Java Bluetooth stack
Design Description

Version 1.0

Doc. No.:

Revision History

Date	Version	Description	Author
2003-11-22	0.1	Initial Draft	Marko Đurić
2003-12-11	0.2	Minor updates	Tomislav Sečen
2003-12-12	0.5	Updated RFCOMM, OBEX, Security	Marko Đurić
2003-12-14	1.0	Final version	Marko Đurić

Table of Contents

Table of figures

1. Introduction

1.0.1. Bluetooth

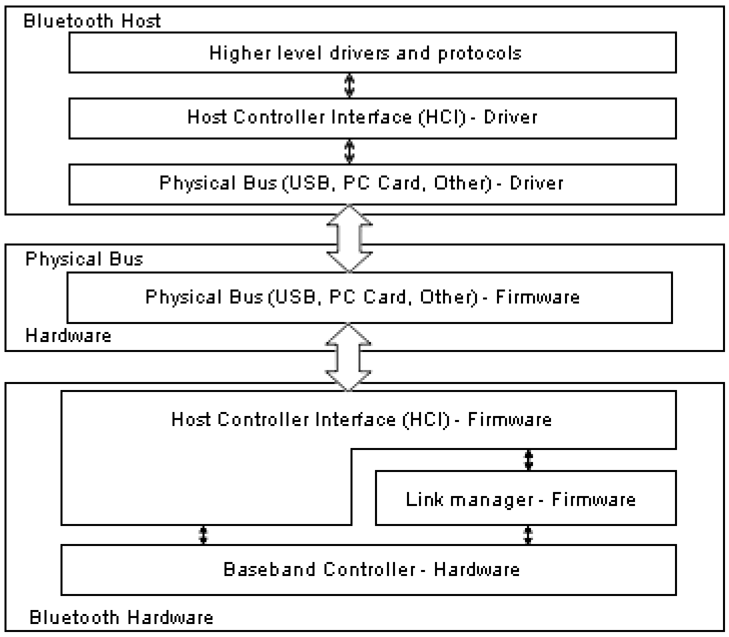
Bluetooth is wireless communication protocol. It was originally invented 1994. by Swedish phone equipment maker Ericsson for wireless communication between mobile devices on short distances (up to 100m). In 1997. some major companies (such as IBM, Intel, Nokia and Toshiba) joined with Ericsson into Bluetooth Special Interest Group (SIG) consortium for further development of Bluetooth protocol. After the release of the first Bluetooth specification (version 1.0), 3COM, Agere, Microsoft and Motorola also joined the SIG consortium.

Bluetooth operates in 2.4GHz radio band that is reserved for Industrial – Scientific – Medical (ISM) purposes. Because it communicates by radio signals, there is no need for optical visibility between devices.

Intention of Bluetooth devices is ad-hoc connectivity of small devices like mobile phones or PDA-s and for that reason the range is limited to 10m (max. of 100m in Class I devices – larger devices with better power supply (more capacity)). It is mostly used for small amount of data transfer between devices (asynchronous mode) or for speech (synchronous mode).

1.0.2. Bluetooth software stack

Hardware used for Bluetooth communication is practically useless without some drivers. It's like the computer hardware – if we have only hardware (i.e. peripherals like keyboards, mouse, printers, scanners, cameras, etc.) we can't use it because the computer doesn't "know" what to do with them. In that case we use drivers for those peripherals. In Bluetooth's "world" it is the same – Bluetooth hardware by it self is not capable of doing anything. For that purpose we need Bluetooth protocol stack.



Bluetooth Host
Higher level drivers and protocols
Host Controller Interface (HCI) - Driver
Physical Bus (USB, PC Card, Other) - Driver
Physical Bus
Physical Bus (USB, PC Card, Other) - Firmware
Hardware
Host Controller Interface (HCI) - Firmware
Link manager - Firmware
Baseband Controller - Hardware
Bluetooth Hardware

Fig. 1.0.1: Bluetooth software stack

Figure 1.0.1. shows lower levels of Bluetooth software stack. At the bottom is *radio layer* (not shown) – medium that transfers Bluetooth signals (radio signals at 2.4GHz). On top of radio layer is *baseband layer*. It is implemented in hardware as baseband controller.

To make Bluetooth communication possible, we must have *Host Controller Interface (HCI)* firmware that implements HCI commands used for passing all data from host computer (mobile phones, PDAs, personal computers, …) to Bluetooth device (Bluetooth chip).

Link manager (LM) (also implemented in firmware) is used to send and receive Link manager protocol (LMP) messages that are used for link set-up, security and control.

Physical Bus is implemented as hardware (firmware) and it is used as link between Bluetooth hardware (Bluetooth chip) and host hardware (mobile device, computer, etc.).

Upper layers are implemented as software (protocols by specifications). First of these layers is *Physical Bus* and *Host controller interface* (drivers). They are used for access to HCI data from hardware (and vice versa).

Higher-level drivers and protocols are device specific. If we only need audio communication, we don't have to use any additional layers, but if we want to make data communication, then it's important to have *Logical Link Control and Adaptation Protocol (L2CAP)*. It is the core layer of the Bluetooth protocol stack and all data must pass through this layer. Purpose of this layer is to deal with packets (Bluetooth communication is packet based communication).

L2CAP is multiplexed protocol. This mean that it can communicate with more than one upper layer (SDP, RFCOMM, HID, TCS, …).

Finally we came to high level of Bluetooth stack. Layers here are very implementation specific. The most used protocols here are RFCOMM and SDP. RFCOMM is replacement for "old" serial cable connection. With this protocol implemented, we can use Bluetooth instead of serial cable, and "don't even know that we communicate without cables". Service Discovery Protocol (SDP) is used for discovery of other Bluetooth devices and their services.

On top of SDP is Object EXchange Protocol (OBEX). This protocol is initially defined by Infrared Data Association (IrDA), and later adopted by Bluetooth SIG. This protocol is useful for file transfer.

1.0.3. Why use Java Bluetooth stack?

Bluetooth stack can be implemented on variety of platforms (J2SE, J2ME, Windows, Windows CE, Linux, etc.). Java implementation of Bluetooth stack is good idea because Java is very distributed, and same code can be used anywhere where exists Java run-time environment (nowadays, that is practically any platform – from small (mobile) devices up to enterprise server systems). Java is also very well standardized through its API specifications.

1.0.4. What is JCP? And what is JSR?

Java Community Process (JCP) is participating project for developing and revising Java technology specifications, reference implementations, and test suites. Over 500 companies and individuals are now members of JCP.

Java Specification Request (JSR) is document that is submitted to Process Management Office (PMO) that one or more JCP members send to propose the development of a new specification or revision of existing specification.

1.0.5. Why is JSR-82 important?

JSR-82 is a document that describes Bluetooth Java APIs. With standardized specification like JSR-82 it is easier to develop end-user Bluetooth applications, because lot of functions are well described in API. Target Java platform is J2ME.

1.0.6. What parts of Bluetooth specifications are covered by the JSR-82?

JSR-82 includes basic support for RFCOMM, OBEX and Service Discovery Protocols (SDP). Specification is primarily targeted at native Bluetooth protocols.

Devices that use JSR-82 API must have at least 512kB of total memory (ROM/FlashROM and RAM), Bluetooth network connection, and compliant implementation of J2ME CLDC configuration (Java 2 Micro Edition Connected Limited Device Configuration).

1.1. Purpose of this document

Purpose of this document is to give design description of Java Bluetooth stack.

1.2. Intended Audience

Product supervisor, development team and application developers that use this implementation of Bluetooth protocol stack.

