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:	Nils T. Kannengießer, M.Sc.
Submission 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 Nils Kannengießer, M.Sc. for his outstanding support, supervision, useful suggestions and encouragement which were 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 at the TUM as well. Thanks to my friends Marin Bukov and Christoph Mahlert for helping me correct the grammar in this work.

Furthermore, I would like to thank my family for the general education and the 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.

Initially, 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 smartphones and other mobile devices. Those running this freely distributed OS are less expensive than other ones with similar hardware specifics. For its openness, user-friendly concepts and developer-friendly software development tools, it has become 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 erfüllen. Deswegen finden sie auch in vielen Anwendungsfällen einen Platz. Diese Arbeit konzentriert sich auf die Realisierung eines solchen Anwendungsfalls und präsentiert eine völlig neue Art und Weise, wie die Fernsteuerung von Notebooks, Rechnern und anderen 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 auf ein Android-Handy läuft, kann sich mit anderen Systemen mit Hilfe der Bluetooth-Technologie verbinden. Durch die Verwendung einer Reihe von Standard-Treibern bietet die Anwendung dem Nutzer zahlreiche Eingabemöglichkeiten, die nicht weniger mächtig sind als diese, die von üblichen Geräten wie Maus und Tastatur angeboten werden. Allerdings erfüllt das Projekt nicht nur dieses Ziel. Es wird ein Framework vorgestellt, welches 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.

Das Open-Source-Betriebssystem Android hat sich in den letzten Jahren als das meistgenutzte Betriebssystem etabliert, welches speziell für mobile Geräte entwickelt ist. Geräte, auf denen 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 die Wahl des Betriebsystemes für dieses Projekt auf Android gefallen.

Contents

List of Figures

List of Tables

1. 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, which 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 capable 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 methods 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.

2. 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 for implementing 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. Consequently, 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 features of regular corded mouse and keyboard, as well as a graphics pad device are successively discussed, since the goal of this project is also to introduce possibilities for supporting the functionality of all three devices. Next, 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 display.

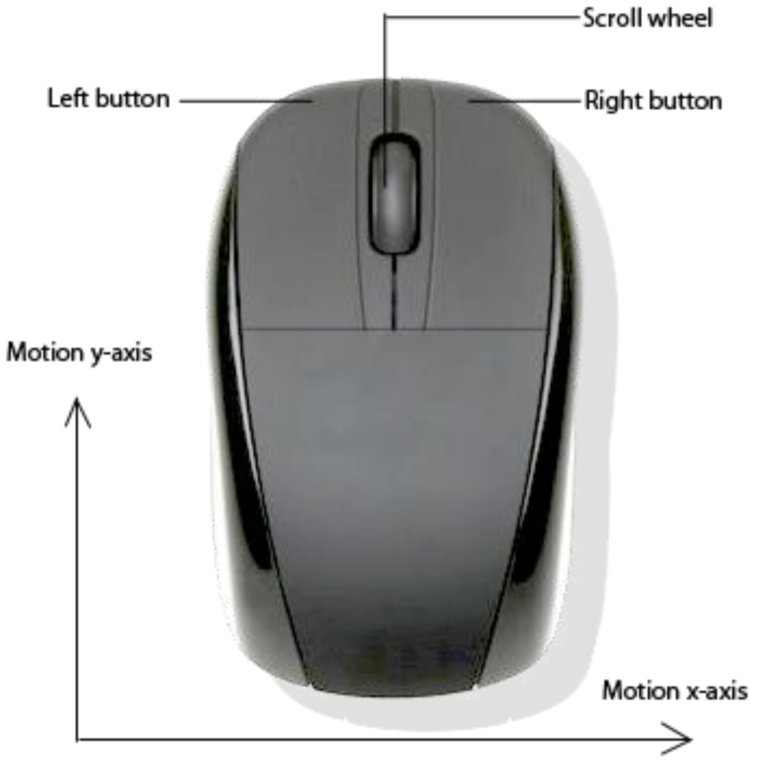


Figure 1: Mouse functionality

As shown on the figure 1 above, the mouse functional capabilities could shortly be 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 design originates 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 shows the concept of the device.

Figure 2: 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. The service provided by the graphics pad is actually the service of a mouse, which is sending data representing absolute screen coordinates instead of relative values. There are also other devices taking advantage of the same type of service. In this work, we will refer to this class of devices as pointer devices.

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 to send 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. It is shortly described on the main page of its vendor - the Bluetooth Special Interest Group [1] 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]) 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. Consequently, one Megahertz is one million and one Gigahertz (GHz) is one billion cycles per second. [3]

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. As a result, pairing makes it possible to filter out interference from other RF devices. There are several methods of pairing, depending on 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]

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 than an ordinary Bluetooth mouse and keyboard, since it introduces the possibility for numeric comparison as part of the Android operating 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] 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]

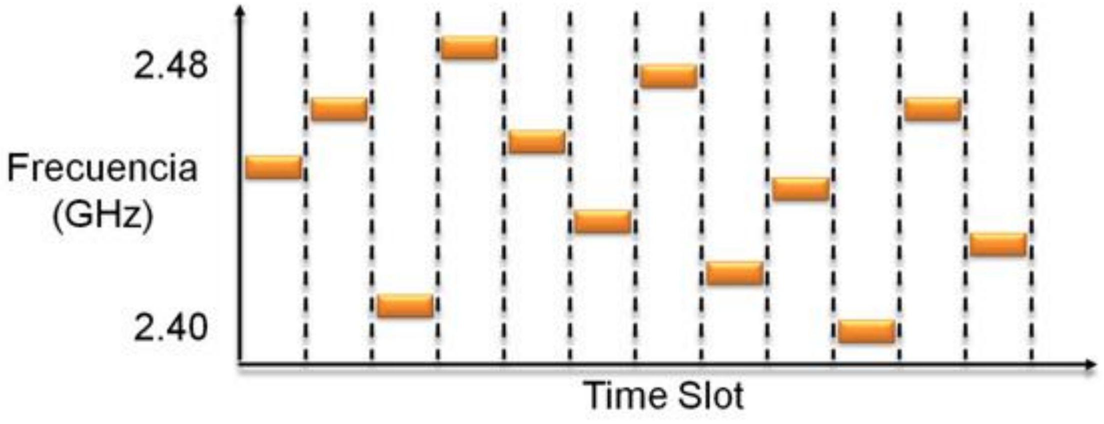


Figure 3: Frequency hopping scheme [3]

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 acquire deep understanding of how exactly the Bluetooth technology works not only on the physical, but also on the software level. 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, special attention is paid 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])

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] organization. Later on, 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]

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]

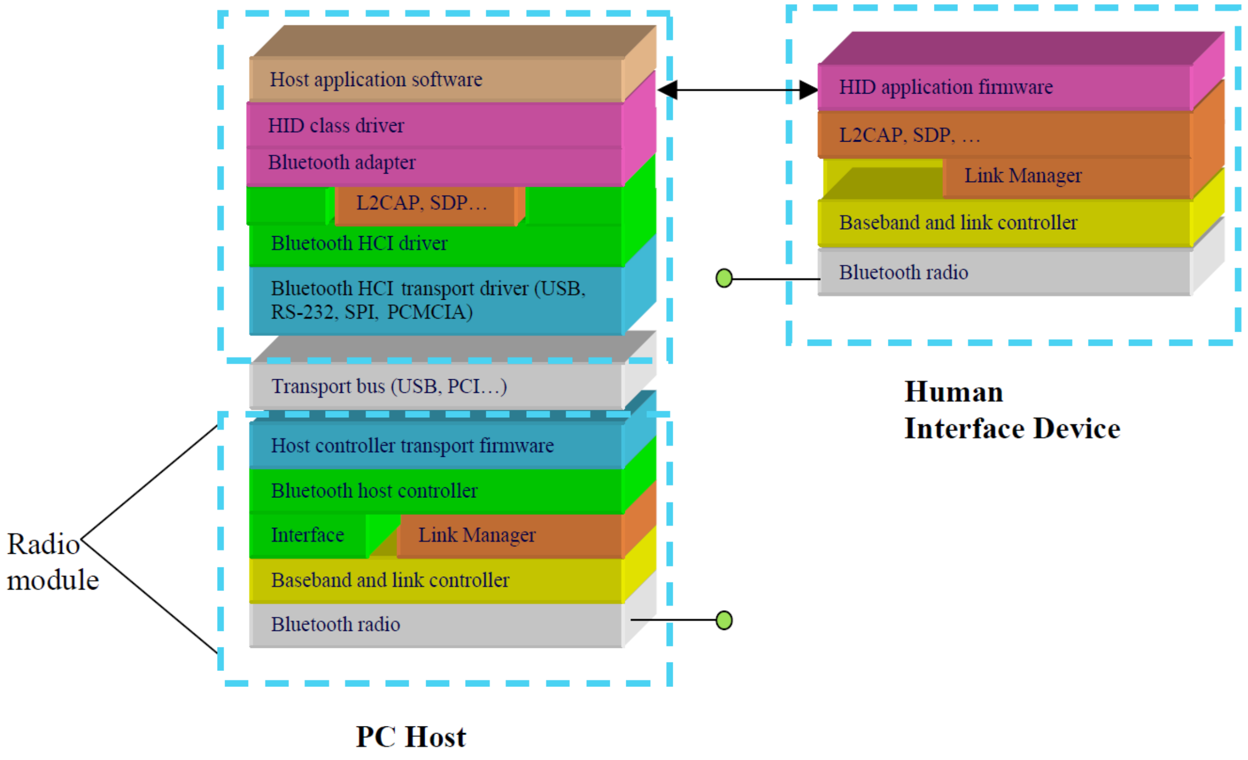

Host application software
HID class driver
Bluetooth adapter
L2CAP, SDP…
Bluetooth HCI driver
Bluetooth HCI transport driver (USB, RS-232, SPI, PCMCIA)
Transport bus (USB, PCI…)
Host controller transport firmware
Bluetooth host controller
Interface
Link Manager
Baseband and link controller
Bluetooth radio
Radio module
PC Host
HID application firmware
L2CAP, SDP, …
Link Manager
Baseband and link controller
Bluetooth radio
Human Interface Device

Figure 4: Bluetooth stack on PC Host and HID [6]

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])

Of greater interest for the implementation of Bluetooth touchpad based on Android OS are the four lowest layers, described in 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.

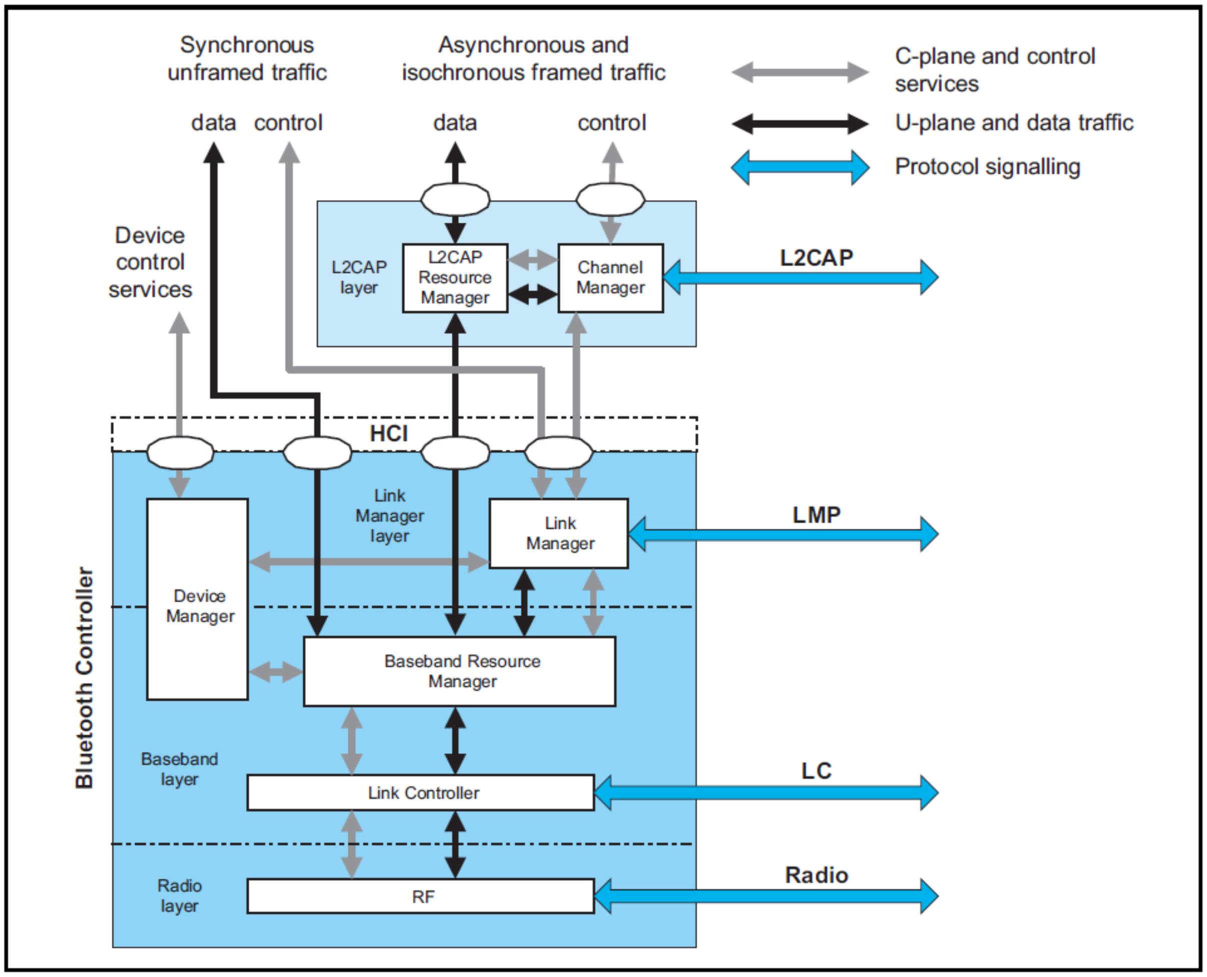


Figure 5: Bluetooth stack architecture [5]

2.2.1.2 Logical Link Control and Adaptation Protocol [5]

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], 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]) 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] 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]) 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]) 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 is crucial for 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 levels and the protocols from the controller stack is not required for the implementation, these are briefly explained, since they form the basics of the L2CAP communication.

2.2.1.3 Host Controller Interface [5]

The Host Controller Interface provides standardized communication between the host and the controller stacks. The interface is responsible for creating both stacks independent in a manner such 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). Consequently, 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]

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 the 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]) Naturally, in order to perform all this tasks the device manager has to access 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 similarly to 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])

2.2.1.5 Link Control [5]

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 in 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 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])

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])

2.2.1.6 Radio Frequency [5]

The radio frequency layer is always the lowest layer in every implementation of the Bluetooth architecture. It builds up the physical channel and is responsible for transmitting and receiving packets of data, respectively. 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 a service of mouse and keyboard input on the Android device side, which the computer side has to discover and subsequently 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.

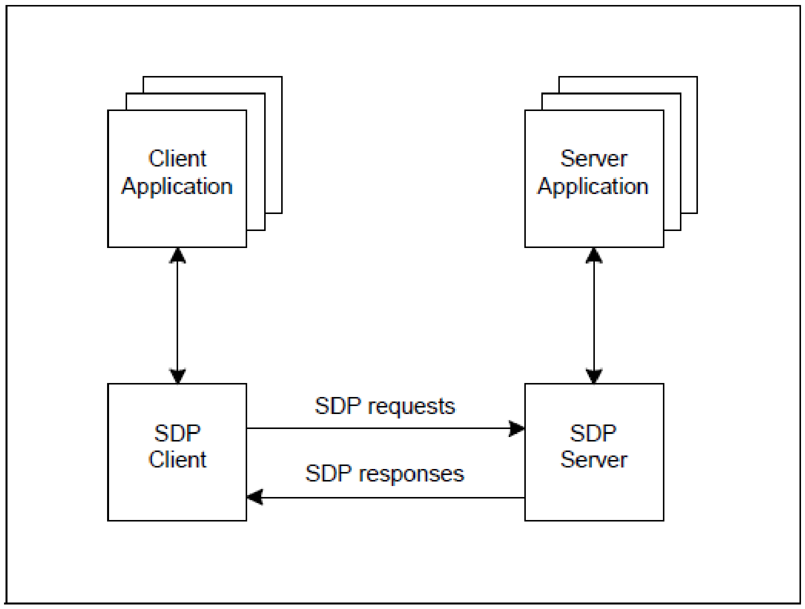


Figure 6: SDP connection scheme [5]

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]

2.2.2.2 Service record [5]

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]) 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 attributes. Figure 7 shows the service record's structure.

