide basic invoke/response-style communications. A single Invoke message is sent from the Initiator to the Responder. The Responder replies with exactly one Response message implicitly acknowledging the invocation. The Initiator in turn acknowledges the Response message. Similar to class 1 messages, state information held at the Initiator as well as the Responder in conjunction with transaction identifiers enables detection of duplicates and lost messages. Furthermore, in order to prevent the Initiator from unnecessary re-transmissions in case of delayed responses, the Responder may notify the Initiator of requests being in process by sending so-called hold on acknowledgements.

The Wireless Transaction Protocol layer furthermore defines a user acknowledgement feature. A flag, the U/P flag, determines the authority being responsible for the initiation of acknowledgements. If this flag is clear, the WTP provider—basically, the WTP protocol layer implementation itself—is allowed to send acknowledgments itself. In case the U/P flag is set, the WTP user—the application using the WTP protocol layer—is explicitly forced to confirm Invoke and Response messages, respectively. This way, user-to-user reliability can be enforced.

Discussion

The Wireless Transaction Protocol, like any other layer of the WAP protocol stack, is explicitly designed to take the special characteristics of wireless links into account. This is reflected in a couple of design decisions.

First of all, WTP is connectionless, obviating the need for explicit setup or tear down phases. The communication is message oriented as opposed to being a mere byte stream. With respect to an integration of CORBA with WTP, message orientation matches the communication paradigm very well. The three transaction classes provide communication primitives for all basic interaction styles, namely unreliable and reliable data push as well as reliable method invocations (with an accompanying result). For example, ordinary synchronous method invocations cleanly match onto class 2 transactions. Operations tagged as oneway in CORBA IDL having an invocation semantics of at-most-once, best-effort, not guaranteeing delivery of the request could be realized either as class 1 or even as class 0 transactions.
Reliability as offered by WTP can give partial guarantees only. The protocol is—and only can be—designed to cope with transient failures of the underlying transport. If at any point in time the wireless link is broken for longer than a specific amount of time, depending on the bearer network, no full guarantees may be given. Before giving a detailed example for clarification purposes, timers and counters used in the protocol have to be explained further.

The WTP standard [WAP00c] defines some bearer-specific default timer and counter values controlling the handling of acknowledgements and message re-transmissions. Those values take the distinct characteristics of the particular bearer into account, like estimated round-trip times. For example, with respect to SMS, a median round-trip time of 10 seconds and a maximum round-trip time of 40 seconds is assumed. When sending a class 2 Invoke message, the base retry interval timer is proposed to be 60 seconds. For bearers supporting IP, the maximum round-trip time is assumed to be 3 seconds, leading to a recommended base retry interval of 5 seconds. Remarkably, although the Wireless Datagram Specification [WAP00a] explicitly defines a GSM GPRS Profile, the higher layer WTP specification does not refer to the GPRS reliability and delay classes introduced in table 1 and table 2 in section GPRS on page 15.

While with SMS a maximum of four re-transmissions is proposed, IP-based bearers are allowed a maximum of eight retries. Consequently, in case of SMS as bearer, the network may be unavailable for approximately up to 4 times 60, or 240 seconds. With IP-based bearers like GPRS, this duration is 8 times 5, or 40 seconds. After that time, the transaction outcome will be determined internally by the WTP providers, independently on the Initiator and Responder side, respectively. Those comparatively short durations underpin the statement of WTP being biased against transient failures only.

![Incomplete WTP Class 2 Transactions](image)

**Fig. 27.** Incomplete WTP Class 2 Transactions

---

81
A detailed, transaction class 2 based example shows three scenarios of incomplete transactions in the event of the network being cut off for a longer time than masked e.g. by re-transmissions (cf. figure 27). In the first case, the link is broken after the `Invoke` message has been sent. Despite of re-transmissions, the Initiator cannot decide whether the invocation has arrived and has been processed by the Responder.

In the second case, the `Invoke` message has been acknowledged implicitly by the Responder through sending a `Response` message. However, the Responder isn’t sure whether the result has reached the Initiator, because the respective acknowledgement will never arrive.

The third and last scenario applies whenever the communication link is broken when the Initiator tries to acknowledge the arrival of the `Response` message. Although the acknowledgement has been sent, the Initiator cannot be sure whether it has been seen by the Responder. However, the harm being induced is rather low, because the primary purpose of the final acknowledgement is to allow the Responder to throw away transaction related information, most notably the `Response` message itself.

All of the scenarios described may occur in any distributed system, where faulty links are an inherent problem. The major difference between fixed and wireless networks is the quality of service being considerably diverse, influencing time-out values being significantly higher. With respect to CORBA, the communication layer must take the WDP timers into account before throwing a `COMM_FAILURE` exception. Users perceive the consequences whenever the wireless link is broken through a stagnant, or apparently “hanging” application.

**WIRELESS SESSION PROTOCOL**

The Wireless Session Protocol or, strictly speaking, the WSP/B protocol in its native form closely resembles HTTP/1.1 [W3C99b], optimized for low-bandwidth bearer networks with relatively long latency. In particular, WSP as specified in [WAP00d] provides means to:

- Establish a reliable session from client to server and release that session in an orderly manner.
- Agree on a common level of protocol functionality using capability negotiation.
- Exchange content between client and server using compact encoding.
- Suspend and resume the session.

When setting up a WSP session, service peers use a capability negotiation scheme, mutually agreeing on an acceptable level of service. The WSP client or *Initiator*, typically a WML browser, thereby dictates the maximum set of capabilities. The WSP server in the role of the Responder may only
accept or further restrict the level of service, but may never reply with a capability setting implying a higher functionality than proposed by the Initiator.

The economy of WSP/B in terms of bandwidth usage is primarily reflected in the protocol design through extensive use of binary encodings. For example, header fields like the HTTP status code, content-type, and language each are represented as a single octet. Moreover, the space of well-known header field names is partitioned into so-called header code pages, with every code page defining a limited number of single octet binary encodings only. A header code page shifting feature can be applied to switch between multiple code pages. The WAP Forum specified a default header code page encompassing all HTTP/1.1 headers. Applications in turn can define their own code pages.

The main purpose of the WSP/B protocol is the invocation of HTTP methods like GET, HEAD, PUT, and POST. WSP/B defines specific messages, or protocol data units (PDUs), for each of these methods. The GET PDU for example contains fields for the Uniform Resource Identifier (URI) plus headers associated with the request. All of those messages are sent as payload of appropriate WTP transactions. The GET method, being one of the request-reply style invocations, for instance is mapped onto a class 2 transaction.

Two distinct primitives, in particular the SUSPEND and RESUME protocol data units, constitute the WSP Session Resume facility. With the SUSPEND primitive, typically the Initiator (but in general the Responder as well) may temporarily freeze a WSP session. At a later time, the session can be resumed. The time a suspended session is retained is a local implementation matter. Through the session resume facility, WSP sessions are made independent of a single transport layer connection.

Within the context of integrating CORBA with the WAP protocol stack, setting up a WSP session accommodates the requirement of the underlying transport being connection-oriented.

The WSP specification [WAP00d] explicitly states that “the Wireless Session Protocol is a session-level protocol family for remote operations between a client and proxy or server”, thereby emphasizing the architectural openness of WSP. Nevertheless, WSP/B as currently defined is tightly bound to HTTP/1.1 style interactions. For example, the primitives belonging to the Method Invocation facility contain dedicated fields for the request URI, request and response headers as well as a request and response body.
Although the basic request/reply paradigm from HTTP applies well to a vast majority of client/server systems, the concrete communication protocol may be radically different from the one defined in the WSP/B PDUs. Consequently, WSP as-is only may provide a basic framework for implementing CORBA-style communications. However, the protocol mismatches just described demand for the definition of specially tailored protocol data units, resulting in an adapted WSP protocol version to be described within the following section.

### 4.10 WAPIOP—A WAP-based Inter-ORB Protocol

After having discussed the general properties of CORBA client/server interactions and the characteristics of WAP with respect to GIOP-style data exchange, this chapter introduces WAPIOP, an optimized protocol for CORBA communication over wireless links.

WAPIOP in fact integrates CORBA with WAP. It extends CORBA by a Wireless Data Representation (WDR), a more space efficient encoding than the original Common Data Representation. Moreover, it defines new message formats, tightly integrated with the Wireless Session Protocol, through specification of new PDU types. The resulting protocol is called WSP/C, a shorthand for WSP/CORBA. The Wireless Data Representation in conjunction with WSP/C constitutes WAPIOP.

The following sections, after presenting the relationship to CORBA standardization work, will discuss basic considerations of the protocol design, together with different implementation strategies. Thereafter, the Wireless Data Representation and the modified GIOP message formats will be introduced. WAPIOP has been implemented prototypically, allowing to investigate specific properties of this protocol. A comparative example follows this chapter, illustrating the bandwidth savings of WAPIOP against native IIOP.

**Relation to CORBA Draft Standard**

Basically, WAPIOP is supplementary to the Wireless Access and Terminal Mobility draft adopted specification from the OMG [OMG01c], addressing the optional requirements defined therein. Particularly, a mapping of GIOP onto wireless transport protocols respectively a wireless/mobility specific ESIOP are requested for. The draft specification explicitly asks for an integration with the WAP protocol stack. Hence, WAPIOP basically can be interpreted as a proposal addressing this particular optional requirements.

**Basic Considerations**

A basic decision when designing new protocols is whether to retain backward compatibility. However, with respect to WAPIOP, this would mean to
Middleware Support based on CORBA

create a semi-optimal solution only. Unfortunately, as has been described in the previous sections, neither CORBA nor WAP (or WSP/B) provide the degree of extensibility necessary for the purposes at hand. Namely, the rather gossipy CDR is an integral part of the CORBA specification, leaving no room for supplementary or alternative encoding schemes. Furthermore, WSP/B is tailored to HTTP/1.1 style communications.

Thus, the decision has been to retain backward compatibility where feasible, but without compromising efficiency. The proposals therefore violate parts of the current CORBA specifications, yet trying to minimize their impact as far as possible. With respect to the Wireless Session Protocol, WSP/C can be seen as a mere extension to WSP/B, using PDU types in the reserved space. Existing implementations—both ORBs and WAP protocol stacks—shall be adaptable with comparatively low effort only. Moreover, the approach proposed here is not intended to replace existing and proven technology, but rather to augment it.

The design of protocols for over-the-air data transmissions has to be driven primarily by the specific characteristics of such communication links. In particular, scarce bandwidth, high latency, and unreliability have to be taken into account.

The Wireless Data Representation strives at optimizing the space requirements of CORBA communications. As has already been noted, WDR violates the CORBA specification, as its counterpart, the Common Data Representation, is a mandatory part of the communication protocol design. Obviously, using only a single encoding scheme is in favor of interoperability, as ORBs are interoperable simply if they are CDR conformant.

However, a one-size-fits-all solution in most cases is almost ever a bad compromise only. With respect to CORBA communications, the trade-off between message size and processing overhead is a crucial factor. The design of CDR clearly shows that message sizes have not been the driving factor. Because bandwidth consumption is of utmost significance in wireless networks, different prerequisites apply here. Thus, it is proposed to allow different encoding schemes to coexist, and to be selected on a per connection basis. The Wireless Data Representation therefore shall be seen as an alternative to the Common Data Representation, rather than as a replacement. The consequences of supporting multiple protocols in parallel are discussed throughout the following sections.

The message formats of WSP/C closely resemble those of original GIOP, but with several modifications and additions, again for the sake of bandwidth savings. Thus, the messages as defined in WSP/C are not binary
compatible with GIOP. However, the changes affect the lower protocol layers only, without inducing incompatibilities at the application level.

Another consideration influencing the design of WAPIOP regards the trade-off between bandwidth savings on the one hand and the processing power to be spent for data compression on the other hand. For example, the GSM General Packet Radio Service provides a maximum transfer rate of approximately 170 kbps (see also GPRS on page 15). A single byte therefore is transmitted in about 47 µs. Current state-of-the-art smartphone CPUs operate on approximately 150 MHz and above, resulting in a clock rate of at most 6.6 ns. Accordingly, at least 7120 processor cycles elapse per byte transferred. With UMTS and a maximum transfer rate of approximately 2 Mbps, this value still remains to be about at least 600 processor cycles. Besides the relatively high processing power being sensibly available for data compression, there is the cost factor. Users will be charged for every byte transferred over the provider’s wireless links.

**IMPLEMENTATION STRATEGIES**  
Basically, there are two different yet supplementary strategies to save bandwidth in GIOP communications. The first, rather simple solution is to retain GIOP and the CDR entirely, while applying (generic) compression algorithms on the data. The other approach—and the one followed by the initial design of WAPIOP—is to optimize the data representation and message formats, hence defining an Environment Specific Inter-ORB Protocol. Of course, compression may be applied additionally, trying to reduce the size of the data to be transmitted even further.

For the first solution—generic compression—no information whatsoever on the data to be transferred is needed. CORBA provides standardized ways to achieve this kind of task. **Interceptors**, as shown in figure 28, are interposed in the communication path between client and target object. Two types of interceptors are defined [OMG01a]:

- **Request-level interceptors**, which perform transformations on a structured request, and
- **Message-level interceptors**, which perform transformations on an unstructured buffer only.

Request-level interceptors receive decoded invocations as CORBA::Request objects, both on the client- and on the server-side. After performing some transformations, the interceptor may subsequently re-invoke the request using the Dynamic Invocation Interface. Accordingly, the result first reaches the interceptor, which again may take appropriate actions. Request-level interceptors for instance may be used for services such as transaction management, access control, or replication.
In contrast, message-level interceptors only see unstructured chunks of data, without further knowledge of their content. Thus, functionality like encryption or compression are areas of application for this kind of interceptor.

Object Request Brokers that use interceptors must know which of them may need to be called, and in what order they need to be called. An ORB that supports interceptors, when serving as a client, uses information in the target object reference, as well as local policy, to decide which interceptors must actually be called during the processing of a specific request sent to a particular target object. This information is agreed upon during a so-called client-target binding process.

The second approach mentioned beforehand—optimizing data representation and message formats—unfortunately cannot be realized sensibly via the interceptor technique. On the one hand, only requests could be handled, not the other types of GIOP messages. On the other hand, client-side interceptors are supposed to use the Dynamic Invocation Interface for request processing, resulting in the generation of a GIOP Request message to be sent to the target nevertheless. The interceptor generally has no clean way to access the communication channel to the server ORB directly. Moreover, first setting up a CORBA::Request object which subsequently has to be interpreted by the interceptor, translating it into a different format, is rather inefficient anyway.

In summary, changing data representation and message formats demands for tight integration into the ORB core, while compression alone could be implemented using interceptors only.

In its basic design, the proposed Wireless Data Representation follows the Common Data Representation only with respect to the fundamental feature
of complete IDL mapping, specifying a representation for all basic IDL types. However, the remaining two features, namely variable byte ordering and data alignment are not inherited directly from the original CDR specification.

The main rationale behind data alignment, according to the CORBA specification [OMG01a], is “to allow primitive data to be moved into and out of octet streams with instructions specifically designed for those primitive data types”.

On some processor architectures, misaligned data accesses incur a performance penalty. For example, the Intel Architecture Optimization Manual [Int97] states that misaligned accesses in the data cache or on the bus cost between three and twelve clock cycles. The ARM processor architecture [Sea01] as another example generates an alignment fault, prohibiting the immediate access of misaligned data using specific machine instructions completely.

However, all this applies solely when reading the data directly from a buffer, an approach only being applicable with languages providing a low-level memory interface, like C or C++. Through an appropriate abstraction, namely a stream-based interface, the problem of misaligned data is reduced to a mere performance issue. This technique anyway must be used by languages not supporting pointers, like for example Java.

The bandwidth waste induced by data alignment can be significant. The maximum alignment boundary is eight—signed and unsigned long long integers as well as double and long double values have to be aligned on addresses being evenly divisible by eight. Thus, in case of single byte values like char and octet, up to seven bytes may be wasted due to alignment rules.

The application programmer typically is neither aware of the effects of data alignment nor has a clean way of actively influencing its implications. For example, depending on a couple of factors not under control of the developer, either of the following functionally equivalent two method declarations may result in a shorter GIOP Request message:

```java
void foo(in octet o, in double d);
void bar(in double d, in octet o);
```

If the data contained in the Request message prepending the method parameters ends at an eight byte boundary, `bar` would be the better choice. In this case, no alignment gaps would be necessary. Otherwise, `foo` would take advantage, because the first parameter, octet `o`, would fit in the alignment gap needed for the `double` value anyway. However, there is no way to
decision which declaration to prefer—and it is generally a sensible decision not to expose such low level details to the application developer.

Although data alignment may be reasonable to be applied in high bandwidth networks, it soon becomes counterproductive in the wireless domain. Thus, the Wireless Data Representation abandons data alignment policies entirely.

The primitive types defined by the CORBA specification are: octet, char, wchar, signed and unsigned short, long and long long, float, double, long double, boolean, and enum.

