BAB II

TINJAUAN PUSTAKA

II.1. Aplikasi

Aplikasi berasal dari kata *application* yang artinya penerapan, lamaran, penggunaan. Secara istilah aplikasi adalah program siap pakai yang direka untuk melaksanakan suatu fungsi bagi pengguna atau aplikasi yang lain dan dapat digunakan oleh sasaran yang dituju. Perangkat lunak aplikasi adalah suatu subkelas perangkat lunak komputer yang memanfaatkan kemampuan komputer langsung untuk melakukan tugas yang diinginkan pengguna. Contoh utama perangkat lunak aplikasi adalah pengolah kata, lembar kerja, dan pemutar media. (*Fricles Ariwisanto Sianturi ; 2013*)

II.2. *Goggle Map*

Google Map adalah layanan aplikasi dan teknologi peta berbasis web yang disediakan oleh *Google* secara grayang memiliki kermampuan terhadap banyak layanan pemetaan berbasis web.(Alqod Elian, Ary Mazharuddin S, Hudan Studiawan : 2012)

Skenario normal untuk sistem aplikasi peta lokasi taman ini adalah sebagai berikut :

1. Aplikasi dijalankan pertama kali, pada saat ini aplikasi dikatakan dalam kondisi *listening* dan siap untuk menerima permintaan koneksi dari *internet*.
2. Aplikasi siap melakukan koneksi ke *internet* dan secara automatis, aplikasi lalu akan memproses permintaan koneksi dari pengguna dengan

menggunakan perintah cari rute.

3. Aplikasi mengirimkan pengguna ke dalam aplikasi Google Map dan menunjukkan rute terdekat dari lokasi pengguna dengan lokasi taman yang dipilih.
4. Masalah akan timbul pada client jika koneksi *internet* pengguna tidak stabil, jika ini terjadi maka pengguna tidak akan bisa melihat rute ke lokasi taman.
5. Jika pengguna ingin keluar dari aplikasi ini maka hanya memilih pilihan "keluar" yang tertera pada aplikasi ini.

II.3. *Android*

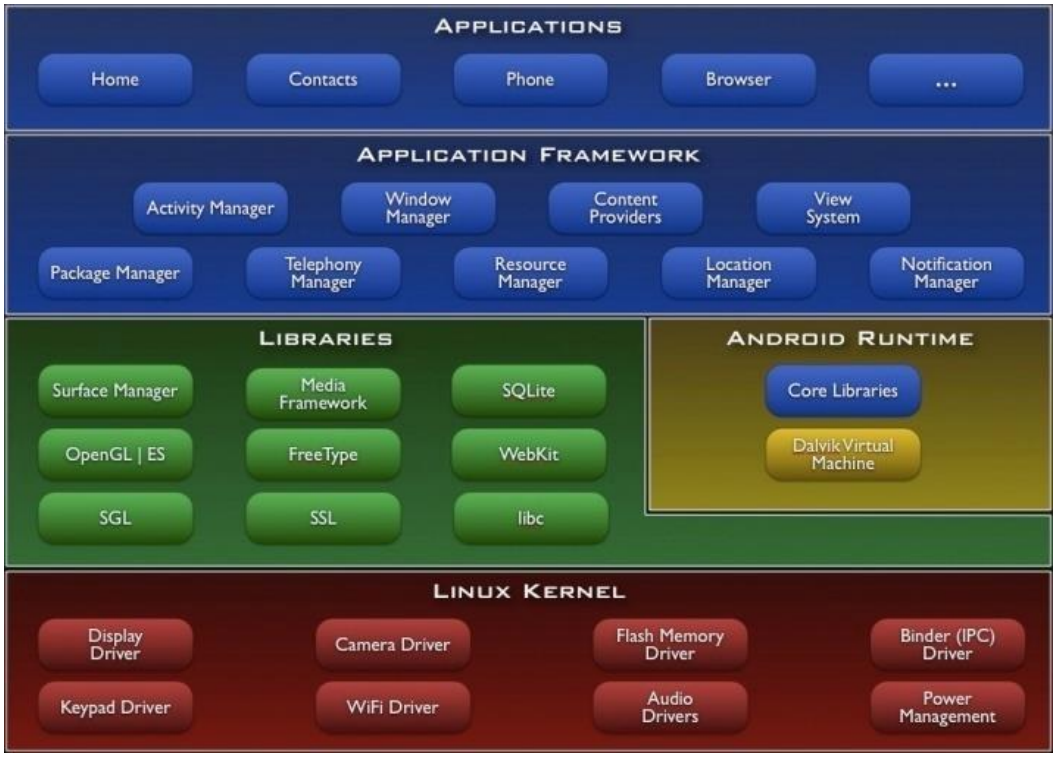
Android merupakan suatu kumpulan software yang dirancang untuk *mobile device*. Aplikasi pada platform ini dikembangkan dalam bahasa pemrograman java. Android sendiri memiliki banyak fitur diantaranya :

