

FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Wirtschaftsinformatik

**Implementation of a Bluetooth touchpad based on Android OS**

Nikolay Kostadinov





FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Wirtschaftsinformatik

Implementation of a Bluetooth touchpad based on Android OS

Implementierung eines Bluetooth-Touchpads

auf Basis von Android OS

|  |  |
| --- | --- |
| Author: | Nikolay Kostadinov |
| Supervisor: | Prof. Dr. Uwe Baumgarten |
| Advisor: | M.Sc. Nils T. Kannengießer |
| IN_Web.jpgSubmission Date: | 15.10.2011 |

Ich versichere, dass ich diese Bachelorarbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. Oktober 2011 Nikolay Kostadinov

I assure the single handed composition of this bachelor thesis only supported by declared resources.

Munich, the October 15th, 2011 Nikolay Kostadinov

Acknowledgements

First of all I would like to thank my advisor M.Sc. Nils Kannengießer for his outstanding support, supervision, useful suggestions and encouragement which was invaluable for the successful completion of this work. Thanks to Prof. Dr. Uwe Baumgarten for giving me a chance to work on this and other marvelous projects. Thanks to everyone from the chair for operating systems in the TUM as well.

Furthermore, I would like to thank my family for general education and ongoing support. Last but not least, I want to thank my more than talented girlfriend Evgeniya Sofronieva, who turned my slobbery hand-drawings into beautiful graphics.

Abstract (English)

Smartphones are gaining popularity both in the corporate and the entertainment sectors. They are gradually becoming a universal device, able to complete a variety of different tasks and fit into various use case scenarios. This work concentrates on realizing a single scenario and presents a completely new way of using a mobile phone for remote control of notebooks, computers and other Bluetooth-enabled devices.

The aim of this work was to develop a touchpad by using the Android OS as a platform. The touchpad application running on Android phone is able to connect to other systems over the Bluetooth radio technology. By using a set of standard supported drivers, the application provides an input service for the user that is not less powerful than the capabilities of ordinary input devices such as mouse and keyboard. The project not only fulfils this goal, but also introduces an extensible framework, which is extremely easy to implement by developers willing to unleash the power of the Bluetooth communication in combination with the widely supported standard drivers for input devices.

The open source Android operating system has established its place as the most popular operating system, designed to power smart phones and other mobile devices. Devices running this freely distributed OS are less expensive, than other devices with similar hardware specifics. For its openness, user-friendly concepts and developer-friendly software development tools, it became the platform of choice for this project.

Abstract (Deutsch)

Smartphones werden immer populärer sowohl in der Unternehmens- als auch in der Unterhaltungsbranche. Sie werden allmählich zu einem universellen Gerät, das in der Lage ist, zahlreiche Aufgaben zu erfühlen. Deswegen findet es auch in vielen Anwendungfälle einen Platz. Diese Arbeit konzentriert sich auf die Realisierung von so einen Anwendungfall und präsentiert eine völlig neue Art und Weise, wie die Fernsteuerung von Notebooks, Rechnern und andere Bluetooth-fähigen Geräten mit Hilfe eines Mobiltelefons betrieben werden könnte.

Das ursprüngliche Ziel dieser Arbeit war es, ein Touchpad auf Basis von Android OS zu entwickeln. Die Touchpad-Anwendung, die aufs Android-Handy läuft kann sich mit anderen Systeme mit Hilfe der Bluetooth-Technologie verbinden. Durch die Verwendung von einer Reihe von Standard-unterstützten Treiber bietet die Anwendung den Nutzer zahlreiche Eingabemöglichkeiten, die nicht weniger mächtig sind als diese, die von üblichen Geräten wie Maus und Tastatur angeboten sind. Allerdings, erfühlt das Projekt nicht nur dieses Ziel. Es wird ein Framework vorgestellt, welche von den Entwicklern sehr einfach zu implementieren ist. Damit können sie Applikationen entwickeln, die sowohl die Vorteile der Bluetooth-Kommunikation, als auch der breit unterstützten Treiber für Eingabegeräte ausnutzen.

Die Open-Source-Betriebssystem Android hat sich in der letzten Jahren als die meistgenutzte Betriebsystem etabliert, die speziell für mobile Geräte entwickelt ist. Geräte, auf die dieses freies OS läuft, sind meistens billiger als andere Geräte mit vergleichbaren Hardware-Spezifikationen. Android ist offen und bietet benutzerfreundliche Konzepte, sowie entwicklerfreundliche  Software-Entwicklungstools. Deswegen ist Android die natürliche Wahl für diesen Projekt.

Contents

[1. Introduction 1](#_Toc304465352)

[2. Bluetooth technology 2](#_Toc304465353)

[2.1 Common Bluetooth devices 2](#_Toc304465354)

[2.2.1 Mouse 2](#_Toc304465355)

[2.2.2 Keyboard 3](#_Toc304465356)

[2.1.3 Graphics pad 4](#_Toc304465357)

[2.1.4 Bluetooth devices 5](#_Toc304465358)

[2.1.5 Bluetooth radio 5](#_Toc304465359)

[2.2 Bluetooth software stack 6](#_Toc304465360)

[2.2.1. Bluetooth protocol architecture 7](#_Toc304465361)

[2.2.2 Service Discovery Protocol 11](#_Toc304465362)

[Glossary 1](#_Toc304465363)

[Bibliography 2](#_Toc304465364)

**List of Figures**

[Figure 1: Mouse functionality 3](file:///G:\work\EclipseWinRepo\%20android-bluetooth-touchpad\BTTouchpad\thesis\BA.docx#_Toc304645378)

[Figure 2: Graphics pad concept 4](file:///G:\work\EclipseWinRepo\%20android-bluetooth-touchpad\BTTouchpad\thesis\BA.docx#_Toc304645379)

[Figure 3:Frequency hopping scheme[3] 7](file:///G:\work\EclipseWinRepo\%20android-bluetooth-touchpad\BTTouchpad\thesis\BA.docx#_Toc304645380)

[Figure 4: Bluetooth stack on PC Host and HID [6] 9](#_Toc304645381)

[Figure 5: Bluetooth stack architecture [5] 10](#_Toc304645382)

[Figure 6: SDP connection scheme [5] 13](#_Toc304645383)

[Figure 7: Service record [5] 14](#_Toc304645384)

[Figure 8: Service attribute [5] 15](#_Toc304645385)

[Figure 9: Attributes in different service classes 16](#_Toc304645386)

[Figure 10: Nexus One [7] 18](#_Toc304645387)

[Figure 11: Android architecture[10] 20](#_Toc304645388)

[Figure 12: Dalvik vs. Java VM 22](#_Toc304645389)

[Figure 13: Bluetooth in Android 24](file:///G:\work\EclipseWinRepo\%20android-bluetooth-touchpad\BTTouchpad\thesis\BA.docx#_Toc304645390)

[Figure 14: Bluetooth client/server app concept 25](file:///G:\work\EclipseWinRepo\%20android-bluetooth-touchpad\BTTouchpad\thesis\BA.docx#_Toc304645391)

[Figure 15: Adding a new service 26](file:///G:\work\EclipseWinRepo\%20android-bluetooth-touchpad\BTTouchpad\thesis\BA.docx#_Toc304645392)

# Introduction

As Android becomes more and more popular, third-party developers are producing an increasing number of Android applications. The applications are small and useful programs, utilizing different combinations of hardware features. Although, most of the mobile devices running Android are supporting Bluetooth communications, very little is done for realizing the vision that a mobile phone could be used as an universal remote control, that could connect to virtually any notebook, computer or other type of device supporting the Bluetooth technology and a standard set of drivers for input services, such as the HID drivers.

The main problem is the missing support for these drivers. The developer would have to dive deep into the lower levels of the operating system’s architecture and write programs interacting directly with the Bluetooth stack as part of the operating system’s core. The input service that would be provided by the application must be described according to the HID protocol. Information on how to do this is also spare. The resulting service description has to be inserted in the registry of an existing SDP module, which is responsible for making it publicly available, so the computer could find it and read it. Since the computer and respectively the user are aware of the service, a communication channel over Bluetooth must be established, so the service would be utilized.  Although, Bluetooth communication is generally supported by APIs that are part of the Android framework, developers are not provided with access to the lower level communication protocols, such as L2CAP. However, if communication on the higher levels is possible, there must be a way of accessing the protocols beneath.