1.3. Scope

Some Bluetooth protocols are already implemented in javabluetooth stack (like SDP) - they will also be covered by this document), but this document mainly addresses to OBEX and RFCOMM protocols and Bluetooth security.

1.4. Definitions and acronyms

1.4.1. Definitions

Keyword	Definitions
Java	The programming language
Bluetooth	Technology for wireless communication
Bluetooth stack	Layered Bluetooth architecture

1.4.2. Acronyms and abbreviations

Acronym or abbreviation	Definitions
HCI	Host Controller Interface
LMP	Link Manager Protocol
L2CAP	Logical Link Control and Adaptation Protocol
SDP	Service Discovery Protocol
HID	Human Interface Device protocol
TCS	Telephony Control Protocol specification
OBEX	Object EXchange protocol
IrDA	Infra-red Data Association
J2SE	Java 2 Standard Edition
J2ME	Java 2 Micro Edition
JCP	Java Community Process
JSR	Java Specification Request

PMO	Process Management Office
CLDC	Connected Limited Device Configuration
SCO	Synchronous Connection-Oriented protocol
ACL	Asynchronous ConnectionLess protocol
BCC	Bluetooth Control Center
DLC	Data Link Connection

1.5. References

B. Hopkins, *R. Anthony*, Bluetooth for Java, Apress, 2003.
Specification of the Bluetooth System, specification Vol.1, version 1.1, Bluetooth SIG, 2001.
Java APIs for Bluetooth Wireless Technology (JSR-82), specification version 1.0a, 2002.
http://wireless.java.sun.com/midp/articles/Bluetooth1
http://www.jcp.org/en/home/index
http://www.niksula.cs.hut.fi/~jiitv/bluesec.html

2. External interfaces

There are no external interfaces to other projects / groups, but since work is based on the javabluetooth stack (javabluetooth.sourceforge.net) one could consider the existing source code to be an (external) interface.

Since L2CAP protocol layer is the one on top of which RFCOMM will be built, one can consider L2CAP to be an external interface for RFCOMM, and HCI classes and methods as an interface to implementing security.

The class diagrams for the L2CAP and HCI layers as they were originally built in the javabluetooth project can be found later in this document.

3. Software architecture

3.1. System specification

Final project output needs to behave exactly as specified by the JSR-82 specification, so system specification=JSR-82.

3.2. Error handling

Since the final result of this project will in fact be a library, it will only respond to the exceptions originating from any layer in the stack and pass them (maybe as an different exception) to the user application which will be using the stack. Some internal software exceptions will be handled internally, and the operations involved retried, but these exceptions are protocol specific and they won't be discussed any further throughout this document.

User application will have to deal with following error conditions:

Error	Action
Device goes out of range during communication	Implement timeouts and retransmission
Error while initializing hardware (i.e. setting COM speed)	Fatal error, instruct user to check hardware settings
Error with security settings	Instruct user to check devices security options

Table 3.2.1: Error handling

4. Detailed software design

Bluetooth stack has layered structure, like referent ISO/OSI network model. It is composed of protocols that are specific to Bluetooth (L2CAP, SDP, etc.) and other adopted protocols (i.e. OBEX).

There are four main groups in Bluetooth stack:

- Bluetooth core protocols – Baseband, Link Manager Protocol, L2CAP and SDP
- Cable replacement protocol – RFCOMM
- Telephony control protocol – TCS Binary
- Adopted protocols – PPP, UDP/TCP/IP, OBEX, WAP

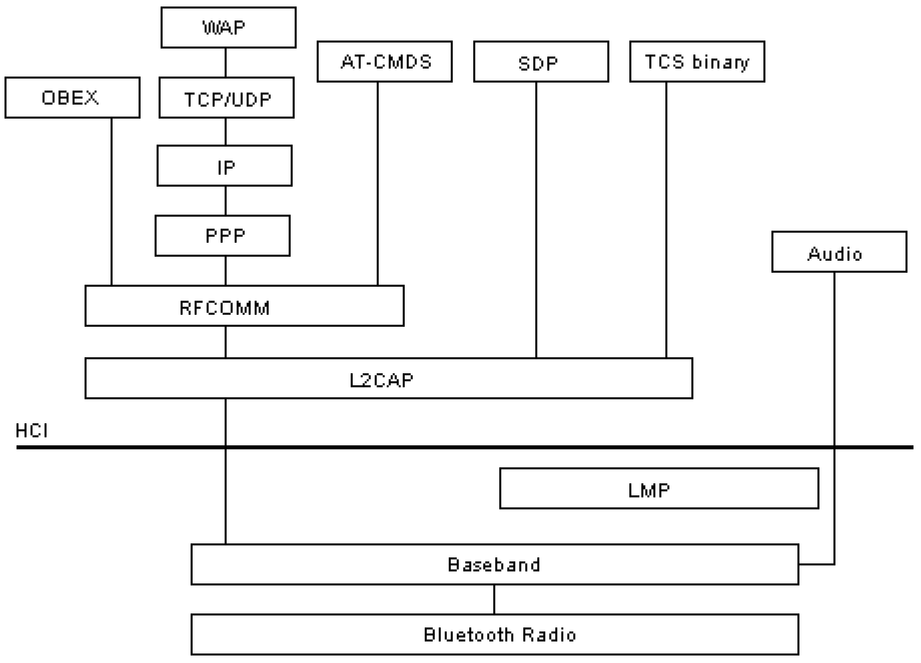


Fig. 4.0.1: Bluetooth stack

4.0.1. Bluetooth radio

Bluetooth radio is hardware for wireless Bluetooth communication. It is the lowest layer of Bluetooth stack and it is in charge of communication between two or more Bluetooth devices. Bluetooth radio signals are in frequency range of 2.4GHz up to 2.4835GHz divided in 79 channels with 1MHz spacing between channels. For added security, channels use frequency hoping (frequency is changed in pseudo-random pattern 1600 times per second).

4.0.2. Baseband layer

Baseband layer enables physical radio-frequency (RF) link between Bluetooth enabled devices that make connection. Its function is to manage Bluetooth channels (frequency hoping, time-slots, etc.) and packet transmissions.

4.0.3. Link Manager Protocol (LMP)

LMP is responsible for link set-up, managing security (authentication, encryption) between Bluetooth devices and control of Bluetooth devices. Link manager messages have higher priority than user data (because the control is more important than user data – if there is no control, there is no data either).

Transport layer provides the ability to transfer data without intimate knowledge of data. There are several types of transport layers – USB, RS232, UART, HCI, RFCOMM, TCP/IP, etc.

4.0.4. Host controller Interface (HCI)

HCI provides a uniform interface method of accessing the Bluetooth hardware capabilities. HCI commands are implemented in HCI firmware by accessing baseband commands, link manager commands, hardware status registers, control registers, and event registers. In come cases there is also Host Controller Transport Layer as an intermediate layer for communication between HCI firmware and host HCI driver.