1. Sebuah *Application Framework* yang menyediakan banyak pengaturan API untuk membangun berbagai macam aplikasi.
2. Dalvik *virtual machine*, memungkinkan pengguna untuk menjalankan aplikasi Android.
3. Pengaturan *Global Positioning System* untuk mendapatkan lokasi pengguna dengan bantuan koneksi *internet*.
4. *Integrated browser*. Web browser berbasis WebKit *engine* terdapat pada browser default pada Android ataupun dapat diintegrasikan dengan aplikasi lain.

5. Lingkungan pengembangan yang lengkap. Termasuk *device emulator*, *tools*, untuk *debugging*, *profiling*, memori dan performa, *plugin* untuk *Android Studio Integrated Development Environment* (IDE). (Alqod Elian, Ary Mazharuddin S, Hudan Studiawan : 2012)

II.3.1. Arsitektur Android

Android terdiri dari beberapa kumpulan *software* yaitu : *Applications*, *Application Framework*, *Libraries*, *Android Runtime* dan *Kernel Linux*. Arsitektur lengkap platform Android dapat dilihat pada Gambar II.1.



Gambar II.1. Arsitektur Android
Sumber : Tim EMS ; 2015 : 4

II.3.1.1. *Aplications*

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.

II.3.1.2. *Aplications 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:

A. Sekumpulan *views*. Digunakan untuk mengatur tampilan pada aplikasi. Contohnya adalah *lists*, *grid*, *text box*, *button*, bahkan *embeddable web browser*.
B. *Content providers*. Komponen yang mengatur agar aplikasi dapat mengakses *resources* dari aplikasi lain (seperti *Contacts*), atau berbagi data dengan aplikasi lain.
C. *Resource manager*. Menyediakan akses kepada *resource non-code* seperti *localized string*, dan *file layout*
D. *Notification manager*. Memungkinkan agar suatu aplikasi dapat menampilkan peringatan yang dapat dikustomasi pada *status bar*.

E. *Activity manager*. Mengatur siklus aplikasi dan navigasi antar aplikasi yang sedang berjalan.

II.3.1.3. *Libraries*

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

1. *System* C *library*. Implementasi *library* C standar(libc).
2. *Media libraries*. Mendukung berbagai format multimedia (termasuk MPEG4, H.264, MP3, AAC, AMR, JPG, PNG).
3. *Surface manager*. Mengatur akses ke subsistem *display*.
4. LibWebCore. *Engine web browser* modern.
5. SGL. *Engine* grafis 2D.
6. 3D *library*. Implementasi OpenGL ES 1.0 yang mendukung akselarasi hardware.
7. FreeType. *Rendering* untuk *bitmap* dan *vektor font*.
8. SQLite. Basis data relasional yang sangat kecil namun sangat ampuh.

II.3.1.4. Android *Runtime*

Tiap aplikasi pada Android memiliki proses masing-masing. Tiap aplikasi tersebut memiliki instans dari Dalvik *Virtual Machine*(VM). Dalvik VM dirancang agar suatu perangkat 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*.

II.3.1.5. 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*.

II.3.2. Komponen Aplikasi

Komponen aplikasi adalah *building block* penting dari aplikasi Android. Setiap komponen adalah titik yang berbeda dimana *system* dapat masuk ke aplikasi. Tidak semua komponen adalah titik masuk yang sebenarnya bagi pengguna, beberapa bergantung pada komponen lain. Namun masing-masing komponen adalah *building block* unik yang membantu mendefinisikan perilaku keseluruhan aplikasi.

Ada empat jenis komponen aplikasi. Setiap jenis memiliki tujuan yang berbeda dan memiliki siklus hidup yang berbeda yang mendefinisikan bagaimana komponen dibuat dan dimusnahkan. Berikut ini adalah empat jenis komponen aplikasi :

1. ***Activities***, *activity* dilambangkan dengan sebuah layar dengan antarmukanya. Sebagai contoh, sebuah aplikasi *email* memiliki *activity*

yang menunjukkan tampilan dari daftar email baru, *activity* lain untuk membuat pesan, dan *activity* lain untuk membaca pesan. Meskipun *activities* bekerja sama membentuk sebuah aplikasi *email*, namun masing-masing merupakan element independen. Dengan demikian aplikasi lain dapat memulai salah satu dari *activity* tersebut (jika aplikasi *email* mengizinkan). Contohnya adalah aplikasi kamera dapat memulai aktifitas di aplikasi *email* untuk membuat pesan, agar pengguna dapat berbagi gambar.

