BAB 2

LANDASAN TEORI

2.1 Teori Umum

2.1.1 Delapan Aturan Emas untuk Perancangan Antarmuka

Delapan aturan emas merupakan salah satu pedoman dalam merancang antarmuka dari sebuah aplikasi. Berikut ini adalah aturan-aturan yang tertera pada delapan aturan emas (Shneiderman et al., 2010):

1. Konsistensi

   Konsisten terhadap urutan tindakan dan rancangan antarmuka sehingga pengguna dapat lebih mudah dan terbiasa menggunakan aplikasi.

2. Kegunaan yang *universal*

   Mengenal kebutuhan dari berbagai pengguna yang berbeda dan mendesain konten yang dinamis sesuai kebutuhan. Memahami perbedaan antara pemula dengan ahli, perbedaan usia, kondisi pengguna, dan perbedaaan teknologi dalam melakukan desain.

3. Memberikan umpan balik yang informatif

   Memberikan umpan balik kepada pengguna ketika melakukan aksi misalnya memberikan kotak dialog peringatan dan suara ketika terjadi kesalahan.

4. Memberikan dialog untuk penutupan

   Urutan aksi harus terorganisir menjadi beberapa bagian seperti bagian awal, bagian tengah dan bagian akhir. Dari bagian tersebut seharusnya

diberikan umpan balik sehingga pengguna dapat mengetahui hasil dari aksi dan melanjutkan ke tahap berikutnya.

5. Adanya penanganan kesalahan

   Sistem yang dibuat sebaiknya dapat mendeteksi kesalahan supaya pengguna tidak melakukan kesalahan yang fatal dan memberikan solusi untuk kesalahan tersebut.

6. Mudah kembali ke tindakan sebelumnya

   Adanya fitur untuk membatalkan tindakan sebelumnya untuk menghilangkan kekhawatiran pengguna ketika mengetahui adanya kesalahan.

7. Mendukung pusat pengendalian internal

   Pengguna ahli biasanya ingin menjadi pengontrol sistem bukan sebaliknya. Sehingga sistem sebaiknya dirancang seakan-akan pengguna adalah pihak memulai aksi bukan perespon.

8. Mengurangi beban ingatan jangka pendek

   Pengguna memiliki keterbatasan kapasitas dalam memproses informasi dan mengingatnya dalam jangka pendek sehingga aplikasi yang dikembangkan sebaiknya tidak membuat pengguna harus mengingat informasi layar-layar sebelumnya.

2.1.2 Lima Faktor Manusia Terukur

Terdapat lima faktor manusia menurut Ben Shneiderman (2010), yaitu:

1. Waktu belajar

   Merancang tampilan yang dapat dipelajari dengan cepat supaya *user* dapat mempelajari dengan cepat.

2. Kecepatan kinerja

   Merancang aplikasi dengan kecepatan yang tinggi sehingga *user* bisa menggunakan aplikasi tersebut dengan lancar.

3. Tingkat kesalahan

   Seberapa banyak kesalahan yang dilakukan oleh *user* dan kesalahan-kesalahan seperti apa yang bias terjadi saat *user* dalam menggunakan aplikasi tersebut.

4. Daya ingat

   Kemampuan *user* mempertahankan pengetahuannya setelah jangka waktu tertentu.

5. Kepuasan subjektif

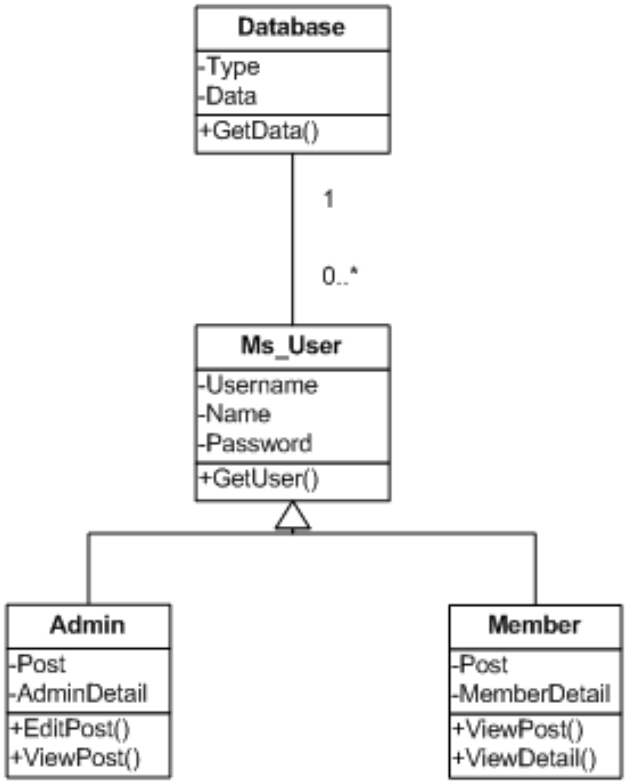
   Kepuasan *user* terhadap berbagai aspek sistem yang dicapai dengan melakukan survei kepada beberapa *user* untuk mendapatkan tingkat kepuasan.

2.1.3 *Unified Modeling Language* (UML)

Menurut Whitten dan Bentley(2007), UML (*Unified Modeling Language*) merupakan kumpulan konvensi pemodelan untuk menggambarkan fungsi-fungsi yang disediakan oleh sistem tersebut.