HCI UML class diagram:

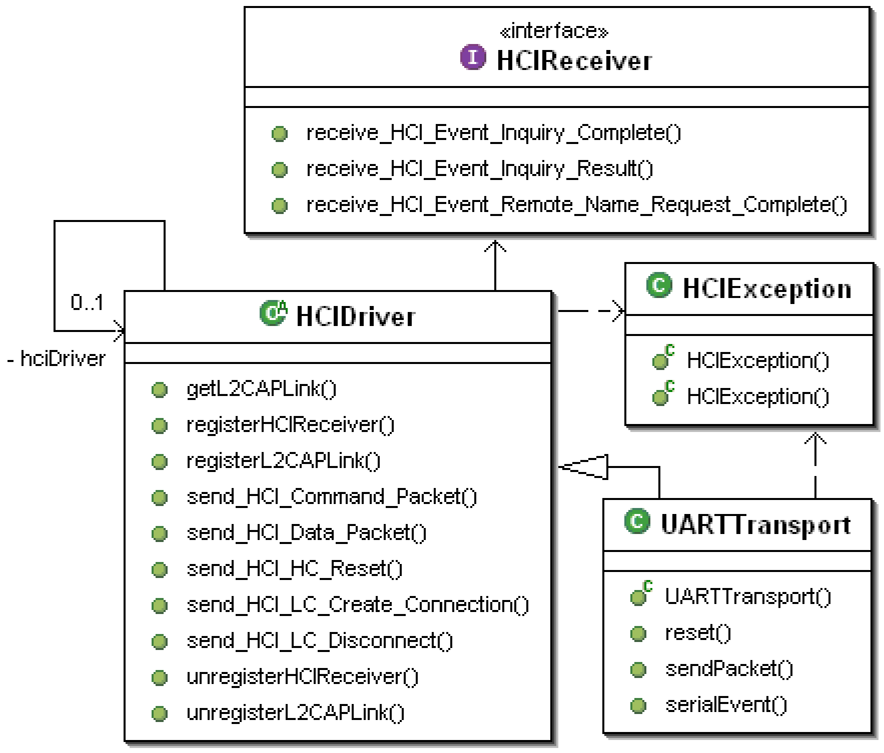


Fig. 4.0.2: HCI UML class diagram

4.0.5. Logical Link Control and Adaptation Protocol (L2CAP)

L2CAP adapts upper-layer protocols to the baseband layer by providing Synchronous Connection-Oriented (SCO) and Asynchronous ConectionLess (ACL) data services to upper-layer protocols. It has protocol multiplexing capability (this means that it's possible to use more than one higher-layer protocol), segmentation and reassembly operation of data packets, and group abstractions.

L2CAP class diagram:

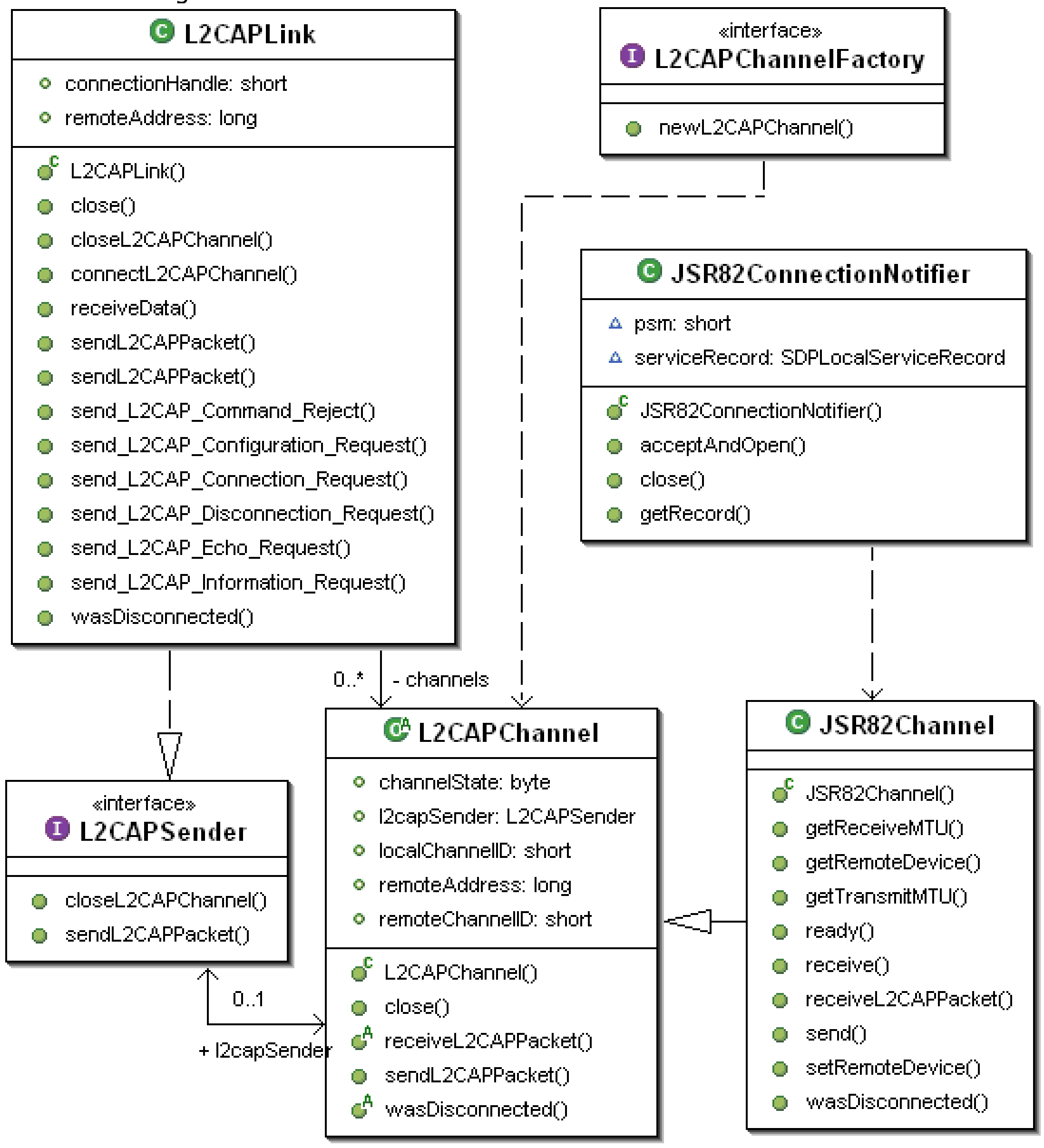


Fig. 4.0.3: L2CAP UML class diagram

4.0.6. Service Discovery Protocol (SDP)

SDP is used for applications to discover which services are available and to determine the characteristics of those available services on Bluetooth devices.

The service discovery mechanism provides the means for client applications to discover the existence of services provided by server applications as well as the attributes of those services. The attributes of a service include the type or class of service offered and the mechanism or protocol information needed to utilize the service.

The server maintains a list of service records that describe the characteristics of services associated with the server. Each service record contains information about a single service. A client may retrieve information from a service record maintained by the SDP server by issuing an SDP request.

There is maximum of one SDP server per Bluetooth device.