2. ***Services***, *service* adalah komponen yang berjalan di belakang untuk menjalankan operasi yang panjang dan menjalankan kerja proses remote. *Service* tidak menyediakan antarmuka bagi pengguna. Sebagai contoh bagaimana sebuah aplikasi pemutar musik dapat berjalan di belakang saat pengguna sedang membuka aplikasi lain. Komponen lain seperti *activity* dapat memulai *service*, membiarkannya bekerja atau mengikatnya untuk berinteraksi.

3. ***Content providers***, *content provider* mengatur pembagian data antar aplikasi. Lewat *content provider*, aplikasi lain dapat membaca ataupun dapat memodifikasi (jika diizinkan) data dari aplikasi lain. Contohnya adalah bagaimana aplikasi peta lokasi seperti Google Map dapat membaca data kontak *phonebook* perangkat Android dimana dia dipasang.

4. ***Broadcast receivers***, *broadcast receiver* adalah komponen yang merespon seluruh pengumuman siaran dari sistem. Banyak siaran berasal dari sistem, contohnya siaran yang mengumumkan bahwa layar telah mati atau

baterai rendah. Aplikasi juga dapat melakukan siaran seperti misalnya mengumumkan bahwa aplikasi telah selesai melakukan *download* data dan tersedia untuk digunakan aplikasi lain. Meskipun *broadcast receiver* tidak memiliki antarmuka tetapi bisa saja komponen ini membuat notfikasi di *status bar* untuk memperingati pengguna.

II.3.3. Sistem Operasi *Android*

Adapun sistem operasi *android* menurut *Alicia Sinsum* pada tahun 2013 adalah sebagai berikut :

1. *Android OS*

   Android OS adalah sistem operasi yang berbasis *Linux*, sistem operasi *open source.* Selain *Android Software Development Kit (SDK)* untuk pengembangan aplikasi, *android* juga tersedia bebas dalam bentuk sistem operasi. Hal ini yang menyebabkan *vendor - vendor smartphone* begitu berminat untuk memproduksi *smartphone* dan komputer *tablet* berbasis *Android*. *Android OS* dapat diunduh dari situs resmi google, yaitu http://www.code.google.com. Saat ini *Android OS* sudah menyebar bukan hanya di *smartphone* saja, tetapi juga di komputer *tablet*.

2. *Android SDK (Software Development Kit)*

   Android SDK adalah *tools* API *(*Aplication Programming Interface*)* yang diperlukan untuk mengembangkan aplikasi pada *platform Android* menggunakan bahasa pemrograman *Java*. Beberapa fitur-fitur *Android* yang paling penting adalah mesin *Virtual Dalvik* yang dioptimalkan untuk

perangkat *mobile*, *integrated browser* berdasarkan *engine open source WebKit*, Grafis yang dioptimalkan dan didukung oleh *libraries* grafis 2D, grafis 3D berdasarkan spesifikasi *opengl* ES 1.0 (Opsional akselerasi perangkat keras).

Fitur - fitur *android* lainnya termasuk media yang mendukung *audio, video*, dan gambar, juga ada fitur *bluetooth*, EDGE, 3G dan *WiFi*, dengan fitur kamera, GPS, dan kompas. Selanjutnya fitur yang juga turut disediakan adalah lingkungan *Development* yang lengkap dan kaya termasuk perangkat emulator, *tools* untuk debugging, profil dan kinerja memori, dan *plugin* untuk IDE *Android Studio*.

3. *AVD (Android Virtual Device)*

Android Virtual Device merupakan emulator untuk menjalankan aplikasi *android*. Setiap AVD terdiri dari sebuah profil perangkat keras yang dapat mengatur pilihan untuk menentukan fitur *hardware* emulator. Misalnya, menentukan apakah menggunakan perangkat kamera, apakah menggunakan *keyboard* QWERTY fisik atau tidak, berapa banyak memori internal, dan lain-lain. AVD juga memiliki sebuah pemetaan versi *Android*, maksudnya kita menentukan versi dari *platform Android* akan berjalan pada emulator. Pilihan lain dari AVD, misalnya menentukan *skin* yang kita ingin gunakan pada emulator, yang memungkinkan untuk menentukan dimensi layar, tampilan, dan sebagainya. Kita juga dapat menentukan SD *Card virtual* untuk digunakan dengan di emulator.