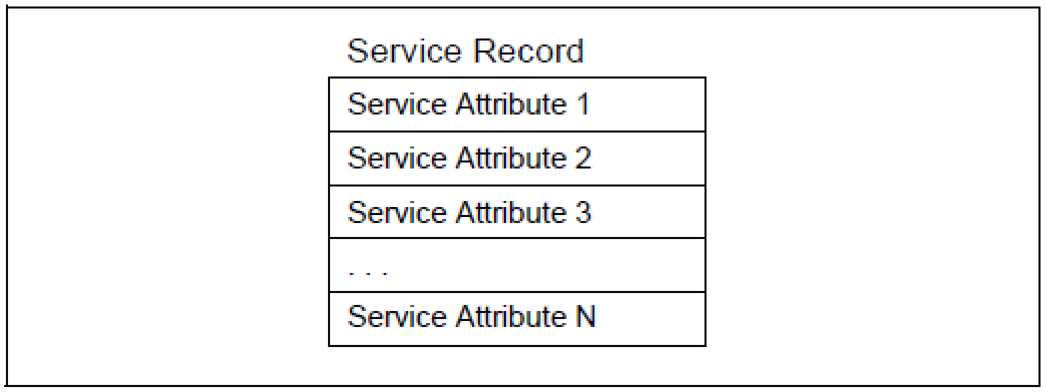


Figure 7: Service record [5]

2.2.2.3 Record handle [5]

An example of 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]) To put it more simply, the service record handle is used as a reference to 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]

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

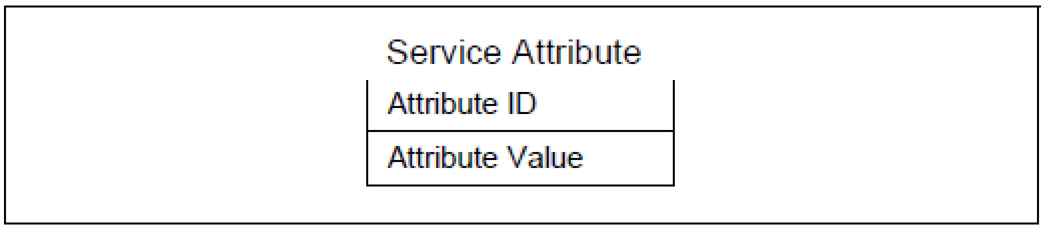


Figure 8: Service attribute [5]

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]) 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:

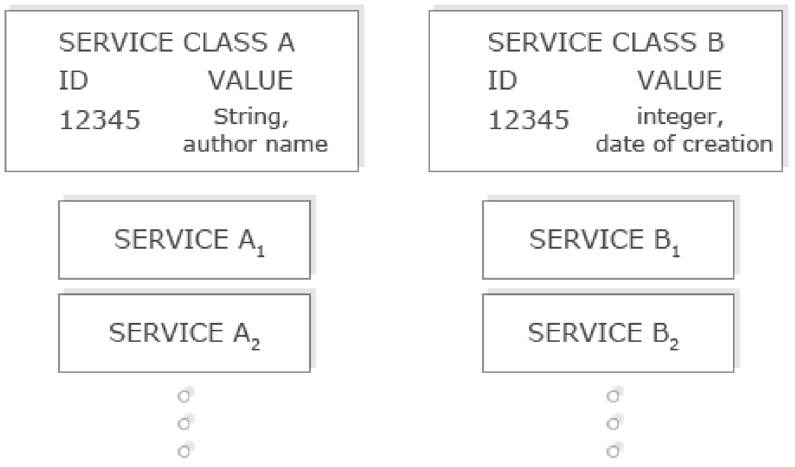


Figure 9: Attributes in different service classes

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]

Further following 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]) 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 with 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]

3. The Android Platform

The previous chapter introduced the specifics of the Bluetooth technology and the characteristics of widely popular Bluetooth devices in order to provide some of the necessary knowledge needed to understand how the Bluetooth touchpad has been implemented on a mobile phone. The following chapter aims to complete the foundation of knowledge by presenting information about the phone, which was used in the process of development, and more importantly on the phone's operating system.

3.1 Hardware details [7]

Figure 10: Nexus One [7]

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, which was specifically designed for a mobile phone and 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]) 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 to the question what Android is: *"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])

Android was initially developed by the Android Inc, which was purchased by Google Corp. in the year of 2005. Presently, Android is developed by the Open Handset Alliance – *"group of 84 technology and mobile companies who have come together to accelerate innovation in mobile and offer consumers a richer, less expensive, and better mobile experience."* (Open Handset Alliance [9]) The code of the operating system is licensed under the Apache License, version 2.0. It could be easily found on the official website of the Android Open Source Project [8].

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, access the Internet and most importantly for this particular work use the Bluetooth module. In contrast to other mobile platforms such as the Windows Phone series, all applications are created equal. The preinstalled core apps and the third-party applications have equal access to the phone's hardware capabilities. This liberal approach seems to provide the fuel for the explosion of innovation in the application development. Developers are able to combine different phone hardware features in many different ways and thus provide more relevant user experience. Combining the video recording features of the phone with its network connectivity might result for example in an application that mimics the features of a 1000$ IP-Camera. The realization of Bluetooth touchpad is just another example. The possibilities are practically limitless. The work of third-party developers is made easy by a full set of tools, which have been built 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].

3.2.1 Android system architecture [8] [10]

The Android system architecture is a software stack and 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.

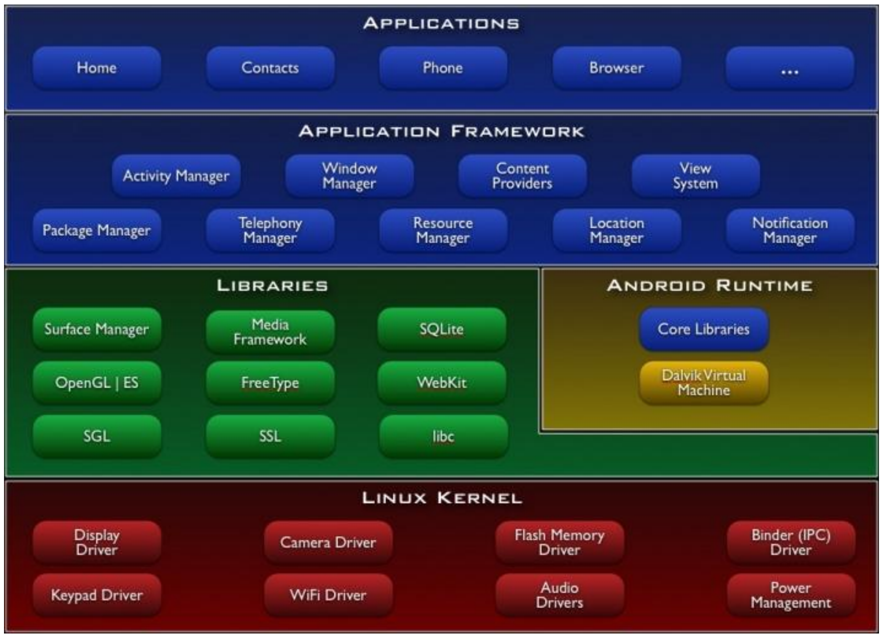


Figure 11: Android architecture [10]

3.2.1.1 Linux kernel [10]

The base of the Android architecture is the Linux 2.6 kernel, which is used as a hardware abstraction layer. Generally, Linux is highly portable platform, easy to compile on various hardware architectures. It is mostly written on C and therefore 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 that has been proven to be extremely reliable in the last decades. 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 battery powered 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 - the standard Bluetooth software stack for Linux. It is additionally discussed later in this chapter.

3.2.1.2 Native libraries [10]

On top of the Linux kernel, one could find the native libraries in green on the figure 11. 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 presents 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 could not be directly accessed by the developer, but is reached through Java interfaces. Components of particular interest in this work are of course those responsible for the Bluetooth communication.

3.2.1.3 Dalvik [10] [11]

On the same architectural 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 needs of the Android project. Unlike the Java virtual machine, developed as universal solution for various machines, Dalvik concentrates explicitly 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 these 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 and license-friendly virtual machine that manufacturers could adopt and use to power variety of devices without having to worry about the license.

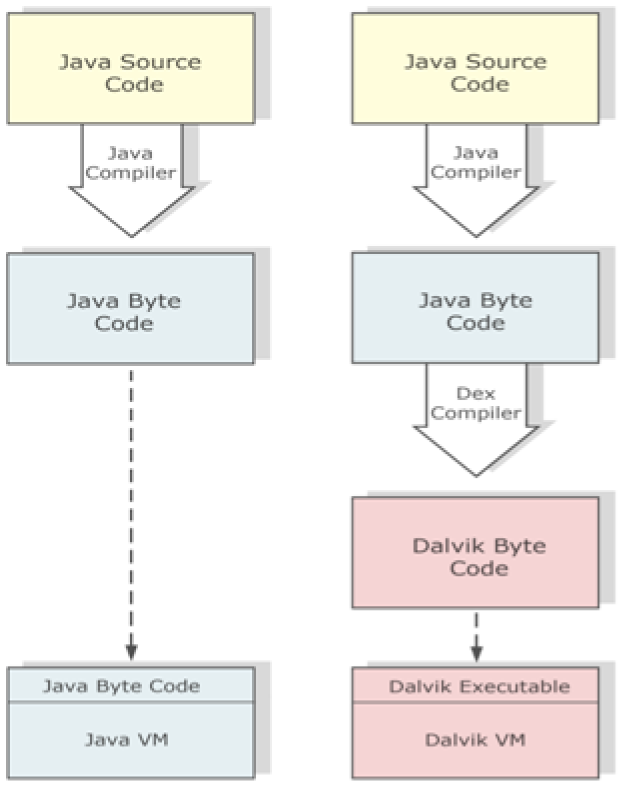


Figure 12: Dalvik vs. Java VM

Figure 12 outlines the differences 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 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 PHP, 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 presents some solutions to the issue in the near future.

3.2.1.4 Application Framework [10]

The application framework is outlined with blue on figure 12 and is entirely written in Java. It provides high level access for developers to almost all the services of the particular device. Furthermore, this part of the platform 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 large number of components, designed specifically for Android. There are many services and managers that build up entire ecosystem of capabilities, such as location, telephony, camera and other sensors, WiFi and many others. Same as the layers previously described, most important for the concrete task of implementing a Bluetooth touchpad, are the components responsible for initialization of Bluetooth communication channels.

3.2.1.5 Android Application [10]

In general, the developer is creating applications by implementing on top of the application framework. Apps 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). 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 that is also preinstalled. The goal if this work is also to describe the process of development of an Android app, which is capable of turning the phone into Bluetooth touchpad.

3.2.2 Bluetooth in Android

In the previous subsection 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 app 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 that 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, visualizing the Android stack. However, instead the different libraries situated on the stack layers, Figure 13 shows separate components on each level, responsible for the Bluetooth support in Android.

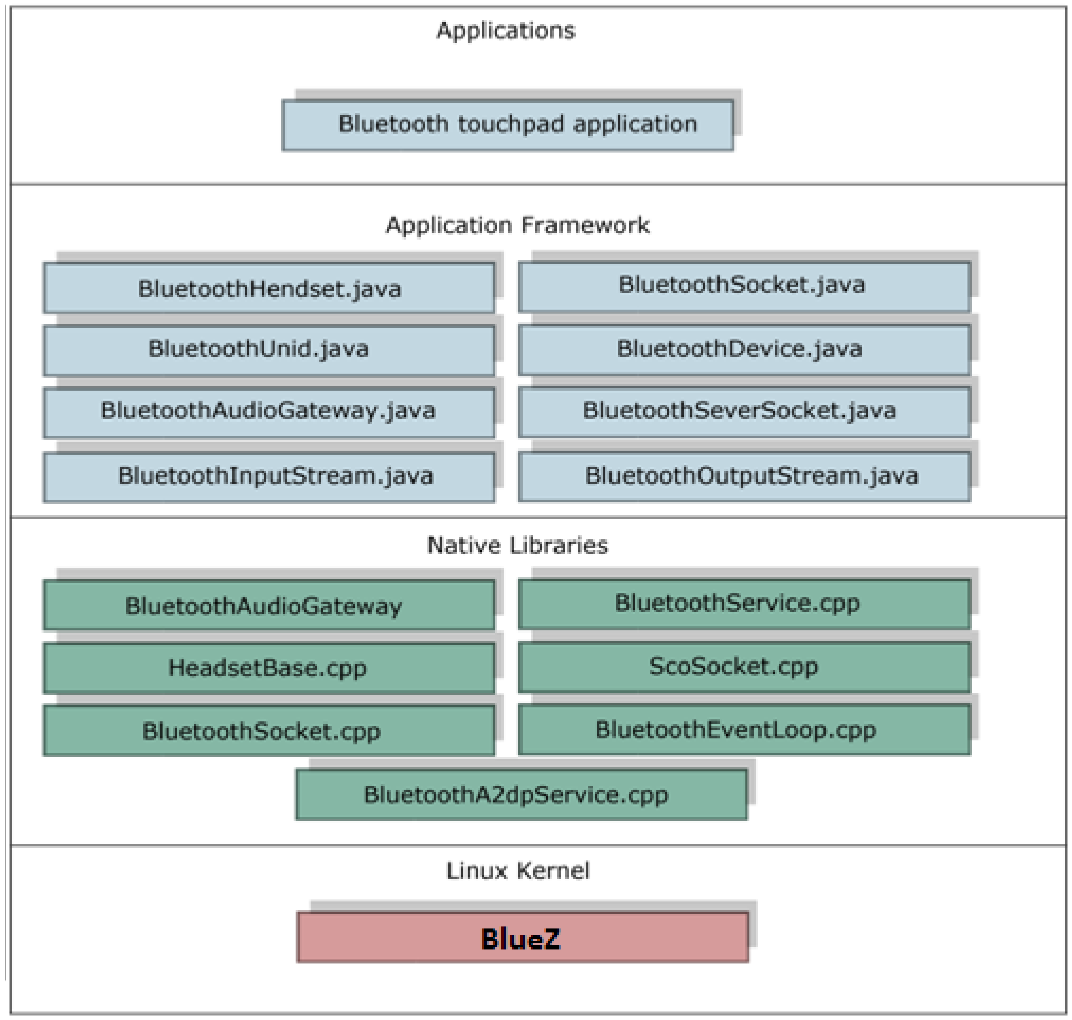


Figure 13: Bluetooth support in Android

The Bluetooth touchpad app is operating on the application level and is handling the user interactions, which are then processed. The respective formatted data is to be sent to the computer over an established Bluetooth channel. When sending the data the application has to call functions from the application framework, especially from the BluetoothSocket component. The Bluetooth components situated in 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 beneath. 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 façade that are implementing the real logic. Furthermore, they are accessing the physical Bluetooth module directly through the Bluetooth stack, present 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 stack supports all core Bluetooth protocols and layers, that were described in detail in the previous chapter. It was initially developed by Qualcomm, and is available for Linux kernel versions 2.4.6 and up.

