# BAB II

# LANDASAN TEORI

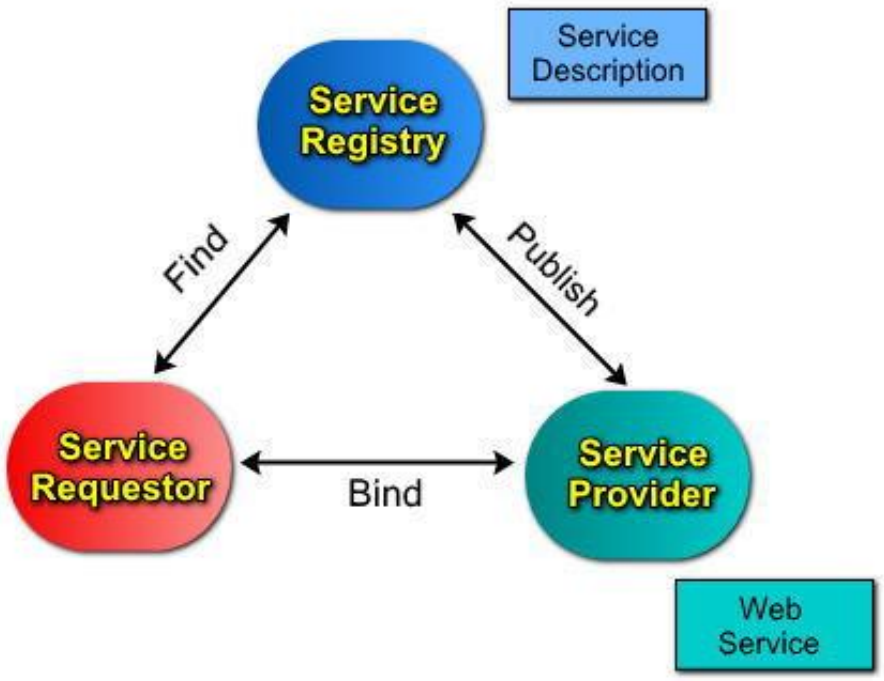
## 2.1 *Web service*

*Web service* adalah salah satu bentuk sistem perangkat lunak yang didesain untuk mendukung interaksi mesin-ke-mesin melalui jaringan.*Web service*memiliki *interface* yang dideskripsikan dalam format yang dapat dibaca oleh mesin. Sistem-sistem lainnya berinteraksi dengan *web service*menggunakan pesan SOAP yang umumnya dikirim melalui HTTP dalam bentuk XML.[5]

Definisi diatas diberikan oleh *World Wide Web Consortium*(W3C) yang merupakan badan yang menciptakan dan mengembangkan standar *web service*. Tetapi secara umum, *web service* tidak terbatas hanya pada standar SOAP saja. Salah satu pustaka yang mengulas lengkap tentang *web service*menyebutkan definisi yang lebih umum: *web service*adalah aplikasi yang diakses melalui *internet* menggunakan protokol standar *internet* dan menggunakan XML sebagai format pesannya.

### 2.1.1 Arsitektur *Web service*

Secara umum, arsitektur *web service* dapat dilihat pada gambar 2.1




*Gambar II.1 Arsitektur web service*

Pada gambar diatas, ada tiga komponen yang membuat *web service*berjalan. Ketiga komponen itu adalah:

1. **Service provide**r, merupakan pemilik *Web service* yang berfungsi menyediakan kumpulan operasi dari *Web service*.
2. **Service requesto**r, merupakan aplikasi yang bertindak sebagai klien dari *Web service* yang mencari dan memulai interaksi terhadap layanan yang disediakan.
3. **Service registry**, merupakan tempat dimana Service provider mempublikasikan layanannya. Pada arsitektur *Web service*, Service

registry bersifat optional. Teknologi *web service* memungkinkan kita dapat menghubungkan berbagai jenis software yang memiliki platform dan sistem operasi yang berbeda.

### 2.1.2 Jenis-Jenis *Web service*

Jenis *web service* dapat dibagi menjadi dua, yaitu REST dan SOAP.

#### 2.1.2.1 *Representational State Transfer* (REST)

REST adalah salah satu jenis *web service* yang menerapkan konsep perpindahan antar *state*. *State* disini dapat digambarkan seperti jika *browser* meminta suatu halaman *web*, maka *server*akan mengirimkan *state* halaman *web* yang sekarang ke *browser*. Bernavigasi melalui *link-link* yang disediakan sama halnya dengan mengganti *state* dari halaman *web*. Begitu pula REST bekerja, dengan bernavigasi melalui *link-link* HTTP untuk melakukan aktivitas tertentu, seakan-akan terjadi perpindahan *state* satu sama lain. Perintah HTTP yang bisa digunakan adalah fungsi GET, POST, PUT atau DELETE. Balasan yang dikirimkan adalah dalam bentuk XML sederhana tanpa ada protokol pemaketan data, sehingga informasi yang diterima lebih mudah dibaca dan diparsing disisi *client*.

Dalam pengaplikasiannya, REST lebih banyak digunakan untuk *web service*yang berorientasi pada *resource*. Maksud orientasi pada *resource* adalah orientasi yang menyediakan *resource-resource* sebagai layanannya dan bukan kumpulan-kumpulan dari aktifitas yang mengolah *resource* itu.Alasan mengapa REST tidak digunakan dalam skripsi ini karena orientasi pada *resource*nya itu,

sedangkan aplikasi *event calendar* membutuhkan pemanggilan metode yang bisa dikerjakan terhadap kumpulan *resource event*. Selain itu, karena standarnya yang kurang sehingga tidak begitu cocok diterapkan dalam aplikasi yang membutuhkan kerjasama antar aplikasi lain, dimana standar yang baik akan sangat berguna karena berbicara dalam satu bahasa yang sama. Beberapa contoh *web service* yang menggunakan REST adalah: Flickr API(*Application ProgramInterface*), YouTube API, Amazon API.