II.4. Jaringan

Pengertian jaringan komputer merupakan sekumpulan komputer serta perangkat - perangkat lain pendukung komputer yang saling terhubung dalam suatu kesatuan. Media jaringan komputer dapat melalui kabel-kabel atau tanpa kabel sehingga memungkinkan pengguna jaringan komputer dapat saling melakukan pertukaran informasi seperti dokumen dan data. Dapat juga melakukan pencetakan pada printer yang sama dan bersama-sama memakai perangkat keras dan perangkat lunak yang terhubung dengan jaringan.

II.4.1. Jenis-Jenis Jaringan

Berdasarkan jangkauan area atau lokasi jaringan dibedakan menjadi 3 jenis, yaitu :

1. LAN (*Local Area Network*)

   Merupakan jaringan lokal yang dibuat pada area tertutup. Misalkan dalam satu gedung atau dalam satu ruangan. LAN biasa digunakan untuk jaringan kecil yang menggunakan *resource* bersama. Seperti penggunaan *printer* secara bersama. LAN dapat menggunakan media komunikasi seperti kabel dan *wireless*.

2. MAN (*Metropolitan Area Network*)

   Merupakan jaringan antara LAN satu dengan LAN lain yang dipisahkan daerah lokasi yang cukup jauh. Contoh penggunaan MAN adalah hubungan antara kantor pusat dengan kantor cabang yang ada di daerah-daerah. Dapat dikatakan MAN merupakan pengembangan dari LAN.

3. WAN (Wide Area Network)

   Merupakan jaringan yang cakupannya lebih luas dari pada MAN. Cakupan WAN meliputi satu kawasan, satu negara, satu pulau, bahkan satu benua. Metode yang digunakan WAN hampir sama dengan LAN dan MAN. (*Choirul Muallifah ; 2013 : 2)*

II.4.2. Jaringan Komputer Bertopologi / Pemasangan

1. Bus

Topologi bus merupakan topologi yang banyak dipergunakan pada masa penggunaan kabel sepaksi menjamur. Dengan menggunakan T-*Connector* (dengan terminator 50ohm pada ujung *network*), maka komputer atau perangkat jaringan lainnya bisa dengan mudah dihubungkan satu sama lain. Instalasi jaringan Bus sangat sederhana, murah dan maksimal terdiri atas 5-7 komputer. Kesulitan yang sering dihadapi adalah kemungkinan terjadinya tabrakan data karena mekanisme jaringan relative sederhana dan jika salah satu node putus maka akan mengganggu kinerja dan trafik seluruh jaringan.

2. *Ring*

Topologi cincin adalah topologi jaringan berbentuk rangkaian titik yang masing-masing terhubung ke dua titik lainnya, sedemikian sehingga membentuk jalur melingkar membentuk cincin. Pada topologi cincin, komunikasi data dapat terganggu jika satu titik mengalami gangguan.

3. *Star*

Topologi bintang merupakan bentuk topologi jaringan yang berupa *konvergensi* dari *node* tengah ke setiap *node* atau pengguna. Topologi jaringan bintang termasuk topologi jaringan dengan biaya menengah. (*Mardison, Al Husni ; 2012 : 59*)

4. *Mesh*

Topologi jala atau Topologi mesh adalah suatu bentuk hubungan antar perangkat dimana setiap perangkat terhubung secara langsung ke perangkat lainnya yang ada di dalam jaringan. Akibatnya, dalam topologi *mesh* setiap perangkat dapat berkomunikasi langsung dengan perangkat yang dituju (*dedicated links*).

5. Pohon

Topologi Pohon adalah kombinasi karakteristik antara topologi bintang dan topologi bus. Topologi ini terdiri atas kumpulan topologi bintang yang dihubungkan dalam satu topologi bus sebagai jalur tulang punggung atau *backbone*. Komputer-komputer dihubungkan ke *hub*, sedangkan *hub* lain di hubungkan sebagai jalur tulang punggung. Topologi jaringan ini disebut juga sebagai topologi jaringan bertingkat. Topologi ini biasanya digunakan untuk interkoneksi antar sentral dengan hirarki yang berbeda. Untuk hirarki yang lebih rendah digambarkan pada lokasi yang rendah dan semakin keatas mempunyai hirarki semakin tinggi. Topologi jaringan jenis ini cocok digunakan pada sistem jaringan komputer. (*Mardison, Al Husni ; 2012 : 59*)

