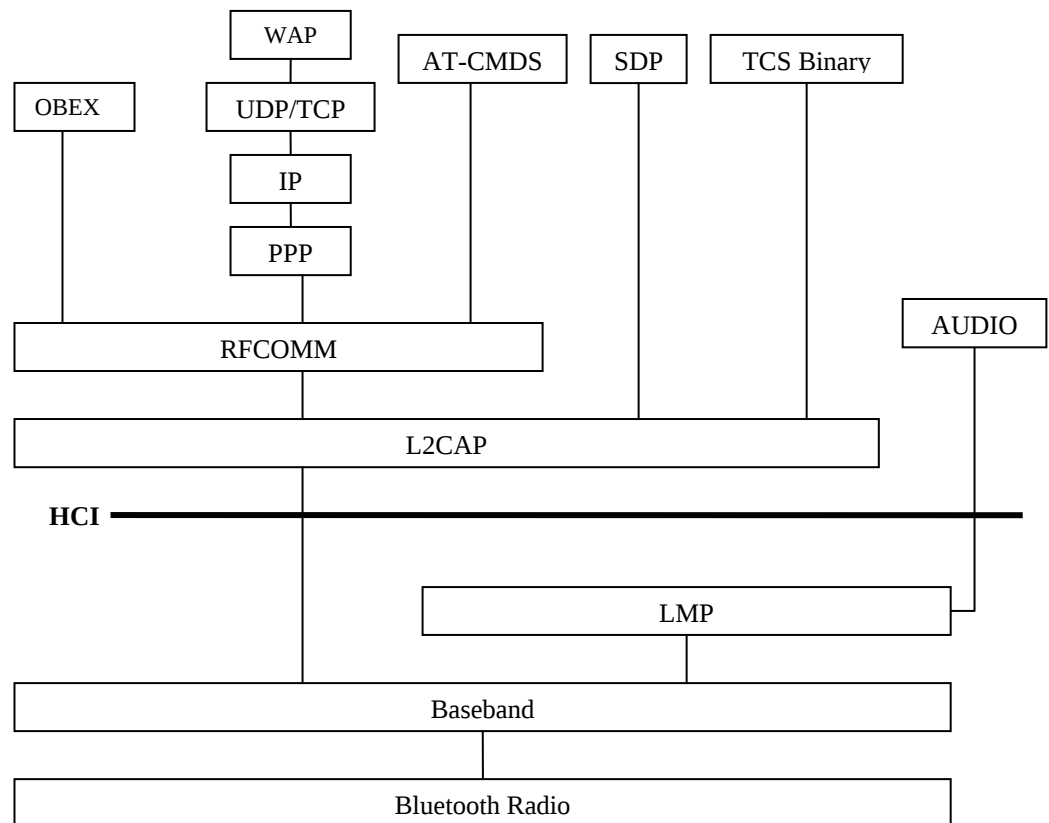


2. LANDASAN TEORI

2.1. *Bluetooth*

Bluetooth adalah teknologi komunikasi *wireless* dengan jarak yang cukup pendek (maksimal 100 meter) yang memakai frekuensi 2,4 GHz. Pada saat ini teknologi *Bluetooth* mulai diterapkan pada telepon selular sebagai fasilitas koneksi antar telepon selular tanpa menggunakan jaringan komunikasi *provider*. *Bluetooth* memiliki keunggulan terpenting daripada koneksi dengan *infrared*, yaitu *Bluetooth* tidak dibatasi dengan LOS dan koneksi dengan *Bluetooth* mampu mengirimkan data dengan kecepatan rata-rata 1 Mbps.

Bluetooth Protocol stack dapat dibagi menjadi 2 komponen, yaitu *Bluetooth host* dan *Bluetooth controller* (atau *Bluetooth modul radio*). *Host Controller Interface* (HCI) menyediakan sebuah standarisasi *interface* antara *Bluetooth host* dengan *Bluetooth controller*. Gambar 2.1. adalah gambar blok diagram dari *Bluetooth protocol stack*.



