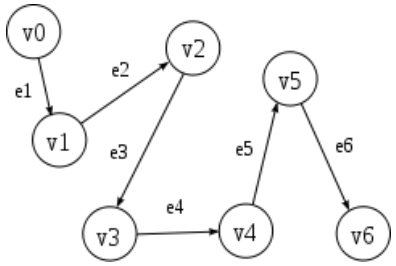
BAB II
TEORI PENUNJANG

2.1 Teori Dasar Graf

Dalam matematika dan ilmu komputer, teori graf adalah cabang kajian yang mempelajari sifat-sifat graf. Secara informal, suatu graf adalah himpunan benda-benda yang disebut simpul (*vertex* atau *node*) yang terhubung oleh sisi (*edge*) atau busur (*arc*). Biasanya graf digambarkan sebagai kumpulan titik-titik (melambangkan simpul) yang dihubungkan oleh garis-garis (melambangkan sisi) atau garis berpanah (melambangkan busur). Suatu sisi dapat menghubungkan suatu simpul dengan simpul yang sama. Sisi yang demikian dinamakan gelang (*loop*). Banyak sekali struktur yang bisa direpresentasikan dengan graf, dan banyak masalah yang bisa diselesaikan dengan bantuan graf. Jaringan persahabatan pada Facebook bisa direpresentasikan dengan graf, yakni simpul-simpulnya adalah para pengguna Facebook dan ada sisi antar pengguna jika dan hanya jika mereka berteman. Perkembanganalgoritma untuk menangani graf akan berdampak besar bagi ilmu komputer[4].

Suatu graph G dapat dinyatakan sebagai **G = <V,E>** Graph G terdiri atas himpunan V yang berisikan simpul pada graf tersebut dan himpunan dari E yang berisi sisi pada graf tersebut. Himpunan E dinyatakan sebagai pasangan dari simpul yang ada dalam V. Sebagai contoh definisi dari graf pada gambar di atas adalah : **V = {1,2,3,4,5,6}** dan **E = {(1,2), (1,5), (2,3), (3,4), (4,5), (5,2), (4,6)}**[1]

Pada digraf maka pasangan-pasangan ini merupakan pasangan terurut. Untuk menyatakan digraf (gambar kedua yang menggunakan tanda panah) kita dapat menggunakan himpunan edge seoerti terdapat pada gambar 2.1 :



Gambar 2.1 gambar dengan node yang sama

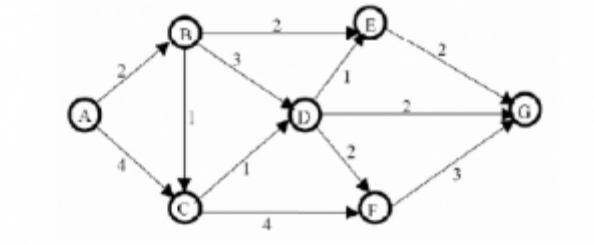
Sebuah struktur graf bisa dikembangkan dengan memberi bobot pada tiap sisi. Graf berbobot dapat digunakan untuk melambangkan banyak konsep berbeda. Sebagai contoh jika suatu graf melambangkan jaringan jalan maka bobotnya bisa berarti panjang jalan maupun batas kecepatan tertinggi pada jalan tertentu. Ekstensi lain pada graf adalah dengan membuat sisinya berarah, yang secara teknis disebut graf berarah atau digraf (*directed graph*). Digraf dengan sisi berbobot disebut jaringan.[4]

Jaringan banyak digunakan pada cabang praktis teori graf yaitu analisis jaringan. Perlu dicatat bahwa pada analisis jaringan, definisi kata "jaringan" bisa berbeda, dan sering berarti graf sederhana (tanpa bobot dan arah)[4].

2.1.1 *Macam – Macam Graf*

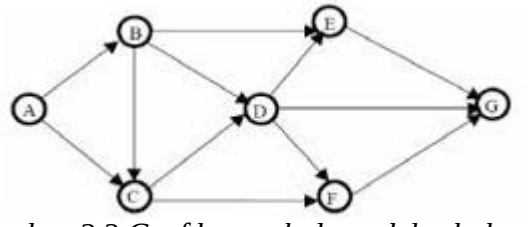
Berdasarkan arah dan bobotnya, graf digolongkan atas 4 jenis, yaitu

1. Graf berarah dan berbobot adalah graf yang setiap sisinya mempunyai orientasi arah dan bobot, seperti terdapat pada gambar 2.2.



