Wireless Access and Terminal Mobility in CORBA

Draft Adopted Specification

May 14, 2001
OMG Document dtc/2001-05-01

Copyright © 2001 Borland Software Corporation.
Copyright © 2001 Highlander Engineering Inc.
Copyright © 2001 Nokia.
Copyright © 2001 Sonera Corporation.
Copyright © 2001 University of Helsinki.
Copyright © 2001 Vertel Corporation.

All Rights Reserved.

The companies and organisations listed above hereby grant a royalty-free license to the Object Management Group (OMG) for world-wide distribution of this document or any derivative works thereof, so long as the OMG reproduces the copyright notices and the below paragraphs on all distributed copies.

The material in this document is provided for evaluation by the OMG. Submission of this document does not represent a commitment to implement any portion of this specification in the products of the above said companies and organisations.

While the information in the publication is believed to be accurate, the companies and organisations listed above make no warranty of any kind with regard to this material including but not limited to the implied warranties of merchantability and fitness for a particular purpose. The companies and organisations listed above shall not be liable for errors contained herein or for incidental or consequent damages in connection with the furnishing, performance or use of this material. The information contained in this document is subject to change without notice.

This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright here on may be reproduced or used in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without permission of the copyright owners. All copies of this document must include the copyright and other information contained on this page.

The copyright owners grant the Object Management Group permission to reproduce and use the information contained in this document. The copyright owners grant a limited waiver of the copyright on this document to members of the Object Management Group so that they can each reproduce up to 50 copies of this document for their internal use as part of the OMG evaluation process.

RIGHTS RESERVED LEGEND. Use, duplication, or disclosure by government is subject to restrictions as set forth in subdivision (c)(1)(ii) of the Right in Technical Data and Computer Software Clause at DFARS 252.227.7013.

All trademarks acknowledged.

Contact Persons

Ke Jin
Borland Software Corporation
951 Mariner's Island Blvd. Suite 120
San Mateo, CA 94404
USA
Phone: +1 650 286 1900
Fax: +1 650 358 3099
Email: kejin@borland.com

Jon Currey
Highlander Engineering Inc.
208 East Pine Street
Lakeland, FL 33801
USA
Phone: +1 863 686 7767
Fax: +1 863 687 7767
Email: jon@highlander.com

Dr. Kimmo Raatikainen
Nokia Research Center
P.O. Box 407
FIN-00045 NOKIA GROUP
Finland
Phone: +358 7180 36275
Fax: +358 7180 36308
Email: kimmo.raatikainen@nokia.com

Dr. Shahzad Aslam-Mir
VERTEL Corporation
5825 Oberlin Dr., Ste# 300
San Diego., CA. 92121
USA
Phone: +1 858 824 4128
Fax +1 858 824 4110
E-mail: sam-aslam-mir@vertel.com

Mr. Jouni Korhonen
Sonera Corporation
P.O. Box 970
FIN-00051 SONERA
Finland
Phone: +358 2040 65342
Fax: +358 2040 64365
Email: jouni.korhonen@sonera.com

Prof. Kimmo Raatikainen
Department of Computer Science
P.O. Box 26 (Teollisuuskatu 23)
FIN-00014 UNIVERSITY OF HELSINKI
Finland
Phonel: +358 9 1914 4243
Fax: +358 9 1914 4441
Email: kimmo.raatikainen@cs.helsinki.fi

Overview 1

This document specifies an architecture and interfaces to support wireless access and terminal mobility in CORBA as requested in the RFP telecom/99-05-05.

1.1 Submission Overview

The submission is organized as follows.

In the current chapter we describe our design rational and make our statement of proof of concept. We also address each mandatory and optional requirement in the RFP as well as issues to be discussed.

In Chapter 2 we present the Architectural Framework. In Chapters 3 through 7 each of the key concepts, that is Mobile IOR, Home Location Agent, Access Bridge, Terminal Bridge, and GIOP Tunneling, is described in details.

Chapter 8 is devoted to handoff and access recovery.

In Chapter 9 we address the issues asked to be discussed: Interoperable Naming Service, Notification Service, and Messaging Service.

In Chapter 10 we summarize the mandatory and optional requirements.

The associated IDL specifications are given in Chapter 11.

1.2 Design Rationale

The basic design principles have been client-side ORB transparency and simplicity.

Transparency of the mobility mechanism to non-mobile ORBs has been the primary design constraint. The submitters have rejected all solutions which would require modifications to a non-mobile ORB in order for it to interoperate with CORBA objects

and clients running on a mobile terminal. In other words, a stationary (non-mobile, or fixed network) ORB does not have to implement this specification in order to interoperate with CORBA objects and clients running on mobile terminals.

The RFP requested a quite comprehensive specification to cover wireless access, terminal mobility and service provisioning in a mobile environment. The submitters decided to take a minimalistic approach. The response has been designed to provide a minimal useful functionality for CORBA applications, in which the client, the server, or both of them are running on a host that can move.

1.3 Proof of Concept

The design is heavily affected by experiences of the EC/ACTS project DOLMEN (AC036) that implemented a prototype of CORBA extensions to support terminal mobility. The DOLMEN solution is described, for example, in the OMG Document telecom/98-08-08.

Most of this specification has been implemented by University of Helsinki as an extension to MICO.

1.4 Mandatory Requirement

Architectural framework

Proposals shall provide an architectural framework, which is compliant to OMA, for a way how mobile terminals can dynamically attach to a mobility domain, detach from such a domain, and move from one mobility domain to another one. In particular, the proposed framework should be compliant to the CORBA Interoperability Architecture. Proposals shall also define the concept of 'mobility domain' in details, or propose an alternative concept relevant to wireless access and terminal mobility.

GIOP mapping onto Internet transport protocol (TCP or UDP) over wireless links

Proposals shall provide GIOP mappings onto TCP or UDP over wireless links. In particular, the proposals shall address handoff and reliability issues. Proposals must motivate the choice between TCP and UDP. If UDP is used, then proposals shall specify how interworking with IIOP is sustained. As minimum the mapping must provide recovery from sudden drop of a link-level connectivity.

Mechanism that hides from CORBA clients the mobility of terminals on which CORBA servers are running

When a CORBA server runs on a mobile terminal, its clients should not be aware that the server may change its location after registering its object reference to a naming service. Therefore, proposals shall specify how the mobility transparency is obtained.

Mechanism for initial access to a new mobility domain

Proposals shall specify how a mobile terminal gets initial access to a mobility domain. In fixed networks initial access is taken care by the network technology and by the administrative domain. Therefore, it is implicit to ORB. When terminals can move from one network to another one and from one administrative domain to another one, it is unrealistic to assume that the same implicit procedure is available everywhere. Therefore, terminal mobility requires that an explicit standard procedure is specified. Proposals shall specify how mobility domains advertise availability of their services to mobile terminals that want to attach to the domain. Alternatively, proposals shall specify how a mobile terminal can contact a mobility domain. Proposals shall also specify how authentication of terminals should be carried out.

Mechanism for finding the necessary basic set of CORBA services in mobility domain

When a mobile terminal has attached a new mobility domain, the CORBA objects on the terminal need to find a basic set of CORBA services in that domain. Proposals shall specify what is the minimum basic set of CORBA services and how objects on a terminal learns what services are available in the domain.

Mechanism for advertising CORBA services available on a mobile terminal

When a mobile terminal wants that it can be contacted, it must provide one or more CORBA servers. Proposal shall specify how CORBA servers on a mobile terminal advertise and register themselves so that CORBA clients can obtain references of those servers.

Mechanism for handoff between mobility domain

In handoff, there are two basic ways of carrying it out: forward and backward. In the backward handoff the mobile terminal has connectivity to the old mobility domain that usually prepares the handoff. In the forward handoff the mobile terminal has lost the connectivity to the old mobility domain but obtained connectivity to a new mobility domain. In this case the handoff is prepared by the new one. In both cases no data should be lost. It should be noted that recovery of connectivity may result in gaining access to the old mobility domain (access recovery) or to a new one (forward handoff). Proposals shall specify the forward and backward handoff procedures as well as the access recovery procedure.

The Architectural Framework does not define the concept of ‘mobility domain’ as requested in the RFP. Instead, concepts of ‘home domain’, ‘visited domain’, and ‘terminal domain’ are used.

The response does not define a GIOP mapping onto Internet transport protocol (TCP or UDP) over wireless links. Instead, the response defines how GIOP messages are to be tunneled between bridges. This was regarded as a more elegant way of using link specific transport mechanisms. The response specifies a generic GIOP Tunneling Protocol and how this protocol is run over TCP, UDP, and WAP Wireless Datagram Protocol (WDP).

An initial access mechanism is not specified since it was regarded too network technology and access provider specific to be in the scope of OMG.

The response specifies a simple discovery mechanism similar to resolving initial references in the ORB pseudo-interface.

The response does not propose a mechanism for advertising CORBA services available on a mobile terminal. This was regard as unnecessary. Instead, objects on the terminal can use either Naming Service or Trader Service either in the Home Domain or in a Visited Domain.

The handoff support is defined as an optional feature. The main motivation was the fact that there are clear business cases for "discrete" terminal mobility. By discrete terminal mobility we mean terminal's ability to change its point of presence when there are no outstanding invocations. Typical examples would be a traveling employee visiting a remote site and a home user whom gets a temporary IP address from the ISP.

1.5 Optional Requirements

GIOP mappings onto other wireless transport protocols

Proposals may provide GIOP mappings onto one of the WAP protocols defined in the WAP 1.0 specifications. Proposals should give motivation for the selection of a WAP protocol.

Proposals may provide GIOP mappings onto wireless transport protocols other than required in 6.5.2.

Wireless/Mobility specific ES-IOP

Proposals may provide specification of an ES-IOP targeted for wireless and mobile networks. If such a specification (xES-IOP) is provided, then the proposal must also provide specification of IIOP/xES-IOP bridge.

This response does not directly address the optional requirements. However, the proposed GIOP Tunneling Protocol over WAP WDP can be regarded as a response to the first optional requirement.

1.6 Issues to be discussed

Relationship to Notification Service [telecom/98-11-01]

Proposals should discuss how Notification Service can be used together with the proposed technology.

Relationship to Messaging Service [orbos/98-05-05]

Proposals should discuss how Messaging Service can be used together with the proposed technology.

Relationship to Interoperable Naming Service [orbos/98-10-11]

Proposals should discuss how Interoperable Naming Service can be used together with the proposed technology.

Proposals should also discuss how CORBA servers and clients on a mobile terminal use Naming Service of the mobility domain.

This response discusses usage of Notification Service, Interoperable Naming Service, and Messaging Service in Chapter 9.

The response specifies, as an optional feature, notifications of terminal mobility events.

1.7 References

[GFD] WAP Forum. WAP General Formats Document. WAP Forum document WAP-188-WAPGenFormats, Version 15-Aug-2000.

[WDP] WAP Forum. Wireless Datagram Protocol Specification. WAP Forum Document WAP-200-WDP, Approved Version 19-February-2000.

Architectural Framework 2

The key concepts in the this specification are:

- Mobile IOR,
- Home Location Agent,
- Access Bridge,
- Terminal Bridge, and
- GIOP Tunneling Protocol.

The **Mobile IOR** is a relocatable object reference. It identifies the Access Bridge and the terminal on which the target object resides. In addition, it identifies the Home Location Agent that keeps track of the Access Bridge to which the terminal is currently attached.

The **Home Location Agent** keeps track of the current location of the terminal. It provides operations to query and update terminal location. The Home Location Agent also provides operations to get a list of initial services and to resolve initial references in the home domain.

The **Access Bridge** is the network side end-point of the GIOP tunnel. It encapsulates the GIOP messages to the Terminal Bridge and decapsulates the GIOP messages from the Terminal Bridge. The Access Bridge also provides operations to get a list of initial services and to resolve initial references in the visited domain. The Access Bridge may also provide notifications of terminal mobility events.

The **Terminal Bridge** is the terminal side end-point of the GIOP tunnel. It encapsulates the GIOP messages to the Access Bridge and decapsulates the GIOP messages from the Access Bridge. The Terminal Bridge may also provide a mobility event channel that delivers notifications related to handoffs and connectivity losses.

The **GIOP tunnel** is the means to transmit GIOP messages between the Terminal Bridge and the Access Bridge. The generic GIOP Tunneling Protocol defines how GIOP messages are transmitted. The protocol also specifies necessary control

messages to establish, release, and re-establish a GIOP tunnel. The proposed GIOP Tunneling Protocol (GTP) is an abstract, transport-independent protocol. This response defines three concrete tunneling protocols, that is the way how GTP messages are transmitted over TCP, UDP and WAP WDP.

The overall architecture is depicted in Figure 2-1. It identifies three different domains: home domain, visited domain, and terminal domain. The **Home Domain** for a given terminal is the domain that hosts the Home Location Agent of the terminal. A **Visited Domain** is a domain that hosts one or more Access Bridges through which it provides ORB access to some mobile terminals. The **Terminal Domain** consists of a terminal device that hosts an ORB and a Terminal Bridge through which the objects on the terminal can communicate with objects in other networks.

