A WAP-Based Session Layer Supporting Distributed Applications in Nomadic Environments

Timm Reinstorf[1], Rainer Ruggaber[1], Jochen Seitz[2], and Martina Zitterbart[1]

[1] Institute of Telematics,
University of Karlsruhe, Germany,
{reinstor, ruggaber, zit}@tm.uka.de

[2] Institute of Communications Technology and Measurement,
TU Ilmenau, Germany,
jochen.seitz@tu-ilmenau.de

Abstract. Nomadic computing imposes a set of serious problems and new requirements onto middleware platforms supporting distributed applications. Among these are the characteristics of wireless links like sudden and frequent disconnection, long roundtrip times, high bit error rates and small bandwidth. But there are also new requirements like handover support and the necessity to use different networks (bearers). All these problems and requirements lead to the demand for an association between client and server that is independent of a transport connection. In this paper, we present a session layer that provides such an association for the middleware platform CORBA based on the Wireless Application Protocol (WAP) that is especially designed for mobile and wireless devices. It turns out that the session protocol in WAP called WSP is not able to fulfill our requirements, thus, it was necessary to define our own session layer. The session layer provides explicit and implicit mechanisms to suspend and resume a session, a reconnection to the session after the bearer was lost or changed and a solution to the lost reply problem. Furthermore, it contains an interface to be used by session-aware applications to control the presented mechanisms themselves on a fine-grained level. This paper presents a detailed description of the session layer, its integration into CORBA, a mapping of GIOP messages onto WTP and selected implementation details.

1 Introduction

Terminal mobility and wireless communications are two of the most important factors to be considered in frameworks for distributed applications. The success of wireless communications (e.g., GSM or UMTS) increased the interest in distributed applications implementing the paradigm of nomadic computing. However, due to the physical characteristics of the wireless communication link special mechanisms are required to handle high error rates, packet loss or even broken connections. These mechanisms are best implemented in a special layer hiding the occurring problems transparently to the distributed applications. We, therefore, propose a general purpose session layer whose services may be used whenever appropriate (compared to the session layer defined in the ISO reference model [10]).

The need for such a session layer was identified in many frameworks for distributed applications, most notably CORBA. Thus, we take this architecture as an example to show how our session layer can be implemented in a given framework. In Sect. 2.1 and 2.2, we introduce CORBA and its components that are affected by the session layer. Ongoing work on wireless access and terminal mobility inside the OMG is presented in Sec. 2.3. For the wireless link, special communication protocols have evolved, as TCP is not well suited for this environment (see Sec. 2.4). A viable solution is the protocol family called WAP (Wireless Application Protocol) which itself has realized the notion of sessions and is described in Sec. 2.5.

Section 3 starts with the identification of the session layer requirments. We found that the session idea defined in WAP is not very appropriate for distributed applications as we show in Sec. 3.2. In the remainder of Sec. 3 we present our design of the session layer.

This layer was implemented using the CORBA implementation ORBacus and the Wireless Transaction Protocol WTP of the WAP suite. First results of this implementation are presented in Sec. 4. Related work is given in Sec. 5. We conclude this paper with a summary of the achieved goals in Sec. 6 and give an outlook on work remaining to be done.

2 Basics

2.1 CORBA

CORBA (Common Object Request Broker Architecture) is a popular middleware platform, facilitating the implementation of object-oriented location-independent distributed client/server-applications [15]. CORBA supports the interoperability between clients and servers written in different programming languages, running on different operating systems and connected by different network technologies, thus, making CORBA the ideal basis for developing applications in a heterogeneous environment.

The central component in CORBA is the ORB (Object Request Broker), which is responsible for transparently forwarding requests and replies to the appropriate entity, thus, allowing a client to invoke operations on a server without knowing anything about the location of the server or its implementation.

2.2 GIOP

CORBA-ORBs use the General Inter-ORB Protocol (GIOP) to inter-operate. GIOP can be mapped onto connection-oriented protocols that meet a set of assumptions. GIOP defines the transport protocol independent properties, e.g.

message formats. ORB-Interoperability issues that are transport protocol dependent are defined in the mapping of GIOP onto the specific transport protocol.

The GIOP specification consists of the Common Data Representation (CDR) that maps IDL data types onto a low-level representation for transmission, the GIOP message formats and a set of GIOP transport assumptions.

GIOP messages are exchanged between CORBA entities to invoke requests, locate object implementations and to manage communication channels. GIOP communication is not symmetric. Therefore, to describe GIOP messages it is necessary to define client and server roles. In the GIOP context, a client is an entity that opens a connection and may send *Request*, *LocateRequest* and *CancelRequest* messages. The server accepts connections and may send *Reply*, *LocateReply* and *CloseConnection* messages. Client and server are allowed to send *MessageError* messages.

Every GIOP message contains the address of the server object (Interoperable Object Reference, IOR). The address of a server object consists of a transport dependent part that contains its network address and an opaque transport independent part that identifies the object inside the server. If a server object can be reached via different addresses, e.g. the server is connected to different networks that use a different addressing scheme, multiple addresses can be added to the IOR.

The assumptions the GIOP definition makes about the transport behavior include that the transport protocol is connection-oriented because a connection defines the scope for identifying requests. The connection has to be reliable, which means that bytes are delivered in the same order they are sent, at most once, and that a positive acknowledgment of delivery is available. The transport provides notification of disorderly connection loss. A client may multiplex connections to multiple target objects onto one transport connection.

Transport-dependent specifications like the addressing of the host and the server-object are defined in the mapping.

The Internet Inter-ORB Protocol (IIOP) is a mapping from GIOP to TCP/IP and must be supported by every ORB implementation. Mappings to other transport protocols may be defined. Altogether GIOP defines a kind of *lingua franca* for different middleware implementations.