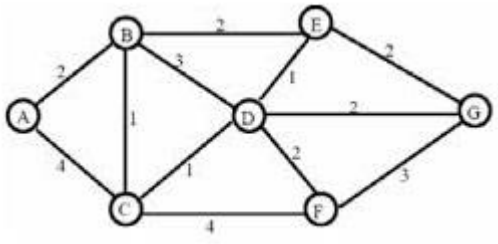
Gambar 2.2 Graf berarah dan berbobot

2. Graf berarah dan tak berbobot adalah graf yang setiap sisinya mempuyai arah dan tidak berbobot seperti terdapat pada gambar 2.3.



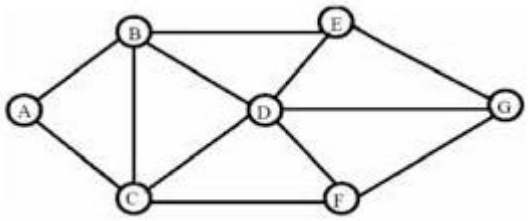
Gambar 2.3 Graf berarah dan tak berbobot

3. Graf tidak berarah dan berbobot adalah graf yang setiap sisinya tidak mempunyai arah tetapi memiliki bobot seperti terdapat pada gambar 2.4.



Gambar 2.4 Graf tidak berarah dan berbobot

4. Graf tidak berarah dan tidak berbobot adalah graf yang setiap sisinya tidak memiliki arah dan bobot seperti terdapat pada gambar 2.5.



Gambar 2.5 Graf tidak berarah dan tidak berbobot

2.2 Jalur Terpendek

Jalur terpendek adalah Lintasan terpendek lintasan minimum yang diperlukan untuk mencapai suatu tempat dari tempat tertentu. Lintasan minimum yang dimaksud dapat dicari dengan menggunakan graf, Di sini menggunakan Graf berarah dan berbobot adalah graf yang setiap sisinya mempunyai arah dan memiliki bobot.[6]

Persoalan lintasan terpendek yaitu menemukan lintasan terpendek antara dua atau beberapa simpul lebih yang berhubungan. Persoalan mencari lintasan terpendek di dalam graf merupakan salah satu persoalan optimasi. Persoalan ini biasanya direpresentasikan dalam bentuk graf. Graf yang digunakan dalam pencarian lintasan terpendek atau shortest path adalah graf bearah tapi berbobot, yaitu graf yang tak bearah akan tetapi

setiap sisinya diberikan suatu nilai atau bobot. Bobot pada sisi graf dapat menyatakan jarak antar kota, waktu pengiriman pesan, ongkos pembangunan, dan sebagainya.

Asumsi yang kita gunakan di sini adalah bahwa semua bobot bernilai positif. Kata "terpendek" jangan selalu diartikan secara fisik sebagai panjang minimum, sebab kata "terpendek" berbeda-beda makanya bergantung pada tipikal perasoalan yang akan diselesaikan. Namun, secara umum "terpendek" berarti meminimalisasi bobot pada suatu lintasan di dalam graf. [6]

Contoh-contoh terapan pencarian lintasan terpendek misalnya:

1. Misalkan simpul pada graf dapat merupakan kota, sedangkan sisi menyatakan jalan yang menghubungkan dua buah kota. Bobot sisi graf dapat menyatakan jarak antara dua buah kota atau rata-rata waktu tempuh antara dua buah kota. Apabila terdapat lebih dari satu lintasan dari kota A ke kota B, maka persoalan lintasan terpendek di sini adalah menentukan jarak terpendek atau waktu tersingkat dari kota A ke kota B.
2. Misalkan simpul pada graf dapat merupakan terminal komputer atau simpul komunikasi dalam suatu jaringan, sedangkan sisi menyatakan saluran komunikasi yang menghubungkan dua buah terminal. Bobot pada graf dapat menyatakan biaya pemakaian saluran komunikasi antara dua buah terminal, jarak antara dua buah terminal, atau waktu pengiriman pesan (message) antara dua buah terminal. Persoalan lintasan terpendek di sini adalah menentukan jalur komunikasi terpenek antara dua buah terminal komputer. Lintasan terpendek akan menghemat waktu pengiriman pesan dan biaya komunikasi.