#### 2.1.2.2 *Simple Object Access Protokol* (SOAP)

SOAP adalah protokol untuk saling bertukar pesan dalam format XML antar komputer di dalam jaringan, biasanya menggunakan HTTP/HTTPS [6]. *Web service*dalam skripsi ini menggunakan SOAP sebagai protokol pengiriman pesannya. Dalam menjalankan tugasnya, SOAP menggunakan struktur XML tertentu dalam pengirimkan *request* kepada *web service*. Penerimaan respon dari *web service*pun memiliki strukturnya sendiri. Agar lebih mudah dalam memahami SOAP, di bawah ini diberikan salah satu contoh struktur dari SOAP *request*:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
        xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:ns1="urn:ExampleAPI">
<SOAP-ENV:Body>
        <ns1:getPeopleByFirstLastName>
                <first>j*</first>
                <last>*</last>
        </ns1:getPeopleByFirstLastName>
</SOAP-ENV:Body></SOAP-ENV:Envelope>
```

Struktur ini terdiri dari dua bagian, *Envelope* dan *Body*. *Envelope* berisi keterangan-keterangan tentang *namespace* yang digunakan dalam dokumen XML ini, dan juga dapat berisi atribut dan *header*.Sedangkan *Body* berisi pesan yang akan dikirimkan ke *web service*. Dalam contoh diatas, *Body* berisi permintaan akan operasi getPeopleByFirstLastName yang disediakan suatu *web service*dengan parameter yang dikirimkan berupa nama depan dengan huruf pertama j dan nama belakang apa saja. Sedangkan contoh SOAP *response* yang didapat dari *server*diberikan dalam contoh dibawah :

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
        xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
        <result>
                <SOAP-ENC:Struct>
                        <id>1</id>
                        <firstName>John</firstName>
                        <lastName>Smith</lastName>
                </SOAP-ENC:Struct>
                <SOAP-ENC:Struct>
                        <id>2</id>
                        <firstName>Jane</firstName>
                        <lastName>Doe</lastName>
                </SOAP-ENC:Struct>
        </result>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Struktur SOAP *response* di atas juga sama dengan struktur SOAP *request* yang diterangkan sebelumnya, tetapi bisa kita lihat disini, isi dari *body* memiliki *tag result* yang berisi balasan atau *output* dari operasi *web service* yang dipanggil pada pesan SOAP *request*. Beda halnya dengan REST, SOAP *web service* lebih menekankan pada *web service* yang berorientasi *service*. Ketimbang mengambil *resource-resource* seperti dalam REST, SOAP lebih ke arah pemanggilan metode-metode yang memanipulasi *resource-resource* yang ada, dalam arti lain, melayani kebutuhan pengguna yang lebih beragam. Beberapa contoh SOAP *web service*adalah Google API, Yahoo! API, dan aplikasi *event calendar.*

### 2.1.3 *Web service Definition Language* (WSDL)

SOAP dapat berjalan sendiri tanpa bantuan WSDL, tetapi hal ini akan menyulitkan pengembang di sisi *client* karena tidak tahu harus mengharapkan operasi dan bentuk data apa. Karena itu WSDL diciptakan sebagai pendefinisi sebuah *web service*.WSDL adalah tata bahasa XML yang digunakan untuk mendeskripsikan sebuah *web service*.WSDL adalah bahasa XML yang digunakan untuk menjabarkan bagaimana *web service*diakses, operasi-operasi *web service*, bagaimana pesan disampaikan, dan struktur dari pesan tersebut [5]. Dalam tugas akhir ini, *web service* yang dibangun akan dijabarkan menggunakan WSDL. Struktur dari WSDL adalah sebagai berikut:

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/">
<types>
</types>
<message>
</message>
<portType>
</portType>
<binding>
</binding>
<service>
</service>
</definitions>
```

Struktur ini terdiri atas lima bagian, yaitu:

1. *Type*s

   Bagian *types* menjabarkan definisi tipe data yang digunakan dalam pertukaran pesan, tipe data disini dapat berupa tipe data sederhana ataupun tipe data yang kompleks.

2. *Messa*ge

   Bagian *message* berisi abstraksi parameter *input* dan *output* yang diharapkan dari pertukaran pesan. Parameter *input* dan *output* ini menggunakan tipe data yang dijabarkan di bagian types.

3. *PortTy*pe

   Bagian *portType* berisi koleksi dari abstraksi operasi-operasi *web service*dan abstraksi pesan-pesan yang digunakan oleh tiap-tiap operasi.

4. *Binding*

   Bagian *binding* adalah bagian yang menjabarkan informasi mengenai protokol transpor yang digunakan dan format pesannya.

5. *Service*

   Yang terakhir adalah bagian service, bagian ini yang mendeskripsikan nama dan lokasi dari suatu *web service*.