2.3 Wireless Access and Terminal Mobility in CORBA

The Object Management Group identified the need for supporting wireless access and terminal mobility in CORBA and issued a request for information (RFI) [14]. The most elaborated reply to this RFP is based on the experience gained in the EC/ACTS project DOLMEN that implemented a prototype of CORBA extensions to support terminal mobility.

The architectural framework described in this reply [6] identifies three different domains: the home domain, which keeps track of the current location of the mobile terminal, the visited domain to which the mobile terminal is connected at a given moment using so-called access bridges, and the terminal domain that is made up by the ORB on the mobile terminal and the terminal bridge that is responsible for connecting to an access bridge.

For communication between terminal and access bridge, a special protocol, the GIOP Tunneling Protocol GTP was defined. This protocol controls the tunneling of GIOP packets between the bridges and provides mechanisms for handoff and connection recovery. Being an abstract protocol, GTP needs to be mapped onto one or more concrete protocols.

The GTP-tunnel spans the unreliable communication link between the mobile terminal and the access node. Therefore, this tunnel must be established before the first method invocation, maintained and re-established to another access node whenever the mobile terminal moves from one area to another. Hence, GTP defines operations for tunnel establishment, tunnel handoff and tunnel release. However, using a connectionless transport protocol like UDP or WDP (as defined in the WAP protocol stack, see Sec. 2.5) makes it hard to determine when a connection is lost or when a handoff should take place.

2.4 TCP-Drawbacks

It is a well-known fact since many years that TCP does not provide a good performance in wireless environments [2,5]. The main reason is the error semantic of TCP and the resulting behavior. If a packet is lost during transmission, TCP assumes a congestion within the network and slows down transmission using the slow start mechanism. While this behavior makes sense in fixed networks, it is in most cases wrong in wireless and mobile networks. Wireless connections typically have much higher error rates including disruptions compared to fixed connections. Thus, packet loss is quite often due to these transmission errors instead of congestion. Furthermore, handover procedures in mobile networks can cause additional packet loss (packets in transit during the handover might be lost). Slowing down the performance is not at all useful in these cases, as the loss does not result from congestion. Several approaches have been proposed to solve this problem, either by splitting up the connection into a wireless and a wired part [1], by doing local retransmissions for lost packets [3], or by additional mechanisms controlling TCP directly [4]. However, all these approaches represent add-ons but not really integrated solutions. Furthermore, mobility increases the complexity of such approaches [7].

Furthermore, TCP does not support mobility of endsystems very well. This is due to the fact, that an IP address does not really identify an endsystem but a point of network attachment. Sockets as the premiere API for using TCP are based on IP addresses and, thus, suffer from the mentioned problems. Systems like Mobile IP [17] support endsystem mobility by integrating additional components and, thus, preserve the reachability of the mobile host with its original address.

2.5 WAP – The Wireless Application Protocol Suite

A viable solution providing Internet services for mobile, wireless devices has been worked out by the Wireless Application Protocol Forum (WAP Forum), which was founded in June 1997 by Ericsson, Motorola, Nokia, and Unwired Planet [23]. The basic objectives of the WAP Forum are to bring different Internet content (e.g., Web pages, push services) and other data services (e.g., stock quotes) to digital cellular phones and other wireless, mobile terminals (e.g., PDAs, laptops). Furthermore, a protocol suite should enable global wireless communication across different wireless network technologies, e.g., GSM, CDPD, UMTS etc. Therefore, the Forum embraces and extends existing standards and technologies of the Internet wherever possible and creates a framework for the description of content and development of applications that scale across a wide range of wireless bearer networks and wireless device types. Hence, the idea of using WAP for CORBA is evident (compare to Sec. 2.3).

Figure 1 gives an overview of the WAP architecture, its protocols and components. The management entities handle protocol initialization, configuration and error conditions (such as loss of connectivity due to the mobile station roaming out of coverage) that are not handled by the protocol itself.

The basis for transmission of data are different bearer services. WAP does not specify bearer services, but uses existing data services and will integrate further services. Examples are message services, such as SMS (short message service), circuit-switched data, such as HSCSD (high-speed circuit switched data), or packet switched data, such as GPRS (general packet radio service) in GSM. Many other bearer services are supported, such as CDPD, IS-136, PHS. No special interface has been specified between the bearer service and the transport layer with its wireless datagram protocol (WDP), because the adaptation of this protocol is bearer-specific. The transport layer offers a bearer independent, consistent datagram-oriented service to the higher layers of the WAP architecture. Communication is done transparently over one of the available bearer services. The transport layer service access point (T-SAP) is the common interface to be used by higher layers independently of the underlying network. If a bearer already offers IP service, then UDP is used as WDP and T-SAP is similar to the socket interface.

The security layer with its wireless transport layer security protocol WTLS is based on the transport layer security (TLS) and has been optimized for use in wireless networks. WTLS can offer data integrity, privacy, authentication, and denial-of-service protection.

The WAP transaction layer with its wireless transaction protocol (WTP) offers a lightweight transaction service. WTP distinguishes between an acknowledgment generated by the WTP entity of the responder and an user-generated acknowledgement, that is explicitly invoked by the receiving application. The service primitives of WTP are `TR-Invoke` to initiate a transaction, `TR-Result` to send back the result of a transaction and `TR-Abort` to abort an existing transaction. The PDUs exchanged between two WTP entities are the `invoke PDU`,

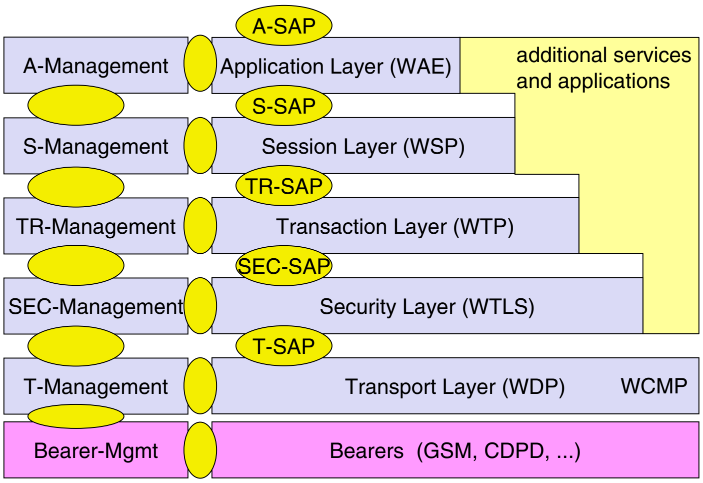

Fig. 1. WAP architecture

`ack PDU`, and `result PDU`. The basis of WTP are the three classes of transaction service:

Class 0 provides unreliable message transfer without a result message.