Boolean, Octet and char are encoded just as in the Common Data Representation, as single byte values. Wide chars (wchar) as well follow the CDR for GIOP version 1.2, as a sequence of octets.

The unsigned integer data types (unsigned short, unsigned long, and unsigned long long) are encoded as variable length unsigned integers (varuint), equivalent to the Wireless Session Protocol [WAP00d] data type (cf. figure 29).

The unsigned integer value is split into 7-bit fragments and put into the Payload field of multiple octets. The Continue bit is set if another octet follows and clear in the last octet of the sequence. Because a conversion between variable length unsigned integers and native values is needed anyway, byte ordering can be fixed, independent of the underlying processor architecture. WDR, just as defined in WSP, uses big-endian ordering, placing the most significant bits in the first octet and the least significant ones in the last octet.

For signed integer data types (short, long, and long long), the first octet is augmented by a Sign bit. The Payload field contains the absolute integer value. The resulting type is called variable length signed integer (varint) (cf. figure 30).

Floating point data types (float, double and long double) inherit the native IEEE standard formats [IEEE85], likewise the Common Data Representation.
Besides the primitive types, encoding schemes for IDL constructed types have to be defined, namely struct, union, sequence, enum, string, and fixed. Besides the latter, the Wireless Data Representation inherits the original CDR definitions. No new data type encoding schemes are defined therein, only structural attributes. For instance, union values are encoded as the discriminant tag, followed by the representation of any selected member. Solely the WDR primitive data type encodings influence the representation of the aforementioned IDL constructed types. For example, enumerations (enum), encoded in CDR as unsigned long, are represented as varuint in WDR. Similarly, the unsigned long length field of string and sequence types, specifying their length, inherit the WDR-defined varuint encoding.

Fixed-point decimal types (fixed) are encoded as a varint, comprising the signed integer part, followed by a varuint containing the fractional portion. This way, rounding errors—as in CDR as well—are effectively prevented.

In the original Common Data Representation, byte ordering applies to all integer and floating-point types, as well as to fixed-point decimals. WDR indeed leaves only floating-point values in a byte-order dependent format. Thus, byte ordering remains a property having to be agreed upon between two communicating parties.

To recapitulate, variable byte ordering as defined in CDR lets the sender of a message dictate the byte ordering scheme, leaving possible byte swapping a responsibility of the receiver. Because mobile stations can be assumed to be the computationally weaker entities, the burden of byte swapping should be shifted to the machines located in the fixed network whenever possible. The session management feature introduced by WAPIOP, being described in section Session Setup on page 93, arranges for this.

As long as byte-order specific data types exist, all communicating entities must be prepared to understand both byte ordering schemes. The reason behind this is that otherwise, if for example two mobile entities using different byte orders want to interact, a bridge translating messages would have to be introduced. This bridge, while generally possible, would be required to have full knowledge of the message formats and content. For example, in order to be able to translate a Request message, the IDL interface definition of the operation called, together with the type information of all parameters, would have to be available. Moreover, the bridge would have to see the message in unencrypted form, introducing a possible security hole.

With respect to WAPIOP, encapsulations are problematic. With the original definition of the Common Data Representation, it makes sense to pre-marshal data that, once generated, remains constant. However, the usefulness of
encapsulations is diminished when some prerequisites are no longer met. In particular, the encoding rules of CDR and WDR differ, resulting in the need for multiple encapsulation instantiations. Byte order is another case, as each encapsulation determines and encodes its own ordering scheme at the time of pre-marshaling, independent of the demands of a specific WAPIOP-based client.

Generating yet another encapsulation instantiation, one for each byte ordering scheme, obviously is not viable. Because WAPIOP-based units must be prepared to understand both byte orders anyway (as has been discussed in the previous section), retaining the approach of determining the ordering scheme at the time of pre-marshaling results in a possible slight overhead on the receiver only.

Another problem area arises when encapsulations with different encodings are used, resulting from the need to translate between them. For example, a (generic) CORBA Event Service may have to distribute an event, particularly a value of type any, which contains its TypeCode as an encapsulation. Both the event producer and the consumers either may communicate using IIOP or WAPIOP with the Event Service. Thus, the service has to act as a bridge, translating the TypeCode encapsulation between different encodings back and forth.

As long as structural information about the content of the encapsulation is known, at most a small processing overhead will be induced. The risk of potential information loss arises only in case there is no such structural information available. However, GIOP and IIOP explicitly use encapsulations in three places only [OMG01a]: type codes, the (IIOP) protocol profile, and in service-specific contexts. All of them are fully specified in CORBA IDL, making their structural information available globally. Thus, only non-standardized, ORB-specific encapsulations are vulnerable to information loss. However, no interoperability guarantees are given for proprietary items anyway.

Altogether, the approach taken by WAPIOP is to retain encapsulations as mechanism, but to allow for different internal encoding schemes. The byte order remains to be determined at the time of pre-marshaling. Although the encoding scheme applied can be non-ambiguously derived by both peers from the protocol in use, type safety can be enhanced without overhead by including appropriate information inside the encapsulation. Instead of indicating only the byte order in a flag at the beginning of each encapsulation, as is done in original CDR, an additional bit identifies the encoding scheme: CDR or WDR. Of course, a more general approach would include a standardized encoding tag field. This way, encapsulations remain to be self-contained entities.
Any

In contrast to the original CDR, type code information for any values is available in WAPIOP on request only. This approach bears the risk that type code information will not be available at the time it is needed. Consider, for example, an any value that is transferred via a WAPIOP connection to a mobile station. After the connection has been closed, the any shall be forwarded to a different machine, requesting the type code. However, such a scenario is rather unlikely, because application level knowledge shall allow to identify cases of that kind.

In CDR, a value of type any is encoded as the TypeCode of the data enclosed, followed by the data itself. Thus, each any is a self-describing value, encompassing its own type plus its value. This strategy allows an any to read itself from a CDR encoded message, as the size of the data is known through the type description.

In WDR, the TypeCode prepending the data enclosed in the any is replaced by a session-specific tag, followed by an octet sequence containing the data itself. On the server-side, this any_tag is mapped onto the corresponding TypeCode for the duration of the session. Encoding the data as a sequence<octet> preserves the ability of an any to read itself from an incoming message stream, because the overall size of the data is known. However, the attribute of an any being self-describing is lost. Nevertheless, this can be considered a minor drawback only. The application programmer extracts the concrete value from the any just as before—typically not referring to the type code anyway—but without inherent type safety. Anyhow, all other values being extracted from the data stream are not type checked as well.

Type safety optionally can be enforced by requesting the TypeCode from the server before trying to extract the concrete value contained in the any. Basically, all this requires a small modification in the concrete implementation of the abstract any interface only. A single additional operation has to be included in the interface: request_type(). As has already been noted, this invocation may fail, thereby raising an exception.

It shall be noted explicitly that this approach implies the need for additional, WAPIOP-specific messages, enabling the client to request the type code information and the server to provide it.

Value Types

The ability to pass objects by value, as described shortly in Value Types on page 70, generally is a valuable feature. However, due to their relative complexity, value types have been omitted from the initial design of WAPIOP.

Object References

Interoperable Object References do not have a dedicated representation. Instead, they are entities composed and structured around primitive types.
Thus, no immediate encoding rules need to be specified for object references. However, alternative strategies for saving bandwidth may be considered.

In well-designed applications, a single object reference will be transferred only once, not multiple times. Thus, a caching scheme where IORs are tagged with a session-specific identifier and are transmitted in full length only the first time, using the short tag in subsequent messages, does not seem to be worth the overhead. A slightly modified solution would be to send only such a—newly generated—tag in the actual message, forcing the client to request the full object reference data on demand, e.g. when invoking a method on the object denoted by the IOR for the first time. However, this would only make sense under the assumption that most object references will never be used, while inducing the overhead of an additional inquiry message otherwise.

Yet another approach would be to include different endpoint informations in multiple profile_data structures. This way, a dedicated WAPIOP profile could be designed, explicitly designed to contain the most basic information only. However, overall message sizes would be reduced only when profile_data is transmitted on a selective basis. This in turn not only would require ORBs to modify IORs on-the-fly, on a connection-specific basis. Moreover, it would result in a loss of interoperability, because different versions of basically the same IOR would exist. As IORs are transmitted to WAPIOP-based units, other profile information, like IIOP profile_data, would be stripped off, inhibiting the IOR to be forwarded to an IIOP-based machine, for which it has become useless.

The approach taken by WAPIOP so far is to re-use the IIOP profile_data, relying on the savings stemming from the more compact representations of primitive data types. No caching or tagging as described beforehand is applied. However, there is one implicit assumption. While the port contained in an IIOP profile designates a TCP port, WAPIOP employs the similar UDP port.

Besides the encoding rules described so far, the message formats are optimized for use in wireless networks as well. Moreover, the messages are tightly integrated with the Wireless Session Protocol, defining dedicated PDU types in the reserved space of WSP/B [WAP00d]. The resulting protocol is called WSP/C—a shorthand for WSP/CORBA—and will be described throughout the following section.

WAPIOP introduces a session setup phase in the CORBA communication protocol, mapped onto the session management facility of WSP. In contrast to GIOP, protocol parameters remaining constant throughout a single con-
connection are negotiated only once, instead of being contained within each message in the accompanying GIOP header. The results of this strategy are twofold. On the one hand, new message types for session management are introduced. On the other hand, the header information is changed accordingly.

A new CORBA message type initiating session establishment is defined in WAPIOP, mapped onto and named after the WSP counterpart Connect. The parameters to be negotiated in the course of session establishment are the protocol version and the desired byte order. The protocol version is already part of the Connect PDU. The desired byte order is mapped onto an extended WSP capability, using an encoding in the currently unassigned space. The headers field included in the WSP Connect PDU can be left empty.

The corresponding, likewise new CORBA message type confirming session establishment again is mapped onto and named after its WSP counterpart ConnectReply. The ConnectReply PDU contains besides the confirmed capabilities a ServerSessionId uniquely identifying the session. This ServerSessionId must be used when referring to session specific state on the server-side. Just as for the Connect PDU, the headers field being part of the WSP ConnectReply are left empty.

Strictly following the WSP specification, the session establishment PDUs Connect and ConnectReply together utilize a single WTP class 2 transaction. Referring to the procedure outlined in the discussion of the Wireless Transaction Protocol on page 79, the Connect message is sent as payload of a WTP class 2 Invoke PDU. The ConnectReply in turn represents the payload of the corresponding WTP class 2 Response protocol data unit. Finally, the ConnectReply is acknowledged through a WTP Ack PDU, thereby terminating the session setup phase.

Establishing a WAPIOP connection may be initiated either by a mobile station or, under some circumstances, by a machine located inside the fixed network. Because the mobile can be assumed to be the computationally weaker entity, it should play the role of the Initiator of the new WSP session, proposing a set of capabilities. Thus, to support the latter case of a session being established by a fixed workstation with a mobile terminal, a session initiation request like that described in WAP Push Framework on page 37 should be introduced.

Because of the session setup phase, nearly all information contained in the general GIOP message header accompanying each GIOP message can be omitted in WAPIOP. Basically, this applies to the GIOP protocol version and the byte order. The magic number, namely the string “GIOP”, has been
withdrawn as well. Because the WSP connection can safely be assumed to be utilized for the WAPIOP session only, there is no real need to additionally tag every message.

The message type as well is not needed anymore. Because the Wireless Session Protocol is message-oriented anyway, the PDU type field being part of each WSP message is used for this purpose. Hence, the only mandatory header field to be preserved is the message size, encoded as a variable length unsigned integer. The remaining, minimal WAPIOP message header is shown in figure 31.

However, there is a last header field to be discussed, namely the fragmentation flag. This flag, being introduced in GIOP version 1.1, allows a single GIOP message to be sent in two or more fragments. The rationale behind message fragmentation solely is to make life easier for ORB implementors, because sometimes it is impractical or undesirable to determine the size of a GIOP message in advance. However, due to high latencies being a typical characteristic of wireless networks, fragmentation tends to be counterproductive. Thus, in the initial WAPIOP design, message fragmentation has been left out intentionally.

The invocation of an operation on a CORBA servant consists of sending a Request message and, in case of a synchronous call, a corresponding Reply. Because the original WSP protocol data units, namely Get, Post, and Reply, are inadequate for CORBA-style method invocations, because their definition is tightly bound to HTTP/1.1, two new PDUs are defined: CORBARequest and CORBAReply. The payload of those protocol data units are the WAPIOP Request and Reply messages, respectively.

**Call Semantics and Transaction Classes.** CORBA defines three different call semantics: synchronous, deferred synchronous, and oneway. Synchronous invocations are the most common case, at least in fixed networks, letting clients block until the result is received. Deferred synchronous calls can be initiated using the Dynamic Invocation Interface only. After sending the request, the client may continue to work, having to poll for the response explicitly. Finally, oneway requests provide a best-effort service only. No guarantees are given whether the request reaches the servant or not. Operations have to be declared as oneway in the IDL definition, making oneway invocations a design-time choice only. For obvious reasons, oneway operations neither may declare out or inout parameters and return values, nor may raise any exception.
Depending on the call semantics, a matching transaction class is chosen. Synchronous and deferred synchronous requests are sent in the context of a class 2 transaction. Thus, the Request message represents the payload of a WTP Invoke PDU. The Reply message in turn embodies the payload of the corresponding WTP Response protocol data unit. Finally, a WTP Ack PDU concludes the synchronous method invocation, triggering the release of server-side context information regarding this operation call.

Oneway methods can utilize either class 1 or class 0 transactions. The decision which class to use may for example be selected at runtime via an appropriate ORB policy.

Request Message. The WAPIOP Request message inherits most fields from its original GIOP counterpart. Namely, these are the request_id, response_flags, target, and service_context fields. The reserved field, being included for purposes of data alignment only, has become superfluous and thus is omitted.

The most important change with respect to the Request header concerns the operation field. In GIOP, the name of the operation being called is sent in plain text, as a string field. This rather wasteful approach is replaced in WAPIOP by using a more economic tag, encoded as a variable length unsigned integer value. The first time an operation is being called, the client generates a new operation_tag uniquely identifying this operation for the duration of the current session. The initial Request message contains the new operation_tag, together with the operation name in plain text. This allows the server to recognize the new mapping of the operation_tag onto the named operation. Subsequent Request messages calling the same operation contain the operation_tag only.

In order to differentiate between both cases, the semantics of the response_flags field is extended, without the need for increasing its size. A single bit representing a flag called operation_mapping is added, being set to one if a new mapping is introduced and zero otherwise.

Altogether, this scheme adds a single field being encoded as a varuint in the first call to each operation only. The overhead is approximately one or two bytes for interfaces of typical size. For each operation being called more than once, this additional effort will almost certainly pay off.

The Request body containing the in and inout parameters of the operation being called is left unchanged in WAPIOP.

Reply Message. Reply messages are sent in response to Request messages if and only if the response_flags field of the corresponding request has been set
Middleware Support based on CORBA

accordingly. While this is the normal case for synchronous method invocations, e.g. for oneway operations no reply may be expected.

The structure of the Reply header remains unchanged in WAPIOP. It contains the request_id of the corresponding request, together with a reply_status indicating the outcome of the operation being called. Furthermore, like in a Request message, a service_context complements the Request header.

A Reply body may accompany the reply message, depending on the reply_status. If its value is NO_EXCEPTION, the body contains a possible return value plus data for all out and inout parameters of the operation. If the reply_status is either USER_EXCEPTION or SYSTEM_EXCEPTION, the body contains an exception. Yet another possibility is LOCATION_FORWARD, instructing the client to reissue the request to an object at a different location, transparently to the calling application. In this case, the Reply body contains the IOR of the object at the new location. The Reply body, too, is unchanged in WAPIOP.

A possible improvement concerns the SystemExceptionReplyBody as defined in GIOP. This structure contains the RepositoryId—a string associated with an OMG IDL definition—of the system exception being raised. Because system exceptions are standardized, they could well be enumerated, assigning them a unique identifier. Instead of transmitting the RepositoryId as a plain string, the much shorter identifier could be used without loss of information.

CORBA clients may explicitly cancel outstanding requests through sending a CancelRequest message, containing the request_id of the request to be cancelled in the header. The sole purpose of this message is to inform the server that no reply for this request is expected anymore. The execution of the corresponding operation is not terminated, only the results are discarded.

For this type of message, a new WSP PDU named after its counterpart CancelRequest is introduced. This protocol data unit is sent in the context of a WTP class 0 transaction, because it has informational nature only.

The message types LocateRequest and LocateReply together allow clients to determine whether a specific server is capable of directly receiving requests for a target object, and if not, to what address requests shall be sent. The definition of the corresponding message headers and bodies is left unchanged in WAPIOP. However, again new WSP PDU types have to be introduced.
Just as ordinary Request and Reply messages, the LocateRequest and LocateReply messages together form a WTP class 2 transaction.

**Message Errors**

If an invalid message is received, the peer is informed via a MessageError message, containing no further data at all. For this message type, again a new WSP PDU type called MessageError has to be introduced.