SDP class diagram:

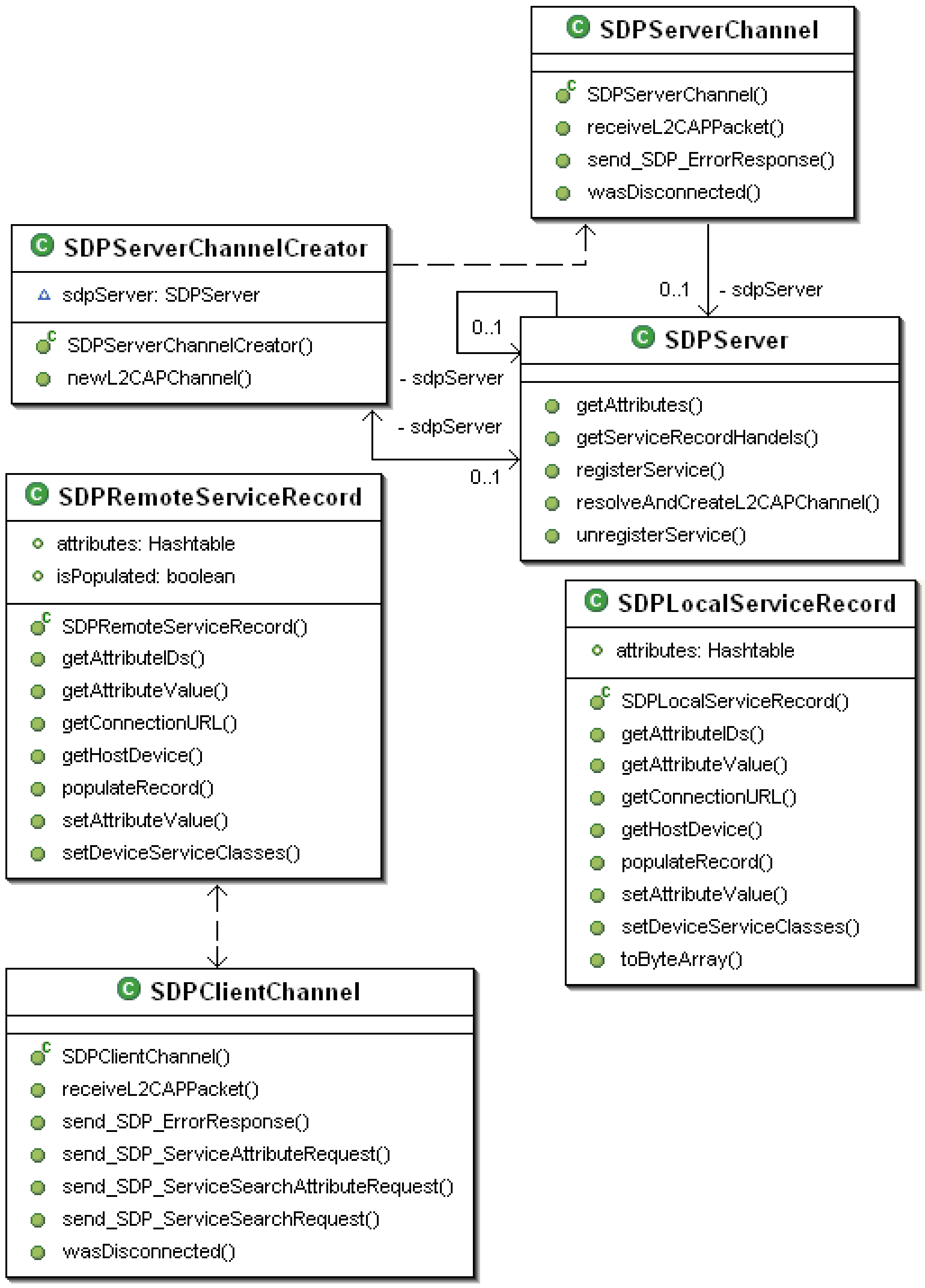


Fig. 4.0.4: SDP UML class diagram

4.0.7. RFCOMM

The RFCOMM protocol provides emulation of serial ports over the L2CAP protocol. It supports up to 60 simultaneous connections between Bluetooth devices, but the real number of simultaneous connections is implementation specific.

Complete communication path in RFCOMM involves two separate applications that are run on two different devices with communication segment between them.

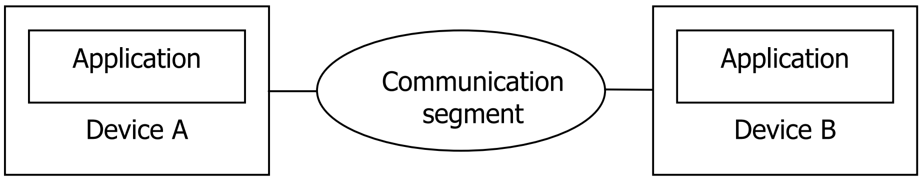


Fig. 4.0.5: RFCOMM communication path

RFCOMM has emulated RS-232 Circuits as follows:

Pin:	Circuit Name:
102	Signal Common
103	Transmit Data (TD)
104	Receive Data (RD)
105	Request To Send (RTS)
106	Clear To Send (CTS)
107	Data Set Ready (DSR)
108	Data Terminal Ready (DTR)
109	Carrier Detect (CD)
125	Ring Indicator (RI)

It is possible to have multiple serial ports between two devices – a Data Link Connection Identifier (DLCI) identifies an ongoing connection between a client and a server application. 6 bits represent the DLCI, but values 1 (reserved due to concept of Server Channels), 62 and 63 are unusable.

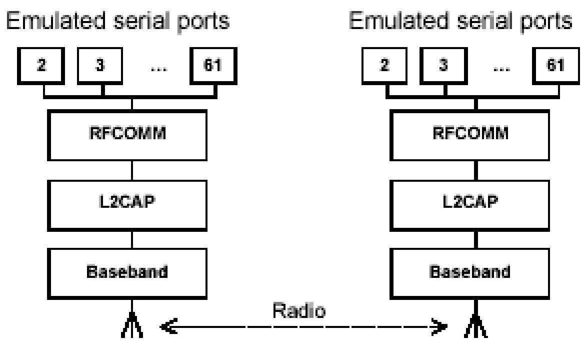


Fig. 4.0.6: Multiple emulated serial ports

Multiple emulated serial ports and multiple Bluetooth devices – if a Bluetooth device supports multiple emulated serial ports and the connections are allowed to have endpoints in different Bluetooth devices, then the RFCOMM entity must be able to run multiple multiplexer sessions.

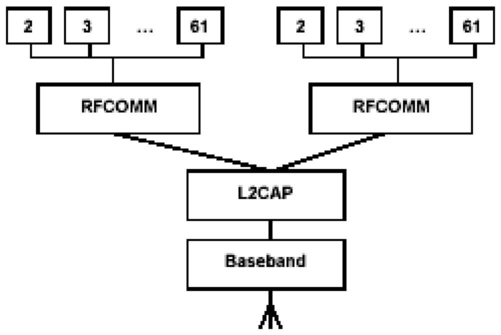


Fig. 4.0.7: Emulating serial ports coming from two Bluetooth devices

Service definition model:

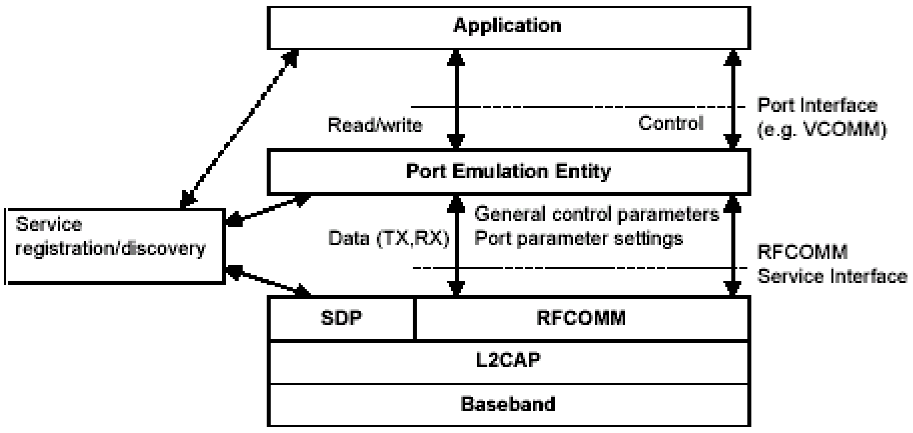


Fig. 4.0.8: RFCOMM reference model

The lowest layer here is also *Baseband* and Baseband protocol is defined by Bluetooth specification.