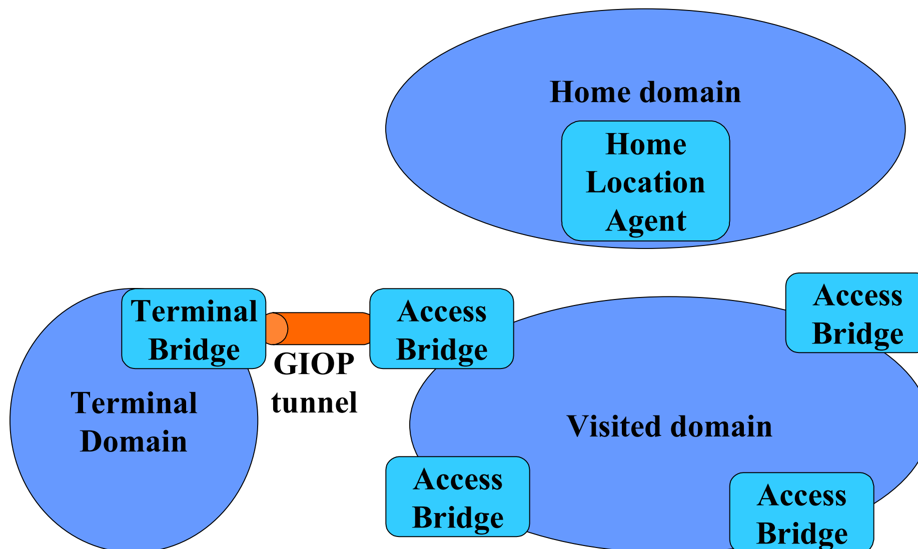


Figure 2-1 Architecture for Terminal Mobility in CORBA

Mobile IOR 3

A Mobile IOR is a special Interoperable Object Reference that hides the mobility of a terminal from clients that invoke operations on target objects located on the terminal. The Mobile IOR provides mobility transparency in a way that is itself transparent to the ORB that a client runs on. Hence the ORB that a non-mobile client runs on is not required to implement the Wireless Access and Terminal Mobility specification for terminal mobility to be available.

A Mobile IOR contains the normal IIOP Profile (TAG_INTERNET_IOP) required in an IOR, plus a 'Mobile Terminal' Profile (TAG_MOBILE_TERMINAL_IOP). There may be more than one IIOP Profile in the Mobile IOR. There can be only one Mobile Terminal Profile instance in the Mobile IOR.

3.1 IIOP Profiles in Mobile IOR

The ORB that a client runs on uses an IIOP Profile from the Mobile IOR (rather than the Mobile Terminal Profile) to route the client's invocations to the Access Bridge currently serving the terminal on which the target object is located.

The IIOP Profile or Profiles in a Mobile IOR have the normal structure defined in IIOP::ProfileBody, but they have additional semantics regarding the address and object key fields within that structure. These semantics are transparent to the client ORB that makes use of one of these Profiles.

3.1.1 Address information in IIOP Profiles in Mobile IORs

Instead of indicating the address of the target object, the host and port information in an IIOP Profile in a Mobile IOR indicate the address of either the target object's terminal's Home Location Agent or the Access Bridge that the terminal was last known to be associated with. When a Mobile IOR is created at the terminal, the terminal ORB chooses whether the address of the terminal's HLA or the Access Bridge the terminal is currently associated with is given in the IIOP Profile.

If the address in the IIOP Profile is that of the terminal's Home Location Agent, rather than its last known Access Bridge, when a client first performs an invocation upon the Mobile IOR, the HLA replies with a GIOP LOCATION_FORWARD message returning the object reference of the Access Bridge that the HLA believes the terminal is currently associated with.

If the address in the IIOP Profile is that of an Access Bridge rather than an HLA, the terminal may no longer be associated with that Access Bridge when a client makes its first invocation upon the Mobile IOR. If the terminal is now associated with another Access Bridge, the contacted Access Bridge replies with a GIOP LOCATION_FORWARD message returning the object reference of the Access Bridge that the terminal is currently associated with.

Similarly, if at any time after a client has made its first invocation upon a Mobile IOR the terminal becomes associated with another Access Bridge, then the contacted Access Bridge will reply to the client's next invocation with a GIOP LOCATION_FORWARD message returning the object reference of the Access Bridge that the terminal is now associated with.

3.1.2 Mobile Object Key Format

In order to allow clients to make invocations from ORBs that only support versions of GIOP prior to version 1.2, Mobile IORs may optionally use a special format for the contents of the object key field within their IIOP Profiles. For details of this format see Section 3.4.

3.2 The Mobile Terminal Profile

The Mobile Terminal Profile within a Mobile IOR contains information that the Home Location Agent and Access Bridges require to provide mobility transparency for target objects that have Mobile IORs. The information is not required by the ORB that a client of a Mobile IOR runs on, and hence only ORBs used to implement Home Location Agents and Access Bridges need to be able to use this profile type.

3.2.1 Mobile Terminal Profile Structure

A Mobile Terminal profile is an IOP::TaggedProfile with a tag value of TAG_MOBILE_TERMINAL_IOP and profile data with the structure defined by MobileTerminal::ProfileBody.

```
const IOP::ProfileID            TAG_MOBILE_TERMINAL_IOP = ????;
```

Note – The constant value is to be assigned by the OMG and added to the IOP module in the CORBA specification.

```
module MobileTerminal {

    typedef sequence<octet> TerminalId;
    typedef sequence<octet> TerminalObjectKey;

    struct Version {
        octet     major;
        octet     minor;
    };

    struct ProfileBody {
        Version              mior_version; // version of Mobile IOR
        octet                reserved;
        TerminalId           terminal_id; // unique terminal identifier
        TerminalObjectKey    terminal_object_key; // object_key on terminal
        sequence <IOP::TaggedComponent> components;
    };

    ...

};
```

The MobileTerminal::ProfileBody structure identifies the version of the Mobile Terminal Profile used, the id of the terminal the target object resides on and the object key of the target object on the terminal. It may optionally include one or more tagged components. A TAG_HOME_LOCATION_INFO component is specified, and may be present in the Mobile Terminal Profile's component list. See Section 3.2.2.

There can be only one Mobile Terminal profile instance in the Mobile IOR.

3.2.2 TAG_HOME_LOCATION_INFO Component

The TAG_HOME_LOCATION_INFO component identifies the Home Location Agent of the terminal on which the Mobile IOR was created. If the mobile terminal has an Home Location Agent, then the TAG_HOME_LOCATION_INFO component must be present in the Mobile Terminal Profile.

If the mobile terminal does not have an Home Location Agent, then the object reference is only valid as long as the current GIOP tunnel between the Terminal Bridge and the Access Bridge exists. Such a terminal is referred as a "homeless terminal" in this specification.

The TAG_HOME_LOCATION_INFO component has a Home Location Agent object reference as its associated value, encoded as the CDR encapsulation of the data structure MobileTermial::HomeLocationInfo.

```
const IOP::ComponentID     TAG_HOME_LOCATION_INFO = ????;
```

Note – The constant value is to be assigned by the OMG and go into the IOP module in the CORBA Core.

```
module MobileTerminal {

    ...

    struct HomeLocationInfo {
        MobileTerminalHomeLocationAgent::home_location agent agent;
    };

    ...

};
```

The TAG_HOME_LOCATION_INFO component can appear at most once in a TAG_MOBILE_TERMINAL_IOP profile.

3.3 Translation to Mobile Target Object

The first time a Home Location Agent or Access Bridge receives a GIOP message for an invocation on a particular Mobile IOR it needs some way to establish the terminal id and object key of the mobile target object, and associate it with the object key included in the GIOP message (so that in the future it will know that messages containing that object key are intended for that same mobile target object.)

In GIOP 1.2 the Home Location Agent or Access Bridge can reply to the first message with the status NEEDS_ADDRESSING_MODE, to request the object reference of the target object. It can then examine the contents of the Mobile Terminal profile within that object reference, to obtain the terminal id and object key. However, that solution excludes clients running on an ORB using GIOP 1.0 or 1.1 from invoking on the Mobile IOR, as the NEEDS_ADDRESSING_MODE status cannot be returned to them by the HLA or Access Bridge.

3.4 Interoperability with GIOP 1.0 and 1.1

Since, in GIOP 1.0 and 1.1 the object key is the only available way of identifying the target from data in a GIOP Request header, a special Mobile Object Key (MOK) format is specified to allow invocations from GIOP 1.0 and 1.1 clients to be made on mobile target objects. It is a structure that may optionally be used to format the contents of the object key in the IIOP profile in the Mobile IOR.

When the MOK format is used, the contents of the object key is an encapsulation of four octets with the ASCII values ‘M’, ‘I’, ‘O’, ‘R’, followed by the structure MobileTerminal::MobileObjectKey.

```
module MobileTerminal {
    ...
    struct MobileObjectKey {
        Version mior_version;
        octet reserved;
        TerminalId terminal_id;
        TerminalObjectKey terminal_object_key;
```

```
        };
    };
```

Use of the MOK format is optional. Even when the MOK format is used, the Mobile Terminal Profile is still included in the Mobile IOR - which means the terminal id and target object information are included twice in the object reference. This redundancy is allowed because the MOK solution is only offered to support legacy ORBs that do not support GIOP 1.2. The GIOP 1.2 mechanism is preferred, and hence always supported to assist the migration of systems to GIOP 1.2 support.

If the MOK format is used, the contents of the formatted key are only examined by the Home Location Agent and Access Bridge, which will use ORBs that implement the this specification. The MOK is not examined by client ORBs, which continue to consider the object key as an opaque piece of data. Hence non-mobile aware client ORBs are able to interoperate with target objects which have Mobile IORs that use the MOK format.

3.5 Additional Type Definitions

The **MobileTerminal** module contains all the type definitions used in this specification. They are provided below.

```
module MobileTerminal {

    ...

    typedef sequence<octet>     GIOPEncapsulation; // used in GIOP tunneling
    typedef sequence<octet>     GTPEncapsulation; // used in GTP forwarding

    enum HandoffStatus {
        HANDOFF_SUCCESS,
        HANDOFF_FAILURE,
        NO_MAKE_BEFORE_BREAK
    };          // used to report status of handoff

    struct GTPInfo {
        short       protocol_id; // identifies GIOP Tunneling Protocol
        Version     gtp_version; // version of the GTP
    };          // identifies the GIOP Tunneling Protocol
    // negative values of protocol_id element are reserved for internal use

    const short     TCP_TUNNELING = 0;
    const short     UDP_TUNNELING = 1;
    const short     WAP_TUNNELING = 2;

    struct AccessBridgeTransportAddress {
        GTPInfo             tunneling_protocol;
        sequence<octet>     transport_address;
    };      // identifies transport access point of the Access Bridge

    typedef sequence<AccessBridgeTransportAddress>
AccessBridgeTransportAddressList;

    typedef string ObjectId; // same as CORBA::ORB::ObjectId
    typedef sequence<ObjectId> ObjectIdList
                // same as CORBA::ORB::ObjectIdList

};
```

Home Location Agent 4

The Home Location Agent keeps track of the Access Bridge that a mobile terminal is currently associated with. That is, which Access Bridge objects on the terminal can currently be invoked. It provides operations to update and to query the current location. It also provides operations resolve initial references in the Home Domain.

4.1 Location Update

The HomeLocationAgent interface provides operations for Access Bridges to carry out location updates and to query the current location of a terminal. The terminal is identified by a terminal identifier, terminal_id. The Home Location Agent may require the use of the CORBA Security Service to invoke the update_location operation.

```
module MobileTerminal {

    interface HomeLocationAgent {

        void update_location (
            in TerminalId terminal_id,
            in AccessBridge new_access_bridge,
        ) raises (UnknownTerminalID, IllegalTargetBridge);

        ...
    };
};
```

Parameters

terminal_id	terminal for which the location update is done
new_access_bridge	object reference of the Access Bridge that wants to serve the terminal

Exceptions

UnknownTerminalID The HLA raises this exception, if it is not the HLA serving the

	terminal identified by the terminal_id
IllegalTargetBridge	The HLA raises this exception, if it does not accept the Access Bridge identified by new_access_bridge to serve the terminal identified by terminal_id.

When an Access Bridge has lost the terminal, it deregisters the location of the terminal by invoking update_location(terminal_id, NIL) at the Home Location Agent.

If an Access Bridge needs to query the current location of a terminal, that is the Access Bridge currently serving the terminal, it can invoke the query_location operation at the Home Location Agent of the terminal. The Home Location Agent may require the use of the CORBA Security Service to invoke the query_location operation.

```
module MobileTerminal {

    interface HomeLocationAgent {

        ...

        void query_location (
            in TerminalId terminal_id,
            out AccessBridge current_access_bridge
        ) raises (UnknownTerminalID, UnknownTerminalLocation);

        ...

    };
};
```

Parameters

terminal_id	identifies the terminal the location of which is queried.
current_access_bridge	object reference of the Access Bridge to which the HLA believes that the terminal is currently attached.

Exceptions

UnknownTerminalID	The HLA raises this exception, if it is not the HLA serving the terminal identified by the terminal_id
UnknownTerminalLocation	The HLA raises this exception, if the given terminal has not registered its current location through an Access Bridge, or the paging procedure did not find the Access Bridge to which the terminal is attached.

4.2 Discovery

The Home Location Agent provides discovery operations so that the terminals can resolve initial references to CORBA services available in the Home Domain. The operations are list_initial_services and resolve_initial_references. They are the same as provided by the ORB pseudo interface for local applications.

```
module MobileTerminal {

    interface HomeLocationAgent {

        ...

        ObjectIdList list_initial_services();
        Object resolve_initial_references(
            in ObjectId identifier
        ) raises(InvalidName);

    };
};
```

4.3 Message Processing

When the Home Location Agent receives a GIOP message targeted to a terminal, its behavior depends on whether or not it currently has an Access Bridge associated with that terminal. If it does, it replies with the LOCATION_FORWARD status and returns the Mobile IOR identifying the current Access Bridge. If not, it replies with the system exception OBJECT_NOT_EXIST (to a Request) or with the UNKNOWN_OBJECT status (to a Locate Request).

4.4 Terminal Ids

The **TerminalId**s need to be unique world-wide.

One possible scheme that may be used to achieve this is to concatenate the following information to produce each identifier:

- IP version (1 byte)
- IP address (4 or 16 bytes), and
- local_id (variable number of bytes).

The IP address can be any IP address that is owned by the organization that is generating the TerminalId, and the local_id is a unique identifier within that organization.

Any other scheme may be used though, as long as it produces globally unique identifiers.

Access Bridge 5

The Access Bridge encapsulates/decapsulates the GIOP messages to/from the Terminal Bridge using a GIOP Tunneling Protocol. It also provides operations to get a list of initial services and to resolve initial references in the visited domain. In addition, the Access Bridge may support handoff. The Access Bridge may also provide notifications related to movements of terminals.