Class 1 provides reliable message transfer without a result message. For the sender the transaction ends with the reception of the acknowledgement, whereas the receiver keeps the transaction state for some time to be able to retransmit the acknowledgement if it receives the same invoke PDU again indicating a loss of the acknowledgement.

Class 2 provides reliable message transfer with exactly one reliable result message. It provides the classical request/response transaction, that may be directly used in client/server systems. In WTP class 2 transactions the result is acknowledged by the initiator of the transaction, resulting in at least 3 exchanged PDUs. An explicit acknowledgement of the `invoke PDU` puts the initiator on "hold on" to prevent a retransmission of the `invoke PDU` because the initiator might assume a packet loss if no result is sent back within a certain timeframe.

No WTP-class requires connection set-up or tear-down, which avoids unnecessary overhead on the wireless link. WTP does not include any flow or congestion control and thus avoids the main drawback of TCP in wireless environments.

The session layer with the wireless session protocol (WSP) currently offers a connection oriented and a connectionless service. It supports HTTP/1.1 functionality, long-lived session state, session suspend and resume, session migration and other features needed for wireless and mobile access to the web.

Finally, on top of it all, the application layer with the wireless application environment (WAE) offers a framework for the integration of different WWW and mobile telephony applications. The main issues here are scripting languages, special markup languages, interfaces to telephony applications, and many content formats adapted to the special requirements of small, handheld, wireless devices.

WAP does not force all applications to use the whole protocol architecture. Applications may use only a part of the architecture.

3 Session Layer

This section will introduce our design of a session layer using protocols from the WAP stack to support request/reply types of communication in general and as a specific application the remote method invocations in CORBA. The session layer is integrated into the mapping of GIOP onto WTP and is called Wireless Transaction Inter-ORB Protocol (WTIOP). When introducing a new layer into an already established framework like CORBA, there are two alternatives how the new layer could interact with the existing infrastructure:

session-aware: The layer could provide new functionality to applications which are aware of the characteristics of mobile communication. These are called session-aware applications in the remainder of the paper.

transparent support: The layer could be transparently integrated into the framework. This will support legacy applications which are not aware of the new session layer.

Our session layer will support both types of applications. In the following we will outline some requirements which have to be met by such a session layer.

3.1 Requirements

We identified the following requirements of the session layer in a wireless environment:

- The session entity has to provide data transport services for applications using a request/reply model for communication.
- The session entity has to be able to detect the loss of the transport connection.
- After a connection loss, the session entity needs to be able to transparently reconnect to the server.
- The session entity should support the change of the bearer used by the transport layer (vertical handover) and thus the change to another network initiated by the client.
- The session entity must provide a solution to the lost-reply problem, emerging if an invocation request was successfully sent to the server before a disconnection occurs. The computed replies have to be stored in the server and delivered to the client on its request [19].

- Session-aware applications should be able to explicitly suspend and resume the session e.g. in case of low battery power.
- A mechanism is needed to inform session-aware applications of events changing the state of the session. These events include session suspend and resume. Furthermore session-aware application should be able to influence e.g. the time interval after which the session entity should try to reconnect in the case of a connection loss.
- As a result of these requirements, the session entities have to provide an association between client and server, which is independent of the underlying transport connection.

Additionally, to support the use of the session layer for CORBA applications, these additional features need to be provided:

- Support of legacy CORBA applications, which are not aware of the session. In this case the session layer has to be inserted transparently into the CORBA environment (ORB).
- Ensure the "at-most-once" semantic of CORBA requests. A request, which is received by the session layer and is handed to the CORBA server, must be ignored if received a second time (e.g. over a different bearer after a disconnection period).

3.2 Integration

In the following we discuss the capabilities of the WAP layers and their use in a session layer fulfilling the requirements defined in the previous section:

WDP: On the transport layer, WAP defines the wireless datagram protocol that offers a bearer independent, consistent datagram-oriented transport service. When using IP based bearers WDP is realized by UDP. However, WDP is a connectionless and unreliable protocol and does not comply with the requirements of GIOP.

WTP: Therefore, there is the need for a service providing reliability to improve WDP. WAP incorporates the wireless transaction protocol WTP for reliable requests and asynchronous transactions. WTP is able to provide the necessary guarantees required by GIOP. Based on the different transaction classes, a fine grained selection of the required guarantees for a special request is possible. Furthermore, WTP is transaction oriented and thus simplifies the adaptation of CORBA requests to WTP invocations compared to stream-oriented transport protocols like TCP.

WSP: WSP is not well suited for our purposes of a generic session layer as it does not fulfill some of the basic requirements presented in Sec. 3.1. WAP is defined to support web-browsing on mobile devices and provides optimizations for HTTP. The core of the WSP design is a binary form of HTTP. It defines binary representations for information in the http header. As we do not use HTTP for sending request and reply messages these optimizations are useless for our purposes.