L2CAP layer is capable of channel multiplexing.

Server applications are registered on local device, and SDP provides services for client applications to discover how to reach server applications on other devices.

RFCOMM provides a transparent data stream and control channel over an L2CAP channel, and also multiplexes multiple emulated serial ports.

The *Port emulation entity* maps a system-specific communication interface (API) to the RFCOMM services. In combination with RFCOMM it makes the port driver.

Applications are used for utilization of serial port communication interface.

Frame types supported in RFCOMM:

Name:	Type:
Set Asynchronous Balanced Mode (SABM)	command
Unnumbered Acknowledgement (UA)	response
Disconnected Mode (DM)	response
Disconnected (DISC)	command
Unnumbered Information with Header check (UIH)	command and response

Specification TS 07.10 defines a multiplexer that has dedicated control channel, DLCI 0. It is used to convey information between two multiplexers.

Supported Control Channel Commands:

Test Command (Test)

Flow Control On Command (FCon)
Flow Control Off Command (FCoff)
Modem Status Command (MSC)
Remote Port Negotiation Command (RPNC)
Remote Line Status (RLS)
DLC parameter negotiation (PN)
Non Supported Command response (NSC)

Start-up procedure of serial port communication over RFCOMM:

- Establish an L2CAP channel to the peer RFCOMM entity, using L2CAP service primitives
- Start the RFCOMM multiplexer by sending SABM (Set Asynchronous Balanced Mode) command on DLCI 0 and await UA response from peer entity.
- Data Link Connections (DLC) can be now established for user data traffic.

Closedown procedure of serial port communication over RFCOMM:
The Device closing the last connection (DLC) on a particular session is responsible for closing the multiplexer by closing the corresponding L2CAP channel. Closing the multiplexer by first sending a DISC command frame on DLCI 0 is optional, but it is mandatory to respond correctly to DISC (with UA response).

Link-loss handling:
If an L2CAP link loss notification is received, the local RFCOMM entity is responsible for sending a connection loss notification to the port emulation/proxy entity for each active DLC.

Flow control in the wired systems is based on RTS/CTS signals, but the flow control between RFCOMM and the lower layer L2CAP depends on the service interface supported by the implementation. RFCOMM has also it's own flow control mechanisms, and L2CAP relies on the flow control mechanism provided by Link Manager layer in the Baseband.

Interaction with other entities:

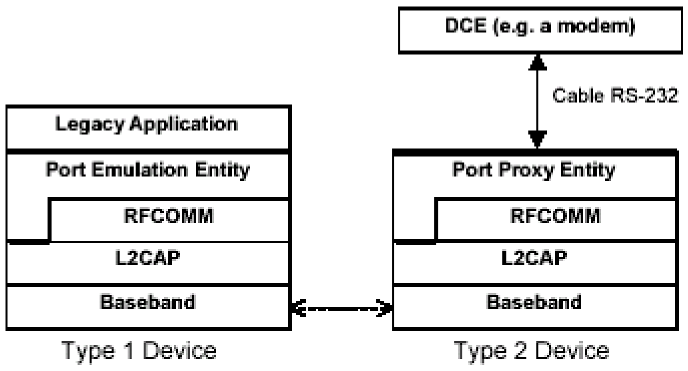


Fig. 4.0.9: The RFCOMM communication model

Type 1 devices are communication endpoints such as computers and printers.
Type 2 devices are part of a communication segment (i.e. modems).

The *Port Emulation Entity* maps a system specific communication interface (API) to the RFCOMM services.
The *Port Proxy Entity* relays data from RFCOMM to the external RS-232 interface linked to a DCE (Data Circuit-Terminating Equipment) – communication parameters are set according to received RPN commands.

RFCOMM Class-diagram:

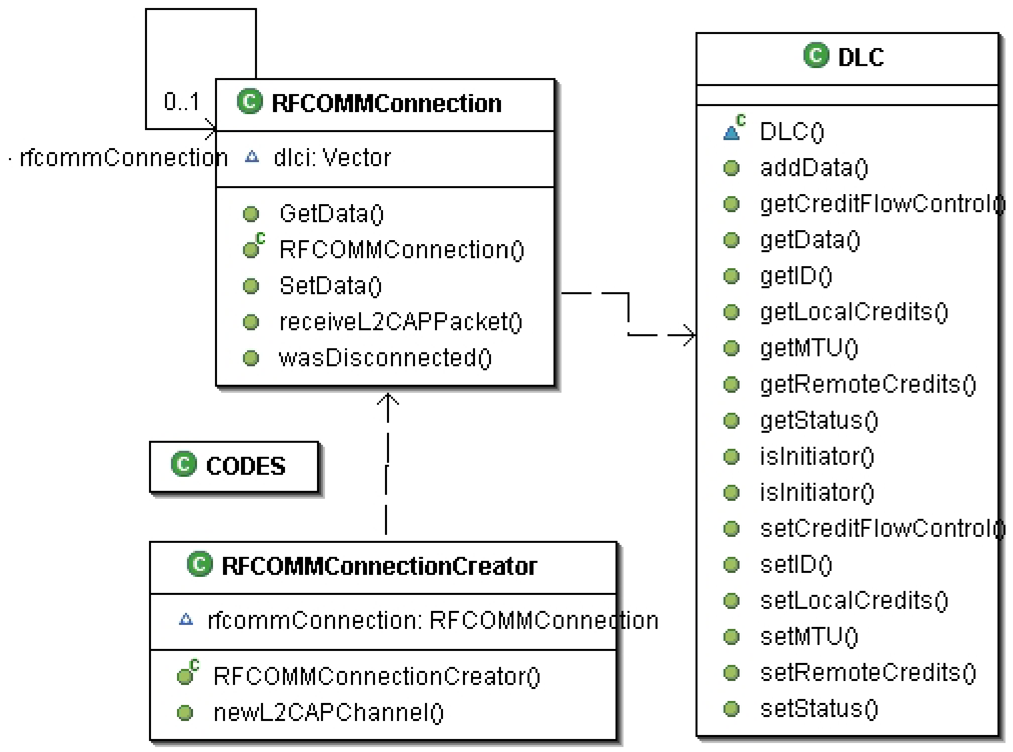


Fig. 4.0.10: RFCOMM UML class diagram

Server creation and initialization through RFCOMMServer class.
RFCOMMConnection class calls RFCOMMConnection that creates new L2CAP channel on which RFCOMM based communication will be established.
RFCOMMConnectionCreator class is used for establishing L2CAP communication channel.
DLC class is holding the channel ID, direction and all other parameters associated with each of DLC connections. It also creates and sends, receives and parses L2CAP packets.
CODES class contains codes for RFCOMMConnection class.

4.0.8. OBject EXchange protocol (OBEX)

OBEX is protocol that Bluetooth adopted from IrDA. Original name of this protocol (defined by IrDA) is IrOBEX, but shorter name OBEX is used instead. By adopting OBEX, it is possible to use it over IrDA IR (light) or Bluetooth technology (radio signals).

OBEX was developed to exchange data objects over an infrared link and was placed within the IrDA protocol hierarchy. However, it can appear above other transport layers, now RFCOMM and TCP/IP. At this moment, it is worth mentioning that the OBEX over TCP/IP implementation is an optional feature for Bluetooth applications supporting the OBEX protocol.

The IrOBEX protocol follows a client/server request-response paradigm for the conversation format.