6. *Linier*

Jaringan komputer dengan topologi runtut (*linear topology*) biasa disebut dengan topologi bus beruntut, tata letak ini termasuk tata letak umum. Satu kabel utama menghubungkan tiap titik sambungan (komputer) yang dihubungkan dengan penyambung yang disebut dengan Penyambung-T dan pada ujungnya harus diakhiri dengan sebuah penamat (*terminator*). Penyambung yang digunakan berjenis BNC (*British Naval Connector*: Penyambung *Bahari Britania*), sebenarnya BNC adalah nama penyambung bukan nama kabelnya, kabel yang digunakan adalah RG 58 (Kabel *Sepaksi Thinnet*). Pemasangan dari topologi bus beruntut ini sangat sederhana dan murah tetapi sebanyaknya hanya dapat terdiri dari 5-7 komputer. (*Choirul Muallifah ; 2013 : 2)*

II.4.3. Manfaat dan Keuntungan Jaringan

Adapun manfaat dan keuntungan membangun jaringan menurut *Choirul Muallifah* tahun 2013 adalah sebagai berikut :

1. Manfaat Jaringan Komputer:
    a. *Sharing resources*
    b. Media komunikasi
    c. Integrasi data
    d. Pengembangan dan pemeliharaan
    e. Keamanan data
    f. Sumber daya lebih efisien dan informasi terkini

2. Keuntungan Membangun Jaringan Komputer Yaitu:
    a. Dapat berbagi peralatan (*peripheral*) dan penggunaanya, seperti printer, *harddisk*, modem.
    b. Memudahkan bertukar data diantara pengguna komputer tanpa harus menggunakan *disket atau flashdisk*.
    c. Kita dapat menggunakan program - program yang ada di komputer pusat.
    d. Bisa mengirim dan menerima *email* dari *internet* dan mencari informasi lain melalui fasilitas *internet*. (*Choirul Muallifah ; 2013 : 2)*

II.5. *Java*

Java memiliki cara kerja yang unik dibandingkan dengan bahasa perograman lainya yaitu bahasa perograman *java* bekerja mengunakan *interpreter* dan juga *compiler* dalam proses pembuatan program, *Interpreter java* dikenal sebagai perograman *bytecode* yaitu dengan cara kerja mengubah paket *class* pada *java* dengan *extensi*. *Java* menjadi *.class*, hal ini dikenal sebagai *class bytecode*, yaitunya *class* yang dihasilkan agar program dapat dijalankan pada semua jenis perangkat dan juga *platform*, sehingga program *java* cukup ditulis sekali namun mampu bekerja pada jenis lingkungan yang berbeda. (*Defni, Indri Rahmayun* ; 2014 : 64)

II.6. Data *Flow* Diagram (DFD)

Data *Flow* Diagram (DFD) adalah alat pembuatan model yang memungkinkan profesional sistem untuk menggambarkan sistem sebagai suatu jaringan proses fungsional yang dihubungkan satu sama lain dengan alur data, baik secara manual maupun komputerisasi. DFD ini sering disebut juga dengan

nama Bubble chart, Bubble diagram, model proses, diagram alur kerja, atau model fungsi. DFD ini adalah salah satu alat pembuatan model yang sering digunakan, khususnya bila fungsi-fungsi sistem merupakan bagian yang lebih penting dan kompleks dari pada data yang dimanipulasi oleh sistem. Dengan kata lain, DFD adalah alat pembuatan model yang memberikan penekanan hanya pada fungsi sistem. DFD ini merupakan alat perancangan system yang berorientasi pada alur data dengan konsep dekomposisi dapat digunakan untuk penggambaran analisa maupun rancangan ,sistem yang mudah dikomunikasikan oleh profesional sistem kepada pemakai maupun pembuat program.. (Yerenia Yuliawan : 2013)

II.7. UML (*Unified Modelling Language*)