The Bluetooth touchpad project is presenting a solution for each of these problems. A service description for a Bluetooth mouse and keyboard is defined and passed to the SDP server of the Android operating system. On the other hand, the resulting application is able of establishing communication by using the required protocols, although this feature is not officially supported. The solutions are packed in an extensible framework, which could be easily implemented by developers and used in other projects. The Bluetooth touchpad app, implemented on top of the framework is abstracting from the mouse and keyboard input specifics and thus provides a number of totally different input capabilities by using the phone’s motion sensors, the phone’s display and even the voice recognition API.

In this work, first the Bluetooth technology and also some specifics of common Bluetooth input devices are presented. The SDP is also explained, since it is playing an important role in the Bluetooth stack. Afterwards, the Android platform and the development phone used in the project are briefly introduced. After building up this foundation of knowledge, the implementation of the framework is reviewed in detail. In the last chapter results of software tests are provided and the application’s performance is measured.

# Bluetooth technology

In order to implement a Bluetooth touchpad based on the Android OS, one must first understand what Bluetooth is and how it works. In this chapter, first some common Bluetooth devices are introduced and then the communication technology behind them is reviewed in detail. Understanding both the hardware specifics of the devices and the architectural model behind Bluetooth is crucial in order to implement a new Bluetooth device from scratch.

## 2.1 Common Bluetooth devices

The task of realizing a Bluetooth touchpad on a mobile phone, running Android OS, could be moreover approached as simulating the behavior of a wireless Bluetooth mouse and keyboard. Subsequently, in order to complete this task, the functionality of both, as well as their physical capabilities and structure must be considered. In the following section the hardware specifics of regular corded mouse and keyboard, as well as a graphics pad device are successively discussed. Then the hardware capabilities of such wireless devices and the underlying Bluetooth technology are further reviewed. Finally, the process of physical connection and identification of Bluetooth devices, as well as some communication security concerns are briefly looked through.

### 2.2.1 Mouse

The main goal of the modern mouse is to translate the motion of your hand into signals that the computer can use as an input method. A simple, standard featured mouse consists of two buttons (left and right) and a scroll wheel, which could also act as a third button. Furthermore, the mouse motions on a flat surface are translated into the motion of a cursor on the computer’s display.



Figure : Mouse functionality

As shown on the figure 1 above, the mouse functional capabilities could be shortly described as the following three basic user interactions:

1) Pressing left/right button

2) Scrolling the wheel up/down

3) Mouse motion on surface

A mouse consists of several sensors that could handle and translate the user interaction into specifically formatted data, which is then sent to the computer for further processing.

### 2.2.2 Keyboard

The modern computer keyboard originates its design from the mechanical, non-electric typewriters invented in the 19th century. Today, it is used to type text and numbers into computer programs, where the interpretation of key presses is left to the underlying software programs. Keyboards often have different or additional keys depending on the manufacturer or the operating system they are designed for. However, the keys of different keyboards have similar size and shape. Furthermore, they are placed in a similar pattern, no matter what language is represented. The user interaction consists of pressing a single or a combination of keys at the same time. The keyboard reports all key presses to the operating system by sending them as specifically encoded data.

### 2.1.3 Graphics pad

The graphics pad (also called drawing tablet) is modern computer input device that enables the user to hand-draw graphics, similarly to the way a person can draw images with a pencil on paper or with fingers and paint on canvas. The ability to detect some or all of the pressure of the stylus and representing them on the computer display is considered to offer a natural way to create computer graphics. Figure 2 below is showing the concept of the device.



Figure : Graphics pad concept

Similarly to the mouse, the pad is able to capture the movement of the stylus or the user’s finger on its surface and translate it into the motion of a cursor. Since the behavior of this device is similar to the behavior of the mouse, the functionality of this device is also included in the Bluetooth touchpad implementation.

### 2.1.4 Bluetooth devices

Other than a regular mouse or keyboard, a wireless device is not using a cable connection for sending the data, but radio frequency technology. Radio frequency devices consist of two components: transmitter and receiver. The transmitter is placed in the device and is able of sending radio signal that encodes information about the user’s actions. In addition, the receiver is connected to the computer and is respectively accepting, decoding and passing the information to the computer’s operating system. Bluetooth is one of the most popular radio frequency technologies that wireless mice and keyboards use. Bluetooth is shortly described on the main page of its vendor- the Bluetooth Special Interest Group [[1](#a)] as “short-range communications technology that is simple, secure, and everywhere. You can find it in billions of devices ranging from mobile phones and computers to medical devices and home entertainment products. It is intended to replace the cables connecting devices, while maintaining high levels of security.” (Bluetooth Basics[[2](#b)]) Indeed, the fact that Bluetooth receivers can accommodate multiple Bluetooth peripherals at the same time is one of the main reasons why the technology has established its status as one of the most popular wireless standards ever.

### 2.1.5 Bluetooth radio

Almost all of the electronic devices today utilize radio frequencies (RF) to communicate with other devices. In order to avoid conflicts during communication, different devices use different frequencies. One of the benefits of the radio frequency technology is that it does not need a clear line of sight between the transmitter and the receiver. Unlike the infrared based communication technology, used for example in TV remote controls, the wireless signal can pass through barriers such as furniture or walls. What is more, the RF technology provides variety of other advantages for the wireless devices- the RF transmitters and receivers are very inexpensive, tiny and light weight. Furthermore they require low power and can therefore run on batteries.

Bluetooth is one of the most widely used RF technologies. It allows a large number of different devices to connect to each other such as: phones, printers, notebooks, tablets etc. Bluetooth devices usually have a range of 5 to 10 meters and operate in the 2.4 GHz range by using RF. One Hertz (Hz) indicates thousand cycles per second or thousand electromagnetic waves per second. Subsequently one Megahertz is one million and one Gigahertz (GHz) is one billion cycles per second. [[3](#c)]

In order for two Bluetooth devices to establish communication channel and transmit data, they must be “paired”. Pairing indicates the process of determining a common frequency and also a common communication code, resulting in a communication channel. Consequently pairing makes it possible to filter out interference from other RF devices. There are several methods of pairing, depending of the type of device and its manufacturer. If both devices have display, which is the case when pairing an Android phone and a Bluetooth capable computer, the “Numeric Comparison” is usually used. A 6-digit numeric code is shown on each display and the user is asked to compare the numbers to ensure they are identical. Once the comparison is successful, one could confirm the pairing and data transfer between both devices may start. If the user has confirmed on both devices and performed the comparison properly, this method provides significant protection from one of the most common attacks- “man in the middle”. [[4](#d)]

On the other hand, devices with limited input capabilities, such as Bluetooth mice and keyboards either require the user to enter a pin, which is predefined and usually easy to guess (“0000” or “1234”) or they do not require any user interaction at all. Obviously, this type of pairing does not provide protection against “man in the middle” attacks. As a consequence, a Bluetooth touchpad realized on an Android phone provides better protection then an ordinary Bluetooth mouse and keyboard, since it provides possibility for numeric comparison as part of the Android operations system.

In addition, Bluetooth devices use encryption schemes to encrypt data in unreadable format, as well the frequency-hopping method. This method causes the two Bluetooth devices to automatically change frequencies. Frequency-hopping “divides the band into 79 channels (each 1 MHz wide) and changes channels up to 1600 times per second”.[[3](#c)] Every Bluetooth device has a physical clock responsible for this frequency change. Therefore, in order to establish a communication channel the devices need to synchronize their clocks and their frequency hopping patterns, a piconet is created. The concept of frequency hopping pattern is shown on figure 3 below. [[3](#c)]



Figure :Frequency hopping scheme [[3](#c)]

A piconet consists of master and between one and seven slaves. The master is responsible for setting the clock time and also the hopping pattern. Slaves, on the other hand, accept the master’s settings. Moreover, a Bluetooth device could be master in only one piconet, but a slave across multiple piconets. Generally, this frequency hopping technology strengthens the security of the Bluetooth communication, because “any device not belonging to the piconet is unable to participate in communications by sending or listening to the data exchanged because it does not have access to the frequency hopping sequence. “ (Bluetooth security mechanisms [3])

## 2.2 Bluetooth software stack

After taking into consideration the hardware specifics of Bluetooth mouse and keyboard, the next logical step of the process of implementation of Bluetooth touchpad on an Android device is to gather deep understanding of how exactly the Bluetooth technology works not only on the physical, but also on the higher levels. In the following chapter the Bluetooth stack architecture is discussed in detail and each abstraction level in the protocol stack, relevant for the implementation of the touchpad, is separately reviewed. Furthermore, a special attention is given to a service layer protocol- Service Discovery Protocol (SDP). The SDP is thoroughly discussed in the second subsection, since a good understanding of the protocol is required when implementing an input service via the standardized Human Interface Device protocol.