2.1.3.1 *Class Diagram*

Class diagram adalah sebuah diagram yang menggambarkan tentang objek-objek bernama kelas yang menyusun suatu sistem dan juga hubungan antar kelas-kelas tersebut (Whitten & Bentley., 2007).



Gambar 2.1 Contoh *Class Diagram*

Terdapat tiga bagian pada suatu *class*, yakni :

1. Nama kelas

   Nama kelas terdapat di bagian pertama *class* yang merepresentasikan nama dari *class* tersebut.

2. Atribut kelas

   Atribut kelas terdapat di bagian tengah *class* yang merepresentasikan tipe data yang dimiliki.

3. Operasi

Operasi terdapat di bagian bawah *class* yang merepresentasikan kegiatan-kegiatan yang akan dilakukan oleh *class* tersebut.

Adapun notasi-notasi yang terdapat pada *class* diagram, seperti :

1. *Visibility*

Visibility digunakan untuk menentukan apakah atribut atau operasi dari sebuah kelas dapat digunakan oleh kelas yang lain.

Visibility	**Simbol**	**Deskripsi**
Private	-	Hanya dapat hanya dapat digunakan oleh kelas yang mendefinisikan
Protected	#	Dapat digunakan oleh kelas yang mendefinisikannya dan turunan dari kelas tersebut
Public	+	Dapat digunakan oleh semua kelas yang berhubungan

Tabel 2.1 Deskripsi Simbol *Visibility*

2. *Multiplicity*

Multiplicity digunakan untuk menentukan banyaknya kelas yang berhubungan dengan kelas yang dimaksud.

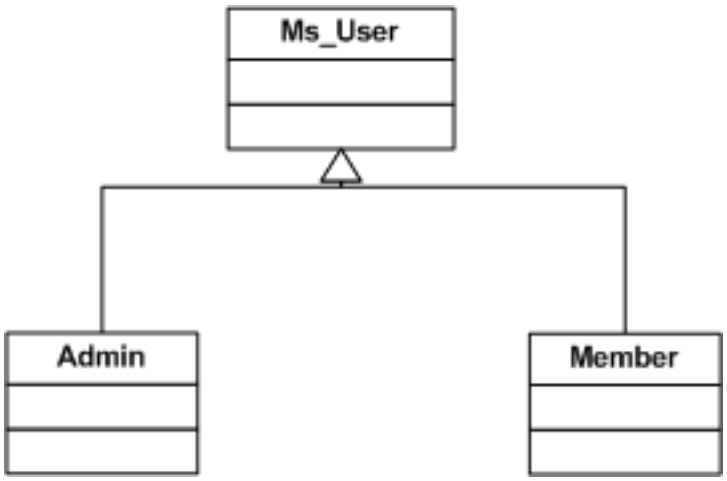
Multiplicity	**Deskripsi**
0…1	Nol atau satu
1	Satu
0…*	Nol atau lebih
1…*	Satu atau lebih

Tabel 2.2 Deskripsi Simbol *Multiplicity*

3. Generalisasi

Generalisasi digunakan untuk merepresentasikan *inheritance* dalam *class* diagram dengan menggambarkan hubungannya melalui *superclass* dan *subclass*. *Superclass* adalah bentuk umum dari sebuah

subclass, dan *subclass* adalah bentuk spesifik dari sebuah *superclass*. *Subclass* memiliki semua properti yang dimiliki oleh *superclass*.



Gambar 2.2 Contoh Notasi Pada *Class* Diagram

4. Asosiasi

Asosiasi digunakan untuk menggambarkan adanya hubungan antar kelas dengan memberikan keterangan relasi dan *multiplicity.* Hubungan asosiasi dibedakan menjadi dua jenis, yaitu:

a. *Uni-directional*

Hubungan *Uni-directional* merupakan hubungan dimana dua kelas saling terhubung tetapi hanya salah satu yang memiliki peran dalam hubungan tesebut.

b. *Bi-directional*

Hubungan *Bi-directional* merupakan hubungan dimana dua kelas saling terhubung dan keduanya memiliki peran dalam hubungan tesebut.

5. Agregasi

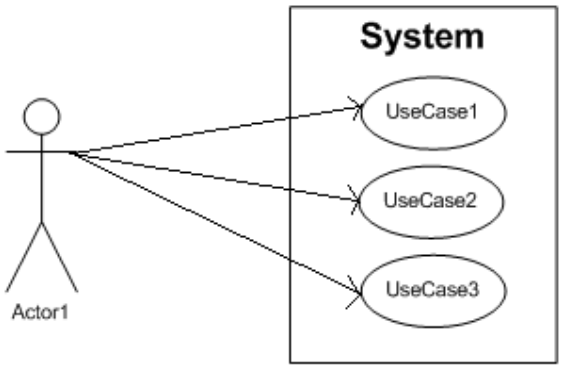
   Agregasi digunakan untuk menggambarkan hubungan dimana sebuah kelas merupakan bagian dari kelas lain, tetapi bukan merepresentasikan *inheritance.*

6. Komposisi

   Komposisi merupakan bentuk lain dari agregasi tetapi hubungan antar kelas komposisi lebih erat dibandingkan agregasi karena kelas A yang merupakan bagian dari kelas B tidak akan ada jika kelas B tidak ada.