## 2.2 Android

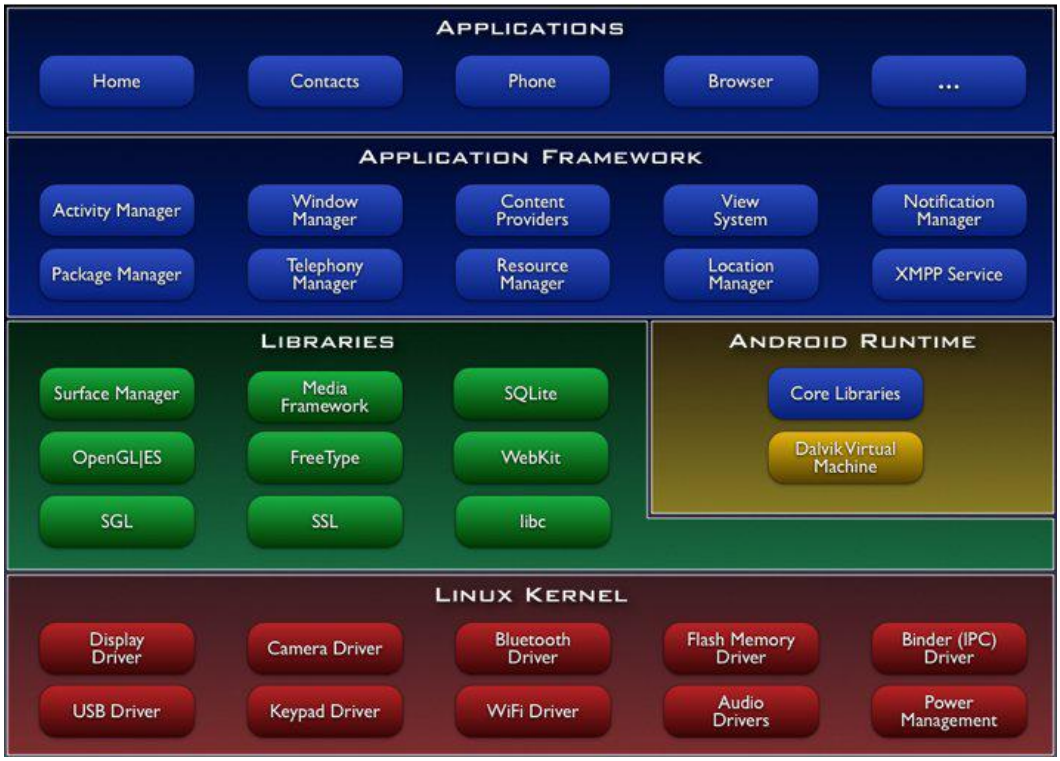
Android merupakan suatu *software stack* untuk *mobile device*. Di dalamnya terdapat sistem operasi, *middleware,* dan *key application*.Aplikasi pada platform ini dikembangkan dalam bahasa pemrograman java. Android sendiri memiliki banyak fitur diantaranya adalah:

- Merupakan sebuah **A*pplication Framework*** sehingga *programmer* dapat menggunakan beberapa fungsi yang telah disediakan.
- **Dalvik *virtual machine***. Tiap aplikasi dalam Android memiliki *instance virtual machine* yang dapat bekerja secara efisien dalam lingkungan memoriyang terbatas.
- ***Integrated browser*.** Web browser berbasis WebKit *engine* terdapat pada browser *default* Android atau pun dapat diintegrasikan dengan aplikasi lain.
- ***Optimized graphics***. *Library* grafis 2D yang kaya dan 3D berbasis OpenGL ES 1.0 yang mendukung akselerasi hardware.
- **SQLite**. Basis data relasional yang ringan namun sangat *powerful.*

- ***Media Support***. Mendukung berbagai format audio, video, dan gambar (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- **GSM *Telephony***. Mendukung fungsi komunikasi GSM.
- **Bluetooth,EDGE,3G** dan **WiFi**. Mendukung komunikasi pada jaringan (tergantung hardware).
- **Kamera**, **GPS**, **kompas** dan ***accelerometer***. Mendukung berbagai fitur yang disediakan oleh *hardware*.
- ***Tools Developer*** yang lengkap. Termasuk *device emulator, tools* untuk *debugging*, *profiling* memori dan performa, *plugin* untuk Eclipse IDE.

### 2.2.1 Arsitektur Android

Android terdiri dari beberapa *stack software* yang terdiri dari: *Applications, Application Framework, Libraries, Android Runtime* dan *Kernel Linux.* Arsitekturlengkap platform ini dapat dilihat pada Gambar 2.2.




*Gambar II.2 Arsitektur platform android*

### *2.2.1.1 Aplication*

*Application* merupakan program yang langsung berhubungan dengan *user*. Baik program yang merupakan bawaan dari Android sendiri maupun program yang dibuat oleh *developer* menggunakan bahasa pemrograman java. Contoh program bawaan dari platform Android sendiri adalah *email client*, program SMS, *calendar*, *maps, web browser*, *contact* dan sebagainya.

### *2.2.1.2 Aplication Framework*

Lapisan ini berisi sekumpulan API yang dapat digunakan oleh *programmer* maupun *core application* dari Android.Lapisan ini dirancang untuk memudahkan penggunaan komponen dari Android sendiri. Aplikasi manapun

dalam Android dapat berbagi fungsi sehingga aplikasi lain dapat memanfaatkannya.

Aplikasi pada Android disusun atas beberapa komponen:

- Sekumpulan *Views*. Digunakan untuk mengatur tampilan pada aplikasi. Contohnya adalah *lists, grids, text box, button*, bahkan *embeddable webbrowser*.
- *Content providers.* Komponen yang mengatur agar aplikasi dapat mengakses *resources* dari aplikasi lain (seperti *Contacts*), atau berbagi data dengan aplikasi lain.
- *Resource Manager*. Menyediakan akses ke pada *resource non-code* seperti *localized string*, grafik dan *file layout*.
- *Notification Manager*. Memungkinkan agar suatu aplikasi dapat menampilkanperingatan yang dapat di kostumasi pada *status bar.*
- *Activity Manager*. Mengatur siklus aplikasi dan navigasi antar aplikasi yang sedang berjalan.