WSP provides session suspend and resume mechanisms, that provide disconnection transparency and enable the change of the bearer during session suspend. However, WSP does not support a reply polling on all requests that have been successfully sent to the server but the reply could not be returned to the client due to network disconnection. In this case it is necessary to resend the request, because replies are discarded after the disconnection is detected in the server and all active WTP transactions belonging to this session are aborted.

Hence, WDP and WTP of the WAP architecture provide a reliable transport service substituting TCP in the wireless domain and will be used in our session layer. WSP on the other hand provides special mechanisms supporting web-browsing on mobile devices but does not provide a really added value for generic distributed applications and will, therefore, not be used in our approach. Thus, we have to define our own session layer that fulfills the requirements pointed out in Sec. 3.1. Throughout this section we provide a detailed description of our session layer.

3.3 Architecture

Using WAP in a session layer raises the question how clients taking advantage of the session layer have to be integrated into an existing environment. Basically, there are two ways how this can be realised:

with gateway: Using a gateway corresponds to the WAP programming model. In this model, clients are not directly connected to a server but indirect via a so-called WAP gateway which is located at the crossover between the wireless and wired network. For use in our session layer this gateway has to fulfill some special functions. Apart from an implementation of the server functionality of WAP it has to support the forwarding of CORBA requests (Fig. 2). Thus, general WAP gateways can not be used in this context. A generic proxy platform for CORBA like π^2 [20] is able to fulfill these requirements. π^2 can be integrated into existing applications and provides a platform for value added services. The gateway solution enables the access of standard CORBA servers using IIOP, as the gateway is able to transform object references in a suitable way. On the other hand, if the system consists of more than one gateway it is necessary to provide a handover mechanism, which further complicates the development and deployment of such an architecture. Furthermore, gateway architectures add an additional point of failure and may lead to performance loss.

without gateway: In this architecture the client sends its request directly to the server without using a gateway. It is necessary that the server implements the WAP interfaces to accept requests sent via WTP. This end-to-end solution of supporting applications in nomadic environments is much simpler as no additional complexity (gateway) or handover mechanisms have to be realized. Using WAP this way creates the problem that WAP does

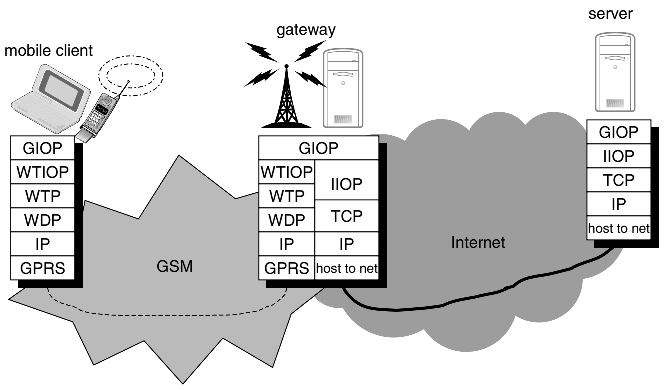

Fig. 2. Using a gateway to connect to a CORBA server via IIOP

not support any flow- or congestion control mechanisms. But because in our scenario mobile devices are connected via links with a small bandwidth, this is not a serious problem for existing networks. Routers in the wired network can handle much higher data rates and therefore can not become congested by packets received from the wireless link.

The session layer that is presented in this paper can be integrated into an architecture with or without a gateway. Using one or the other alternative is a trade-of as none is superior in all characteristics.

3.4 Concept

The purpose of the session layer is to provide a client-server context in which requests and corresponding replies can be transmitted. Multiple transactions can be invoked simultaneously in a single session. The important fact regarding the common problem of disconnections in wireless communication is the session independence of the underlying transport connection. The session layer may be suspended during periods of disconnection and resumed after regaining network connectivity which may use a different bearer. Requests that are already accepted by the session entity but have not been sent to the server and computed replies on the server that have not been sent to the client will not be discarded while a session is in a suspended state. In the following we will present how establishing, suspending and resuming a session works.

A session is identified by a session id, that is assigned by the server in the first acknowledge sent from the server to the client.

Session establishment. Depending whether the session layer is used as an end-to-end connection or via a gateway, there will be one session on the client

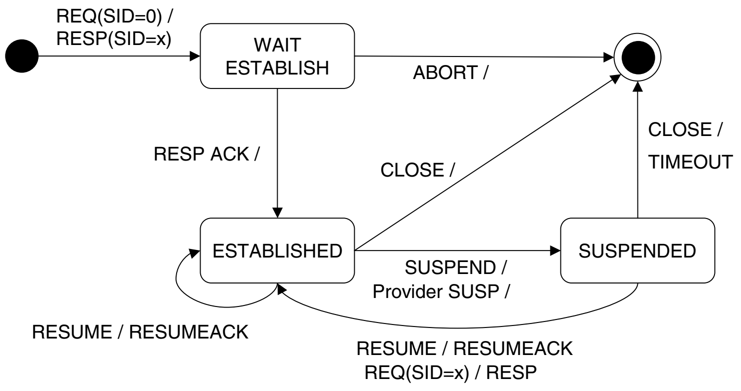

Fig. 3. Session state chart at server side

associated with every server the client uses or otherwise only one session between the client and the gateway. In the second case all GIOP connections from the client are tunneled over the single session between client and gateway. The session is always initiated by the client. To avoid the overhead of connection establishment (like three-way-handshake in TCP) the session is established implicitly during the first request from the client to a server. The server will create a new session on receipt of such a request and set the state of this session to `WAIT_ESTABLISH` (see Fig. 3). Together with the reply the server transmits a newly assigned session id back to the client, which has to be used in further transactions to identify the session. The client will set the state of the newly created session to `ESTABLISHED` after receiving the reply to the first request (including the session id). If the WTP acknowledgement to this reply is lost, the server will destroy the session, but the client will assume, that the session is still open. The client will not find out that the session is invalid before it sends a new request to the server which is answered with an WTP `TR-Abort(User)` by the server.