### 2.2.1. Bluetooth protocol architecture

The main goal of the Bluetooth technology is to replace the cables, connecting portable and also fixed consumer electronic devices. The head advantages of Bluetooth are outlined in the Bluetooth Core Specification as “robustness, low power, and low cost”. And furthermore, “many features of the core specification are optional, allowing product differentiation”, creating the foundation of an open standard. (Bluetooth Core Specification v2.0, Vol. 1, Part A, p. 13 [[5](#e)])

#### 2.2.1.1 Overview

Bluetooth has a layered architecture consisting of variety of protocols with different level of abstraction. The low-level core protocols are defined by the Bluetooth Special Interest Group [[1](#a)] organization. In time additional protocols from other organizations and vendor bodies have been adopted and all together have resulted in an open specification for a radio system that provides the network infrastructure to enable short range wireless communication. Although, Bluetooth stack implementations tend to vary across different vendors, protocols like LMP, L2CAP and SDP are considered mandatory and found in each stack realization. In addition, other protocols such as the HCI and RFCOMM have established as universally supported. [[5](#e)]

The Bluetooth protocol stack could be logically divided into two separate protocol stacks – the “controller stack” and the “host stack”. In general, the controller stack is implemented in low cost silicon device that contains a microprocessor and the Bluetooth radio. On the other hand, the host stack, which is responsible for the higher level protocols is either implemented as a part of the operating system (Bluez is the Linux Bluetooth software stack and is part of the Linux kernel) or is additionally installed (Widcomm is Bluetooth stack for windows developed by Widcomm Inc. and must be separately installed). The host and the controller stack are connected through the HCI pipe, providing standardized communication means between them. Alternatively, in some integrated devices such as Bluetooth mice and keyboards both host and controller stacks as well as HCI are naturally run on the same microprocessor. [[6](#f)]



Figure 4: Bluetooth stack on PC Host and HID [[6](#f)]

The graphic visualization of figure 4 illustrates scenario of communication between a PC host and HID as an example implementation and is described in the HID Specification: “The host is a personal computer and has the upper layers of the Bluetooth software running on its native processor and is connected to a Bluetooth radio module via a transport bus such as USB. The HID in this example has its firmware embedded with the radio firmware, running on the same CPU, for the lowest possible cost implementation.” (HID Specification, p. 20 [[6](#f)])

Of greater interest for the implementation of Bluetooth touchpad based on Android OS are the four lowest layers, described on figure 5 below, as well as one common service layer protocol - SDP (Service Discovery Protocol), which is not shown for clarity. In the following, the role and specifics of the low level protocols of the Bluetooth stack, as well as their core functional blocks are separately explained.



Figure 5: Bluetooth stack architecture [[5](#e)]

#### 2.2.1.2 Logical Link Control and Adaptation Protocol [[5](#e)]

The Logical Link Control and Adaptation Protocol (L2CAP) is the first layer of the host stack and thus provides connection-oriented, as well as connectionless data services to the higher level protocols by supporting protocol multiplexing, segmentation and reassembly of packets. It allows applications or upper level protocols to send and receive data packets up to 64 kB. One of the main tasks is to handle segmentation and reassembly of packets. The L2CAP layer contains two architectural blocks, which are using the L2CAP protocol to communicate - channel manager and L2CAP resource manager.

According to the Bluetooth Core Specification [[5](#e)], the channel manager is “responsible for creating, managing and destroying L2CAP channels for the transport of service protocols and application data streams”. (Bluetooth Core Specification v2.0, Vol. 1, Part A, p. 24 [[5](#e)]) In order for a L2CAP channel between two Bluetooth devices to be established, the channel managers of both devices must communicate with each other by using the L2CAP protocol and connect their endpoints to the appropriate entities. The channel managers then have to interact with their corresponding local link managers and create new logical links and configure them to provide QoS for the specific type of data, which is to be exchanged.

The functionality of the resource manager is outlined in the Bluetooth Core Specification [[5](#e)] as “managing the ordering of submission of PDU fragments to the baseband and some relative scheduling between channels to ensure that L2CAP channels with QoS commitments are not denied access to the physical channel due to Bluetooth controller resource exhaustion”. (Bluetooth Core Specification v2.0, Vol. 1, Part A, p. 24 [[5](#e)]) For instance, this behavior is required because the Bluetooth controller, implementing the controller stack, does not have infinite buffering capability and neither has the HCI pipe limitless bandwidth. Additionally the L2CAP Resource Manager is also able to “carry out traffic conformance policing to ensure that applications are submitting L2CAP service data units within the bounds of their negotiated QoS settings.” (Bluetooth Core Specification v2.0, Vol. 1, Part A, p. 24 [[5](#e)]) Since the Bluetooth data transport model assumes “well-behaved” applications, it is left for the developer to deal with this problem when implementing on the top of the Bluetooth protocol stack.

Understanding the L2CAP protocol layer has crucial importance when implementing a HID and respectively realizing the Bluetooth touchpad on Android device, since the channel connection on the L2CAP level is the highest level on which communication is taking place. Although, a deep understanding of lower abstraction level and the protocols from the controller stack is not required for the implementation, these are briefly explained, since they are the basis underneath the L2CAP communication.

#### 2.2.1.3 Host Controller Interface [[5](#e)]

The Host Controller Interface provides standardized communication between the host and the controller stacks. The interface is responsible for creating both stacks independent in such manner, that each of them could be swapped with minimal to none adaptation. There are several HCI standards, each using a different hardware interface to transfer the same data packets between host and controller stack. PCs, for example, use USB (Universal Serial Bus), mobile devices such as phones, PDAs and tablet computers, on the other hand, use UART (Universal Asynchronous Receiver Transmitter). Conclusively, the HCI transport layer provides a common device driver interface to USB, UART and others by abstracting away transport dependencies.

#### 2.2.1.4 Link Manager Protocol [[5](#e)]

A Bluetooth device’s Link Manager Protocol (LMP) is responsible for the link setup between Bluetooth units. It also carries out the authentication and encryption by generating, exchanging and checking the link and encryption keys, as well as controlling and negotiating of baseband packet sizes. There are two entities with different functional tasks using the Link Manager Protocol- Device Manager and Link Manager.

The device manager is responsible for the general behavior of the Bluetooth device by handling all the operations of the Bluetooth unit, that are not related to data transfer, such as “inquiring for the presence of other nearby Bluetooth devices, connecting to other Bluetooth devices, or making the local Bluetooth device discoverable or connectable by other devices.” (Bluetooth Core Specification v2.0, Vol.1, Part A, p. 25 [[5](#e)]) Naturally, in order to perform all this tasks the device manager has to gain access to the lower baseband layer and communicate with the Baseband Manager.

The link manager, as outlined by its name is responsible for “creation, modification and release of logical links” and same as the device manager accomplishes this task by communicating with the corresponding link manager in the other Bluetooth device by using the link manager protocol. Moreover, the Link Manager is controlling some transport attributes such as “enabling of encryption on the logical transport, the adapting of transmit power on the physical link, or the adjustment of QoS settings for a logical link.” (Bluetooth Core Specification v2.0, Vol. 1, Part A, p. 25 [[5](#e)])

#### 2.2.1.5 Link Control [[5](#e)]

The Link Control layer enables the physical RF link between Bluetooth devices to form the piconet by using both circuit and packet switching. Two important architectural blocks are located on the Baseband and Link Control layer - the baseband resource manager and the link controller.

