ARCHITECTURAL OVERVIEW OF INTEL’S BLUETOOTH SOFTWARE STACK

Kris Fleming, Mobile Computing Group, Intel Corporation
Uma Gadamsetty, Mobile Computing Group, Intel Corporation
Robert J Hunter, Mobile Computing Group, Intel Corporation
Srikanth Kambhatla, Mobile Computing Group, Intel Corporation
Sridhar Rajagopal, Mobile Computing Group, Intel Corporation
Sundaram Ramakesavan, Mobile Computing Group, Intel Corporation

Index words: Bluetooth, mobile computing, WDM, Windows

ABSTRACT
Bluetooth wireless technology was created to enable many different usage models from networking to cable replacement. Implementation of these usage models and of issues specific to Bluetooth, such as service discovery and security, involve mapping the Bluetooth protocol stack into operating systems’ frameworks in order to ensure seamless integration. This paper describes the architecture of the software stack implemented at Intel to support the usage models on Microsoft’s operating systems based on WDM technology.

The Bluetooth Special Interest Group (SIG) was launched in May 1998. Its goal was to develop the specifications for a low-powered, short range, RF-based wireless communication technology. When added on to a notebook computer, a Bluetooth module enables the notebook computer to talk wirelessly to other Bluetooth devices including cellular phones, PDAs, headsets, access points, and other notebook computers.

An implementation based on the architecture described in this paper is available from Intel and was demonstrated at the December 1999 LA Bluetooth Developer’s Conference and the Spring 2000 Intel Developer’s Forum.

INTRODUCTION
Imagine a person with a PDA walking into an office and automatically waking up the notebook computer as a reaction. Further imagine that the notebook computer automatically synchronizes with the PDA for any calendar and task information entered overnight. When reminded of a meeting, the notebook computer is taken to a nearby conference room for a presentation while maintaining Intranet connectivity using a LAN access point. In the conference room business cards are exchanged electronically amongst the participants using notebook computers, PDAs, and cellular phones. All of this can be achieved with Bluetooth software support in the notebook computers. Bluetooth technology promotes the Always On, Always Connected and Anything, Anywhere, Anytime vision of seamless mobile computing for mobile personal computers (PCs) [1].

GOALS OF THE PAPER
This paper presents an overview of the architecture for Intel’s Bluetooth software stack on a PC. It attempts to provide answers to the following questions:

• What does the Intel stack do?
• How do third-party Bluetooth application developers write applications specific to Bluetooth technology?
• How can third-party Bluetooth hardware or device developers develop custom drivers for their hardware that work with the Intel stack?
• What is expected from a Bluetooth device so that it interoperates with the Intel stack?

INDEX WORDS: Bluetooth, mobile computing, WDM, Windows

ARCHITECTURAL OVERVIEW OF INTEL’S BLUETOOTH SOFTWARE STACK

1 All information contained in this paper relating to Intel’s Bluetooth software product and future plans are subject to change without notice.
This paper can also serve as an example for similar development efforts in other operating systems.

**Bluetooth Usage Models and Architecture Goals**

The Bluetooth Specification [4, 5] addresses a variety of usage models for Bluetooth devices. Intel’s implementation of the Bluetooth stack either directly enables or provides the infrastructure for supporting the following Bluetooth SIG 1.0 usage models in its 1.0 release:

- **Object transfer.** This is an ability to transfer an object from one device to another. Objects include files and directories.

- **Data access points.** Access points enable access to the Internet, e-mail, and fax for these types of wireless connections for a notebook computer with Bluetooth: a) to a PSTN plug, b) to a cellular phone, and c) to a LAN access point.

- **Synchronization.** This involves synchronization of business cards, calendars, and task information between Bluetooth devices.

- **Ultimate Headset.** A Bluetooth headset can be wirelessly connected to a PC and appear as an audio playback or recording device.

In addition to the SIG usage models mentioned above, additional goals of Intel’s implementation of the Bluetooth stack include the following:

- **Support for legacy PC applications.** There are a variety of legacy comm port applications for file transfer in the market today such as ProComm Plus® and Intellisync®. Supporting these legacy applications for achieving wireless file transfer between PCs enabled with Bluetooth technology would be convenient for end users.

