



Information Technology for European Advancement

**Opening mobile platforms for the development  
of component-based applications  
(VIVIAN – ITEA 99040)**

**Deliverable Report D2**

***VIVIAN Platform Prototype***

**December 2001**

This document will be treated in strict confidentiality.  
It will be seen only by persons who have signed the Declaration of non-Disclosure  
(see <http://www.itea-office.org/button> “documents”).

*Edited by Titos Saridakis (NOKIA) in December 2001*

## Table Of Contents

1 LTP Abstract Specification .....	1
1.1 Introduction .....	1
1.2 Bluetooth L2cap protocol.....	1
1.3 L2CAP Tunneling.....	1
1.3.1 LTP Tunneling protocol.....	1
1.4 references .....	2
2 Mapping wireless CORBA to Bluetooth .....	3
2.1 Introduction .....	3
2.1.1 Problem definition.....	3
2.1.2 GIOP tunneling over L2CAP .....	3
2.1.3 Scope and assumptions .....	5
2.2 Events and Actions .....	6
2.3 DATA Packet Format .....	8
2.3.1 Maximum Payload .....	8
2.4 references .....	8
3 LTP Implementation in VIVIAN .....	10
3.1 Introduction .....	10
3.2 MIWCO .....	10
3.3 LTP .....	10
3.4 BLueZ .....	10
3.5 Software Versions Used.....	10
3.6 Adding LTP to MIWCO.....	10
3.6.1 How the Implementation Is Done.....	11
3.6.2 Implementation Notes.....	12
3.7 References .....	12

## 1 LTP ABSTRACT SPECIFICATION

### 1.1 Introduction

This document describes abstract specification of L2CAP Tunneling Protocol (LTP), which enables the GIOP tunneling over Bluetooth L2CAP protocol. Because the specification is just another tunneling protocol specification for Wireless CORBA, the document follows the format of Wireless CORBA specification [1].

### 1.2 Bluetooth L2cap protocol

L2CAP provides connection-oriented data services, a reliable channel and ordered delivery of messages using the mechanisms available at the Baseband layer. It also provides notification of disorderly connection lost. However, L2CAP have limits for packet size, so GTP message segmentation and reassembly have to be implemented on L2CAP Tunneling Protocol in order to provide a possibility to send messages of any size [2].

### 1.3 L2CAP Tunneling

In L2CAP Tunneling the GTP messages are transmitted using *L2CAP Tunneling Protocol* (LTP) in the payload of L2CAP packets.

The transport end-point is given as a string: <BD\_ADDR#PSM>, where <BD\_ADDR> is a Bluetooth device address in coloned hexadecimal notation (7F:00:00:01:05:B3, for example) and PSM is protocol/service multiplexer (10, for example).

#### 1.3.1 LTP Tunneling protocol

The L2CAP Tunneling Protocol (LTP) provides the reliability and ordered delivery of messages assumed by the GIOP Tunneling Protocol. LTP assumes that it does not get corrupted data.

LTP defines encapsulation of GTP messages. It also supports segmentation and reassembly of GTP messages.

A LTP message is the payload of a L2CAP packet. The structure of a LTP message is FLV: flags-length-value. The network byte order is always used to express numeric values.

- The Flags field is one octet. It is used to denote fragmentation (segmentation).
- The Length field is 2 octets telling the length of the Value field in the network byte order.
- The Value field contains the payload of a LTP message.

The LTP message is:

**LTP\_Data:** sent by the Terminal or Access Bridge. The Flags field (one octet) indicates fragmentation (segmentation). The Length field (2 octets) tells the length of the Value field. The Value field contains a GTP message or a part of it.

