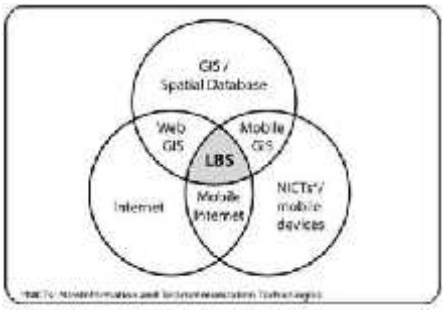
BAB II
TEORI PENUNJANG

2.1 *Location Based Service*

Location Based Service adalah aplikasi yang bergantung pada lokasi tertentu dan didefinisikan pula sebagai layanan informasi dengan memanfaatkan teknologi yang mengetahui posisi sesuatu. Layanan berbasis lokasi menggunakan teknologi *Positioning System*, teknologi ini memungkinkan para pengguna dapat memperoleh informasi lokasi sesuai dengan kebutuhannya [3].

LBS termasuk dalam kategori teknologi yang sama dengan *geographic information system* (GIS), dan aplikasi *global positioning system* (GPS), yaitu dikenal dengan teknologi geospatial. Teknologi ini terdiri atas perangkat untuk mengumpulkan, menyimpan, menganalisa dan mendistribusikan data yang sesuai dengan kebutuhan pengguna tehadap sistem koordinat bumi. Layanan ini menjadi sangat penting bagi penggunanya karena mampu menghubungkan antara lokasi geographic informasi terhadap lokasi penggunanya, hal ini sangat mendukung era mobilitas seperti pada masa ini. Keberadaan aplikasi LBS merupakan hasil penggabungan dari tiga buah teknologi yaitu *New Information and Communication Technologies* (*NICTS*), internet dan GIS dengan menggunakan database spasial. Teknologi LBS ini terdiri atas perangkat-perangkat yang digunakan untuk mengumpulkan, menyimpan, menganalisa dan mendistribusikan data dan informasi pada berdasarkan sistem koordinat geographic bumi secara *real-time.* Identifikasi kordinat pengguna memungkinkan aplikasi LBS untuk menyediakan layanan bagi pengguna perangkat mobile [3].



Gambar 2.1. Location Based Service Sebagai Simpang Tiga Teknologi.

Secara garis besar jenis layanan berbasis lokasi juga dapat dibagi menjadi dua, yaitu:

1. *Pull Service,* Layanan diberikan berdasarkan permintaan dari pelanggan akan kebutuhan suatu informasi. Jenis layanan ini dapat dianalogikan seperti mengakes suatu web pada jaringan internet [3].
2. *Push Service,* Layanan ini diberikan langsung oleh *service provider* tanpa menunggu permintaan dari pelanggan, tentu saja informasi yang diberikan tetap berkaitan dengan kebutuhan pelanggan [3].

2.1.1 Komponen *Location Based Service*

Dalam layanan berbasis lokasi terdapat lima komponen penting yang dibutuhkan untuk menggunakan layanan ini, yaitu:

1. *Mobile Devices,* Suatu perangkat keras yang digunakan oleh pengguna untuk meminta informasi yang dibutuhkan. Informasi dapat diberikan dalam bentuk suara, gambat dan *text*.

2. *Communication Network,* Jaringan komunikasi yang mengirim data pengguna dan informasi yang diminta dari *mobile* terminal ke *service provider* kemudian mengirimkan kembali informasi yang diminta ke pengguna. *Communication Network* dapat berupa jaringan seluler (GSM, CDMA), *Wireless Local Area Network* (WLAN), atau *Wireless Wide Area Network* (WWAN).

3. *Positioning Component,* Untuk memproses suatu layanan maka posisi pengguna harus diketahui apakah itu menggunakan *communication network* atau menggunakan *Global Positioning System* (GPS).

4. *Service and Aplication Povider,* Penyedia layanan menawarkan berbagai macam layanan kepada pengguna dan bertanggung jawab untuk memproses informasi yang diminta oleh pengguna.