- **Support for applications native to the OS.** This is part of Intel’s goal of seamless integration into the OS. When a Bluetooth device comes in range of a notebook computer, this support enables the applications that are already in the OS (such as dial-up networking and direct cable connect) to be able to work with the new device.

- **Support for APIs native to the OS.** This is the other goal of proper integration into the OS: enabling APIs native to the OS wherever possible to access Bluetooth devices. One example is the support for Win32 Comm API for accessing Bluetooth serial ports.

- **Support for third-party development.** Part of the goal of industry standard implementation is to enable third parties to add value to the software stack without re-implementing the entire stack. To accomplish this goal, Intel plans to publish a user-mode API interface for functionality specific to Bluetooth, which is not already covered in APIs native to the OS, and a new kernel-mode API. The user-mode API is for development of user-mode applications for Bluetooth technology, while the kernel-mode API is for custom drivers for Bluetooth devices.

**Bluetooth Hardware Overview**

The Bluetooth specification defines a low-power short-range radio and protocol stack supporting a personal communication bubble. The radio has an operational range of 10 meters at 0 dbm and operates in the unlicensed 2.4 GHz ISM frequency band. A single connection supports a maximum asymmetric data transfer rate of 721 kbits/s or a maximum of three voice channels. The voice channels use synchronous communication and support mono-audio using a 64 kbit/s CVSD-encoded stream. All connections use a Time-Division Duplex (TDD) scheme to support bi-directional communication.

At the start of any connection, the unit initiating the connection is assigned temporarily as a master. This assignment is valid only during this connection. The master may have active connections to up to seven other units, known as slaves. The configuration of a master unit connected to one or more slave units is called a piconet. Link management mechanisms allow radio units to time-division-multiplex between master and slave, allowing them to act as bridges between piconets, forming a scatternet. Mechanisms also exist to allow both masters and slaves to request new connections and accept new connections. The goal of the radio specification is to enable multiple virtual cables rather than a single cable-replacement capability.

Spread spectrum technology is used to operate in noisy environments and to allow multiple piconets to co-exist without a noticeable loss in throughput. A frequency-hopping scheme, with a rate of up to 1600 hops per second over 79 one MHz channels, is used to spread the signal over the entire available ISM spectrum.

Various error correction schemes are available to support reliable channels. Packets may be protected using 1/3 and 2/3 rate forward error correction (FEC) schemes to help correct bit errors caused by weak signals near the range.

---

* Other brands and names are the property of their respective owners.
limit. An automatic repeat request (ARQ) scheme helps to reliably transmit link-level frames.

The radio’s link-level protocol supports both authentication and privacy, allowing users to develop a domain of trust between their personal devices. Connections may require a one-way, two-way, or no authentication. Authentication is based on a challenge-response algorithm. Encryption is used to protect the privacy of the connection. A stream cipher well suited to a silicon implementation is used with secret key lengths of up to 128 bits (selectable via 8-bit granularity). The authentication and privacy support are appropriate for the short-range nature of the radio, and applications requiring stalwart protection are encouraged to implement stronger security mechanisms.

Bluetooth Protocol Overview

Bluetooth specifications include protocols [4] and profiles [5]. Protocols specify the workings of an individual component (RFCOMM or L2CAP, for example), while the profiles specify how a set of protocols can be used for implementing a particular usage model. Profiles are important to have a common understanding of the protocol stack in order to promote interoperability of usage model implementations.

The Link Management Protocol (LMP), baseband, and radio are typically implemented in the Bluetooth hardware modules. These modules can interface to the host using different interfaces. However, all Bluetooth controllers should implement the Bluetooth Host Controller Interface (HCI).

The Logical Link Control and Adaptation Protocol (L2CAP) implements a second link-layer protocol to address protocol multiplexing, segmentation, and re-assembly. L2CAP hosts a set of client protocols. A couple of such protocols are the Service Discovery Protocol (SDP) and a serial cable emulation protocol called RFCOMM. Figure 1 summarizes the Bluetooth protocol stack.

A more complete description of the protocols and profiles can be found in the Bluetooth specifications.

A Note on Implementation Target

The target operating systems are WDM operating systems. For Release 1.0 implementation of the architecture...
described here, we are targeting Windows98*, Windows98SE* and Windows2000*. The primary hardware target is notebook computers with embedded Bluetooth USB [2] modules, although appropriate hardware could enable the software stack on desktops also.