Basically, such errors may happen in the course of an active WTP transaction, e.g. in case of an ill-formed Request message, or outside the context of any transaction. In the former case, the MessageError will be sent as the WTP Response PDU. In the latter case, the MessageError is sent in the context of a class 0 transaction.

**Session Shutdown**

A CloseConnection message may be sent either by the server or, since GIOP 1.2, additionally by the client, terminating the client/server-relationship for both parties. This message closely resembles the WSP Disconnect PDU, allowing to map CloseConnection messages onto Disconnect protocol data units directly. This message is sent in the context of a class 0 transaction. While the original CloseConnection message is empty, the Disconnect PDU contains the ServerSessionId of the session to terminate.

With respect to WAPIOP, terminating the session invalidates all mappings introduced in the course of the conversation. As a result of session shutdown, both the client and the server may safely destroy all session-specific state.

**Message Fragmentation**

Because message fragmentation is unsupported in WAPIOP, Fragment messages are not part of the protocol and result in a MessageError message being sent back to the client.

**TypeCode Determination**

The approach of type codes being initially omitted when transmitting values of type any demands for the definition of two new message types not present in GIOP: TypeCodeRequest and TypeCodeReply, respectively. The TypeCodeRequest message is sent by an entity in order to request the full TypeCode definition corresponding to an any_tag that has been sent in a previous message. As the original WSP does not provide any comparable protocol data units, new PDU types have to be introduced.

Due to the request/reply style, TypeCodeRequest and TypeCodeReply PDUs are sent in the context of a WTP class 2 transaction.

**TypeCodeRequest.** The TypeCodeRequest message consists of a header only, containing the ServerSessionId uniquely identifying the session on the server-side plus the any_tag of the type code being requested.
**TypeCodeReply.** A TypeCodeReply message is sent as a response to a TypeCodeRequest message. This message consists of a header and a body, respectively.

The header of a TypeCodeReply message contains a single field only, named `type_code_status`, indicating the result of the request. This status field has the type `TypeCodeStatusType`, an enumeration containing the values `UNKNOWN_TYPE_CODE` and `SUCCESS`. The TypeCodeReply message is accompanied by a body in case of `SUCCESS` only. The body then contains the TypeCode object being requested.

### 4.11 Evaluation

The WAPIOP protocol has been designed to account for low bandwidth networks. After having introduced the Wireless Data Representation and the WSP/C message types, both are evaluated in this section, comparing them with the original CORBA Common Data Representation and IIOP.

The size of the encodings of primitive data types is summarized in table 3. Octet, boolean, and char all occupy a single byte in both encoding schemes. Floating point types float, double, and long double too have the same sizes.

<table>
<thead>
<tr>
<th>Primitive Data Type</th>
<th>CDR size [bytes]</th>
<th>WDR size [bytes]</th>
</tr>
</thead>
<tbody>
<tr>
<td>octet, boolean, char</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>signed &amp; unsigned short</td>
<td>2</td>
<td>1 - 3</td>
</tr>
<tr>
<td>signed &amp; unsigned long</td>
<td>4</td>
<td>1 - 5</td>
</tr>
<tr>
<td>signed &amp; unsigned long long</td>
<td>8</td>
<td>1 - 10</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>long double</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>enum</td>
<td>4</td>
<td>1 - 5</td>
</tr>
</tbody>
</table>

**WDR versus CDR**

Regarding the signed and unsigned integer types, instead of occupying fixed space (2, 4, and 8 bytes for short, long, and long long, respectively), the overall length varies with the absolute value of the integer encoded (cf. table 4). In the worst case, however, the additional *Continue* bit utilized for controlling the compression may result in the WDR representation being larger as when using original CDR. Altogether, the WDR encoding scheme favors the more common small absolute values over rather infrequent bigger ones.
Enumerations

Enumerations are encoded as numerical values, according to the order identifiers appear in the `enum` declaration. Albeit it is theoretically possible to declare enumerations with $2^{32}$ members, as CDR allows due to representing them as unsigned long values, it is typically very uncommon to have more than 127 identifiers. In such cases, because WDR uses a variable length unsigned integer encoding, each `enum` value would be represented as a single byte only.

Strings

String values are encoded in CDR as an unsigned long specifying their length followed by the string itself together with a trailing null. In contrast to this, WDR uses a variable length unsigned integer for the length followed by the characters of the string without the trailing null. Hence, string values of up to 128 bytes in length occupy four bytes less in WDR than in CDR (cf. table 5).

Sequences

With sequence types as well, their length (the number of members) is encoded as unsigned long in CDR and as variable length unsigned integer in WDR. Again, because sequences typically are not overly large, it is comparatively likely that the WDR representation needs one or two bytes only.

Fixed-Point Decimals

The WDR encoding of fixed-point decimal (fixed) value uses a varint for the signed integer part followed by a varuint containing the fractional portion. In the original CDR, every decimal digit (as well as the sign) is encoded in a half-octet. Thus, the overall length grows linearly with the number of significant digits, while trailing zeros are always encoded. The proposed WDR encoding scheme is more space efficient, growing only logarithmically with the number of significant digits for the integer and fractional parts, respectively.

Any and TypeCodes

Values of type `any` are encoded in CDR as their `TypeCode` followed by the data itself. The `TypeCode` is replaced in WDR by a session-specific `any_tag`
being encoded as a varuint. The savings gained from this approach are pointed out based upon a simple example, presuming the following declaration of a structure:

```c
struct mystruct
{
    long l;
    string s;
}
```

When transferring a parameter of type `mystruct`, assuming its members to have the values 42 and “test” respectively, the CDR encoding consumes 93 bytes overall, as shown in table 6. Thereby, the TypeCode uses 80 bytes while the data itself occupies the remaining 13 bytes.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Data</th>
<th>Content / Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00 00 00 0F</td>
<td>TypeCode description lead-in (tk_struct)</td>
</tr>
<tr>
<td>4</td>
<td>00 00 00 48</td>
<td>length of TypeCode encapsulation</td>
</tr>
<tr>
<td>8</td>
<td>00</td>
<td>byte order of encapsulation (big endian)</td>
</tr>
<tr>
<td>9</td>
<td>00 00 00</td>
<td>alignment gap</td>
</tr>
<tr>
<td>12</td>
<td>00 00 00 11</td>
<td>RepositoryId of <code>mystruct</code> structure: length</td>
</tr>
<tr>
<td>16</td>
<td>“IDL:mystruct:1.0”</td>
<td>RepositoryId: string with trailing null</td>
</tr>
<tr>
<td>33</td>
<td>00 00 00</td>
<td>alignment gap</td>
</tr>
<tr>
<td>36</td>
<td>00 00 00 09</td>
<td>structure name: length</td>
</tr>
<tr>
<td>40</td>
<td>“mystruct”</td>
<td>structure name: string with trailing null</td>
</tr>
<tr>
<td>49</td>
<td>00 00 00</td>
<td>alignment gap</td>
</tr>
<tr>
<td>52</td>
<td>00 00 00 02</td>
<td>number of members inside structure</td>
</tr>
<tr>
<td>56</td>
<td>00 00 00 02</td>
<td>name of member #1: length</td>
</tr>
<tr>
<td>60</td>
<td>“l”</td>
<td>name of member#1: string with trailing null</td>
</tr>
<tr>
<td>62</td>
<td>00</td>
<td>alignment gap</td>
</tr>
<tr>
<td>64</td>
<td>00 00 00 03</td>
<td>TypeCode of member #1: tk_long</td>
</tr>
<tr>
<td>68</td>
<td>00 00 00 02</td>
<td>name of member #2: length</td>
</tr>
<tr>
<td>72</td>
<td>“s”</td>
<td>name of member #2: string with trailing null</td>
</tr>
<tr>
<td>74</td>
<td>00</td>
<td>alignment gap</td>
</tr>
<tr>
<td>76</td>
<td>00 00 00 12</td>
<td>TypeCode of member #2: tk_string</td>
</tr>
<tr>
<td>80</td>
<td>00 00 00 2A</td>
<td>value of member #1</td>
</tr>
<tr>
<td>84</td>
<td>00 00 00 05</td>
<td>value of member #2: length</td>
</tr>
<tr>
<td>88</td>
<td>“test”</td>
<td>value of member #2: string with trailing null</td>
</tr>
</tbody>
</table>

In contrast to this, the WDR encoding needs only 7 bytes—as opposed to 93 bytes for CDR—for the mere data transfer (cf. table 7), together with the any_tag identifying the associated TypeCode. The TypeCode in turn—which
Middleware Support based on CORBA

is transmitted only on explicit request—would occupy additional 36 bytes in a separate message (plus header information), as shown in table 8.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Data</th>
<th>Content / Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>07</td>
<td>any_tag identifying associated TypeCode</td>
</tr>
<tr>
<td>1</td>
<td>2A</td>
<td>value of member #1</td>
</tr>
<tr>
<td>2</td>
<td>04</td>
<td>value of member #2: length</td>
</tr>
<tr>
<td>3</td>
<td>&quot;test&quot;</td>
<td>value of member #2: string without trailing null</td>
</tr>
</tbody>
</table>

**Alignment**

While data type representations alone are more space efficient in WDR than in CDR, full WSP/CORBA messages may benefit from the condensed encoding scheme even more due to abandoning alignment rules. One example for this has already been shown beforehand: while the TypeCode for the structure definition uses 80 bytes in CDR, only 36 bytes are needed in WDR.

**General WAPIOP Header**

An additional saving in every WSP/C message results from the reduced general header information to be transferred (see figure 32).
Due to the header fields omitted, the relocation of the message type into the WSP header, and the variable length WDR encoding, the resulting general WAPIOP message header typically needs one or two bytes only, depending on the overall message size. In contrast to this, the original GIOP header constantly occupies twelve bytes.

The overall bandwidth savings obviously are application dependent. Thus, it is not possible to deduce one single number or percentage of reduction in terms of bytes to be transferred. However, based upon a sample application, an exemplary comparison is drawn between a WAPIOP conversation and its IIOP counterpart.

The functionality of the sample application is very similar to the WAP-based Mobile Group Schedule introduced on page 39. A (single) server manages the set of appointments as well as the participants. Clients may retrieve the names of potential participants and are allowed to propose new appointments. Additionally, it is possible to query the server for agreed upon appointments during a specified time span. The IDL definitions of the sample application are as follows:

```idl
module GroupSchedule
{
  typedef unsigned short MinutesType;

  struct TimeType
  {
    unsigned short hour, min;
  };

  struct DateType
  {
    unsigned short day, month, year;
  };

  enum TimeSpanType { Day, Week, Month };

  typedef unsigned short PersonId;

  typedef sequence<PersonId> PersonIdSeq;

  struct PersonType
  {
    PersonId id;
    string name;
  };

  typedef sequence<PersonType> PersonSeq;

  enum VisibilityType { isPrivate, isPublic };
} GroupSchedule
```
struct AppointmentType
{
    DateType date;
    TimeType time;
    MinutesType duration;
    string description;
    VisibilityType visibility;
    PersonIdSeq attendees;
};

typedef sequence<AppointmentType> AppointmentSeq;

interface AppointmentDatabase
{
    PersonSeq GetPersonDescriptors();
    void ProposeAppointment(in PersonId initiator,
                            in AppointmentType appointment);
    AppointmentSeq GetAppointments(in DateType startDate,
                                    in TimeSpanType timeSpan);
};

Example Scenario
A single, consistent interaction scenario has been defined in order to make the results comparable. Albeit the example is greatly simplified, it still serves the purpose it is intended for, namely to give a clue on the effectiveness of the different approaches investigated. In detail, the sequence of steps the client performs are as follows:

1. Connect to the server.
2. Query for the descriptions of potential participants.
3. Propose two new appointments.
4. Retrieve known appointments.
5. Disconnect from the server.

Bandwidth Consumption
The focus of the comparison is put on the mere payload, neglecting the overhead induced by lower protocol layers. With respect to IIOP communications, this means that only the GIOP messages itself are accounted for. The traffic induced by Ethernet, IP, and TCP is not considered.

Correspondingly, the WAPIOP communication costs comprise only the WSP/C-related part. All lower levels including WDP—which effectively is implemented in terms of UDP in the IP-based test environment—are ignored as well.

The full conversation is shown in table 9. The WAPIOP communication starts with a setup phase where session specific parameters are negotiated. As opposed to that, the IIOP connection does not use such a feature but includes connection specific parameters in every general message header.
Middleware Support based on CORBA

Four request/reply-style method invocations follow. In the second call to the ProposeAppointment function, the strategy of substituting plain text operation names through an integer-based operation_tag field pays off. While the first call uses 95 bytes, the second one needs only 76 bytes. The 19 bytes saved result solely from the operation name that has to be transmitted in the first invocation only.

The overall bandwidth usage in this particular case is 520 bytes for WAPIOP versus 915 bytes for IIOP, or a saving of roughly 44%. Hence, even without additional compression, the Wireless Data Representation together with the condensed WSP/C message formats may result in noticeable reduction of communication costs.

<table>
<thead>
<tr>
<th>Step</th>
<th>WSP/C</th>
<th>IIOP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Client</td>
<td>Server</td>
</tr>
<tr>
<td>1. Connect to Server</td>
<td>Connect</td>
<td>ConnectReply</td>
</tr>
<tr>
<td>2. Query Participants</td>
<td>CORBARquest</td>
<td>CORBARReply</td>
</tr>
<tr>
<td>3.1 First Proposal</td>
<td>CORBARquest</td>
<td>CORBARReply</td>
</tr>
<tr>
<td>3.2 Second Proposal</td>
<td>CORBARquest</td>
<td>CORBARReply</td>
</tr>
<tr>
<td>4. Retrieve Appointments</td>
<td>CORBARquest</td>
<td>CORBARReply</td>
</tr>
<tr>
<td>5. Disconnect from Server</td>
<td>Disconnect</td>
<td>CloseConnection</td>
</tr>
<tr>
<td>Total</td>
<td>520</td>
<td>915</td>
</tr>
</tbody>
</table>

Due to the similar functionality of the CORBA sample application and the WAP-based prototype, it is sensible to conduct another evaluation comparing WAPIOP with WAP. It shall be noted explicitly that such an attempt may lead to misinterpretations. Both approaches are radically different and hence it is not reasonable to look e.g. at differences in bandwidth consumption only. Furthermore, the numbers presented shall give a clue only instead of being generalized. However, the ensuing discussion still may point out subtle hints of universal nature.

For example, the bandwidth consumption of a request message sent when proposing a new appointment in the WAP-prototype—with the same parameters as used above—is 264 bytes. WAPIOP needed 95 respectively 76 bytes for a similar message. The significant difference has several rea-
sons. Firstly, all parameters are passed as plain text as opposed to the binary format used in WAPIOP. Moreover, information identifying the session on server-side has to be included in every request.

With respect to the reply message, the difference is even bigger. The WAP response in the example uses 447 bytes—in contrast to 4 bytes for the WAPIOP reply. One of the primary reasons is that a full WML deck containing not only the acknowledged new appointment, but additionally all elements and data necessary for the user to navigate further has to be returned. Basically, the information just entered—and thus already known—is sent back to the mobile terminal once more.

This makes a general difference between the two approaches (WAP and CORBA) evident. The CORBA application has to transmit relevant data only and may maintain state across several invocations. Moreover, rendering the data is realized by the code resident on the device. As opposed to that, WAP needs WML tags for this purpose, bloating the size of that data to be sent. Furthermore, the WAP-based solution suffers from its stateless nature. Every information to be displayed must be transferred to the client and will be lost—or has to be transmitted again—at the time of the next request.

However, there is another difference to be considered. In case of the CORBA application, its code first has to be transferred to the client device prior to the first use. This startup overhead does not apply to WAP, but probably will pay off when then application is used for a longer time, as supposedly would be the case in the group schedule sample. In the example, the—comparatively simple—client-side code needs approximately 26 KB. Fairly oversimplified, according to the message sizes reported beforehand, after about 40 newly proposed appointments the CORBA-based sample application has compensated for its additional downloading overhead.

4.12 Summary and Conclusions

This chapter examined distributed object middleware as a basis for the design, development, and deployment of M-Solutions. The Common Object Request Broker Architecture has been selected as the best suited DOM for this purpose. Its openness, vendor-independence, heterogeneity support, and extensibility together with the broad spectrum of implementations—some of them being available at no charge even including the full source code—justify this decision.
Moreover, integration of wireless access and terminal mobility is a declared objective of the OMG [OMG01c]. Because the draft standard primarily focuses on the mobility aspect—how CORBA clients as well as servers may be made mobile in a transparent way—respective considerations intentionally have been omitted. Instead, this work basically complements the standardization process, particularly with respect to the protocol related sections.

Foremost in the context of a prototype application, namely the BeamerCalendar, several general aspects of M-Solutions based on CORBA have been investigated. One of the results is the remarkable diversity of different standards which should be complementary, but in fact are contradictory instead. In addition, suitable options for the deployment of CORBA-based client-side code have been illuminated.