#### *2.2.1.3 Libraries*

Android mendukung beberapa *library* C/C++ yang digunakan pada berbagai komponen Android. Kemampuan ini dapat diakses oleh *developer* melalui Android *application framework*. Beberapa *library* diantaranya adalah:

- *System* C *library*. Implementasi *library* C standar (libc).
- *Media Libraries*. Mendukung berbagai format multimedia (termasuk MPEG4,H.264, MP3, AAC, AMR, JPG, PNG).
- *Surface Manager*. Mengatur akses ke subsistem *display*.

- LibWebCore. *Engine web browser* modern.
- SGL. *Engine* grafis 2D.
- 3D *Library*. Implementasi OpenGL ES 1.0 yang mendukung akselerasi hardware.
- FreeType. *Rendering* untuk *bitmap* dan *vector font.*
- SQLite. Basis data relasional yang kecil namun sangat ampuh.

#### *2.2.1.4 Android Runtime*

Tiap aplikasi pada Android memiliki proses-nya masing-masing.Tiap aplikasi tersebut memiliki *instans* dari Dalvik *virtual machine* (VM). Dalvik *virtual machine* dirancang agar suatu *device*dapat menjalankan beberapa VM secara efisien. Dalvik VM *mengeksekusi* file dengan format Dalvik *Executable format* (.dex) yang dirancang untuk meminimalkan *memory footprint* . Dalvik VM berbasis *register*, dan dapat menjalankan kelas-kelas yang dikompilasi dengan bahasa pemrograman java dan ditransformasikan menjadi format .dex. Dalvik VM sendiri bergantung pada Kernel Linux untuk fungsi dasarnya, seperti *threading* dan manajemen memori secara *low-level*.

#### 2.2.1.5 Linux Kernel

Android menggunakan Kernel Linux versi 2.6 sebagai sistem utama. Fungsi kernel yang digunakan antara lain untuk keamanan, manajemen memori, manajemen proses, manajemen jaringan dan *driver model*. Kernel juga berfungsi sebagai *layer* abstrak antara *hardware* dan lapisan lainnya pada *software stack*.

### 2.2.2 Anatomi Aplikasi Android

Terdapat empat *building blocks* pada aplikasi Android yaitu:

- *Activity*
- *Intent Receiver*
- *Service*
- *Content Provider*

Tidak semua aplikasi membutuhkan keempat blok ini, tetapi suatu aplikasi dibuat menggunakan kombinasi beberapa blok ini. Setelah memutuskan blok komponen yang akan digunakan, blok tersebut didaftarkan pada suatu *file* yang disebut dengan AndroidManifest.xml. *File* XML ini digunakan untuk menyatakan komponen apa saja yang dibutuhkan oleh sebuah aplikasi dan kemampuan serta kebutuhan aplikasi tersebut.

#### *2.2.2.1 Activity*

*Activity* paling umum diantara keempat *building block*.*Activity* biasanya suatu tampilan (*screen*) dalam suatu aplikasi. Tiap *activity* diimplementasikan sebagai satu kelas yang meng-*extend* base class *Activity*. Kelas yang dibuat akan menampilkan *user interface* yang terdiri dari beberapa *views* dan akan merespon suatu *event*. Kebanyakan, aplikasi terdiri dari beberapa *screen*. Contohnya aplikasi pengiriman pesan akan memiliki *screen* untuk mendaftarkan pesan yang masuk, menulis pesan dan *screen* lainnya untuk melakukan pengaturan. Tiap *screen* pada contoh ini akan diimplementasi sebagai suatu *activity*. Aplikasi dapat berpindah ke *screen* lain dengan cara memulai suatu *activity* baru. Pada suatu kasus tertentu, *activity* akan mengembalikan nilai kepada *activity* sebelumnya. Contohnya

*activity* untuk memilih foto akan mengembalikan foto yang dipilih pada *activity* yang memanggilnya. Pada saat *screen* baru terbuka, screen sebelumnya akan dihentikan dan *state* nya akan disimpan pada *history stack*. *User* dapat melakukan navigasi ke *activity* sebelumnya melalui *history* tersebut.*Screen* akan di hapus dari *history stack* apabila sistem memutuskan bahwa *screen* tersebut tidak layak untuk berada pada memori. Android menyimpan *history stack* untuk setiap aplikasi yang dijalankan dari tampilan awal (*home screen*).

#### *2.2.2.2 Intent* dan *Intent Filter*

Android menggunakan kelas khusus bernama *intent* untuk berpindah dari *screen* ke *screen* lainnya. Sebuah *intent* mendeskripsikan apa yang ingin aplikasi penuhi. Bagian struktur data dari *intent* yang paling penting adalah *action* dan *data* dari aksi yang dilakukan.Nilai *action* yang biasanya adalah MAIN, VIEW, PICK, EDIT dan sebagainya.Data diekspresikan sebagai URI.Sebagai contoh, untuk melihat informasi suatu *contact* seseorang, kita membuat *intent* dengan action VIEW dan data sebagai URI yang merepresentasikan orang tersebut.

Terdapat suatu kelas yang dinamakan IntentFilter.Apabila *intent* adalah suatu permintaan untuk melakukan sesuatu, IntentFilter adalah deskripsi yang dapat dilakukan oleh *intent activity*.*Activity* yang dapat menampilkan informasi *contact* dari seseorang akan mempublikasikan IntentFilter yang menyatakan bahwa ia dapat menangani *action* VIEW apabila dipakai pada data yang merepresentasikan seseorang. Suatu aplikasi mempublikasikan IntentFilter pada *file* AndroidManifest.xml.