**FUNDAMENTAL DESIGN**

The architecture is based on the notion of treating the radio interface as a logical bus in the PC. Devices dynamically come into or go out of communication range of the Bluetooth module built into the notebook computer. This is analogous to hardware being plugged into or taken out of a physical bus inside the PC. The differences are in terms of the frequency with which Bluetooth devices transition into or out of communication range, and the large number (potentially) of Bluetooth devices that get virtually plugged into the PC when compared to a physical bus.

The responsibility of enumerating devices that are in range of the Bluetooth bus rests with an RF Bus Driver (RFBD). In addition to WDM bus driver functionality, RFBD processes the results of the inquiry process for discovering Bluetooth devices in range: the user drives this inquiry process. This is followed by discovery of services within these devices using Bluetooth SDP. The protocol part of SDP is implemented by RFBD. The top-level searches are handled in the Bluetooth Executive (described in the next section).

The Bluetooth usage models are supported by RFBD loading the appropriate client drivers based on device and service information obtained above.

Custom drivers can be loaded in response to devices discovery using Bluetooth Plug and Play [7]. This provides a framework for third parties to leverage unique features in their devices while reusing the Intel stack.

**IMPLEMENTING THE BLUETOOTH PROTOCOL STACK**

RFBD is at the core of Intel’s Bluetooth software stack as shown in Figure 3. It co-exists with an HCI driver in the same binary. The HCI driver is responsible for all interactions with the Bluetooth Host Controller present on the Bluetooth module interfacing to the PC. It also serves to abstract the Bluetooth hardware interface in terms of a private interface exposed by it.

---

* Other brands and names are the property of their respective owners.

---

2 While multiple Bluetooth radios can physically be present in the same notebook computer, multiple such radios within a meter of each other are limited by RF physics on how well they work together.
Figure 4: USB interfaces to access HCI and data

The HCI commands are transported using the USB control pipe. HCI events are transported on the USB interrupt pipe, asynchronous data are transported as USB bulk data, and synchronous data are transported over USB isochronous pipes as specified in the HCI USB specification. The Bluetooth USB minidriver on the operating systems’ USB stack is responsible for abstracting the HCI transport specifics and presenting the data, events, and commands to the HCI driver in a transport-independent manner. An IRP-based [6] interface is defined under the HCI driver for this purpose, and all HCI transport drivers use this interface to talk to the HCI driver. Having such a common interface enables support for different HCI transport drivers.

The Bluetooth drivers loaded by the RFBD (referred to henceforth as the client drivers) talk to the RFBD using a kernel-mode interface called the RF Bus Driver Interface (RFBDI). The RFBDI is similar to other bus driver interfaces published by Microsoft such as the USBDI* or the 1394BDI* [6].

Client drivers typically implement minidriver or port driver functionality for some class driver in the OS. This is important to ensure seamless integration into the OS. The class drivers expose interfaces to these drivers as domain-specific APIs in the user mode.

There is one key module in the user mode and it is called the Bluetooth Executive (BTEXEC). Among other tasks, the BTEXEC is responsible for exposing a user-mode interface that all user-mode applications can use to access Bluetooth functionality. This interface is called the Bluetooth API (BTAPI).

All Bluetooth UI components developed by Intel use BTAPI to interface with the driver stack. The idea is that the BTAPI should contain all the information typically needed from the kernel stack by third-party application developers.

There could be transport-specific, platform-specific, or vendor-specific drivers not covered here.

Kernel-Mode Client Drivers

Two client drivers that are needed for supporting the usage models mentioned in this paper are the RFCOMM driver and the audio driver, as shown in Figure 5 below.

Figure 5: RFCOMM driver emulates serial ports

The RFCOMM driver is responsible for implementing multiple virtual comm ports over RFCOMM connections with each device supporting the protocol. These virtual comm ports are used to support both legacy comm port and IrOBEX-based applications for Bluetooth technology. The latter category includes file transfer and synchronization. RFCOMM is implemented as several drivers that together implement the RFCOMM protocol, comm port emulation functionality, and the interfaces to either VCOM M or serial.sys, depending on the OS.

In addition, PPP over L2CAP is the basis for dial-up networking and PPP-based LAN access points. These can also be supported by the RFCOMM driver set mentioned above.