Gambar 2.1. Gambar blok diagram *Bluetooth Protocol stack*

Sumber : Sun Microsystems. *Java APIs for Bluetooth Wireless Technology (JSR-82), Specification version 1.0a*. Austin: Motorola Wireless Software, Applications & Services, 5 April 2002 : 9

Tabel 2.1. Tabel Protokol dan *Layer* pada *Bluetooth Protocol stack*

Protocol Groups	Protocol in the Stack
Bluetooth Core Protocols	Baseband, Link Manager Protocol (LMP), L2CAP and SDP
Cable Replacement Protocol	RFCOMM
Telephony Control Protocol	TCS Binary
Adopted Protocol	PPP, UDP/TCP/IP, OBEX, WAP

Sumber : Sun Microsystems. *Java APIs for Bluetooth Wireless Technology (JSR-82), Specification version 1.0a*. Austin: Motorola Wireless Software, Applications & Services, 5 April 2002 :8

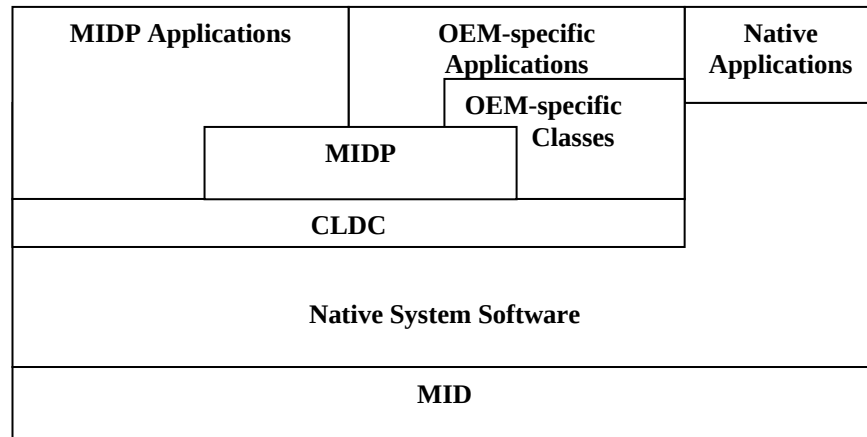
Tabel 2.1. memperlihatkan bahwa *Bluetooth protocol stack* terdiri dari protokol yang secara khusus ditujukan untuk teknologi *Bluetooth*, seperti L2CAP dan SDP, dan protokol adopsi seperti OBEX. Tabel 2.1. juga memperlihatkan bahwa *Bluetooth protocol stack* dapat dibagi menjadi 4 *layer* menurut tujuan masing-

masing. *Layer Baseband* bertugas mengadakan hubungan fisik RF antara unit *Bluetooth* dalam membuat sebuah koneksi. *Layer Link Manager Protocol* (LMP) bertugas pada menetapkan konfigurasi hubungan antar unit *Bluetooth* dan mengatur aspek keamanan, seperti enkripsi dan autentifikasi. *Layer L2CAP* bertugas mengadaptasi protokol *layer* yang berada diatas *layer L2CAP* pada *layer Baseband* yaitu dengan jalan mengadakan proses *multiplex* pada bermacam-macam logika koneksi yang dilakukan oleh *layer* yang berada diatas. *Layer Service Discovery Protocol* (SDP) bertugas sebagai tempat untuk meminta informasi unit, informasi *service* dan karakteristik dari *service*. *Layer RFCOMM* bertugas mengemulasikan kontrol dan sinyal data dari RS-232 pada *layer Baseband Bluetooth*, serta menyediakan kemampuan transmisi pada *service* tingkat atas yang menggunakan interaksi *serial* sebagai mekanisme transmisinya. *TCS Binary* bertugas mendefinisikan kontrol sinyal panggilan untuk pengadaan audio dan data antar unit *Bluetooth*.

2.2. MIDP (*Mobile Information Device Profile*)

MIDP adalah salah satu spesifikasi *profile* yang buat dengan berdasarkan CLDC (*Connected Limited Device Configuration*) yaitu konfigurasi yang dipakai untuk peralatan *wireless* berkapasitas memori kecil dengan koneksi jaringan yang tidak tetap/terputus-putus, seperti telepon selular, PDA dan lain-lain.

MIDlet adalah aplikasi yang dibuat dengan menggunakan J2ME dengan menggunakan *Profile* MIDP. Arsitektur tingkat tinggi dari sebuah aplikasi MIDP ditujukan oleh gambar 2.2. berikut.