The tuple (session id, server name) identifies the session on client side. At the server the session id alone is unique. In the case of a transport disconnection during the first WTP transaction, the session establishment fails and has to be retried by the client (after regaining network connectivity).

Suspending a session. Figure 4 shows a state chart defining the session states at client side. At the client there are two possible events which cause a session to change into the `SUSPENDED` state. A session may be suspended either by the application (explicit suspend) or by the underlying transport layer (implicit suspend):

explicit suspend: A session-aware application may suspend a session explicitly by calling the `suspend` operation of the session. The session entity will send a

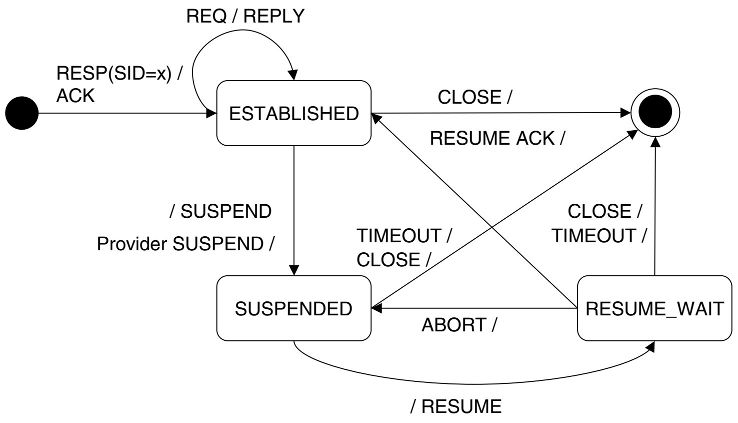

Fig. 4. Session state chart at client side

Suspend message to the server and change into SUSPENDED state. The session entity will continue accepting incoming replies but instead of handing them to the application (CORBA ORB), the reply will be stored internally. On the other hand requests from the ORB will be rejected during SUSPENDED mode. The session will remain suspended until explicitly resumed by the application.

implicit suspend: A session may be implicitly suspended in the case of detection of longer network disconnections (possibilities to detect disconnections are discussed further down). In this case the session entity will continue accepting requests from the ORB but store them internally instead of sending them. All synchronous method invocations by the ORB will remain blocked. Session-aware applications are notified by the session entity of the suspend event. The session-aware application may as a response to such a notification ask the session entity to stop accepting requests and, therefore, unblock any synchronous calls (returning an error). The client will try to resume the session by itself transparently to the application.

On the server side the session is suspended either by receiving a Suspend message from the client or also by the detection of network disconnection of the client. The server caches available but unsent or unacknowledged replies while the session is in SUSPENDED state (see Fig. 5). Once the session is reestablished by the client, the client can poll for missing replies. Thus, the retransmission of already sent requests is not necessary. The server may close suspended sessions after a specific amount of time to avoid keeping open "zombie" sessions which will not be resumed e.g. because the client has crashed. This timeout should be rather long to allow the session to survive long disconnections of the client.

Detection of network disconnection. Generally speaking, a disconnection is a state in which no WDP datagrams can be exchanged between the client and

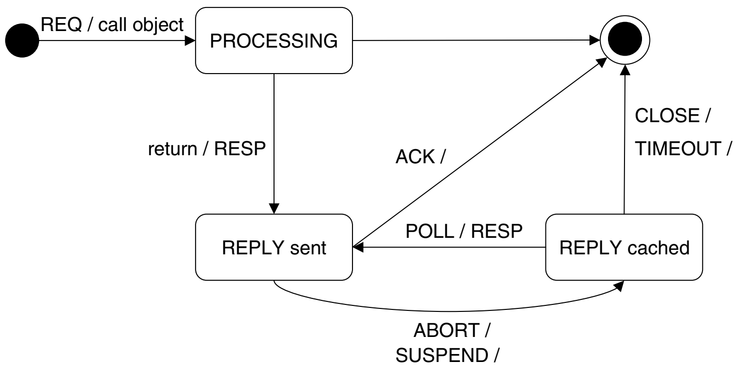

Fig. 5. Server side state chart of a request

server. We assume that these disconnections are caused by the loss of network connectivity of the client (e.g., because of loss of radio coverage).

The session entity on the server side may detect the unreachability of the client when trying to send results to the client via WTP transactions (class 1 or 2). The WTP entity will not get acknowledgements for these transactions and, therefore, abort the transaction and signal the Abort to the session layer. The session will change into `SUSPENDED` mode. If there are no outstanding results to be sent by the server, the disconnection will not be detected. Because of the passive role of the server this is not a problem. It is the responsibility of the client to resume a session, and the server will simply assume that there was a disconnection when it receives a Resume message.

On the client side the detection of network disconnection is critical. Unfortunately there are situations in which the session entity can not detect the disconnection directly by events from the WTP layer. That is the case when the WTP entity has sent a request (as an WTP invoke, see Sec. 3.5), received a "hold on" acknowledgement from the WTP server and then waits for the result. If during that period the network is disconnected, and no more requests are sent by the session entity, the WTP entity will not abort the transaction, because the disconnection is not detected. After a "hold on" message the client waits for an infinite time to receive a result. If on the other hand the network is disconnected before the WTP entity receives the acknowledgement, it will (after some retransmissions and timeouts, compare Fig. 6) abort the transaction. The session will be suspended by this Abort.

There are two ways to solve these problems:

1. A component outside the session layer may signal the network disconnection to the session entity. This component may be the operating system or a management entity of the WAP protocol stack (see Sec. 2.5).
2. A "hold on" message on the session layer may be introduced, which is sent from the server to the client in periods where no other messages (results or

acknowledgements) are sent. The time interval after which such messages are sent could be increased each time after sending a "hold on" message to reduce transmission costs in networks, where the amount of transferred data is charged (like in GPRS networks in Germany). The client would expect a message (result, acknowledgement, "hold on") after these time intervals. The missing of a message would cause the client to presume a disconnection and to suspend the session implicitly.