5. *Data and Content Provicer,* Penyedia layanan tidak selalu menyimpan semua data yang dibutuhkan yang bisa diakses oleh pengguna. Untuk itu, data dapat diminta dari *data and content provider*.



Gambar 2.2. Komponen Dasar Location Based Service.

2.1.2 Metode Pengambilan Posisi Pada *Location Based Service*

Untuk menggunakan *Location Based service* ini dibutuhkan metode untuk pengambilan posisi pengguna untuk digunakan sebagai acuan sistem untuk memberikan layanan yang dibutuhkan oleh pengguna. Ada beberapa metode pengambilan posisi pengguna dalam *location based service* yaitu :

1. *Terminal-Based Positioning*

Pelacakan pada *terminal-based positioning* ini mengandalkan *global positioning system* (GPS) dan satelit untuk melacak posisi pengguna. Adapun metode, yaitu :

a. ***A-GPS (Assisted-Global Positioning System)***

GPS adalah *satellite based location system*, yang dimana GPS *receiver* mengukur jarak ke satelit GPS menggunakan sinyal spread spectrum. Posisi dapat dihitung apabila posisi dari masing-masing satelit dan jarak satelit di ketahui.

2. *Network-Based Positioning*

Terdapat beberapa metode pada *network base* ini yang dimana pada pelacakan ini menggunakan sinyal dari perangkat seluler dan ditangkap oleh jaringan. Ada beberapa metode dari *Network-Based Posisitioning*, yaitu :

a. ***Cell of Origin***

Metode ini adalah metode yang paling sederhana. Bekerja berdasarkan *cell* jaringan yang ada, di mana masing-masing *cell* djangkau oleh satu BTS (Base Tower Station). Lokasi ditentukan berdasarkan letak dari ponsel

yang berada pada *cell*. Jadi penentuan Lokasi dari ponsel akan terdeteksi dari BTS yang menjangkaunya. Metode ini adalah metode paling sederhana, namun mempunyai tingkat akurasi yang rendah, sebab hanya mengandalkan satu BTS dan ponsel untuk mendapatkan sudut lokasi.

b. ***Time of Arrival***

Metode ini sebagai sinyal elektromagnetik bergerak dengan kecepatan cahaya. mengetahui kecepatan dan perbedaan waktu antara pengiriman dan penerimaan jarak dapat dihitung.

c. ***Angle of Arrival (AOA)***

Metode ini membutuhkan minimal tiga BTS. Penentuan lokasi didapatkan dari sudut (angle) yang dibentuk oleh Ponsel dengan BTS-BTS yang menjangkaunya. Sudut ini biasa disebut Triangulasi (segitiga). Tiga BTS akan membentuk segitiga, dan setiap BTS akan mendeteksi lokasi ponsel berdasarkan garis sudutnya. Metode ini lebih akurat dibanding *Cell of Origin.*

d. ***Time difference of Arrival (TDOA)***

Metode ini berdasarkan pada perbedaan waktu kedatangan. Jadi informasi lokasi dihasilkan dari perpotongan jarak bujur lingkaran yang dibentuk atas perbedaan waktu kedatangan yang dihasilkan. Metode ini memiliki tingkat akurasi yang lebih baik.

2.2 *Assisted – Global Positioning System (A-GPS)*

A-GPS merupakan teknologi yang menggunakan *server* bantuan untuk memotong waktu yang dibutuhkan untuk menentukan lokasi penggunaan GPS, sehingga tidak perlu lagi proses komputasi data waktu dari 3 satelit secara langsung di perangkat *smartphone.* Hal ini berguna di daerah perkotaan atau bahkan di dalam ruangan yang banyak penghalang. Teknologi A-GPS saat ini sudah terpasang pada semua jenis *smartphone* khususnya android. Namun secara tidak sadar teknologi A-GPS masih disebut dengan GPS. A-GPS menjadi lebih umun disebut dengan layanan berbasis lokasi atau *Location Based Service* (LBS).

Server bantuan penyedia data informasi satelit yang dibutuhkan oleh *system* GPS biasanya di *supply* oleh *Network* operator SIM *card* karena sering kali menara jaringan seluler memiliki unit penerima GPS dan unit penerima tersebut secara terus-menerus men-*download* informasi data satelit yang ada di angkasa dan memprosesnya.