   Ada beberapa macam persoalan lintasan terpendek, antara lain:

   a. Lintasan terpendek antara dua buah simpul tertentu (a pair shortest path).
   b. Lintasan terpendek antara semua pasangan simpul (all pairs shortest path).

c. Lintasan terpendek dari simpul tertentu ke semua simpul yang lain (single-source shortest path).
d. Lintasan terpendekan antara dua buah simpul yang melalui beberapa simpul tertentu (intermediate shortest path).

2.3 Algoritma Dijstrak

Algoritma Dijkstrak adalah suatu algoritma untuk menemukana jarak terpendek pada tiap lokasi (*vertices*) pada suatu graf. Algoritma Dijstrak di temukan oleh seorang ilmuwan computer berkebangsaan Belanda Edsger Wybe Dijstrak [2]. Algoritma Dijkstra adalah sebuah algoritma untuk memecahkan permasalahan jalur terpendek (*shortest path problem*) pada sebuah graf (*directed graf*) atau graf tidak berarah (*undirected graf*) dengan bobot-bobot sisi (*edge weights*) yang bernilai tidak negative[9].

Dinama metode Algoritma Dijstrak ini bertujuan untuk menemukan jalur terpendek berdasarkan bobot terkecil dari satu titik ke titik lainnya. Misalkan titik mengambarkan gedung dan garis menggambarkan jalan, maka algoritma Dijkstra melakukan kalkulasi terhadap semua kemungkinan bobot terkecil dari setiap titik.

Algoritma Dijkstra diterapkan untuk mencari lintasan terpendek pada graf berarah. Namun, algoritma ini juga benar untuk graf tak berarah. Algoritma Dijkstra mencari lintasan terpendek dalam sejumlah langkah. Algoritma ini menggunakan prinsip greedy. Prinsip greedy pada algoritma dijkstra menyatakan bahwa pada setiap langkah kita memilih sisi yang berbobot minimum dan memasukannya dalam himpunan solusi.

Properti Algoritma dijstrak[7] :

Misalkan sebuah graf berbobot dengan n buah simpul dinyatakan dengan matriks M=[mij], yang dalam hal ini:

mij = bobot sisi (i,j) *(pada graf tak berarah mij =mji)*

mii = 0

mij = ∞ , jika tidak ada sisi dari simpul I ke simpul j

Selain matriks M, juga menggunakan tabel S=[si], yang dalam hal ini:

si = 1, jika simpul i termasuk ke dalam lintasan terpendek

si = 0, jika simpul i tidak termasuk ke dalam lintasan terpendek dan tabel D=[di], yang dalam hal ini

di = panjang lintasan dari simpul awal a ke simpul i

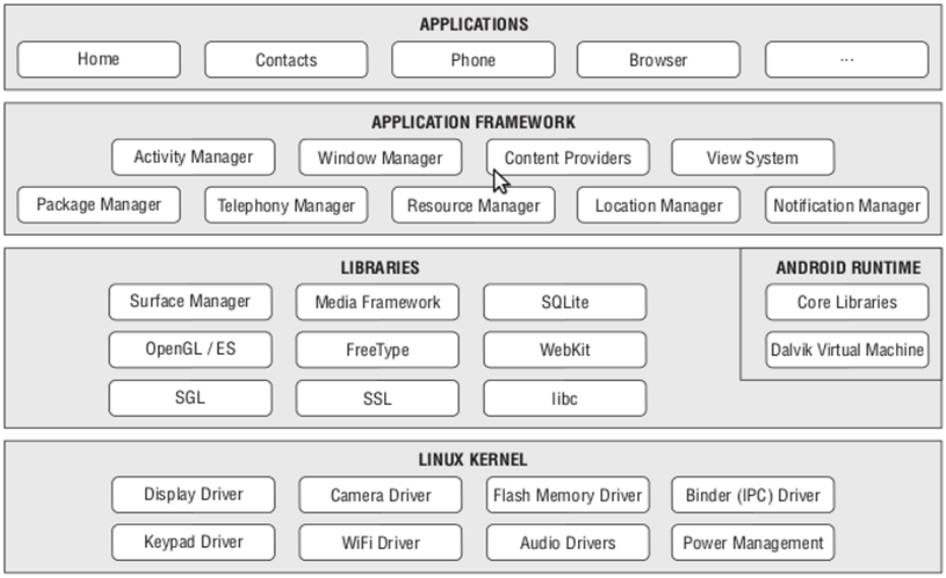
2.4 Android

Android merupakan suatu *software stack* untuk *perangkat* mobile. Di dalamnya terdapat sistem operasi, *middleware,* dan *key application*. Aplikasi pada *platform* ini dikembangkan dalam bahasa pemrograman java. Android sendiri memiliki banyak fitur, diantaranya[3]:

1. Merupakan sebuah *Application Framework* sehingga *programmer* dapat menggunakan beberapa fungsi yang telah disediakan.
2. Dalvik *virtual machine*. Tiap aplikasi dalam Android memiliki *instance virtual machine* yang dapat bekerja secara efisien dalam lingkungan memori yang terbatas.
3. *Integrated browser.* Web browser berbasis WebKit e*ngine* terdapat pada browser *default* Android ataupun dapat diintegrasikan dengan aplikasi lain.
4. *Optimized graphic. Library* grafis 2D yang kaya dan 3D berbasis OpenGL ES 1.0 yang mendukung akselerasi hardware.
5. SQLite. Basis data relasional yang ringan namun sangat *powerful.*
6. *Media Support.* Mendukung berbagai format audio, video dan gambar (MPEG, H.264, MP3, AAC, AMR, JPG, PNG, GIF).
7. GSM *Telephony.* Mendukung fungsi komunikasi GSM.
8. Blutooth, EDGE, 3G dan WiFi. Mendukung komunikasi pada jaringan (tergantung hardware).
9. Kamera, GPS, kompas dan *accelerometer.* Mendukung berbagai fitur yang disediakan oleh *hardware*.
10. *Tools Developer* yang lengkap. Termasuk *device emulator, tools* untuk *debugging, profiling* memori dan performa, *plugin* untuk Eclipse IDE.

2.4.1 *Arsitektur Android*

Sistem operasi Android dibagi menjadi lima bagian dalam empat lapisan utama, seperti yang dapat di lihat pada gambar 2.6 di bawah ini.



Gambar 2.6 Arsitektur Android

Penjelasan masing-masing lapisan:

1. *Kernel* Linux: Merupakan *kernel* (inti) dari sistem operasi Android. Lapisan ini meliputi seluruh *driver* pada tingkatan *device* untuk berbagai jenis komponen *hardware* dari perangkat Android.
2. *Libraries*: Meliputi seluruh kode yang merupakan fitur utama dari sistem Android. Contohnya *Library* SQLite yang merupakan *library* untuk yang menyediakan layanan *database* sehingga aplikasi dapat menggunakannya sebagai tempat penyimpanan data.Contoh lainnya *Library* Webkit yang menyediakan segala fungsi yang berkaitan dengan *web browsing*.
3. Android *runtime*: Berada pada lapisan yang sama sebagai *library*. Android *runtime* menyediakan inti *libraries* yang memungkinkan pengembang untuk menulis aplikasi Android dengan bahasa pemrograman Java. Android *runtime* juga menyertakan Dalvik *virtual machine* (aplikasi Android di kompilasi kedalam file Dalvik yang dapat dieksekusi). Dalvik merupakan mesin virtual spesial yang di rancang secara khusus untuk android dan dioptimalkan untuk perangkat *mobile* sumber batere dengan memory dan CPU yang terbatas.
4. *Application framework*: Memperlihatkan berbagai kemampuan dari sistem

android. Disediakan untuk pengembang aplikasi, sehingga dapat digunakan dalam aplikasi yang dibuat

5. *Application*: Merupakan lapisan teratas. Segala macam aplikasi yang dibuat akan ditempatkan pada lapisan ini.

2.4.2 *Struktur Aplikasi Android*

Arsitektur Android mengadopsi konsep *component reuse*, yang memungkinkan untuk mempublikasikan dan berbagi *activities*, *service* dan data dengan aplikasi lain, dengan batasan akses kelola. Berikut layanan aplikasi yang merupakan landasan arsitektur dari semua aplikasi Android[4]:

1. *Activity Manager*: mengontrol siklus hidup suatu Aktivitas, termasuk manajemen tumpukan aktivitas
2. *Views*: Digunakan untuk membuat antarmuka *user* untuk aktivitas yang dibuat.
3. *Notification Manager*: Menyediakan mekanisme konsisten dan *nonintrusive* yang memberi peringatan kepada *user.*
4. *Content Providers*: Membiarkan aplikasi untuk berbagi data
5. *Resource Manager*: Mendukung non-kode sumber seperti *string* dan grafis sebagai *tool* ekternal.