In order to address the special characteristics of mobile, cellular networks, the architectural design of M-Solutions should be streamlined accordingly. As opposed to distributed applications being deployed in the context of stable and reliable fixed networks, a different programming model should be employed, accounting—amongst other things—for sudden connection losses. Thus, the synchronous interaction style should be replaced by an asynchronous one as far as possible, essentially decoupling client and server. Appropriate support utilizing the CORBA Event Service has been introduced, discussing the potential for its integration with the wireless network infrastructure.

Besides those investigations related to the application layer, the properties of CORBA communication protocols have been examined. After an in-depth investigation of the WAP protocol stack, a WAP-based Environment Specific Inter-ORB Protocol called WAPIOP has been proposed. WAPIOP accounts for the special characteristics of wireless communication, especially in terms of reduced bandwidth consumption.

CORBA-based M-Solutions generally can be identified as a promising approach to application development in the context of mobile networks. As opposed to WAP, CORBA provides much more flexibility and permits the development of more sophisticated applications, because the mobile device itself can be more intelligently used. Essentially, in order to apply the Extended Client-Server Model introduced on page 27, placing functionality on the mobile station and using its resources is a prerequisite. This way, application-specific methods and strategies can be employed in terms of reducing the influence of high latencies inherent to mobile networks.

However, a drawback of CORBA-based M-Solutions is the need to download code onto the mobile device prior to being able to use the particular
Middleware Support based on CORBA

application. This may be accomplished either via a (serial) data cable from the home PC or over-the-air. While this induces a comparatively high startup overhead, however, the amount of data to be transferred when using the application can be expected to be lower than that of a WAP-based counterpart. With respect to this, it shall be remembered that each time information is to be transferred to the mobile station, full WAP pages must be sent. As opposed to that, only data that has changed needs to be forwarded to the mobile device. Hence, the initial overhead could well pay off soon, with overall communication costs being lower than those resulting from a comparable WAP solution.
Besides selecting an appropriate technology for the development of M- Solutions, the dynamic characteristics of application usage in the context of mobile networks have to be examined and suitable approaches have to be identified. In contrast to traditional desktop computing where applications are installed in a static manner, this situation changes significantly with respect to mobile users. The behavior in such settings must be considered to be highly vivid. The users’ desires and preferences must be expected to change rather frequently, and to depend not only on varying moods but additionally on the current location.

This chapter concentrates on those dynamic aspects, focusing particularly on the problems regarding service discovery and location dependency. Because emphasis is put on CORBA as distributed object middleware, which itself does not include appropriate support, the integration of the proposed approaches with CORBA will be examined further. In addition to that, scalability-related issues complement the investigations and will be reported on.

State of the Art

One trend visible with respect to distributed systems are so-called spontaneous networks [Mat00]. As opposed to the more static nature of traditional (fixed) networks, user mobility requires appropriate support for highly dynamic systems where clients as well as services have to be integrated into and removed from constantly changing environments. The special characteristics inherent to such settings pose a couple of new and different requirements on the underlying technologies. Resources participating in such scenarios for instance should be rather loosely coupled, setting up
communication channels when needed, and interacting for a limited time only. Configuration is another problem area, requiring to support automat-isms without the need for manual intervention where possible. One promi-nent technology targeted at spontaneous networks, namely Jini, is used as a basis for the investigations and will be introduced in the subsequent section.

5.1 Introduction to Jini

The Jini technology coins the term *impromptu networking* for systems of the kind described [Sun01b]. Jini provides mechanisms to federate groups of devices and components into a single, dynamic distributed system called a *community* [Sun00g]. The high degree of inherent dynamics demands for suitable strategies keeping the federation in a consistent state—or reconstituting consistency in a timely fashion, respectively—even and in particular in case of failures. Another challenge posed by such vivid systems being constantly in flux is to keep the need for manual administration as low as possible—aiming ultimately at zero-administration effort. Jini’s declared objective is to provide suitable, working solutions for those problems.

**SERVICES**

The Jini architecture is centered around *services*. This rather general term denotes entities providing some functionality for its users, for instance compute or storage services, communication channels, filters, and much more.

The functionality a service provides is reflected in its interface. Naturally, due to the origins of the Jini technology, those interfaces are defined using the Java programming language.

While the interface defines the abstract contract between the service pro-vider and its clients only, a concrete service implementation must provide a proxy conforming to the interface, again written in Java. This proxy later will be downloaded dynamically into the client’s address space, thus avoiding the need to pre-install any drivers accessing the service.

Proxies in turn may provide the service on their own, or setup communication channels with one or more external servers. The interaction between a proxy and an external server thereby may employ any protocol and does not necessarily need to be Java-based.

**LOOKUP SERVICE**

Services are made available to clients by exporting them to so-called *lookup services*. Lookup services basically are directory services, mapping interfaces onto the proxies of currently available corresponding implementations. When registering a service instance, user-definable attributes can be
attached to the respective proxy. This is supposed to allow a more fine-grained selection of services.

The Lookup service is a Jini conformant service on its own. Hence, lookup services may be enrolled with other lookup service instances, permitting to build federations of lookup services e.g. for reasons of fault tolerance. Furthermore, proxies encapsulating other kinds of naming or directory services can be registered with Jini lookup services, providing a way to setup bridges.

When intending to become a member of a Jini community, clients first have to perform a so-called *discovery* process [Sun00h]. The underlying standard discovery protocol sends a multicast to a well-known address. A lookup service receiving such multicast inquiry messages replies by returning its own proxy. A flowchart visualizing the overall process is shown in figure 33 [Sun01b].

Newly started lookup services in turn adhere to a complementary protocol, the multicast announcement. This way, the Jini community is made aware of other lookup services as well, without the need for periodical polling.

In addition to the basic multicast protocol versions, corresponding unicast variants exist. Under some circumstances, this enables the location of lookup services not reachable otherwise. For instance, performing multicasts may be prohibited, or target lookup services could be located on remote networks.

A major challenge posed by highly dynamic, spontaneous networks Jini communities represent is to maintain them in a consistent state even in the event of failures. For example, service implementations may crash, devices may leave the scope of the underlying network, and much more. The destructive effects of such faulty behavior shall be limited as far as possible.
Jini follows a rather pragmatic approach with respect to failures, leading to the self-healing nature Jini communities are declared to have [Sun01c]. The basic idea is to never establish permanent connections [Rad99], but to limit their lifetime in advance. Whenever a resource under control of Jini is utilized, a so-called lease has to be acquired. This lease grants access to the resource for an agreed upon duration only. If the client wants to continue employing the resource, the lease has to be renewed. After the lease expires, the resource is assumed to be no longer used. Jini clients as well as servers have to follow this rule in order for this strategy to have the desired effects. Namely, the worst that may happen is to unnecessarily block a resource until the current lease expires. Hence, the lease duration is a critical design decision, determining the size of this uncertainty window.

The lease concept is applied consistently throughout the original Sun Jini implementation. For example, services exported to the lookup service have to acquire a lease—after its expiry, the registered proxy will silently be discarded. However, Jini has no means to enforce lease utilization by third party software. In particular, this means that client as well as server implementors have to follow the guidelines constituting well-behaved Jini participants [Edw99]—requiring for instance to establish leases between third party Jini services and their clients. A couple of utilities and helper classes are provided for this purpose [Sun00i].

Among the environmental assumptions the Jini technology makes regarding Jini-enabled devices is to have some memory and processing power [Sun00g]. Particularly, the device must be equipped with a Java virtual machine and has to be connected to the network, permitting to communicate with the lookup service over the RMI protocol [Sun00j].

In cases where these prerequisites are not met, the device still can become part of a Jini community. It then relies on a proxy to be run on a different piece of hardware fulfilling the assumptions mentioned before. The Jini Surrogate Architecture [Sun01d] provides both the necessary facilities to implement communication gateways for interconnects, as well as a place where processing power can be apportioned to a surrogate that acts on behalf of an attached device.

**Investigations**

The applicability of Jini with respect to service discovery will be discussed in detail in the following sections. Its integration with CORBA, the distributed object middleware of choice within this work, will be examined thereafter. Scalability-related investigations round up this chapter.
5.2 Discovery and Lookup for Mobile Devices

Jini technology has intentionally been developed for being applied in a local scope only. This fact is not alone reflected in the typical examples referred to in Jini-related overview papers like finding the nearest printer [Sun99b] or interacting with a room monitoring system on a portable computer [Sun01b]. Much more important, the technical design of Jini is tailored to local area networks, be it Ethernet, wireless LAN technologies like Bluetooth, or comparable technologies. When trying to employ Jini in the context of mobile, cellular networks, different requirements have to be considered and shall influence its design decisions.

One vital example for an environmental assumption not holding anymore with respect to mobile devices is the ability to perform multicasts on the local subnet. The process of discovering lookup services basically relies on this ability.

Although a unicast variant of the discovery protocol complements the multicast request/response counterparts, its intended use is rather different. For the multicast versions to work, only a well-known multicast address has to be agreed upon. In contrast, unicast discovery requires to know the full address of a lookup service, particularly including the (network) name of the hosting machine, hence requiring administrative knowledge and lowering the flexibility of the discovery process.

Moreover, the unicast protocol is TCP-based. With respect to mobile network communications, this is not an option. When employing IP-capable bearers, like for example GPRS, in conjunction with the WAP protocol stack, an adaptation of the unicast discovery protocol in general would be possible. However, finding the proper machine hosting a local lookup service still would be an open problem. Furthermore, the advantages of multicast discovery—namely in terms of fault tolerance regarding lookup server failures—would disappear.

The rationale behind the design of the discovery protocols primarily has been to provide a simple way of finding local lookup services. Multicasts are by definition limited to local subnets, while permitting some degree of control about the visible scope through determining the maximum hop count. This notion of locality is not valid—at least not immediately—in mobile networks.

However, the network provider for the most part has a—albeit with current technologies somewhat fuzzy—knowledge of the user’s location. This information generally could be used to locate nearby lookup services. The Jini surrogate architecture could be employed in a distinct form here, with
the surrogate conceptually serving as a fixed-network proxy for the mobile device, taking on the role of the discoverer.

After having appropriate proxies of Jini lookup services at hand, clients may query for local services. Those queries are primarily based on interface types. The consequences of this approach are manifold, both advantageous and disadvantageous, most of them having a general background without specifics regarding mobile networks.

One benefit of interface-based queries is to permit subtype matching per se. Hence, a query for interface type A may retrieve interfaces of type B, provided that B inherits from A. This is generally a useful feature, as it allows for example services to evolve rather seamlessly.

However, there are drawbacks as well. First of all, the query transmitted to the lookup service has to contain the bytecode of the interface. The overhead of transmitting such binaries can be significant. Moreover, relying on the equality of serialized objects may be error-prone, further introducing class versioning problems.

Lastly, the user typically does not want to gain access to a specific interface, but rather to a particular kind of service. This is a problem inherent in interface-based queries, and has impacts especially with respect to open, wide-area systems. Namely, interfaces must be well-known—prior to service lookup—thus having to be published to potential clients somehow in advance.

Different approaches exist, lessening the degree of coupling queries with language-level interface types. One example is the CORBA Trading Object Service [OMG00b], introducing a level of indirection known as service type model. Each service type includes a symbolic name, a (base) interface type, as well as (optionally) properties further qualifying a concrete service instance. Service types can be organized in hierarchies, reflecting interface type inheritance and property type aggregation. Clients querying the trader repository have to specify service type names only, thus obviating the need to know interface types in advance and including them in queries sent.

Another approach implemented in the context of the Ninja project—a scalable, fault-tolerant, distributed, composable service-platform—is called the Secure Service Discovery Service (SDS) [GW+99]. Both service descriptions as well as queries in SDS employ the eXtensible Markup Language (XML) [W3C98a] to encode attributes like cost, performance, location, and device- or service-specific capabilities [CZH+99]. The designers of SDS emphasize the advantages gained by using XML, leveraging the flexibility and semantic-rich content of this self-describing syntax.
The semantic power of queries in SDS is comparable to those of the CORBA Trading Object Service. However, the size of XML-based plain text queries can be expected to be higher than their binary CORBA counterparts. Hence, in summary and with respect to mobile networks, among the approaches discussed, the CORBA Trading Object Service generally is the most promising approach regarding query semantics.

Querying a single Jini lookup service—or an ensemble of interworking replicas—accommodates for the notion of locality as introduced above only very roughly. For mobile users, location as well as distance play important roles when trying to discover suitable services.

Discussing a sample scenario shall clarify the point. When trying to find the nearest vegetarian restaurant—a rather prominent and widely used example of location based services—it is the user’s current location defining the center and origin of the ensuing lookup process. The distance of concrete service offers primarily shall drive the search, probably implying the order of results. Additionally, the scope shall be limited by a maximum distance, specified optionally by the requesting user.

Jini as-is does not account for this kind of location information. Although it could be included in terms of service attributes accompanying the lookup service entries, this would only solve parts of the problem. In particular, defining the scope of a search should be an integral part of the lookup service interfaces. Interpreting this information, thereby determining the way a lookup process propagates through the network, shall be left to be implemented by the lookup service infrastructure.

The scope of a single Jini lookup service is limited to local subnets, due to the nature of multicasts. In general it is possible—and proposed in relevant Jini literature like [Edw99] and [Li+00]—to setup bridges between multiple Jini communities by mutually exporting lookup services. However, this is a rudimentary approach only, being inherently unscalable [Cal98].

The CORBA Trading Object Service provides a more sophisticated approach to setup federations of trader repositories. The standard contains the concept of links, permitting to explicitly setup paths for service query propagation, thus allowing to establish trading graphs. Policies can be used both inside the trader infrastructure and by the outside user to control the propagation of queries. However, only simple hop count limitations can be applied to influence the scope of queries while there is still no notion of locality as set forth above.

The kind of federations as discussed so far are not well-suited for large-scale, wide-area systems anyway. Both are capable of representing flat...
structures only. Introducing hierarchies is a well-known, often used approach fostering scalability—with the Secure Service Discovery Service aforementioned being one example [CZH+99].

With respect to M-Solutions, hierarchically organized service repositories at lower levels may encompass smaller regions while those at higher levels could subsume the overall area covered by their descendants. Location-based services could be registered at the lowest level repository covering the full area the service is targeted at. Queries then would start at the lowest level repository available, returning the most localized service instances first. Propagating queries up in the hierarchy may retrieve additional, more broad offers. As a possible extension, forwarding queries to immediate siblings—e.g. in case of no matches being found up the hierarchy—could retrieve potential service offers in the vicinity.

The cell structure of installed mobile networks supposedly could give clues about a sensible (hierarchical) structure of a service repository infrastructure. In many cases, the cell structure is driven—besides requirements implied by environmental factors—by the user density. Higher user rates demand for smaller cells. Additionally, higher frequented areas can be supposed to provide a richer set of service offers. Lastly, those services naturally tend to be more localized.

Automated Service Announcement

Jini lookup services are the focal point of brokering, returning matching proxies of registered services to requesting clients. Interestingly, references to Jini lookup services are not only returned to clients as a result of performing the discovery process. Additionally, newly introduced lookup services announce their existence to clients in the same subnet via the multicast announcement protocol [Sun00h]. Thus, after joining a Jini community, lookup service instances are made available through a push-style protocol.

The general Jini lookup process can be either pull- or push-style oriented. Clients first have to setup a service template describing the requested interface and (optionally) desired properties. Thereafter, the lookup services can be queried explicitly for registered services matching the template, in a pull-style way. Moreover, lookup services provide methods letting clients register service templates, sending notifications in case of relevant events like newly announced services or deleted ones. This push-style notification semantics is provided for an agreed upon duration only—through applying leases.

With respect to M-Solutions, these mechanisms provide appropriate basics for dynamic service discovery. However, they are limited to a single Jini community only. In particular, push-style service announcement via notifi-
cations would require—without support from the mobile network—manual interventions when changing location and thus the current Jini community.

Regarding the Jini community just left, the lease bound to lookup service notifications eventually will expire, leading at most to obsolete and hence annoying service announcements. Additionally, currently known service offers may become outdated, due to leaving their scope. Moreover, the service template would have to be registered with the lookup services of the newly joined communities in order to receive relevant notifications.

Appropriate support from the mobile network infrastructure may be rather helpful. For instance, cell changes could trigger forwarding of service templates, together with resigning notifications from the old Jini community and setting them up in the new one. Immediately triggering service template matching in the new community, thereby pushing locally registered services to the mobile device could result in a list of up-to-date offers.

5.3 Integrating Jini and CORBA

In the context of a prototype application called RoamX, some relevant issues regarding Jini and its joint appliance with CORBA in a single application have been investigated [RS00a]. On the one hand, this project permitted to derive information about the usefulness of the Jini surrogate architecture in two ways—both regarding the integration of legacy code as well as the incorporation of non-Jini-capable devices. On the other hand, the difficulties arising when bridging between Jini and the CORBA Trading Object Service could have been explored. Yet another aspect investigated concerns the movement of users across multiple administrative domains.