4. Service implementation

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

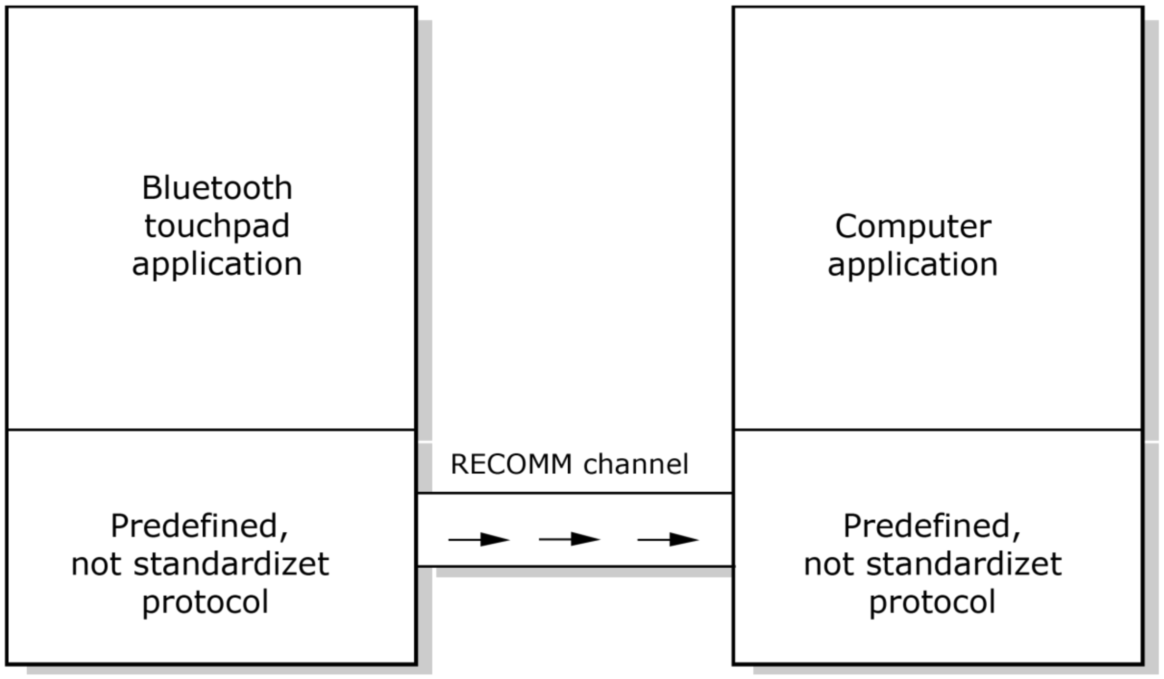

Bluetooth touchpad application
Computer application
RECOMM channel
Predefined, not standardizet protocol
Predefined, not standardizet protocol

Figure 14: Bluetooth client/server concept

The Bluetooth touchpad app is processing the user input and sending data packets to application running on the computer. The data is sent through an 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 host side, which implies various other problems (ex. compatibility with different operating systems). Second, the developer has to define a protocol that the Bluetooth touchpad app 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 very simple solution to the problem. As can be expected, there are already several apps - realizations of similar architectural models in the official Android market.

The implementation of the Bluetooth touchpad, which is described in detail in this and the next chapters, takes totally different engineering approach. Instead of developing an application for the computer host side, one could 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 would be then identifying the Touchpad app as 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 logical order. The goal is to form detailed view that can fully explain how the Bluetooth touchpad service is created and used by the drivers on the host side.

Implementing a Bluetooth touchpad means creating and providing a service or more precisely an input service for mouse and keyboard that the computer is able to utilize. But before taking advantage of 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 already been 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 services, the device is able to provide for the computer. This interaction is shown in Figure 15.

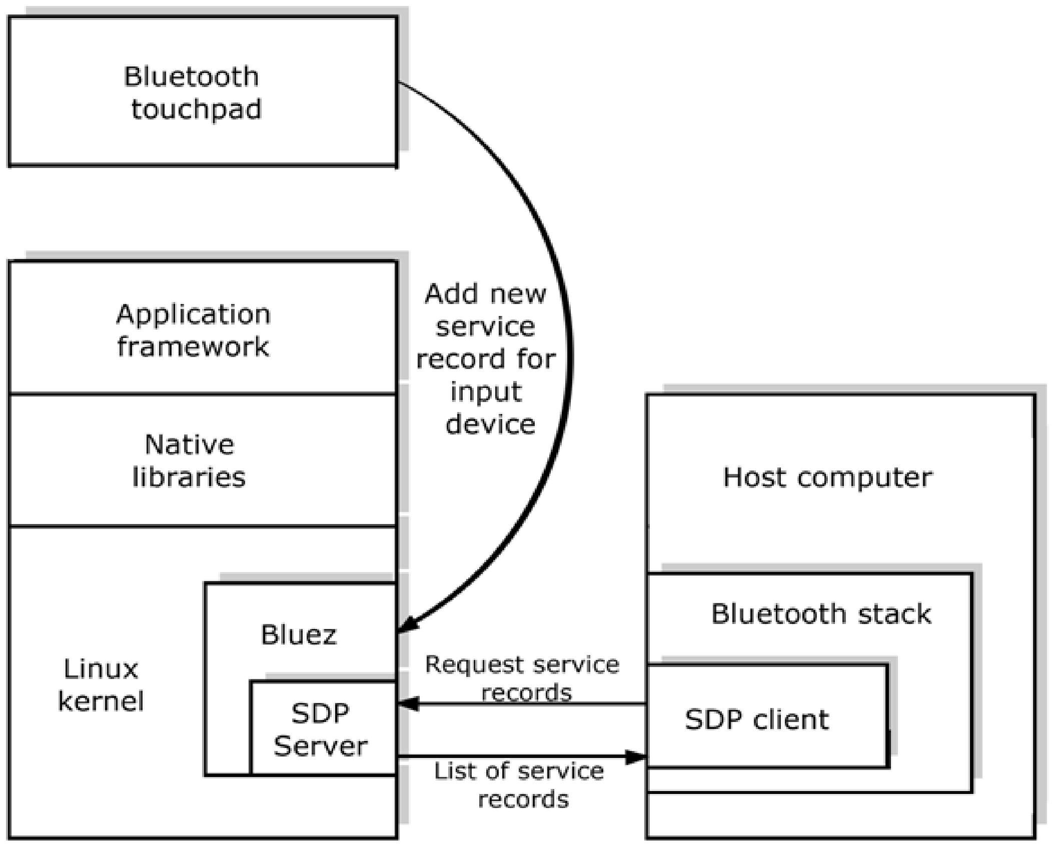


Figure 15: Adding a new service

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 in the registry of the SDP module would not be a problem, if a Java library form the application framework would provide the needed functionality. Unfortunately, such library does not exist in the application framework and in the native libraries either. However, the Linux kernel is running on the lowest level of Android. 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. Consequently, the new service description would 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 that 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 file must be deployed and executed. In the following, these tasks are solved and the solutions are described in detail, since they present the most sophisticated and important part of this work.

4.1 Bluez API and sdptool

Bluez is 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, official documentation of this API does not exist. Besides, there is very little unofficial documentation. The best tutorial showing some basic functionalities of the API is "Chapter 4: Bluetooth programming in C with BlueZ" of the "An introduction to Bluetooth programming" [12], written by Albert Haung. As a result, 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 of special interest: **add_hid_keyb** and **add_hid_wiimote**. They are both creating a new service record by using HID - the Human Interface Device protocol. Furthermore they are both adding it to the SDP registry, afterwards. 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 of them are presented in this section and others, which are expected by the Human Interface Device protocol, are outlined 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 documentation.

4.2.1 ServiceClassIDList attribute [5]

As previously explained in the second chapter 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 is then automatically building 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 [5]

The record handle, which was referenced in the second chapter, is presenting 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 [5]

The process of searching for services by the SDP client is also called “browsing” and its logic is actually based on the BrowseGroupList attribute, which similarly to the record handle and the ServiceClassIDList attribute is common to all services. The BrowseGroupList also contains UUIDs that 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 Browse-GroupList attribute 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 technique to make the service public, so the SDP client of the computer host can find it.

4.2.4 LanguageBaseAttributeIDList and human-readable attributes [5]

SDP provides a method for supporting multiple languages for human-readable attributes, whose values are strings that could be presented directly to the user. There are also several universal human–readable attributes, 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.

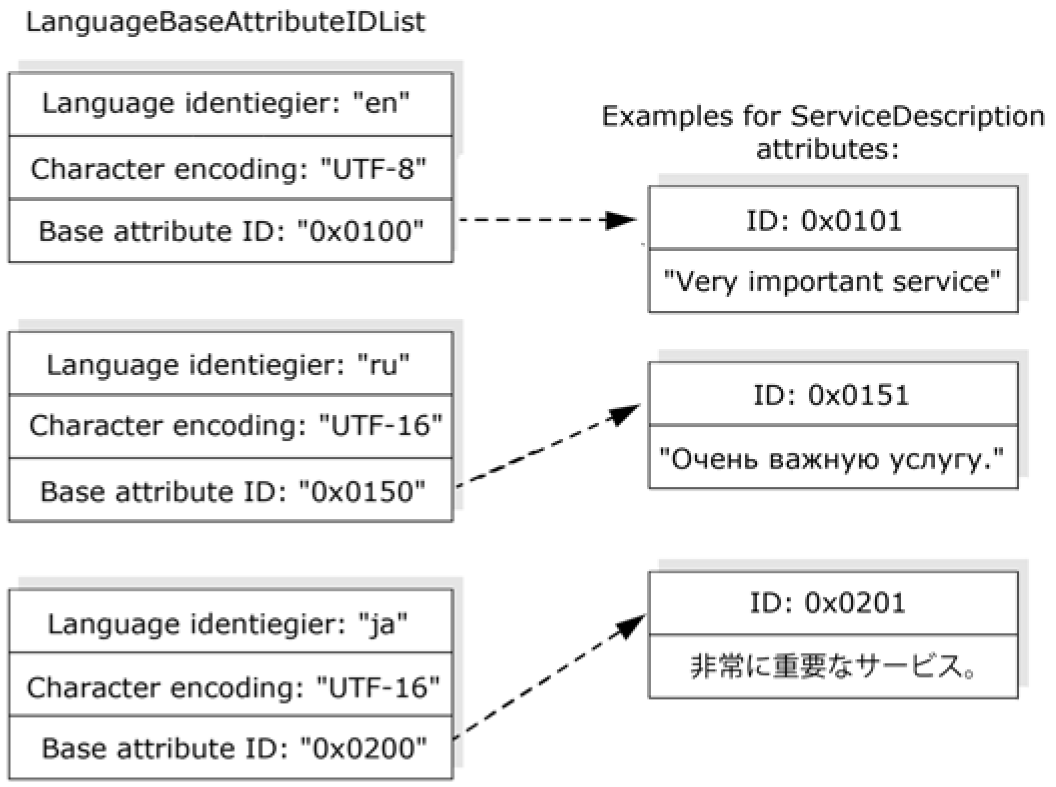

LanguageBaseAttributeIDList
Language identiegier: "en"
Character encoding: "UTF-8"
Base attribute ID: "0x0100"
Language identiegier: "ru"
Character encoding: "UTF-16"
Base attribute ID: "0x0150"
Language identiegier: "ja"
Character encoding: "UTF-16"
Base attribute ID: "0x0200"
Examples for ServiceDescription attributes:
ID: 0x0101
"Very important service"
ID: 0x0151
"Очень важную услугу."
ID: 0x0201
非常に重要なサービス。

Figure 16: LanguageBaseAttributeIDList structure

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]. Figure 16 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 element 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 Touchpad 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 elegant solution of an interesting problem. As expected, 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 app might support other languages.

4.2.5 ProtocolDescriptorList [5]

Before a service is utilized, a communication should be established between the providing and the using parties. The ProtocolDescriptorList attribute describes one or more communication protocol stacks that are used to access 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 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 Protoco-lDescriptorList example for IrDA-like printer:

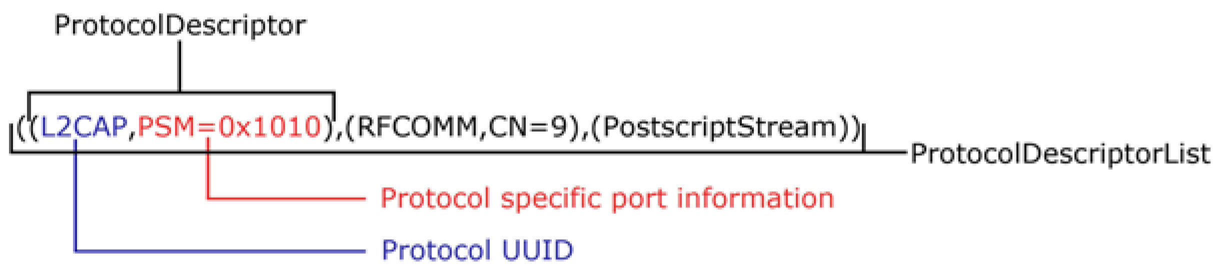

ProtocolDescriptor
((L2CAP,PSM=0x1010),(RFCOMM,CN=9),(PostscriptStream))
ProtocolDescriptorList
Protocol specific port information
Protocol UUID

Figure 17: ProtocolDescriptorList of IrDA-like printer [5]

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 a pair 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 [5]

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], 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])

4.3 Human Interface Device

The Universal Serial Bus (USB) Specification [14] presents a concept of grouping devices with similar data reporting characteristics into device classes and proposes a concept of 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."* (HID Specification v. 1.0, p.18 [6])

In the previous section we started describing the implementation of HID service record. The HID service description is of high importance for the realization of the Bluetooth touchpad. The figure 18 below outlines the role, which it plays in the implementation.

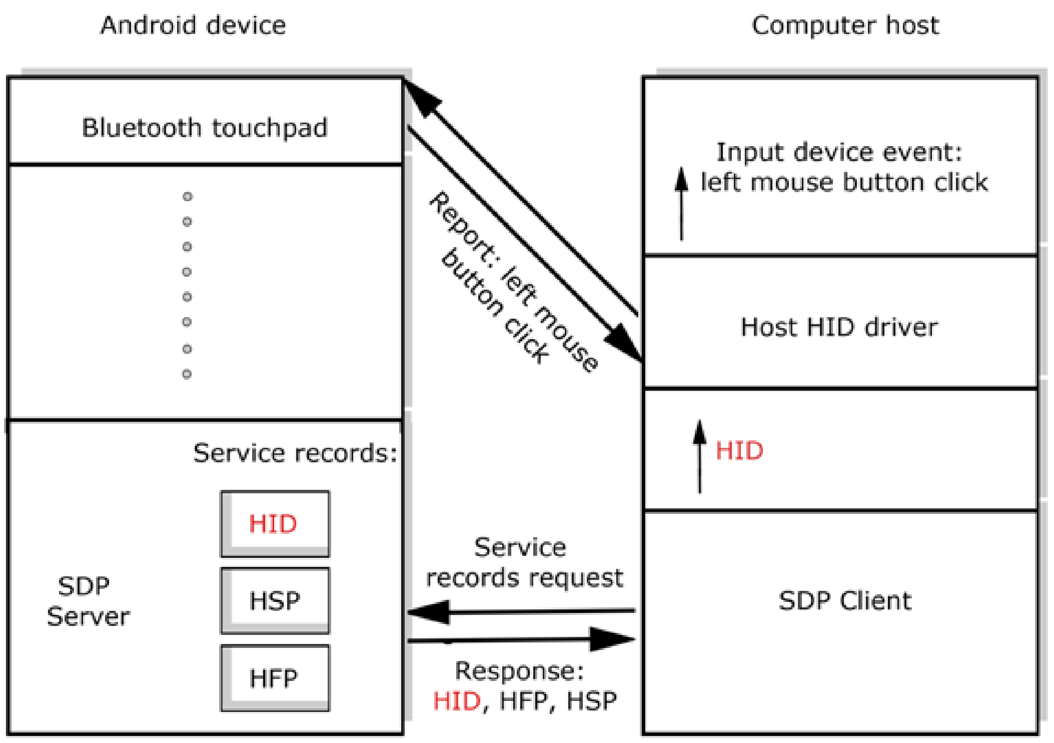


Figure 18: The role of HID service record

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, its SDP server would respond with the records of those services. Accordingly, the SDP client would recognize the HID service record and would 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 behavior, the encoding and the meaning of the data packets the service provider is sending. Then the HID Host driver would be able to establish wireless connection to the Bluetooth touchpad application over the L2CAP channel. The Bluetooth touchpad app would 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 could understand the semantics of the reported data and notify the operating system about the event.