The audio driver is responsible for exposing the headset as a playback and recording device to the system. It is also responsible for processing the audio stream coming over the air interface and piping it into the streaming class driver.

Programming Interfaces

RFBDI is an IRP-based kernel mode interface similar to USBDI or 1394 BDI interfaces published in the DDKs for

---

* Other brands and names are the property of their respective owners.
Windows 2000*. It supports access to the following features:

- audio data stream
- connection-oriented data stream
- connectionless L2CAP interface
- connection-oriented event handling (examples of these events include those from L2CAP, SDP, and security)
- kernel-mode service discovery functions
- Bluetooth TCI
- L2CAP QoS

BTAPI provides access to features specific to Bluetooth technology in the kernel stack that typically cannot already be accessed through existing Windows* APIs. BTAPI provides access to the following:

- Information relating to Bluetooth device discovery. This includes access to local and remote device information and properties, and an ability to initiate device discovery.
- Radio configuration and status. This includes an ability to change duration and length of page and inquiry scans, access to RSSI, and an estimation of link quality.
- Service discovery information. This includes access to device services and attributes, and an ability to add or remove services on the local host.
- Security information. The user can set Bluetooth security modes for the local device, authorize use of a device in an ad-hoc manner, and communicate trust status with the stack.
- Power management. The user can set radio power states using BTAPI and control wake-up states in the software.

Applications use the Win32* Comm API for performing open, close, read, write, and flow control functions on the virtual comm ports exposed by the RFCOMM driver. In the case of the server, the application creates an SDP record for its service, which results in the creation of a comm port being created, opens the port, and connects to the server.

Usage of virtual comm ports for Bluetooth technology is characterized by the following features:

- Applications need to close and re-open the ports after each session.
- Link loss is indicated by dropping all communication signals low (this includes CD, CTS, DTR, DSR and RTS). The applications need to close the port on a link loss.
- Any reads and writes to the port will fail when the devices are not connected.
- BTAPI functions are needed for addition and deletion of SDP records, inquiry, and service discovery.
- There is no support for baud rate pacing; 5-, 6-, and 7-bit word lengths; parity and stop bits; and xon / xoff flow control.

**INTEL UI AND END-USER EXPERIENCE**

Intel's UI for controlling and configuring Bluetooth connections and devices is closely integrated into the OS UI components. Some of the UI elements are as follows:

- A Windows Explorer* extension for displaying and managing remote devices and connections. The Explorer extension also supports file transfer by drag and drop of the files to be transferred.
- A control panel applet specifically for managing the local Bluetooth device. It is also used for changing policies applicable to the local device.
- A system tray icon for displaying radio status, security-related pop-ups, and connection statistics.

As noted earlier, several native OS applications are enabled on Bluetooth devices, and (as described in the section on open architecture below) third parties have the ability to develop independent Bluetooth applications on the Intel stack.

A few usage scenarios are described below illustrating different aspects of the end-user experience.

**File Transfer**

Files can be transferred over the air in a couple of different ways:

- The user can perform a drag and drop operation onto the destination device using the Explorer extensions.

* Other brands and names are the property of their respective owners.
A independent file transfer application could be launched. This application could be a legacy file transfer application such as ProComm Plus or Intellisync, or it could be a Bluetooth application.

To illustrate further the end-user experience with the first scenario, the file transfer application on one notebook computer is typically started, and the Explorer is launched on the second notebook computer. The user would then search for the first computer under the Bluetooth Devices folder under the My Computer category of the Explorer and drag and drop the desired file onto the first notebook computer.

**Device Discovery**

Device discovery can be triggered manually or automatically at some preset intervals. This is a policy that needs to be set using the control panel. Once such a policy is set, remote devices are discovered when the appropriate trigger fires. Bluetooth specifications allow a device to be in non-discoverable mode where it will not be scanning for device discovery requests from other devices.

**DEVICE INTERACTION**

Figure 6 shows the interactions that result when a file is transferred between two devices using Bluetooth technology.

![Figure 6: File transfer between devices](image)

On the left-hand side of the picture you see a PC with a Bluetooth module interfacing to the USB controller. The right-hand side of the picture shows another Bluetooth device. For the purposes of this example, it is assumed that both devices implement file transfer profile. An inquiry process results in discovery of the device in the UI and an attempt to obtain services result in the PC learning about the file transfer service on the remote device.