2.1.3.2 *Use Case Diagram*

Use case diagram adalah sebuah diagram yang menggambarkan interaksi antara sistem dengan pengguna. Jadi, *use case* diagram menggambarkan siapa yang akan berinteraksi atau menggunakan sistem tersebut (Whitten & Bentley., 2007).



Gambar 2.3 Contoh *Use Case* Diagram

Use case diagram dideskripsikan oleh *use case narrative* yang menggambarkan bagaimana *user* berinteraksi dengan untuk menyelesaikan pekerjaannya.

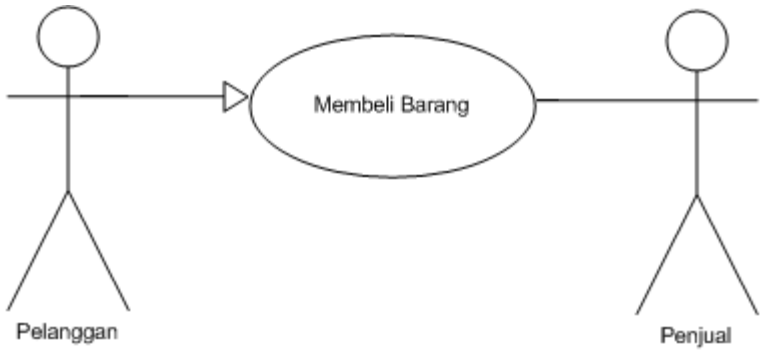
Elemen	Keterangan
Nama *use case*	Nama *use case* harus merepresentasikan tujuan yang ingin dicapai oleh *use case* tersebut dan diawali dengan kata kerja.
Aktor	Aktor yang menjalankan *use case*.
Event yang dijalankan	Langkah-langkah yang harus dilakukan untuk menyelesaikan
Precondition	Kondisi yang harus dipenuhi sebelum menjalankan *use case.*
Postcondition	Hasil yang dicapai setelah *use case* berhasil dijalankan.

Tabel 2.3 Deskripsi *Use Case Narrative*

Hubungan-hubungan pada *use case diagram*:

a. Asosiasi

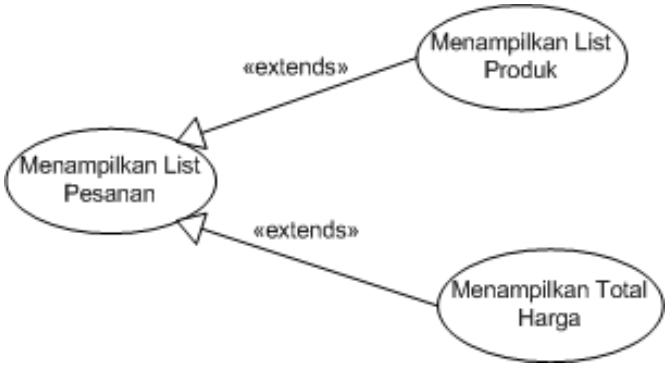
Asosiasi menggambarkan adanya interaksi antara *use case* dengan aktor. Asosiasi digambarkan dengan simbol garis yang bisa mempunyai anak panah atau tanpa anak panah. Asosiasi dengan anak panah yang berasal dari aktor menunjuk ke *use case* menandakan bahwa aktor inisiator *use case* tersebut. Asosiasi tanpa anak panah menandakan interaksi antara *use case* dengan aktor penerima.



Gambar 2.4 Contoh Asosiasi pada *Use Case* Diagram

b. *Extends*

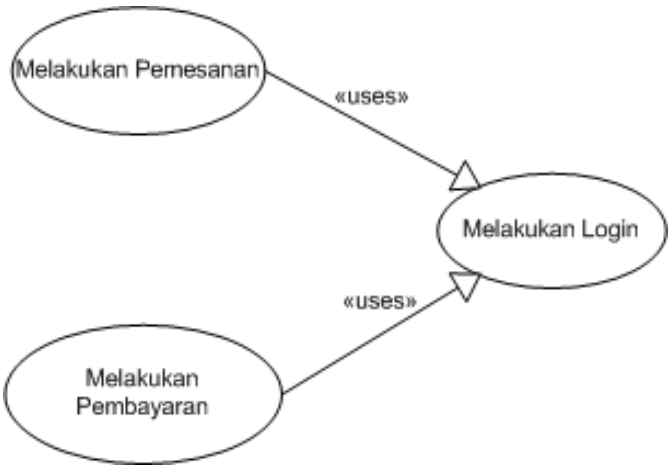
Extends digunakan ketika fungsi *use case* terdiri dari beberapa tahap yang sulit sehingga mudah dimengerti. *Extends* akan menghasilkan *use case* baru yang mewakili fungsi dari *use case* awal.