2.5 Pemrograman Berorientasi Objek

Pemrograman berorientasi objek (*object-oriented programming* disingkat OOP) merupakan paradigma pemrograman yang berorientasikan kepada objek, dimana semua data dan fungsi dibungkus dalam *class-class* atau *object-object*. Setiap *object* dapat menerima pesan, memproses data, mengirim, menyimpan dan memanipulasi data. Beberapa *object* berinteraksi dengan saling memberikan informasi satu terhadap lainya. Masing-masing *object* harus berisikan informasi mengenai dirinya sendiri dan dapat dihubungkan dengan *object* yang lain. Konsep dasar dari Pemrograman berorientasi objek antara lain[1]:

1. Kelas (*Class*)

   Merupakan model yang berisi kumpulan *attribute* dan *method* dalam suatu unit untuk suatu tujuan tertentu. Sebagai contoh *class* manusia memiliki

attribute berat, tinggi, usia kemudian memiliki *method* makan, minum, tidur. *Method* dalam sebuah *class* dapat merubah *attribute* yang dimiliki oleh *class* tersebut. Sebuah *class* merupakan dasar dari modularitas dan struktur dalam pemrograma berorientasi objek.

2. Atribut (*Attribute*)

   Attribute dari sebuah *class* adalah variabel global yang dimiliki sebuah *class*, *attribute* dapat memiliki hak akses *private, public* maupun *protected*.

3. Metode (*Method*)

   Method adalah fungsi atau prosedur yang dibuat oleh seorang *programmer* di dalam suatu *class*. Dengan kata lain, *method* pada sebuah *class* hampir sama dengan fungsi atau prosedur pada pemrograman prosedural. Pada sebuah *method* di dalam sebuah *classi* juga memiliki izin akses seperti halnya *attribute*.

4. Objek (*Object*)

   Merupakan perwujudan dari *class*, setiap *object* akan mempunyai *atrribute* dan *method* yang dimiliki oleh *class*-nya, contoh: andi, budi, yani merupakan *object* dari *class* manusia. Setiap *object* dapat berinteraksi dengan *object* lainnya meskipun berasal dari *class* yang berbeda.

5. Abstraksi (*Abstraction*)

   Adalah suatu cara untuk melihat suatu *object* dalam bentuk yang lebih sederhana. Sebagai contoh kita tidak melihat sebuah mobil sebagai ribuan komponen elektronik, sistem mekanik dengan empat bauh ban, jok, stir kemudi dan sebagainya. Dengan abstraksi, suatu sistem yang kompleks dapat dipandang sebagai kumpulan subsistem-subsistem yang lebih sederhana, seperti halnya mobil merupakan suatu sistem yang terdiri atas berbagai subsistem, seperti subsistem kemudi, subsistem pengereman dan sebagainya.

6. Enkapsulasi (*Encapsulation*)

   Merupakan suatu mekanisme untuk menyembunyikan atau memproteksi suatu proses dari kemungkinan interferensi atau penyalahgunaan dari luar sistem dan sekaligus menyederhanakan penggunaan sistem tersebut. Memastikan pengguna sebuah *object* tidak dapat mengganti keadaan dalam dari sebuah *object* dengan cara yang tidak layak. Hanya metode dalam *object*

tersebut yang diberi ijin untuk mengakses keadaannya. Setiap *object* mengakses *interface* yang menyebutkan bagaimana *object* lainnya dapat berinteraksi dengannya. *Object* lainnya tidak akan mengetahui dan tergantung kepada representasi dalam *object* tersebut.

7. Polimorfisme (*Polymorphism*)

   Merupakan konsep yang memungkinkan digunakannya suatu *interface* yang sama untuk memerintah suatu *object* agar melakukan suatu tindakan yang mungkin secara prinsip sama tetapi secara proses berbeda. Contoh, bila seekor burung menerima pesan "gerak cepat", dia akan menggerakan sayapnya dan terbang. Bila seekor singa menerima pesan yang sama, dia akan menggerakan kakinya dan berlari. Keduanya menjawab sebuah pesan yang sama, namun yang sesuai dengan kemampuan hewan tersebut.