The baseband resource manager is responsible for providing access to the radio medium and has two major functions. According to the Bluetooth Core specification it has a “scheduler that grants time on the physical channels on the physical channels to all of the entities that have negotiated an access contract.“ The other function is respectively to negotiate the access contracts with the corresponding entities on the higher level. An access contract is “a commitment to deliver a certain QoS that is required in order to provide a user application with an expected performance.” (Bluetooth Core Specification v2.0, Vol. 1, Part A, p. 25 [[5](#e)])

The Link Controller is responsible for both encoding and decoding of Bluetooth data packets on the physical channel. “The link controller carries out the link control protocol signaling (in close conjunction with the scheduling function of the resource manager), which is used to communicate flow control and acknowledgement and retransmission request signals.” (Bluetooth Core Specification v2.0, Vol. 1, Part A, p. 26 [[5](#e)])

#### 2.2.1.6 Radio Frequency [[5](#e)]

The radio frequency layer is always the lowest layer in every implementation of the Bluetooth architecture. It builds up the physical channel and is accordingly responsible for transmitting and receiving packets of data. As a result, everything in Bluetooth runs over the RF Layer, defining the requirements for the Bluetooth radio transceiver.

### 2.2.2 Service Discovery Protocol

The Service Discovery Protocol (SDP) is used to exchange information about services, which devices are providing for each other. Implementing the Bluetooth touchpad includes providing a service of mouse and keyboard input on the Android device side, which the computer side has to discover and consequently use. It is indeed, the SDP that makes it possible for the service, hosted on the phone, to be discovered by the PC host. For this reason, SDP is reviewed in detail.

#### 2.2.2.1 Overview

In the Bluetooth protocol stack SDP is bound directly to L2CAP. The L2CAP data channels are used to transfer information between two Bluetooth devices, about the services one of them is providing. In particular, client applications are enabled to discover the services, provided by server applications. Furthermore, clients can gain access to the attributes of these services, such as the type/class of service and other information needed to utilize the service. Figure 6 below shows the simplified connection scheme of SDP client and server.



Figure 6: SDP connection scheme [[5](#e)]

The SDP server maintains a list of service records. Each service record is responsible for exactly one service and contains its specifics and information needed for its utilization. A client can retrieve a single service record or all records maintained by the SDP server with a single SDP request. However, a separate connection must be established if the client decides to use the service. SDP is only responsible for discovery of services and providing the necessary information about them, but not for their utilization. For example, while the SDP request and response are exchanged via the L2CAP layer of the Bluetooth stack, a service might require a connection on higher level, such as RFCOMM.

Bluetooth devices are allowed to have only one SDP server and only one SDP client. Depending on their general purpose, some devices do not need SDP server and have only client or the opposite. If multiple applications are providing services on the same device, a single SDP server is responsible for making the service record of each service available to the SDP clients. In the same way, multiple client applications can use a single SDP client to send requests to one or more SDP servers on their behalf. [[5](#e)]

#### 2.2.2.2 Service record [[5](#e)]

In order to explain the meaning of the service record, the Bluetooth Core Specification gives a short definition of a service: “A service is any entity that can provide information, perform an action, or control a resource on behalf of another entity. A service may be implemented as software, hardware, or a combination of hardware and software.” (Bluetooth Core Specification v2.0, Vol. 4, Part B, p. 118 [[5](#e)]­) The entire information about a service, provided by a Bluetooth device, is maintained by the SDP server as service record. Furthermore, the service record presents different information aspects about the service by using service attributes. Figure 7 shows the service record’s structure.



Figure 7: Service record [[5](#e)]

#### 2.2.2.3 Record handle [[5](#e)]

For instance, an attribute mandatory for all service records is the service handle. This is a 32-bit unique within the SDP server identifier for the service record. As outlined in the Bluetooth Core Specification, “if SDP server S1 and SDP server S2 both contain identical service records (representing the same service), the service record handles used to reference these identical service records are completely independent. The handle used to reference the service on S1 will be meaningless if presented to S2.” (Bluetooth Core Specification v2.0, Vol. 4, Part B, p. 119 [[5](#e)] ­) To put it more simply, the service record handle is used to reference a specific service within a SDP server/client. This is particularly important for the task of implementing the Bluetooth touchpad. Since the Android device does not have a service record describing mouse and keyboard input service, one should be added in the SDP server, when the service is provided by certain application. When the application and correspondingly the service is not active any more, we use the predefined record handle to specify which service record, the SDP server should remove from its list.

#### 2.2.2.4 Service attribute [[5](#e)]

Each service attribute describes a unique characteristic of service and consists of attribute ID and attribute value as shown on the figure 8.



Figure 8: Service attribute [[5](#e)]

The attribute ID is precisely described in the Bluetooth Core Specification as “16-bit unsigned integer that distinguishes each service attribute from other service attributes within a service record”, and further “identifies the semantics of the associated attribute value.” (Bluetooth Core Specification v2.0, Vol. 4, Part B, p. 120 [[5](#e)] ­) Moreover, all attribute IDs for a service are specified by a service class definition, which respectively also assigns the meanings of the corresponding attribute values. Thus, the attribute value is represented as data element with variable length. In general, any type of data element is permitted for the attribute value, as long it is specified in the service class and a corresponding ID is assigned for it.

The roles of attribute ID, attribute value and service class could be easily demonstrated by the following example, shown on figure 9:



Figure 9: Attributes in different service classes

The service class A defines the value for the attribute with the ID 12345 as a string, containing the name of the service’s author. Every service, instance of A (A1, A2 …) will have an attribute with ID 12345 and its value will be specifying the author. Service, which is instance of B might also have attribute with ID 12345 that has totally different semantics.

#### 2.2.2.5 Service class [[5](#e)]

Similarly to the object-oriented philosophy each service is an instance of a service class, which provides the definitions of all attributes in the service. The attribute definition consists of unique within the service class numeric value for the attribute ID on the one hand, and description of the semantics and format of the attribute value, on the other. Although, different service classes might define different semantics for the value of attributes with the same attribute ID (as shown in the example on figure 9), there are some attributes that are common to all services. For example, the ServiceClassIDList attribute is found in every service. It represents a list, the first entity of which is the UUID. The UUID is universally unique identifier for the service that is guaranteed to be unique across all space and time.

For the most part, a service record contains attributes from several service classes that are related to each other in such manner, that each service class is a subclass of another service class. As simple as it is, “a service subclass definition differs from its superclass in that the subclass contains additional attribute definitions that are specific to the subclass.” (Bluetooth Core Specification v2.0, Vol. 4, Part B, p. 122 [[5](#e)] ­) Further following the principal of inheritance, in order to define a new service class that is a subclass of an existing service class, one must only define the additional attributes, which are specific to the new service.

The ServiceClassIDList is of significant importance, since its value is the first that is being examined when processing a service record. The ServiceClassIDList attribute of a service is a list containing the unique identifier of the service, followed by the unique identifiers of all the superclass services. The identifiers in the ServiceClassIDList attribute are listed starting from the most specific class and ending to the most general class.

The Bluetooth core specification gives an illustrative example, representing the ServiceClassIDList attribute value of a color postscript printer with duplex capability:

**DuplexColorPostscriptPrinterServiceClassID,**

**ColorPostscriptPrinterServiceClassID,**

**PostscriptPrinterServiceClassID,**

**PrinterServiceClassID**

As previously outlined, the first UUID in the list represents the unique identifier of the service in question, followed by the UUIDs of gradually becoming more general services. [[5](#e)]

# The Android Platform

In order to provide some of the necessary knowledge needed to understand how a Bluetooth touchpad is implemented on mobile phone, the previous chapter introduced the specifics of the Bluetooth technology and the characteristics of widely popular Bluetooth devices. The following chapter aims to complete the background view by providing some information about the phone, which was used in the process of implementation, and more importantly on the phone’s operating system.

## 3.1 Hardware details [[7](#g)]



Figure 10: Nexus One [[7](#g)]

The Bluetooth touchpad has been successfully implemented on the Google Nexus One phone, which is also the first Google flagship smart phone. The phone, shown on figure 10, is manufactured by Taiwan’s HTC Corporation and became available on January 5, 2010.