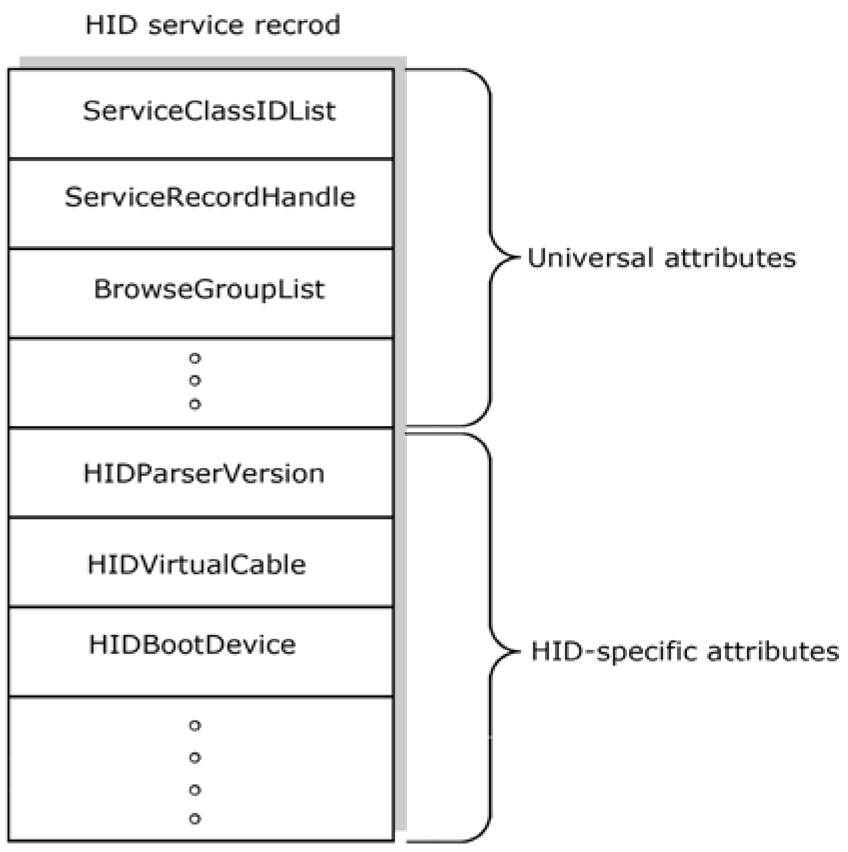


Figure 19: Structure of HID service record

As show on figure 19 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 these service attributes are described, which 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 of the Touchpad app. Hence, the explanations are very important for the realization of the Bluetooth touchpad and therefore referenced in the native source code.

4.3.1 HIDDeviceReleaseNumber [6]

The HIDDeviceReleaseNumber has an attribute ID 0x0200 and its value is 16-bit unsigned integer presenting the device release number. The version support system is the same as in the BluetoothProfileDescriptorList. The first 8 bits are responsible for the major version and the last 8 bits for the minor version. Moreover, upward-compatible changes are incrementing the minor version and incompatible changes the major version. In the implementation of the Bluetooth touchpad, the HIDDeviceReleaseNumber attribute is playing very important role. The application is actually providing two HID service descriptions, which are slightly different. The goal is to support a service for regular mouse and keyboard and a service for pointer and keyboard. However, the computer host must be able to differentiate between the two service records. Consequently, we define the release number of the first service as 0x0100 (v. 1.00) and for the second we assign 0x0200 (v. 2.00). If this step is omitted, the computer host will not be aware that the service has changed and might try to interpret the data report of pointer movement as report of regular mouse movement. The difference between the two descriptions is further explained in the subsection about the HIDDescriptorList attribute.

4.3.2 HIDParserVersion [6]

The HIDParserVersion attribute has an ID 0x0201 and its value is also a 16-bit unsigned integer presenting a number, responsible for the HID parser version. Its version support systematic is the same as in the HIDDeviceReleaseNumber - the first 8 bits are for the major and the last 8 bits for the minor version. The value of this attribute must present the version of the HID parser that is expected to parse the HID attributes. This value is needed by the HID host driver in order to know if it is actually able to parse the service description or not. In our Bluetooth touchpad implementation the value is set to 0x0111, since the service description conforms to the current Device Class Definition for Human Interface Devices [15] version 1.11, part of the USB Specification [14] series.

4.3.3 HIDDeviceSubclass [6]

The HIDDeviceSubclass has an attribute ID 0x0202 and its value is an 8-bit unsigned integer. It represents the type of device the service is describing (keyboard, mouse, pointer, joystick etc). The semantics of the attribute’s value is explained by the tables, which could be found in the Bluetooth assigned numbers specification [16]. According to these tables the value of the HIDDeviceSubclass attribute in the Bluetooth touchpad implementation is 0x33. It is representing a remote control, which has the functionality of both a keyboard and pointing device. The value is graphically explained by figure 20 below. It is important to note that bit number 0 is sent first on the air.

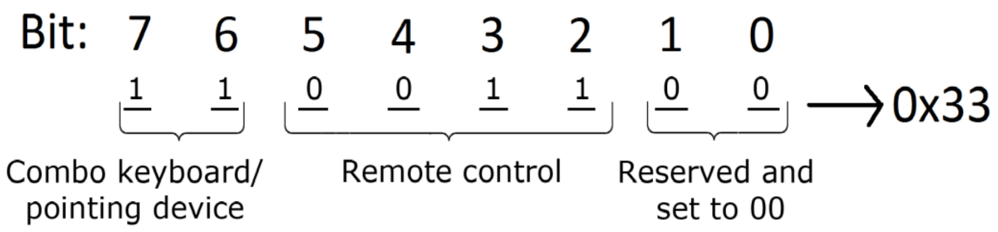


Figure 20: Device Subclass value

4.3.4 HIDCountryCode [6]

The HIDCountryCode attribute has an ID of 0x0203 and its value is an 8-bit integer, identifying which country the hardware is localized for. If the hardware is not localized, the value is 0. However, keyboards are usually localized in order to indicate the language on their keys and since our touchpad application is also supporting a keyboard input method, this value should not be set to 0. The valid county codes are further listed in the Device Class Definition for Human Interface Devices [15]. In the Touchpad implementation the standard US localization has been chosen. Therefore the HIDCountryCode has a value of 0x21.

4.3.5 HIDVirtualCable [6]

The HIDVirtualCable has an attribute ID of 0x0204 and an 8-bit Boolean value, which is responsible for indicating if the device supports virtual cables. If it does, and the HIDVirtualCable attribute is set to true a 1:1 relationship with the host is established on connection. This means that only one host could receive data from the HID device, as long they are virtually cabled, although the Bluetooth technology allows for several hosts to do so. Further, in case of connection drop for any unknown reason, the devices must try to establish the connection again. Which device is waiting for connection and which is trying to connect is specified by the HIDReconnectInitiate and HIDNormallyConnectable attributes. In the Bluetooth touchpad implementation the HIDVirtualCable attribute is set to true, since the requirements for the virtual cable support are met. Moreover, it is of significant importance that the Android HID device is expected to reconnect to the computer host in case the connection is lost. This is reviewed in detail in the next subsection, where the HIDReconnectInitiate and HIDNormallyConnectable attributes are explained.

4.3.6 HIDReconnectInitiate and HIDNormallyConnectable [6]

The HIDReconnectInitiate attribute has an ID of 0x0205 and the HIDNormally-Connectable has an ID of 0x020D. There are both 8-bit boolean values. As explained in the subsection about the HIDVirtualCable, our touchpad device is supporting virtual cables. So a connection reestablishment is expected in case of connection drop. The HIDReconnectInitiate and HIDNormallyConnectable attributes are responsible for indicating if the HID device shall wait for the host's connection attempt or vice versa. The following table describes the four possible cases, which are then separately discussed.

Case	**HIDReconnectInitiate**	**HIDNormallyConnectable**	**HID**	**Computer host**
1	false	false	do nothing	do nothing
2	false	true	wait for connection	connection attempt
3	true	false	connection attempt	wait for connection
4	true	true	connection attempt & wait for connection	connection attempt & wait for connection

Table 1: HIDReconnectInitiate and HIDNormallyConnectable logics

In case 1 both attributes are set to false and the host is expected to restore the connection, but cannot do so without user intervention. In case 2 the HIDReconnectInitiate is set to true and the HIDNormallyConnectable to false. This is the standard setting for Bluetooth mouse and keyboard and the host is expected to connect to the HID device. However, in the Bluetooth touchpad implementation the HIDReconnectInitiate is set to false and the HIDNormally-Connectable to true (case 3). This indicates that the reconnection attempt is left to the HID device. Nevertheless the reason for this has nothing to do with the HID specification, but with a limitation of the Bluetooth stack in Android. When a HID device is waiting for connection it opens a L2CAP server socket. Unfortunately, L2CAP server sockets are not supported in Android, since a Bluetooth daemon is responsible for refusing every connection attempt of the host. This issue is further discussed in the next chapter.

4.3.7 HIDSDPDisable [6]

The HIDSDPDisable attribute has an ID of 0x0208 and its value is 8-bit Boolean indicating whether a HID device allows a host to query its SDP server when the control and interruption channels are open. Some devices have very restricted resources and set this attribute to true, since they are not able to support additional L2CAP connections to the SDP server, when there are already two L2CAP connections running for the control and interruption channels. However, the Android phone is very powerful device and therefore this parameter is set to false in the implementation.

4.3.8 HIDBatteryPower [6]

The HIDBatteryPower has an attribute ID of 0x0209 and its value is 8-bit Boolean indicating whether the device runs on battery or is connected to power source. If the attribute value is set to true (HID device runs on battery), a special power management mode is triggered in the host. It is significantly extending the battery life, for example by putting the device in park mode when it is not active for long time. However, the attribute value is set to false in the implementation, since no significant gains in the Android phone’s battery life are made by the special power treatment. The Bluetooth communication is low-power feature and is already consuming far less energy than the phone’s display for example.

4.3.9 HIDRemoteWake [6]

The HIDRemoteWake attribute has an ID of 0x020A and its value is also 8-bit Boolean. If set to true, the HID device would be able to wake up a host that is in sleep or hibernate mode. Mouse and keyboard are typical examples for devices that have this parameter set to true and therefore appear in the host’s set of devices, which could wake it up. Consequently, the value of the HIDRemoteWake attribute is set to true in the Bluetooth touchpad implementation.

4.3.10 HIDProfileVersion [6]

The HIDProfileVersion attribute has an ID of 0x020B and a 16-bit integer value indicating the version number of the Bluetooth HID Specification that has been taken into account when the service class was implemented. The attribute must not be mistaken with the HIDParserVersion attribute, which indicates the version of the HID Parser as one of the USB class drivers running on the computer host side. As the last HID Specification [6] is version 1.00 and became available 2003, this is also the version number implemented in the Bluetooth touchpad service description.

4.3.11 HIDBootDevice [6]

The HIDBootDevice has attribute ID of 0x020E and its value is 8-bit Boolean, which when set to true indicates that the device is supporting the boot protocol mode. For example, when the user is configuring the computer's BIOS settings, this is done only under boot mode. Therefore most of the keyboards are supporting this protocol in order to enable user input. Nevertheless, in the Bluetooth touchpad implementation this attribute is set to false, despite the fact that a keyboard input method is also available. On the one hand, to support the protocol additional commands must be implemented and on the other, there is no guarantee that the Bluetooth touchpad application will be running when the computer boots. However, future realizations of the Touchpad app might support the boot mode protocol.

4.3.12 HIDLANGIDBaseList [6]

The HIDLANGIDBaseList has an attribute ID of 0x0207 and its value unlike the attributes previously presented is neither an integer nor a Boolean, but a data sequence. Its semantic is almost completely identical with the one of the LanguageBaseAttributeIDList. The attribute is presenting the same solution for multiple languages support for human-readable HID attributes. Each element of the HIDLANGIDBaseList has base attribute ID responsible for a single language. The human-readable universal attributes are then defined with an attribute ID offset from each of these base ID values, rather than with an absolute attribute ID. The only difference is that the elements of the HIDLANGIDBaseList, unlike the elements in the LanguageBaseAttributeIDList consist of only language identifier and base ID. The value specifying the character encoding in the LanguageBaseAttributeID element is not required in the HIDLANGIDBase element, because according to the HID Specification [6] only UNICODE (UCS-2) is used for encoding of human-readable strings. In the particular implementation of the app only one language is supported, so the HIDLANGIDBaseList has a single element of two values: {0x0409, 0x0100} the first is the language identifier, which indicates English (United States) according to the USB Language Identifiers Specification [17]. The second value is the base ID for the primary language, which same as in the LanguageBaseAttributeIDList must be defined as 0x0100.

4.3.13 HIDDescriptorList [6]

In the previous subsections almost all the attributes in the service record that are expected by the HID protocol were described. The attributes provide for the HID driver on the computer host side the required information about the behavior and the specifics of the HID device. However, one problem remained unsolved. The figure 21 below is describing the issue graphically.

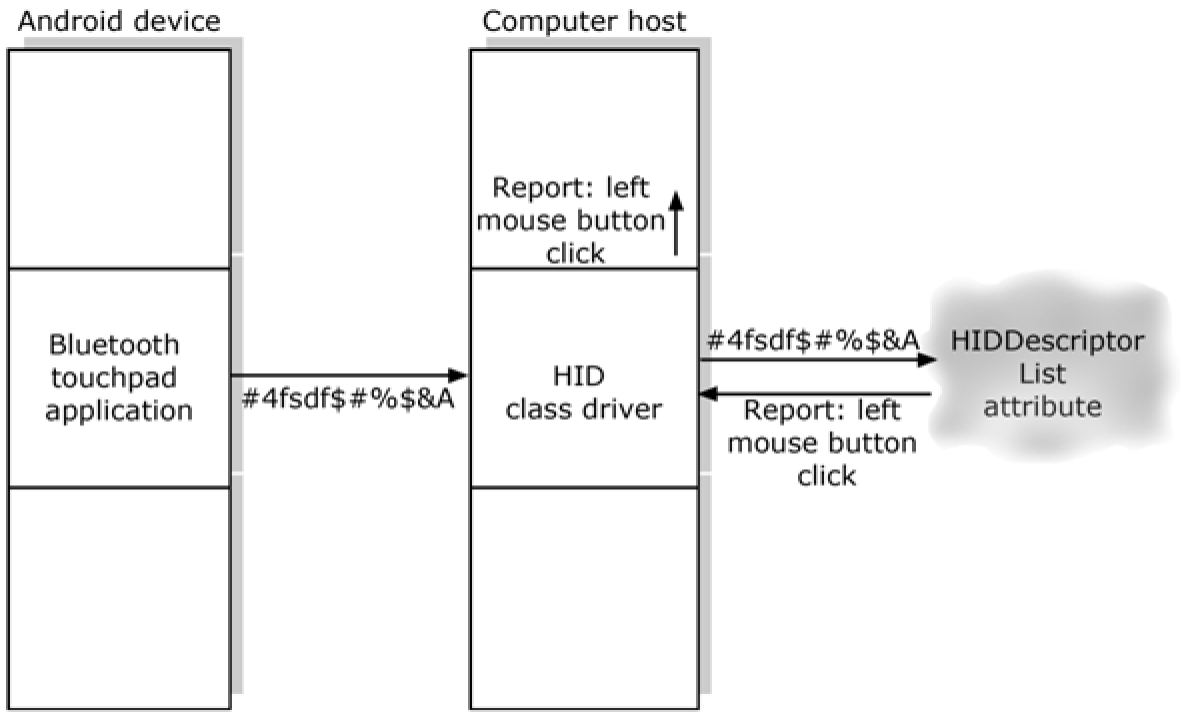

Android device
Computer host
Report: left mouse button click
Bluetooth touchpad application
#4fsdf$#%$&A
HID class driver
#4fsdf$#%$&A
Report: left mouse button click
HIDDescriptor List attribute

Figure 21: The role of HIDDescriptorList attribute

When the user interacts with the Android device, the Bluetooth touchpad application is processing the user actions. Furthermore, the application is encoding packets of data, which contain the information about these actions. The encoded data is then sent to the HID class driver on the computer host side over the established L2CAP connection. The HID class driver has already parsed the HID service record, which was passed by the SDP client, as previously shown on figure 18. However, none of the attributes, described in the previous subsections, explains how the HID class driver is supposed to decode the data, which might contain information about mouse event, keyboard event, joystick event or any other type of input device event. In the previous subsections all the attributes that are part of the Bluetooth touchpad implementation were reviewed in detail except the HIDDescriptorList. It is the HIDDescriptorList attribute that contains the information needed by the HID class driver in order to decode the data sent by the Touchpad application. In this section the last and certainly most complex attribute is presented. The information concerning the structure of the attribute is not found in the HID Specification [6], but in the Device Class Definition for Human Interface Devices Specification [15] that is part of the USB Specifications Group [14].