##### 1.3.1.1 Fragmentation (segmentation)

The Flags field is used to denote fragmentation of the Value field:

- 0x00: middle segment
- 0x01: first segment

- 0x02: last segment
- 0x03: unfragmented message

#### 1.4 references

- [1] Object Management Group. *Wireless Access and Terminal Mobility in CORBA Specification, Final Adopted Specification*. (OMG Document dtc/2001-06-02), June 2001, Available: <http://www.omg.org/cgi-bin/doc?dtc/2001-06-02>
- [2] Bluetooth SIG, *Specification of the Bluetooth System – Version 1.1, Volume 1 & 2*. February 2001, Available: <http://www.bluetooth.com/developer/specification/specification.asp>

## 2 MAPPING WIRELESS CORBA TO BLUETOOTH

### 2.1 Introduction

#### 2.1.1 Problem definition

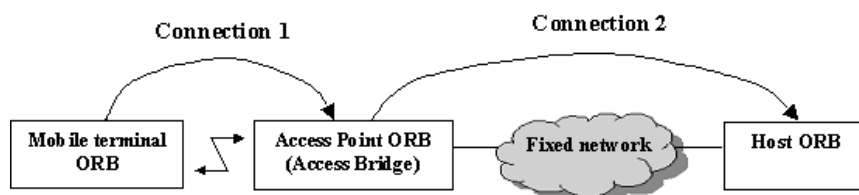
*Wireless Access and Terminal Mobility in CORBA Specification* [1] (Wireless CORBA), a Final Adopted Specification of OMG, specifies the architecture and methods how CORBA can be used over wireless links. The specification is now a standard and it can be assumed that it will be widely used approach in the future. However, the specification provides only the basis for bringing the CORBA to wireless networks letting the network specific solutions for implementers.

A new wireless technology, Bluetooth [2], has drawn attention in recent years and is expected to be the most widely used short-range communication technology between mobile devices in the future. First Bluetooth cards have already come onto the markets, and the breakthrough is expected to happen in few years.

Bringing Wireless CORBA to Bluetooth consists of several problems. The actual mapping of Wireless CORBA to Bluetooth covers only the communication between Bluetooth and Wireless CORBA architecture (GIOP Tunneling). The other problems are on the Bluetooth side, out of the scope of Wireless CORBA and current subproject.

#### 2.1.2 GIOP tunneling over L2CAP

In ordinary CORBA, there exists an end-to-end TCP/IP connection between the client and the server, and the GIOP messages are transmitted using the Internet Inter-ORB protocol (IIOP). In the Wireless CORBA approach, the GIOP connection between the client and server consists of two different connections: one between the mobile host and the access point (Access Bridge), and another between the access point and the other end of the CORBA connection that can be a host in a fixed network or a mobile host (see Figure 1-1). In order to maintain the interoperability between ordinary ORBs and mobile ORBs, the GIOP connection in the fixed network should be a normal IIOP connection over TCP/IP. Another connection over a one-hop wireless link (e.g. Bluetooth) does not necessarily have to be over TCP/IP, which is not a very convenient solution for wireless links without any extensions [3]. Rather, a more optimized wireless link-specific protocol tuned for better performance can be used.



**Figure 1-1. GIOP connections in wireless CORBA**

Wireless CORBA does not provide mapping of GIOP messages to some concrete protocol like IIOP in ordinary CORBA. Instead, it specifies a tunneling protocol, which encapsulates and decapsulates GIOP messages over the real transport protocol on the wireless media. The GIOP Tunneling Protocol (GTP) is an abstract, transport-independent protocol, which defines the message formats for establishing, releasing, and re-establishing the tunnel, but also for transmitting GIOP messages. It also defines the messages for maintaining the GIOP connection through the Access Bridge. GTP is an abstract protocol and, for this reason, needs to be mapped onto some concrete protocol. GTP is designed in such a way that a concrete tunneling

protocol is provided as an adaptation layer between GTP and the real transport layer protocol (see Figure 1-2). It is assumed that implementation of GTP is provided by the Wireless CORBA project [4]. Thereby, in order to bring wireless CORBA to some particular wireless network, the adaptation layer between GTP and some protocol of this network has to be implemented.