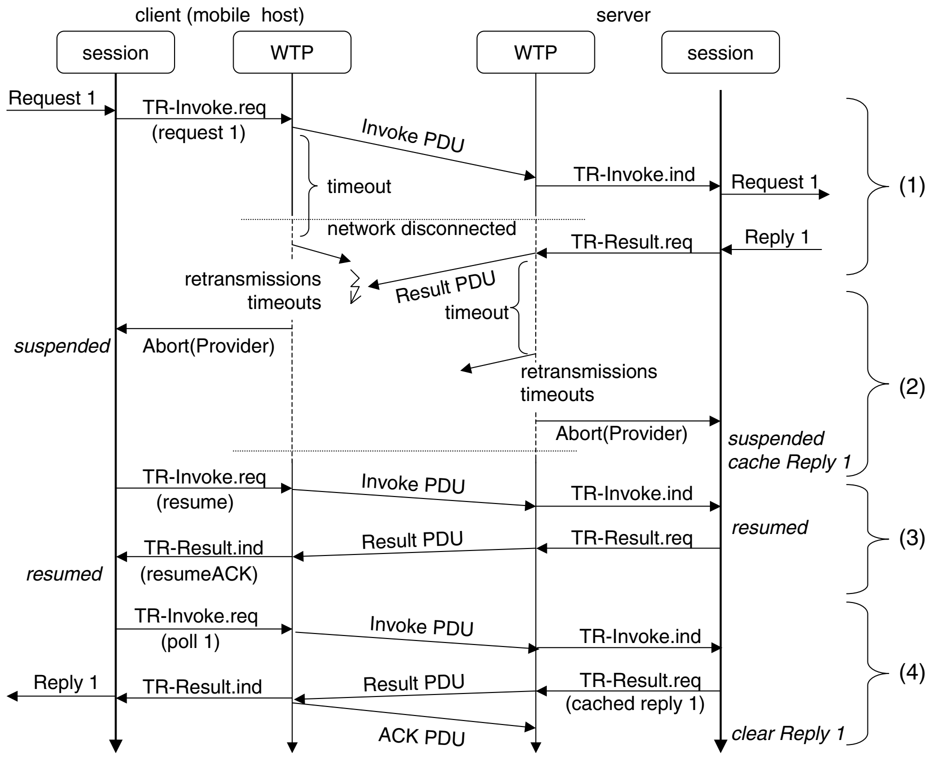

Fig. 6. Session suspend and resume

Resuming a session. The client tries to resume the session by building a new network connection and sending a Resume message to the server. Because the session is independent of a specific transport connection, the session may be resumed over a different bearer. The question which bearers are available and which have network connectivity is outside the scope of the session layer. Again the operating system or WAP management entities may provide such information or an approach as presented in [8] may be used to guard the availability of network devices as well as the network connectivity of these devices. The server acknowledges the resuming of the session by returning a ResumeACK message to the client. Once the session state on the client is changed to `ESTABLISHED`, stored requests are sent and missing replies are polled from the server.

Figure 6 shows an example sequence of exchanged messages for suspending and resuming a session. At the beginning the session is already established (by request number 0, not shown in the diagram) and the next request shall be sent. The request is transmitted in a WTP `invoke PDU` to the server and the session layer hands the request to the CORBA ORB, which returns the reply (1). The client was disconnected from the network right after sending the WTP `invoke PDU`. The client tries to retransmit the WTP message, because it is not acknowledged by either a "hold on" acknowledgement or implicitly by a WTP `result PDU`. After several retransmissions the WTP entity aborts the WTP transaction by which the session at client side is implicitly suspended (2). The same occurs on the server which tries to retransmit the WTP `result PDU`, because it is not acknowledged by the client. The WTP entity also aborts the transaction, the session at server side is implicitly suspended, too. The server caches the reply. After regaining network connectivity, the client session entity sends a Resume message to the server, which responds with a ResumeACK message and changes back into the `ESTABLISHED` state (3). On receipt of the ResumeACK message, the client starts to poll for every missing reply. In this case, Reply 1 is missing, therefore, the client sends a PollReply message for Reply 1 to the server. The server responds with a CachedReply message containing the requested Reply 1. After the CachedReply message is acknowledged by the client, the server destroys the cached reply (4).

3.5 Mapping onto WTP

As stated above, we use WTP as the reliable transport protocol which is used by the session layer to send and receive requests and replies, respectively. WTP directly supports the request/reply communication model and is, therefore, more suited than byte-streaming protocols like TCP. The session layer maps the GIOP 1.1 messages directly onto WTP transactions as shown in Table 1.

Table 1. Mapping of GIOP messages onto WTP

GIOP msg	WTP primitive	trans.class	Initiator
Request	TR-Invoke	2	client
Reply	TR-Result	2	server
LocateRequest	TR-Invoke	2	client
LocateReply	TR-Result	2	server
CancelRequest	TR-Invoke	1	client
MessageError	TR-Invoke	1	client
MessageError	TR-Result	2	server
CloseConnection	TR-Invoke	1	server
Fragment	TR-Invoke	1	client/server

Whether transaction class 1 or 2 of WTP is used to send a GIOP *Request* (or *LocateRequest*) depends on the three different types of CORBA calls:

oneway: If the request is a oneway request (no response expected), always transaction class 1 is used (transaction class 0 could be used as well, because the CORBA standard does not demand the reliable transmission of oneway requests).
synchronous: If a response is expected, transaction class 2 is used for synchronous calls.
deferred: IIOP implementations usually use a synchronous invocation in the ORB to transmit these calls and, thus, transaction class 2 is used for deferred synchronous calls.

Besides the GIOP messages the session layer uses a few control messages to suspend, resume and close a session.