Gambar 2.5 Contoh *Extends* pada *Use Case* Diagram

c. *Includes/uses*

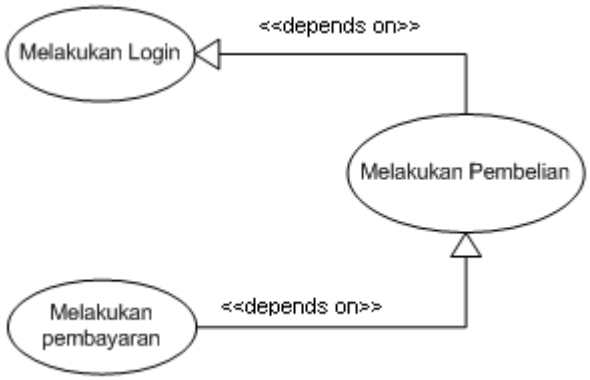
Includes digunakan ketika *use case* melakukan beberapa langkah yang sama yang menjadi sebuah entitas baru dengan sebutan *use case* abstrak sehingga mengurangi redundansi dalam *use case* diagram.



Gambar 2.6 Contoh *Includes* pada *Use Case* Diagram

d. *Depends on*

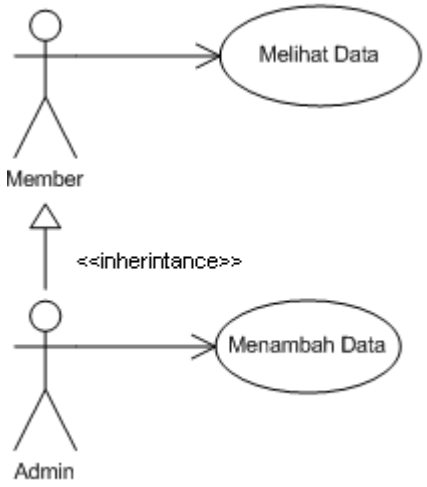
Depends On digunakan untuk menggambarkan hubungan *use case* dimana suatu *use case* tidak dapat dilakukan apabila *use case* yang lain belum dilakukan.



Gambar 2.7 Contoh *Depends on* pada *Use Case* Diagram

e. *Inheritance*

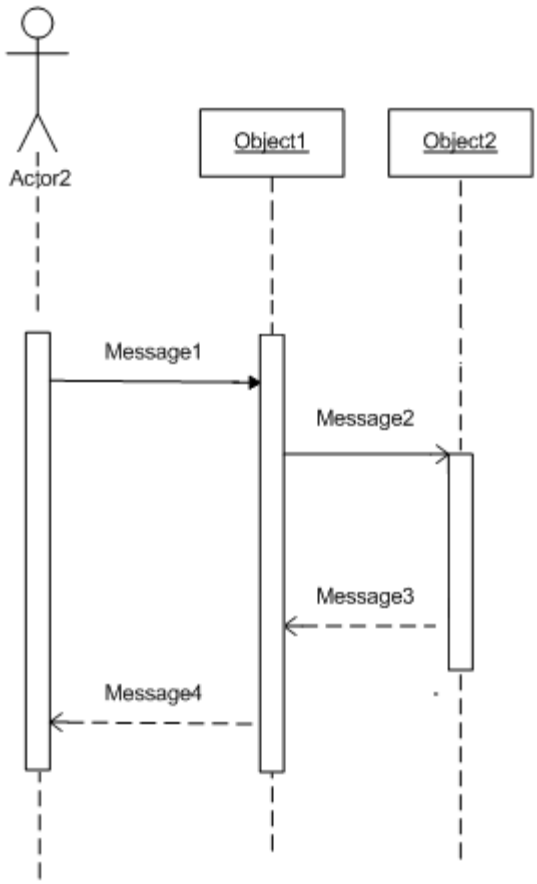
Inheritance digunakan untuk menggambarkan hubungan dua atau lebih aktor dalam satu sistem dengan sebuah aktor abstrak untuk menyederhanakan diagram karena aktor abstrak memiliki langkah yang dimiliki aktor lain.



Gambar 2.8 Contoh *Inheritance* Pada *Use Case Diagram*

2.1.3.3 *Sequence Diagram*

Sequence diagram menggambarkan model logika dari sebuah *use case* dengan pesan yang dikirimkan antar objek dalam waktu tertentu (Whitten & Bentley., 2007).



Gambar 2.9 Contoh *Sequence Diagram*

Berikut elemen-elemen pada *sequence* diagram :

1. Aktor

   Aktor merupakan simbol yang digunakan untuk mewakili pengguna dalam berinteraksi dengan objek kelas *interface*.

2. Objek

   Objek merupakan simbol yang digunakan untuk mewakili kelas-kelas pada *class* diagram.

3. Aktivasi

   Aktivasi merupakan simbol yang digunakan untuk menggambarkan lamanya waktu dari objek saat digunakan.

4. Pesan

   Pesan merupakan simbol yang digunakan untuk menyampaikan *method* dari setiap objek yang ada.

5. *Self-call*

   Self-call merupakan simbol yang digunakan untuk menunjukkan bahwa sebuah objek mengirimkan pesan kepada objek itu sendiri.