The HIDDescriptorList has an attribute ID of 0x0206 and its value is data sequence of at least one data element, called HIDDescriptor. Each HIDDescriptor is responsible for describing a single input device. The HIDDescriptor itself is a data sequence of attributes. Each attribute consist of ID and value pair. The figure 22 is showing the concept of the HIDDescriptorList.

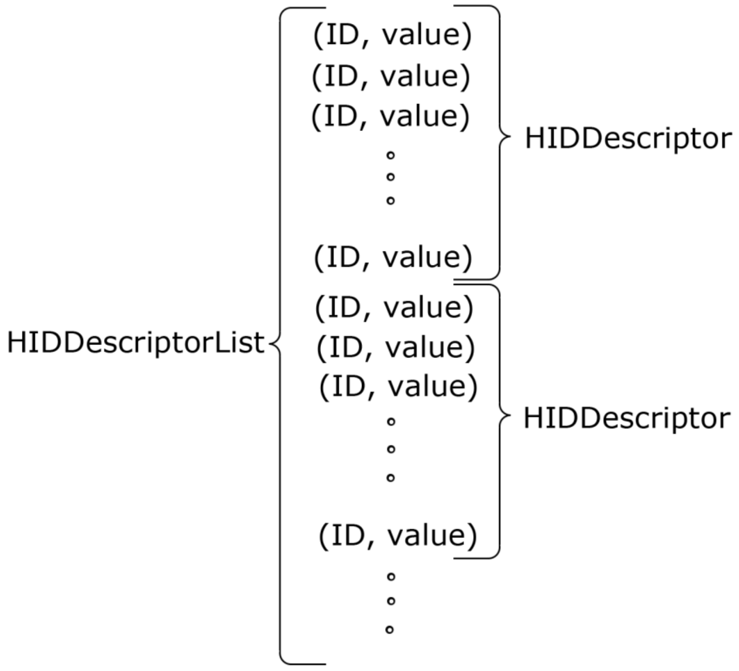
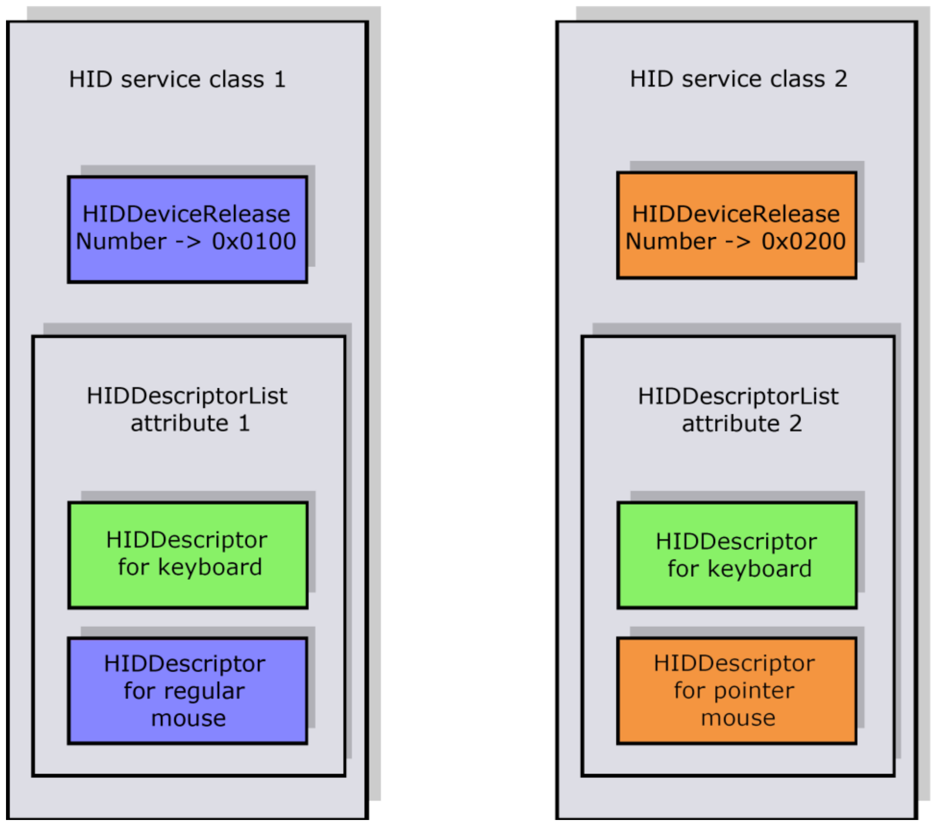


Figure 22: HIDDescriptorList concept

In the Bluetooth touchpad implementation one could actually distinguish between two separate HID service classes, which vary only in their HIDDescriptorList and their HID-DeviceReleaseNumber attributes. The HIDDescriptorList attribute of the first service class has two HIDDescriptors that are describing a regular mouse and a keyboard. The HIDDescriptorList attribute of the second service class has also two HIDDescriptors. The first is describing a pointer that is sending absolute screen coordinates and the second is the same keyboard HIDDescriptor as in the first service class. The figure 23 outlines this issue.


HID service class 1
HIDDeviceRelease Number -> 0x0100
HIDDescriptorList attribute 1
HIDDescriptor for keyboard
HIDDescriptor for regular mouse
HID service class 2
HIDDeviceRelease Number -> 0x0200
HIDDescriptorList attribute 2
HIDDescriptor for keyboard
HIDDescriptor for pointer mouse
Figure 23: Supported HID service classes

When the user starts the touchpad application he has to choose which HID service record is to be inserted in the registry of the SDP server. This is additionally discussed in the next chapter.

The complete hex-value arrays for HIDDescriptors describing a regular mouse and a keyboard could be easily found in the Device Class Definition for Human Interface Devices Specification [15]. Furthermore, the HIDDescriptor for the pointer is actually a slightly modified HIDDescriptor for a regular mouse. However, the definition of a HIDDescriptor is containing the information about the semantics of the data reports – bytes sent by the Bluetooth touchpad application, describing mouse or keyboard events. In appendix A, the connection between the HIDDescriptor for regular mouse and a byte array, representing a data report of mouse movement and mouse click, is graphically outlined.

4.4 Compiling Linux executable

After writing the source code that is going to create and add the service record to the SDP registry in Android by directly interacting with Bluez, it certainly has to be compiled in a machine language. Therefore, a compiler is needed that could compile the code for the specific operating system - in this case a modified Linux kernel. Fortunately, the Native Development Kit for Android contains such compiler. The NDK further contains several stable APIs exposed by headers in the **<ndk>/platforms/android-X/arch-arm/usr/lib** directory. These APIs are located in this directory, so the developers would be able to compile native code that is using their functionality. The native code is usually used in components responsible for some performance critical operations. Furthermore, the libraries are providing an interface to low-level system resources. Unfortunately, the Bluetooth shared library that is naturally referenced in our source code, is not among them. Of course, in order to compile the executable, the Bluetooth shared library has to be set in the directory where the other libraries reside, so the compiler could find it. To place the library in the specific directory, we must first find it. System libraries in the Linux operating system are usually located in directories such as **/lib**, **/usr/lib** or **/user/local/lib**. By running a file manager application such as Root Explorer [18] on a rooted Android phone, one could access the system files. The process of rooting a phone and the consequent benefits are referenced in the next chapter. The **libbluetooth.so** - the Bluetooth shared system library is found in the **./system/lib/ directory**, as expected. There is only one step left, in order to compile the source. In the source several components, part of the Bluez API and also the Linux socket API are referenced through includes of their headers.

```
#include <sys/socket.h>
#include <bluetooth/bluetooth.h>
#include <bluetooth/hci.h>
#include <bluetooth/hci_lib.h>
#include <bluetooth/l2cap.h>
#include <bluetooth/sdp.h>
#include <bluetooth/sdp_lib.h>
```

The compiler must be shown the location of these respective headers, since they are also not included in the provided NDK. The headers could be found in the source code of the operating system itself. Consequently, the easiest thing to do is to download the code of the Android operating system by following the instructions on the official Android Source [8] web page. Once the code is available, the headers could be easily traced. The last thing to do is to modify the native manifest file – Android.mkl, so the compiler is able to find them and compile the executable. This is done by adding the following lines:

```
LOCAL_C_INCLUDES :=<android_source>/external/bluetooth/bluez/lib/ \
                                        <android_source>/system/core/include
```

The process of compiling the source code has been hereby explained in detail, since finding a way to work it out, turned out to be one of the most time consuming tasks in the project realization.

5. The Bluetooth framework

In the previous chapter, we described the implementation of an executable, which aims to create and add a new HID service record in the registry of the SDP server of Android. When the computer host's SDP client connects and requests the available records, the Android SDP server responds by sending the records, one of which is the HID service record. The HID service record is a description of an input service for keyboard and mouse. The description contains the necessary information, which the computer host needs in order to correctly utilize it. However, the service itself is provided by the Bluetooth touchpad application. It is the app that connects to the waiting host and starts sending data reports, comprising the information of mouse clicks and movements and keyboard presses. The HID host driver on the computer side is then responsible for decoding and understanding them. This is possible since it has previously parsed the HID service record, which also describes how the reports are to be handled.

It is an ambition of this project to create a small Bluetooth HID framework for Android devices. The framework allows developers to implement Bluetooth applications that utilize the functionality of HID devices, such as mouse, keyboard and a pointer device. (The graphics pad, which was presented in the second chapter, as well as other devices, is actually using the HID service of the pointer device class.) Furthermore, the framework must be developer-friendly in such way that the developer would not need to worry about the executable that is creating and adding the service, about opening the communication channels, and manually composing the data reports. Figure 24 is conceptually explaining this issue.

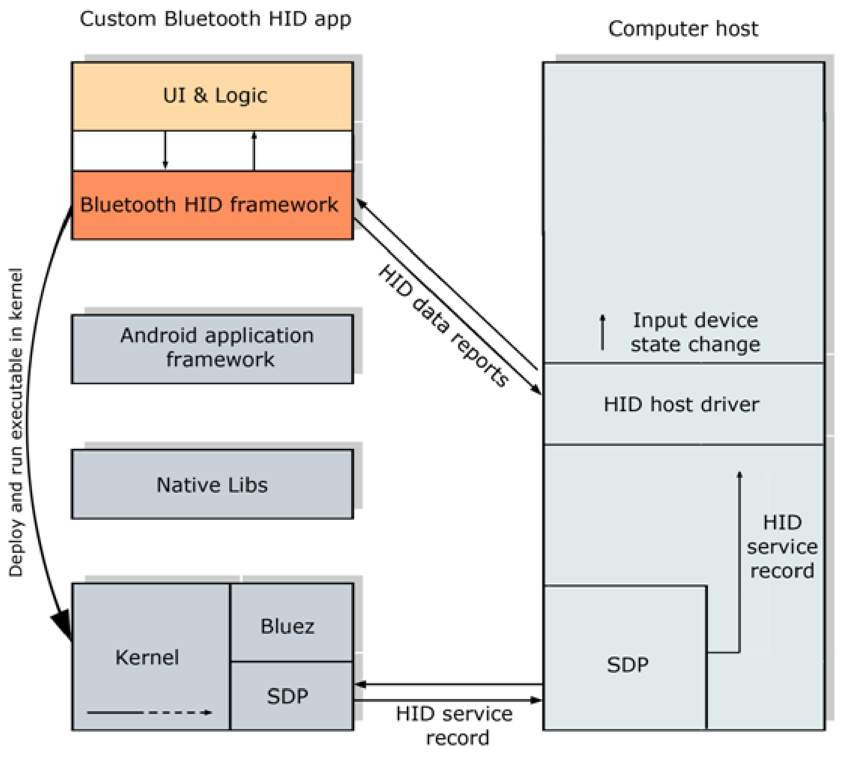


Figure 24: Bluetooth HID framework concept

The developer has the task to implement the UI of the application and its logic that handles different user interactions. For example, the application might process touch screen events or events from the motion sensor or even the user’s voice commands that are then turned into text and sent to the computer host as keyboard input. However, the Bluetooth HID framework will handle the deploying and running of the executable in the kernel. It will further handle the establishment of the communication channels with the host computer and will additionally provide a nice and easy to implement interface for building the data reports. The developer will only be concerned with the logic, which transforms the user interactions into mouse and keyboard data reports. The Bluetooth touchpad application realized in this project is actually also structuring its UI and logic components on top of the Bluetooth HID framework. When browsing the project’s source code, one will notice that the Bluetooth HID framework is packed in the **btframework** package, which itself contains three packages, each representing a single independent component. In the following chapter, these components are reviewed separately and then their collaboration with the on-top custom components, provided by the developer, is conceptually clarified.

5.1 The Bluetooth SDP component

The Bluetooth SDP component has the functionality to deploy and run the executable, which has previously been compiled and has the intent to create and add the HID service record. The specifics of the implementation and the compilation of this executable have been reviewed in detail in the previous chapter. The SDP component actually provides only two functions that are public and thus must be called by the components that are utilizing the framework:

```
public void registerHID(final int recordType) {…}
public void unregisterHID(){…}
```

The first command is responsible for deploying and running the executable, which is then adding an HID service record to the registry of the SDP server. By passing an integer argument, the developer could specify, which one of the two available HID service records should be added. One of the service records is defining an input service of regular mouse and a keyboard and the other is describing a pointer and a keyboard. This issue has also been previously discussed in the last chapter, where the HIDDescriptorList has been presented. The second command is again running the executable by passing different parameters. The executable is then removing the previously added service record. This command should be called when the application is no longer active, and therefore the service not available. Notably, the commands are not returning any result, so it seems they do not provide information if the operation has been successful or not. As a matter of fact, they are both executing their logic in a separate Thread. The Android Developer Guidelines [10] prescribe that time consuming operations should be run in their own process and not in the UI Thread. By not complying, the developer risks the appearance of "Activity not responding." notification on the screen, which will then close the whole application. Through the "register" and "unregister" functions the SDP component is actually providing functionality for asynchronous calls, which are delivering their result through a listener. The listener is a very simple interface, which is also defined in the SDP component:

```
public static interface SDPStateListener {
                public void onRegisterComplete(int state);
                public void onUnregisterComplete(int state);
        }
```

An instance that implements the interface should be passed to the constructor when initiating the SDP component. After the "register" and "unregister" functions are done, they will use this listener to deliver the result as an integer value, which will describe either a success code or one of several error codes, each describing a different problem.

Furthermore, in order to run the executable when the "register" function is called, the SDP component has to complete two tasks. First, it has to deploy the executable file in the internal memory of the phone where it could be executed from. When compiling the C native code, the Android NDK places the executable file in the **/libs/armeabi** directory of the project. The developer must then manually move it to the **/assets** directory. This is the default directory, where asset files could be easily accessed through the APIs provided by the Android application framework. The APIs also provide the functionality to cache files in the so-called "PRIVATE_MODE". When caching in "PRIVATE_MODE" the files are actually stored in the internal memory and more specifically in the **/data/data/<application_package_name>/files/** directory. The **<application_package_name>** tag represents the main package description of the Android application. For the particular case of the Bluetooth touchpad the executable file would be deployed in the **/data/data/tum.betriebsysteme.kostadinov/files/** directory.