Navigasi dari suatu *screen* ke *screen* lainnya dipenuhi dengan cara me-*resolve intent*. Untuk melakukan navigasi ke depan, *activity* memanggil startActivity(myIntent). Sistem akan mencari IntentFilter untuk semua aplikasi dan memilih *activity* yang IntentFilter-nya yang paling mendekati dengan myIntent. *activity*baru akan mendapat informasi tentang *intent* yang menyebabkan *activity* tersebut dijalankan. Proses pencarian *intent* terjadi sewaktu runtime pada saat startActivity dipanggil. Hal tersebut membawa keuntungan sebagai berikut:

- *Activity* dapat melakukan *reuse functionality* dari komponen lain dengan cara melakukan permintaan dalam bentuk *intent*.
- *Activity* dapat di *replace*kapan pun dengan *activity* yang ekuivalen.

#### *2.2.2.3 Intent Recivier*

*IntentReceiver* dapat digunakan pada saat kita menginginkan kode kita dijalankan akibat reaksi dari *external event*tertentu.Contohnya, pada saat telepon berbunyi, atau pada saat ada data pada jaringan, atau pada suatu waktu tertentu. IntentReceiver tidak menampilkan *user interface*, meskipun ia dapat menggunakan NotificationManager untuk memperingatkan *user* apabila sesuatu terjadi. *IntentReceiver* didaftarkan pada *file* AndroidManifest.xml, tetapi secara manual juga bisa ditambahkan dengan fungsi Context.registerReceiver(). Suatu aplikasi tidak harus berjalan pada *IntentReceiver*nya agar bisa dipanggil, tetapi sistem yang akan menjalankan aplikasi tersebut. Apabila diperlukan, pada saat IntentReceiver mendapat *trigger*, aplikasi juga dapat mengirimkan *intent broadcast* kepada aplikasi lainnya dengan fungsi Context.broadcastIntent().

#### *2.2.2.4 Service*

*Service* adalah sebuah kode yang hidup terus-menerus dan berjalan tanpa memiliki tampilan. Contoh yang sederhana adalah suatu aplikasi *media player* yang sedang memainkan lagu dari *play list.* Pada aplikasi *media player*, kemungkinan terdapat satu atau lebih activity yang memperbolehkan *user* memilih lagu dan memainkannya. Tetapi musik yang sedang diputar tidak boleh ditangani oleh activity karena *user* akan menginginkan musik tetap diputar meskipun ia berpindah ke *screen* baru. Pada kasus ini activity *media player* dapat menjalankan *service* menggunakan fungsi Context.startService() untuk menjalankan *service* di *background* yang memutar musik. Sistem akan tetap memutar musik sampai selesai. Selain itu *activity* dapat terhubung dengan *service* menggunakan fungsi Context.bindService(). Pada saat terhubung, aplikasi dapat berkomunikasi melalui *interface* yang diberikan oleh *service* yang bersangkutan. Pada contoh *media player*, *interface service* akan memberikan fungsi *pause*, *rewind* dan sebagainya.

#### *2.2.2.5 Content Provider*

*Content provider* digunakan untuk pertukaran data antara aplikasi. Data yang dipertukarkan bersifat *shared*. *Content provider* adalah sebuah kelas yang berisi *method* standar yang memungkinkan aplikasi lain dapat menyimpan dan mengambil suatu data yang ditangani oleh *content provider*.

### 2.2.3 Siklus Hidup Aplikasi Android

Pada banyak kasus, tiap aplikasi pada Android masing-masing memiliki Linux proses. Proses ini diciptakan untuk aplikasi tersebut pada saat kode program tersebut akan dieksekusi, akan terus berjalan sampai tidak dibutuhkan lagi dan sistem memerlukan memori untuk aplikasi lain. Konsep dasar program Android adalah suatu proses pada aplikasi tidak langsung di atur oleh aplikasi tersebut. Proses tersebut ditentukan oleh sistem melalui kombinasi : informasi aplikasi bagi sistem, seberapa pentingnya aplikasi tersebut bagi *user* dan ketersediaan memori. Contoh kasus *life-cycle bug* adalah *IntentReceiver* yang menciptakan *thread* pada saat menerima *intent* pada *method* onReceiveIntent() dan selesai dari fungsi tersebut. Setelah fungsi tersebut selesai, sistem menganggap bahwa *IntentReceiver* tersebut menjadi tidak aktif sehingga ia memiliki proses yang tidak dibutuhkan (kecuali terdapat komponen aplikasi lain di dalamnya). Oleh karena itu, sistem dapat menghapus proses tersebut kapanpun apabila dibutuhkan. Hal tersebut akan menghapus semua *thread* yang berjalan pada proses tersebut. Solusi masalah ini adalah memakai *Service* dari IntentReceiver, sehingga sistem mengetahui bahwa ada suatu yang aktif dalam proses tersebut.

Sebagai *programmer*, kita harus mengetahui bagaimana cara yang tepat untuk merancang aplikasi. Kesalahan pada perancangan akan membuat suatu aplikasi akan dihentikan oleh sistem pada saat yang tidak diinginkan. Untuk menentukan proses mana yang akan dihapus pada situasi *low memory*, Android

memiliki "*importancehierarchy*" berdasarkan komponen dan *state* yang sedang berjalan. Urutan berdasarkan kepentingan tersebut adalah:

1. ***Foreground process*.** Adalah suatu proses yang dibutuhkan untuk berinteraksi dengan *user* saat ini. Berbagai komponen suatu aplikasi dapat mengakibatkan proses di dalamnya menjadi *foreground proses*. Suatu proses dikatakan *foreground* apabila memenuhi kondisi sebagai berikut:
    - Menjalankan Activity pada *top screen* yang sedang berinteraksi dengan *user* (setelah *method* onResume() dipanggil).
    - Mempunyai IntentReceiver yang sedang berjalan (method IntentReceiver.onReceiveIntent() sedang dieksekusi).
    - Memilik service yang fungsi *callback*-nya ( Service.onCreate(), Service.onStart() atau Service.onDestroy()) sedang dieksekusi.