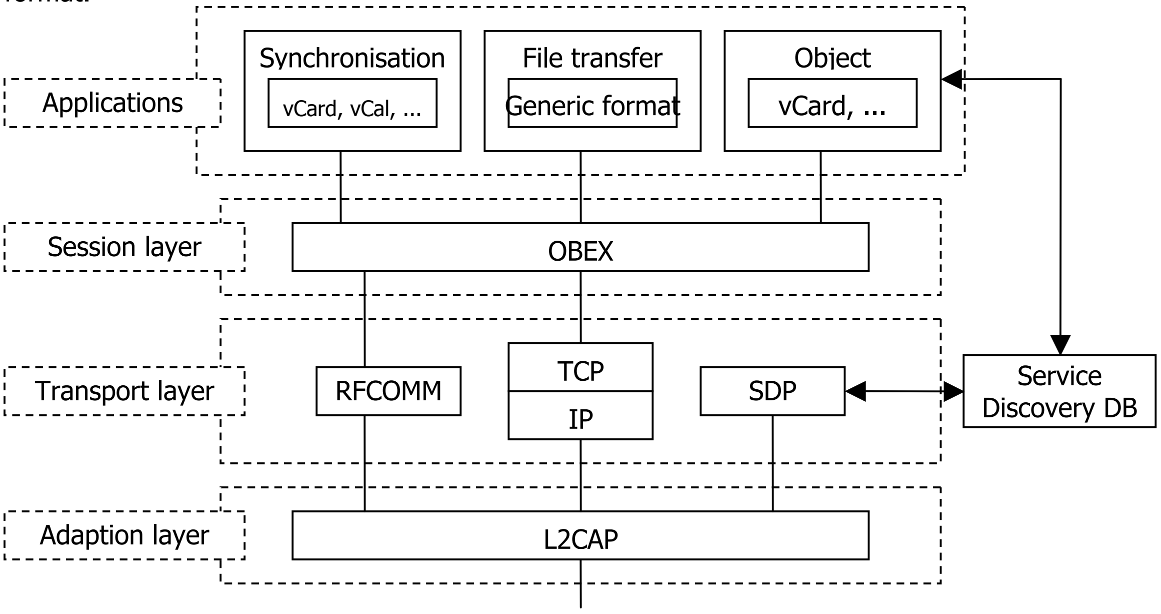


Fig. 4.0.11: OBEX hierarchy diagram

In the Bluetooth system, the purpose of the OBEX protocol is to enable the exchange of data objects. The typical example could be an object push of business cards to someone else. A more complex example is synchronizing calendars on multiple devices using OBEX.

For the Object Push and Synchronization applications, content formats can be the vCard, vCalendar, vMessage, and vNotes. Those content formats describe the formats for the electronic business card, the electronic calendar and scheduling, the electronic messages and mails, and the electronic notes, respectively.

OBEX protocol can transfer an object by using the **Put** and **Get** operations. One object can be exchanged in one or more Put requests or Get responses. The model handles both information about the object (e.g. type) and object itself. It is composed of headers, which consist of a header ID and value. The header ID describes what the header contains and how it is formatted, and the header value consists of one or more bytes in the format and meaning specified by header ID.

The specified headers are: Count, Name, Type, Length, Time, Description, Target, HTTP, Body, End of Body, Who, Connection ID, Application Parameters, Authenticate Challenge, Authenticate Response, Object Class, and User-Defined Headers.

Session protocol – the OBEX operations are formed by response-request pairs. Requests are issued by the client and responses by the server. After sending a request, the client waits for a response from server before issuing a new request.

Connect operation:
An OBEX client starts the establishment of an OBEX connection.
At the remote host, the OBEX server receives the Connect-request, if it exists. The server accepts the connection by sending the successful response to the client.

Disconnect operation:
The disconnection of OBEX session occurs when an application, which is needed for an OBEX connection, is closed or the application wants to change the host to which the requests are issued. The client issues the Disconnect-request.

Put operation:

When the connection has been established between the client and server the client is able to push OBEX objects to the server.

A Put-request consists of one or more request packets, depending on how large the transferred object is, and how large the packet size is. A response packet from the server is required for every Put-request packet.

Get operation:

When the connection has been established between the client and server, the client is also able to pull OBEX objects from server.

Other operations:

Other operations consist of SetPath, and Abort operation.

OBEX over RFCOMM – The Bluetooth devices supporting the OBEX protocol must satisfy the following requirements:

The device supporting OBEX must be able to function as a client, a server, or both.

All servers running simultaneously on a device must use separate RFCOMM server channels.

Applications (service/server) using OBEX must be able to register the proper information into the service discovery database. This information for different application profiles is specified in the profile specifications.

OBEX server start-up on RFCOMM:

When a client sends a connecting request, a server is assumed to be ready to receive requests. However, before the server is ready to receive (i.e. is running) certain prerequisites must be fulfilled before the server can enter the listening mode:

The server must open an RFCOMM server channel

The server must register its capabilities into the service discovery database

Connection establishment:

A client initiates the establishment of a connection. However, the following sequence of tasks must occur before the client is able to send the first request for data:

By using the SD protocol described in SDP specification, the client must discover the proper information (e.g. RFCOMM channel) associated with the server on which the connection can be established.

The client uses the discovered RFCOMM channel to establish the RFCOMM connection

The client sends the Connect-request to the server, to establish an OBEX session. The session is established correctly if the client receives a successful response from the server.

Disconnection:

Using the Disconnect-request does the disconnection. When the client has received the response, the next operation is to close the RFCOMM channel assigned to the OBEX client.