GIOP Tunneling Protocols are described in Chapter 7. The handoff procedures are described in Chapter 8.

5.1 Discovery

The Access Bridge provides discovery operations so that the terminals can resolve initial references to CORBA services available in the Visited Domain. The operations are list_initial_services and resolve_initial_references. They are the same as provided by the ORB pseudo interface for local applications.

```
module MobileTerminal {

    interface AccessBridge {
        ObjectIdList list_initial_services();

        Object resolve_initial_references(
            in ObjectId identifier
        ) raises(InvalidName);

        ...

    };
};
```

5.2 Query

The Access Bridge also provides query operations that can be used to query whether or not a specific terminal is attached to the bridge, and the address information for the Access Bridge:

```
module MobileTerminal {

    interface AccessBridge {

        ...

        Boolean terminal_attached (
            in TerminalId terminal_id
        );

        void get_address_info (
            out AccessBridgeTransportAddessList transport_address_list
        );

        ...

    };
};
```

If the HLA requires the CORBA Security Service to be used in location update, then the Access Bridge must use the CORBA Security Service to protect the usage of the terminal_attached operation. The Access Bridge may also use the CORBA Security Service to protect the get_address_info operation.

5.3 Message Processing

The Access Bridge acts as a relay between the server and client. It maintains bindings between terminal_id and the transport address of the GIOP tunnel to the terminal. For each terminal the Access Bridge keeps a state of outstanding invocations. An outstanding invocation is a GIOP message to which a reply is expected.

When the bridge gets a message targeted to a terminal, it encapsulates the message to the GIOP tunneling protocol in use and sends it to the GIOP tunnel address associated with the terminal_id.

If the Access Bridge does not have a tunneling association with the terminal, then it can query the current location of the terminal from the HLA or it can replace the IOR so that the HLA is in the IIOP Profile. In both cases the Access Bridge must reply with the LOCATION_FORWARD status.

If the IOR does not have TAG_HOME_LOCATION_INFO component or the Access Bridge does not know the HLA of the terminal, then the Access Bridge must reply with the system exception OBJECT_NOT_EXIST to a Request and with the UNKNOWN_OBJECT status to a Locate Request.

If the Access Bridge gets a reply the target of which is on a terminal that has moved to a new Access Bridge, it can use the forwarding mechanism described in Chapter 8. If the Access Bridge does not support handoff, then it should silently discard the Reply message.

When the Access Bridge gets an encapsulated GIOP message from a terminal, it decapsulates the message and forwards it to the target.

5.4 Mobility Event Notifications

The Access Bridge may, optionally, raise terminal mobility related events through a Notification Service Event Channel. The following Event types are defined so that if the Access Bridge does this, it may use standard events:

```
module MobileTerminalNotification {

    struct HandoffDepartureEvent {
        MobileTerminal::TerminalId terminal_id;
        MobileTerminal::AccessBridge new_access_bridge;
    };

    struct HandoffArrivalEvent {
        MobileTerminal::TerminalId terminal_id;
        MobileTerminal::AccessBridge old_access_bridge;
    };

    struct AccessDropoutEvent {
        MobileTerminal::TerminalId terminal_id;
    };

    struct AccessRecoveryEvent {
        MobileTerminal::TerminalId terminal_id;
    };

    ...

};
```

When a terminal moves from an old Access Bridge to a new Access Bridge, the old Access Bridge supplies the HandoffDepartureEvent and the new Access Bridge supplies the HandoffArrivalEvent.

When a terminal establishes the GIOP tunnel to the Access Bridge for the first time, then Handoff, then the the new Access Bridge supplies the HandoffArrivalEvent with NIL as reference to the old Access Bridge. When a terminal closes the GIOP tunnel to the Access Bridge, then the Access Bridge supplies the HandoffDepartureEvent with NIL as reference to the new Access Bridge.

When an Access Bridge detects that transport connectivity to a terminal has dropped, it supplies the AccessDropoutEvent. If the terminal re-establishes the GIOP Tunnel to the same Access Bridge, then the Access Bridge supplies the AccessRecoveryEvent if

it has supplied the AccessDropoutEvent. If the terminal re-establish the GIOP Tunnel to a new Access Bridge, then the old Access Bridge supplies the HandoffDepartureEvent and the new Access Bridge supplies the HandoffArrivalEvent.

Terminal Bridge 6

The Terminal Bridge encapsulates/decapsulates the GIOP messages to/from the Access Bridge using a GIOP Tunneling Protocol. The Terminal Bridge may support handoff. As an optional feature, the Terminal Bridge may also provide notifications of mobility related events for mobility-aware applications on the mobile terminal.

GIOP Tunneling Protocols and handoff procedures are described in Chapters 7 and 8, respectively.

6.1 Mobility Event Notifications

The Terminal Bridge may, optionally, raise terminal mobility related events through a Notification Service Event Channel. The following Event types are defined so that if the Terminal Bridge does this, it may use standard events.

```
module TerminalMobilityNotification {

    ...

    struct TerminalHandoffEvent {
        MobileTerminal::AccessBridge new_access_bridge;
    };

    struct TerminalDropoutEvent {
        MobileTerminal::TerminalId terminal_id;
    };

    struct TerminalRecoveryEvent {
        MobileTerminal::TerminalId terminal_id;
    };
};
```

When the Terminal Bridge detects that it has lost transport connectivity to the Access Bridge, it supplies the TerminalDropoutEvent. When the GIOP Tunnel has been re-established, then the Terminal Bridge generates the TerminalRecoveryEvent if the Access Bridge is the same as before. If the Access Bridge is different, then the Terminal Bridge supplies the TerminalHandoffEvent.

When a handoff takes place, the Terminal Bridge supplies the TerminalHandoffEvent. The Terminal Bridge also supplies the TerminalHandoffEvent, when the Terminal establishes the GIOP Tunnel to an Access Bridge for the first time. When the Terminal Bridge closes the GIOP Tunnel, then it supplies the TerminalHandoffEvent with NIL as the new_access_bridge.

GIOP Tunneling 7

A GIOP tunnel is the means to transmit GIOP and tunnel control messages between a Terminal Bridge and an Access Bridge. There is only ever one GIOP tunnel between a given Terminal Bridge and Access Bridge. However, a graceful handoff behavior is defined so that the Terminal Bridge can seamlessly transfer the GIOP Tunnel from the current Access Bridge to a new one. If the terminal can have simultaneous transport connectivity to two Access Bridges, then the Terminal Bridge creates a new tunnel to a new Access Bridge before shutting down the tunnel to the previous Access Bridge.

A tunnel is shared by all GIOP connections to and from the terminal it is associated with. The tunneling protocol allows multiplexing between the GIOP connections.

The GIOP Tunneling Protocol (GTP) is an abstract, transport-independent protocol. It defines message formats for establishing, releasing, and re-establishing (recovery) the tunnel as well as for transmitting and forwarding GIOP messages. The GTP protocol also defines messages for establishing and releasing GIOP connections through the Access Bridge. Figure 7-1 depicts the protocol architecture.

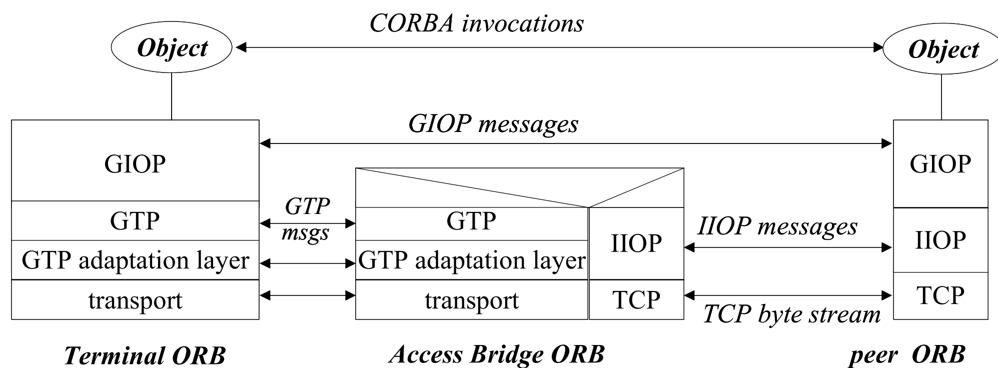


Figure 7-1 GIOP Tunneling Protocol Architecture

Since the GIOP Tunneling Protocol is an abstract protocol, it needs to be mapped onto one or more concrete protocols. This specification defines three concrete tunneling protocols: TCP Tunneling, UDP Tunneling, and WAP Tunneling.

The GTP is designed so that the specification of a concrete tunneling protocol is simple. The specification of a concrete tunneling protocol is provided as an adaption layer between the GIOP Tunneling Protocol and a transport layer protocol. The adaptation layer needs only to define how the transport is to be used and the data format of the transport address of the transport end-point.

7.1 Tunnel Establishment

GIOP tunnel establishment consists of two phases: 1) Transport end-point detection and 2) Establishment of the GIOP tunnel. Transport end-point detection is discussed below. The establishment of the GIOP tunnel is specified in Chapter 7.2.

7.1.1 Transport End-Point Detection

The detection of transport end-points on the link, network, and transport layers. It also depends on the provider of the Access Bridge. Therefore, transport end-point detection is out of the scope of this specification.

7.2 GIOP Tunneling Protocol

The GIOP Tunneling Protocol (GTP) assumes that the underlying concrete tunneling protocol (that is, the adaption layer between the GTP and a transport protocol) provides the same reliability and ordered delivery of messages assumed by the GIOP. If the underlying transport protocol does not provide this level of service, then the adaption layer that resides between the GTP and the actual transport protocol will provide this level of service.

7.2.1 GTP Message Structure

All GTP messages contain a header of eight octets and contents of variable (possibly null) length.

The GTP header has the structure of

```
struct GTPHeader {
    unsigned short seq_no;
    unsigned short last_seq_no_received;
    octet gtp_msg_type;
    octet flags;
    unsigned short content_length;
};
```

The seq_no element runs from 1 (0x0001) to 65535 (0xFFFF). The value 0x0000 can only appear in tunnel establishment request messages and an associated reply. The sequence number counting follows the usual modulo arithmetic with the exception that the seq_no 0x0001 follows the seq_no 0xFFFF.

The last_seq_no_received element indicates the highest sequence number of GTP messages received by the sender.

The gtp_msg_type element indicates the GIOP Tunneling Protocol message type. It defines how the receiver should interpret the body of the GTP message.

The flags element indicates the Endianness used in the GTP header and in GTP control messages. The leftmost bit tells the Endianness: 0x00 Big-Endian and 0x80 Little-Endian. The remaining seven bits are reserved for future usage.

The content_length element (unsigned short) tells the length of the GTP message.

7.2.2 GTP Messages

The GTP Messages are listed in the table below. Descriptions of the messages are given in the following subsections.

Table 7-1

Message name	gtp_msg_type	GTP version
IdleSync	0x00	1.0, 2.0
EstablishTunnelRequest	0x01	1.0, 2.0
EstablishTunnelReply	0x02	1.0, 2.0
ReleaseTunnelRequest	0x03	1.0, 2.0
ReleaseTunnelReply	0x04	1.0, 2.0
HandoffTunnelRequest	0x05	2.0
HandoffTunnelReplyCompleted	0x06	2.0
OpenConnectionRequest	0x07	1.0, 2.0
OpenConnectionReply	0x08	1.0, 2.0
CloseConnectionRequest	0x09	1.0, 2.0
CloseConnectionReply	0x0A	1.0, 2.0
ConnectionCloseIndication	0x0B	1.0, 2.0

Table 7-1

Message name	gtp_msg_type	GTP version
GIOPData	0x0C	1.0, 2.0
GIOPDataReply	0x0D	1.0, 2.0
GTPForward	0x0E	2.0
GTPForwardReply	0x0F	2.0
Error	0xFF	1.0, 2.0

7.2.3 IdleSync Message

The IdleSync message does not have a message body.

Source: Terminal Bridge and Access Bridge

Description: It is used by the Terminal Bridge and the Access Bridge to acknowledge GTP messages after some implementation dependent timeout. This allows the other side of the tunnel to release sent messages in a timely fashion, during a period when no messages are being sent in the opposite direction. If messages are being sent in the opposite direction, there is no need to send this message, as the synchronization occurs through the gtp_header.last_seq_no_received element of each sent message.

Special Notes: None

Forwardable: Yes (This GTP message can be encapsulated and sent in the GTPForward message). This will be used by either the Terminal Bridge or an old Access Bridge to acknowledge replies to forwarded GTP messages.

7.2.4 EstablishTunnelRequest Message

The EstablishTunnelRequest message has a message body containing the CDR encoded value of

```
union EstablishTunnelRequestBody switch (RequestType) {
    case InitialRequest: InitialRequestBody initial_request_body;
    case RecoveryRequest: RecoveryRequestBody recovery_request_body;
};
```

with the following definitions

```
typedef short RequestType;
const short InitialRequest = 0;
const short RecoveryRequest = 1;

struct InitialRequestBody {
    MobileTeriminal::TerminalId terminal_id;
    MobileTerminal::HomeLocationAgent home_location_agent_reference;
    unsigned long time_to_live_request;
};

struct RecoveryRequestBody {
    MobileTeriminal::TerminalId terminal_id;
    MobileTerminal::HomeLocationAgent home_location_agent_reference;
    struct LastAccessBridgeInfo {
        MobileTerminal::AccessBridge access_bridge_reference;
        unsigned long time_to_live_request;
        unsigned short last_seqno_received;
    } last_access_bridge_info;
    unsigned long time_to_live_request;
};
```

Source: Terminal Bridge

Description: This message is sent by the Terminal Bridge to establish or re-establish a tunnel with an Access Bridge. The terminal_id and home_location_agent_reference will be used by the Access Bridge to accept or deny the request and to make the location update at the Home Location Agent of the terminal

The time_to_live_request element is used to indicate the terminal's desired life expectancy (in seconds) of this tunnel association upon should it be dropped.