    Hanya akan terdapat beberapa proses seperti itu dalam sistem, proses ini hanya akan dihapus apabila sistem tidak memiliki pilihan lain. Contohnya apabila memori tinggal sedikit sehingga proses *foreground* tidak bisa lagi berjalan. Halini diperlukan untuk menjaga *user interface* tetap interaktif dengan *user*.
2. ***Visible process*.** Adalah proses yang memegang Activity yang dapat dilihat*user* pada layar tetapi tidak *foreground* (pada saat *method* onPause()dipanggil). Hal ini dapat terjadi, sebagai contoh *activity foreground* muncul dengan dialog sehingga *activity* sebelumnya terlihat di belakangnya. Proses ini sangat penting dan tidak akan di hapus kecuali untuk menjaga aplikasi *foreground* tetap berjalan.

3. ***Service process*.** Adalah proses yang memegang Service yang dimulai dengan method StartService().Meskipun proses ini secara langsung tidak dapat dilihat oleh *user*, biasanya proses ini menjalankan perintah dari *user* (seperti memutar lagu, men-*download* data dari jaringan dan lain-lain). Sistem akan tetap membiarkan proses ini berjalan selama proses *foreground* dan *visible* masih bisa berjalan.
4. ***Background process***. Adalah proses yang memegang Activity yang saat ini tidak dapat dilihat oleh *user* (method onStop() sedang dipanggil). Proses ini tidak secara langsung mempengaruhi *user*. Apabila *activity life cycle* telah dipenuhi dengan benar, sistem dapat menghapus proses ini sewaktu-waktu untuk memberikan memori tambahan. Biasanya banyak ditemukan proses semacam ini, sehingga proses tersebut disimpan dalam LRU (*Least RecentlyUsed*) untuk menjaga agar proses yang baru saja dilihat oleh *user* akan dihapus paling akhir pada kondisi *low memory*.
5. ***Empty Process*.** Adalah proses yang tidak memegang suatu komponen yangsedang aktif. Alasan mengapa membiarkan proses semacam ini adalah untukmelakukan *cache* sehingga proses *startup* menjadi lebih cepat apabila aplikasitersebut akan dijalankan kembali. Sistem seringkali menghapus prosessemacam ini untuk mencapai keseimbangan dari *system resource* antaracached processes dan kernel caches.

## 2.3 UML

Unified Modeling Language (UML) adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sistem perangkat lunak. Unified Modeling Language (UML) adalah himpunan struktur dan teknik untuk pemodelan desain program berorientasi objek (OOP) serta aplikasinya.[1] UML adalah metodologi untuk mengembangkan sistem OOP dan sekelompok perangkat tool untuk mendukung pengembangan sistem tersebut.[1] UML mulai diperkenalkan oleh Object Management Group, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP sejak tahun 1980-an.[1] Sekarang UML sudah mulai banyak digunakan oleh para praktisi OOP.[1] UML merupakan dasar bagi perangkat (tool) desain berorientasi objek dari IBM.[1]

UML adalah suatu bahasa yang digunakan untuk menentukan, memvisualisasikan, membangun, dan mendokumentasikan suatu sistem informasi.UML dikembangkan sebagai suatu alat untuk analisis dan desain berorientasi objek oleh Grady Booch, Jim Rumbaugh, dan Ivar Jacobson.Namun demikian UML dapat digunakan untuk memahami dan mendokumentasikan setiap sistem informasi.Penggunaan UML dalam industri terus meningkat.Ini merupakan standar terbuka yang menjadikannya sebagai bahasa pemodelan yang umum dalam industri peranti lunak dan pengembangan sistem.

### 2.3.1 Sejarah UML

Sampai era tahun 1990 puluhan metodologi pemodelan berorientasi objek telah bermunculan di dunia. Diantaranya adalah: metodologi booch, metodologi coad, metodologi OOSE, metodologi OMT, metodologi shlaer-mellor, metodologi

wirfs-brock, dsb. Masa itu terkenal dengan masa perang metodologi (method war) dalam pendesainan berorientasi objek. Masing-masing metodologi membawa notasi sendiri-sendiri, yang mengakibatkan timbul masalah baru apabila kita bekerjasama dengan kelompok/perusahaan lain yang menggunakan metodologi yang berlainan.

Dimulai pada bulan Oktober 1994 Booch, Rumbaugh dan Jacobson, yang merupakan tiga tokoh yang boleh dikata metodologinya banyak digunakan mempelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek.Pada tahun 1995 direlease draft pertama dari UML (versi 0.8) [1].Sejak tahun 1996 pengembangan tersebut dikoordinasikan oleh Object Management Group (OMG – http://www.omg.org).