Gambar 2.2. Arsitektur tingkat tinggi dari sebuah aplikasi MIDP

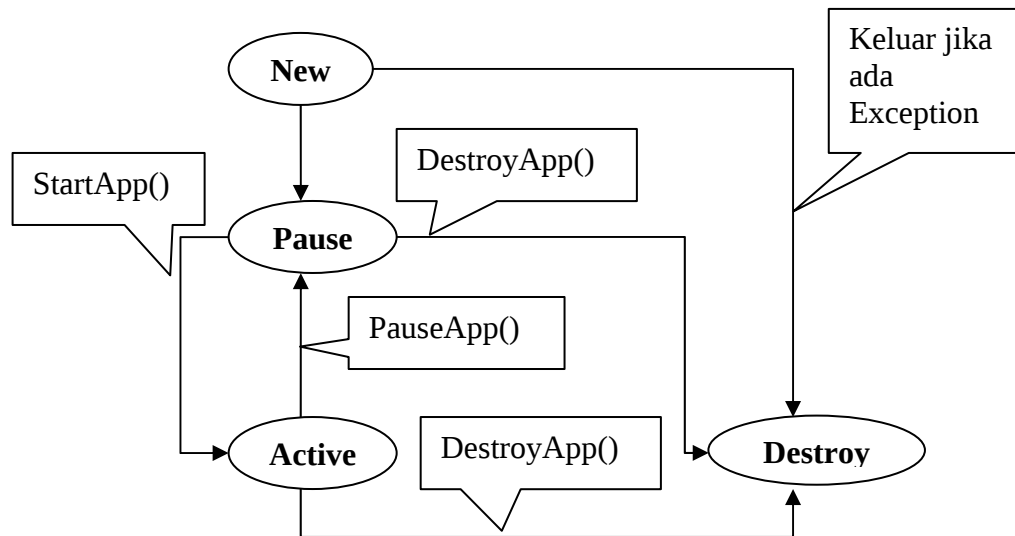
Sumber: Sun Microsystem. *Mobile Information Device Profile (JSR-37)*. Palo Alto: Sun Microsystem ,inc., 15 December 2000 : 26

Secara umum terdapat beberapa hal penting dalam membuat sebuah aplikasi *MIDlet*, yaitu menyangkut *lifecycle*, *user interface*, *command handling*, *deployment* dan *application management*.

2.2.1. Lifecycle

Lifecycle dari sebuah aplikasi *MIDlet* ditangani oleh *Application Management Software* (AMS). AMS adalah sebuah lingkungan tempat siklus dari sebuah aplikasi *MIDlet* dapat diciptakan, dijalankan, dihentikan maupun dihilangkan. AMS sering juga disebut sebagai *Java Application Manager* (JAM).

Lifecycle MIDlet mempunyai beberapa *state*, yaitu *Pause*, *Active* dan *Destroy*. Ketika masing-masing *state* dipanggil, beberapa *method* standar bawaan dari J2ME yang sesuai juga dipanggil. Gambar 2.3. merupakan gambar *lifecycle* dari sebuah MIDP.



Gambar 2.3. *Lifecycle* dari sebuah MIDP

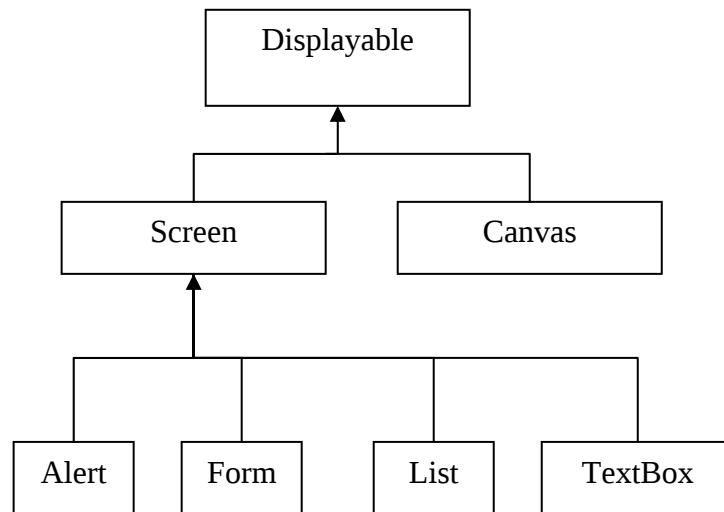
Sumber: Sun Microsystem. *Mobile Information Device Profile (JSR-37)*. Palo Alto: Sun Microsystem ,inc., 15 December 2000 : 121 (telah diolah kembali)

Dari gambar 2.3. dapat dijelaskan sebagai berikut:

1. Ketika *MIDlet* pertama kali diciptakan dan diinisialisasi, *MIDlet* akan berada pada state “*pause*”.
2. Apabila terjadi kesalahan selama konstruksi *MIDlet*, maka state akan berpindah ke state “*destroy*”, dan *MIDlet* akan batal diciptakan dengan memanggil fungsi standar *destroyApp()*.
3. Jika *MIDlet* berhasil dijalankan, maka *MIDlet* akan berada pada state “*active*”, dalam hal ini fungsi standar yang dipanggil adalah *startApp()*.
4. Tetapi jika ditengah proses *MIDlet* dihentikan sementara, *MIDlet* akan berada pada state “*pause*” dengan memanggil fungsi standar *pauseApp()*. Pada state inilah diperlukan proses pembersihan terhadap *garbage collector* yang dihasilkan.

2.2.2. User Interface

User interface dari MIDP terdiri dari API-API yang *High Level* dan *Low Level*. *Low Level* API berbasis pada class *Canvas* sedangkan *High Level* API berbasis pada *Screen*, contoh dari *High Level* API adalah *Alert*, *Form*, *List*, *TextBox*. Gambar 2.4. *Class diagram* dari *User Interface*.

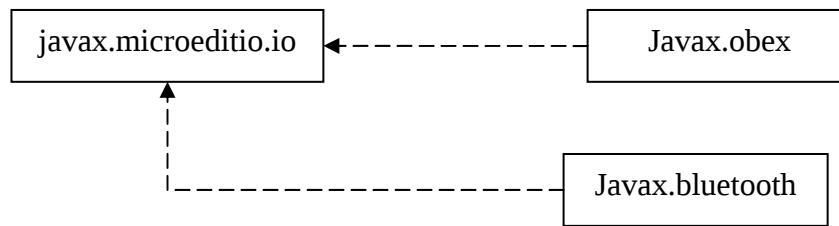


Gambar 2.4. Class diagram dari tipe screen pada MIDP

Sumber : Hartanto, Antonius Aditya. *Java 2 Micro Edition : Mobile Interface Device Programing*. Jakarta: P.T.Elex Media Komputindo, 2003 :15

2.3.MIDP Dan Bluetooth API

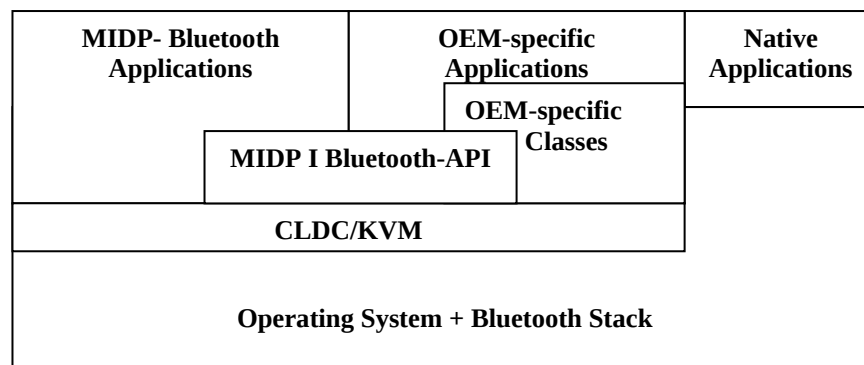
Java API untuk Bluetooth atau Java Specification Request 82 (JSR-82) adalah suatu API yang untuk mendefinisikan sebuah optional paket bagi Bluetooth untuk Java 2 Micro Edition (J2ME), dengan tujuan yaitu mendefinisikan arsitektur dan asosiasi kebutuhan API untuk membuka jalan bagi perkembangan aplikasi Bluetooth. MIDP adalah peralatan yang menjadi target utama dari spesifikasi dari Java API untuk Bluetooth. Java API untuk Bluetooth didisain untuk dapat beroperasi pada *Connected Limited Device Configuration* (CLDC) dan MIDP sehingga pada peralatan tersebut dapat dikembangkan lagi suatu aplikasi yang menggunakan Bluetooth. JSR-82 mendefinisikan *Package* yang ada menjadi 2, yaitu *javax.bluetooth* dan *javax.obex*. setiap *package* tersebut adalah 2 buah *package* yang terpisah dalam implementasinya dengan CLDC, sehingga dapat digunakan salah satu atau keduanya secara bersamaan. *Package javax.bluetooth* berisi inti Bluetooth API, sedangkan *package javax.obex* berisi *Object Exchange protocol* (OBEX) API. Gambar 2.5. diperlihatkan struktur dari 2 *package* tersebut dengan *package javax.microedition.io*