Currently, there is a second flagship smart phone available - the Google Nexus S and a third one is also expected to appear in the next few months. The Nexus One uses the open source Android operating system, specifically designed for a mobile phone, which is further discussed later in this chapter. As described on the phone’s official Google page it features “a large 3.7" OLED display for deep contrast and brilliant colors and a 1GHz Qualcomm Snapdragon™ chipset for blazing speeds.” (Google Nexus One [[7](#g)]) What is more important for our specific goal, the phone has a Bluetooth module capable of Bluetooth 2.0. Conclusively, since the phone has a fast processor, more than enough memory and a Bluetooth radio module, the implementation of a Bluetooth touchpad, should be absolutely possible, as far the hardware is concerned.

## 3.2 The operating system

On the home section of the official Android website, one could find a simple and elegant answer of the question what is Android. “Android is an open-source software stack for mobile devices and a corresponding open-source project led by Google. We created Android in response to our own experiences launching mobile apps. We wanted to make sure that there was no central point of failure, so that no industry player can restrict or control the innovations of any other. That's why we created Android, and made its source code open.” (Android Open Source Project [[8](#h)])

Android was initially developed by the Android Inc, which was purchased by Google Corp. in the year of 2005. Currently, Android is developed by the Open Handset Alliance- “consortium of 80 [hardware](http://en.wikipedia.org/wiki/Computer_hardware), software, and [telecommunication](http://en.wikipedia.org/wiki/Telecommunication) companies devoted to advancing [open standards](http://en.wikipedia.org/wiki/Open_standard) for mobile devices.” (Wikipedia, Android [[9](#i)]) The code of the operating system is licensed under the Apache License and furthermore easily found on the official website of the Android Open Source Project [[8](#h)].

The main goal of Android is to enable third-party developers to create applications that can utilize all the functionality the phone has to offer. Android apps can make calls, send text messages, use the camera, gain Internet access and most importantly for this particular work use the Bluetooth module. In contrast to other mobile platforms such as the Windows Phone series, all the applications are created equal. The phone’s core applications and the third-party apps have equal access to the phone’s hardware capabilities. This liberal approach seems to provide the foundations for the explosion of the innovation in the application development. Developers are able to combine different phone features in many different ways and thus provide more relevant user experience. Combining the video recording features of the phone with its network capabilities might result for example in an application that mimics the features of a 1000$ IP-Camera. The implementation of Bluetooth touchpad is just another example. The possibilities are practically limitless. According to Wikipedia [[9](#i)] by July 2011 there are at least 250 000 Android apps, that are downloaded more than 4.5 billion times. The work of the third-party developers is made easy by a full set of tools, which are built from the ground up alongside the platform and are specifically designed to provide high productivity. Furthermore, the applications are to be written mainly on the extremely popular and beloved by developers Java programming language.In the rest of this chapter the Android architecture is reviewed and special attention is spent on the components that are directly connected with the task of implementing a Bluetooth touchpad as an application on top of the Android framework. [[9](#i)]

### 3.2.1 Android system architecture[[8](#h)][[10](#j)]

The Android system architecture is a software stack, where each layer presents several programs that support different operating system functionalities. The stack as shown on figure 11 can be subdivided into 5 layers: The kernel and low level tools, shown in red; native libraries in green; the Android Runtime in yellow, the framework layer and on top of all - the applications, both in blue. In the following each layer is separately discussed.

Figure 11: Android architecture[[10](#j)]

#### 3.2.1.1 Linux kernel [[10](#j)]

The base for the Android architecture is the Linux 2.6 kernel, which is used as a hardware abstraction layer. Generally, Linux is highly portable platform, which is easy to compile on various hardware architectures. It is mostly written on C, which allows hardware manufacturers to easily port Android to large number of different devices. Furthermore, Linux provides a proven driver model and large number of existing drivers.

Clearly, another major concern for an operating system, intended to run on mobile devices is security. Android relies extremely on Linux for security, since in the last decades it has been proven to be extremely reliable. As a matter of fact, all Android apps run as independent processes with permissions set by the Linux system.

Above all, Linux provides memory management, process management, file system access, networking etc. The kernel is also modified to fulfill some special needs of the platform - such as better power management that is obviously crucial for the mobile devices. Of special interest in the context of this work are, as can be expected, the Bluetooth drivers in the Linux kernel. These are realized through Bluez, which is the standard Bluetooth software stack for Linux and is additionally discussed in this chapter.

#### 3.2.1.2 Native libraries [[10](#j)]

On top of the Linux kernel, one can find the native libraries, in green on the figure. They are written in C and C++, already compiled and preinstalled by vendor for particular hardware abstraction. Indeed, they are responsible for the core power of the Android platform. Each component provides a powerful set of functions such as database functionality through SQLite - a full-featured SQL database, a fast web rendering engine - WebKit (used in Safari, Chrome etc.), 3D graphics libraries with the OpenGL technology and many others. However, most of this functionality is not directly accessible by the developer, but is reached through Java interfaces. Components of particular interest are surely those responsible for the Bluetooth communication.

#### 3.2.1.3 Dalvik[[10](#j)][[11](#l)]

On the same level, one could also find the Android Runtime and its main component - the Dalvik virtual machine, which is actually the heart of the Android operating system. It was designed by Dan Bornstein and his team at Google specifically for the Android project. Unlike the Java virtual machine developed as universal solution for various machines, Dalvik concentrates particularly on mobile devices and the long-term constraints and challenges they present - battery life and processing power. Dalvik was built from the ground up to confront those limitations.

Another reason for implementing a new virtual machine from scratch is the licensing issue. In contrast to the Java programming language which is free, the Java virtual machine is not. Although, several open source alternative to the Java VM exist, such as the openJDK and the Apache Harmony project, the engineers from Google decided to develop a truly open-source and license-friendly virtual machine, that manufacturers could adopt and use to power variety of devices without having to worry about the license.

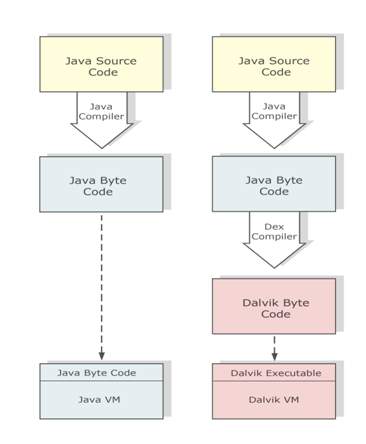


Figure 12: Dalvik vs. Java VM

Figure 12 outlines the difference between the Java VM and the Dalvik VM. When writing a Java application you have to compile your Java source code into standard Java byte code, which then runs on the Java virtual machine. Similarly, when developing an Android application, the Java source code is also compiled to Java byte code by the same compiler. However, this code is then once again recompiled with the Dex compiler to Dalvik byte could, which could be then executed on the Dalvik virtual machine. Since the Java source code is not directly compiled into Dalvik byte code, but firstly into Java byte code, positive side effects have emerged. Theoretically, the developers might write Android applications in any other language that compiles down to Java byte code. For example, they could use Python or Ruby or other popular script languages. However, in order to build the application one still needs the appropriate Java libraries that are shipped with the SDK. Nevertheless, it is very likely that the open source community present solutions to the issue in the near future.

#### 3.2.1.4 Application Framework [[10](#j)]

The application framework is outlined with blue on figure 12 and is entirely written in Java. It provides a high level access for the developer to almost all of the services of the particular device. Furthermore this is the part of the platform that is best documented and extensively covered and therefore extremely easy to use. Most of the developers are working exclusively only with the application framework, which in most of the cases is enough for building fantastic applications. The framework contains not only the common Java libraries, but also a large number of components, specifically designed for Android. There are many services and managers that provide entire ecosystem of capabilities, an application can tap into, such as location, telephony, camera and other sensors, WiFi and many others. Same as the layers previously described, for the concrete task of implementing a Bluetooth touchpad, most important are the components responsible for initialization of Bluetooth communication channels.