If the Access Bridge is already serving the Terminal Bridge (recovery of a lost tunnel), then it will reply with ACCEPT status, and does not need to do any external notification of the Home Location Agent, or others. In this case, it will change the terminal tunnel state back to active and begin delivery of any messages past those indicated by the last_access_bridge_info.last_seqno_received element.

If the Access Bridge is not already serving this Terminal (new tunnel establishment), then the access recovery procedure described in Chapter 8.4 is carried out.

Special Note: The gtp_header.seq_no and gtp_header.last_seq_no_received elements are always set to zero in this message.

Special Note: With regard to the various time_to_live parameters in all GTP messages, if the parameter is set to 0, then if sent by the terminal this indicates that the Access Bridge does not need to maintain any state or forward messages for a disconnected terminal. If sent by an Access Bridge, then the Access Bridge is indicating that it will not maintain any state and will not forward any messages for this terminal. In other word, the handoff will not be supported for this terminal.

Forwardable: No (This message cannot be encapsulated and sent via a GTPForward message)

7.2.5 EstablishTunnelReply Message

The EstablishTunnelReply message has a message body containing the CDR encoded value of

```
union EstablishTunnelReplyBody switch (ReplyType) {
    case InitialReply: InitialReplyBody initial_reply_body;
    case RecoveryReply: RecoveryReplyBody recovery_reply_body;
};
```

with the following definitions

```
typedef short ReplyType;
const short InitialReplyt = 0;
const short RecoveryReply = 1;

enum AccessStatus {
    ACCESS_ACCEPT,
    ACCESS_ACCEPT_RECOVERY,
    ACCESS_ACCEPT_HANDOFF,
    ACCESS_ACCEPT_LOCAL,
    ACCESS_REJECT_LOCATION_UPDATE_FAILURE,
    ACCESS_REJECT_ACCESS_DENIED
};

struct InitialReplyBody {
    AccessStatus status;
    MobileTerminal::AccessBridge access_bridge_reference;
    unsigned long time_to_live_reply;
};

struct RecoveryReplyBody {
    AccessStatus status;
    MobileTerminal::AccessBridge access_bridge_reference;
    struct OldAccessBridgeInfo {
        unsigned long time_to_live_reply;
        unsigned short last_seqno_received;
    } old_access_bridge_info;
    unsigned long time_to_live_reply;
};
```

Source: Access Bridge

Description: This message is sent by the Access Bridge in response to an EstablishTunnelRequest message. The status element has the following possible values:

- ACCESS_ACCEPT: in InitialReplyBody, indicates the successful establishment of a new tunnel; not used in RecoveryReplyBody

- ACCESS_ACCEPT_RECOVERY: in RecoveryReplyBody it indicates the successful re-establishment of an old tunnel to the old Access Bridge; not used in InitialReplyBody.
- ACCESS_ACCEPT_HANDOFF: in RecoveryReplyBody it indicates the successful re-establishment of an old tunnel to a new Access Bridge; not used in InitialReplyBody.
- ACCESS_ACCEPT_LOCAL: in InitialReplyBody, indicates acceptance of access without location update at HLA (so called homeless terminal).
- ACCESS_REJECT_LOCATION_UPDATE_FAILURE: The location update at the Home Location Agent failed and the Access Bridge does not support homeless terminals.
- ACCESS_REJECT_ACCESS_DENIED: Access was denied by the Access Bridge. Generic reason. May be sent if a connection bridge is out of resources and cannot accept any more Tunnels.

The ACCESS_ACCEPT_RECOVERY status indicates that the tunnel was established to the same Access Bridge as the last time a tunnel was established for this terminal. The Access Bridge will immediately set it's next GTP header gtp_header.seq_no to the next to the value of the last_access_bridge_info.last_seqno_received element obtained in the EstablishTunnelRequest message, and will re-send any GTP messages lost when the tunnel was dropped. Likewise, the Terminal must immediately set it's next GTP header gtp_header.seq_no to the next to the value of the old_access_bridge_info.last_seqno_received element of the RecoveryReplyBody, and will re-send any GTP messages lost when the tunnel was dropped.

If the tunnel was established to a new Access Bridge, then the Terminal Bridge should use the old_access_bridge_info.last_seqno_received element to indicate if any GTP messages sent by the terminal were lost by the old Access Bridge during a non-graceful handoff, and re-send them via GTPForward messages.

The time_to_live_reply element (not the old_access_bridge_info.time_to_live_reply element) is used to indicate the Access Bridge's agreed to life expectancy of this tunnel association, and will be less than or equal to the terminal's requested time to live.

Special Note: The gtp_header.seq_no and gtp_header.last_seq_no_received elements are always set to zero in this message.

Forwardable: No.

7.2.6 ReleaseTunnelRequest Message

The ReleaseTunnelRequest message has a message body containing the CDR encoded value of

```
struct ReleaseTunnelRequestBody {
    unsigned long time_to_live;
};
```

Source: Terminal Bridge and Access Bridge

Description: This message may be sent by either the Terminal Bridge or the Access Bridge to gracefully tear down a tunnel. If sent by the Terminal Bridge, the time_to_live represents the time it desires the Access Bridge to maintain connections and forward outstanding GIOP messages for this terminal. If sent by the Access Bridge then this time_to_live parameter represents the time it is willing to continue to forward GIOP messages for this terminal.

The sender of this message will send no more GTP messages directly on this tunnel. And will wait until it receives the reply before releasing the transport connectivity. The sender of this message will initiate the tear down of the transport connectivity after receipt of the reply.

Special Notes: None.

Forwardable: No.

7.2.7 ReleaseTunnelReply Message

The ReleaseTunnelRequest message has a message body containing the CDR encoded value of

```
struct ReleaseTunnelReplyBody {
    unsigned long time_to_live;
};
```

Source: Terminal Bridge and Access Bridge

Description: This message is sent by either the Terminal Bridge or the Access Bridge to acknowledge the graceful tear down of a tunnel. The time_to_live sent in this message must be less than or equal-to the time_to_live sent in the ReleaseTunnelRequest message. If sent by the terminal, the time_to_live parameter represents the time it desires the Access Bridge to maintain connections and forward outstanding GIOP messages for this terminal. If sent by the Access Bridge then this time_to_live parameter represents the time it's willing to continue to forward GIOP messages for this terminal.

The sender of this message will send no more GTP messages directly on this tunnel.

Upon sending or receiving this message, each end of the tunnel (Terminal and Access Bridge) may begin silently tearing down GIOP connections upon which there are no outstanding GIOP request messages.

The tunnel association for this terminal will be set to inactive_forwarding if the negotiated time_to_live is non-zero, and set to disconnected (and/or deleted) if time_to_live was negotiated to zero.

Special Notes: None.

Forwardable: No.

7.2.8 HandoffTunnelRequest Message

The HandoffTunnelRequest message has a message body containing the CDR encoded value of

```
struct HandoffTunnelRequestBody {
    MobileTerminal::AccessBridgeTransportAddress
        new_access_bridge_transport_address_list;
};
```

Source: Access Bridge

Description: This message is sent by the Access Bridge to the Terminal Bridge in the network initiated handoff described in Chapter 8.2.

The Terminal Bridge will use the new_access_bridge_transport_address_list to attempt to establish a tunnel to a new Access Bridge.

The sender of this message will send no more GTP messages directly on this tunnel until it received a HandoffTunnelReply message or times out after some implementation specific timeout waiting for the Terminal to establish a new Access Bridge. If it times out, then the Access Bridge may send a ReleaseTunnelRequest message to begin gracefully tearing down the tunnel. It will however continue to accept GTP messages sent by the Terminal Bridge and will hold them to either discard or process dependent upon the success or failure of the handoff.

The tunnel association for this terminal will be set to handoff_in_progress until receipt of a HandoffTunnelReply message.

Special Notes: None.

Forwardable: No.

7.2.9 HandoffTunnelReply Message

The HandoffTunnelReply message has a message body containing the CDR encoded value of

```
struct HandoffTunnelReplyBody {
    MobileTerminal::HandoffStatus status;
};
```

Source: Terminal Bridge

Description: This message is sent by the Terminal Bridge in response to HandoffTunnelRequest message.

If the Terminal Bridge successfully established a new AccessBridge, then status is set to HANDOFF_SUCCESS. The Terminal Bridge sends a ReleaseTunnelRequest message to the Access Bridge and waits for ReleaseTunnelReply message from the Access Bridge.

If the terminal does not support "make-before-break", then the Terminal Bridge should not try to establish connectivity to a new Access Bridge but to send a HandoffTunnelReply with status set to NO_MAKE_BEFORE_BREAK. The Terminal Bridge sends a ReleaseTunnelRequest message to the Access Bridge and waits for a ReleaseTunnelReply message from the Access Bridge. After that the Terminal Bridge establish a tunnel to a new Access Bridge (see Chapter 8.2.5).

If the terminal could not establish a tunnel to a new Access Bridge, then it will return a HANDOFF_FAILURE status in this message. The tunnel will then remain open and active until released by either endpoint via the ReleaseTunnelRequest / ReleaseTunnelReply sequence.

Special Notes: None.

Forwardable: No.

7.2.10 OpenConnectionRequest Message

The OpenConnectionRequest message has a message body containing the CDR encoded value of

```
struct OpenConnectionRequestBody {
    Object target_object_reference;
    unsigned long open_connection_request_id;
    unsigned long timeout;
};
```

Source: Terminal Bridge and Access Bridge

Description: This message is sent by either the Terminal Bridge or the Access Bridge to allocate a connection on the remote end of the tunnel. The open_connection_request_id will be returned in the OpenConnectionReply message. This handle is used so that the target_object_reference does not need to be returned in the OpenConnectionReply message.

The target_object_reference will be used by the receiver to connect to the target object.

The timeout is sent as an indication to the receiver of the sender's desired connection timeout. The receiver should return an error if this connection cannot be established within this period. Note that this timeout is by definition approximate because it does not take into account the transmission time of the request message.

Special Notes: The Access Bridge may use GIOP::TargetAddress type (see CORBA 2.4.2; page 15-34) instead of CORBA::Object type and encode it as an encapsulation.

Forwardable: No. New connections should be made through the current Access Bridge.

7.2.11 OpenConnectionReply Message

The OpenConnectionReply message has a message body containing the CDR encoded value of

```
struct OpenConnectionReplyBody {
    unsigned long open_connection_request_id;
    OpenConnectionStatus status;
    unsigned long connection_id; // 0xFFFFFFFF indicates failure
};

enum OpenConnectionStatus {
    OPEN_SUCCESS,
    OPEN_FAILED_UNREACHABLE_TARGET,
    OPEN_FAILED_OUT_OUT_RESOURCES,
    OPEN_FAILED_TIMEOUT,
    OPEN_FAILED_UNKNOWN_REASON
};
```

Source: Terminal Bridge and Access Bridge

Description: This message is sent by either the Terminal Bridge or the Access Bridge in response to a OpenConnectionRequest message. The open_connection_request_id element is the same as that passed in the OpenConnectionRequest message for which this is a reply. If a connection was established, the connection_id (allocated by the receiver of the OpenConnectionRequest message) is returned, and status is set to OPEN_SUCCESS.

If the connection could not be established within the requested time period, then the connection_id is set to 0xFFFFFFFF and the status element is used to relay the failure reason.

Special Notes: None.

Forwardable: Yes. This is due to the fact that outstanding OpenConnectionRequests may have been in progress during a transition to a new Access Bridge. However, if the new connection has no outstanding messages on it, then it should be closed, and a connection_id = 0xFFFFFFFF returned in this forwarded message with status = OPEN_FAILED_TIMEOUT.

7.2.12 CloseConnectionRequest Message

The OpenConnectionRequest message has a message body containing the CDR encoded value of

```
struct CloseConnectionRequestBody {
    unsigned long connection_id; // 0xFFFFFFFF denotes all connections for sender
};
```

Source: Terminal Bridge and Access Bridge

Description: This message is sent by either the Terminal Bridge or the Access Bridge to close a currently open connection. If the connection_id is set to 0xFFFFFFFF, then all connections associated with this Tunnel should be closed.

Special Notes: None.

Forwardable: Yes. This will be used by either the Terminal Bridge or an old Access Bridge to gracefully shut down open GIOP connections after a terminal has moved to a new Access Bridge.

7.2.13 CloseConnectionReply Message

The CloseConnectionReply message has a message body containing the CDR encoded value of

```
struct CloseConnectionReplyBody {
    unsigned long connection_id; // same as in request
    CloseConnectionStatus status;
};

enum CloseConnectionStatus {
    CLOSE_SUCCESS,
    CLOSE_FAILED_INVALID_CONNECTION_ID,
    CLOSE_FAILED_UNKNOWN_REASON
};
```

Source: Terminal Bridge and Access Bridge

Description: This message is sent by either the Terminal Bridge or the Access Bridge in response to a CloseConnectionRequest message. The connection_id element is the same as is sent in the CloseConnectionRequest message for which this is a reply.

Special Notes: None.

Forwardable: Yes. This will be used by either the Terminal or an old Access Bridge, to gracefully shut down open connections after a terminal as moved to a new Access Bridge.

7.2.14 ConnectionCloseIndication Message

The ConnectionCloseIndication message has a message body containing the CDR encoded value of

```
struct ConnectionCloseIndicationBody {
    unsigned long connection_id; // 0xFFFFFFFF means all connection for recepient
    ConnectionCloseReason reason;
};

enum ConnectionCloseReason {
    CLOSE_REASON_REMOTE_END_CLOSE,
    CLOSE_REASON_RESOURCE_CONSTRAINT,
    CLOSE_REASON_IDLE_CLOSED,
    CLOSE_REASON_TIME_TO_LIVE_EXPIRED,
    CLOSE_REASON_UNKNOWN_REASON
};
```

Source: Terminal Bridge and Access Bridge

Description: This message is sent by either the Terminal Bridge or the Access Bridge to alert the other end of the tunnel that a connection was asynchronously closed, (not in response to a CloseConnectionRequest message).