Gambar 2.5. Struktur dari *package Bluetooth* dengan *package javax.microedition.io*

Sumber: Sun Microsystem. *Java APIs for Bluetooth Wireless Technology (JSR-82), Specification version 1.0a*. Austin: Motorola Wireless Software, Applications & Services, 5 April 2002 : 11

Hubungan antara MIDP dengan *Bluetooth* API adalah hubungan yang tidak tergantung antar satu dengan lainnya. Maksudnya adalah MIDP dapat beroperasi bersama-sama dengan *Bluetooth* API dan tidak saling mengganggu jika salah satu tidak ada. Gambar 2.6. adalah contoh arsitektur dari suatu aplikasi yang terdapat MID, CLDC dan *Bluetooth* beroperasi bersama-sama.



Gambar 2.6. Diagram arsitektur dari suatu aplikasi yang terdapat MIDP, CLDC dan *Bluetooth* beroperasi bersama-sama.

Sumber : Sun Microsystem. *Java APIs for Bluetooth Wireless Technology (JSR-82), Specification version 1.0a*. Austin: Motorola Wireless Software, Applications & Services, 5 April 2002 : 12

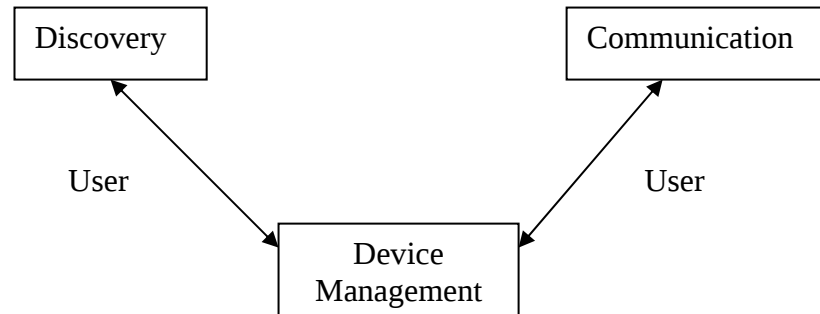
2.4. Arsitektur JSR-82

Berdasarkan pada cakupan dari spesifikasi JSR-82, pengalaman fungsi dapat dibagi menjadi 3 kategori besar, yaitu:

1. *Discovery*.
2. *Device Management*.

3. Communication.

Hubungan antar fungsi tersebut pada aplikasi diperlihatkan pada gambar 2.7.



Gambar 2.7. Hubungan antar fungsi yang ada pada JSR-82

Sumber : Sun Microsystem. *Java APIs for Bluetooth Wireless Technology (JSR-82), Specification version 1.0a*. Austin: Motorola Wireless Software, Applications & Services, 5 April 2002 :10

2.4.1. Discovery

Discovery adalah suatu cara dari sebuah peralatan *mobile* dalam mencari peralatan yang lain dan membuat koneksi, serta mempelajari tentang kemampuan dari peralatan yang lain. *Discovery* terdiri dari *device discovery*, *service discovery* dan *service registration*.

2.4.1.1 Device Discovery

Device Discovery adalah upaya dari unit *Bluetooth* untuk mencari unit *Bluetooth* lain. Proses yang terjadi pada saat *Device Discovery* berlangsung disebut Pencarian atau *inquiry*. *Device Discovery* biasanya digunakan pada tahap awal aplikasi yang berstatus *client* dijalankan, yaitu mencari unit aplikasi yang berjalan sebagai *server*.

Beberapa *class* yang berhubungan dengan *Device discovery* yaitu:

1. *Interface javax.bluetooth.DiscoveryListener*

Interface ini memungkinkan bagi sebuah aplikasi untuk memberi respon pada suatu permintaan koneksi. *Interface* ini juga digunakan untuk mencari layanan. *Method deviceDiscovery()* dipanggil setiap kali sebuah unit ditemukan disaat pencarian unit. Jika masa pencarian selesai atau dibatalkan, *method inquiryCompleted()* dipanggil.

2. Class *javax.bluetooth.DiscoveryAgent*

Class ini menyediakan *method* untuk *service* dan *device discovery*. Untuk *device discovery*, *class* ini menyediakan *method* *startInquiry* () untuk menempatkan unit lokal dalam status pencarian dan *method* *retrieveDevice*() untuk melihat kembali informasi tentang unit yang telah ditemukan pada masa pencarian yang sebelumnya.