#### 3.2.1.5 Android Application [[10](#j)]

By using the application framework, the developer is creating applications. They are mostly written in Java, although C and C++ code could be also compiled by using an additional Development Kit, called NDK (Native Development Kit). Applications could be described as small programs running on top of the operating system. Everything the users see on an Android phone is an application. Some applications come preinstalled, for example the system management application, the application responsible for the phone calls, the phone book app etc. Third-party applications could be easily downloaded through the Android market application, which is also preinstalled. The goal if this work is also to describe, how an Android application is developed, which is capable of turning the phone into a Bluetooth touchpad.

### 3.2.2 Bluetooth in Android

In the previous chapter the Android stack has been presented and the difference between the architectural layers has been explained. As outlined above, this work aims to explain the realization of a Bluetooth touchpad as an Android application. Obviously, understanding the Bluetooth support in Android is very important for completing this task. Typically, for an application to use a hardware component such as camera, motion sensors and so on, it has to call a component from the application framework, which itself is calling functions from the native libraries, which are finally accessing the hardware over the Linux kernel drivers. As can be expected, the same is true for an application utilizing the Bluetooth module of the device. Figure 13 is very similar to the Figure 11, which is visualizing the Android stack. However, instead the different libraries situated on the stack layers, Figure 13 shows separate components on each respective level, responsible for the Bluetooth support in Android.

The Bluetooth touchpad application is operating on the application level and is handling the user interactions, which are then processed and the respective formatted data is to be send to the computer over an established Bluetooth channel. When sending the data the application is to call functions from the application framework, especially from the BluetoothSocket component. The Bluetooth components situated on the application framework layer are implemented in Java. Nevertheless, central functions of these components are not implemented, but only defined through the Java Native Interface (JNI) technology. They are the bridge between the application framework layer and the native libraries underneath. The native components supporting Bluetooth are implemented entirely in C++ and C and therefore much faster. To put it more simply, the Java components could be viewed as a shell or as a facade, which is intended to ease the work of the developer. The actual work is then done by the native libraries under the facade, which are implementing the real logic. Furthermore, they are accessing the physical Bluetooth module directly through the Bluetooth stack, implemented as a part of the Linux kernel, the lowest level on the figure 13. Bluez is the standard Bluetooth stack for Linux. As of 2006, the [BlueZ](http://en.wikipedia.org/wiki/BlueZ) stack supports all core Bluetooth protocols and layers, that were described in detail in a previous chapter. It was initially developed by [Qualcomm](http://en.wikipedia.org/wiki/Qualcomm), and is available for [Linux kernel](http://en.wikipedia.org/wiki/Linux_kernel) versions 2.4.6 and up.

Figure : Bluetooth support in Android

# Service implementation

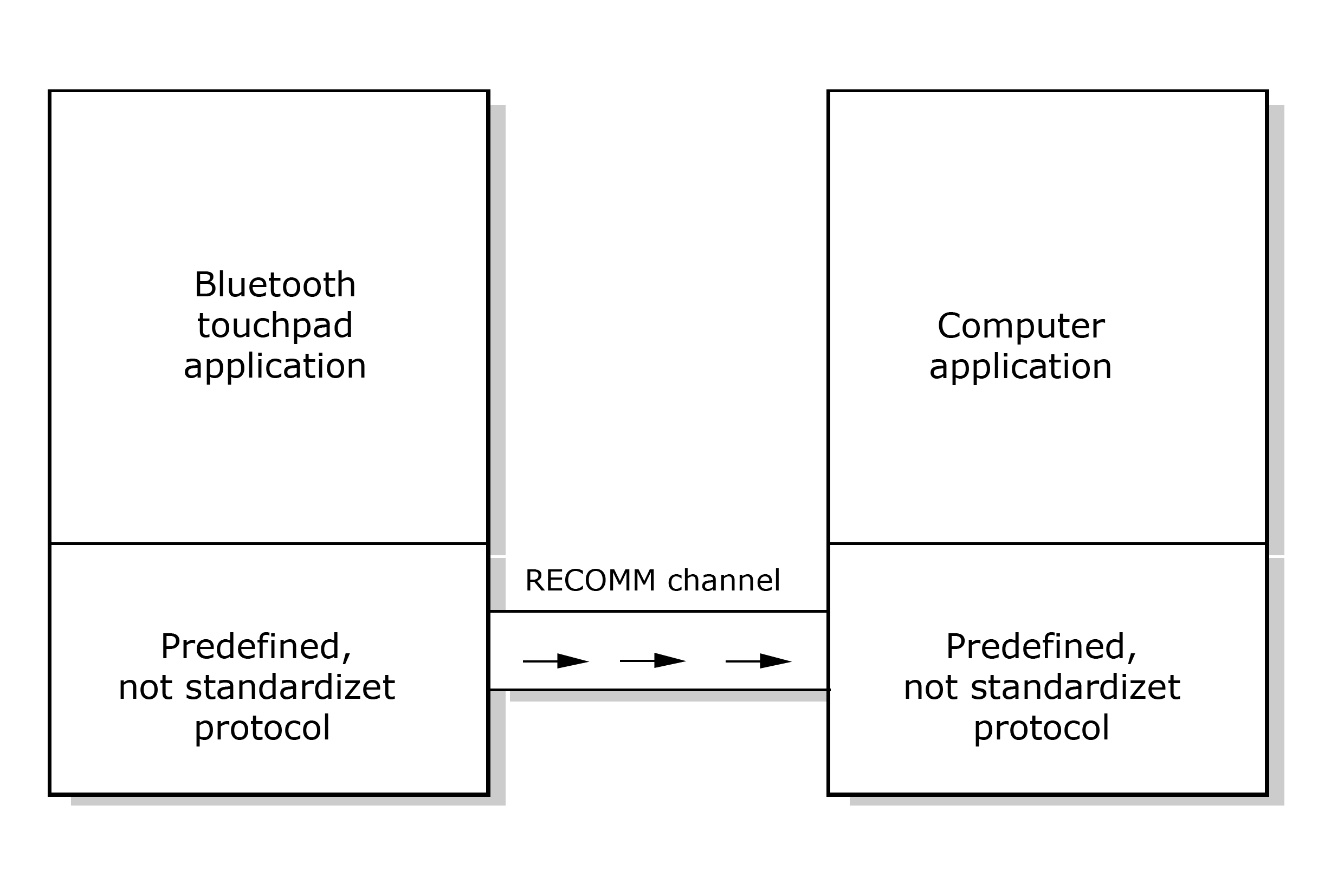
A possible realization of Bluetooth touchpad running as an application on Android mobile device is visualized on the figure 14.

Figure 14: Bluetooth client/server concept

The Bluetooth touchpad app is processing the user input and sending the respective data to application running on the computer. The data is sent through a RFCOMM channel, which is high level Bluetooth protocol, usually used for transferring data between applications. However the data is not encoded in any standard format, but is encoded in format designed by the developer. This method is implying several obvious disadvantages. First, the developer has to implement additional application to run on the computer side. Second, the developer has to define a protocol, which the Bluetooth touchpad application and the computer application will be using to communicate. Third, the user will not be able to start using the touchpad simultaneously, but has to install and run the computer application first. Nevertheless, despite these limitations and the worse user experience, this method offers a very simple solution to the problem. As can be expected, there are already several realizations of applications using similar architectural models in the official Android market.

The implementation of the Bluetooth touchpad, which is described in detail in the following chapter, takes totally different engineering approach. Instead of developing an application for the computer side, we take advantage of the Human Interface Device drivers, which are standard part of the Bluetooth stacks of the most modern operating systems. The same drivers are actually used, when connecting simple Bluetooth mouse or keyboard. As a matter of fact, the computer will be then identifying a Bluetooth desktop device - consisting of both mouse and keyboard. The following chapter will present one by one the different problems and their respective solutions in a logical order. The goal is to form a detailed view that can fully explain how the Bluetooth touchpad service record is created and used.