If all open connections for this tunnel association were closed, then the connection_id element will be set to 0xFFFFFFFF.

The reason element is used to indicate the reason for the connection closure. The element field has the following meanings:

- CLOSE_REASON_REMOTE_END_CLOSE: The remote end of the GIOP connection closed the connection.
- CLOSE_REASON_RRESOURCE_CONSTRAINT: The sender closed this connection because of a resource constraint.
- CLOSE_REASON_RIDLE_CLOSED: The sender closed the connection after an implementation dependent timeout and after all outstanding GIOP requests had been completed and the connection could be safely closed.
- CLOSE_REASON_RTIME_TO_LIVE_EXPIRED: The time_to_live for this terminal who had moved expired.

The receiver of this message should mark the indicated connections as deleted in its local data structures. If a ConnectionCloseIndication message is received for a connection_id not valid on the receiver, (probably because the receiver had already deleted it locally), then the message will be silently discarded.

Special Notes: None.

Forwardable: Yes. This will be used by either the Terminal Bridge or an old Access Bridge to indicate asynchronous connection closures after a terminal has moved to a new Access Bridge. This is used to indicate that the time_to_live has expired with the reason set to CLOSE_REASON_TIME_TO_LIVE_EXPIRED. It is also sent with the reason set to CLOSE_REASON_IDLE_CLOSED if all outstanding GIOP requests have been completed and the connection was safely closable.

7.2.15 GIOPData Message

The GIOPData message has a message body containing the CDR encoded value of

```
struct GIOPDataBody {
    unsigned long connection_id;
    unsigned long giop_message_id;
    MobileTerminal::GIOPEncapsulation giop_message;
};
```

Source: Terminal Bridge and Access Bridge

Description: This message is sent by either the Terminal Bridge or the Access Bridge and contains an encapsulated GIOP message. The giop_message_id element is assigned by the sending bridge. It is used by the receiving bridge in GIOPDataReply message to indicate to which GIOP message it is a reply. The connection_id is the receiver's connection on which this message is to be sent.

Special Notes: None.

Forwardable: Yes. This will be used by either the Terminal Bridge or an old Access Bridge to forward GIOP messages.

7.2.16 GIOPDataReply Message

The GIOPDataReply message has a message body containing the CDR encoded value of

```
struct GIOPDataReplyBody {
    unsigned long giop_message_id;
    DeliveryStatus status;
};

enum DeliveryStatus {
    DELIVERY_SUCCESS,
    DELIVERY_FAILED_INVALID_CONNECTION_ID,
    DELIVERY_FAILED_UNKNOWN_REASON
};
```

Source: Terminal Bridge and Access Bridge

Description: This message is sent by either the Terminal Bridge or the Access Bridge to acknowledge a GIOP message. If the GIOP message cannot be delivered to a connection by the receiver of the GIOPData message, then the status element must be set to the appropriate failure code.

Special Notes: None.

Forwardable: Yes. This will be used by either the Terminal Bridge or an old Access Bridge to forward replies to GIOPData messages.

7.2.17 GTPForward Message

The GTPForward message has a message body containing the CDR encoded value of

```
struct GTPForwardBody {
    MobileTerminal::AccessBridge access_bridge_reference;
        // source if sent by Access Bridge, destination if sent by Terminal Bridge
    unsigned long gtp_message_id;
    MobileTerminal::GTPEncapsulation gtp_message;
        // including GTP header
};
```

Source: Terminal Bridge and Access Bridge

Description: This message is sent by either the Terminal Bridge or the Access Bridge to forward messages to/from an old Access Bridge. The gtp_message_id is allocated by the receiver so that it can identify the GTP message in the GTPForwardReply message. This handle is used so that the access_bridge_reference does not need to be returned in the GTPForwardReply message.

If the message is sent by a Terminal, then the gtp_from terminal operation will be invoked on the access_bridge_reference to forward the message to the "old" Access Bridge; see Chapter 8.5.

If the message is sent by an Access Bridge, the access_bridge_reference will be the source of the forwarded GTP message.

Special Notes: None.

Forwardable: No. An GTPForward message cannot be encapsulated in another GTPForward message. However, Access Bridges can forward forwarded messages given to them by invoking the gtp_from_terminal and gtp_to_terminal operations.

7.2.18 GTPForwardReply Message

The GTPForwardReply message has a message body containing the CDR encoded value of

```
struct GTPForwardReplyBody {
    unsigned long gtp_message_id;
    ForwardStatus status;
};

enum ForwardStatus {
    FORWARD_SUCCESS,
    FORWARD_ERROR_ACCESS_BRIDGE_UNREACHABLE,
    FORWARD_ERROR_UNKNOWN_SENDER,
    FORWARD_UNKNOWN_FORWARD_ERROR
};
```

Source: Terminal Bridge and Access Bridge

Description: This message is sent by either the Terminal Bridge or the Access Bridge in response to a GTPForward message. The gtp_message_id element is the same as passed in the GTPForward message for which this is a reply.

If this reply message is sent by an Access Bridge, the FORWARD_SUCCESS status Indicates that the encapsulated GTP message was delivered to the old Access Bridge. Any needed GTP replies or GTP error messages will be returned in separate GTPForward messages from that Access Bridge. However, if the status is either FORWARD_ERROR_ACCESS_BRIDGE_UNREACHABLE or FORWARD_ERROR_UNKNOWN_SENDER, then the terminal should consider the tunnel on that access bridge to be lost.

If this reply message is sent by a Terminal Bridge, upon receipt of this message the Access Bridge will call back to the originating Access Bridge (by mapping gtp_message_id back to the access_bridge_reference and the gtp_message_id given through the gtp_to_terminal operation) by invoking it's gtp_from_terminal operation to deliver the status field. The FORWARD_SUCCESS status indicates that the encapsulated GTP message was accepted by the Terminal GTP engine. If the Terminal has already forgotten about or given up on the Access Bridge who sent the forwarded GTP message, then the status will be set to FORWAD_ERROR_UNKNOWN_SENDER. The Access Bridge will then consider that terminal lost, and begin tearing down it's tunnel end as if the time_to_live had expired.

The Access Bridge sends the GTPForwardReply message by invoking the gtp_from_terminal operation at the originating Access Bridge, that is the one who invoked the gtp_to_terminal. The Access Bridge replaces the value of the gtp_message_id element in the message by the value used in the gtp_to_terminal operation.

Special Notes: None.

Forwardable: No. However, Access Bridges can forward forwarded messages given to them by invoking the gtp_from_terminal and gtp_to_terminal operations.

7.2.19 Error Message

The Error message has a message body containing the CDR encoded value of

```
struct ErrorBody {
    unsigned short gtp_seq_no; // seq_no element in GTP header
    ErrorCode error_code;
};

enum ErrorCode {
    ERROR_UNKNOWN_SENDER,
    ERROR_PROTOCOL_ERROR,
    ERROR_UNKNOWN_FATAL_ERROR
};
```

Source: Terminal Bridge and Access Bridge

Description: This message is sent by either the Terminal Bridge or the Access Bridge to handle GTP protocol errors and to initiate a shutdown. The gtp_header.seq_no of the GTP message is provided for debugging purposes since this tunnel will be immediately destroyed.

Special Notes: None.

Forwardable: Yes. This will be used by either the Terminal Bridge or an old Access Bridge to cause a disorderly shutdown since the Terminal Bridge and the old Access Bridge are obviously out of sync.

7.3 TCP Tunneling

In TCP Tunneling the GTP messages are transmitted in a byte stream without any padding or message boundary marker.

The transport end-point is given as a string: <ip_address>:port_number, where <ip_address> is either a DNS name of a host or an IP address in dotted decimal notation.

7.4 UDP Tunneling

In UDP Tunneling the GTP messages are transmitted using the framing protocol, called UDP Tunneling Protocol, described below, in the payload of UDP datagrams.

The transport end-point is given as a string: <ip_address>:<port_number>, where <ip_address> is an IP address in dotted decimal notation (123.45.67.89, for example) so that the terminal does not need to do a DNS lookup.

7.4.1 UDP Tunneling Protocol

The UDP Tunneling Protocol (UTP) provides the reliability and ordered delivery of messages assumed by the GIOP Tunneling Protocol. UTP assumes that it does not get corrupted data.

UTP defines encapsulation of GTP messages. It also supports segmentation and re-assembly of GTP messages and selective acknowledgements.

UTP is chunk-based in the sense that several GTP messages can be concatenated in one UTP message. A UTP message is the payload of a UDP datagram. A UTP message contains a UTP header and one or more UTP chunks.

The UTP header is four bytes: UTP Sequence Number (unsigned short) and Number of UTP chunks (unsigned short) in the UTP message. The network byte order (that is Big-Endian) is always used to express numeric values. In UTP strings are always in 8-bit ANSI ASCII format.

The basic structure of a UTP chunk is TFLV: type-flags-length-value. However, some chunks do not have Flags, Length, and/or Value field.

- The Type field is one octet.
- If present the Flags field is one octet. It is used to denote fragmentation.
- The Length field is 0-2 octets telling the length of the Value field in the network byte order if the Value field can be of variable length.
- The Value field if present contains the payload of a UTP chunk.

The UTP chunks are:

1. ***InitialAccessRequest***: sent by the Terminal Bridge. The Flags (one octet) and Length (unsigned short) fields are present. The Value (variable length) field contains a cookie (sequence of octets) and the transport address end-point of the Terminal Bridge (string).
2. ***InitialAccessReply***: sent by the Access Bridge. The Flags (one octet) and Length (unsigned short) fields are present. The Value (variable length) field contains a cookie (sequence of octets) and the transport address end-point of the Access Bridge (string).
3. ***Pause***: sent by the Terminal or Access Bridge. No Flags, Length, and Value field. The receiving bridge should interpret this chunk so that the sending bridge will silently discard all UTP messages until it receives the Resume chunk.
4. ***Resume***: sent by the Terminal or Access Bridge. No Flags, Length, and Value field. The receiving bridge should interpret this chunk so that the sending bridge will starts to accept the UTP chunks again.
5. ***Acknowledge***: sent by the Terminal or Access Bridge. No Flag Field. The Length (one octet) tells the number of entries in the Value field. The actual length of the Value field in octets is the content of the Length field multiplied by two. The first unsigned short tells the highest Sequence Number of UTP messages received in order. The rest unsigned shorts tell which other UTP messages has been received.
6. ***GTPData***: sent by the Terminal or Access Bridge. Flags (one octet) indicate fragmentation. The Length field (unsigned short) tells the length of the Value field.

7.4.2 Fragmentation

The two rightmost bits of the Flags field are used to denote fragmentation of the Value field:

- 0x00: middle segment

- 0x01: first segment

- 0x02: last segment

- 0x03: unfragmented chunk

7.4.3 InitialAccessRequest

The chunk Type is 0x01. The Flags field (one octet) indicate fragmentation. The Length field is two octets indicating the length of the Value field as an unsigned short.

The Value field contains CDR encoded value of

```
struct InitialAccessRequestChunk {
    sequence<octet> cookie;
    string terminal_bridge_udp_address;
};
```

where cookie is some bit-pattern selected by the Terminal Bridge and terminal_bridge_udp_address is a string containing the IP address (in dotted decimal notation) of the terminal and the UDP port number to which the Access Bridge shall send the UTP messages (“123.45.67.89:9876”, for example).

The InitialAccessRequest chunk can only be sent by the Terminal Bridge.

7.4.4 InitialAccessReply

The chunk Type is 0x02. The Flags field (one octet) indicate fragmentation. The Length field is two octets indicating the length of the Value field as an unsigned short.

The Value field contains CDR encoded value of

```
struct InitialAccessReplyChunk {
    sequence<octet> cookie;
    string access_bridge_udp_address;
}
```

where cookie is the bit-pattern received in the InitialAccessRequest from the Terminal Bridge and access_bridge_udp_address is a string containing the IP address (in dotted decimal notation) of the Access Bridge and the UDP port number to which the Terminal Bridge shall send the UTP messages.

The InitialAccessReply chunk can only be sent by the Access Bridge.

7.4.5 Pause

The chunk Type is 0x03. The chunk does not have other field.

The receiving bridge should interpret this chunk so that the sending bridge will silently discard all UTP messages until it sends the Resume chunk.

Both Access and Terminal Bridge can use this chunk.

7.4.6 Resume

The chunk Type is 0x04. The chunk does not have other field.

The receiving bridge should interpret this chunk so that the sending bridge will accept UTP messages again.

Both Access and Terminal Bridge can use this chunk.

7.4.7 Acknowledge

The chunk Type is 0x05. The chunk does not have the Flags field. The Length (one octet) tells the number of entries in the Value field. The actual length of the Value field in octets is the content of the Length field multiplied by two.

The first unsigned short in the Value field tells the highest Sequence Number of UTP messages received in order. The rest unsigned shorts tell which other UTP messages has been received.

Both Access and Terminal Bridge can use this chunk.

7.4.8 GTPData

The chunk Type is 0x06. The Flags field (one octet) indicate fragmentation. The Length field (unsigned short) tells the length of the Value field.

The Value field contains a GTP message or a part of it.

7.5 WAP Tunneling

The WAP Tunneling Protocol (WAPTP) uses the Wireless Application Protocol (WAP) to transmit GTP messages between Terminal and Access Bridge.

The main design principle in WAPTP has been simplicity of the implementation. It is assumed that WAPTP will be used in small embedded devices with limited capabilities.

WAPTP ensures that the assumptions stated by GTP are no violated, specifically that no corrupted data is delivered and that the order of GTP messages is preserved.

7.5.1 Wireless Datagram Protocol

WAPTP uses the Wireless Datagram Protocol (WDP) [WDP] of the WAP specification. It operates above the data capable bearer services supported by multiple network types. WDP specification describes reference models for wide variety of networks.

WDP provides a service similar to UDP, such as transmission of unreliable datagrams and use of port numbers to identify multiple applications in one transport address.