The Bluetooth modules in the two devices have the radio, baseband, and LMP implemented in them. They also have HCI implemented as a means of interfacing with the host controller. On the PC side, access to the device HCI is enabled through USB. The PC communicates with the other device using over-the-air packets.

The inquiry process is initiated by a user command (or is initiated automatically after some fixed period of time, based on user policy settings on the PC), and it results in a command to the Bluetooth host controller on the module through the HCI. This results in the LMP and baseband firmware executing the inquiry sequence. The PC discovers the other Bluetooth device, and this is communicated back to the host driver stack through some HCI events carried across USB. Thus the HCI driver communicates virtually with the HCI firmware on the host Bluetooth module to learn about the new device. To achieve this, the USB minidriver interfaces virtually with the USB firmware on the module, and the USB stack, native to the OS, interfaces with the USB controller and interfaces with the USB hardware on the host module.

Once the HCI driver knows about the new device through the event, the RFBD propagates this information to the user. The user drives further actions on the device, particularly those relating to service discovery. In order to perform service discovery, the SDP and L2CAP implementations in the PC and the other Bluetooth device communicate in accordance with the Bluetooth specifications and profiles.

Once the PC discovers that the other device supports file transfer, the user on the PC can initiate file transfers to the other device. In this case, the PC is the client using the file transfer service on the other device. Initiation of file transfer is performed by a drag and drop operation on the file. The RFCOMM driver on the PC stack communicates with the RFCOMM implementation on the device to set up the RFCOMM channels on the L2CAP.

Once RFCOMM channels have been established, the two applications use the IrOBEX protocol to exchange files, in accordance with the Bluetooth profiles.

This example has glossed over details relating to individual implementation features including inquiry, driver loading, service discovery, L2CAP, security, RFCOMM, virtual comm port creation, and user interfaces.
OPEN ARCHITECTURE

There are several ways in which third parties can add value to the stack.

Third-party PC hardware needs an HCI transport driver to work with the Intel stack. Such a driver will interface with the HCI driver and would be responsible for communicating with the hardware to deliver HCI commands from the PC and to return HCI events and data back to the PC.

Third-party kernel-mode client drivers are layered above the RFBD. These drivers handle devices external to the PC that communicate with the PC using Bluetooth protocols. The main goal for these drivers is to enable custom handling of the hardware in a way that best uses innovative features in the hardware while still retaining the Bluetooth SIG standard over the air interface. Bluetooth Plug and Play (explained below) is used to load these custom drivers on RFBD.

Third-party user-mode applications are also enabled on the stack to enable custom handling of device or Bluetooth UI. These applications interface to the stack by using BTAPI for functionality specific to the Bluetooth technology and by using standard Microsoft APIs (such as the Win32 comm API) for data transfers.

These third-party applications can be independent applications or they can be launched from within the Intel UI. An application that falls in the latter category is the file transfer application (when a file is dragged and dropped, for example).

Third-party user-mode applications can also be layered on third-party kernel-mode client drivers. This happens when the kernel-mode drivers export an API that is used by the user-mode applications.

SUPPORT FOR BLUETOOTH FEATURES

This section addresses Bluetooth security, power management, and Bluetooth Plug and Play issues.

Security

Bluetooth security has its basis in encryption, authentication, and authorization. Bluetooth security modes work differently depending on whether they are turned on and on which layer is enforcing them.


Power Management

BTAPI allows configuration of several power-related items:

- It enables radio power to be automatically set depending on the system power state.
- It allows control of page and inquiry scan policies for the radio.
- It enables control of wake-up policies: the system can wake up when a device is discovered, or it can be woken up when a new connection request has been received. Both of these can be set for a particular device, a particular class of device, or for any device.
- Device discovery policy determines when the discovery process is initiated and whether it is initiated automatically under system control. This policy can be set on a per system power state basis.
- The events indicated by HCI can also be selectively masked on a per system power state basis.

Not all of these are necessarily exposed to the end user, but these are capabilities available for applications through BTAPI. The applications can expose these to the users as appropriate.