After the executable file is deployed in the internal memory of the phone, the SDP component is entirely set to complete its second task and run the executable. The only way to run it, however, is to open the Linux command console and pass the specific commands. Fortunately, the command console could be accessed through one of the APIs in the Android application framework. The next step is to pass the commands. However, these require root permissions in order to work. In the Linux operating system, root is the conventional name of the user that has all rights or permissions to all files and programs. Furthermore, it is the only user that is allowed to modify the root directory of the system. This is exactly what we intend to do by adding the new service record in the SDP registry, which is part of the Linux kernel.

Unfortunately, the Android architecture presents certain limitations for the developers. For security reasons, applications are not allowed to gain root permissions. Nevertheless, there is a solution to this problem. Advanced users could "root" their phone, in order to use its full capabilities. Still the process of "rooting" an Android phone is complex and connected with risks of destroying the phone software. Additionally, "rooting" a phone means giving up the manufacturer's guarantee. Therefore, the requirement of a rooted phone is presenting one of the greatest limitations of the Bluetooth touchpad application.

Consequently, if the phone is rooted, the Bluetooth application could easily gain root permissions and pass the commands to the console. The commands for running the executable are:

```
# sudo su
# cd /data/data/tum.betriebsysteme.kostadinov/files/
# ./sdp register [-relative/-absolute]
```

The first command is responsible for providing the root permissions for the process. The second line is navigating to the directory, where the executable file is located. The third one is then running it by passing "register" as first parameter and a second parameter that indicates which of the two available service records is to be added to the SDP registry. By passing "unregister" parameter to the executable, it will remove the previously added record. This is done by the **unregister()** function of the SDP component.

5.2 The L2CAP component

As could be expected, the L2CAP component is responsible for establishing a communication channel with the computer host by using the L2CAP protocol from the Bluetooth software stack. As a matter of fact, the Android application framework presents a Bluetooth API, which unfortunately does not support communication on the L2CAP level. However, a careful examination of the API source code, written in the Java programming language, indicates that it actually provides the functionality needed to initiate the L2CAP connection. Nevertheless, this functionality is hidden by a class constructor which is declared as "protected", and therefore could not be accessed by components declared in other packages. The intention of the Android engineers has probably been to make this functionality available in a future version of Android. And the reason for this decision is maybe the still missing possibility for creating a L2CAP server, which is waiting for incoming connections. Yet, if connecting is possible, it would be enough to realize the Bluetooth touchpad project.

The Java programming language includes an API called "The Reflection API" [19]. It is actually implemented by programs that intend either to examine or to modify the runtime behavior of other Java programs running in the Java virtual machine. "Reflections" is a very powerful API, which is intended to be used with caution only by advanced developers, since it enables applications to perform certain operations that would not be otherwise possible. One of the features of the technology is to provide access to all constructors and methods of other APIs, no matter if they are declared protected or private. Fortunately, the "Reflection API" [19] is also supported in the Android application framework. [19]

In the Bluetooth HID framework realized in this project, the Reflection technology is used only once in order to call a constructor of a single class – BluetoothSocket.class, which is hidden from the developers. The functionality behind the constructor is both complete and stable. It is unavailable, probably because it is a single part of incomplete bundle of functions. However, it remains clear that the API should be used for testing only, or if all standard methods are failing. The reason for this is that it is severely neglecting the concepts of the object oriented Java programming language, and the overuse of the technique might actually do more bad then good.

Similarly to the SDP component, the L2CAP component also requires a listener instance to be passed on initialization. The listener is implementing an interface, which is declared in the component:

```
public static interface EventListener {
            public void onSocketConnectionFailed(int port);
            public void onBytesRead(int port, int bytesRead, byte[] buffer);
            public void onSocketConnected(int port);
      }
```

Same as the SDP component, the L2CAP component is also running asynchronously in a separate Thread once its **connect()** function is called. Hence, the listener technique makes it possible to notify the calling instance, when the connection is successfully established or if it is interrupted, or even when a message sent by the computer host is received. In general, the publicly available functions of the component are:

```
public void connect(BluetoothDevice device){…}
public void write(byte[] bytes){…}
public void cancel(){…}
```

The **write()** function could be called once the connection is established. It is used to send the data reports to the HID host driver on the computer side. After the connection is no longer needed it could be safely suspended with the **cancel()** function.

5.3 The Report component

As mentioned in the previous section, the L2CAP component provides the functionality for sending data to the HID host driver on the computer side. The developer is free to send any type of data, but one actually needs to send mouse and keyboard data reports. The data reports must conform to the specification provided by the HIDDescriptorList attribute in the service record, which is described in appendix A. Instead of building the data reports by specifying each byte separately, the developer could use the data manipulation units presented by the Report component. They are actually data patterns providing the structure of the different data reports that could be easily manipulated. For example, the developer could initiate a mouse report and then specify a left mouse click. By using these data patterns, the developer does not need to worry if the data reports are conforming to the description specified in the HIDDescriptorList attribute of the service record. Sample code showing the use of the data patterns is presented in appendix B.

5.4 Bluetoooth HID framework in action

In the previous subsections the components of the Bluetooth framework have been explained separately. In order to implement the framework, the developer must implement the graphic user interface and the logic, which describes how the user interaction with the Android device is to be translated into mouse and keyboard reports. In the following, we explain how the framework is to be utilized by the components built by the developer. In the following the third-party components are abstracted as a single component - the UI&Logic component. Then, their interactions with the components of the Bluetooth HID framework is shown with an UML sequence diagram.

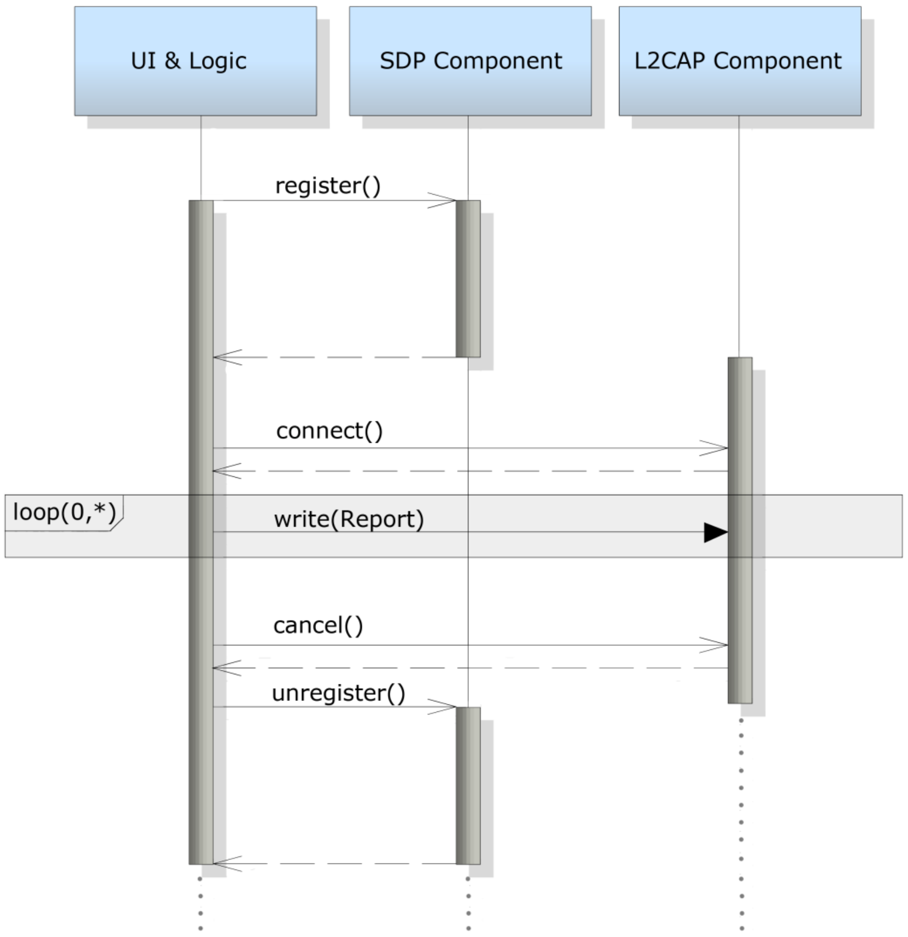


Figure 25: Bluetooth HID framework in action

The UI & Logic entity is asynchronously calling the **register()** function of the SDP component, in order to add the service description in the registry of the SDP server, where from it could be requested by the computer host. Then another asynchronous call is triggered upon the L2CAP component, which is requested to establish the Bluetooth connection with the computer host using the L2CAP protocol. Once the connection is established, the UI & Logic can send any number of data reports presenting mouse and keyboard events, by using the data patterns, provided by the Report component. After the UI & Logic is done sending data reports, for example when the user is closing the application, it must call once again the L2CAP component in order to safely discontinue the connection. Since the application will not be active anymore, and consequently the input service for mouse and keyboard will not be available, the service record must be removed from the registry of the SDP server of the Android phone. Therefore, the UI & Logic must call the **unregister()** function, for example shortly before the application is closed.

5.6 Implementing the framework

The Bluetooth touchpad application is introducing a sample implementation of the previously described framework. The implementation aims to outline the strength of the framework by enabling the realization of several different use cases, which are called “Options”. The user can choose and simultaneously change the currently active Option. The main Option is, of course, the Touchpad. The Touchpad has exactly the same functionality as the touchpad of an ordinary Notebook, which is used instead of a mouse device. As can be expected, keyboard is also supported. There are two Options, which are taking advantage of the service record describing a pointer instead of an ordinary mouse. These are the Pointer and the Paintpad. The first is using the phone motion sensors to provide the functionality of pointing device, which is used for presentations. The Paintpad presents an innovative way to draw, which is very similar to the way one would draw with graphics pad, introduces in the second chapter. Another interesting Option is the “Voice”. The Voice is taking advantage of the phone voice recognition API and transforms the user’s speech into keyboard input. A joystick with some basic functionality is also present. It is very suitable for some simple Linux games.

Since the Bluetooth framework makes the realization of these Options extremely easy, their exact implementation is not in the focus of this work. On the Android Bluetooth Touchpad project website [20], one could find also videos, showing the different Options in action. It is the intent of this work to provide the framework to third-party developers, who can then implement their own Bluetooth visions and ideas.

5.7 Problem with the Windows Bluetooth stack

As previously indicated in this chapter, the Bluetooth input devices should wait for the computer host to establish the communication channel. At least, this is what the Human Interface Device Specification prescribes. Once the communication is established and then interrupted for any reason whatsoever, (for example, the device has run out of battery) both devices are able to re-establish the communication channel. The problem, which also represents another limitation of the framework, is that the Android device is unable to open the L2CAP server socket, which is supposed to wait for incoming connections by the computer host. The Bluez framework, being part of the Linux kernel, has still not provided this functionality to the developers. Indeed, for this reason, any device running the Windows operating system is unable to connect and utilize the input service. Although it seems plausible that an Android device connects to a waiting Windows host when an established communication channel is interrupted, this could never happen, since the channel has not been initiated in the first place. However, this crucial problem does not appear if the computer host is running the Linux operating system. By processing the service description, Linux examines if the Bluetooth device is supposed to re-establish the connection in case of interruption. If this is true and the device is also marked as “trusted” by the user, Linux is directly opening a L2CAP server socket, which is waiting for incoming connections by the device. The Bluetooth touchpad application takes advantage of this behavior and connects to the computer host. It could then start sending data packets, which are describing input events.

6. Related work

There is a single project with goals similar to these of the Bluetooth touchpad project. It is an open source project, called AndroHID [21]. It was introduced by Manuel L. in the Chair of Operating Systems of TU Munich. While, it does not achieve the implementation of both mouse and keyboard input services, it basically shows a way this could be done. Understanding the principals behind the AndroHID [21] concept has played significant role in the process of realization of the Bluetooth touchpad project. The figure 26 below is showing the way AndroHID [21] works.

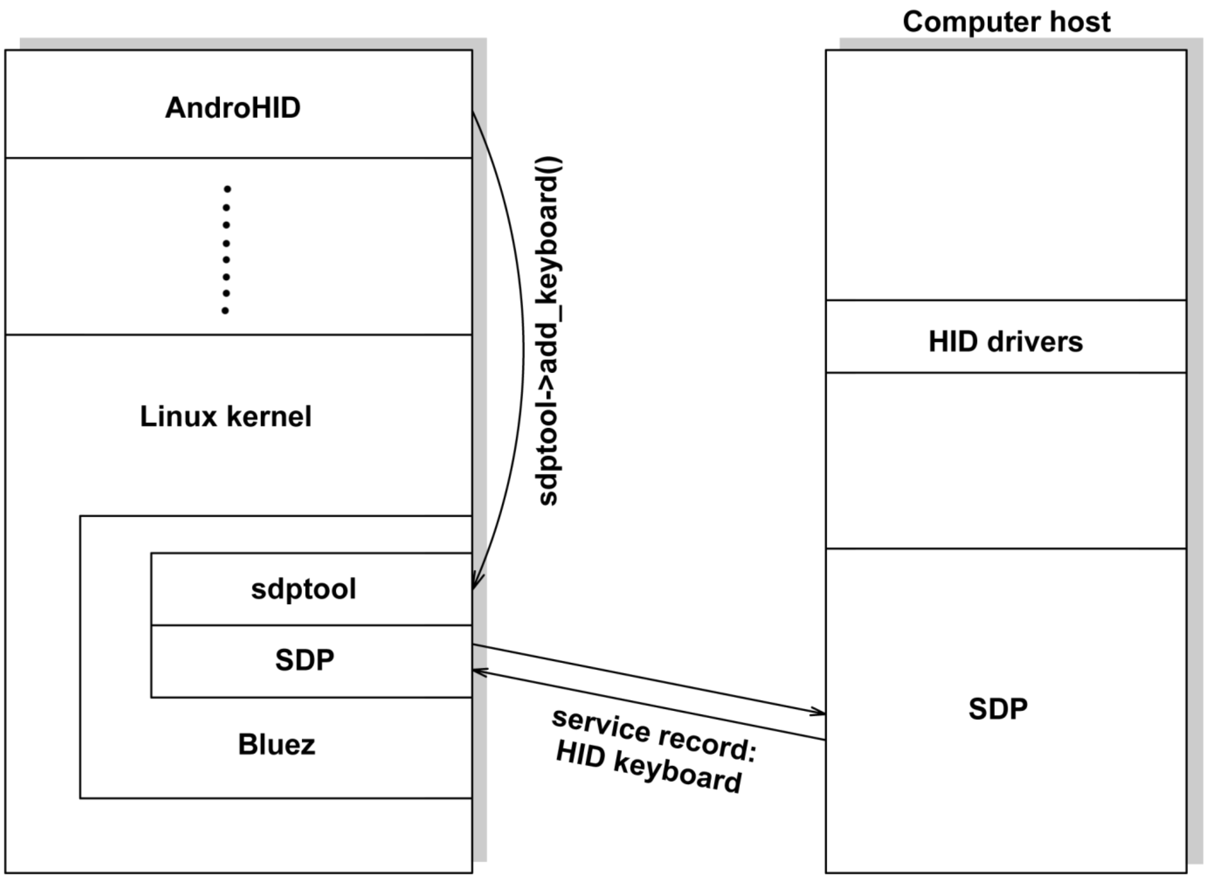


Figure 26: AndroHID concept

The AndroHID [21] application, similarly to the Bluetooth touchpad app is also interacting with the Linux kernel and more specifically with its Bluetooth stack - Bluez.

However, instead creating a new service description from scratch, the AndroHID [21] app is calling a function of the sdptool, which is part of Bluez. The sdptool could be accessed from the command terminal and provides an interface for performing SDP queries on Bluetooth devices and other administration operations. It also provides the functionality to directly add a predefined input service record in the SDP registry. AndroHID [21] is calling one of these functions, which inserts a service record, describing a keyboard. Same as the scenario in the Bluetooth touchpad application, once the record is in the registry, it could be requested by the computer host. The keyboard service, which is then provided by the app itself, could be utilized by the computer host.