Pada sistem A-GPS, telepon genggam akan menangkap sinyal satelit yang lalu dikirimkan ke *server* penyedia layanan telepon, hasil perhitungan lokasi yang dilakukan oleh *server* dikirimkan kembali pada telepon genggam. Peta juga dapat dikirimkan oleh *server* tersebut, atau sudah disimpan pada telepon genggam. Tentunya sistem ini hanya berfungsi bila jaringan telepon genggam mampu dan disediakan bagi pengguna. Oleh karena perhitungan-perhitungan dilakukan oleh *server* penyedia jaringan telepon, maka telepon genggam tidak memerlukan prosesor yang canggih.

Bantuan dari *server* juga memungkinkan telepon genggam untuk mengunci satelit lebih cepat pada saat dinyalakan (TIFF = *Time To First Fix*). Pada beberapa produk telepon genggam, fasilitas A-GPS ini dapat dimatikan dan digantikan oleh alat lain, misalnya *bluetooth* GPS. Sehingga tidak memerlukan transfer data dengan *server* operator seluler [10].



Gambar 2.3. Alur Kerja A-GPS.

2.3 *Google Maps API*

API atau *Application Programming Interface* merupakan suatu dokmentasi yang terdiri dari interface, fungsi, kelas, struktur dan sebagainya untuk membangun sebuah perangkat lunak. Dengan adanya API ini, maka memudahkan programmer untuk "membongkar" suatu software untuk kemudian dapat dikembangkan atau diintegrasikan dengan perangkat lunak yang lain. API dapat dikatakan sebagai penghubung suatu aplikasi dengan aplikasi lainnya yang memungkinkan programmer menggunakan *system function*. Proses ini dikelola melalui *operating system*.

Keunggulan dari API ini adalah memungkinkan suatu aplikasi dengan aplikasi lainnya dapat saling berhubungan dan berinteraksi. Bahasa yang digunakan oleh Google Maps yang terdiri dari HTML, JavaScript dan AJAX serta XML, memungkinkan untuk menampilkan peta Google Maps di website lain. Google juga menyediakan layanan Goole Maps API yang memungkinkan para pengembang untuk mengintegrasikan Google Map ke dalam website masing-masing dengan menambahkan data point sendiri. Dengan menggunakan Google Map API, Google Map dapat ditampilkan pada website eksternal. Agar aplikasi Google Map dapat muncul di website tertentu, diperlukan adanya API key. API key merupakan kode unik yang digenerasikan oleh Google untuk suatu website tertentu, agar server Google Map dapat mengenali [8].

2.4 *Android*

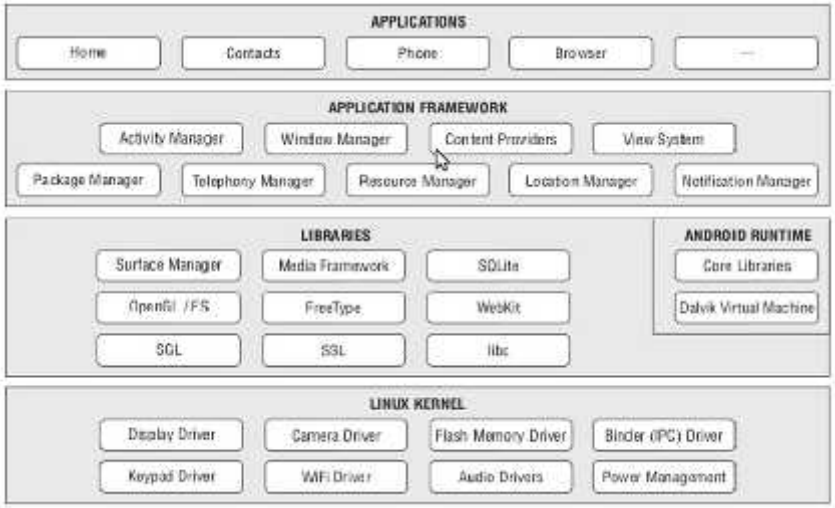
Android merupakan suatu software stack untuk perangkat mobile. Di dalamnya terdapat sistem operasi, middleware, dan key application. Aplikasi pada platform ini dikembangkan dalam bahasa pemrograman java. Android sendiri memiliki banyak fitur, diantaranya [4]:

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 engine* 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. Bluetooth, 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 2.3 di bawah ini.