The purpose of RoamX is to provide support for the user-controlled management of a distributed X desktop potentially spanning multiple administrative domains. Instead of terminating x-based applications and restarting them manually elsewhere, applications can be suspended and restored selectively on different machines while preserving their context and state. The migration of the GUI is accomplished by an extended version of xmove, a pseudoserver for X window movement, developed originally by Ethan Solomita, James Kempf and Dan Duchamp at the University of Columbia [SKD94].

The RoamX infrastructure, as depicted in figure 34, is comprised primarily of three components. The xmove proxy server intercepts X11 protocol connections and routes them to a real X server, enabling the migration of an application’s GUI. The xmove browser is the user interface component. The
browser displays the user’s distributed X desktop—the set of all currently active applications, potentially hosted on different machines—and allows the migration of X applications via drag-and-drop operations. The information describing the user’s distributed X desktop is stored on a JavaCard, a smartcard with an integrated Java virtual machine [Sun99c]. Lastly, the third component called the *xmove broker* acts as a mediator between the other two parties.

**Fig. 34.** Components of the RoamX Infrastructure and their Interaction

![Diagram of RoamX Infrastructure Components](image)

**Experiences**

The implementation of the RoamX prototype application lead to a couple of results regarding different topics. The most important experiences gained are described in the ensuing paragraphs.

**Jini Surrogate Architecture**

The Jini surrogate architecture as described in [Sun01d] has been utilized successfully two times in the implementation of RoamX. Firstly, the JavaCard acts—from a conceptual point of view—as a non-Jini-capable client. The *xmove browser* can be interpreted as its surrogate, taking on the role of the Jini client.

The second time the Jini surrogate architecture has been applied concerns the *xmove proxy server* component, providing functionality to be exported to mobile clients in form of a Jini service. The implementation effort related to pure third party delegation of the Jini-specific duties from the legacy *xmove proxy server* to its *xmove broker* surrogate has been proven to be comparatively low. The Jini utility classes [Sun00i] were of great help, lowering the overhead of this rather mechanical task considerably.

**Locating Surrogates**

An interesting aspect that has been revealed in the course of the RoamX implementation concerns the task of instantiating or respectively locating the surrogate. The Jini Surrogate Architecture document [Sun01d] silently assumes the existence of a *surrogate host* running the surrogate code. However, finding a suitable host quickly can become rather complicated, especially when considering scalability.
For instance, in the particular application scenario discussed, a single surrogate—namely, one xmove broker—may serve multiple xmove proxies concurrently. This required to setup an appropriate infrastructure besides the Jini community visible to the outside world. Each xmove broker announces itself to xmove proxy servers through a standard CORBA Trading Object Service [OMG00b]. Newly started xmove proxy servers can query the Trading Object Service, selecting their xmove broker surrogate. A dynamic property thereby is used to implement a simple load balancing scheme. When querying the trader, the registered brokers are called back, returning their current load as a function of the number of connected proxy servers, connected clients, and the load of the broker’s machine.

The RoaMx prototype application bears another distinctive feature, opening the need for a variation of the surrogate architecture to be applied. In the implementation, the intention has been to allow users to cross multiple administrative domains. RoaMx users are supposed to live in a (virtual) roaming domain. The mapping of a roaming user onto a concrete administrative domain is done via a locally administered domain mapping service (cf. figure 34). In the particular case, roaming users are identified by X.509 certificates [ITU-509], having to be mapped onto a UNIX account and a MIT-MAGIC-COOKIE-1 [Epp91] needed to gain access to the X11 server, respectively.

With respect to the Jini surrogate architecture, this made splitting responsibilities between the xmove proxy server and its xmove broker surrogate a necessity. The broker, acting as the mediator between the two administrative domains, thus partly implements the exported functionality, being complemented by the xmove proxy server.

### 5.4 Scalability in Large Jini Communities

Jini Connection Technology has been designed to provide simple mechanisms enabling devices to form impromptu systems. In a typical setting, there tend to be a rather low number of fairly steady services. However, there may be situations where those prerequisites are kind of inverted. For instance in cases where a comparatively high number of similar services are to be provided, the protocol overhead induced by active service implementations in terms of discovery and join responsibilities is not negligible. Especially when the service implementations themselves are tied to the Jini community via radio networks, scalability can become an issue.

In a project called JSalsa—an acronym for Jini Service Agglomerations in Large Scale Assemblages—scalability aspects of Jini in this regard have
been investigated [ERS00]. Basically, a setting has been modeled where a large number of computationally weak entities export services to a Jini community using wireless communication technology. In particular, those entities are products tagged with smart labels, stored in a wireless, smart warehouse. Besides developing suitable strategies and protocols, their effectiveness has been evaluated based on a simulation prototype [Ehl00]. The background of the project, as well as relevant results are reported on.

**Smart Labels**

Smart labels are highly integrated circuits embedded for example in tags, tickets, or documents [Mil99]. In contrast to traditional barcodes, radio signals are used for communication purposes, thus avoiding the need for a direct line of sight. Smart labels are equipped with an antenna, with the electromagnetic fields of base stations inducing enough power to activate them. Current devices provide persistent storage of up to some kilobytes only, being accessible—both readable and writable—over-the-air. Future smart labels furthermore will incorporate processing power as known e.g. from smart cards [FH99].

**Wireless Smart Warehousing**

In the wireless, smart warehouse, products tagged with smart labels are arranged in shelves, with a base station being used for communication purposes. Each base station thereby covers a limited area—typically several shelves—as shown in figure 35. Products, grouped by product types, shall be exported as Jini services to clients, namely portable computers containing electronic shopping lists.

The exported services shall provide functionality at two different levels. On the one hand, clients shall be able to retrieve general as well as summary information regarding the product group as a whole, like product descriptions and the overall number of items available, respectively. On the other
hand, it shall still be possible to address single product entities, e.g. in order to query for expiry dates.

The characteristics of wireless data communication substantially affect an implementation of such a smart warehouse. Particularly, Code Division Multiple Access (CDMA) has been employed in JSalsa, allowing multiple senders to transmit data concurrently on the same carrier [Stü96]. However, communication over radio signals is error-prone. In case of CDMA, the error rate mainly depends on the noise strength, the individual signal power, and the number of objects sending simultaneously. The bit error rate rises exponentially with the number of interfering signals. For instance, in the case of identically distributed power levels and a number of 31 bits to represent one single data bit, the data bit error rate reaches 1% at a number of 10 interfering signals [Rap96].

Obviously, and not only due to severe restrictions in terms of processing power, smart labels may not implement full Jini services on their own. Again, an appropriate way to integrate smart labels into a Jini community is to apply the surrogate architecture as introduced above. Base stations take over the tasks of the surrogate, acting like a gateway between the radio network with the smart labels and the lookup services.

However, even with delegation, the resource consumption of Jini services inhibits a naive implementation creating one service for each particular smart label. Fortunately, the requirements as set forth above permit to group several smart labels under supervision of a single Jini service, as long as accessing single smart labels remains possible at the interface level.

The potentially large number of smart labels together with the characteristics of wireless communications demand for suitable implementation strategies when realizing Jini services of the envisaged kind. Two basic concepts are introduced, fostering scalability in different yet supplementary ways, namely distributed self-organization, being explained below, and agglomerations.

Inspired by the basic mechanisms of Jini, particularly multicast discovery and leasing, single items form dynamic groups, called federations, with membership validation. Since smart labels do not really lease services from one another, but confirm their existence as members of federations periodically, the term ticketing is used instead. Those groups are limited in size and contain single product types only. One smart label takes on the role of managing a single federation, respectively.

Federations in turn establish hierarchically organized agglomerations, as shown in figure 36. Because the number of levels in a single hierarchy is
also limited, multiple agglomerations can exist for one product type. **Federation managers** at the root of each single hierarchy keep a complete agglomeration together and know about all items contained, possibly via multiple stages of indirection. Those federation managers are responsible for creating the Jini service representing the whole agglomeration.

Agglomerations only provide a basic concept for managing large number of items in a scalable way. What remains to be solved is a means to setup agglomerations in a dynamic manner, in addition to suitable mechanisms of keeping them together. In a vivid setting like the smart warehouse, agglomerations must be supposed to be constantly in flux. They may grow dynamically when new products are delivered, as well as being decomposed on demand, e.g. when items are removed. The latter happens even more frequently, for instance when customers put products into their shopping carts.

The requirements for an algorithm managing agglomerations—namely, distributed self-organization—are fault-tolerance, reliability, quick self-stabilization and economic resource usage in terms of memory consumption and bandwidth usage. Moreover, the characteristics of a radio network, such as broadcast abilities, bandwidth, noise-ratio and crosstalk have to be considered.

The following description of the algorithm behind distributed self-organization starts in a worst case situation—a completely new re-organization without previously saved states. Initially, all smart labels are mutually unknown. It is neither necessary nor intended that each label gets to know each other, as is the case in standard election algorithms [Mat89]. In particular, message flooding has to be avoided. An appropriate solution is that only a small, random number of smart labels initially create and announce empty federations. The announcements, too, are delayed randomly. All other items are simply listening, in a kind of discovery phase, for announce-
ments of federations. Thereafter, smart labels request a ticket from one federation manager, again chosen randomly.

The manager of a federation can accept the request for a ticket, granting it by sending an acknowledgment. If the request is rejected, e.g. when the limit of members of this level has been reached, no response is sent back. After a time-out, when no acknowledgment has been received, the requesting label itself creates and announces an empty federation. This leads to a larger number of federations in the start-up phase. However, a subsequent section describes how federations optimize themselves in terms of their size. The advantages of this method are avoiding unnecessary retries and long waits as well as a fast achievement of a stable state. All smart labels are organized in federations within a bounded time. The time-out values depend mostly on the total number of smart labels. Simulation runs have shown that a few seconds for a discovery time-out is an appropriate value.

After all items are organized in federations, they have to renew their tickets with their manager periodically. While renewing their ticket, multiple error conditions can arise. For example, items may not reach their manager due to communication failures. Furthermore, federation managers possibly may have been removed, e.g. when a customer takes the associated item out of the shelf. Items not receiving an acknowledgment from their manager can proceed in two ways. In case they hold information about other federations, they try to join one of them. Otherwise, abandoned items create and announce new, empty federations.

It is very likely that after the initial start-up phase many federations are, compared to others, rather empty. Some of them may be comprised of a single item only, the federation manager itself. These federations should merge with other ones. The strategy followed by JSalsa is that all smart labels try to move to federations that are larger than their own. But this simple idea raises some problems: Where do the labels take their information from? What happens if many items break with their federations to join a few larger ones? The latter case can lead to an undesired “ping-pong” effect. Most labels will be rejected, leading to the creation of new, empty federations, trying the same procedure again and again.

This problem has been tackled by utilizing the broadcast property of radio networks. Smart labels renewing their ticket receive the current size of their federation in the acknowledgment. Because the acknowledgment—like any other message sent by the base station—is a broadcast, smart labels are able to snoop communications, maintaining a list of known federations together with their size.
In order to prevent the ping-pong effect mentioned above, federation members switch to larger federations only with a low probability. Completely empty federations are broken up by their managers only if there are any known free federations to join. Additionally, base stations recognize the amount of network traffic and remotely ask all smart labels to adjust their time-out values. This is done by periodically inserting a particular control parameter into messages sent by the base station.

First simulations have shown that the algorithm applied in JSalsa in fact is self-stabilizing. The number of unorganized labels converges against zero. It has been shown that a number of about ten concurrent messages, without error correction, forms an upper bound for the stability of the algorithm.

A sample simulation run is shown in figure 37. The setting starts in a worst-case state with 5000 items belonging to 8 different product types being completely unorganized. After 25 seconds, only 10 items still remain unorganized. The other 4990 items have formed 165 federations, hierarchically organized into 11 agglomerations. After 45 seconds, the complete set of items has been organized into 174 federations and 14 agglomerations.

The number of unorganized items again raises above zero for one of the following reasons. Either the ticket renewal message was lost or the federation manager has been removed. For example, figure 37 clearly demonstrates the effects of removing a top-level federation manager by the simulation.
after 400 seconds. The reorganization first results in an increased number of federations and agglomerations. However, the numbers will lower again after a period of time, depending on a parameter specifying the probability of triggering the creation of new federations. In the example shown, the value of this parameter was chosen very low.

Further simulation runs were performed, varying e.g. the total number of items, the maximum size of federations, the number of levels in a single agglomeration, and the duration of the discovery phases [Ehl00]. Those simulations clearly showed the effects those parameters have on the algorithm’s effectiveness. For example, if the number of items is very high and the discovery phase is too short, the number of bit errors will rise significantly, resulting in longer stabilization times. However, if parameter values are chosen in a reasonable fashion, self-stabilization remained an invariable property of the algorithm.

5.5 Summary and Conclusions

This chapter investigated M-Solutions in spontaneous environments. The requirements posed in such highly dynamic settings largely parallel those of applications in the context of mobile networks.

The focus first has been on issues regarding service discovery. In its original design, Jini relies fully on the infrastructure defined by the underlying network, particularly on subnet boundaries. This is obviously not suitable with respect to the setting under scrutiny. Hence, it has been shown how to enhance Jini through the inclusion of a notion of locality.

Querying for service offers has to take location information into account as well. Query semantics in Jini is driven by interface hierarchies only. It has been argued that this requires a considerable effort in terms of standardization. While it may be possible to identify generic interfaces for several well-known services like printing and e-mail access, this is not suitable in general for a multifaceted set of services to be expected in the real world. Alternative approaches like XML-based service descriptions must be considered to be more appropriate.

Regarding the cell-based design of mobile networks, strategies for setting up federations of lookup services have been discussed. Moreover, techniques allowing the automated announcement of services have been introduced.
Jini in its origins is focused on the Java programming language. However, the Jini Surrogate Architecture promises support enabling the delegation of Jini-related tasks to a third party. This way, resource-limited devices as well as legacy components may be integrated into Jini communities. The applicability of this approach has been verified in the course of the investigations concerning the integration of CORBA and Jini.

Finally, Jini has been illuminated from a rather different perspective. As opposed to typical settings with a comparatively low number of fairly steady services, the situation basically has been inverted. In an example application—namely a wireless, smart warehouse—a high number of products tagged with smart labels expose Jini services. In terms of protocol overhead induced by active services, the scalability of Jini has been investigated. It has been shown that agglomeration and distributed self-organization are appropriate strategies to be applied in such settings.
In this chapter, the scope of the investigations is expanded by another dimension, namely mobility. This regards not only the technical support for migration of single clients and servers, but the inclusion of mobility at the logical level. Based on a prototype application where different types of clients are on the move while accessing a location-dependent service, a strategy exploiting the cellular network infrastructure and employing mobile agents as representatives for mobile users is investigated.

M-Solutions as referred to in this work are applications running on small devices like smartphones, communicating over wireless links of a cellular, wireless network with servers located in the (fixed) Internet. Besides location-based applications where users access a single and isolated service for some time, the cellular structure may be reflected in the infrastructure as well as the application architecture itself. A logically single service may be realized internally as an ensemble of multiple cooperating service instances, each serving a limited region only. Mobile clients in turn want to access this service, retrieving local information, depending on their current location. Thus, the server may change over time, corresponding to the user’s motion. Together with the idea of placing a proxy as a representative in the fixed network, making this proxy mobile and let it follow its user is an obvious and promising approach to be examined further.

State of the Art

Before delving into the details of such an infrastructure for M-Solutions and the architecture of single services, the basic ideas of proxies as well as mobile agent technology will be introduced briefly in the subsequent sections.
6.1 Fixed Network Proxies

Systems supporting mobile users have to account for the special characteristics imposed by their technical surroundings. One well-known approach to cope with some of the relevant problems in this area—particularly low bandwidth, high latency and unreliability—is to utilize (fixed) network proxies (cf. figure 38). Inserting a—more or less intelligent—agent between the mobile client and the target service has a couple of benefits [LSG96]:

- proxies may execute complex functions, relieving processing-limited mobile devices;
- proxies may be used to reduce the amount of communication required with the mobile device, thus reducing the amount of air interface bandwidth consumed;
- proxies may account for mobile devices that are in a disconnected state;
- proxies may shield network-based applications from the mobility of their clients; and
- proxies may shield applications from the heterogeneity of mobile devices.

Several systems practically employ such proxies. For example, the wireless Web browser developed in the context of the W4 project [Bar94] uses a proxy to apply filtering of the data stream, depending on the available bandwidth. The ParcTab ubiquitous computing environment [WSA+95] as well as the networking infrastructure of InfoPad [LBS+95] do use proxies to hide mobility from the network-based applications. The advantage of this approach is that applications written for fixed users may be re-used directly for mobile users.

One disadvantage of all such proxy-based solutions stressed in [LSG96] is that the proxy introduces another entity on the path between the client and the target server, probably resulting in a performance degradation. However, the list of benefits mentioned above typically outweighs this drawback.