> *"WDP supports several simultaneous communication instances from a higher layer over a single underlying WDP bearer service. The port number identifies the higher layer entity above WDP."* [WDP, 5.2]

> *"The services offered by WDP include application addressing by port numbers, optional segmentation and reassembly and optional error detection. The services allow for applications to operate transparently over different available bearer services."* [WDP, 5.1]

If the used bearer does not provide segmentation and reassembly (SAR), then it is the responsibility of the WDP implementation to do it.

> *"If the underlying bearer does not provide Segmentation and Reassembly the feature is implemented by the WDP provider in a bearer dependent way."* [WDP, 7.1]

The maximum size of datagram is bearer dependent. It is assumed that the GTP implementation does not attempt to send GTP messages that are larger than the maximum datagram size for given bearer (this implies that the ORB also knows this limitation and fragments GIOP messages accordingly).

WDP ensures the correct order of datagram segments, but not the order of datagrams themselves.

7.5.2 WAP Tunneling Protocol

In WAPTP GTP messages are transmitted in Invoke PDUs of WAP WDP, one GTP message in one WDP datagram.

WDP datagrams are not guaranteed to preserve order, so WAPTP MUST delay the delivery of GTP messages that have higher sequence number than expected.

7.5.3 WAPTP address types

The WDP supports several address types including IP addresses (both IPv4 and IPv6), MSISD (a telephone number) in various flavors (IS_637, ANSI_136, GSM, CDMA, iDEN, FLEX, TETRA), GSM_Service_Code, TETRA_ISI, and Mobitex MAN. The WDP transport address end-points are given as

```
struct WDPAddressFormat {
    octet wdp_version;// mostly 0x00, depends on bearer; see [WDP]
    octet wap_assigned_number;// identifies network, bearer, address
                              // type combination; see [WDP, Appendix C]
    unsigned short wap_port;// Port number
    string address;
};
```

The most usual address types are IP address and telephone number (MSISDN). An IP address must be in the decimal dotted notation, e.g. 123.1.2.23, so that the terminal does not need to make a DNS lookup. All possible stringified formats of telephone numbers are specified in [GFD].

Handoff and Access Recovery 8

Generally, a handoff consists of three distinct phases: the information gathering phase, the decision phase, and the execution phase. Bridge handoff, that is the handoff which is visible on the ORB level, is a part of the execution phase in cases where the mobile terminal moves from one Access Bridge to another.

The handoff support is an optional feature of this specification. The version of the GIOP Tunneling Protocol identifies whether (version 2.0) or not (version 1.0) handoff support is available.

There are two different cases of handoff: the backward handoff and the forward handoff (access recovery). The first one is the normal case whereas the second one is performed in order to re-establish connectivity after a sudden loss. In the following we use the term **handoff** to mean the backward handoff and the term **access recovery** to mean the forward handoff.

The handoff may be **network initiated** or **terminal initiated.** The access recovery is always terminal initiated.

8.1 Initiation

The AccessBridge interface contains the start_handoff operation, which is called by an external handoff control application to initiate the handoff procedure. In the **MobileTerimal** module there is also the HandoffCallback interface that contains the report_handoff_status operation, which is used by the Access Bridge to report the outcome status of handoff to the external handoff control application.

```
module MobileTerminal {

    ...

    interface HandoffCallback {
        void report_handoff_status (
            in HandoffStatus status
        );
    };

    ...

};
```

Parameters

status	outcome status of handoff procedure

```
module MobileTerminal {

    ...

    interface AccessBridge {

        ...

        void start_handoff(
            in TerminalId terminal_id,
            in AccessBridge new_access_bridge,
            in HandoffCallback handoff_callback_target
        );
        ...
    };

    ...

};
```

Parameters

terminal_id	identifies the terminal to be moved to a new Access Bridge.
new_access_bridge	reference to the new Access Bridge
handoff_callback_target	object to which the status of handoff will be reported.

8.2 Network Initiated Handoff

The network initiated handoff starts when an external application invokes the start_handoff operation in the Access Bridge currently serving the terminal. In the description below this Access Bridge is referred to as the old Access Bridge. The Access Bridge to which the terminal moves is referred to as the new Access Bridge.

The handoff procedure assumes that the terminal can establish connectivity to the new Access Bridge before releasing the connectivity to the old Access Bridge. If this cannot be done, then the alternative procedure that is described in Chapter 8.2.5 must be used.

8.2.1 Old Access Bridge

1. The old Access Bridge gets involved when the start_handoff operation is invoked on it.

2. The old Access Bridge invokes the transport_address_request operation in the new Access Bridge, which returns a list of transport addresses of the new Access Bridge and a Boolean value indicating whether or not the new Access Bridge accepts the terminal.

3. If the terminal is not accepted, then the old Access Bridge only reports the HANDOFF_FAILURE status by invoking the report_handoff_status operation at the handoff_callback_target and the handoff procedure is (unsuccessfully) completed. The old Access Bridge continues to serve the Terminal Bridge as the current Access Bridge.

4. If the terminal was accepted by the new Access Bridge, then the old Access Bridge sends the **HandoffTunnelRequest** message to the Terminal Bridge.

5. The following two steps (6 and 7) can take place in any order.

6. When the old Access Bridge gets the **HandoffTunnelReply** message from the Terminal Bridge, then
   - if the status indicates a failure in handoff, then the old Access Bridge reports the HANDOFF_FAILURE status by invoking the report_handoff_status operation at the handoff_callback_target and the handoff procedure is (unsuccessfully) completed. The old Access Bridge continues to serve the Terminal Bridge as the current Access Bridge.
   - if the status indicates a successful handoff, then the old Access Bridge waits for the **ReleaseTunnelRequest** message from the Terminal Bridge. After that it send the **ReleaseTunnelReply** message to the Terminal Bridge and releases its transport end-point to the Terminal Bridge.

7. When the new Access Bridge invokes the handoff_completed operation at the old Access Bridge, then the old Access Bridge knows that the new Access Bridge has taken the responsibility of the terminal.

8. It is assumed that the handoff status received by the old Access Bridge from the Terminal Bridge and the new Access Bridge is same. If they are not the same, then the old Access Bridge takes implementation depended actions to recover this error situation.

9. The old Access Bridge notifies all other Access Bridges interested in movements of the terminal (see Chapter 8.6).

10. If the old Access Bridge supports Mobility Event Notifications, it generates a notification of a departing terminal.

11. The old Access Bridge reports the handoff status by invoking the report_handoff_status operation at the handoff_callback_target.

8.2.2 New Access Bridge

1. The new Access Bridge gets involved when the old Access Bridge invokes the transport_address_request operation at the new Access Bridge. If the new Access Bridge does not accept the terminal, then nothing need to be done. The new Access Bridge should take the invocation of the transport_address_request operation only as a hint of a forthcoming handoff because the Terminal Bridge may use the access recovery procedure instead of the handoff procedure; see Chapter 8.2.5.

2. The new Access Bridge gets the **EstablishTunnelRequest** message from the Terminal Bridge.

3. The new Access Bridge invokes the update_location operation at the Home Location Agent.

4. The new Access Bridge sends the **EstablishTunnelReply** message to the Terminal Bridge.

5. The new Access Bridge invokes the handoff_completed operation at the old Access Bridge.

6. If the location update failed, then the new Access Bridge frees its transport end-point to the Terminal Bridge.

7. If the location update was successful and the new Access Bridge supports Mobility Event Notifications, it generates a notification of an arriving terminal.

8.2.3 Terminal Bridge

1. The Terminal Bridge gets involved when it receives the **HandoffTunnelRequest** message from the old Access Bridge.

2. The Terminal Bridge establishes transport connectivity to the new Access Bridge. If this fails, then the Terminal Bridge sends the **HandoffTunnelReply** message to the old Access Bridge that indicates a handoff failure, and the handoff procedure is (unsuccessfully) completed. The Terminal Bridge continues to use the GIOP Tunnel to the old Access Bridge.

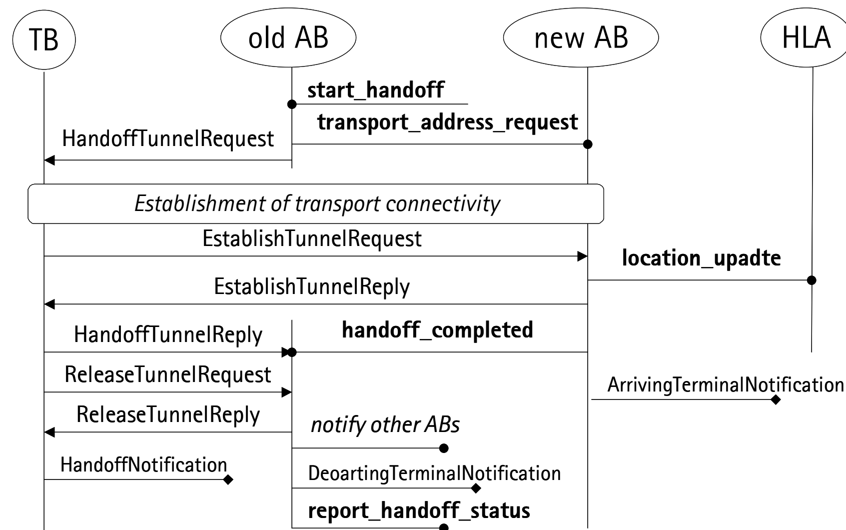
3. The Terminal Bridge sends the **EstablishTunnelRequest** message to the new Access Bridge,

4. The Terminal Bridge waits for the **EstablishTunnelReply** message from the new Access Bridge.

5. The Terminal Bridge sends the **HandoffTunnelReply** message to the old Access Bridge.

6. If the request of tunnel establishment was rejected, then the Terminal Bridge continues to use the tunnel to the old Access Bridge.
7. If the tunnel to the new Access Bridge was granted, then the Terminal Bridge sends the **ReleaseTunnelRequest** message to the old Access Bridge. After receiving the **ReleaseTunnelReply** message from the old Access Bridge, the Terminal Bridge can release its transport end-point to the old Access Bridge.
8. If the Terminal Bridge supports Mobility Event Notifications, it generates a notification of handoff.

8.2.4 Message Sequence Chart



8.2.5 Alternative Handoff Procedure.

If the terminal cannot have simultaneous transport connectivity to the old and new Access Bridge, then the following procedure is used by the Terminal Bridge.

1. The Terminal Bridge gets involved when it receives the **HandoffTunnelRequest** message from the old Access Bridge.
2. The Terminal Bridge sends the **HandoffTunnelReply** message to the old Access Bridge in which the handoff status NO_MAKE_BEFORE_BREAK.
3. The Terminal Bridge sends the **ReleaseTunnelRequest** message to the old Access Bridge and waits for the **ReleaseTunnelReply** from the old Access Bridge.
4. The Terminal Bridge releases its transport end-point to the old Access Bridge.
5. The Terminal Bridge establish GIOP Tunnel to the new Access Bridge using the access recovery procedure described in Chapter 8.4.

The old Access Bridge sees from the handoff status of NO_MAKE_BEFORE_BREAK that the terminal will use the access recovery procedure instead of the handoff procedure. The new Access Bridge sees this alternative handoff procedure as usual access recovery procedure.

8.2.6 IDL

```
module MobileTerminal {

    ...

    interface AccessBridge {

        ...

        void transport_address_request(
                // Called by the old Access Bridge at the new Access Bridge
            in TerminalId terminal_id,
            out AccessBridgeTransportAddressList new_access_bridge_addresses,
            out boolean terminal_accepted
        );

        ...

    };

    ...

};
```

Parameters

terminal_id	identification of terminal that will move
new_access_bridge_addresses	list of transport addresses that the terminal can contact in order to establish transport connectivity
terminal_accepted	FALSE, if the called Access Bridge does not accept the terminal.

```
module MobileTerminal {

    ...

    interface AccessBridge {

        ...

        void handoff_completed(
                // called by the new Access Bridge at the old Access Bridge
            in TerminalId terminal_id,
            in HandoffStatus status
        ) ;

        ...

    };

    ...

};
```

Parameters

terminal_id	identifies the terminal
status	status of handoff

8.3 Terminal Initiated Handoff

The terminal initiated handoff procedure requires that the terminal can establish connectivity to the new Access Bridge before releasing the connectivity to the old Access Bridge. If this cannot be done, then the terminal initiated handoff must be done using the access recovery mechanism: The Terminal Bridge closes connectivity to the old Access Bridge and then carries out the access recovery to the new Access Bridge.

Below we describe action taken by the Terminal Bridge and by the new and old Access Bridges.

8.3.1 Terminal Bridge

1. The Terminal Bridge establishes transport connectivity to the new Access Bridge.
2. The Terminal Bridge sends the **EstablishTunnelRequest** message to the new Access Bridge.
3. The Terminal Bridge waits for the **EstablishTunnelReply** message from the new Access Bridge.

4. If the tunnel establishment was rejected, then the Terminal Bridge releases its transport end-point to the new Access Bridge and the handoff procedure is (unsuccessfully) completed. The Terminal Bridge continues to use the GIOP Tunnel to the old Access Bridge.

5. The Terminal Bridge sends the **ReleaseTunnelReuqest** message to the old Access Bridge.

6. After receiving the **ReleaseTunnelReply** message from the old Access Bridge, the Terminal Bridge can release its transport end-point to the old Access Bridge.

7. If the Terminal Bridge supports Mobility Event Notifications, it generates a notification of handoff.

8.3.2 New Access Bridge

1. The new Access Bridge gets involved when it receives the **EstablishTunnelRequest** message from the Terminal Bridge.

2. The new Access Bridge invokes the location_update_operation at the Home Location Agent.

3. If the location update failed, then the new Access Bridge sends the **EstablishTunnelReply** message to the Terminal Bridge and releases its transport end-point to the Terminal Bridge and the handoff procedure is (unsuccessfully) completed.