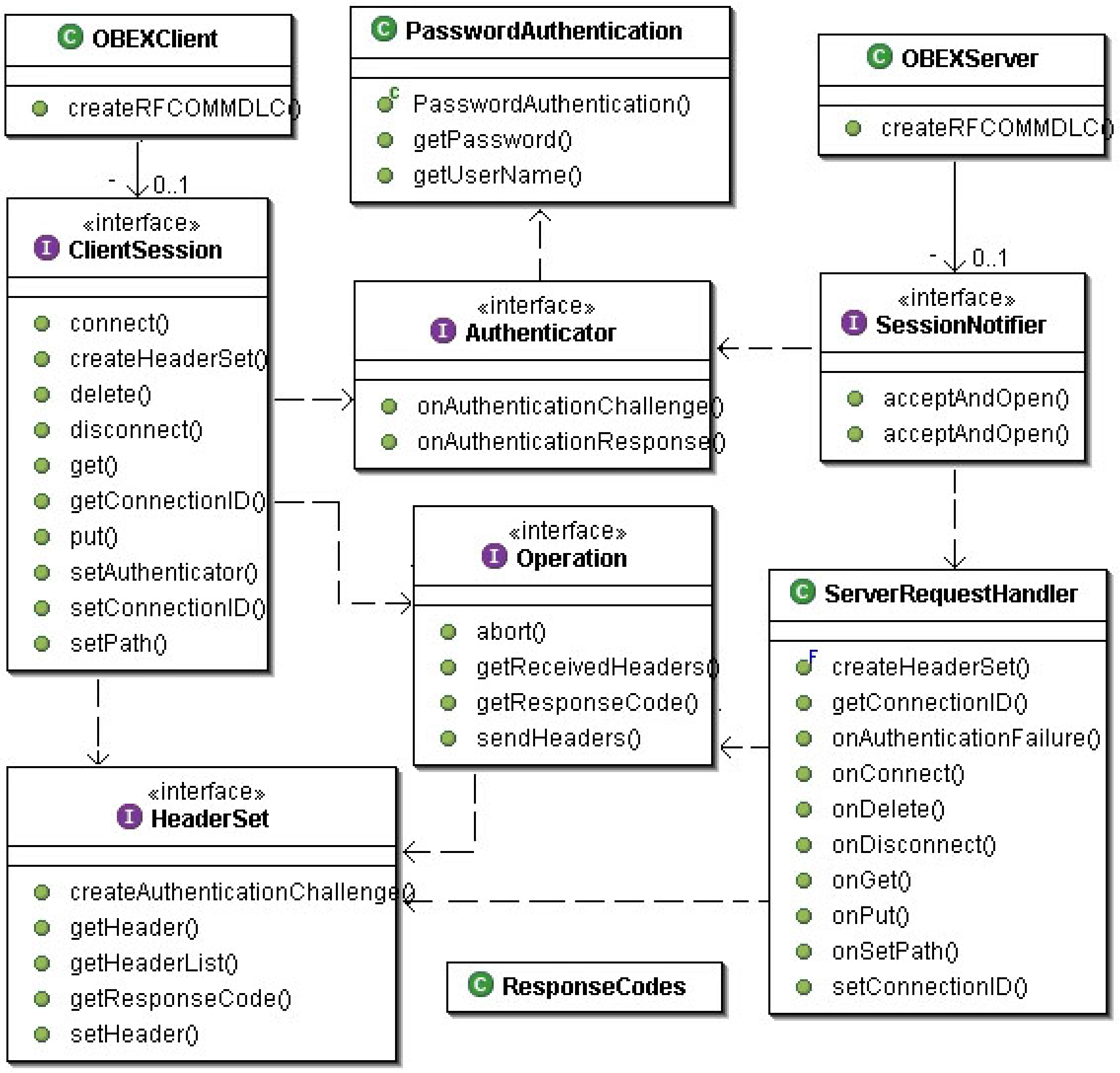


Fig. 4.0.12: OBEX UML class diagram

OBEXClient class creates RFCOMM DLC on the client side that is used to connect to the server with the data received using SDP. It also implements push, pull and abort operations.

ClientSession interface provides methods for OBEX requests and provides way to define headers for OBEX operations (CONNECT, PUT, GET, SETPATH and DISCONNECT).

HeaderSet interface defines the methods that set and get values of OBEX headers.

PasswordAuthentication class holds user name and password combinations.

Authenticator interface provides a way to respond to authentication challenge and authentication response headers. When a client or a server receives authentication challenge or authentication response header the *onAuthenticationChallenge()* and *onAuthenticationResponse()* methods are called respectively, by the implementation.

Operation interface provides ways to manipulate a single OBEX PUT or GET operation.

SessionNotifier interface defines a connection notifier for server-side OBEX connections. When a SessionNotifier is created and calls *acceptAndOpen()* method, it will begin listening for clients to create a connection at the transport layer.

ServerRequestHandler class defines an event listener that will respond to OBEX requests made to the server by appropriate response codes.

OBEXServer class creates RFCOMM DLC on the server side that listens for OBEX client requests. When the request is received, connection is established.

4.0.9. Bluetooth security

JSR-82 API needs implementation of Bluetooth Control Center (BCC) that specifies basic properties of Bluetooth device. One of those properties is Bluetooth security.

BCC sets security level for device, maintains a list of earlier discovered devices (trusted one and not trusted one), it gives mechanism for pairing up and authorization of devices that are communicating for the first time.

There is need for supply BCC with primary security mechanisms and interfaces. Majority of this part is closely related with Link and Baseband layer, so the interfaces to those layers will be implemented by HCI layers method invocations concerning authentication, authorization and encryption processes.

During the process of authentication need for PIN (Personal Identification Number) is supposed. PIN can exist in device's memory or user can supply one. Except the fact about who is supplying PIN decision must be made about how many various PINs you can enter (one or more). The most important thing about PIN is that it should never be transferred between devices. It is only used for personal authentication.

Basic Bluetooth security modes already implemented by the Bluetooth specification:
Mode 1: No security measures. Every other device can contact this device without going through authentication, encryption and authorization processes.
Mode 2: Security procedures are initiated after channel establishment request has been received at L2CAP level
Mode 3: Security measures are activated at LMP level before a channel is created and opened for communication

There are four entities for maintain security at the link layer:

BD_ADDR	48 bits
Private user key, authentication	128 bits
Private user key, encryption Configurable length (byte-wise)	8-128 bits
RAND	128 bits

Table 4.0.1: Entities used in authentication and encryption

Process of authentication is based on shared link key between devices. If two devices communicates for the first time, there is no link key and it needs to be created. In order to create link key, initialization key must be created first, but that is always case because it is automatically created at the beginning of the communication.

Initialization key is derived from BD_ADDR, IN_RAND (random number), PIN length and PIN code using algorithm E22. After initialization key has been created, devices forms the link key. Link key can be a unit key or combinational key. Unit key is generated with first usage of the device and it is stored in non-volatile memory. It could be changed, but not very easy. Algorithms for link key generation are described in Bluetooth specification v1.1.

Authentication begins after link key is generated.

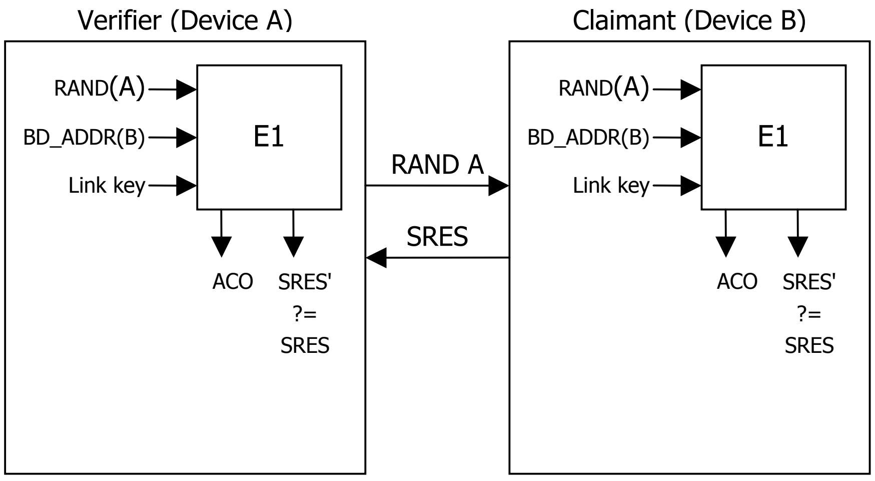


Fig. 4.0.13: Description of authentication process

At the beginning of authentication process the verifier sends the random number to claimant. Both participants then use authentication function E1 with the claimant's Bluetooth Device Address, shared link key and sent random number. Claimant then sends response to verifier, and verifier compares response with his result. If they match, the claimant is authenticated. To achieve mutual authentication, the claimant just need to issue another random number to verifier and wait for response.

If the authentication fails, there is a delay between new authentications, and that delay doubles each time that authentication fails, until the maximum delay time is reached (from the same address).

Another important parameter in encryption process is Authenticated Ciphering Offset (ACO). Along with link key and generated random number it makes input entries for E3 algorithm for generation of encryption key. ACO and link key are known to both devices, and every time when device enters encryption mode new encryption key is generated.