6. *Return*

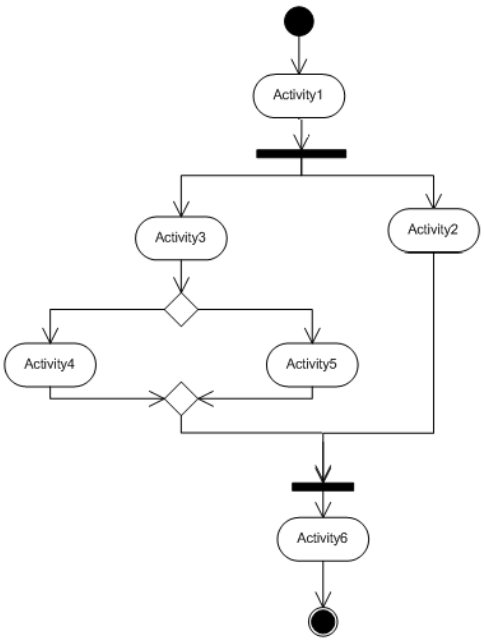
   Return merupakan simbol yang digunakan untuk menandakan respon dari pesan yang disampaikan oleh suatu objek.

7. *Frame*

   Frame merupakan simbol yang digunakan untuk menandakan area yang mengalami perulangan, seleksi, atau ketentuan.

2.1.3.4 *Activity Diagram*

Activity diagram merupakan gambaran mengenai alur sebuah proses bisnis, langkah dalam sebuah *use case*, dan logika sebuah tindakan objek (Whitten & Bentley, 2007).



Gambar 2.10 Contoh *Activity Diagram*

Elemen-elemen pada *activity* diagram:

1. *Initial node*

   Initial node merupakan simbol yang merepresentasikan mulainya

2. *Action*

   Action merupakan simbol yang merepresentasikan suatu aktivitas yang digambarkan sesuai urutan *actions*.

3. *Flow*

   Flow merupakan simbol yang merepresentasikan jalur dari satu aktivitas ke aktivitas lain

4. *Decision/Merge*

   Decision/Merge merupakan simbol yang merepresentasikan keadaan kondisional. *Decision* mengindikasikan kondisi untuk memilih salah satu alur aktivitas sedangkan *merge* mengindikasikan bergabungnya alur aktivitas yang dipisahkan oleh *decision* sebelumnya.

5. *Fork/ Join*

   Fork merupakan simbol yang mengindikasikan adanya dua atau lebih *actions* yang berlangsung secara bersamaan sedangkan *join* mengindikasikan berakhirnya sebuah proses paralel.

6. *Activity final*

   Activity final merupakan simbol yang merepresentasikan akhir sebuah proses.

2.1.4 XP (*Extreme Programming*)

Extreme Programming merupakan salah satu jenis metode pengembangan perangkat lunak dari *Agile Software Development*. Metode XP mengutamakan kepentingan pengguna pada setiap tahap pengembangannya.

Tahap-tahap yang digunakan dalam XP (Shneiderman et al., 2010):

1. *Planning*

   Pada tahap ini dilakukan penyusunan *stories* yang menggambarkan fitur dan fungsi yang dibutuhkan. Masing-masing *story* diberi nilai dan diurutkan berdasarkan prioritasnya. Kemudian *developer* akan menentukan waktu yang diperlukan untuk setiap *story* tersebut. *Story* yang membutuhkan waktu yang lebih lama bisa dipecahkan menjadi beberapa *story* lagi.

2. *Design*

   Desain dalam XP menggunakan prinsip KISS (*Keep It Simple, Stupid!*). Konsep tersebut akan memberikan pedoman bagaimana implementasi dari setiap *story* yang ada. Dalam XP, desain dapat dilakukan sebelum dan sesudah tahap *coding* karena adanya *refactoring* yang dilakukan setelah tahap *coding* dilaksanakan.

3. *Coding*

   Tahap ini diawali dengan merancang *unit test* yang akan melakukan evaluasi setiap *story*. Pengembang akan menganalisis cara mengimplementasikan *story* tersebut.

4. *Testing*

   Pada tahap *testing* dilakukan pengujian kode dengan *unit test* yang disediakan.

Kelebihan XP dibandingkan metode Waterfall dan Scrum (Shneiderman et al., 2010) adalah:

1. Pada Waterfall, jika suatu tahap sudah selesai dan terjadi perubahan maka tidak bisa tahap tersebut. Perubahan hanya bisa dilakukan dengan memulai proses kembali ke awal. Sedangkan XP mengakomodasi perubahan tahap harus mengulang keseluruhan proses.
2. Scrum mempunyai proses perulangan *sprint* yang tidak mengijinkan adanya perubahan selama proses *sprint* berlangsung. Sedangkan XP terdapat proses perulangan yang mengijinkan perubahan selama perubahan tersebut tidak mengganggu proses lain.

2.2 Teori Khusus

2.2.1 LBS (*Location Based Services*)

LBS berarti penyedia jasa *mobile* menyediakan informasi dengan konteks yang terkait kepada pengguna berdasarkan lokasi dan preferensi mereka (Dhar & Varshney, 2011). LBS pada umumnya mencakup iklan berbasis lokasi, navigasi, dan layanan *location check-in*. Dengan memanfaatkan LBS, aplikasi pada *smartphone* dapat mengetahui lokasi penggunanya.

Meskipun memberi manfaat yang besar bagi penggunanya, LBS perlu memanfaatkan informasi tentang lokasi pengguna. Hal ini dapat memicu perhatian pengguna atas masalah privasi karena mereka mereka diawasi oleh penyedia jasa. Perhatian pengguna terhadap resiko privasi dapat berdampak negatif terhadap penggunaan LBS (Tao Zhou, 2012).