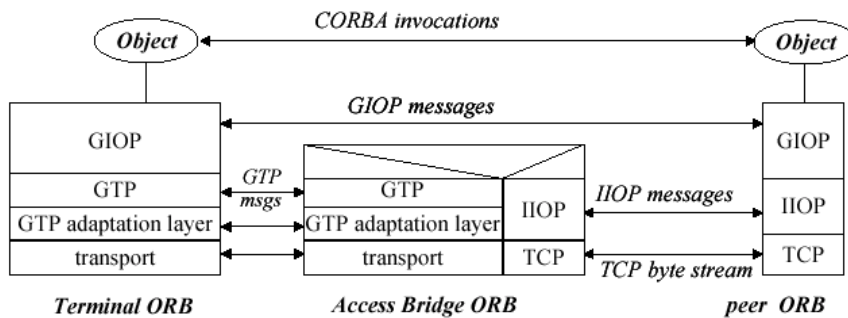
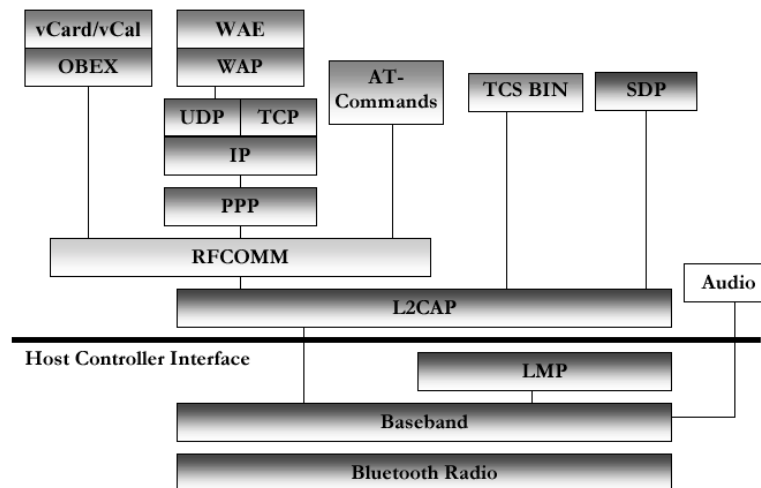


Figure 1-2. GIOP Tunneling Protocol Architecture [1]

Because the purpose of tunneling is just to deliver GIOP messages over wireless links, tunneling should be done as low as possible in the Bluetooth stack (Figure 1-3) to have minimum overhead.

Figure 1-3. Bluetooth protocol Stack [2]



On the other hand, CORBA requires from the transport protocol the following [5]:

1. The transport should be connection oriented.
2. The transport should be reliable. Bytes should be received in the order they were sent and delivery should be acknowledged.
3. The transport can be viewed as byte stream (no arbitrary message size limitations, fragmentation, or alignments are enforced.).
4. The transport should provide some reasonable notification of disorderly connection loss.

Any Bluetooth profile, or even the RFCOMM protocol, is not good for that, because they have many additional features that are not needed for tunneling causing just more overhead. The Host Controller Interface (HCI) is not sufficient either because it does not support protocol multiplexing and de-multiplexing for upper layers (running two Bluetooth applications on same machine would cause problems).

The best protocol for tunneling is L2CAP, which is right above HCI having low overhead and still providing protocol multiplexing and de-multiplexing for upper layers. L2CAP provides connection-oriented data services, a reliable channel and ordered delivery of messages using the mechanisms available at the Baseband layer. It also provides notification of disorderly connection lost. However, L2CAP have limits for packet size, so GTP message segmentation and reassembly have to be implemented in L2CAP Tunneling Protocol in order to provide a possibility to send messages of any size.

The document specifies L2CAP Tunneling Protocol (LTP), which enables the GIOP tunneling over Bluetooth L2CAP protocol.

The purpose of the LTP is to transmit the GTP messages, be a link between GTP and L2CAP protocols, and to maintain the L2CAP channel between Access Bridge and Terminal Bridge.

LTP provides the reliability and ordered delivery of messages assumed by the GIOP Tunneling Protocol. LTP defines encapsulation of GTP messages. It also supports segmentation and reassembly of GTP messages. LTP defines message formats for establishing and releasing L2CAP channel as well as for transmitting GTP messages.