Although very simple this method proposes several limitations. First, the service description is predefined in the code of the sdptool. Consequently, one could not change or modify it in any way. Second, sdptool provides a service record for regular keyboard and another service record for Wii remote control. A service record for mouse or mouse and keyboard combined is not available. Third, one must carefully examine the HIDDesciptorList of the predefined keyboard service, in order to know how the data reports are to be structured. This means, one should examine the code of the sdptool, which is very poorly documented.

Another drawback in the AndroHID [21] is the fact that it was build for Android, version 1.6. This early version of Android does not support Bluetooth on the application framework level. Hence, the application is directly interacting with the Bluez module in the system core in order to establish a communication channel. However, implementing functions on such low level in the operating system stack ought to be very difficult. Bad implementation might result in very unstable program. As mentioned in the previous chapter, the Touchpad application is taking advantage of the Bluetooth infrastructure, which has been built on both the native libraries level and the application framework level. Furthermore, it has been built by Google engineers for the Android version 2.1 and turns out to be far more stable.

In conclusion, the AndroiHID [21] application is introducing one very simple solution to the common problem. Examining the way AndroHID [21] works has provided some valuable knowledge for building the Bluetooth touchpad application. Naturally, the Bluetooth touchpad app, which has been built several years after the AndroHID [21] and is also taking advantage of newer Android version, is far more powerful. While the AndroHID [21] project is presenting a standalone application, the Bluetooth touchpad project is providing an extensible framework, which could be reused for the development of other Bluetooth-oriented applications. The framework was reviewed in detail in the previous chapter.

7. Software testing

According to Prof. Florian Matthess [22] of the TU Munich, there are four different steps in the process of software testing and accordingly four different types of tests. In order for a software product to be complete, all four must be performed. The logical sequence of the steps is shown on the figure 27 below.

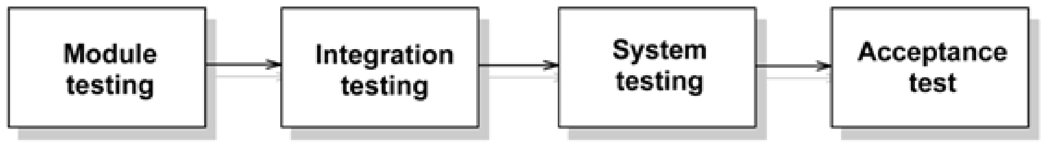


Figure 27: Software testing steps

The first test to perform after the implementation is complete is the module test. A software program typically consists of several components or several modules. These are supposed to be as independent from each other as possible. In the module testing step, each module is tested separately. The influence of other modules is completely ignored. There are two types of module tests: black-box and white-box testing. However, it is highly recommended that both are performed. Black-box tests are testing the functionality of the software. Test cases are built around the software specifications and the predefined requirements. Black-box testing is verifying if the software does what it is supposed to do. Knowledge of the application structure and implementation is not required. White-box tests, on the other hand, are testing the internal structures of the program. In white-box testing, the knowledge about the system internals is required and used to design test cases. The testing developer chooses inputs in order to exercise paths through the code and determine the appropriate outputs. [22]

The second step of integration testing, other than the module tests is not testing a single component, but the interaction of multiple components. There are three methods of integration testing. In the "Bottom-Up Integration" method components are put together in growing subsystems, which are merging to each other in the complete system. The "Integration with Dummies" prescribes that the whole system is to be built with "dummy"-components. The dummies are then to be replaces one by one with working components until the whole system is present. In the "Big Ban Integration" components are directly assembled in the complete system, which is then to be tested. However, this method is not recommended, since the sources of the emerging errors would not be easily found. [22]

The third step - system testing requires a complete system. It is done after the integration test has finished and all components are functional and put together. In the system test, the software is tested from the customer’s perspective. Many of the functions, provided by the software to the user, are working only when the complete system is present. Furthermore, the system test is testing the complete software solution and also the hardware. Conclusively, not only the functionality, but also the performance of the integrated system is measured. [22]

The previous three methods are all performed by the software provider. However, the acceptance test, which is the last step is done by the users/customers themselves at the very end of the project. It is conducted to determine if the requirements of the specification or the particular contract are met. [22]

Nevertheless, all four steps are highly recommended if a large software system is to be developed. On the other hand, the Bluetooth touchpad application is relatively small project. It is done within two months by a single developer. Additionally, the touchpad application is not intended to be released as a public commercial product. The main reason for this is its incompatibility with the Windows operating system, which was presented in a previous chapter. Since the application is not the result of specification or a contract, the acceptance testing is not relevant. Furthermore, being a very small project, the Bluetooth touchpad software does not consist of large number independent components, which are to be tested separately in a module test and together in an integration test, afterwards. In theory, after omitting the module and the integration tests, emerging errors in the system testing would be very difficult to find and remove. However, this deficiency is compensated by the fact that the application is relatively small and is developed by a single software engineer, who has a complete understanding of the system’s architecture and its components. System test that is testing the functionality and the performance of the whole application and the underlying device could turn out to be very useful. The test is performed from the user’s point of view and all the features are tested in different scenarios. If any errors emerge, they are to be immediately removed by the developer.

7.1 Manual vs. automated tests

When testing software, there are two major types of testing methods - manual and automated. So, the tests could be done manually by people, who are specifically haired to try out the product in many different scenarios. The advantage is that a thinking tester will find ways to best explore the application aside from the predefined ways presented to him. However, since the Bluetooth touchpad is small experimental project, which is not intended for the wide comercial public, testing aside the predefined ways is not required. On the other hand, in the automated testing approach, the software vendor is using third-party tools, which will execute several predefined scenarios. The major advantage of this approach is efficiency. Since the tests are done by automated tools the factor of human failure (for example wrong input) is excluded. Another particular advantage, which is very important for the Bluetooth touchpad project is the fact that the automated tests could be easily executed many times and more importantly on different hardware vessels with different versions of the Android firmware. By designing an automated test, the Bluetooth touchpad project provides the opportunity for third-party developers to test the application on different Android devices, running different versions of the operating system. As previously mentioned, some devices are unable to utilize the full power of the software, since they are running custom firmware. The automated test would allow developers to distinguish between these devices.

7.2 Third-party testing tools

The Google engineers have provided some powerful testing tools as part of the Android software development kit. These testing tools have been built around the JUnit test framework. JUnit has been specifically designed to test programs written in the Java programming language. For the automated testing process of the Bluetooth touchpad app, a third-party testing tool is used that in its core is taking advantage of the JUnit framework. Robotium *"is a test framework created to make it easy to write powerful and robust automatic black-box test cases for Android applications. With the support of Robotium, test case developers can write function, system and acceptance test scenarios, spanning multiple Android activities."(*Robotium home page [23]*)* Consequently, Robotium is providing all the functionality needed for the automated test scenarios part of the Bluetooth touchpad project. An Android test project, which is including the Robotium framework, packaged in a jar file, is added to the Bluetooth touchpad repository. Third-party developers are provided the option to execute the test cases and test the Bluetooth touchpad application on their particular Android devices.

7.3 Test requirements and design

In order to create a test plan for a system test, one must refer to the system requirements, which are predefined by the contractor. However, the Bluetooth touchpad application is experimental project and does not have a contractor. Therefore, the requirements would be derived from the definition of the expected system behavior. If the system is behaving according to this definition, the requirements are met. The test plan would consist of several test cases that are introducing different scenarios or different aspects of this expected (required) behavior. The system behavior is outlined by the figure 28 below, which is showing an UML conceptual states diagram. Furthermore, the different states are graphically supported by screenshots of the application in action, which are to be found in appendix C.

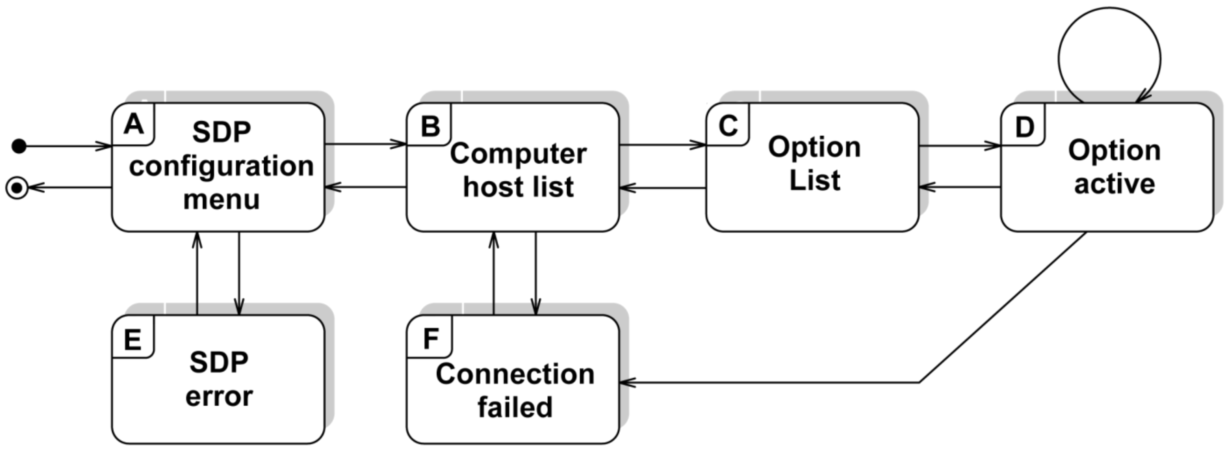


Figure 28: Touchpad application states diagram

Each state is marked with a letter in the left upper corner for better clarity. After starting the application, the user has to choose one of two available SDP configurations (State A). There is a service record describing a regular mouse and keyboard and another describing a pointer and keyboard. After the configuration is chosen, the new service record is inserted in the SDP registry. However, since the Bluetooth framework is using low-level functions and interacting directly with the Linux kernel, this process might fail on some devices. The reason for this is that hardware manufacturers (for example HTC Corp.) are modifying the kernel's source code for their specific needs. Of course, the application would not crash, but handle the situation appropriately by notifying the user of the error with a dialog (State E). Accepting the dialog's message is leading back to the list of available SDP configurations (State A), so the user could try again. If the process of adding the service record in the SDP registry is successful, the user is to choose from a list of paired to the Android device computer hosts (State B). As a matter of fact, interaction with the computer host is also required before choosing its address from the list and attempting connection establishment. The user has to prompt the host to refresh the list of services, which are provided by the Android device in order for the new service record to be found. However, if this step on the computer host side is omitted or the host is unreachable for any reason the connection attempt will fail. The application has to once again handle the error and show an error dialog, which is explaining the situation to the user (State F). Naturally, accepting the dialog's message leads back to the list of devices, so the user could try connecting again (State B). If the connection is successful, the user has to choose from the list of available "Options". (Touchpad, Keyboard, Gamepad, Voice etc.) Once an option is chosen, the user could start utilizing it (State D). When the "Option" is active the application is translating the user interaction with the device into data reports, which are then sent to the computer host. The "Option" could stay active for a time period limited only by the battery charge of the Android device and the computer host respectively. However, if the connection between the Android device and the computer host is interrupted (for example the Laptop runs out of battery), the error message notifying about a connection problem is shown (State F). By accepting it, the user goes back to the list of available devices (State B) and could try reconnecting to the same or another host. While the “Option” is active (State D), the user could go back to the list of available Options (State C) by pressing the back physical button of the Android phone and choose another Option. Furthermore, if the user wants to connect to another host, one presses the back button once again and goes back to the list of paired devices (State B). Another press would lead to the list of SDP configurations, where the user could change the service record and consequently the service configuration in use. The next press of the back button would be the last, since it quits the application.

The Android testing project, utilizing the Robotium black-box testing framework for its purpose, is providing four test scenarios. Each scenario aims to test the behavior of the framework by putting it through a different set of states. Scenario A is testing the behavior of the touchpad software when running on a phone, which has a modified Linux kernel and does not allow the insertion of the new record in the registry. Scenario B is testing the case when the computer host is unreachable and the connection could not be established. Scenario C is simulating the accidental interruption of the connection when the user is already utilizing the input service (the Option is active). Scenario D is testing the complete behavior of the system in its best case - the SDP registry is successfully updated with the new service, the device can connect freely to the computer host and the connection is not interrupted. The tables in the next section present the results of the different test cases for two particular Android devices - the Google Nexus One, which was introduced in a previous chapter and the HTC Desire. The HTC Desire’s hardware is almost identical to the one of the Google Nexus One. However, its Android firmware is customized by the HTC manufacturer.

7.4 Test results and interpretation

SCENARIO	States	GOAL	DEVICE	RESULT
A	A, E, A	Test behavior if service registration fails	HTC Desire	Success

Table 2: Test scenario A, HTC Desire

SCENARIO	States	GOAL	DEVICE	RESULT
A	A, E, A	Test behavior if service registration fails	Google Nexus One	Failed

Table 3: Test scenario A, Google Nexus One

The HTC Desire is completing successfully Scenario A. This means, the device is unable to register the new service. Possible cause of this behavior is the modification made by the manufacturer in the operating system. The Google Nexus One is failing the test. Conclusively, the registration has completed successfully and the error - state E never occurs. For the next tests only the Google Nexus One would be used, since a successful registration of the service is required.

SCENARIO	States	GOAL	DEVICE	RESULT
B	A, B, F,B	Test behavior if connection to host could not be established	Google Nexus One	Success

Table 4: Test scenario B, Google Nexus One

If the host could not be reached (the notebook's Bluetooth feature is disabled), the application is handling the situation as expected.

SCENARIO	States	GOAL	DEVICE	RESULT
C	A, B, C, D, F, B	Test behavior if connection to host is lost while service is utilized	Google Nexus One	Success

Table 5: Test scenario C, Google Nexus One

If the connection is lost while the phone is sending data reports of user interaction events, the applications is showing an error message and is giving the user the option to reconnect manually or to connect to another host. Behavior is as expected.

SCENARIO	States	GOAL	DEVICE	RESULT
D	A, B, C, D, C, B, A	Test behavior in best case scenario	Google Nexus One	Success

Table 6: Test scenario D, Google Nexus One

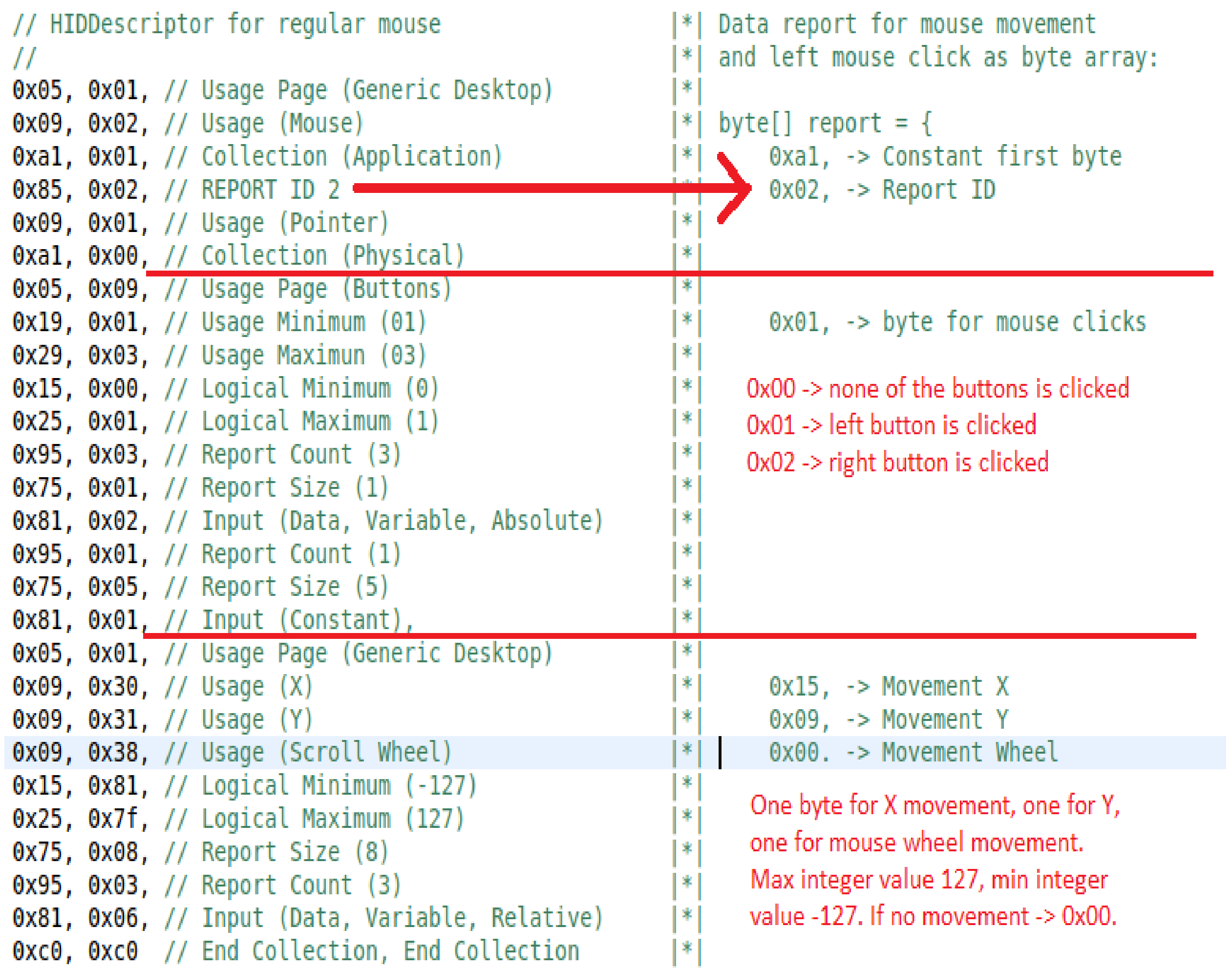
In the best case scenario, everything runs as expected. This proves the application capability of providing an input service. It is highly recommended for developers, wishing to implement the framework, to execute the tests and see if their device is behaving like the Nexus.