2.4.1.2. Service Discovery

Service discovery menyediakan *method* untuk mencari sebuah *service* pada sebuah unit *server Bluetooth* yang telah terkoneksi.

beberapa *class* yang berhubungan dengan *Service discovery* yaitu:

1. Class *javax.bluetooth.UUID*

Class UUID mempunyai bentuk *unsigned integer* dengan besarnya 16 bit, 32 bit, atau 128 bit. *Class* ini digunakan untuk mewakili sebuah identitas universal yang unik yang digunakan secara luas sebagai suatu nilai dari sebuah atribut *service*. *Service* yang diwakili oleh *UUID* yang hanya akan dapat ditemukan oleh pencarian *Service Discovery Protocol* (SDP) *Bluetooth*.

2. Class *javax.bluetooth.DataElement*

Class ini berisi bermacam-macam tipe data yang dapat dipakai oleh sebuah *service Bluetooth*. *Class* ini juga memunculkan sebuah interface untuk membangun dan mengambil suatu nilai dari sebuah atribut *service*.

3. Interface *javax.bluetooth.ServiceRecord*

Interface ini mendefinisikan *Service Record Bluetooth* yang berisi atribut ID. Sebuah atribut *Bluetooth ID* adalah sebuah 16 bit *unsigned integer* dan nilai dari atribut tersebut adalah sebuah *DataElement*.

4. Class *javax.bluetooth.DiscoveryAgent*

Class ini menyediakan suatu *method* untuk menemukan *service* dan unit. *Class* ini membantu *service discovery* dalam keadaan *non-blocking* dan menyediakan cara untuk membatalkan sebuah transaksi pencarian *service* yang dalam proses.

5. *Interface javax.bluetooth.DiscoveryListener*

interface ini memungkinkan sebuah aplikasi untuk menspesifikasikan sebuah *event listener* yang diberi respon dari sebuah unit dan *service discovery event*.

2.4.1.3. *Service Registration*

Service registration adalah tanggung jawab dari sebuah *server Bluetooth* dalam menjalankan tugasnya yang meliputi : menciptakan sebuah *service record* yang mendeskripsikan *service* yang ditawarkan oleh sebuah aplikasi, menerima koneksi dari *client* yang meminta suatu *service* yang ditawarkan oleh sebuah aplikasi, dan lain-lainnya. *Class* yang berhubungan dengan *service registration* yaitu:

1. *Interface javax.bluetooth.ServiceRecord*

Service record mendeskripsikan sebuah *service Bluetooth* kepada *client*. *Service record* terdiri dari sebuah set atribut *service* dengan tiap atribut adalah sebuah pasangan yang tetap antara sebuah atribut ID dan sebuah atribut nilai. *Service record* menyediakan informasi yang cukup untuk sebuah *client Service Discovery Protocol (SDP)* mengadakan koneksi dengan *service Bluetooth* pada unit *server*

2. *Class javax.bluetooth.LocalDevice*

class ini menyediakan sebuah *method getRecord ()*, sehingga sebuah aplikasi *server* dapat digunakan untuk mengambil *ServiceRecord* miliknya. *Server* dapat memodifikasi *ServiceRecord* dari objek dengan menambahkan atau mengubah atribut. *Service Record* terbaru ditempatkan pada *Service Discovery DataBase (SDDB)* dengan menjalankan *notifier.acceptAndOpen ()* atau menggunakan *updateRecord () method* dari *LocalDevice*.

3. *Class javax.bluetooth.ServiceRegistrationException extends*

java.io.IOException

ServiceRegistrationException dikeluarkan apabila sebuah usaha untuk mengubah atau menambah sebuah *service record* dalam SDDB gagal.

2.4.2 Device Management

Pada *device management* terdapat cara-cara untuk mengatur sebuah unit. Pengaturan tersebut meliputi cara mengatur respon dari unit lokal terhadap *client*, mengatur agar unit dapat ditemukan oleh unit lain, mengatur masalah keamanan dan lain-lain. *Device management* dapat dibagi menjadi *Generic Access Profile* (GAP) dan *Security*.

2.4.2.1. Generic Access Profile (GAP)

GAP memiliki *class* yang sangat penting dalam objek *Bluetooth*, seperti *LocalDevice* dan *RemoteDevice*. *Class* yang berhubungan dengan GAP adalah

1. *Class javax.bluetooth.LocalDevice*

Class ini menyediakan akses dan kontrol pada unit *Bluetooth* lokal. *Class* ini didisain untuk memenuhi kebutuhan dari GAP sebagai bagian dari spesifikasi *Bluetooth*.