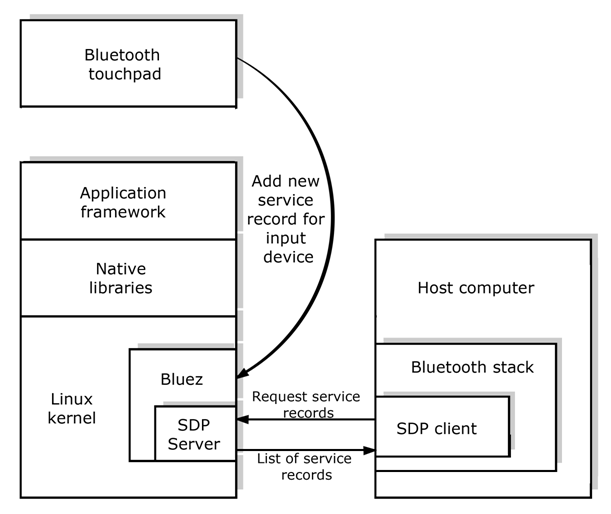
Implementing a Bluetooth touchpad means creating a service or more precisely an input service for mouse and keyboard that the computer is able to utilize. But before starting to use the input service, the computer has to find it and read its description in order to know how to use it properly. This process has been already clarified in the second chapter, describing how the Service Discovery Protocol works. The SDP client running as a part of the Bluetooth stack of the computer is connecting to the SDP server running as a part of the Bluetooth stack of the input device (in our case the Android device) and is then requesting a particular or all the service records, which are actually descriptions of the services, the device is able to provide for the computer. This interaction is shown by the Figure 15 below:

Figure : Adding a new service

As previously described in the third chapter, developers are supposed to use the application framework in order to access the low levels of the Android operating system. APIs of the application framework provide access to the native libraries, which are then directly calling the low level functions of the Linux kernel. The abstraction provided through this method is certainly responsible for making the process of implementing very developer-friendly. The task of adding a new service record would not be a problem, if a Java library form the application framework would provide the needed functionality. Unfortunately, there is no such library in the application framework and in the native libraries either. However, on the lowest level of Android, the Linux kernel is running. As shown on the figure 15 above, the only possibility to add the service record in the SDP server is to deploy and run a Linux executable, which then adds the new service record by directly calling functions from the Bluetooth software stack in the kernel - Bluez. Thus, the new service will be present, when the host computer is requesting the list of service records. Accordingly, three tasks must be taken into consideration. First thing to do is writing the native C code, which would create and add the new service record to the SDP registry by interacting with Bluez. Second, the native C code must be compiled. Third, the resulting Linux executable must be deployed and executed. In the following, these tasks are solved and the solution is described in detail, since it is the most sophisticated and important part of this work.

4.1 Bluez API and sdptool

Bluez is a powerful communication stack for Linux that also contains extensive API, which allows the developer to fully exploit the Bluetooth resources of the particular devices. Unfortunately, an official documentation of this API does not exist. Besides, there is very little unofficial documentation. The best tutorial, which shows some basic functionalities of the API is “Chapter 4: Bluetooth programming in C with BlueZ” of the “An introduction to Bluetooth programming” [[12](#m)], written by Albert Haung. The only thing, a developer could do is to figure out the API by reading through the Bluez source code. However, instead examining the whole source code of the framework, it is far more reasonable to examine a component, which is written on top of the API and is calling its functions. Such component is the “sdptool”. The sdptool is a Linux command line tool, which provides an interface for performing SDP queries on Bluetooth devices and more importantly for our particular goal - the functionality of adding several predefined service records in the SDP registry. The tool is part of Bluez and therefore open source. Its code could be also found in the Android project and is located in file: ”**<android\_source>/external/bluetooth/bluez/tools/sdptool.c**”. Two functions in the C source code are particularly interesting: **add\_hid\_keyb** and **add\_hid\_wiimote**. They are both creating a new service record by using HID, the human interface device protocol. And they are both adding it to the SDP registry. Obviously, the first is creating an input service for keyboard and the latter an input service for the Wii remote control. Although far not as comfortable as API documentation or tutorial, the examination of this functions provides some of the skills needed to implement the service.

## 4.2 Defining the service record

A service record, as previously described in the chapter about SDP, is a list of attributes, which describes the specific service. Defining the service record is simply putting together this list of attributes. Some attributes are described in this section and others, which are expected by the Human Interface Device protocol, are described in more detail in the next section. In the following, we give up of presenting code examples, but explain each attribute that is part of the implementation separately. The explanations are extremely useful for understanding the source code and therefore referenced in the code’s documentation.

### 4.2.1 ServiceClassIDList attribute

As previously explained this is a list containing the UUIDs of the service class and all its superclasses. An UUID is universally unique identifier of a service class. Bluez provides curtain functionality, so the developer has to set only the UUID of the HID service class as first entity of the list. The framework then automatically builds the list by iteratively adding the superclasses, starting with the UUID, added by the developer and ending with the UUID of the last most general class.

### 4.2.2 ServiceRecordHandle attribute

The record handle is unique ID within the SDP server. It is used to reference the service record in the registry. In our particular implementation the service is removed from the registry, when the Bluetooth touchpad application, which is providing it, is closed. The record handle then comes into hand by pointing which service has to be deleted. The record handle is a 32-bit unsigned integer, which could be defined by the developer in a random fashion. Thus, it is important not to choose an integer, which is already occupied by another record.

### 4.2.3 BrowseGroupList attribute

The process of searching for services by the SDP client is also called browsing and is actually based on the BrowseGroupList attribute, which similarly to the record handle and the ServiceClassIDList attribute is common to all services. The BrowseGroupList actually contains UUIDs, which are not representing service classes, like in the ServiceClassIDList attribute, but browse groups. A service could be associated with one or more browse groups. When a client is searching for services, it is usually searching for the services, which are members of the root browse group. Moreover, a service is a member of certain browse group, when its BrowseGroupList contains the group’s UUID value. Bluez provides functionality to define the attribute with the root browse group’s UUID as part of its list. Naturally, we take advantage of this functionality to make the service public, so the SDP client of the computer host can find it.

### 4.2.4 LanguageBaseAttributeIDList and human-readable attributes

SDP provides a method for supporting multiple languages for human-readable attributes, whose values are strings, which are presented directly to the user. There are also universal human –readable attributes, which are common to all services. The ServiceName attribute contains the name of the service, which should be brief and suitable for displaying under an icon, representing the service. It has an ID offset of 0x0000. The ServiceDescription attribute contains a description of the service, which must be less than 200 characters in length. Its ID offset is 0x0001. The ProviderName attribute contains the name of the organization providing the service and has an ID offset of 0x0002. The ID offsets are part of the mechanism for supporting multiple languages, which is further explained.

The LanguageBaseAttributeIDList is list containing at least one member attribute. Each of this member attributes is responsible for supporting a single language. The structure of the list and the member attributes is shown below on figure 16.

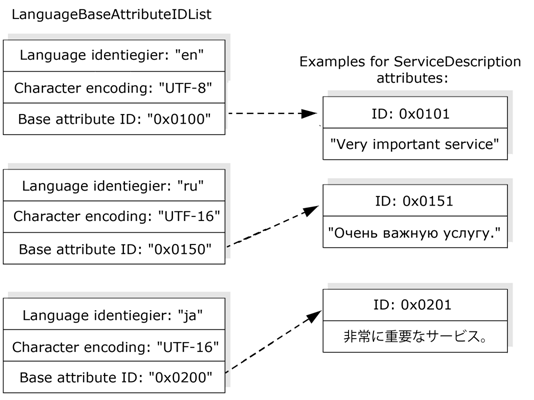
Each member is actually a triplet containing language identifier, character encoding and attribute ID. The first element of the triplet is the language identifier that indicates the natural language according to ISO 639:1988 (E/F): “Code for the representation of names of languages” [[13](#n)]. The figure shows, that the three attribute members of the LanguageBaseAttributeIDList are presenting English, Russian and Chinese. The members also have an element that is indicating the character encoding, used for the natural language. The last element of the triplet is the base attribute ID for the member. The base attribute ID value of the first member attribute of the LanguageBaseAttributeIDList must be set to 0x0100 by definition and represents the primary language, supported by the service. The base attribute ID is then used by the human-readable attributes and their particular ID offsets. This method is also explained on figure 16, by the example service descriptions in three different languages. To encode a ServiceDescription attribute its ID must be matched with a base attribute ID of member attribute in the LanguageBaseAttributeIDList. The ServiceDescription attribute has an ID offset of 0x0001. So if we take the second attribute as an example, which has an ID of 0x0151 and ID offset of 0x0001, the entity in the LanguageBaseAttributeIDList that should be used for decoding has a base attribute ID of 0x0150. This entity is then indicating that the UTF-16 character encoding was used to encode the ServiceDescription attribute’s value, which contains information written in the Russian natural language. Although, in our particular implementation, the LanguageBaseAttributeIDList has a single element that is presenting English as a primary language and simple UTF-8 character encoding, the system of multiple language support for human-readable attributes has been explained, since it presents an elegant solution of an interesting problem. Accordingly, the implementation also includes the ServiceName, ServiceDescription and ProviderName attributes, which are encoded with UTF-8 and present information written in English. Future versions of the touchpad implementation might support other languages.