4. The new Access Bridge invokes the handoff_in_progress operation at the old Access Bridge.

5. The new Access Bridge sends the **EstablishTunnelReply** message to the Terminal Bridge.

6. If the new Access Bridge supports Mobility Event Notifications, it generates a notification of an arriving terminal.

8.3.3 Old Access Bridge

1. The old Access Bridge gets involved, when the new Access Bridges invokes the handoff_in_progress operation at the old Access Bridge.

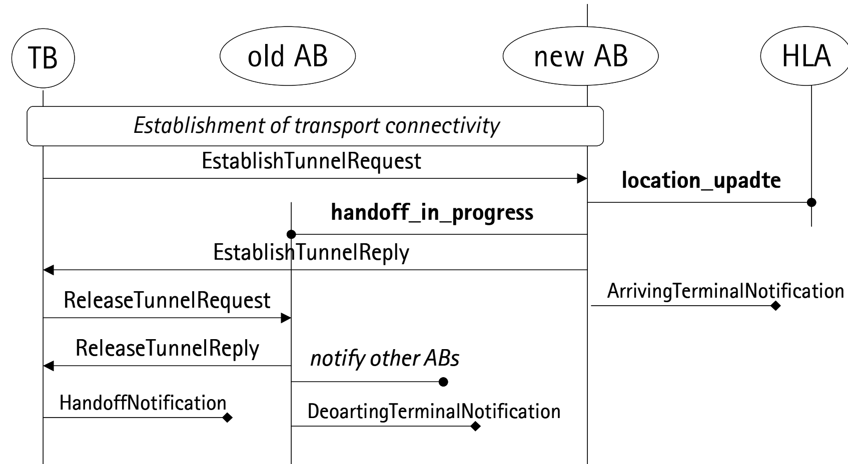
2. The old Access Bridge waits for the **ReleaseTunnelRequest** message from the Terminal Bridge.

3. After sending the **ReleaseTunnelReply** message to the Terminal Bridge, the old Access Bridge can release its transport end-point to the Terminal Bridge.

4. The old Access Bridge notifies all other Access Bridges interested in movements of the terminal (see Chapter 8.6).

5. If the old Access Bridge supports Mobility Event Notifications, it generates a notification of a departing terminal.

8.3.4 Message Sequence Chart



8.3.5 IDL

```
module MobileTerminal {

    ...

    interface AccessBridge {

        ...

        void handoff_in_progress (
                // called by the old Access Bridge in the new Access Bridge
            in TerminalId terminal_id,
            in AccessBridge new_access_bridge
        );

        ...

    };

    ...

};
```

Parameters

terminal_id	identifies the terminal
new_access_bridge	reference of the new access bridge

8.4 Access Recovery

When the Terminal Bridge detects that the connectivity to the Access Bridge is lost, a dropout notification is generate in the terminal domain and the Terminal Bridge starts the access recovery procedure. There are two possible successful outcomes of the access recovery procedure:

- The access is re-established to the same Access Bridge as before.
- The access is established to a new Access Bridge.

8.4.1 Recovery to the Old Access Bridge

Terminal Bridge

1. The Terminal Bridge establishes transport connectivity to an Access Bridge.
2. The Terminal Bridge sends the **EstablishTunnelRequest** message to the Access Bridge.
3. The Terminal Bridge waits for the **EstablishTunnelReply** message from the Access Bridge.
4. From the **EstablishTunnelReply** message the Terminal Bridge learns that the Access Bridge is the same as before and which is the last GTP message that the Access Bridge has received. The Terminal Bridge retransmits the lost GTP messages.
5. If the Terminal Bridge supports Mobility Event Notifications, it generates a recovery notification.

Old Access Bridge

1. The old Access Bridge receives the **EstablishTunnelRequest** from the Terminal Bridge.
2. From the **EstablishTunnelRequest** message the Access Bridge learns that the tunnel establishment is access recovery to it and which is the last GTP message that the Terminal Bridge has received.
3. The Access Bridge sends the **EstablishTunnelReply** message and retransmits the lost GTP messages.
4. If the old Access Bridge supports Mobility Event Notifications, it generates an access recovery notification only if it has generated the access dropout notification for the terminal.

8.4.2 Recovery to New Access Bridge

Terminal Bridge

1. same as in recovery to the old Access Bridge
2. same as in recovery to the old Access Bridge
3. same as in recovery to the old Access Bridge
4. From the **EstablishTunnelReply** message the Terminal Bridge learns that the Access Bridge is a new one and which is the last GTP message that the old Access Bridge has received. Another possibility is that the EstablishTunnelReply indicates location update failure, which terminates the recovery procedure.
5. If the Terminal Bridge supports the Mobility Event Notifications, then it generates a handoff notification.
6. The Terminal Bridge retransmits the GTP messages that the old Access Bridge has lost thru the new Access Bridge.

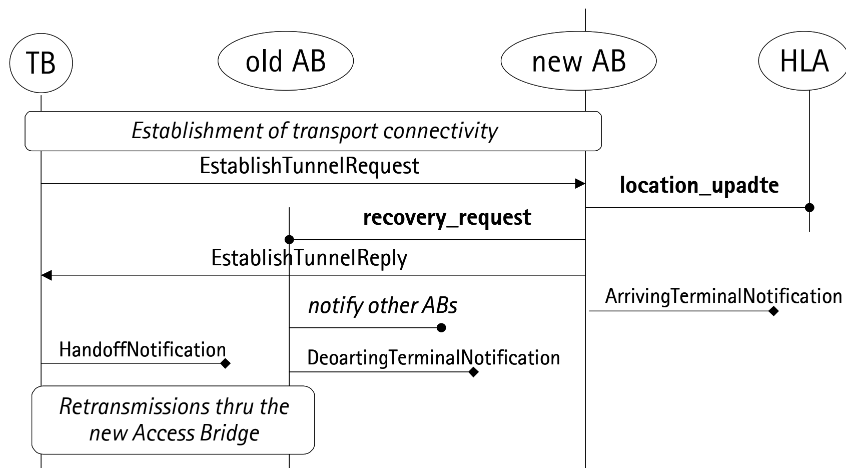
New Access Bridge

1. The new Access Bridge receives the **EstablishTunnelRequest** from the Terminal Bridge.
2. From the **EstablishTunnelRequest** message the Access Bridge learns that the tunnel establishment is access recovery to a new Access Bridge and which is the last GTP message that the Terminal Bridge has received.
3. The new Access Bridge invokes the location_update operation at the Home Location Agent.
4. If the location update fails, the new Access Bridge sends the **EstablishTunnelReply** message that indicates location update failure and completes the recovery procedure by releasing its transport end-point to the Terminal Bridge.
5. If the location update was successful, the new Access Bridge invokes the recovery_request operation at the old Access Bridge.
6. The new Access Bridge sends the **EstablishTunnelReply** message to the Terminal Bridge.
7. If the new Access Bridge supports The Mobility Event Notifications, it generates a handoff arrival notification.
8. As long as needed the new Access Bridge forwards GTP messages between the Terminal Bridge and the old Access Bridge(s).

Old Access Bridge

1. The old Access Bridge gets involved when the new Access Bridge invokes the recovery_request operation at it.
2. The old Access Bridge notifies other Access Bridges interested in movements of the terminal (see Chapter 8.6).
3. If the old Access Bridge supports Mobility Event Notifications, it generates a notification of a departing terminal.
4. The old Access Bridge retransmits the GTP messages that the Terminal Bridge has lost thru the new Access Bridge.

Message Sequence Chart



IDL

```
module MobileTerminal {

    ...

    interface AccessBridge {

        ...

        void recovery_request (
                // called by the new Access Bridge in the old Access Bridge
            in TerminalId terminal_id,
            in AccessBridge new_access_bridge,
            in unsigned short highest_gtp_seqno_received_at_terminal,
            out unsigned short highest_gtp_seqno_received_at_access_bridge
        );

        ...

    };

    ...

};
```

Parameters

terminal_id — identifies the terminal

new_access_bridge — reference to the new Access Bridge

highest_gtp_seqno_received_at_terminal — highest GTP sequence number that the Terminal Bridge has received from the Access Bridge

highest_gtp_seqno_received_at_access_bridge — highest GTP sequence number that the Access Bridge has received from the Terminal Bridge

8.5 GTP Message Forwarding

The GIOP requires that replies are sent in the same GIOP connection as the request came in. Since an Access Bridge is the GIOP connection end-point, replies must go through it even if the terminal has moved to another Access Bridge. Therefore, the AccessBridge interface contains two operations to be used in relaying GTP messages between the Terminal Bridge and an old Access Bridge thru the current Access Bridge.

When an old Access Bridge receives a GIOP message the actual destination of which is on a terminal that has moved, the old Access Bridge creates the corresponding GTP message(s) and invokes the gtp_to_terminal operation at the current Access Bridge. The old Access Bridge may use the query_location operation available in the

HomeLocationAgent interface to learn the current Access Bridge. The current Access Bridge uses the **GTPForward** message to deliver the GTP message to the Terminal Bridge.

When the Terminal Bridge wants to send a GIOP message thru an old Access, the Terminal Bridge creates the corresponding GTP message(s) and sends the **GTPForward** message(s) to the current Access Bridge. The current Access Bridge invokes the gtp_from_terminal operation at the old Access Bridge.

```
module MobileTerminal {

    ...

    interface AccessBridge {

        ...

        void gtp_to_terminal (
            in TerminalId terminal_id,
            in AccessBridge old_access_bridge,
            in unsigned long gtp_message_id,
            in GTPEncapsulation gtp_message
        ) raises (TerminalNotHere);

        ...

    };

    ...

};
```

Parameters

terminal_id	identifies the terminal
old_access_bridge	identifies the Access Bridge from which the reply comes
gtp_message_id	a handle used in a possible GTP reply message to identify to which GTP message the reply is
gtp_message	octet sequence containing the GTP message

Exceptions

TerminalNotHere	indicates that the terminal has moved from the invoked Access Bridge

```
module MobileTerminal {

    ...

    interface AccessBridge {

        ...

        void gtp_from_terminal (
            in TerminalId terminal_id,
            in unsigned long gtp_message_id,
            in GTPEncapsulation gtp_message
        ) ;

        ...

    };

    ...

};
```

Parameters

terminal_id	identifies the terminal from which the GTP message is coming
gtp_message_id	a handle to be used in a possible GTP reply message to identify to which GTP message the reply is
gtp_message	octet sequence containing the GTP message

8.6 Terminal Tracking

An Access Bridge needs to know the current Access Bridge of the terminal as long as the Terminal Bridge has open GIOP connections thru the Access Bridge. Therefore, the AccessBridge interface has two operations related to terminal tracking.

When a terminal moves from Access Bridge A to Access Bridge B, then Access Bridge A notifies the Access Bridge from which the terminal came (let it be Access Bridge C) and all other Access Bridges that have subscribed handoff notice of that terminal from the Access Bridge A (let they be Access Bridges D and E). If the Access Bridges C, D, and E still want to follow the terminal, they must subscribe the handoff notice from the Access Bridge B, that is to invoke the subscribe_handoff_notice operation at the Access Bridge B.

When the terminal moves from the Access Bridge B, the Access Bridge B notifies the Access Bridge A and those Access Bridges who has subscribed the notice. The operation is handoff_notice.

```
module MobileTerminal {

    ...

    interface AccessBridge {

        ...

        void handoff_notice (
            in TerminalId terminal_id,
            in AccessBridge new_access_bridge
        ) ;

        ...

    };

    ...

};
```

Parameters

terminal_id	identifies the terminal that has just moved
new_access_bridge	reference to Access Bridge to which the terminal has moved

```
module MobileTerminal {

    ...

    interface AccessBridge {

        ...

        void subscribe_handoff_notice (
            // called by an Access Bridge who wants to follow terminal movements
            in TerminalId terminal_id,
            in AccessBridge interested_access_bridge
        )  raises (TerminalNotHere);

        ...

    };

    ...

};
```

Parameters

terminal_id	identifies the terminal to be followed
interested_access_bridge	reference to Access Bridge that wants to receive a handoff

notice when the terminal moves again

Exceptions

TerminalNotHere	indicates that the terminal has moved from the invoked Access Bridge

Issues to be Discussed 9

9.1 Notification Service

This specification does not affect the use of Notification Service. However, the Notification Service is expected to quite popular service to be used by applications in mobile terminal. Therefore, the reference of Notification Service in the visited domain should be available through the discovery operations of the Access Bridge.

This proposal specifies Mobility Event Notifications; see Chapters 5.4 and 6.1.

9.2 Use of Interoperable Naming Service

This specification does not affect the use of Interoperable Naming Service. The reference of Naming Service in the visited domain should be available through the discovery operations of the Access Bridge.

9.3 Usage of Messaging Service

This proposal does not give any specific uses for Messaging Service, nor is use of Messaging Service in any way affected by the proposed technology. However, it is possible to reduce the amount of messages sent with Messaging Service somewhat with simple optimization.

Since the Access Bridges used by the mobile terminal are better informed on the terminal's whereabouts and connectivity, it would be beneficial to implement Messaging Router functionality in the Access Bridge. In particular, the following two cases seem to benefit from this optimization, the first one more than the second:

- Assume the mobile terminal's connectivity has dropped, so it is inaccessible. Now the last Messaging Router on the path to the terminal will keep on pinging the mobile terminal through its current Access Bridge. But if the Access Bridge is itself

the last Router, it both knows that the terminal is unavailable and will know when the terminal becomes available, so it can wait without needlessly pinging until the terminal's connectivity is restored.

- In this case, we start with the same situation as above, but now the mobile terminal recovers connectivity at another Access Bridge. Now, after handoff between the old and new Access Bridge is complete, the last Router will receive a LOCATION_FORWARD and has to invoke the new Access Bridge also, whereas if the old Access Bridge is the last Router, it can send the message to the terminal directly, which saves a couple of sent messages.

Mandatory/Optional Requirements 10

10.1 Mandatory Requirements

All products compliant to this specification must support the Mobile IOR as specified in Chapter 3.

10.1.1 Home Location Agent

A product compliant to this specification must implement all four operations specified in the HomeLocationAgent interface (see Chapter 4):

- update_location,
- query_location,
- list_initial_services, and
- resolve_initial_reference.