8. Conclusion

This work presented the development of the Bluetooth touchpad application, which is designed to run on an Android phone. It allows the user to connect its phone to a Linux computer host and use it as an input device. The Bluetooth touchpad is using the embedded HID drivers of the Linux operating system and does not require the installation of any additional software on the computer side. Due to its requirement for a rooted phone with unmodified Bluetooth stack and furthermore its incompatibility on the computer host side with other operating systems including Windows, the software is not intended for commercial public release on the Android application market. However, the Bluetooth touchpad project provides powerful framework, which could be easily implemented by developers, who want to create applications utilizing the HID protocol. The core advantage of the framework is that developers are not required to have any understanding of the Bluetooth technology, the Service Discovery Protocol and the Human Interface Device protocol. It is the ambition of this project to live through the open source philosophy and help others - companies, developers, etc. in their Bluetooth related initiatives. The source code, licensed under the GPL v3 of both the touchpad application and its testing project are available on the Android Bluetooth Touchpad [20] official website. Furthermore, sample application accompanied with tutorial is providing information on how to utilize and implement the framework. Videos demonstrating the Bluetooth touchpad app in action are also present. In conclusion, the project introduced in this work initially aimed to realize a single use case scenario of Bluetooth touchpad running as application on Android phone. However, it has gone further by not only presenting other use case scenarios, but also providing this vary same opportunity to other third-party developers.

Appendix

Appendix A: HIDDescriptor for regular mouse

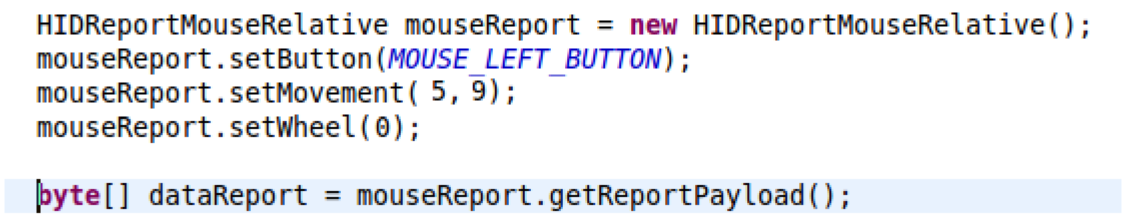


```
// HIDDescriptor for regular mouse                          |*| Data report for mouse movement
//                                                          |*| and left mouse click as byte array:
0x05, 0x01, // Usage Page (Generic Desktop)                 |*|
0x09, 0x02, // Usage (Mouse)                                |*| byte[] report = {
0xa1, 0x01, // Collection (Application)                     |*|     0xa1, -> Constant first byte
0x85, 0x02, // REPORT ID 2                                  |*|     0x02, -> Report ID
0x09, 0x01, // Usage (Pointer)                              |*|
0xa1, 0x00, // Collection (Physical)                        |*|
0x05, 0x09, // Usage Page (Buttons)                         |*|
0x19, 0x01, // Usage Minimum (01)                           |*|     0x01, -> byte for mouse clicks
0x29, 0x03, // Usage Maximun (03)                           |*|
0x15, 0x00, // Logical Minimum (0)                          |*|
0x25, 0x01, // Logical Maximum (1)                          |*|
0x95, 0x03, // Report Count (3)                             |*|
0x75, 0x01, // Report Size (1)                              |*|
0x81, 0x02, // Input (Data, Variable, Absolute)             |*|
0x95, 0x01, // Report Count (1)                             |*|
0x75, 0x05, // Report Size (5)                              |*|
0x81, 0x01, // Input (Constant),                            |*|
0x05, 0x01, // Usage Page (Generic Desktop)                 |*|
0x09, 0x30, // Usage (X)                                    |*|     0x15, -> Movement X
0x09, 0x31, // Usage (Y)                                    |*|     0x09, -> Movement Y
0x09, 0x38, // Usage (Scroll Wheel)                         |*|     0x00. -> Movement Wheel
0x15, 0x81, // Logical Minimum (-127)                       |*|
0x25, 0x7f, // Logical Maximum (127)                        |*|
0x75, 0x08, // Report Size (8)                              |*|
0x95, 0x03, // Report Count (3)                             |*|
0x81, 0x06, // Input (Data, Variable, Relative)             |*|
0xc0, 0xc0  // End Collection, End Collection               |*|
```

0x00 -> none of the buttons is clicked
0x01 -> left button is clicked
0x02 -> right button is clicked

One byte for X movement, one for Y, one for mouse wheel movement. Max integer value 127, min integer value -127. If no movement -> 0x00.

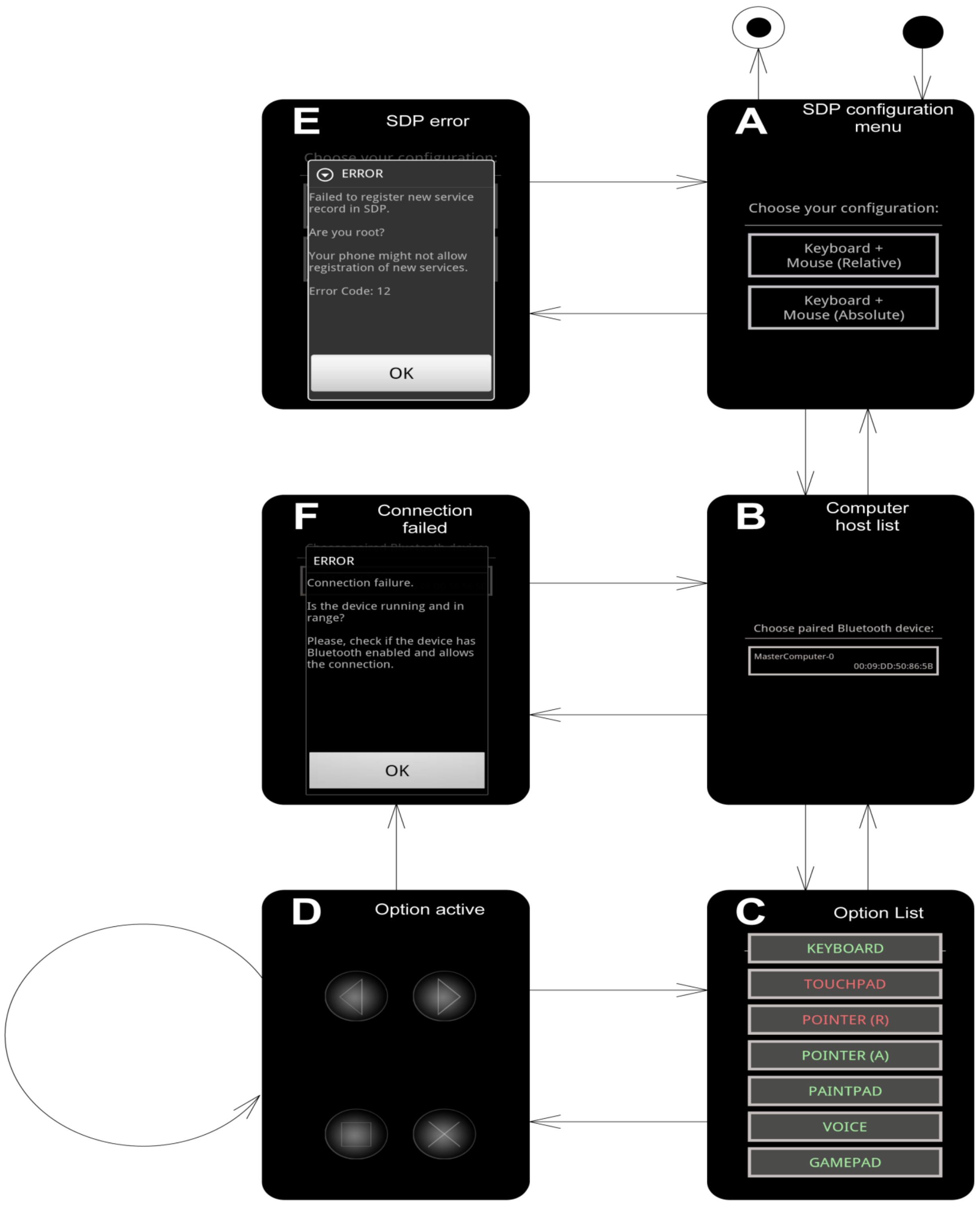
Appendix B: Report package, example usage



```
HIDReportMouseRelative mouseReport = new HIDReportMouseRelative();
mouseReport.setButton(MOUSE_LEFT_BUTTON);
mouseReport.setMovement( 5, 9);
mouseReport.setWheel(0);

byte[] dataReport = mouseReport.getReportPayload();
```

Appendix C: Bluetooth touchpad UI states


E
SDP error
ERROR
Failed to register new service record in SDP.
Are you root?
Your phone might not allow registration of new services.
Error Code: 12
OK
A
SDP configuration menu
Choose your configuration:
Keyboard + Mouse (Relative)
Keyboard + Mouse (Absolute)
F
Connection failed
ERROR
Connection failure.
Is the device running and in range?
Please, check if the device has Bluetooth enabled and allows the connection.
OK
B
Computer host list
Choose paired Bluetooth device:
MasterComputer-0
00:09:DD:50:86:5B
D
Option active
C
Option List
KEYBOARD
TOUCHPAD
POINTER (R)
POINTER (A)
PAINTPAD
VOICE
GAMEPAD

Glossary

Glossary

Human Interface Device Human Interface Device is specifying either a specific class of devices or the type of device known as Human Interface Devices, for example the keyboard is such a device.

Application Programming Interface An Application Programming Interface 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.

Service Discovery Protocol The service discovery protocol provides a mean for applications to discover which services are available and to determine the characteristics of those available services.

Software Development Kit A Software Development Kit is typically a set of development tools that allows for the creation of applications for a certain software package, software framework, hardware platform, computer system, video game console, operating system, or similar platform.

Native Development Kit In the Android universe, the Native Development Kit (NDK) is a companion tool to the SDK, which lets you build performance-critical portions of your apps in native code.

Host Controller Interface In the Bluetooth stack, the Host Controller Interface provides a command interface to the baseband controller and link manager and access to hardware status and control registers. This interface provides a uniform method of accessing the Bluetooth baseband capabilities.

Radio Frequency Radio Frequency is a frequency within the electromagnetic spectrum associated with radio wave communication. Many wireless technologies are based on Radio Frequency field communication.

Quality of Service Quality of Service is a networking term that specifies a guaranteed amount of data transferred from one place to another or processed in a specified amount of time.

General Public License The General Public License is the license that accompanies some open source software and details how the software and its source code can be freely copied, distributed and modified.

Universal Serial Bus Universal Serial Bus is external bus standard that supports the physical connection of peripheral devices, such as mice, modems, and keyboards.

Personal Digital Assistant The Personal Digital Assistant is a handheld device that combines computing, telephone, Internet and networking features.

Wii The Wii is a small video game console manufactured by Nintendo Corp.

Unified Modeling Language The Unified Modeling Language is set of general-purpose notations for specifying and visualizing complex software and architectural concepts.

Unicode Unicode is a standard in computing for representing characters as integers

Universal Transformation Format The Universal Transformation Format is a method for converting Unicode characters, which are 16 bit long. UTF-8 is compressing them into 8-bits characters.

Virtual Machine The Virtual Machine is a self-contained operating environment that behaves as if it is a separate computer.

Java Native Interface The Java Native Interface is programming framework that enables Java code running in a Java VM to call libraries written in other languages such as C, C++.

Radio Frequency Communication (RFCOMM) RFCOMM is a high level Bluetooth transport protocol built on top of the L2CAP protocol.

Java Java is high-level programming language developed by Sun Microsystems.

Java Development Kit The Java Development Kit is a SDK for Java Programms.

HID	Human Interface Device
API	Application Programming Interface
SDP	Service Discovery Protocol

SDK	Software Development Kit
NDK	Native Development Kit
HCI	Host Controller Interface
RF	Radio Frequency
QoS	Quality of Service
GPL	General Public License
USB	Universal Serial Bus
PDA	Personal Digital Assistant
UML	Unified Modeling Language
UTF	Universal Transformation Format
JDK	Java Development Kit
VM	Virtual Machine
JNI	Java Native Interface
RFCOMM	Radio Frequency Communication

Bibliography

Bibliography

[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] Open Handset Alliance. (2011) Android. [Online]. http://www.openhandsetalliance.com/index.html

[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] Albert Huang. (2008) An Introduction to Bluetooth Programming. [Online]. http://people.csail.mit.edu/albert/bluez-intro/c404.html

[13] ISO 639:1988 (E/F). Code for the representation of names of languages. [Online]. http://cool.conservation-us.org/lex/iso639.html

[14] USB Org. Universal Serial Bus Specification. [Online]. http://www.usb.org

[15] USB Org. (2001, June) Device Class Definition for Human Interface Devices v. 1.11. [Online]. http://www.usb.org/developers/devclass_docs/HID1_11.pdf

[16] Bluetooth Org. (2006, Dec.) Bluetooth Assigned Numbers - Bluetooth baseband. [Online]. http://netlab.cs.ucla.edu/wiki/files/class_of_device.pdf

[17] USB Org. (2000, Mar.) USB Language Identifiers. [Online]. http://www.usb.org/developers/docs/USB_LANGIDs.pdf

[18] Speed Software. (2011, May) Root Explorer. [Online]. https://market.android.com/details?id=com.speedsoftware.rootexplorer&feature=search_result

[19] Oracle Inc. The Reflection API. [Online]. http://download.oracle.com/javase/tutorial/reflect/index.html

[20] Nikolay Kostadinov. (2011, Oct.) Android Bluetooth Touchpad Project. [Online]. http://code.google.com/p/android-bluetooth-touchpad/

[21] Manuel L. AndroHID. [Online]. http://code.google.com/p/androhid/wiki/AndroHid

[22] Florian Mathes, "Einführung in die Softwaretechnik," 2009.

[23] (2011) Robotium. [Online]. http://code.google.com/p/robotium/

[24] Bluez Project. (2010) Bluez. [Online]. http://www.bluez.org/