Gambar 2.4. 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 [6]:

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 [7]:

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 *Unified Modeling Language*

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

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.

2.6.1 *Diagram Unified Modeling Language*

Unified Modeling Language (UML) menyediakan 10 macam diagram untuk memodelkan aplikasi berorientasi objek, yaitu [8]:

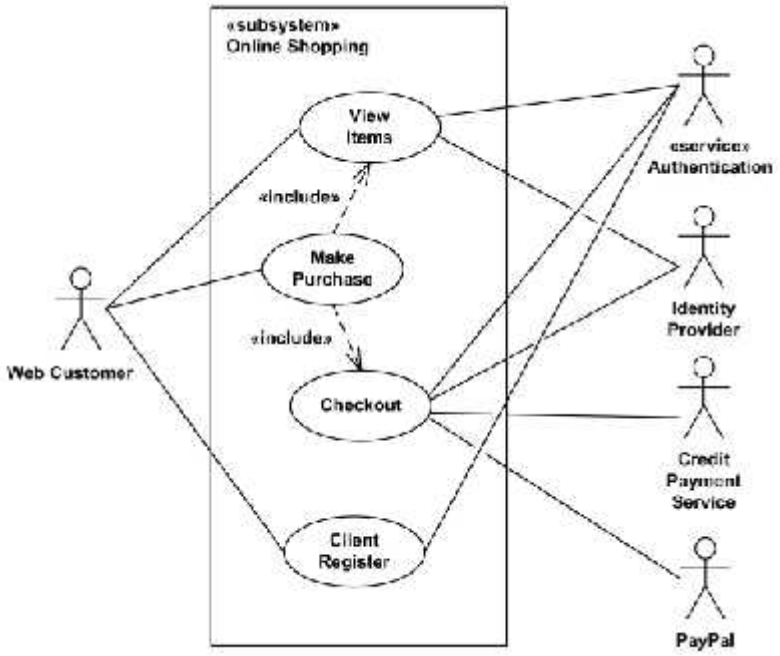
1. *Use Case Diagram,* untuk memodelkan proses bisnis.
2. *Conceptual Diagram,* untuk memodelkan konsep-konsep yang ada di dalam

aplikasi.

3. *Sequence Diagram,* untuk memodelkan pengiriman pesan (*message*) antar objek.
4. *Collaboration Diagram,* untuk memodelkan interaksi antar objek.
5. *State Diagram,* untuk memodelkan perilaku objek di dalam sistem.
6. *Activity Diagram,* untuk memodelkan perilaku *Use Case* dan objek di dalam sistem.
7. *Class Diagram*, untuk memodelkan struktur kelas.
8. *Object Diagram*, untuk memodelkan struktur objek.
9. *Component Diagram,* untuk memodelkan komponen objek.
10. *Deployment Diagram,* untuk memodelkan distribusi aplikasi.

2.6.2 *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 2.4 berikut.