#### *Segmentation and Reassembly*

CORBA requires that the transport can be viewed as byte stream (no arbitrary message size limitations, fragmentation, or alignments are enforced.). On the other hand, the data packets defined by the L2CAP are limited in size (a packet length is implementation specific: it can be up to 64 kilobytes in length, but L2CAP implementations must support only a minimum MTU size of 48 bytes; the default MTU value is 672 bytes). Large GTP messages must be segmented into multiple smaller L2CAP packets prior to their transmission over the air. Similarly, multiple received L2CAP packets may be reassembled into a single larger GTP message. The segmentation and reassembly functionality is necessary to meet CORBA requirements.

The transport end-point is given as a string: <BD\_ADDR#PSM>, where <BD\_ADDR> is a Bluetooth device address in coloned hexadecimal notation (7F:00:00:01:05:B3, for example) and PSM is protocol/service multiplexer (10, for example).

### 2.1.3 Scope and assumptions

As LTP is an adaptation layer between GTP and L2CAP, it defines how the transport is to be used and is only responsible for transmission of GTP messages over it. The Wireless CORBA specification does not specify the transport connectivity establishment process and formally it should not be carried out by LTP. But the transport connectivity should be established before the transport could be really used and that is why the description of events and actions necessary for establishing and releasing a connection was included into the document.

LTP provides connection-oriented data services, reliability, ordered delivery of messages, and notification of disorderly connection loss using the mechanisms available at the Bluetooth L2CAP and Baseband layers.

LTP was designed to be simple. It neither provides means to configure parameters of underlying L2CAP channel nor has timers by its own. Rather it relies on default settings of Bluetooth protocol stack implementation.

## 2.2 Events and Actions

Events and actions between LTP as well as upper and lower layers are defined in the following sections. Naming convention used is presented in the Figure 3-1. Events coming from above layers are called Requests and their corresponding replies are called Confirms. Events coming from below are called Indications and their replies are called Responses.

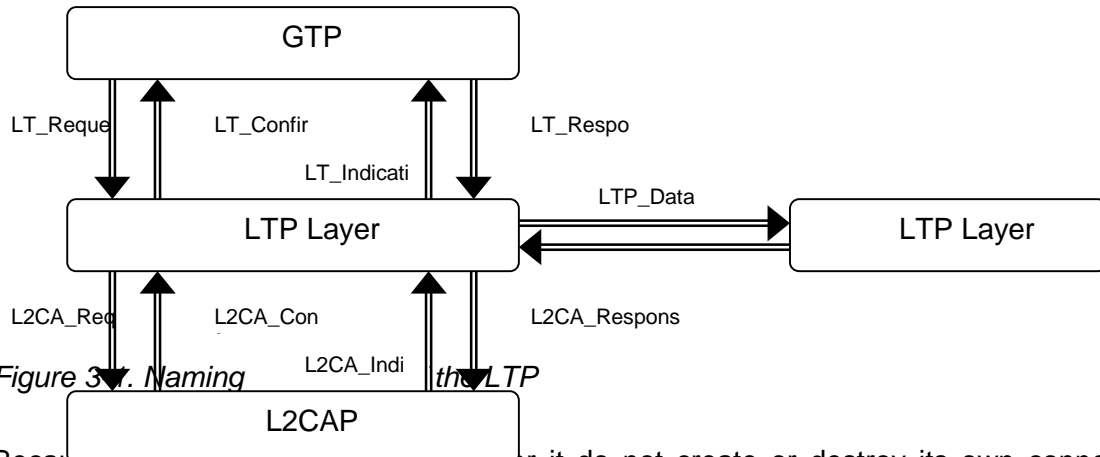


Figure 3-1. Naming the LTP

Because LTP is just an adaptation layer it do not create or destroy its own connection, and hence it do not have signal Requests and Responses. LTP translates requests and responses coming from upper layer into requests and responses required by L2CAP and vice versa. The only events and actions available between LTP layers are sending and receiving GTP data.

Events are all incoming messages to the LTP. Events are partitioned into three categories: Indications and Confirms from lower layers, Requests and Responses from higher layers, and data from peers.