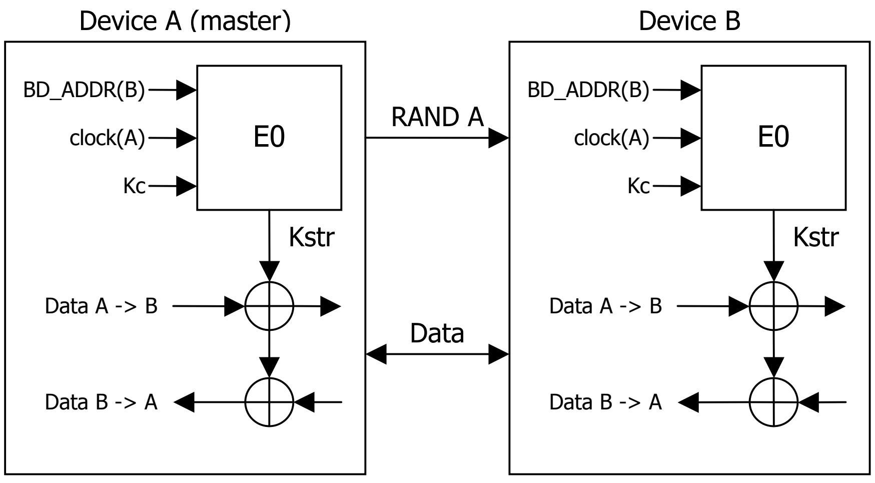


Fig. 4.0.14: Description of encryption process

Encryption algorithm E0 has following parameters – BD_ADDR, 26 bits of master real-time

clock and encryption key Kc. EN_RAND is issued by master before entering encryption mode. With E0 encryption key is derived by another encryption key.
JSR-82 API implemented security features:
Security measures by JSR-82 API specification are implemented in link layer. JSR-82 specifies API functions to activate authentication, encryption and authorization functions. One way for using those function is to specify them as optional parameters in URL of service that is wanted to use.

Bluetooth Server can request authentication by adding the authenticate parameter to the connection URL string
(String url = "btspp://localhost:81283209832382323…..32;authenticate=true").
Same rule applies for authorization and encryption keywords.
Clients can request authentication and encryption in the same way – there is no point of using authorization for clients so it's not allowed for clients at all.

When server calls Connector.open() method, a new Service Record is created based on optional attributes that reside in the URL string. That record can be edited afterwards if there is any need to do it. Record after that goes to SDDB, before a client makes a connection.

At the client side client application opens a connection to that service. I order to achieve its goal it needs to go through all security requirements specified in server application's record. Those security procedures are activated through HCI interface method invocation. BCC is example of centralized security manager that manages all this connection requests and takes specified actions in order to protect security policy.

Another issue concerns applying security policies on already created connections. Suppose that devices have already established connection and remote device wants to authenticate itself. BCC has interface to link layer (through interface to HCIDriver).

Bluetooth devices can be divided in trusted and non-trusted. That's another thing to worry about. BCC has a table of known devices. Devices that are marked as trusted have access to all services no matter of service security attributes (encryption, authentication, authorization). Naturally, there are non-trusted ones that have limited access.

Bluetooth security class diagram:

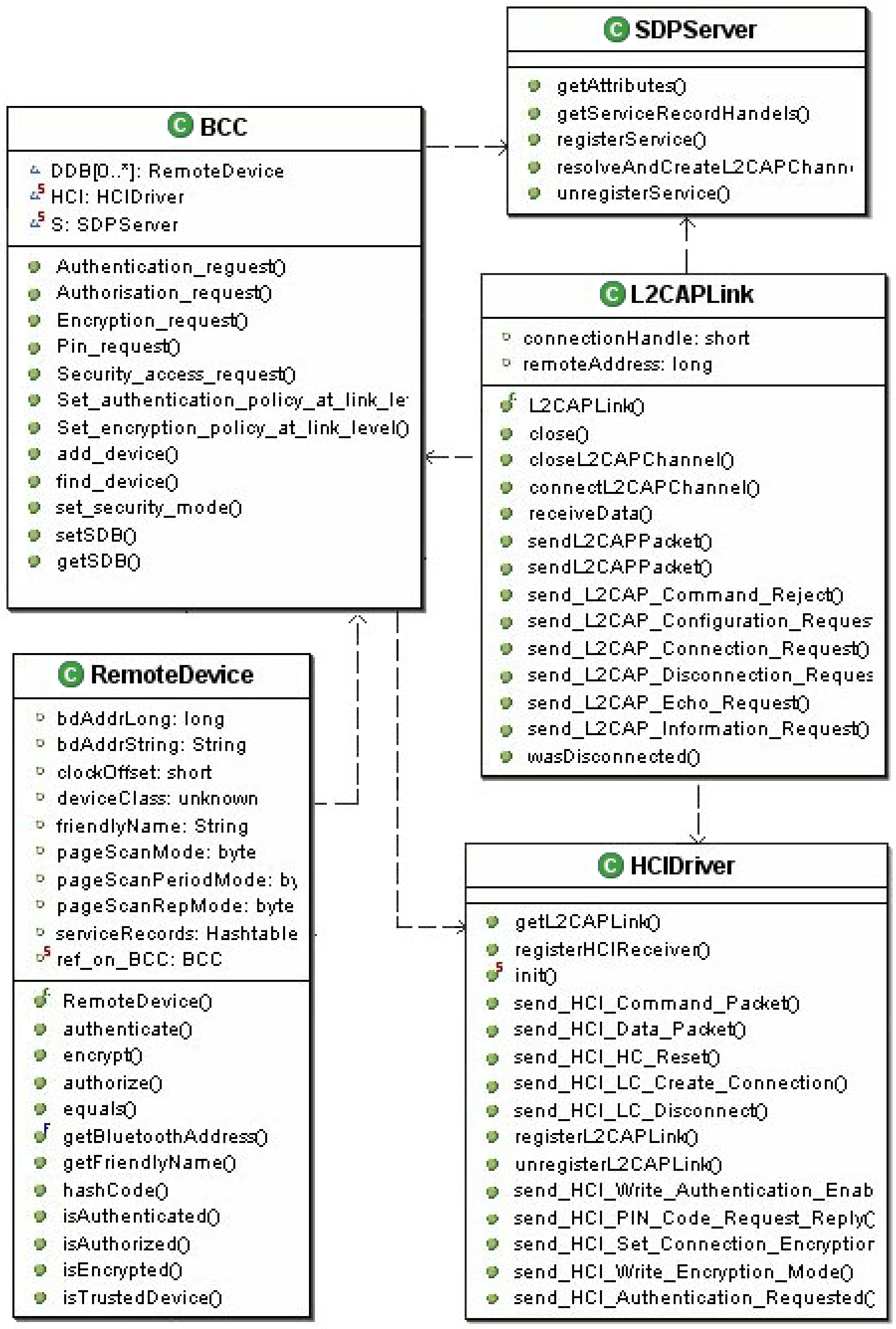


Fig. 4.0.15: Bluetooth security UML class diagram

BCC class is used for sending authentication, authorization, encryption, security and PIN requests. It is also responsible for setting security at link level, and for finding devices.

SDPServer class resolves and creates L2CAP channel, register and unregister services and gets attributes and service record handles.

L2CAPLink class is responsible for L2CAP channel connection and disconnection (closing channel). It sends and receives L2CAP packets, and makes L2CAP requests.

RemoteDevice class represents the security features of remote devices, like Bluetooth address, friendly name, requested security level, etc. There is also possibility for finding out security measurements set on the device.

HCIDriver class is responsible for communication with host controller interface (HCI) layer. It sends HCI packets, and is used for register and unregister L2CAP link.

5. Approvals

Name	Title	Date yyyy-mm-dd	**Signature**