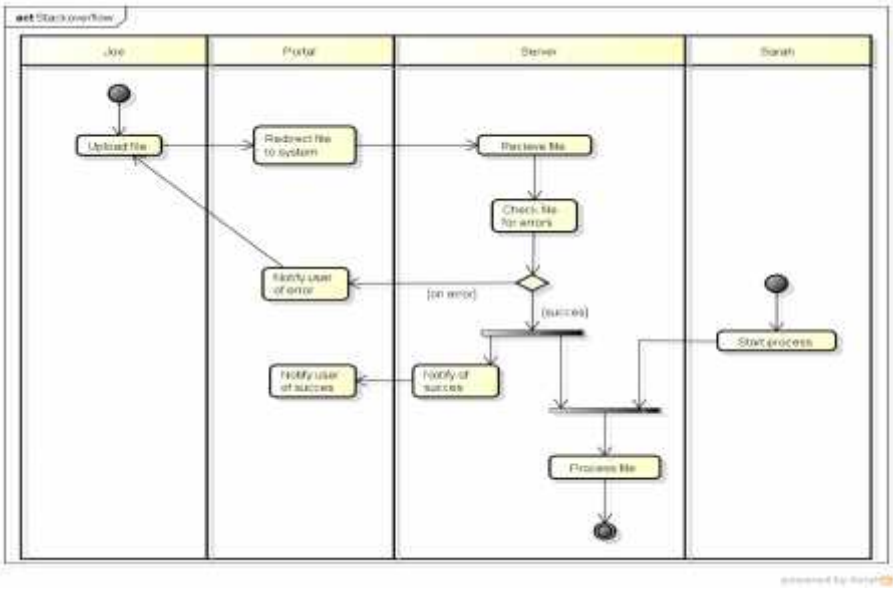
Gambar 2.5. Contoh diagram use case

2.6.3 *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

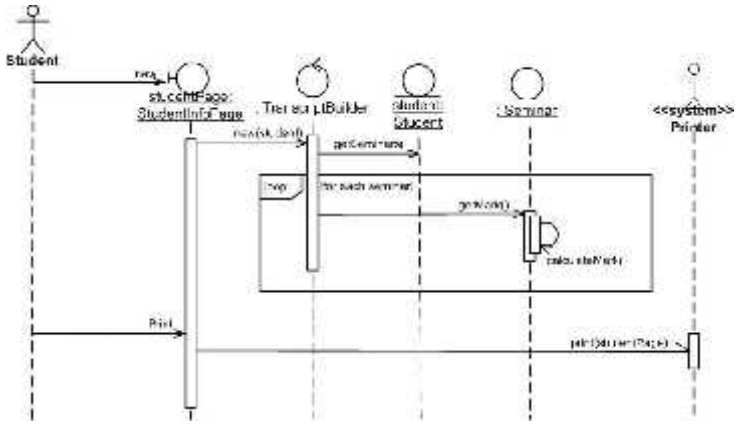
bertanggung jawab untuk aktivitas tertentu. Contoh *Activity* diagram dapat dilihat pada gambar 2.5. berikut.



Gambar 2.6. Contoh diagram activity

2.6.4 *Sequence Diagram*

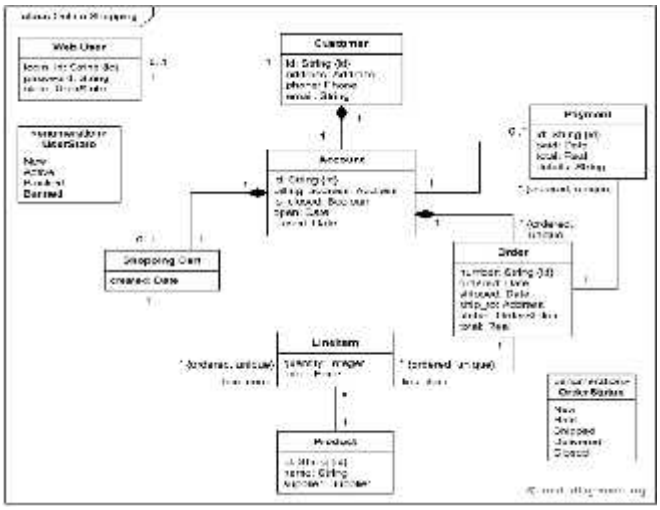
Sequence diagram menjelaskan secara detil urutan proses yang dilakukan dalam sistem untuk mencapai tujuan dari *use case*: interaksi yang terjadi antar *class*, operasi apa saja yang terlibat, urutan antar operasi dan informasi yang diperlukan oleh masing-masing operasi. Contoh *sequence* diagram dapat dilihat pada gambar 2.6 berikut.



Gambar 2.7. Contoh diagram sequence

2.6.5 *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 2.8 berikut.



Gambar 2.8.Contoh class diagram