Table 2. Mapping of session control messages onto WTP

session msg	WTP primitive	trans.class	Initiator
Suspend	TR-Invoke	0	client
Resume	TR-Invoke	2	client
ResumeACK	TR-Result	2	server
CloseSession	TR-Invoke	1	client/server
PollReply	TR-Invoke	2	client
CachedReply	TR-Result	2	server

In the following we will discuss facts concerning the length in bytes of a GIOP message sent over WTP. The WTP standard defines an optional feature for segmentation and reassembly (SAR) of large messages (WTP invokes and results) which exceeds the MTU (maximum transfer unit: the maximum number of bytes, that can be sent in one packet) of the underlying bearer. The reason is, that many of the bearers used by WTP (like IP or GSM SMS) have a "lost one lost all" approach concerning SAR [9]. If the WTP implementation supports SAR, selective retransmission of lost segments is used to minimize the number of resent bytes. Because WTP supports no more than 256 segments in one invoke (or result) message, the size of an invoke sent by the session layer should not exceed $256 \times MTU$. Either the session layer can ensure this by introducing SAR at session layer or in our case with sending GIOP messages, the ORB can be instructed to divide large requests into fragments and send them as GIOP fragments.

4 Implementation Details

The implementation of the session layer is done in Java using ORBacus and Jannel. Both implementations had to be modified to work together. Where ORBacus has to be modified to support a transaction oriented protocol like WTP, there was no publicly available Java implementation of the client side protocols of WAP. Most of this functionality had to be implemented by ourselves.

4.1 CORBA-Details

We use ORBacus by OOC as a CORBA implementation [16]. It is available with source code and is published under the ORBacus Royalty-Free Public License free of charge for non-commercial use. It includes the Open Communication Interface (OCI). This interface is below the GIOP layer where new mappings on transport protocols can be plugged in the ORB.

The OCI is designed for the use of bytestream-oriented transport protocols. As the session layer provides a message-oriented interface, we use a slightly adapted version of the OCI. This new interface preserves the structure of data sent and received by the ORB, which are GIOP messages. This is necessary to map the GIOP messages onto WTP-transactions. The identification of GIOP messages in form of request ids is also handed through this new interface to the session. We adapted the ORB to transparently use the new OCI interface with the session protocol and the old one with other protocols (like IIOP).

```
module wtiop {
  const unsigned long TAG_WTIOP_IOP = 4;
  const unsigned short IP = 1;

  struct Version {
      octet major;
      octet minor;
  };

  struct ProfileBody {
      Version wtiop_version;
      unsigned short bearerType;
      string host;
      sequence<octet> address;
      unsigned short port;
      sequence<octet> object_key;
  };
};
```

Fig. 7. IDL of WTIOP

In order to include a new PlugIn it is necessary to describe the address format in IDL. If the server supports multiple bearers we use multiple TaggedProfiles (one for each bearer) in a CORBA IOR (interoperable object reference). Right now only bearers providing IP are supported which may be used on a local wireless LAN or wireless WAN such as GSM or GPRS. Figure 7 shows the IDL file describing the address format. To identify the WTIOP profile of an object we defined a tag with a value of four because it is not used otherwise. For official use it is necessary to get a tag assigned from the OMG. Every server needs a globally unique hostname (which may be a DNS name). This is used together with the session id to identify a session at client side. The port number refers to the WDP port number the server uses, and the address field contains the address in a bearer dependent encoding. This may be an IP address (or DNS name) for bearers providing IP, but could also contain e.g. a telephone number for other bearers. The `bearerType` field defines the bearer for which the address is valid.

The notification of session-aware applications is implemented by using the callback mechanism of the so called *Info* objects defined in the OCI. With these objects, applications can register callback methods which will be invoked when the particular event occurs. The standard OCI already defines callbacks for connection and disconnection events. These will be called by the session layer in case of session establishment and closing a session, respectively. We extended the *Info* objects with the possibility for applications to register additional callback methods which will be invoked in case of suspending and resuming the session.

4.2 WAP-Details

For our implementation of WTIOP we needed an implementation of the WAP protocols, specifically of the WDP and WTP protocol layers. Unfortunately we could not find a complete free open source implementation of these protocols. With Kannel ([11]) there is an open source project building an open source WAP gateway. Because Kannel's design is influenced by its use as a gateway, it is not well suited to reengineer a wap-stack implementation from its source code. Jannel is a Kannel port to Java by Empower Interactive, Inc. which better fulfilled our needs. Also because we use the Java version of ORBacus and wanted to implement the session layer in Java as well, the Java implementation was preferred. But the Jannel implementation only contained the server side implementation of the WAP protocols. Thus, we implemented the client part of WTP by ourselves (mainly added the client state tables of WTP). Beside that, we had to make some minor changes to the original Jannel implementation to enable the use of WTP with our own session layer instead of WSP.

5 Related Work

There are several other approaches which deal with the mobility of clients and their wireless attachment to existent networks e.g. by supporting continuous communication during periods of mobility.

The end-to-end approach to host mobility presented in [21] allows to migrate one end of an active TCP connection to a different IP address. This approach as well as our design assumes that normally it is the client that changes its network attachment point (the IP address) and, therefore, no update of the location of the client has to be conducted, only existing connections must be migrated. If a server changes its location, the paper proposes to use a dynamic DNS update to reflect this change. However, there is no mechanism included to signal the event of connection-migration to the application which uses the TCP-connection and, furthermore, longer network disconnections cause an abort of the TCP-connection as in the original TCP (as a result of a TCP timeout). Another paper [22] by the same authors states that there is a need for notification of mobility-related events to applications and to hide longer periods of disconnection by introducing a session layer, but currently no implementation is available. Similar end-to-end approaches are used for migrating Java-Sockets [13] and TCP [18]. Split-connection proxies are used in MSOCKS [12] and OMIT [7]. However, all of them do not provide solutions to handle long disconnections or add a substantial overhead.