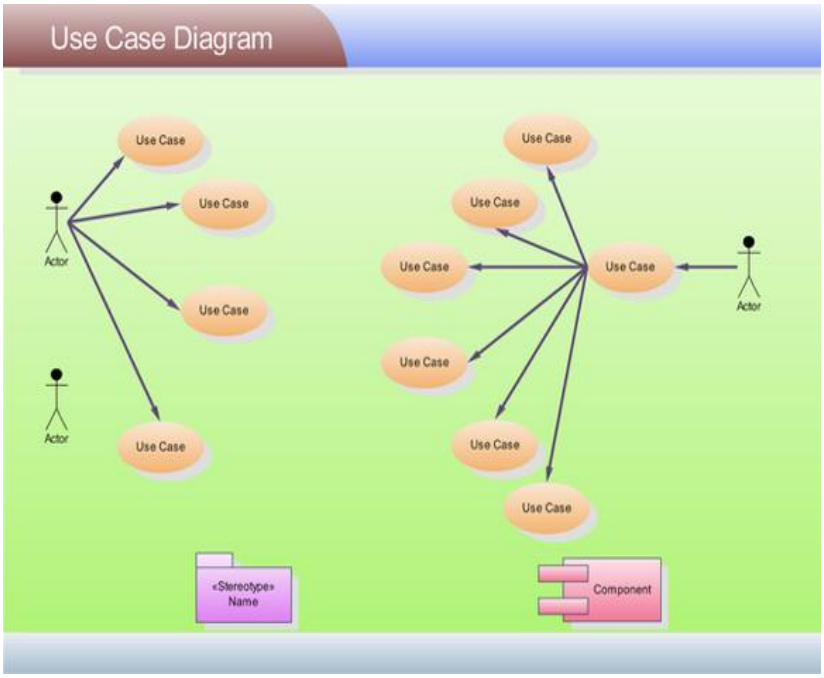
Unified Modelling Language (UML) adalah sebuah "bahasa" yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem. Dengan menggunakan UML kita dapat membuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan *class* dan *operation* dalam konsep dasarnya, maka ia lebih cocok untuk penulisan peranti lunak dalam bahasa-bahasa berorientasi objek seperti C++, *Java*, C# atau *VB.NET*. Walaupun demikian, UML tetap dapat digunakan untuk modeling aplikasi prosedural dalam VB atau C. Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan *syntax*/semantik. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram piranti lunak. Setiap

bentuk memiliki makna tertentu, dan UML *syntax* mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan. Notasi UML terutama diturunkan dari 3 notasi yang telah ada sebelumnya: *Grady Booch OOD* (*Object-Oriented Design*), Jim Rumbaugh OMT (*Object Modeling Technique*), dan Ivar Jacobson OOSE (*Object-Oriented Software Engineering*). Sejarah UML sendiri cukup panjang. Sampai era tahun 1990 seperti kita ketahui 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 group/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 dirilis *draft* pertama dari UML (versi 0.8). Sejak tahun 1996 pengembangan tersebut dikoordinasikan oleh *Object Management Group* (OMG – http://www.omg.org). Tahun 1997 UML versi 1.1 muncul, dan saat ini versi terbaru adalah versi 1.5 yang dirilis bulan Maret 2003. *Booch*, *Rumbaugh* dan *Jacobson* menyusun tiga buku serial tentang UML pada tahun 1999. Sejak saat itulah UML telah menjelma menjadi standar bahasa pemodelan untuk aplikasi berorientasi objek

Dalam pembuatan skripsi ini penulis menggunakan diagram *Use Case* yang terdapat di dalam UML. Adapun maksud dari *Use Case* Diagram diterangkan dibawah ini.