2.2.2 *Instant Messaging*

Instant messaging merupakan sebuah teknologi komunikasi *mobile* yang sangat populer, dan *real-time* (Lin & Chiu, 2010). Sistem *instant messaging* terdiri dari beberapa fitur atau komponen, antara lain identitas pengguna, profil pengguna, *database* pengguna, *presence awareness*, pesan instan, *asynchronous chat*, ukuran pesan, kemudahan penggunaan, *multi-user chat*, dan keamanan (Kadirire, 2007). Semua komponen tersebut telah tersedia pada hampir semua aplikasi *instant messaging* yang ada sekarang dengan berbagai tambahan fitur.

2.2.3 Android

Android adalah sistem operasi yang berbasis *kernel* linux dengan rancangan khusus oleh Google dan bersifat *open source*. Google memberikan akses yang luas kepada pada *developer* untuk menggunakan *tools* dan *library* dalam pengembangan aplikasi-aplikasi Android. Pada awalnya, Android dikembangkan oleh perusahaan Android Inc. namun perusahaan tersebut dibeli oleh Google pada tahun 2005 (Conder & Darcey, 2011).

Perkembangan Android ditentukan oleh sebuah konsorsium yang bernama *Open Handset Alliance* (OHA) pada bulan November 2009. OHA adalah Aliansi perusahaan-perusahaan *software, hardware* dan telekomunikasi seperti Intel, Nvidia, Google, Samsung, Sprint, T-Mobile, Motorola, serta beberapa anggota lainnya. Aliansi tersebut bertujuan untuk memberikan kontribusi yang lebih besar sehingga Android dapat berkembang lebih baik (Conder & Darcey, 2011).

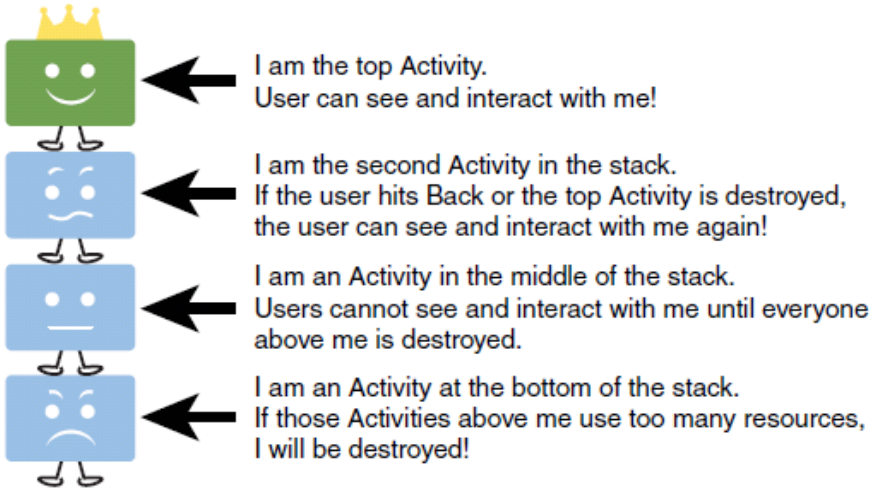
2.2.3.1 Komponen Aplikasi

Beberapa komponen aplikasi pada Android yang dibutuhhkan oleh *developer* ketika mengembangkan aplikasi Android (Conder & Darcey, 2011):

1. *Activities*

   Activities merupakan layar-layar yang memiliki tugas masing-masing sehingga membentuk sebuah aplikasi Android. Sistem operasi

Android akan melacak semua objek *Activity* yang berjalan dengan menempatkan semua *Acitivity* pada sebuah tumpukan. Ketika *Activity* baru dijalankan, *Activity* di atas tumpukan akan berhenti dan *Activity* yang baru akan berpindah ke atas tumpukan. Ketika *Activity* selesai, *Activity* akan dihapus dari tumpukan dan *Acitivity* akan dijalankan kembali dalam tumpukan tersebut.



Gambar 2.11 Proses *Activity* pada aplikasi android (Conder & Darcey, 2011)

2. *Intents*

Intents merupakan mekanisme pesan *asynchronous* yang digunakan oleh sistem operasi Android untuk mencocokkan *task requests* dengan *Activity* atau *Services* yang sesuai dengan mengirimkan *event broadcast Intents* untuk sistem.

3. *Content Providers*

Content providers mengatur data pada aplikasi supaya data aplikasi internal dapat diekspos untuk aplikasi lain.

Provider	**Fungsi**
MediaStore	Data audio-visual di telepon dan penyimpan internal
CallLog	Menerima dan mengirim panggilan
ContactsContract	*Database* kontak telepon
UserDictionary	Kamus untuk memprediksi inputan *user*

Tabel 2.4 Contoh *Content Providers*

4. *Broadcast Recievers*

   Broadcast Recievers merupakan komponen yang menerima *request* dari sebuah *event* dan meneruskan *request* tersebut kepada sistem.

2.2.3.2 Versi Android

Berikut versi-versi Android yang telah dikembangkan hingga saat ini:

1. Android versi 1.1

   Android versi 1.1 dirilis pada bulan Februari 2009 yang merupakan versi pertama,

2. Android versi 1.5 (Cupcake)

   Android versi 1.5 dirilis pada bulan April 2009 dengan penambahan fitur berikut:

   - Fitur untuk merekam dan menonton video dengan modus kamera.
   - Fitur untuk menggunggah video ke Youtube® dan gambar Pisaca® langsung dari telepon.
   - Kemampuan terhubung secara otomatis ke headset *Bluetooth* dalam jarak tertentu.