[8] introduces a system of dynamic network reconfiguration which monitors the availability of network interfaces and their status of physical connection. This enables applications to choose or change the network to be used for their communication based on availability, costs and provided services like throughput or latency. A similar system is needed by our session layer to be able to choose a bearer when resuming.

6 Conclusion and Future Work

This paper introduced a generic session layer based on WAP for distributed applications in the nomadic environment. This session layer relieves distributed application programmers from dealing with complex connection establishment and recovery in the wireless and mobile environment. Although this session layer is independent of any given architecture for distributed applications we showed its functionality and usefulness within a CORBA environment.

The session layer we conceived and implemented fulfills the requirements identified in Sec. 3.1. The implicit session establishment during the first request avoids the overhead of explicit session setup messages. Network disconnections are detected by the session layer even in cases in which the WTP protocol will not detect them. Both session-aware and legacy CORBA applications are fully supported. Session-aware applications may itself control the state of the session and must therefore be notified whenever an external session state modification has occurred. This is done via a special callback interface. But legacy applications will also benefit from the session layer, because network disconnections and reconnections are transparently handled. Without the session layer an error would occur in the case of a disconnection. Automatic reconnection and resuming of the session in these cases is provided. The lost reply problem introduced by disconnections or network handovers are solved transparently to the CORBA application by polling missing replies from the server after resuming the session.

Nevertheless, there still is some work to be done. First of all, the session layer shall also be evaluated through measurements on different systems. Furthermore, session establishment and recovery should also consider different transport service providers (in addition to bearers providing IP). Hence, dynamic network reconfiguration should be possible.

Finally, the session layer will be an integral part of our π^2 architecture [20], a framework supporting nomadic computing.

References

[1] Ajay Bakre and B.R. Badrinah. I-TCP: Indirect TCP for Mobile Hosts. In *Proceedings of the 15th International Conference on Distributed Computing Systems ICDCS-15*, May 1995.

[2] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz. Improving TCP/IP Performance over Wireless Networks. In *Proceedings of the 1st ACM International Conference on Mobile Computing and Networking (MOBICOM'95)*, 1995.

[3] E. Brewer and R. Katz. A network architecture for heterogeneous mobile computing. *IEEE Personal Communications*, 5(5):8–24, October 1998.

[4] K. Brown and S. Singh. M-TCP: TCP for mobile cellular networks. *ACM Computer Communications Review*, 27(5):19–43, October 1997.

[5] R. Caceres and L. Iftode. Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments. *IEEE Journal on Selected Areas in Communication*, 13(5):850–857, June 1995.

[6] J. Currey, K. Jin, K. Raatikainen, S. Aslam-Mir, and J. Korhonen. Wireless access and terminal mobility in corba. OMG Document telecom/2001-02-01, Object Management Group OMG, February 2001. Revised Submission to RFP telecom/99-05-05.

[7] Andreas Fieger and Martina Zitterbart. Migration support for indirect transport protocols. In *Proceedings of the International Conference on Universal Personal Communications*, San Diego, California, October 1997.

[8] Jon Inouye, Jim Binkley, and Jonathon Walpole. Dynamic network support for mobile computers. In *Proceedings of the Third ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '97)*, Budapest, Hungary, September 1997.

[9] Inprise Corporation and Highlander Engineering Inc. Wireless access and terminal mobility. ftp://ftp.omg.org/pub/docs/telecom/2000-05-05.pdf, May 2000.

[10] Iso/iec is 7498: Information processing systems – open systems interconnection – basic reference model. International Standard, 15. Oktober 1984.

[11] Kannel: Open source WAP and SMS gateway. http://www.kannel.org/, 2001.

[12] D. Maltz and P. Bhagwat. Msocks: An architecture for transport layer mobility, 1998.

[13] Tadashi Okoshi, Masahiro Mochizuki, Yoshito Tobe, and Hideyuki Tokuda. Mobilesocket: Session layer continuous operation support for java applications. Technical report, Graduate School of Media and Governance, Keio University, October 1999.

[14] Object Management Group (OMG). Telecom Domain Task Force: Request for Information (RFI) - Supporting Wireless Access and Mobility in CORBA. ftp://ftp.omg.org/pub/docs/telecom/98-06-04.pdf, June 1998.

[15] Object Management Group (OMG). CORBA/IIOP Specification Version 2.3.1. ftp://ftp.omg.org/pub/docs/formal/99-10-07.pdf, October 1999.

[16] Object Oriented Concepts (OOC). http://www.ooc.com, 2001.

[17] Charles Perkins. *IP Mobility Support/IP Encapsulation within IP*, October 1996. RFC 2002+2003.

[18] Xun Qu, Jeffrey Xu Yu, and Richard P. Brent. A mobile TCP socket. Technical Report TR-CS-97-08, Canberra 0200 ACT, Australia, 1997.

[19] Rainer Ruggaber and Jochen Seitz. A transparent network handover for nomadic CORBA users. In *Proceedings of the 21st International Conference on Distributed Computing Systems ICDCS-21*, Phoenix, Arizona, USA, April 2001.

[20] Rainer Ruggaber, Jochen Seitz, and Michael Knapp. Π^2 - a Generic Proxy Platform for Wireless Access and Mobility in CORBA. In *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC'2000)*, pages 191–198, Portland, Oregon, USA, July 2000.

[21] Alex C. Snoeren and Hari Balakrishnan. An end-to-end approach to host mobility. In *Proc. 6th International Conference on Mobile Computing and Networking (MobiCom)*, August 2000.

[22] Alex C. Snoeren, Hari Balakrishnan, and M. Frans Kaashoek. Reconsidering internet mobility. In *Proc. 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, 2001.

[23] Wireless Application Protocol Forum (WAP-Forum). http://www.wapforum.org/, 2000.