BLUETOOTH PLUG AND PLAY

The need for custom drivers for third-party hardware drives the need for Bluetooth Plug and Play (PnP) [7]. Third parties would want to write custom drivers for their hardware to make use of the unique features of their hardware. Bluetooth PnP provides a means for identification of the right driver to load in response to devices discovered.

SUMMARY

This paper presents an overview of what the Intel Bluetooth software stack implements. This could be viewed as an example Bluetooth implementation on the PC platform.

The stack provides for feature enhancements both from ISVs and IHVs.

Third-party Bluetooth application developers can add user-mode components on top of BTAPI for application
management, and Win32 comm API for data transfer and flow control. In addition, kernel-mode drivers developed on RFBDI can expose a user-mode framework that is different from BTAPI, if needed.

Drivers can be developed on RFBDI for several reasons, with a custom user-mode framework being one of them. Other reasons include the development of support for new usage models and the development of custom support for new hardware devices. The Intel stack provides the infrastructure on which new usage models can be hosted. This infrastructure includes implementation of the base Bluetooth protocols. Implementation of custom drivers for new hardware uses the Bluetooth PnP framework.

In order to interoperate with the Intel stack, device manufacturers need to conform to the Bluetooth air protocol and the applicable profiles.

Following are the key takeaways from Intel’s driver stack implementation:

- The driver stack is for WDM-based operating systems like Windows98® and Windows2000®. It is to be noted that in Windows98, we do need a VxD module to interface with VCOMM.
- The driver stack can be enabled on any PC platform. For enabling it on desktops, hardware solutions like a USB dongle are needed.
- The stack supports Bluetooth SIG 1.0 specifications and profiles.
- The stack enables native OS applications on Bluetooth devices where possible.
- BTAPI enables access to functionality specific to Bluetooth technology from user-mode applications.
- RFBDI enables third parties to write kernel-mode drivers for their Bluetooth peripherals.

ACKNOWLEDGMENTS
Ron Mosgrove, Jon Inouye, Steve Lo, Chad Zhu, Bailey Cross, Bruce Clemens, Anurag Prakash, Diane Stevenson, and Jim Kardach were also involved in the architecture definition at various stages. Thanks are also due to Jon, Ron and Steve for detailed review of the final paper.

REFERENCES

AUTHORS’ BIOGRAPHIES
Kris Fleming is a Senior Software Engineer in the Bluetooth Communication Architecture group at Intel Corporation. He has a B.S. degree in electrical and computer engineering and a M.S. degree in computer engineering from the University of Maine. In addition, he has an M.S. degree in information networking from the Information Networking Institute at Carnegie Mellon University. His research work has focused on mobile computing and wireless networking. Kris was one of the initial members of the Intel team, chaired the Bluetooth SIG HCI working group, and is currently the co-chair of the Personal Area Networking Bluetooth 2 SIG working group. His e-mail is kris.d.fleming@intel.com.

Uma Gadamsetty joined Intel in 1995 as a senior software engineer and is a team member of Mobile Communications Operation. He has developed device drivers and protocol implementations under a wide array of operating systems. His e-mail is uma.gadamsetty@intel.com.

Robert J. Hunter is a senior software engineer in the Mobile Communications Operations department. Robert is responsible for Ambler software development and led the Bluetooth USB HCI Specification task force. His email is robert.j.hunter@intel.com.

Srikanth Kambhatla has been working on Windows NT OS and HAL extensions and drivers in many capacities in Intel since 1991. In addition to Bluetooth, he has been involved in the definition of PC Plug and Play, Multiprocessor specs for PCs, IrDA Control and IrDA PnP standards. Srikanth has an M.S. degree in computer science and engineering. His e-mail is srikanth.kambhatla@intel.com.

* Other brands and names are the property of their respective owners.
Sridhar Rajagopal has been at Intel since 1993. He has a M.S. degree in electrical engineering from Arizona State University. He has worked in various software development organizations and projects at Intel and is currently working on the Intel Bluetooth Software suite. His e-mail is sridhar.rajagopal@intel.com.

Sundaram Ramakesavan has an M.S. degree in computer science from Queen's University, Canada. He has worked in various Telecommunications and Data Communications projects at Nortel Networks. He is currently working on the UI elements of the Intel Bluetooth Software suite. His e-mail is ramu.ramakesavan@intel.com.