8. Pewarisan (*Inheritance*)

   Merupakan konsep mewariskan *attribute* dan *method* yang dimiliki oleh sebuah *class* kepada *class* turunannya. Dengan konsep ini, *class* yang dibuat cukup mendefinisikan *attribute* dan *method* yang spesifik di dalamnya, sedangkan *attribute* dan *method* yang lebih umum akan didapatkan dari *class* yang menjadi induknya. Mengatur polimorfisme dan enkapsulasi dengan mengijinkan *object* didefinisikan dan diciptakan dengan jenis khusus dari *object* yang sudah ada. *Object-object* ini dapat membagi (dan memperluas) perilaku mereka tanpa harus mengimplementasikan ulang perilaku tersebut.

2.6 UML

Unified Modelling Language (UML) adalah sebuah bahasa pemrograman yang telah menjadi standar untuk merancang dan mendokumentasikan sistem perangkat lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem dan sudah digunakan secara luas dan menggunakan notasi yang sudah dikenal untuk analisa dan desain berorientasi objek.

Unified Modelling Language (UML) merupakan bahasa *visual* untuk pemodelan dan komunikasi mengenai sebuah sistem dengan menggunakan diagram dan teks-teks pendukung.[1]

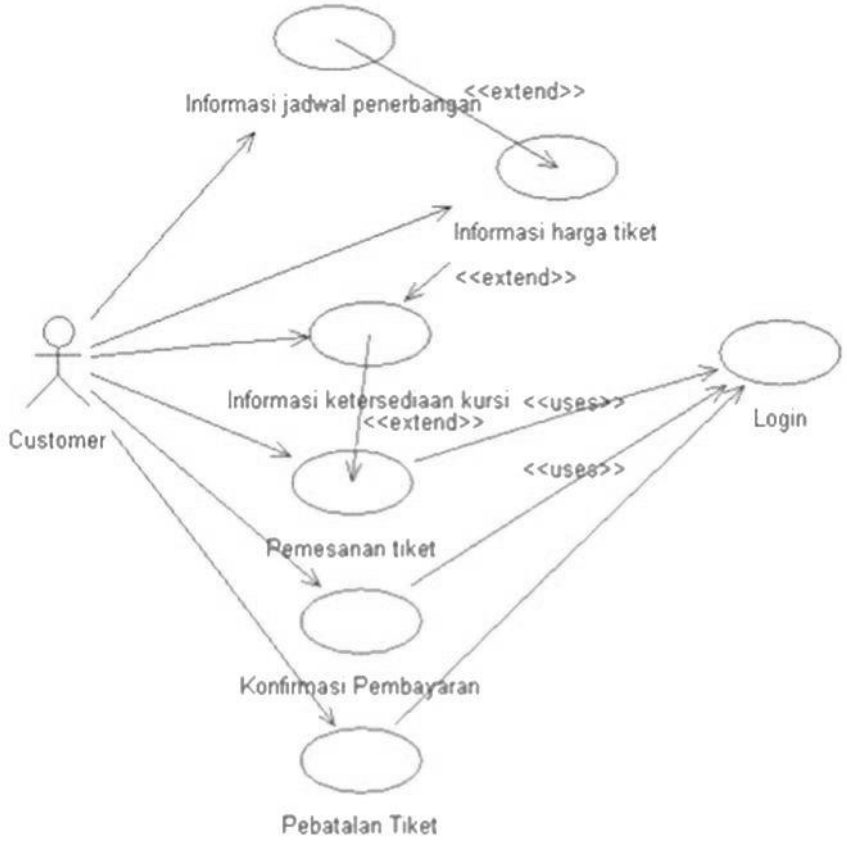
UML menawarkan sebuah standar untuk merancang model sistem. Dengan menggunakan UML kita dapat membuat model untuk semua jenis aplikasi perangkat lunak, sistem operasi apapun serta ditulis dalam bahasa pemrograman apapun. Pemodelan merupakan gambaran dari suatu proses yang dituangkan kedalam pemetaan dengan kaidah-kaidah tertentu.

2.7 Diagram UML

UML menyediakan 10 macam diagram untuk memodelkan aplikasi berorientasi objek[1], dimana pada pembangunan aplikasi ini hanya menggunakan 4 diagram UML yaitu :

a. 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 anatara 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 dilihat pada gambar 2.7 berikut.