### 2.3.2 Diagram UML

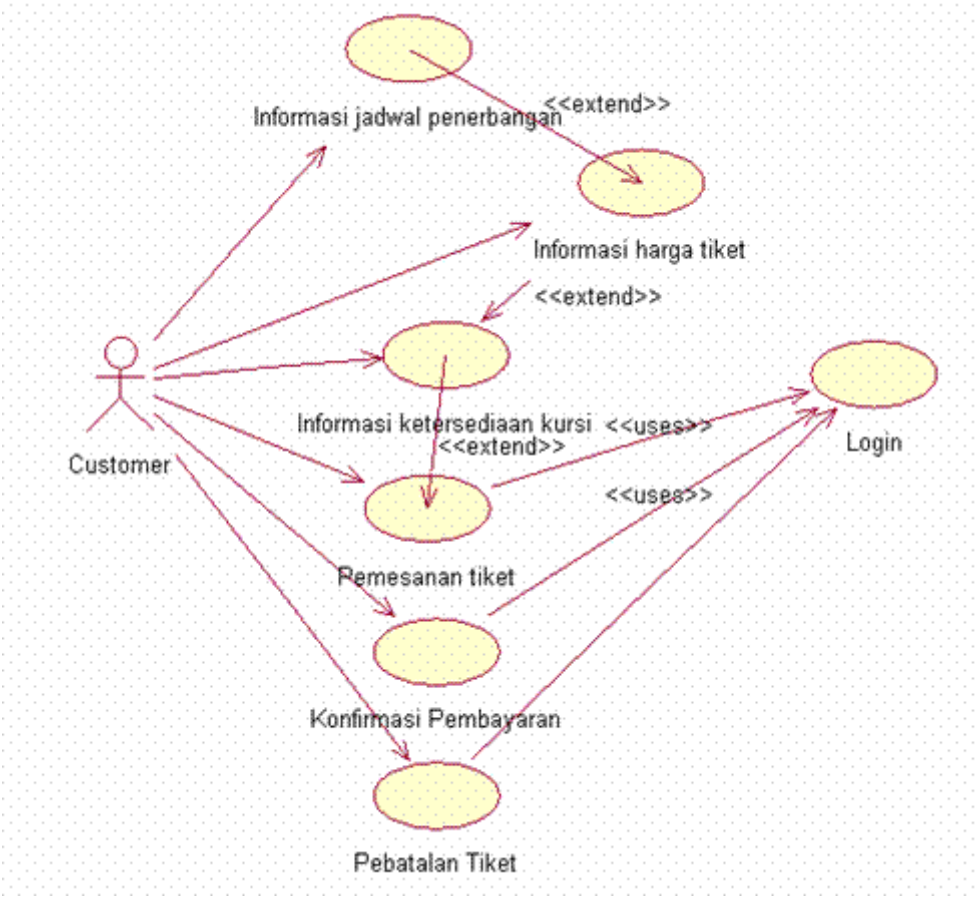
UML menyediakan 10 macam diagram untuk memodelkan aplikasi berorientasi objek, yaitu:

1. Use Case Diagram untuk memodelkan proses bisnis.
2. Conceptual Diagram untuk memodelkan konsep-konsep yang ada di dalam aplikasi.
3. Sequence Diagram untuk memodelkan pengiriman pesan (message) antar objects.
4. Collaboration Diagram untuk memodelkan interaksi antar objects.
5. State Diagram untuk memodelkan perilaku objects di dalam sistem.
6. Activity Diagram untuk memodelkan perilaku Use Cases dan objects di dalam system.

7. Class Diagram untuk memodelkan struktur kelas.
8. Object Diagram untuk memodelkan struktur object.
9. Component Diagram untuk memodelkan komponen object.
10. Deployment Diagram untuk memodelkan distribusi aplikasi.

#### 2.3.2.1 Use Case Diagram

*Use case diagram* menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal. Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*. Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain. Penggambaran Use case diagram dapat di lihat dalam gambar 2.3 di bawah ini.



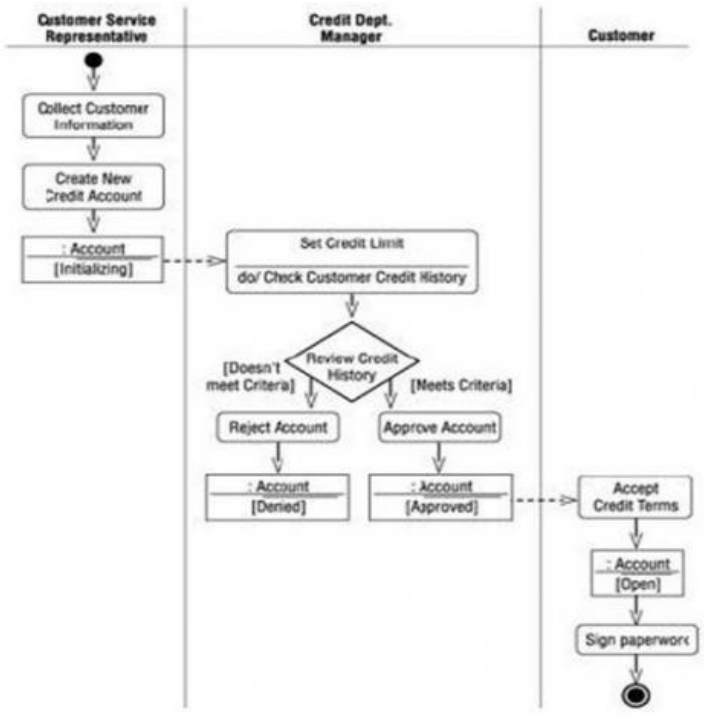

*Gambar II.3 Contoh use case diagram*

### 2.3.2.2 Activity Diagram

*Activity diagrams* menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi. *Activity diagram* merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan behaviour internal sebuah sistem (dan interaksi antar subsistem)

secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

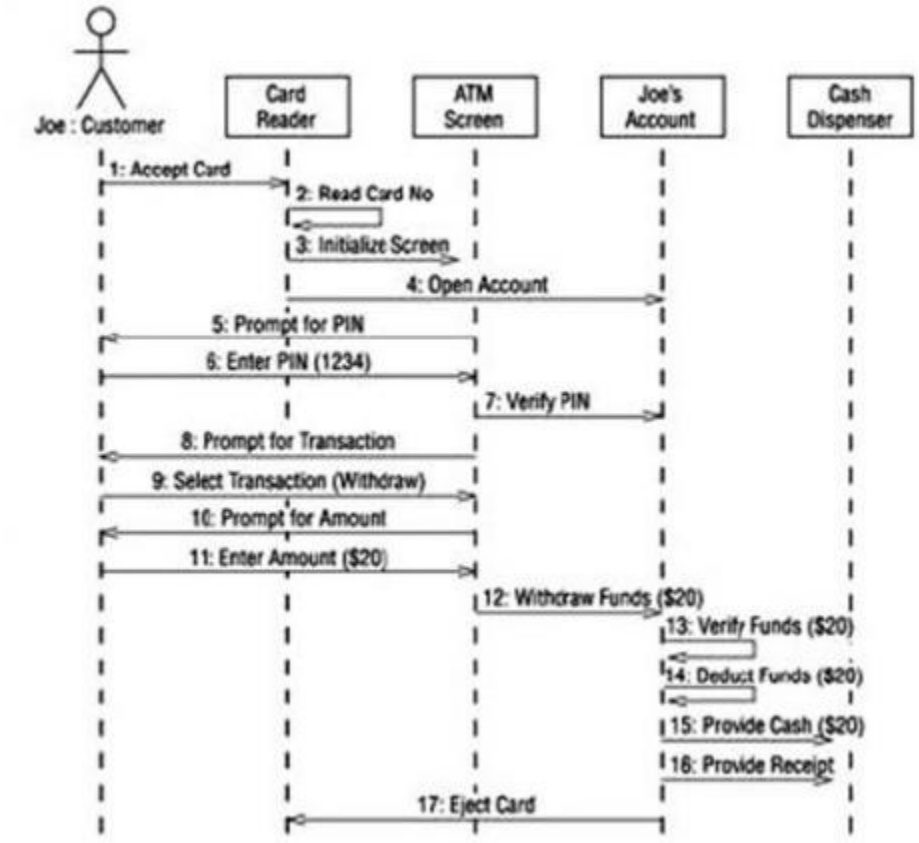
Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas. *Decision* digunakan untuk menggambarkan behaviour pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal. *Activity diagram* dapat dibagi menjadi beberapa *object swimlane* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu. Contoh Activity diagram dapat di lihat pada gambar 2.4 berikut ini.




*Gambar II.4 Contoh activity diagram*

### 2.3.2.3 Sequence Diagram

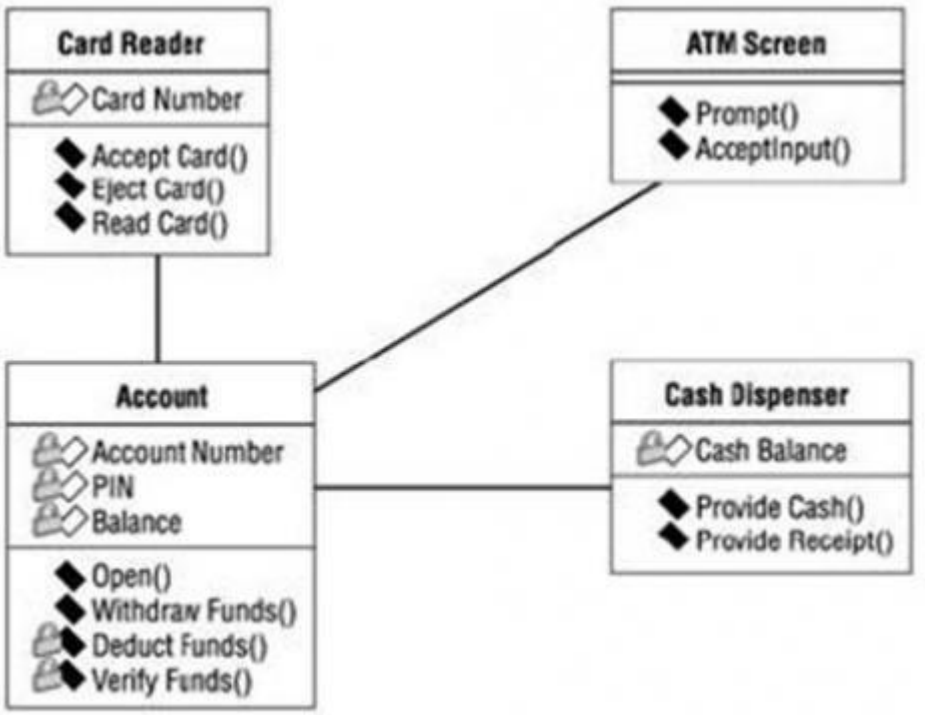
Sequence diagram menjelaskan secara detil urutan proses yang dilakukan dalam sistem untuk mencapai tujuan dari use case: interaksi yang terjadi antar class, operasi apa saja yang terlibat, urutan antar operasi, dan informasi yang diperlukan oleh masing-masing operasi. Contoh sequence diagram dapat di lihat pada gambar 2.5 berikut ini.




*Gambar II.5 Contoh sequence diagram*

### 2.3.2.4 Class Diagram

Class diagram merupakan diagram yang selalu ada di permodelan sistem berorientasi objek.Class diagram menunjukkan hubungan antar class dalam sistem yang sedang dibangun dan bagaimana mereka saling berkolaborasi untuk mencapai suatu tujuan. Penggambaran class diagram dapat di lihat dalam gambar 2.6 dibawah ini.




*Gambar II.6 Contoh class diagram*