The proxies discussed so far have silently been assumed to remain, once created, at a fixed position. However, this view is too restrictive with respect to services in the context of cell-based mobile networks, and mobile agent technology may be employed to make these proxies mobile.
6.2 Mobile Agents

In the early 90’s, General Magic Inc. developed an innovative system called Telescript [Whi95], allowing a piece of code to migrate from one place to another, temporarily suspended while being transferred and continuing its work when arriving at the destination. Remarkably, the inventor was granted a patent [Whi97] on a technology now known as mobile agents.

Mobile agents thus are self-contained code snippets that are being executed on top of a runtime platform. They may interact with the local system in a controlled way, through a set of well-defined interfaces provided by the agent platform. At discrete points in time when the agent decides to do so, the migration of the whole agent to a destination place is triggered. The agent’s code as well as its current state thereby will be transferred. This way, an agent may act as a representative for its originator, acting autonomously on its behalf. As opposed to traditional distributed systems where the code remains at a fixed position and the communication is remote, this situation is basically inverted with respect to mobile agent technology—the agent code moves and interaction is done locally.

The mobile agent paradigm basically provides four advantages over the classic client/server model [Mat98]:

- **Bandwidth usage**
  Function shipping as exercised by mobile agents can reduce bandwidth usage considerably when compared to data shipping as in case of remote communication. Of course, this depends on the amount of data to be transferred and the degree of filtering an agent can do. The goal obviously is to reduce to overall amount of bandwidth used—not to forget the costs induced by the migration of the agent, comprising the code as well as the (filtered) data.

- **Latency**
  Because mobile agents are located near (from the communication point of view) to the resource accessed, the influence of network latencies is largely reduced.

- **Offline operation**
  After bringing the mobile agent on its way—provided the ability of acting autonomously—the originator may go offline. Only at the time the agent has done his work or needs further instructions a new communication must take place.

- **Local resource usage**
  Typically, the resources used by a mobile agent are those of the platform it currently resides on. For instance due to security reasons, access to a
particular resource may be restricted to local use only, thus inhibiting remote usage anyway.

Irrespective of the communication costs, which are application-dependent anyway, mobile agent technology maps quite naturally onto the setting under scrutiny. The cell-based design of mobile network infrastructures together with the users’ mobility are perfect prerequisites for employing mobile agent systems.

Investigations

After having discussed several fundamental considerations, a general system architecture for M-Solutions will be introduced in the following section. Thereafter, the applicability of this system architecture will be examined in the context of a prototype application, reporting on the experiences gained.

6.3 Mobile Agent-Based System Architecture

A general system architecture for M-Solutions shall take the characteristics of the underlying setting into account in order to be as effective as possible.

**Exploiting the Cell-Based Network Structure**

Mobile networks like GSM or the emerging UMTS share one common property—they utilize a cellular structure. When developing applications, namely M-Solutions, to be deployed in such cell-based networks, their intrinsic advantageous characteristics shall be exploited.

**Location-Based Services**

One specific characteristic of cellular systems is that they allow for the provision of location-based services in a very natural manner. This issue has already been discussed in the section on Discovery and Lookup for Mobile Devices on page 113, introducing only the basic idea of focusing service access on the current cell, however. When approaching a suitable implementation strategy, this topic has to be reconsidered. Additionally, the possibility of users being—probably constantly—on the move has to be taken into account.

**Scalability Aspects**

Before suggesting a concrete system architecture, a different property of cell-based structures has to be considered—particularly their intrinsic ability to scale, provided that the size of single cells may vary. In structurally dense areas with a high user population, cells are typically much smaller than e.g. in rural environments. The number of (location based) services provided as well as the actual number of active users can be supposed to
basically parallel the cell sizes, naturally suggesting to reflect the anatomy of the cell-based mobile network in the system architecture.

Adopting a cellular structure has impacts on the implementation of services. In particular, when striving to inherit their intrinsic ability to scale, a single, logically coherent service may essentially be composed of several cooperating instances, each one being limited to a local scope only.

Summarizing the crucial points as set forth above, this section proposes a system architecture for services in the context of mobile, cellular networks based on mobile agent technology. Agent platforms are a natural match for such networks, especially taking into account the mobility of the clients. Agent systems intentionally provide a secure execution environment for foreign entities—namely mobile agents—temporarily accessing local services [Mat98].

Basically reflecting the cell-based structure of mobile networks, each cell offers its local set of services in the realm of an agent runtime platform (cf. figure 39). The runtime platform is responsible for shielding the local resources from potentially malicious agents (and vice versa), as well as protecting agents from each other.

Clients employ mobile agents as their proxies. This not only permits mobile clients to take advantage of the general benefits provided by the proxy approach as aforementioned. Because agents are mobile, they may follow their owners as they move. Hence, service usage may be focused on local offers in a natural way.

Services are made available to clients via the Jini connection technology. Thereby, services may be exported either to single or multiple cells, and several service instances may cooperate. Besides using Jini for service dis-
covery purposes, leases are applied for resource control. Client agents hence have to apply for a lease in order to use the resources offered by the local agent platform. Furthermore, leases have to be requested for each service accessed.

6.4 Prototype Implementation

The mobile agent-based system architecture introduced in the previous section has been investigated and evaluated in the context of a concrete application scenario. The selection of this scenario primarily has been driven by the aim of maximizing the demands posed on the underlying architecture.

The application that has been chosen realizes a car sharing system called CarPAcities [RS00b]. One distinctive feature of this application domain—constituting a primary reason it has been selected—is its vivid nature, with all its users being mobile even when accessing the service. Two distinct prototypes have been implemented, targeted at examining different problem areas. The first one has been a simulation [Lüt00], allowing to investigate basic properties like scalability and effectiveness of the whole approach. In the second prototype [Lic00], real agent technology as well as Jini have been employed, permitting to focus on more technical, implementation oriented topics.

An overview of CarPAcities is shown in figure 40. Three different parties may participate in the CarPAcities system—people looking for a ride, those willing to share their own car with others as well as car rentals providing cars for pooling. This leads to the basic abstractions Passenger and Car.

From an organizational point of view, it makes no difference whether the Car is owned by a private individual or belongs to a car pool. It is only the duration of its participation in the system that differs. Persons willing to share their car may make it available temporarily for dispatch on a trip by trip basis.

Itinerary is the third basic abstraction. An itinerary describes the planned path from a starting point to an endpoint together with a possibly incomplete schedule. When an itinerary is created, only the starting time has to be known, perhaps including an additional waiting time. The waiting time specifies the maximum amount of time a Passenger is willing to wait for a ride to begin. As itineraries are travelled along, the schedule will be completed.
Passengers looking for a ride register with the system. The same applies to Cars of private individuals offering rides. Cars from the pool are registered statically. Passengers and Cars are both mobile entities and therefore should need to be connected only temporarily. Hence, according to the proposed system architecture, after registering they are represented in the system through mobile agents acting autonomously on their behalf. When necessary, the agent calls back its originator.

The system is responsible for mapping Passengers onto Cars based upon their respective itinerary. CarPAcities does not strive for globally optimal planning, but for semi-optimal decisions based on local knowledge only. Due to the large number of cars and people on their way at any point in time, limiting decisions to a local scope doesn’t seem to be restrictive but rather necessary. Obviously, the more participants, the better the overall profit of the system will be.

Along the lines of the system architecture under scrutiny, the clients—namely the Passengers and the Cars, or actually their drivers—place mobile agents as their representatives inside the fixed network.

One important job of the agents is to perform a preselection of the ride offers, driven by user specific heuristics. At the time a new Car agent arrives, its itinerary is announced and forwarded to the Passenger agents.

By placing a mobile agent as a representative inside the fixed network, the general benefits gained through the proxy approach can be exploited. First of all, preselection may significantly reduce the amount of network traffic over wireless links. Obviously, the more filtering is done by the agent, the more bandwidth can be saved—an effect being application dependent. In CarPAcities, it is sensible to assume that—especially in highly frequented areas—the total number of offers is much higher than the potential matches and thus the filtering process can be assumed to be rather effective.
Circumventing Latencies

Moreover, the influence of high latencies of wireless links can be reduced through the agent approach. Offers will be processed in bulk by the agents in the first place, forwarding only some potentially interesting ones to the mobile device. In general, batching could be applied in order to minimize the effects latencies induce. However, to what extent it will be possible to limit the number of communication transactions is highly application dependent. In this particular scenario, batching is sensible only at the time a Passenger agent arrives, by sending the set of known offers in a single packet. Thereafter, e.g. when new Car agents appear, batching their offers becomes less sensible, as their arrival times and intervals are not known in advance.

Resource Economization

Of less importance, but nevertheless worth mentioning, are the resource savings on the mobile device due to lower processing power consumed. Because of the preprocessing performed by the mobile agents, a reduced amount of work is left to be done on the mobile station. Having in mind the significant difference between the standby and active (talk) times—a factor of 35 to 50 is common—every opportunity shall be taken to save valuable battery power.

Separation of Concerns

As has already been mentioned, proper separation of concerns is the key to the effectiveness of the agent-based proxy approach. To what extent the agents may relieve the mobile devices obviously is application dependent. In this particular application scenario, agents may be rather effective. Depending on the negotiation strategy chosen and the degree of autonomy of the agents, communication may be reduced down to a single transaction—accepting the (single) best offer selected by the agent and forwarded to the mobile device—for each part of an itinerary.

USER TRACKING

The clients of CarPAcities are entirely mobile entities. Cars follow their route from start- to endpoint. Passengers basically do the same, but with their itineraries split into parts. On their way, they cross multiple cells of the mobile network and thus of the mobile agent-based system infrastructure.

Mobile agents in their role as proxies shall follow their respective users. This requires some kind of interaction between the mobile agent and an entity capable of providing location information regarding the mobile device. Basically, there are two distinct approaches, namely with and without involvement of the mobile network provider.

GPS-Based Tracking

The first solution is independent of the provider, relying on location tracking through the Global Positioning System (GPS). The accuracy of the Standard Positioning Service (SPS) [DoD95] available for civilians is about 100 m. GPS receivers can be realized as integrated circuits, embeddable in small devices. The Benefon ESC! [Ben] for example includes a full GPS-based navigation system.
Smartphones equipped with a GPS receiver could monitor their own location, calling back their associated agents on demand. Provided an agent knows the dimensions of its current cell, it could instruct the mobile device to send a callback only when leaving the cell. Taking into consideration the irregular shape of real-world cells, however, this approach quickly becomes rather complicated. Especially in urban areas with small cells, the accuracy of 100 m further complicates the situation. Even when the mobile agent-based system infrastructure is designed around a more regular geographical structure, the drawbacks of the GPS approach—most notably its complexity and induced communication overhead—remain significant.

The second solution to tracking the location of mobile users is to rely on the collaboration with the network provider. As mobile devices move from cell to cell, handovers as described in more detail on page 10 are triggered. This information could be forwarded to the agent platforms and the user’s mobile agents, respectively. However, this assumes the user to agree with the propagation of his location information for reasons of privacy.

Nevertheless, one problem remains to be solved. Handover decisions are somewhat fuzzy, and may be accompanied by a ping-pong effect. Thus, the information about the handover should not be forwarded directly to the agents. Instead, the propagation shall be delayed, absorbing potential subsequent (multiple) cell changes as far as possible. One approach could be to clone the agent at the time a handover occurs, forwarding it to the new cell. The old agent, however, could remain active for a limited amount of time, until the user may be expected not to return to this cell.

Although being an application dependent aspect, it can be assumed that this approach is rather harmless. The additional resource consumption is negligible, and the information provided by the supplementary agent may still be of value for the user. However, it shall be kept in mind that this situation may result in increased requirements on the agent implementation, because now multiple agents may coexist and possibly have to interact in order to function properly.

Placing multiple cooperating agents as representatives of a single mobile user into the fixed network may be profitable anyway. In CarPA cities, for example, passengers are basically interested in ride offers in their vicinity. However, limiting the scope of the search to a single cell may be too restrictive. As opposed to that, an approach where multiple agents flood adjacent cells is more sensible, especially in urban areas with small cells. The scope in this setting should be limited by a predefined distance, requiring each cell to know and provide its dimensions.
As soon as not only a single representative but multiple ones are active on a single mission, the need for a controlled way of interaction and cooperation quickly arises. Again, a sensible solution for this problem is application dependent. While in simple situations no cooperation at all may be required—e.g. in informational services where data is propagated to clients only—this cannot be assumed to be the common case.

In the initial CarPAcities prototype, an approach where a single agent, called the master agent, has the responsibility of managing the ensemble has been applied. This master agent is a central point for information gathering. Ride offers from all other agents are—after being filtered—forwarded to the master agent (as opposed to the mobile device). The master agent thus has the possibility of further evaluating the offers, resulting in another selection process. Hence, even less information exchange with the mobile has to be conducted.

A different approach would be to use a flat, symmetric structure with agents having equal rights and responsibilities. This obviously would be in favor of failure tolerance—the single point of failure the master agent represents no longer exists. However, if each agent acts autonomously without coordination with its siblings, a higher degree of interaction with the mobile device would result. Generally, a more dynamic solution combining the advantages of both approaches introduced so far, e.g. using a master being elected on demand, would be a better choice.

Agent ensembles are not only appropriate with respect to Passengers, but additionally for Cars. Just as is the case for a single Passenger, each Car shall be visible not only in the cell it currently resides in. Instead, Cars shall be known some time in advance along their route. This gives the essential opportunity of doing pre-planning to a limited degree.

Basically, Car agents in CarPAcities are propagated to the next k cells according to their itinerary. The selection of k is a critical design factor. It not only has a direct influence on the communication effort, k also determines the amount of time the system will be able to do pre-planning.

As Cars move along their itinerary from cell to cell, each cell change triggers some actions. The cell just left may forget about the Car. Additionally, information about the Car will be propagated to the next k cells. This allows the subsequent k-1 cells to update their local state associated with a Car, e.g. by adjusting the estimated arrival time. The k-th cell in turn will learn about a new Car, possibly starting new negotiation processes for registered and waiting Passengers.
Recalling the master agent approach, which is applied in the CarPAcities prototype with respect to the Car agents, this means that the master Car agent is the only agent being always up-to-date. All other agents have a somewhat outdated view until the update is performed via propagation. In order to limit the inconsistency window to a predefined degree, relevant updates have to be propagated—besides on cell changes—in a timely fashion.

A concrete example is illustrated in figure 41. The scenario shows the itineraries of two Cars and one Passenger just starting their trips. At the zoomed cell, the Passenger may change to the second Car. Information about the parties is propagated to the next $k=3$ cells in this example.

Propagating Car agents in advance along their itinerary requires to know which cells will be visited. In the simulation prototype, this problem has been simplified considerably. A so-called NavigationService has been used for this purpose. The NavigationService accesses a map defining the infrastructure—namely cities and roads—together with the underlying cell structure. Hence, this service could be used to deterministically derive routing information.

In a real-world scenario, however, things are more complicated. One possible solution to predict the movement of Cars is to rely on the cooperation with (GPS-based) navigation systems—a kind of equipment becoming increasingly ubiquitous nowadays. The planned route could be communicated to the agents. A local instantiation of a (different kind of) NavigationService could be used in conjunction with the data from the navigation system, ultimately driving the propagation process.

Traffic jams, road works, and resulting loop ways place more requirements on a movement prediction scheme. In parts, this information is available
today in modern navigation systems. Travel and Traffic Information (TTI) is disseminated through multiple delivery methods [KM99], for example via the Radio Data System-Traffic Message Channel (RDS-TMC) and the emerging protocol known under the name TPEG [TPEG00a][TPEG00b]. The latter is a bearer independent service, making traffic information available not only via radio signals to dedicated receivers, but additionally—amongst others—over the Internet. This way, relevant information resulting in deviations or rearrangements of the planned route could be derived, both from the navigation system and from the agents.

Along the lines of the system architecture under scrutiny, the CarPAcities prototype implementation makes use of the Jini technology to implement its service infrastructure. Four services are provided: the RuntimeService, the RegistrationService, the NavigationService already introduced beforehand, and the ConnectionService, as depicted in figure 42. Each service is available in every cell.

The NavigationService, as previously noted, is responsible for itinerary planning. It provides methods to retrieve the next cell on the path to a specified destination. The ConnectionService is used to setup a communication channel between two parties, possibly residing in different cells.

The RegistrationService acts as the primary contact point for Passengers and Cars. Passenger agents register their interest in rides with this service. They remain registered with the service until they accept a ride offer or cancel their registration themselves. Cars are registered with the service either if they are available in a local car pool or if they are in transit at most k cells away from the current one. Each Car thereby is associated with an agent used for negotiation purposes. This agent knows about the Car’s itinerary, the available seats along this itinerary, and estimated arrival times.
The RuntimeService provides a secure runtime platform for the execution of the various agents within their respective lease interval. The RegistrationService collaborates with the RuntimeService during ride negotiation. New ride offers, for example, are dispatched to the registered Passenger agents.