Actions are caused by events. Actions are also partitioned into three categories: Confirms and Indications to higher layers, Request and Responses to lower layers, data transmission to peers.

Mapping of signalling events to corresponding actions is presented on the Table 3-1.

Table 3-1. LTP signalling event-action relationships

Event	Action
LT_EstablishConnectionReq	Send lower layer L2CA_ConnectReq message
L2CA_ConnectInd	Send upper layer LT_EstablishConnectionInd message
LT_EstablishConnectionRsp	Send lower layer L2CA_ConnectRsp message
LT_EstablishConnectionRspNeg	Send lower layer L2CA_ConnectRspNeg message
L2CA_ConnectCfm	Send upper layer LT_EstablishConnectionCfm message
L2CA_ConnectCfmNeg	Send upper layer LT_EstablishConnectionCfmNeg message
LT_ReleaseConnectionReq	Send lower layer L2CA_DisconnectReq message
L2CA_DisconnectInd	Send upper layer LT_ReleaseConnectionInd

The names of events coming from and actions directed to upper layer are exemplary. They should be in compliance with the names of upper layer protocol correspondent actions and events.

The names of events coming from and actions directed to L2CAP correspond to the names of L2CAP actions and events defined in the Bluetooth Specification [2].

A LTP implementation upper interface is determined by particular wireless CORBA implementation. LTP was developed as a new transport to MIWCO [4], and provides interface similar to other MIWCO transports. MIWCO is an implementation of the Telecom Wireless CORBA Final Adopted Specification. It is a product of the wCORBA research project at the University of Helsinki and is implemented as an extension to the Open Source MICO ORB [6].

Lower interface of LTP implementation is determined by an implementation of Bluetooth protocol stack, which in our case is BlueZ [7]. BlueZ is official Linux Bluetooth stack. It provides support for core Bluetooth layers and protocols. Originally BlueZ was developed by Qualcomm Incorporated and then became an Open Source project. BlueZ provides standard Unix socket interface to all its layers including L2CAP. But unlike TCP stream sockets, entirely meeting the CORBA requirements, BlueZ has sequenced packet sockets, which demands segmentation and reassembly to be implemented in LTP.

#### 2.2.1.1 LTP to LTP Data Events and Actions

- *LTP\_Data*

*LTP\_Data* event indicates that a GTP message has been received as a payload of a LTP data packet, while *LTP\_Data* action is a GTP message transmission. The LTP doesn't care about the contents of the message; it just handles it as raw data without any particular meaning.

### 2.3 DATA Packet Format

The LTP data packet consists of a header and a payload. The packet is sent to the L2CAP layer as raw data i.e. the L2CAP layer doesn't care the contents of the packet.

The structure of a LTP packet is represented on Figure 4-1. The network byte order is always used to express numeric values.

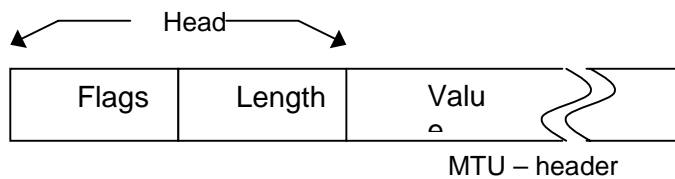


Figure 4-1. LTP packet format

Flags:

- One octet
- Describes fragmentation of the Payload field
- 0x00: middle segment
- 0x01: first segment
- 0x02: last segment
- 0x03 unfragmented message

Length:

- Two octets
- The length of the Payload field

Value:

- GTP message as a payload

#### 2.3.1 Maximum Payload

The smallest maximum payload size in one LTP message is the minimum L2CAP MTU – LTP header size. At maximum the payload size is the maximum L2CAP MTU – LTP header size. Default size is the default L2CAP MTU – LTP header size. However the real size may vary between the minimum and maximum values depending on the L2CAP implementation. As a result of these calculations we get the following limits:

- Minimum: 48 bytes – 3 bytes = 45 bytes
- Default: 672 bytes – 3 bytes = 669 bytes
- Maximum: 65535 bytes – 3 bytes = 65532 bytes

### 2.4 references

- [1] Object Management Group. *Wireless Access and Terminal Mobility in CORBA Specification, Final Adopted Specification*. (OMG Document dtc/2001-06-02), June 2001, Available: <http://www.omg.org/cgi-bin/doc?dtc/2001-06-02> [Accessed 20.12.2001].