- *Widgets* dan folder yang baru yang dapat ditambahkan ke dalam layar utama.

3. Android versi 1.6 (Donut)

   Android versi 1.6 dirilis pada bulan September 2009 dengan pembaharuan berikut:

   - Pembaharuan *Voice search*.
   - Teknologi yang mendukung *Code Division Multiple Access/ Evolution Data Only* (CDMA/ EVDO), 802.1x, *Virtual Private Network* (VPN), *text-to-speech engine* serta kemampuan *dial contact*.
   - Dukungan resolusi layar *wide video Graphics Array.*

4. Android versi 2.0/2.1 (Eclair)

   Android versi 2.0 dirilis pada bulan Oktober 2009 dengan pembaharuan berikut:

   - Pengoptimalan kecepatan perangkat keras.
   - Perubahan *interface* dengan browser baru dan dukungan HTML5.
   - Dukungan *flash* untuk kamera.
   - Peningkatan Google Maps 3.1.2.

5. Android versi 2.2 (Froyo)

   Android versi 2.2 dirilis pada bulan Mei 2010 dengan pembaharuan berikut:

   - Pembaharuan kecepatan aplikasi tambahan dalam implementasi *Just In Time*.

- Peningkatan dukungan *Microsoft Exchange* seperti keamanan, *auto discovery*, dan sinkronisasi kalender.
- Pembaharuan USB *tehtering* dan *product hotspot*.
- Pengungguhan *file* pada aplikasi *browser*.

6. Android versi 2.3 (Gingerbread)

   Android versi 2.3 dirilis pada Bulan Desember 2010 dengan pembaharuan berikut:

   - Dukungan tingkat resolusi yang lebih besar.
   - Perubahan tampilan antarmuka.
   - Pembatasan spefisikasi minimum agar bisa di-*upgrade* ke versi 2.3.

7. Android versi 3.0/3.1 (Honeycomb)

   Pembaharuan yang dilakukan pada Android versi 3.0 adalah sebagai berikut:

   - Pembaharuan desain yang disesuaikan dengan tablet.
   - Mendukung *multi* prosesor.

8. Android versi 4.0 (Ice Cream Sandwich)

   Android versi 4.0 dirilis pada Bulan Oktober 2011 dengan pembaharuan berikut:

   - Penambahan perangkat fotografi.
   - Penambahan fitur pengenalan wajah untuk membuka kunci.
   - Berbagi informasi dengan NFC.

2.2.4 *Web Service*

Web service adalah sebuah aplikasi perangkat lunak yang memungkinkan komputasi terdistribusi dengan basis XML dan tiga teknologi inti seperti WSDL, SOAP, dan UDDI (Mohanty et al, 2012).

2.2.4.1 WSDL (*Web Services Description Language*)

WSDL (*Web Services Description Language*) adalah sebuah XML-*based language* yang mendeskripsikan *service request* dengan menggunakan protokol-protokol yang berbeda dan juga encoding (Sabraoui et al, 2012).

2.2.4.2 SOAP (*Simple Object Access Protocol*)

SOAP (*Simple Object Access Protocol*) adalah sebuah protokol untuk pertukaran informasi antar komputer dalam jaringan yang dapat digunakan pada beragam *messaging system*, dan dapat dikirim menggunakan beragam *transport protocol* (Sabraoui et al, 2012).

2.2.4.3 UDDI (*Universal Description, Discovery and Integration*)

UDDI (*Universal Description, Discovery and Integration*) adalah sebuah *service registry* bagi pengalokasian *web service* yang mengombinasikan SOAP dan WSDL dalam membentuk registry API bagi pendaftaran dan pengenalan *service* (Sabraoui et al, 2012).

2.2.5 JSON (*JavaScript Object Notation*)

JSON (*JavaScript Object Notation*) merupakan format pertukaran data yang ringan, mudah dibaca dan ditulis oleh *developer*. JSON mudah diterjemahkan dan dibuat oleh komputer karena menggunakan format teks yang tidak bergantung pada bahasa pemrograman apapun sehingga JSON ideal sebagai bahasa pertukaran-data (Anonim1). Sehingga JSON bisa digunakan sebagai bahasa pertukaran data antara *web-service* dan Android.

2.2.6 PHP (*Hypertext Preprocessor*)

PHP adalah sebuah bahasa *server-side scripting* yang bersifat *open source* yang banyak digunakan untuk pemograma *website* dan umumnya diintegrasikan dengan HTML (*Hypertext Markup Language*) untuk menghasilkan sebuah *interface* yang menarik (Achour, 2009).

File PHP ditandai dengan *extension file* .php yang harus diterjemahkan oleh *web-server* menjadi bahasa HTML untuk diteruskan ke *browser*. Pemograman PHP dapat berdiri sendiri atau dapat juga diintegrasikan dengan HTML. Kode PHP ditulis dalam *tag* PHP (<?php ?>) dimana *tag* tersebut menandakan *block* program dari PHP seandainya program PHP diintegrasikan dengan kode HTML.