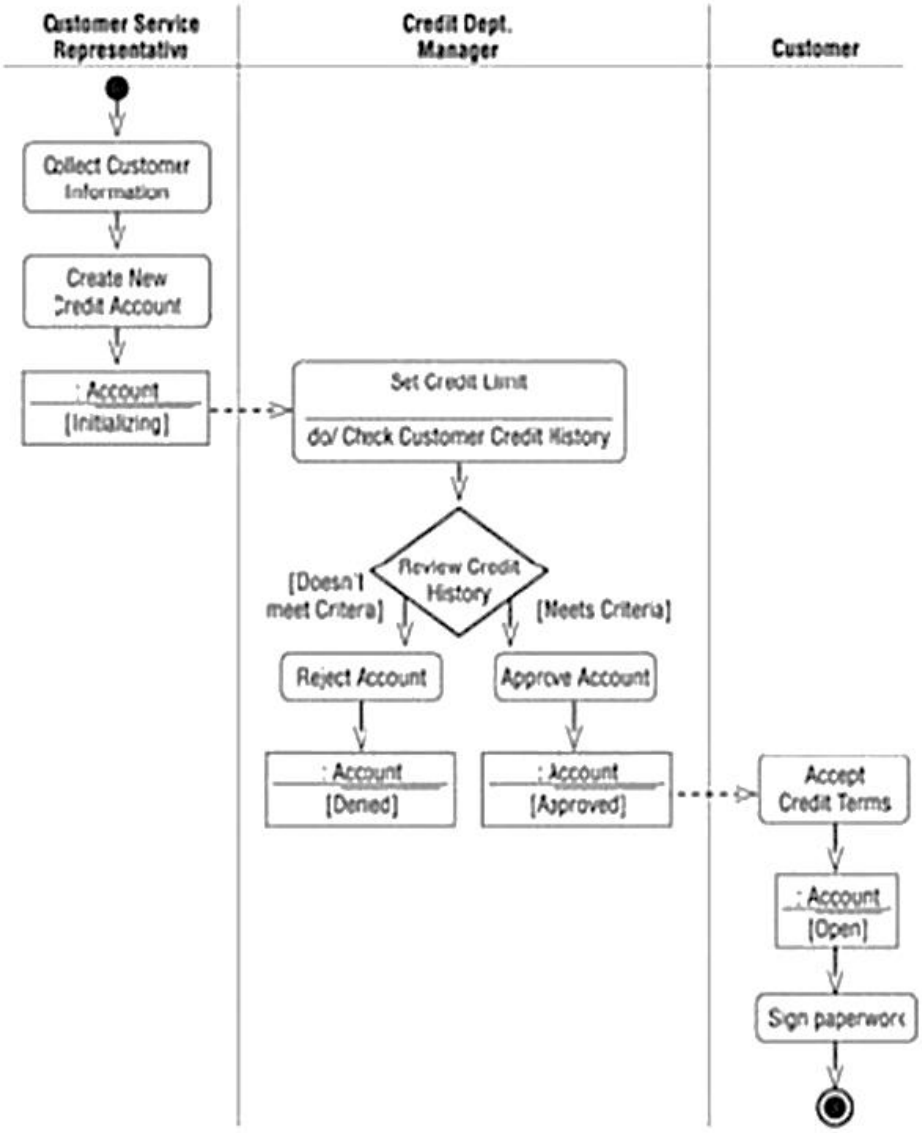
Gambar 2.7. Contoh Diagram Use Case

b. Activity Diagram

Activity diagram 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 diagrami* merupakan *state diagram* khusus, dimana 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 swimline* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu. Contoh *Activity* diagram dapat dilihat pada gambar 2.8 berikut.