Figure : LanguageBaseAttributeIDList structure

### 4.2.5 ProtocolDescriptorList

Before a service is utilized, a communication should be established between the providing and the using party. The ProtocolDescriptorList attribute describes one or more communication protocol stacks that are used to gain access to the service, described by the service record. The ProtocolDescriptorList consists of data elements known as protocol descriptors. The protocol descriptor itself is also a list of data elements. The first element is the UUID that is naturally responsible for identifying the particular protocol. The other elements are protocol-specific parameters, such as protocol version, connection port etc. The protocol descriptors that are used when accessing the service are listed in the ProtocolDescriptorList by starting with the lowest protocol in the stack and ending with the highest protocol. Figure 17 is visualizing a ProtocolDescriptorList example for IrDA-like printer:

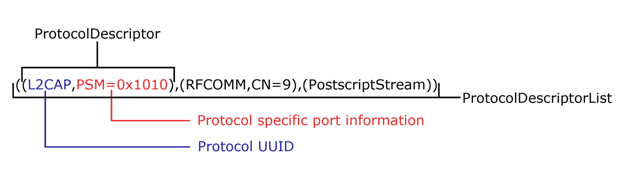


Figure 17: ProtocolDescriptorList of IrDA-like printer

The ProtocolDescriptorList consists of four protocol descriptors, starting with the lowest in the stack, which is identified by its UUID as the L2CAP protocol. The protocol descriptor also contains protocol specific information about the communication port. The Protocol Service Multiplexor (PSM) is playing a role similar to the port number in TCP/IP communication. In the description of the service for the Bluetooth touchpad, we implement a ProtocolDescriptorList, that has two pairs of protocol descriptors, both describing communication channels via L2CAP on the lower level and HID on the higher.

**((L2CAP, PSM=0x11), (HID), (L2CAP, PSM=0x13), (HID))**

A reason for this is that the Bluetooth touchpad service, which is actually a HID service for input devices, demands the establishment of control and interruption communication channels. They are using respectively the 0x11 and 0x13 PSM ports on the L2CAP level.

### 4.2.6 BluetoothProfileDescriptorList

The BluetoothProfileDescriptorList similarly to the ProtocolDescriptorList is a list of data elements called ProfileDescriptors. The ProfileDescriptor itself is pair of two specific data elements, containing information about a Bluetooth profile to which the service described in the service record conforms. As can be expected, the first element is the UUID of the Bluetooth profile. The second element is 16-bit unsigned integer indicating the profile’s version. The first 8 bits are responsible for the major version and the last 8 bits for the minor version. The initial version is actually a major version 1 and minor version 0. In the Bluetooth touchpad implementation the BluetoothProfileDescriptorList has a single profile descriptor with UUID indicating the Bluetooth HID Profile and version set to initial. This is actually the up to date version of the HID specification, which is also used when implementing the touchpad. Future versions of the HID Profile must conform to the Bluetooth Core Specification [[5](#e)], according to which: “When upward compatible changes are made to the profile, the minor version number will be incremented. If incompatible changes are made to the profile, the major version number will be incremented.” (Bluetooth Core Specification v2.0, Vol. 3, Part B, p. 150 [[5](#e)])

## 4.3 Human Interface Device

The Universal Serial Bus (USB) Specification presents a concept of grouping devices with similar data reporting characteristics into devices classes and present a single driver for the whole group. Furthermore, the devices are capable of describing themselves to the class driver and inform what controls they have and how they report data. This method makes it possible for future devices to be developed without the need to modify the existing host driver software. HID is one of the most popular USB device classes and presents “powerful and extensible data reporting system that was designed specifically for the needs of Human Interface Devices.” (BT HID Document).

In the previous section we started to describe the implementation of HID service record. The HID service description is of high importance for the realization of the Bluetooth touchpad. The figure x.x below outlines the role, which it plays in the implementation.   
  
// Figure x.x

After the HID service record is implemented and deployed it is already part of the service registry of the SDP server in Android. When the SDP client of the computer host requests the available services, which the Android device provides, the SDP server will respond with the records of those services. Accordingly, the SDP client will recognize the HID service record and will send it to the HID host driver. The HID Host driver is able to parse the HID service record, which contains information about the service provider, the service provider’s behaviour, the encoding and the meaning of the data packets the service provider is sending. Then, the HID Host driver is able to establish wireless connection to the Bluetooth touchpad over the L2CAP channel. The Bluetooth touchpad can start sending data packets- reports containing information about user interaction events, for example a click of the left mouse button. Thanks to the descriptions in the HID service record, the HID Host driver can understand the reported data and notify the operating system about the event.   
  
Figure x.x

As show on the figure x.x above,the HID service record is actually a list of service attributes. In the previous section we described some universal attributes that are part of the HID service record, but also common to all service records. In this section the service attributes are described, that are specific to the HID protocol. Once again, we give up of presenting code samples, but explain all the attributes that are part of the implementation. Hence, the explanations are very important for the implementation of the Bluetooth touchpad and therefore referenced in the native source code.

Glossary

**Human Interface Device**

**Application Programming Interface** An Application Programming Interface (API) is an

interface implemented by a software program that enables it to interact with other

software. It facilitates interaction between different software programs similar to the

way the user interface facilitates interaction between humans and computers.

**Android**

**Bluetooth**

**Service Discovery Protocol**

**Radio frequency**

**Bluez**

**Linux**

**Widcomm**

HID Human Interface Device

API Application Programming Interface

L2CAP Logical Link Control and Application Protocol

SDP Service Discovery Protocol

RF Radio frequency

HCI

QoS Quality of serive

RFCOMM

# Bibliography

x

|  |  |
| --- | --- |
| [1] | Bluetooth Special Interest Group. (2011) [Online]. <http://www.bluetooth.com> |
| [2] | Bluetooth Special Interest Group. (2011) Bluetooth Basics. [Online]. <http://www.bluetooth.com/Pages/Basics.aspx> |
| [3] | Seguridad Mobile. (2011) Bluetooth Security Mechanisms. [Online]. <http://www.seguridadmobile.com/bluetooth/bluetooth-security/security-mechanisms.html> |
| [4] | MindTree Consulting Anindya Bakshi. (2007, Dec.) Bluetooth Secure Simple Pairing. [Online]. <http://www.wirelessdesignmag.com/PDFs/2007/1207/wd712_coverstory.pdf> |
| [5] | Bluetooth Org. (2004, Nov.) Bluetooth Core Specification v2.0. [Online]. <https://www.bluetooth.org/Technical/Specifications/adopted.htm> |
| [6] | Bluetooth Org. (2003, May) HID Specification v. 1.0. [Online]. <https://www.bluetooth.org/Technical/Specifications/adopted.htm> |
| [7] | Google Inc. Google Nexus One. [Online]. <http://www.google.com/phone/detail/nexus-one> |
| [8] | Android Project. (2011) Android Open Source Project. [Online]. <http://source.android.com/> |
| [9] | Wikipedia. (2011) Android. [Online]. <http://en.wikipedia.org/wiki/Android_(operating_system)> |
| [10] | Android Project. (2011) Android Developers Website. [Online]. <http://developer.android.com/> |
| [11] | David Ehringer. The Dalvik Virtual Machine Architecture. [Online]. <http://davidehringer.com/software/android/> |
| [12] | Bluez Project. (2010) Bluez. [Online]. <http://www.bluez.org/> |

x