- 
- [2] Bluetooth SIG, *Specification of the Bluetooth System – Version 1.1, Volume 1 & 2*, February 2001, Available: <http://www.bluetooth.com/developer/specification/specification.asp> [Accessed 20.12.2001].
- [3] E. Amir, H. Balakrishnan, S. Seshan, R. Katz, “Efficient TCP over Networks with Wireless Links”, May 1995.
- [4] University of Helsinki, Department of Computer Science. Research Project Wireless CORBA. Available: <http://www.cs.helsinki.fi/u/kraatika/wCORBA.html> [Accessed 20.12.2001].
- [5] Object Management Group, *The Common Object Request Broker: Architecture and Specification, V2.4.2*, February 2001, [Accessed 20.12.2001].
- [6] MICO open source ORB URL: <http://www.mico.org/> accessed 20.12.2001
- [7] BlueZ Bluetooth Protocol Stack for Linux. URL: <http://bluez.sourceforge.net>, accessed 20.12.2001.

### 3 LTP IMPLEMENTATION IN VIVIAN

#### 3.1 Introduction

The purpose of implementing LTP layer to the CORBA middleware was to show that the tunnelling of GTP messages over a L2CAP connection is possible. This implementation is a demonstrative prototype solution only and it's done using MIWCO wireless CORBA [1] and BlueZ Bluetooth protocol stack [2].

#### 3.2 MIWCO

MIWCO is an implementation of the Telecom Wireless CORBA (wCORBA) Final Adopted Specification and it is done at the University of Helsinki. MIWCO itself is extension to the open source MICO ORB [3].

Extension includes the following new parts (Wireless parts) to enable mobility.

- Home Location Agent (Forwards CORBA invocations to terminal's current access point).
- Access Bridge (Provides an access point to wired networks for mobile terminals and acts as a proxy for CORBA objects).
- Terminal Bridge (Acts as access bridge's counterparts on the terminal side of the wireless link).

#### 3.3 LTP

LTP is a GTP adaptation layer developed by Lappeenranta University of Technology and it tunnels GTP messages over Bluetooth L2CAP connection. More information about LTP can be found from its documentation.

#### 3.4 BLueZ

BlueZ is an open source Bluetooth protocol stack for the Linux operating system and it offers socket interface to the L2CAP layer.

#### 3.5 Software Versions Used

The following software versions were used when implementing the LTP.

- BlueZ version 1.2
- MIWCO version 011109
- Linux kernel 2.4.9

The testing of this implementation was done with HCIEmu versions 0.2 which emulates Bluetooth devices. HCIEmu is available from BlueZ's web page.

#### 3.6 Adding LTP to MIWCO

In order to implement the LTP a new transport to MIWCO was needed. Implementing the transport consists of the following subtasks.

- New address format
- Parser for the new address format
- New transport
- New transport server

In addition some modifications to the MIWCO were also needed. These modifications include the following.

- Identifier for the LTP tunnelling was added to the MobileTerminal.idl file. This identifier is an octet that has value of 13.
- Function `GTPAddress::make_address()` (in `wcorba/proxy.cc`) was modified so that it recognizes the identifier of LTP tunnelling.
- Support for the creation of LTP address format was added into the constructor of `AccessBridge_impl` class (in `wcorba/AccessBridge_impl.cc`).

### 3.6.1 How the Implementation Is Done

There are four new classes added in the following files:

- `orb/ltp_address.cc` - LTP address format
- `orb/ltp_addressparser.cc` - parser for the address format
- `orb/ltp_transport.cc` - actual transport class
- `orb/ltp_transporterver.cc` - transport server

The directory `include/mico` contains correspondingly named header files for the e-mentioned classes. In addition several files were modified. These files are:

- `include/mico/MobileTerminal.idl`
- `include/mico/impl.h`
- `orb/address.cc`
- `orb/Makefile`
- `wcorba/proxy.cc`
- `wcorba/AccessBridge_impl.cc`

Modifications are marked with comments in all the files except `Makefile`. The next paragraphs briefly describe added classes.