2. *Class javax.bluetooth.RemoteDevice*

Class ini menyediakan informasi dasar tentang suatu unit *mobile*, termasuk juga alamat *Bluetooth* unit dan nama yang digunakan oleh *Bluetooth* unit. *RemoteDevice* juga berisi *method* yang dapat digunakan setiap saat untuk minta perubahan dalam masalah keamanan.

3. *Class javax.bluetooth.BluetoothStateException extends java.io.IOException*

Exception ini digunakan jika sebuah unit tidak dapat melakukan respon pada suatu permintaan yang seharusnya diterima. Sebagai contoh adalah sebuah unit tidak dapat melakukan permintaan respon ketika unit tersebut terkoneksi pada unit lain.

4. *Class javax.bluetooth.DeviceClass*

Class yang mendefinisikan nilai dari tipe unit dan tipe *service* dari suatu unit.

2.4.2.2. Security

ada 3 sistem keamanan yang dipakai dalam manajemen *Bluetooth* adalah *authentication*, *encryption* dan *authorization*.

1. *Authentication*

Bluetooth authentication adalah suatu upaya untuk memastikan identitas dari sebuah unit yang mengadakan hubungan. Proses *authentication* ini melibatkan interaksi antar unit dan respon skematik yang membutuhkan sebuah 128 *bit shared link key* yang berupa kode PIN. Jika kode PIN tidak sama, maka *authentication* gagal.

2. *Encryption*

Encryption dapat dilakukan pada komunikasi sambungan data antar 2 unit *Bluetooth*. Jika diaktifkan, maka akan proses *Encryption* akan dilakukan pada semua transmisi data dalam koneksi tersebut. Oleh karena *Encryption* membutuhkan *shared link key*, maka *Encryption* membutuhkan *authentication*.

3. *Authorization*

Bluetooth Authorization adalah prosedur dimana sebuah pengguna dari sebuah unit *server* diberi akses pada sebuah *service* tertentu pada suatu unit *client* tertentu. Sama seperti *Encryption*, *Authorization* juga membutuhkan *Authentication* dari identitas *client*.

2.4.3 *Communication*

Unit *Bluetooth* lokal dalam melakukan komunikasi, harus menggunakan protokol yang sama. *Java API* untuk *Bluetooth* menyediakan protokol koneksi tingkat tinggi pada *service* pada *layer* L2CAP RFCOMM dan OBEX pada *Bluetooth*.

2.4.3.1. *Logical Link Control and Adaptation Protocol (L2CAP)*

Protokol L2CAP dapat digunakan untuk komunikasi yang berorientasi pada koneksi (komunikasi 2 arah) dan komunikasi yang tidak berorientasi pada koneksi (komunikasi banyak arah). Komunikasi yang berorientasi pada koneksi dibuat dengan menggunakan *connect service primitive* yang disediakan pada *layer* L2CAP. komunikasi yang tidak berorientasi pada koneksi digunakan untuk komunikasi group. Pada *Java API* untuk *Bluetooth*

(JSR-82), untuk saat ini protokol L2CAP belum dapat digunakan pada komunikasi grup atau komunikasi yang tidak berorientasi pada koneksi.

komunikasi yang berorientasi pada koneksi perlu dikonfigurasi setelah komunikasi diadakan. Parameter yang menjadi konfigurasi adalah :

1. *Maximum Transmission Unit (MTU)*,

MTU adalah besarnya data yang dapat dikirim berdasarkan kemampuan untuk menerima. Besar MTU ditentukan oleh aplikasi yang dijalankan, jika tidak maka nilai MTU akan diisi 672 *byte*, inilah yang disebut *incoming* MTU. Aplikasi dapat mengkonfigurasi besar MTU yang hendak diterimanya dari unit yang terkoneksi, inilah yang disebut *outgoing* MTU. Jika aplikasi tidak mengkonfigurasi nilai *outgoing* MTU, maka nilai tersebut akan sama atau kurang dari dengan *incoming* MTU. Dari segi jalannya data, MTU dapat dibagi menjadi 2, yaitu TransmitMTU dan Receive MTU. TransmitMTU adalah besar maksimum data dalam *bytes* yang dapat dikirim pada unit lain dalam bentuk *payload*. ReceiveMTU adalah besar maksimum data dalam *bytes* yang dapat diterima dari unit lain dalam bentuk *payload*.