10.1.2 Access Bridge

A product compliant to this specification must implement GIOP Tunneling Protocol version 1.0 and either TCP, UDP or WAP Tunneling as described in Chapter 7. The product must also implement the following operations specified in the AccessBridge interface (see Chapter 5):

- list_initial_services,
- resolve_initial_reference,
- terminal_attached, and
- get_address_info.

The product must also acts as a relay between an ORB server and an ORB client fulfilling the message processing requirements of Chapter 5.3.

10.1.3 Terminal Bridge

A product compliant to this specification must implement GIOP Tunneling Protocol version 1.0 and either TCP, UDP or WAP Tunneling as described in Chapter 7.

10.2 Optional Requirements

10.2.1 Home Location Agent

None.

10.2.2 Access Bridge

An Access Bridge may provide notifications of mobility related events through the NetworkMobilityChannel (Chapter 5.3) and support handoff.

An Access Bridge implementation supporting handoff MUST implement the GIOP Tunneling Protocol version 2.0 (Chapter 7) as well as the handoff and access recovery procedures and the mechanisms to GTP messaging forwarding and terminal tracking as described in Chapter 8 for an Access Bridge in any of its possible role.

The HandoffCallback interface and the following operations specified in the AccessBridge interface (Chapter 8) must be implemented:

- start_handoff,
- transport_address_request,
- handoff_completed,
- handoff_in_progress,
- recovery_request,
- gtp_to_terminal,
- gtp_from_terminal,
- handoff_notice, and
- subscribe_handoff_notice.

10.2.3 Terminal Bridge

An Terminal Bridge may provide notifications of mobility related events through the TerminalMobilityChannel (Chapter 6.1) and support handoff.

An Terminal Bridge implementation supporting handoff MUST implement the GIOP Tunneling Protocol version 2.0 (Chapter 7) as well as the handoff and access recovery procedures as described in Chapter 8 for the Terminal Bridge.

IDL 11

The IDL of this specification is arranged in three modules: MobileTerimal, MobilityEventNotifications, and GTP (GIOP Tunneling Protocol).

11.1 Module MobileTerminal

```
//File: MobileTerminal.idl

#ifndef _MOBILE_TERMINAL_IDL_
#define _MOBILE_TERMINAL_IDL_

#include <orb.idl>
#include <IOP.idl>

#pragma prefix "omg.org"

module MobileTerminal {

    interface HomeLocationAgent;
    interface AccessBridge;

    typedef sequence<octet>  TerminalId;
    typedef sequence<octet>  GIOPEncapsulation;
    typedef sequence<octet>  GTPEncapsulation;

    struct Version {
        octet major;
        octet minor;
    };

    struct ProfileBody {
        Version                                  mior_version;
        octet                                    reserved;
        TerminalId                               terminal_id;
        sequence<octet>                          terminal_object_key;
        sequence<IOP::TaggedComponent>           components;
    };

    struct HomeLocationInfo {
        HomeLocationAgent  agent;
    };

    struct MobileObjectKey {
        Version            mior_version;
        octet              reserved;
        TerminalId         terminal_id;
        sequence<octet>    terminal_object_key;
    };

    enum HandoffStatus {
        HANDOFF_SUCCESS,
        HANDOFF_FAILURE,
        NO_MAKE_BEFORE_BREAK
    };
```

```
const short  TCP_TUNNELING = 0;
const short  UDP_TUNNELING = 1;
const short  WAP_TUNNELING = 2;

struct GTPInfo {
    short    protocol_id; // negative values are reserved for internal use
    Version  gtp_version;
};

struct AccessBridgeTransportAddress {
    GTPInfo                tunneling_protocol;
    sequence<octet>        transport_address;
};

typedef sequence<AccessBridgeTransportAddress>
    AccessBridgeTransportAddressList;

typedef string ObjectId; // same as CORBA::ORB::ObjectId
typedef sequence<ObjectId> ObjectIdList;
                    // same as CORBA::ORB::ObjectIdList

exception IllegalTargetBridge {};
exception TerminalNotHere {};
exception UnknownTerminalId {};
exception UnknownTerminalLocation {};
exception InvalidName{}; // same asCORBA::ORB::InvalidNam

interface HomeLocationAgent {

    void update_location (
        in TerminalId          terminal_id,
        in AccessBridge        new_access_bridge
    ) raises (UnknownTerminalId, IllegalTargetBridge);

    void query_location (
        in  TerminalId             terminal_id,
        out AccessBridge           current_access_bridge
    ) raises (UnknownTerminalId, UnknownTerminalLocation);

    CORBA::ORB::ObjectIdList list_initial_services ();

    Object resolve_initial_references (
        in CORBA::ORB::ObjectId  identifier
    ) raises (CORBA::ORB::InvalidName);

};

interface HandoffCallback {

    void report_handoff_status (
        in HandoffStatus  status
```

```
        );

    };

    interface AccessBridge {

        CORBA::ORB::ObjectIdList list_initial_services ();

        Object resolve_initial_references (
            in CORBA::ORB::ObjectId  identifier
        ) raises (CORBA::ORB::InvalidName);

        boolean terminal_attached (
            in TerminalId  terminal_id
        );

        void get_address_info (
            out AccessBridgeTransportAddressList  transport_address_list
        );

        void start_handoff (
            in TerminalId           terminal_id,
            in AccessBridge         new_access_bridge,
            in HandoffCallback      handoff_callback_target
        );

        void transport_address_request (
            in  TerminalId                              terminal_id,
            out AccessBridgeTransportAddressList
new_access_bridge_addresses,
            out boolean                                 terminal_accepted
        );

        void handoff_completed (
            in TerminalId       terminal_id,
            in HandoffStatus    status
        );

        void handoff_in_progress (
            in TerminalI        terminal_id,
            in AccessBridge     new_access_bridge
        );

        void recovery_request (
            in  TerminalId          terminal_id,
            in  AccessBridge        new_access_bridge,
            in  unsigned short      highest_gtp_seqno_received_at_terminal,
            out unsigned short      highest_gtp_seqno_received_at_access_bridge
        );

        void gtp_to_terminal (
```

```
        in TerminalId             terminal_id,
        in AccessBridge           old_access_bridge,
        in unsigned long          gtp_message_id,
        in GTPEncapsulation       gtp_message
    ) raises (TerminalNotHere);

    void gtp_from_terminal (
        in TerminalId             terminal_id,
        in unsigned long          gtp_message_id,
        in GTPEncapsulation       gtp_message
    );

    void handoff_notice (
        in TerminalId         terminal_id,
        in AccessBridge       new_access_bridge
    );

    void subscribe_handoff_notice (
        in TerminalId         terminal_id,
        in AccessBridge       interested_access_bridge
    ) raises (TerminalNotHere);

  };

};

#endif
```

11.2 Module MobilityEventNotification

```
//File: MobileTerminalNotification.idl

#ifndef _MOBILE_TERMINAL_NOTIFICATION_IDL_
#define _MOBILE_TERMINAL_NOTIFICATION_IDL_

#include <orb.idl>
#include <IOP.idl>

#include "MobileTerminal.idl"

#pragma prefix "omg.org"

module MobileTerminalNotification {

    struct HandoffDepartureEvent {
        MobileTerminal::TerminalId          terminal_id;
        MobileTerminal::AccessBridge        new_access_bridge;
    };

    struct HandoffArrivalEvent {
```

```
        MobileTerminal::TerminalId          terminal_id;
        MobileTerminal::AccessBridge        old_access_bridge;
    };

    struct AccessDropoutEvent {
        MobileTerminal::TerminalId  terminal_id;
    };

    struct AccessRecoveryEvent {
        MobileTerminal::TerminalId  terminal_id;
    };

    struct TerminalHandoffEvent {
        MobileTerminal::AccessBridge  new_access_bridge;
    };

    struct TerminalDropoutEvent {
        MobileTerminal::TerminalId  terminal_id;
    };

    struct TerminalRecoveryEvent {
        MobileTerminal::TerminalId  terminal_id;
    };

};

#endif
```

11.3 Moduke GTP GIOP Tunneling Protocol

```
//File: GTP.idl

#ifndef _GTP_IDL_
#define _GTP_IDL_

#include "MobileTerminal.idl"

#pragma prefix "omg.org"

module GTP {

    struct GTPHeader {
        unsigned short      seq_no;
        unsigned short      last_seq_no_received;
        octet               gtp_msg_type;
        octet               flags;
        unsigned short      content_length;
    };

    typedef short  RequestType;
```

```
const short  INITIAL_REQUEST = 0;
const short  RECOVERY_REQUEST = 1;

struct InitialRequestBody {
      MobileTerminal::TerminalId                  terminal_id;
      MobileTerminal::HomeLocationAgent     home_location_agent_reference;
      unsigned long                                      time_to_live_request;
};

struct RecoveryRequestBody {
      MobileTerminal::TerminalId        terminal_id;
      MobileTerminal::HomeLocationAgent  home_location_agent_reference;
      struct LastAccessBridgeInfo {
            MobileTerminal::AccessBridge  access_bridge_reference;
            unsigned long               time_to_live_request;
            unsigned short              last_seq_no_received;
      } last_access_bridge_info;
      unsigned long  time_to_live_request;
};

union EstablishTunnelRequestBody switch (RequestType) {
      case INITIAL_REQUEST: InitialRequestBody initial_request_body;
      case RECOVERY_REQUEST: RecoveryRequestBody
recovery_request_body;
};

typedef short  ReplyType;
const short  INITIAL_REPLY = 0;
const short  RECOVERY_REPLY = 1;

enum AccessStatus {
      ACCESS_ACCEPT,
      ACCESS_ACCEPT_RECOVERY,
      ACCESS_ACCEPT_HANDOFF,
      ACCESS_ACCEPT_LOCAL,
      ACCESS_REJECT_LOCATION_UPDATE_FAILURE,
      ACCESS_REJECT_ACCESS_DENIED
};

struct InitialReplyBody {
      AccessStatus                status;
      MobileTerminal::AccessBridge  access_bridge_reference;
      unsigned long               time_to_live_reply;
};

struct RecoveryReplyBody {
      AccessStatus                status;
      MobileTerminal::AccessBridge  access_bridge_reference;
      struct OldAccessBridgeInfo {
            unsigned long   time_to_live_reply;
            unsigned short  last_seq_no_received;
```

```
        } old_access_bridge_info;
        unsigned long  time_to_live_reply;
    };

    union EstablishTunnelReplyBody switch (ReplyType) {
        case INITIAL_REPLY: InitialReplyBody initial_reply_body;
        case RECOVERY_REPLY: RecoveryReplyBody recovery_reply_body;
    };

    struct ReleaseTunnelRequestBody {
        unsigned long  time_to_live;
    };

    struct ReleaseTunnelReplyBody {
        unsigned long  time_to_live;
    };

    struct HandoffTunnelRequestBody {
        MobileTerminal::AccessBridgeTransportAddressList
new_access_bridge_transport_address_list;
    };

    struct HandoffTunnelReplyBody {
        MobileTerminal::HandoffStatus  status;
    };

    struct OpenConnectionRequestBody {
        Object                  target_object_reference;
        unsigned long           open_connection_request_id;
        unsigned long           timeout;
    };

    enum OpenConnectionStatus {
        OPEN_SUCCESS,
        OPEN_FAILED_UNREACHABLE_TARGET,
        OPEN_FAILED_OUT_OF_RESOURCES,
        OPEN_FAILED_TIMEOUT,
        OPEN_FAILED_UNKNOWN_REASON
    };

    struct OpenConnectionReplyBody {
        unsigned long               open_connection_request_id;
        OpenConnectionStatus        status;
        unsigned long               connection_id;
    };

    struct CloseConnectionRequestBody {
        unsigned long  connection_id;
    };

    enum CloseConnectionStatus {
```

```
        CLOSE_SUCCESS,
        CLOSE_FAILED_INVALID_CONNECTION_ID,
        CLOSE_FAILED_UNKNOWN_REASON
};

struct CloseConnectionReplyBody {
        unsigned long               connection_id;
        CloseConnectionStatus       status;
};

enum ConnectionCloseReason {
        CLOSE_REASON_REMOTE_END_CLOSE,
        CLOSE_REASON_RESOURCE_CONSTRAINT,
        CLOSE_REASON_IDLE_CLOSED,
        CLOSE_REASON_TIME_TO_LIVE_EXPIRED,
        CLOSE_REASON_UNKNOWN_REASON
};

struct ConnectionCloseIndicationBody {
        unsigned long               connection_id;
        ConnectionCloseReason       reason;
};

struct GIOPDataBody {
        unsigned long                          connection_id;
        unsigned long                          giop_message_id;
        MobileTerminal::GIOPEncapsulation      giop_message;
};

enum DeliveryStatus {
        DELIVERY_SUCCESS,
        DELIVERY_FAILED_INVALID_CONNECTION_ID,
        DELIVERY_FAILED_UNKNOWN_REASON
};

struct GIOPDataReplyBody {
        unsigned long   giop_message_id;
        DeliveryStatus  status;
};

struct GTPForwardBody {
        MobileTerminal::AccessBridge           access_bridge_reference;
        unsigned long                          gtp_message_id;
        MobileTerminal::GTPEncapsulation       gtp_message;
};

enum ForwardStatus {
        FORWARD_SUCCESS,
        FORWARD_ERROR_ACCESS_BRIDGE_UNREACHABLE,
        FORWARD_ERROR_UNKNOWN_SENDER,
        FORWARD_UNKNOWN_FORWARD_ERROR
```

```
    };

    struct GTPForwardReplyBody {
        unsigned long  gtp_message_id;
        ForwardStatus  status;
    };

    enum ErrorCode {
        ERROR_UNKNOWN_SENDER,
        ERROR_PROTOCOL_ERROR,
        ERROR_UNKNOWN_FATAL_ERROR
    };

    struct ErrorBody {
        unsigned short      gtp_seq_no;
        ErrorCode           error_code;
    };

};

#endif
```