1. *Use Case Diagram*

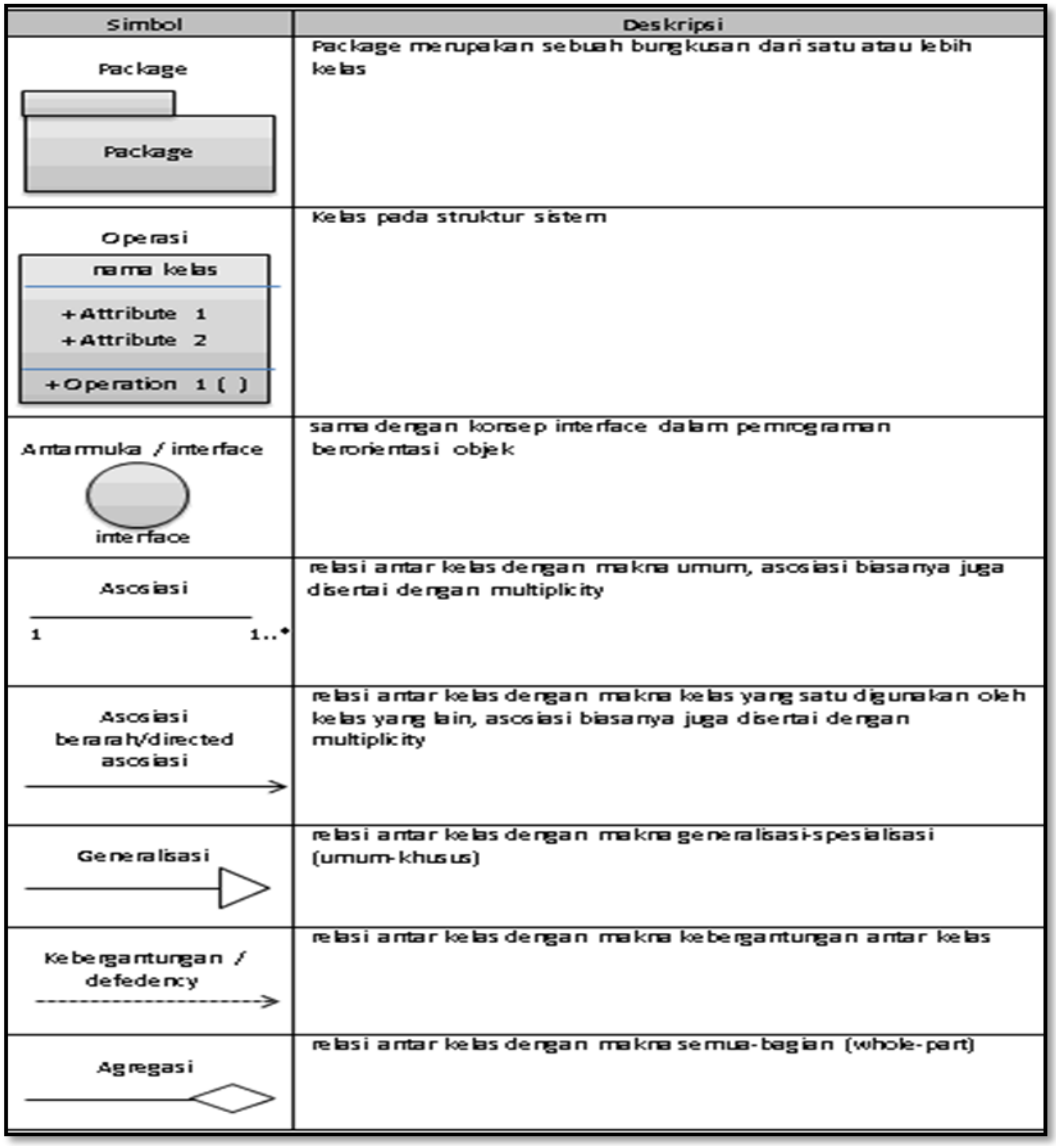
Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya login 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. *Use case diagram* dapat sangat membantu bila kita sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem. 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. *(Yuni Sugiarti ; 2013 : 41)*



Gambar II.2. ***Contoh Use Case Diagram***
Sumber : yuni sugiarti ; 2013 : 41

2. *Class Diagram*

Diagram kelas atau *class diagram* menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi. Berikut adalah simbol-simbol pada diagram kelas :



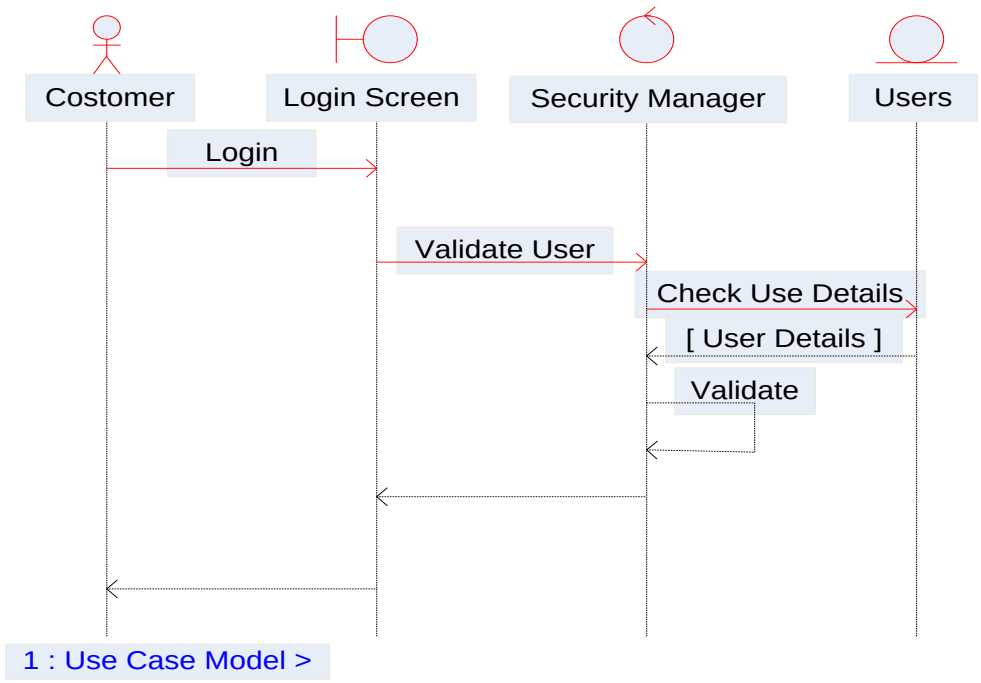
Simbol	Deskripsi
Package Package	Package merupakan sebuah bungkusan dari satu atau lebih kelas
Operasi nama kelas +Attribute 1 +Attribute 2 +Operation 1 ( )	Kelas pada struktur sistem
Antarmuka / interface interface	sama dengan konsep interface dalam pemrograman berorientasi objek
Asosiasi 1 1..*	relasi antar kelas dengan makna umum, asosiasi biasanya juga disertai dengan multiplicity
Asosiasi berarah/directed asosiasi	relasi antar kelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai dengan multiplicity
Generalisasi	relasi antar kelas dengan makna generalisasi-spesialisasi (umum-khusus)
Kebergantungan / defedency	relasi antar kelas dengan makna kebergantungan antar kelas
Agregasi	relasi antar kelas dengan makna semua-bagian (whole-part)