2. *Flush Timeout*

Flush Timeout yaitu besarnya waktu untuk *link controller* terus mencoba untuk mengirim data sebelum menghapus data tersebut atau di ketahui statusnya. Nilai *default* dari *flush timeout* adalah ditentukan oleh *stack* yang digunakan untuk koneksi.

3. *Quality of Service (QoS)*

QoS yaitu untuk mendeskripsikan kepadatan aliran. Parameter dari QoS ditentukan oleh *stack Bluetooth*. Untuk saat ini, parameter QoS belum ditunjang oleh Java API untuk Bluetooth (JSR-82), yang mendefinisikan nilai parameter dari QoS adalah *stack* dari *Bluetooth*.

Beberapa *Class* yang berhubungan dengan L2CAP, yaitu:

1. *Interface javax.bluetooth.L2CAPConnection extends*

javax.microedition.io.Connection

Interface ini mewakili koneksi L2CAP. *Interface* ini mengandung *method* untuk mengambil MTU yang digunakan oleh sebuah koneksi dan untuk mengirim serta menerima data.

2. *Interface javax.bluetooth.L2CAPConnectionNotifier extends*

javax.microedition.io.Connection

Method yang terdapat dalam *interface* ini adalah *acceptAndOpen ()*.

Method ini digunakan *server* L2CAP untuk mengetahui jika ada koneksi masuk dari *client*.

3. *Class javax.bluetooth.BluetoothConnectionException extends*

java.io.IOException

Exception ini dikeluarkan jika *Bluetooth* tidak dapat sukses melakukan koneksi.

2.4.3.2. RFCOMM

Protokol RFCOMM menyediakan emulasi *serial port* RS-232 pada antar 2 buah unit *Bluetooth*. Protokol RFCOMM mempunyai profil yang digunakan untuk mengembangkan aplikasi yang menggunakan protokol RFCOMM, profil tersebut adalah *Serial Port Profile* (SPP). Protokol RFCOMM mempunyai bentuk koneksi yaitu koneksi aliran (*Stream Connection*). Pada koneksi RFCOMM, hanya boleh ada 1 sesi RFCOMM yang dapat aktif antara sepasang unit dalam waktu bersamaan, tetapi setiap sesi dapat terdiri dari banyak koneksi. Banyak koneksi yang mampu dibuat bersamaan adalah tergantung pada implementasi. Sebuah unit dapat memiliki lebih dari satu buah sesi RFCOMM selama tiap sesinya tersambung pada sebuah unit yang berbeda.

Sebuah *server* SPP harus menginisialisasi *service* yang ditawarkan dan mendaftarkan *service* tersebut pada SDDb. Sepasang objek yang dapat mewakili dari sebuah *service serial port* adalah *interface javax.microedition.io.StreamConnectionNotifier* untuk mendengarkan jika ada *client* yang meminta koneksi, dan *interface javax.bluetooth.ServiceRecord* untuk

mengambarkan *service* tersebut dan cara melakukan koneksi dengan *service*. Unit *client* dapat melihat *DeviceClass* dari sebuah *server* untuk mendapatkan informasi tentang jenis dari *server* dan *service* yang ditawarkan.

2.4.3.3. OBEX

Layer protokol OBEX adalah *layer* adopsi pada API *Bluetooth*. OBEX adalah Protokol yang dikembangkan oleh *Infrared Data Association* untuk melakukan “*pushing*” atau “*pulling*” objek dari dan kepada *client* dan *server*. Saat ini pada teknologi *Java* yang terdapat pada telepon selular masih tidak dapat menunjang API untuk protokol OBEX, sehingga protokol OBEX tidak dapat digunakan pada aplikasi *Java* pada telepon selular. OBEX melakukan transfer objek dengan melakukan atau mengadakan sebuah sesi OBEX. Sebuah sesi OBEX dimulai dengan sebuah permintaan koneksi dan diakhiri dengan pemutusan koneksi. Antara permintaan koneksi dan pemutusan koneksi, *client* dapat mengambil objek dari *server* atau menaruh objek pada *server*. Objek tersebut dapat berupa *file*, *vCards*, *bytes array* dan sebagainya. Seperti HTTP, OBEX menyediakan informasi tambahan antara *client* dan *server* dengan menggunakan *header* dari paket data.