#### 3.6.1.1 LTPAddress

Implements the LTP address format. The form of this address format is "proto:XX:XX:XX:XX:XX:XX#PSM" where "XX:XX:XX:XX:XX:XX" is of type `BD_ADDR`, PSM is service multiplexer and proto is "bt-l2cap". Also in this class are methods for setting and getting the properties of a L2CAP socket.

#### 3.6.1.2 LTPAddressParser

This class parses both `BD_ADDR` and PSM out of an LTP address given as a string. The address format recognised by this parser class should have the '#' character separating the address part from PSM part. If this separator is found then is checked if the `BD_ADDR` part is 17 characters long. So the address to be recognised by this address parser should look like "XX:XX:XX:XX:XX:XX#PSM" because proto is removed before passing the rest of the address to the parser.

#### 3.6.1.3 LTPTransport

`LTPTransport` class is responsible for data transfer over L2CAP. It has functions to establish/close a socket connection and it also takes care of writing and reading from the socket. This class has an internal buffer in order to implement stream-like socket type interface to upper layers.

#### 3.6.1.4 LTPTransportServer

LTPTransportServer class is responsible for initialising and listening the socket and therefore responsible for accepting connections..

#### 3.6.2 Implementation Notes

There are some implementation specific features in this LTP prototype. These features are discussed in the following paragraphs.

##### 3.6.2.1 Reading from Socket

CORBA is based on Stream sockets, which have a feature of reading the incoming data in several parts. However, Bluetooth stack BlueZ supports only sequential socket packets, which do not support the option of reading the incoming message in parts. Instead the socket discards the rest of the message after first reading, regardless whether the message was completely read or not. To overcome this obstacle an internal buffer was added into the LTPTransport class. When there's some data coming on a socket the whole data chunk is read to this buffer no matter how many bytes were asked. At the same time a counter is set to point to the beginning of the internal buffer. Next the requested amount of bytes is copied from the internal buffer to the caller's buffer and the counter is increased. On the next read request data is copied directly from the internal buffer if the counter was not pointing to the end of it. When the counter reaches the end of the internal buffer socket is read and so on. By default the size of the internal buffer is 8192 bytes as defined in the `ltp_transport.cc` file.

##### 3.6.2.2 Header Conflict with BlueZ

If Bluetooth header file `bluetooth.h` is included in any of LTP's header files or in the common include file `impl.h` it causes a conflict that prevents the whole MIWCO compiling. To avoid that functions `sockaddr` and `getsockaddr` in the `LTPAddress` class use void pointers instead of the `sockaddr_l2` structure. Also for the same reason some of the basic BlueZ functions are duplicated in the very beginning of the `ltp_address.cc` file.

##### 3.6.2.3 Segmentation

This prototype doesn't support LTP message segmentation / reassembly as defined in the LTP specification. Support for this feature is done via BlueZ stack's socket interface. However, the maximum size of a message is limited by the size of LTPTransport's internal buffer.

##### 3.6.2.4 Debug Output

It's possible to define each function in LTP to print some debug information that was used when developing the LTP. This debug output is off by default and can be set on by defining `LTP_DEBUG`. The printing of this information is not implemented through a MIWCO debug level because it's highly unlikely that someone would need this kind of information when working with LTP. So the usage of definition was made in order to exclude some unneeded source code.

### 3.7 References

- [1] University of Helsinki, Department of Computer Science. Research Project Wireless CORBA. URL: <http://www.cs.helsinki.fi/u/kraatika/wCORBA.html>, accessed 20.12.2001
- [2] BlueZ Bluetooth Protocol Stack for Linux. URL: <http://bluez.sourceforge.net>, accessed 20.12.2001.
- [3] MICO open source ORB URL: <http://www.mico.org/>