Gambar II.3. ***Contoh Class Diagram***
Sumber : yuni sugiarti ; 2013 : 59

3. *Sequence Diagram*

Diagram *Sequence* menggambarkan kelakuan/prilaku objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambarkan diagram *sequence* maka harus diketahui objek - objek yang terlibat dalam sebuah *use case* beserta metode - metode yang dimiliki kelas yang diinstansiasi menjadi objek itu.

Banyaknya diagram *sequence* yang harus digambar adalah sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram *sequence* sehingga semakin banyak *use case* yang didefinisikan maka diagram *sequence* yang harus dibuat juga semakin banyak.



Gambar II.4. *Contoh Sequence Diagram*
Sumber : yuni sugiarti ; 2013 : 41

4. *Activity Diagram*

Activity diagram menggambarkan berbagai alur aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alur berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

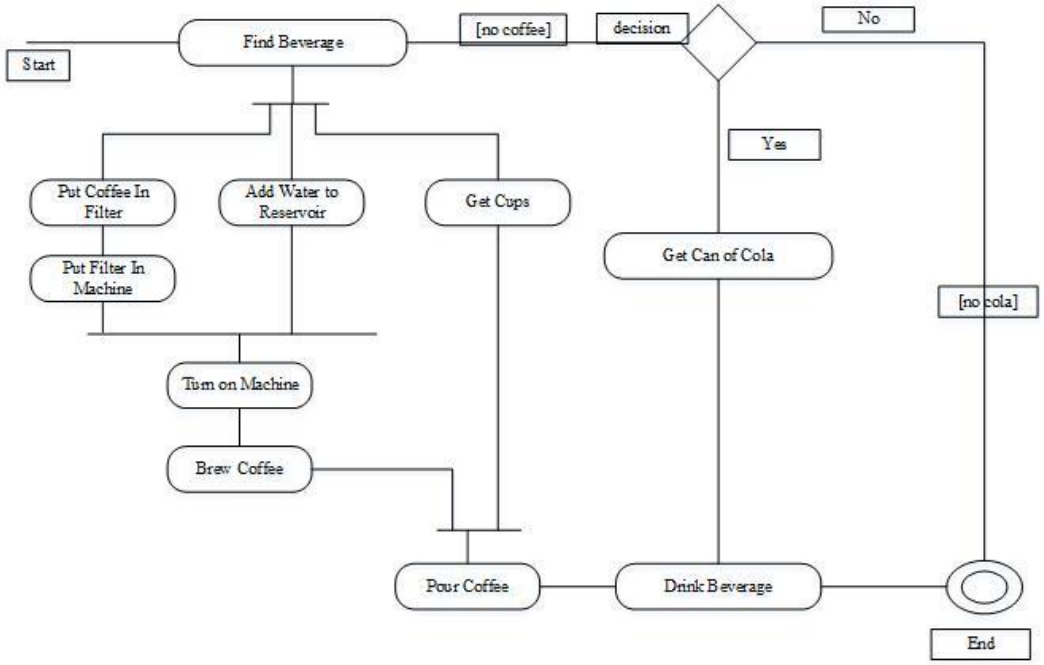
Activity diagram merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state*

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.

Sama seperti *state*, standar UML menggunakan segi empat dengan sudut membulat untuk menggambarkan 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.



Gambar II.5. Contoh Activity Diagram
Sumber : yuni sugiarti ; 2013 : 75