The Jini concept of leasing is used extensively throughout the implementation. Passengers and Cars, for example, hold leases to the RegistrationService. If their leases expire, their registration is removed from the system. This way, local resources are controlled and managed through Jini leases.

As has already been introduced in the section entitled Leases on page 111, the lease concept contributes significantly to the self-healing nature of Jini communities [Wal99]. Especially in the presence of ill-behaved clients, the negative effects due to unsolicited resource consumption can be limited. However, in scenarios like ours, where mobile users are only temporarily connected and network links are unreliable, leases may be counterproductive when applied too frequently. It is not always sensible to force a mobile user to set up a connection periodically just to renew a lease. Jini itself provides utility classes to bypass this situation. A helper class that may run in its own process is delegated the task of lease renewal. The lifetimes of the client and the lease renewal process in this scenario are completely independent. Thus, if a delegating client crashes, its leases may still be renewed, eliminating the self-healing effect of leases completely.

However, third party leasing is not necessary in this application, because appropriate lease durations can be derived directly from known or measurable attributes. The cell size in combination with the Car’s average velocity gives a sensible hint on the time span the Car is expected to reside in a cell (cf. figure 43). The average velocity, in turn, can be calculated by monitoring Car movement via GPS. Thus, if no unforeseen events occur, leases do not have to be renewed at all. Traffic jams, if not already known to the system by different means, can be detected via GPS monitoring either, and the durations of renewed leases can be adjusted accordingly. Expired leases trigger the deletion of obsolete data in the cell’s services.

A sophisticated system architecture not only has to be effective during normal operation. Particularly in the event of failures—which happen quite often with respect to mobile networks due to their comparatively low reliability—a mature design continues to work, compensating erroneous behavior as much as possible.

Cell-based structures, besides being scalable, furthermore are intrinsically more resilient to failures than typical centralized—or centrally controlled—
systems. Provided that the autonomy of each single cell is ensured, outages will affect limited regions only, while the overall system is still operative.

However, while cell-based structures are in favor of robust architectures, they provide no guarantees whatsoever on their own. The design of services has to take several considerations and guidelines into account in order for the overall system architecture to meet the goal of being resilient to failures.

Agent propagation as exercised by CarPA cities is one exemplary measure taken against loss of service in case of cell outages. In normal operation, for instance a Car agent is not only registered in the cell where its driver currently resides. Rather, the Car agent is propagated along its planned itinerary \( k-1 \) cells ahead.

Assuming one of those cells goes out of service, say the \( f \)-th cell, the following will happen—besides the inevitable fact that no ride offers from cell \( f \) will arrive. At the time agent propagation will be triggered, anyway, all agents will—logically—move one cell ahead. Of course, from an implementation point of view, this movement reduces to transferring some state information only, except for the first and last cell in the chain. In the first cell, the agent will be destroyed while a new agent will be created in the now last cell. At the faulty cell \( f \), however, an error will occur, indicated by a communication failure. The consequences that result are subject to the propagation strategy.

The most basic approach is the chain style propagation as mentioned beforehand. Each agent, when being triggered, forwards state information to its successor. In the style of a chain reaction, this event will act as a trigger, thus initiating further propagation along the path. Due to the faulty cell \( f \)—provided no countermeasures are taken—cells subsequent to \( f \) will not be triggered. Therefore, their leases will eventually expire. Consequently, the area behind \( f \) will not be covered anymore, implicating that the effective propagation path length \( k \) successively declines to zero. Only after passing the faulty cell, full coverage will be reconstituted.

A more sophisticated approach retaining coverage as far as possible is the master agent initiated propagation. Here, the master agent explicitly triggers propagation on every successive agent. Thus, only the attempt to transfer state information to the faulty cell \( f \) will fail. All other agents will continue to work as expected. One drawback of this solution is the increased overhead. Not only has a—basically redundant—communication to be performed with every successive agent, this furthermore requires the master agent to have knowledge of all of its successors.
Yet another approach is rather similar, but inverts the direction of communication. Each agent thereby calls back the master early enough before its lease expires. However, this is basically in contradiction to the aim of the lease concept as applied. Those leases just have the purpose of an agent whose lease expires has become obsolete, e.g. due to changing the originally planned itinerary.

One special case—being an immediate implication of the asymmetric master agent approach—arises in case the master agent fails, respectively when it is affected by the cell outage itself. Reminiscent of the callback technique previously described, the first (alive) successive agent must detect the master agent failure, declaring itself as new master. This requires each agent at least to know all its predecessors in order to prevent failure of multiple cells, thus again increasing overhead in terms of state information necessary.

Of course, a multitude of different approaches is conceivable, especially more symmetric ones without a dedicated master. However, corresponding investigations remain to be performed. Nevertheless, the discussion as set forth above demonstrates that sensible, failure tolerant solutions pertaining to cell outages are possible but demand for elaborate feasibility studies.

Another important feature fostering failure resilience is the use of agent-based proxies. Considering the susceptibility to communication breakdown of wireless networks—probably for longer periods—agents as applied in the proposed system architecture play a central role. However, their usefulness can be boosted further—not only with respect to lease management, but with a special focus thereon—by exploiting network operator knowledge.

The provider not only knows whether a mobile station currently is reachable, turning on and off the device furthermore results in a sign-on and sign-off procedure to be performed, respectively. Propagating such state changes to the agents could trigger appropriate actions. For instance, some service utilizations—like CarPA cities—could switch agents in a mode increasing their autonomy. Thus, in times where users cross areas with no reception, the agents may continue their work, keeping the interaction with the service alive as they see fit. Without knowledge of the user’s current location and his movement, agents for example could renew their leases until being contacted again. A comparatively long timeout, however, still has to be applied in order to prevent orphaned agents.
6.5 Summary and Conclusions

In this chapter, a mobile-agent based system architecture for the development and deployment of M-Solutions has been proposed and evaluated. The investigations were primarily based upon a prototype application, a car sharing system called CarPACities.

The cellular structure of current mobile networks is adopted by the general system architecture, aimed at inheriting the intrinsic scalability of such cell-based designs. Each cell—whereas a 1-to-1 mapping of network cells onto those service cells is not necessary—employs a mobile agent platform to export local services. Services in turn are exported via the Jini technology.

Mobile agents act as fixed network representatives of mobile users, which may be either passengers or (drivers of) cars in this particular application scenario. The mobile agents thereby increase the degree of decoupling between mobile stations and services providing ride offers. Besides reducing the bandwidth consumption over the wireless link, the agents help in reducing the influence of high latencies because most communication is performed locally between the agents and the local service instance, while the mobile station may remain inactive, thus additionally saving valuable (battery) resources. It order for agents to be effective, however, a proper separation of concerns has to be applied. The higher the autonomy of the agent, the better the profit gained will be.

In order to further increase the effectiveness of mobile agents, appropriate strategies for following their users have been discussed. While relying on GPS for this purpose is possible, this opens the need for additional, costly hardware. Instead, information from the network operator could be exploited, provided the user agrees—for reasons of privacy—on forwarding location information to third parties.

A different aspect being investigated concerns agent ensembles. Under some circumstances, as is the case for instance in CarPACities, it is advantageous to place several agents in the fixed network. This permits to cover a larger area at once, instead of relying on a single agent residing in the user’s current cell only. Those agent ensembles, however, have to be managed appropriately. While one solution would be to keep the autonomy of every agent, this for example has the drawback of potentially higher communication costs with the mobile station. In the approach applied in the prototype, a single agent is elected as the master, thus allowing for a higher degree of coordination between the agents before consulting the originating user.
The agent ensembles likewise have to follow their originators. However, this requires to predict the movement of the respective user—obviously an application-specific task. In CarPAcities, this is accomplished in terms of a (general) navigation service which is used for route planning.

Service discovery has been realized—according to the discussions in the previous chapter—by means of the Jini technology. Jini thereby not only has been applied for the purpose of querying services available locally. Moreover, the Jini concept of leases has been utilized. Through leases it has been shown to be possible of providing a comparatively easy resource management scheme inducing a small amount of communication only.
After having discussed several strategies and techniques approaching middleware support for applications in the context of mobile, cellular networks, the experiences gained and lessons learned will be recapitulated and summarized within this final chapter. Subsequently, emerging standards in this field will be introduced. Finally, the influence of those upcoming technologies on the work presented will be discussed, together with other future directions.

Recapitulation

Due to the pervasiveness of the Internet and the proliferation of related standards like the World Wide Web as well as ubiquitous services as for instance e-mail, distributed system technologies in general are comparatively popular. Support for building distributed applications is available in several ways. Communication protocols, data exchange formats, generic services, and much more. In order to provide a coherent framework for the development of distributed applications, support techniques often are subsumed in so-called middleware platforms.

Most related (de-facto) standards are tailored for use in the realm of fixed networks and desktop computing. Several properties and characteristics of the underlying communication system are silently assumed, like high reliability and throughput. In addition to that, resource consumption plays a minor role only. In case those prerequisites are not met, support from current technologies—be it for example Web standards or distributed object middleware—quickly becomes inadequate. The distributed applications basically cease to work properly, at least with a reasonable end-user perceived quality of service.
Conclusions

Distinct Prerequisites

In order to provide suitable support in the context of less reliable networks suffering e.g. from lower throughput and higher latencies, the design of such middleware has to be reconsidered. Moreover, low-capable end devices pose additional requirements on sensible strategies for the development of such M-Solutions. Slow processors, a limited amount of memory, small displays, and low battery capacities have to be taken into account. Lastly, the cellular design of modern mobile networks shall be exploited. Besides their intrinsic scalability, cellular structures can simplify the integration of location-dependency considerably.

Exemplary Investigations

The problems inherent in the setting under scrutiny are manifold. The requirements posed by the environment—primarily wireless communication, small devices, and cellular networks—are significantly different when compared to those driving the design of most current off-the-shelf middleware. Moreover, additional demands not present in the realm of fixed networks arise. This thesis aims at solving some of the most important problems, but it intentionally does not—and basically may not at the current time—propose a single, coherent framework and middleware architecture for the design, development, and deployment of M-Solutions. It does, however, suggest a comparatively high level system architecture. This architecture originates from several, initially isolated investigations of core problems at hand. The examinations performed are summarized below in a condensed way, highlighting some important aspects.

WAP-based Middleware Support

The Wireless Application Protocol as mobile counterpart of the ubiquitous and supposedly omnipotent World Wide Web is one the few middleware standards explicitly tailored for use in wireless environments on top of small end devices. The general idea behind WAP obviously is to continue the impressive success of the Web in the realm of mobile networks. However, the acceptance of WAP is still limited—not without reason.

One of its drawbacks probably is the need for content providers to develop two completely independent and diverse versions of their Web pages. Although the limitations imposed by mobile stations and WML browsers require a separate representation anyway, it is currently not possible to generate both versions out of a common base document due to considerable markup language diversities and—to a minor degree—lack of tool support.

The need for a gateway on the path between origin Web server and WML client is another downside often criticized. Because the gateway must have full access to the content transferred, it is a security sensitive entity. Hence, the gateway is almost required to be located in the same administrative domain as the origin server. However, the gateway to use typically is a (static) configuration property of the mobile station, complicating the more appropriate dynamic selection.
Finally, the actually two-tiered client/server-architecture of WAP-based M-Solutions—disregarding the logically void gateway—hampers the development of more sophisticated applications. A single client communicates directly with a single Web server. Although the Web server in principle can distribute requests to another machine, for instance depending on location information, this results in a rather awkward application architecture. In particular, realizing stateful servers then becomes critical. The origin Web server is responsible for session management, requiring appropriate collaboration with all other servers involved.

A different programming paradigm is that of dedicated applications running on the mobile device, communicating with servers located in the fixed network. This way, the limitations and shortcomings of the WAP model as described beforehand disappear. Particularly, it becomes possible to split responsibilities in a static, or even better—according to the extended client/server-architecture—in a dynamic manner between (low power) mobile device and (high power) service provider. Executing parts of the functionality on the client enables the utilization of intelligent programming strategies aimed at leveraging end-user perceived quality of service. For instance, application specific techniques like pre-fetching, lazy writeback, and request batching may be applied where appropriate, reducing the influence of wireless communication characteristics like low bandwidth, high latencies, and unreliability. Especially with respect to the latter, well-designed applications may still remain useful even in case of temporarily broken communication links.

The CORBA communication protocols—as opposed to the WAP protocol stack—are an evident example for being biased towards use in high throughput networks. The gossipy nature of GIOP and its rather wasteful bandwidth utilization requires a redesign tailored for use in networks where bandwidth is scarce and costly. The WAP protocol stack provides a sophisticated basis in this respect. Explicitly interwoven protocol layers together with combined transaction and session support offering several transaction classes for unreliable as well as reliable messages with and without response provide a good starting point. The resulting definition of a CORBA-compliant Environment Specific Inter-ORB Protocol called WAPIOP represents one step in the direction of tailoring CORBA for use in resource limited settings.

Reducing bandwidth consumption on the wireless links is an important goal anyway. Splitting responsibilities may help here to a great extent, provided the separation of concerns is performed cleverly. In this regard, delegating some client-specific tasks to an entity located in the fixed network may boost bandwidth savings additionally. Such a proxy approach further decreases the degree of coupling between origin server and mobile client.
Conclusions

even more. Hence, another important task of the proxy may be to keep in contact with the origin server in case of temporary unavailability of the mobile terminal, both in case of transient communication failures as well as powered-down devices.

Proxy Agents

The higher the autonomy of the proxy, the better this strategy works. While dumb proxies are already beneficial, the ability to execute functionality inside the proxy is even more advantageous. Agent technology provides appropriate abstractions in this respect—their primary intention in fact is to act autonomously on behalf of their users. The argument of cleverly splitting responsibilities having a considerable impact on the effectiveness of this approach applies again in this context. However, there is no general guideline as the respective decisions and heuristics are application specific.

Mobile Agent-Based System Architecture

Regarding the mobility of the user together with the desire of including location-dependency as an integral part of a suitable system architecture for M-Solutions, it is reasonable to equip the aforementioned agents with the ability to move. The places visited by those mobile agents, according to the mobile agent-based system architecture proposed in this thesis, are agent platforms structurally reflecting the cell-based design of the underlying mobile network. Each platform offers locally available services to visiting agents. Mobile agent technology is explicitly designed for the execution of foreign and therefore potentially malicious code, hence including sophisticated security features protecting from fraud. Besides the intrinsic incorporation of locality, the scalability inherent to the cellular design thus is permitted to be inherited.

Service Discovery

The services offered by each platform have to be made available to potential clients by appropriate means. Basically reflecting the situation found in spontaneous networks, the local environment has to be explored when an agent arrives at a new place. The system architecture proposed employs the Jini technology for this purpose.

Resource Management

After having found a service of interest, service usage must be facilitated in a controlled way. The user behavior in the setting under scrutiny is highly dynamic and unpredictable. It cannot be assumed that applications are well-behaved and cooperative, following for instance logon and logoff rules consistently. Hence, resource management has to be rather flexible, accounting for noncooperative and faulty clients. The Jini technology includes the concept of leases for this purpose, enabling the establishment of temporally limited contracts. The proposed system architecture utilizes leases for resource management, not only regarding service usage but for the lifetime of mobile agents as well. Although the purpose of mobile agents is to shield from unreachable and powered-down end devices, suitable strategies preventing from orphaned agents have to be followed.
Services are exported by each single agent platform, thus determining the area the respective service is visible. The scope of a particular service, however, may be different than the area served by a single platform. Although of course services may be registered with more than one platform, scalability may require a logically single service to be implemented in terms of multiple cooperating instances. The proposed system architecture explicitly allows for such deployment schemes.

With respect to scalability, the proposed approach in fact is promising, as the user behavior in terms of service usage can well be expected to parallel that of phone calls. In areas with low population and hence weak infrastructure, cells tend to be larger. Structurally dense regions with a more vivid infrastructure conversely often employ smaller cells. Mobile agents in this context result in a kind of automatic load distribution in a quite natural manner. Without mobile agents, however, this effect would be much harder to achieve, legitimating their use already for this single reason.

Mobile agents moreover generally may follow the users they represent, enabling the access of location-sensitive information. In case of a compound service realization as mentioned beforehand, this may result in changing the particular service instance responsible for the user’s current location, respectively. Service implementations must take this consideration into account, as user and session specific state held inside the service implementation may have to be forwarded as well.

Furthermore, under some circumstances, a single agent as representative may be insufficient. Due to the cell-based infrastructure and the limited area served by each service instance, multiple agents may cooperate, forming coherent agent ensembles working together on a common task.

The mobile agent-based system architecture introduced so far encompasses several aspects relevant for the design, development, and deployment of M-Solutions. It is by no means complete, and a tight integration with mobile phone networks is still to be performed. The current state of the art in this field is by far not mature enough to justify such an attempt. Perspectives for future developments arising from emerging standards, however, are still worth to be investigated in the context of this thesis.

**Emerging Standards**

Mobile network technologies like GSM are primarily designed for making voice calls. Due to the increasing demand for mobile data services, the telecommunication industry aims at defining a kind of framework enabling third parties to develop and deploy sophisticated M-Solutions. Currently,
those endeavors are reflected in a couple of emerging standards being introduced in the subsequent sections.