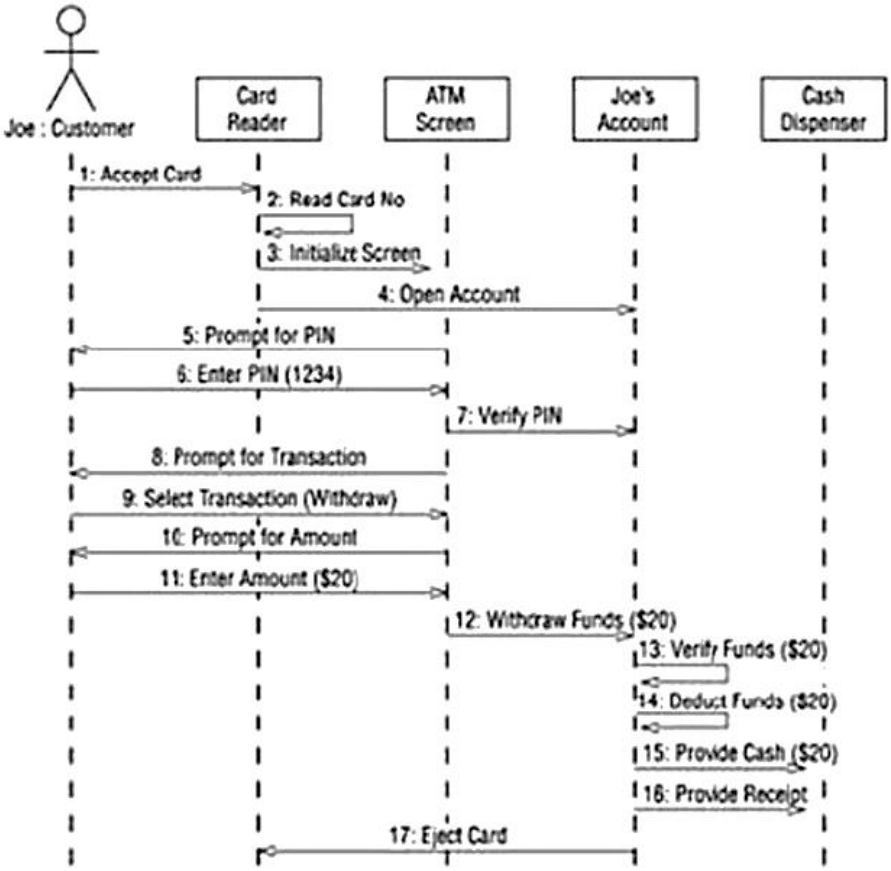


Gambar 2.8. Contoh Diagram Activity

c. Sequence Diagram

Sequence diagram menjelaskan secara detil urutan proses yang

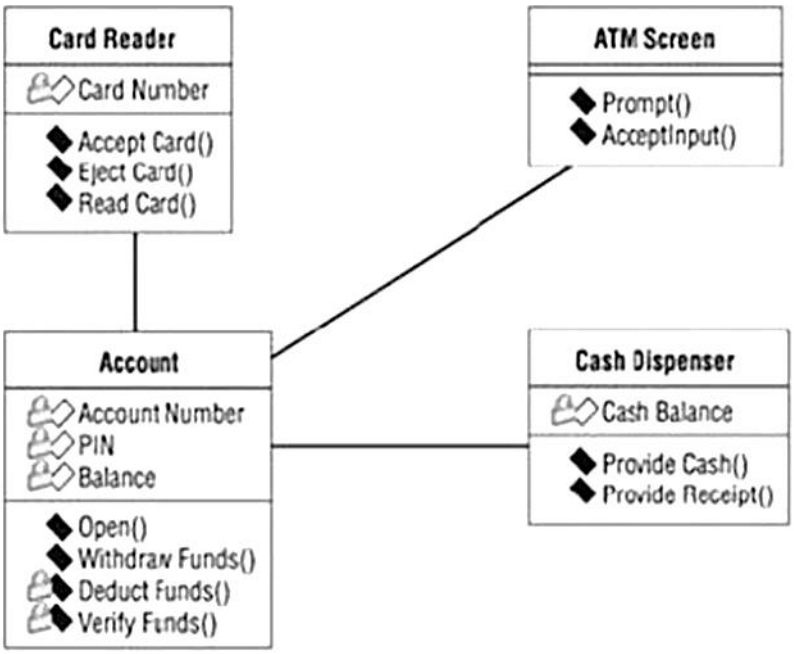
dilakukan dalam sistem untuk mencapai tujuan dari *use case*: interaksi yang terjadi antar *class*, operasi apa saja yang terlibat, urutan antar operasi dan informasi yang diperlukan oleh masing-masing operasi. Contoh *sequence* diagram dapat dilihat pada gambar 2.9 berikut.



Gambar 2.9. Contoh Diagram Sequence

d. Class Diagram

Class diagram merupakan diagram yang selalu ada di permodelan sistem berorientasi objek. *Class* diagram menunjukan hubungan antar *class* dalam sistem yang sedang dibangun dan bagaimana mereka saling berkolaborasi untuk mencapai suatu tujuan. Penggambaran *class diagram* dapat dilihat dalam gambar 2.10 berikut.



Gambar 2.10 Contoh Diagram Class