### 7.1 Global Multimedia Mobility

Basically, the extended UMTS system architecture presented on page 19 can be seen as a concrete implementation of a more general model, the *Global Multimedia Mobility* introduced by the ETSI [ETSI99a] and shown in figure 44. The rational behind this model is to clearly separate applications and services from the underlying network infrastructure [ETSI99b]. The traditional boundaries between fixed/mobile, private/public, and business/residential are strived to be removed. End-to-end communication between application front- and back-ends shall be possible independent of the equipment they run on, the particular access network they use, and the core networking technology employed.

The model is rather general, intentionally leaving open questions about what service is used, on which devices, by whom, and who initiates the communication. Thus, traditional voice calls are a part of the model as well as peer-to-peer and client/server applications in both pull and push modes. Examples of envisaged services and applications include video conferencing, mobile commerce, infotainment and edutainment, telematics, teleworking, and different kinds of location-based services [UF2000].

![GMM—Global Multimedia Mobility](image)

The anticipated service infrastructure is primarily composed of three entities: end-user applications communicating with application servers, which in turn rely on functionalities offered by network providers. In the estab-
lishment of such an open service infrastructure, both network operators and third party developers have to be provided with appropriate standards. The Open Service Architecture (OSA), as shown in figure 45, constitutes a framework for application server and network service development, independent of individual network providers and underlying networking technologies [ETSI99g].

Network functionality is offered through so-called Service Capability Features (SCF), defining abstract interfaces only. In turn, Service Capability Servers (SCS) provide concrete implementations of those interfaces. The OSA defines a set of standardized service capability features for basic functionality, but still allows for the incorporation of proprietary services with non-standardized interfaces. These proprietary services may be accessed through the generic framework, including mechanisms for discovery and interface description. The implementation of both service capability servers and application servers shall be independent of programming languages and operating systems. Accordingly, the specification proposes to use CORBA, providing OSA interface definitions in CORBA IDL [ETSI99h].

The standardized service capability features are further subdivided in framework SCFs and non-framework SCFs [ETSI99i]. The former provide common utilities e.g. for authentication and access control as well as for registration/discovery of available SCFs (of both kinds). Non-framework SCFs include all network related services like call control and user location.

The OSA defines some mechanisms governing the interaction between service capability servers, the framework, application servers, and end-users, respectively [ETSI99g].

Before application servers can interact with the framework, a service agreement has to be established, consisting of an off-line and/or an on-line part. The off-line part may rely on physically exchanging messages, while the on-line part requires applications to (digitally) sign a service agreement.
Once a service agreement exists, application and framework are mutually authenticated. After successful authentication, application servers can use the framework’s discovery feature to obtain references to other service capability features available. Non-framework SCFs must have registered with the discovery service previously. Before applications can use SCFs discovered, authentication and authorization have to be applied again between the application server and the target service capability server.

So far, only the application server itself is proven to be allowed to access the framework and some service capability servers, respectively. However, end-user related security still has to be handled. For example, even if an application server in principle is granted access to a specific SCS, requests may be rejected either by the end-user or by the network itself. The former may happen for example due to end-user specific privacy settings, e.g. prohibiting the network to reveal the user’s location. The latter case may apply e.g. when an end-user is not entitled to use a network service referred to by the specific SCS, possibly because the user hasn’t subscribed to this particular service. It should be noted that the Open Service Architecture does not include any standardization efforts regarding the interaction between the end-user and the application server.

**Virtual Home Environment**

Besides providing standards regarding the development of services, the ETSI documents also consider deployment scenarios. Users shall be enabled to organize the services they have subscribed to in a flexible manner. The Virtual Home Environment (VHE) is a concept for the management of a Personal Service Environment (PSE), portable over network boundaries and across different front-end devices [ETSI99i]. The ultimate goal is to present the user a consistent view on personalized features, customized to the terminal type currently used, independent of the user’s location and point of access.

Each user manages his own Personal Service Environment. The PSE contains a set of user profiles, further subdivided in user interface profiles and user service profiles. The former contains terminal type specific information, customizing the look-and-feel of a service’s user interface. The latter comprises service specific settings, personalizing the service for a specific subscriber. User profiles may be activated automatically, based upon criteria like time of day, terminal type used, and current location.

**7.2 MExE**

The Open Service Architecture defines a framework for service development, leaving room for different concrete implementation strategies. One
particular proposal first submitted by the ETSI and now maintained and
developed further by the Third Generation Partnership Project (3GPP) is
called Mobile Station Application Execution Environment (MExE).

The architectural model depicted in figure 46 shows different MExE
services, which may be located in the circuit switched as well as in the packet
switched domain, being under control of the network provider or third
parties [MExE00]. Those services may be accessed using standard transport
mechanisms like the Wireless Application Protocol (WAP) [WAP98], the
Hypertext Transfer Protocol (HTTP) [W3C99b], or proprietary protocols.

MExE provides a standardized application execution environment for dif-
ferent kinds of mobile terminals, including mobile phones, PDAs, as well
as notebooks. Those equipment is split into three categories, labelled class-
marks [3GPP00a]. MExE classmark 1 encompasses devices with limited
input/output facilities and low processing power, capable of using WAP.
MExE classmark 2 comprises contemporary, sophisticated devices with
enhanced display capabilities and medium processing power and memory
capacities, providing a PersonalJava runtime environment [Sun99a].
Finally, MExE classmark 3 includes appliances capable of running a full
Java 2 Micro Edition, Connected Limited Device Configuration virtual
machine [Sun00d].

Prior to using a MExE service, the MExE mobile station (MExE MS) has to
announce its device characteristics to the MExE service environment
(MSE). Among those characteristics are the classmarks the MExE MS sup-
ports—a single device may support several classmarks—and, optionally,
additional properties like screen size, color capabilities, supported audio/
video encodings, and the version number of the Java runtime environment.
This way, the MSE can decide whether it can support this particular user,
possibly customizing the service for the specific end device and type of
access network used [3GPP00b]. For example, bandwidth consumption
shall be minimized when radio links are involved. The MExE MS may also
Conclusions

spontaneously initiate a new capability negotiation process, e.g. in case of altered conditions like when a mobile terminal is removed from its docking station.

Capability negotiation is based on the Composite Capability/Preferences Profiles (CC/PP) specification [W3C00a] as defined by the World Wide Web Consortium [3GPP99a]. The goal of the CC/PP framework is to specify how client devices express their capabilities and preferences (the user agent profile) to the server that originates content (the origin server). The origin server considers the user agent profile in order to produce and deliver content appropriate to the client device. The framework describes a standardized set of CC/PP attributes—a vocabulary—that can be used to express a user agent profile in terms of capabilities, and the user’s preferences for the use of these capabilities. This is implemented using the Extensible Markup Language (XML) [W3C98a] application Resource Description Framework (RDF) [W3C99a]. The properties as well as the actual schema are based upon the Wireless Application Protocol User Agent Profile Specification [WAP99a].

Other mandatory parts of MExE are related to the discovery, transfer, installation, configuration, execution, and termination of services. For the purpose of service discovery, [3GPP00b] simply proposes to use a browser, disregarding mechanisms of how discovery itself has to be performed.

After a service has been discovered, it has to be transferred to the MExE mobile terminal. Again, the specification only gives raw suggestions, like using HTTP, FTP [Pos+85], or Bluetooth [BT99]. However, the user has to explicitly allow the installation of new code on his device.

Newly installed services have to be configured and personalized. This includes setting up user interface and user service profiles, as described in the section introducing the Virtual Home Environment on page 152. This way, the user influences not only the appearance of the service. Particularly, access control mechanisms allow to adjust the actions services can perform, and the resources they may use.

Particularly crucial resources to be controlled are network connections, as they may potentially induce costs. Accordingly, the MExE specification mandates to give the user extensive control over the service’s communication behavior. Users shall be able to obtain information about all open connections, as well as being informed on all ongoing communications. For this purpose, the MExE runtime environment shall manage a journal of all network events, like sending SMS messages, initiating circuit and packet switched connections, and sending data over those connections. As an
optional feature, users shall be notified in case of relevant network related events.

MExE includes Quality of Service (QoS) [3GPP00c] negotiation as an optional part. QoS aware MExE executables may request a desired and a lowest acceptable QoS level from the underlying communications network on a per stream basis. Each MExE executable may have multiple streams open simultaneously. QoS parameters can be set upon connection initiation as well as during ongoing communications. The network has the option of rejecting a QoS request.

The MExE architecture includes an (optional) MExE QoS manager component, exporting a well defined QoS API to MExE executables. This manager is responsible for the QoS negotiation between a MExE executable and the underlying transport network, mapping between the QoS parameter formats of the MExE executable and those of the network. For this purpose, the MExE QoS manager relies on a network provided interface for QoS control, e.g. as defined for GPRS/UMTS in [3GPP99a] and [3GPP99b]. Moreover, the manager is responsible for monitoring the QoS delivered by the network, attempting to re-negotiate the QoS if it falls below the lowest QoS traffic class acceptable by the MExE executable.

MExE defines a generic security framework, identifying three security domains [3GPP00a]:

- **MExE Security Operator Domain**
  This domain includes MExE executables authorized by the user’s home network operator.

- **MExE Security Manufacturer Domain**
  All MExE executables authorized by the manufacturer of the MExE mobile terminal belong to this domain.

- **MExE Security Third Party Domain**
  MExE executables that have been developed by third parties and that are authorized by a trusted authority (e.g. certification authorities) are part of this domain.

There is a fourth class of MExE applications. However, this class is not considered as a separate domain:

- **MExE Security Untrusted**
  All MExE executables not belonging to any of the three aforementioned security domains are deemed as untrusted. Ultimately, this comprises all MExE executables that are neither authorized nor whose authorization cannot be verified.
Conclusions

A MExE device shall either support all three security domains or no domains at all, in which case all MExE executables are regarded as members of the MExE Security Untrusted class [3GPP00b]. Supporting the Untrusted class is mandatory.

The user has a high degree of access control over the actions a MExE executable carries out. The MExE security framework contains detailed descriptions of which actions are in principle allowed to be performed by MExE executables, depending on their security domain [3GPP00b]. In case the MExE executable directly or indirectly performs specific actions, users must be prompted to either allow or deny permissions on a per action, per session, or blanket basis.

Future Directions

The emerging standards described in the preceding sections clearly demonstrate the endeavor of the telecommunications industry to define standards enabling not only third party service providers to develop M-Solutions in the context of a coherent and secure framework. The standards, however, are mostly on a comparatively high level, leaving open a couple of important questions.

The work described throughout this thesis in a sense complements those standards, particularly the Open Service Architecture, proposing approaches for several of those open questions. An integration of the strategies followed and the specific implementation techniques suggested seems to be both possible—there are no obvious inconsistencies or contradictions—and instrumental. Albeit the OSA is characterized as a self-contained architecture appearing as a single unit, this is neither imperative nor recommendable, primarily for reasons of scalability. In this respect, a prototypical implementation along the lines of the mobile agent-based system architecture, explicitly incorporating the cellular network infrastructure seems to be sensible and should be performed.

Moreover, additional prototypes implementing the Open Service Architecture are required, employing different technologies not being examined throughout this work. This way, it becomes possible to perform a more elaborate comparative analysis.

Interestingly, the Open Service Architecture encompasses functionality permitting third parties to request different kinds of information from the network provider. For instance, the user status may be queried and location reports may be retrieved. Where appropriate, this data may be supplied in a
push manner, either periodically or triggered by relevant events like for instance location updates. This allows some of the optimizations proposed throughout this work, e.g. concerning asynchronous message delivery and user tracking, to be put into practice and investigated further.

Yet another facet originating immediately from this work is to round up the Environment Specific Inter-ORB Protocol based upon the WAP stack. To start with, the missing parts of the protocol should be supplemented. In particular, support for value types permitting to pass objects by value as method parameters may prove to be rather useful with respect to M-Solutions. Especially Java-based clients may benefit from transferring not only data but full objects instead. Because Java allows to download code dynamically, client-side applications can be augmented on-the-fly with previously unknown functionality. This way, the extended client/server model could be applied even more effectively.

Integrating the mobility aspect constituting the standardization effort regarding Wireless CORBA [OMG01c] with the mobile agent-based system architecture suggested is required as well. Throughout this work, it has silently been assumed that the client-side M-Solution keeps—more or less permanent—in contact with one or more mobile agents acting as fixed network representatives. Those agents, however, are mobile CORBA servants migrating between different agent platforms. Hence, the object references denoting such agents basically must be represented as Mobile Interoperable Object References. In terms of concrete implementation strategies, appropriate support—probably along the lines of the prototype backing the OMG’s draft specification—still has to be provided.

In addition to technology related examinations, a sophisticated simulation environment representing a mobile network infrastructure together with its subscribers could provide a framework for the investigation of non-functional aspects. Certainly, this is a demanding and challenging job due to the overall complexity. Modeling user behavior in a realistic way is another complicated task. Based upon anonymized statistical data from network providers, however, it shall be possible to derive valuable hints in this respect. Tracing single users would allow for several kinds of investigations regarding for instance the effort needed to let a mobile agent follow its user. Another particular point of interest is to define and examine suitable ways alleviating the influence of short successive handovers. Additional investigations considering the overall user behavior become feasible as well. Particularly, expected usage patterns could be derived and scalability issues being examined in depth.

Altogether, this thesis aimed at providing suitable support for the design, development, and deployment of M-Solutions—applications in the context
of mobile, cellular networks. Due to the complexity of this topic, it is neither possible nor sensible for the time being to suggest a single, coherent, sophisticated, and mature framework for the realization of M-Solutions. However, the investigations performed and the results obtained shall give at least valuable hints driving further examinations in this realm. The first real M-Solution running on a mobile phone and accessing services still has to come.
[3GPP] Third Generation Partnership Project


[3GPP00a] Third Generation Partnership Project: Mobile Execution Environment (MExE): Service Description, Stage 1. 3GPP TS 22.057, V 4.0.0 (2000)

[3GPP00b] Third Generation Partnership Project: Mobile Station Application Execution Environment (MExE): Functional Description, Stage 2. 3GPP TS 23.057, V 4.0.0 (2000)


[ARIB] Association of Radio Industries and Businesses
[Ben] Benefon Oyj, Salo, Finland
[CWTS] China Wireless Telecommunication Standard Group


[EMC] EMC World Cellular Database


[Eri] Ericsson Mobile Communications AB


[ETSI] European Standards Telecommunications Institute
[ETSI91a] European Standards Telecommunications Institute: General Description of a GSM PLMN. GSM Recommendations 01.02 (1991)

[ETSI91b] European Standards Telecommunications Institute: Bearer Services Supported by a GSM PLMN. GSM Recommendations 02.02 (1991)


[ETSI99j] European Standards Telecommunications Institute: Terminal Equipment to Mobile Station (TE-MS) multiplexer protocol. TS 101 369, V 7.1.0 (1999)

[Eur01] Eurotechnology.com: The unofficial independent imode FAQ. (2001)


[GSM] GSM Association, Avoca Court, Temple Road, Blackrock, Co. Dublin, Ireland


[Han] Handspring, Inc.


[IrDA99b] Infrared Data Association: *Specifications for Ir Mobile Communications (IrMC)*. V 1.1 (1999)


[Li+00] Li, Sing; et al.: *Professional Jini*. Wrox Press Ltd. (2000)


[MG] META Group, Inc.


[Mot] Motorola, Inc.


[Nex] Nextel Communications, Reston, Virginia


[Nok] Nokia Group, Finland


[NTT] NTT DoCoMo, Inc., Japan


[OSF-DCE] Open Software Foundation: *The Portal to the World of the Distributed Computing Environment (DCE)*.

[OW] Openwave Systems Inc.


[Palm] Palm, Inc.


[Pra00] Prasad, Maneesh: *Location Based Services*. Article by GISdevelopment.net (2000)


[Rad99] Raduchel, B.: *Java and Jini will be in Everything*. Interview by Clare Haney, IDG News Service, April 9 (1999)


[RS00a] Rothkugel, Steffen; Sturm, Peter: *Spontaneous Networking with CORBA, Jini & JavaCards in R 0 a mX, a Mobile X-Desktop*. DOA’00, 2nd International Symposium on Distributed Objects & Applications, Antwerp, Belgium, September (2000)

[RS00b] Rothkugel, Steffen; Sturm, Peter: *C r l P A c í t ê s: Distributed Car Pool Agencies in Mobile Networks*. ASA/MA 2000 Joint Symposium on Agent Systems and Applications/Mobile Agents, Zürich, Switzerland, September (2000)

[Sam] Samsung Electronics Co., Ltd.


[Sym] Symbian Ltd., UK


[T1] Standards Committee T1 Telecommunications


[Tri] Trium, Mitsubishi Electric Group, France

[TTA] Telecommunications Technology Association

[TTC] Telecommunication Technology Committee