2.2.7 MySQL

MySQL adalah sebuah perangkat lunak sistem manajemen basis data SQL atau DBMS yang *multithread* dan *multiuser* dan bersifat RDBMS

yang terdistribusikan secara gratis di bawah lisensi GPL (*General Public Lisence*) sehingga MySQL dapat digunakan secara gratis (Welling & Thomson, 2009).

2.2.8 XMPP (*Extensible Messaging and Presence Protocol*)

2.2.8.1 Pengertian XMPP

XMPP merupakan sebuah protokol yang mendukung komunikasi secara *real-time* untuk berbagai aplikasi, termasuk *instant messaging*, *multi-user chat*, konferensi suara dan video, *real-time game,* sinkronisasi data, dan pencarian menggunakan XML (Moffitt, 2010).

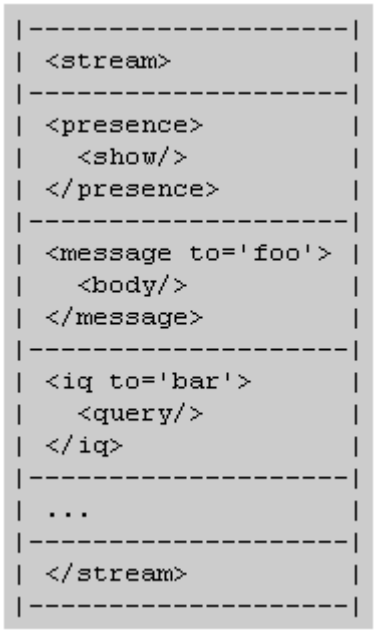
Kelebihan menggunakan XMPP (Saint-Andre et al., 2009):

1. Lebih dari 10 tahun pengembangan, XMPP terbukti menghasilkan banyak aplikasi yang stabil secara luas dan teruji.
2. XMPP aman karena menyediakan dukungan *built-in* untuk proses enkripsi dan *authentication* yang kuat.
3. Teknologi XMPP tersebar dalam arsitektur *client-server* terdesentralisasi dengan jumlah *server* yang tak terbatas. Setiap orang atau organisasi dapat menjalankan server sendiri XMPP dan menghubungkannya ke seluruh jaringan menggunakan infrastruktur *internet* standar seperti *Domain Name System* (DNS), dan sertifikat yang tersedia secara bebas melalui XMPP *Standards Foundation* (XSF) untuk mengaktifkan federasi aman dari lalu lintas XMPP.

4. XMPP sangat *extensible* karena XMPP memberikan layanan yang cepat dari satu tempat ke tempat lain dengan XML. XMPP telah digunakan untuk berbagai aplikasi melampaui instant messaging, termasuk *game*, jejaring sosial, *Voice over IP* (VoIP), kolaborasi *real-time*, peringatan dan pemberitahuan, sindikasi data dan lain lain.
5. XMPP sudah mempunyai standarisasi yang ditinjau dalam Internet *Engineering Task Force* (IETF) dan ekstensi XMPP dipublikasikan untuk umum. Pendekatan ini telah menghasilkan teknologi yang kuat yang dapat diimplementasikan secara bebas di setiap lisensi.
6. XMPP adalah sebuah komunitas yang bagus dan terbuka untuk sejumlah besar produk perangkat lunak dan jaringan komunikasi.

2.2.8.2 Stanza

Server dan *client* XMPP berkomunikasi melalui pengiriman *stanza*. *Stanza* adalah elemen XML yang terbentuk dengan *complete* dan fleksibel (sesuai spesifikasi). XML *stanza* didefinisikan oleh tiga elemen, yakni <message/>, <presence/>, dan <iq/>.



Gambar 2.12 Contoh *Stanza* (Saint-Andre)

a. ***Message Semantics***

Elemen <message/> merupakan mekanisme *push* dimana sebuah entitas mendorong informasi kepada entitas lain, mirip dengan komunikasi yang terjadi dalam sistem seperti *email*. Semua <message/> harus memiliki sebuah atribut “to” untuk menentukan siapa penerima pesan.

b. ***Presence Semantics***

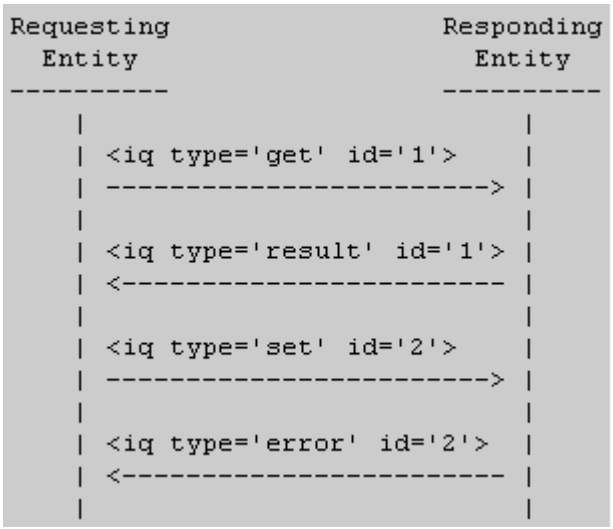
Elemen <presence/> merupakan mekasnisme suatu *broadcast* atau *publish-subscribe* dimana beberapa entitas akan menerima informasi suatu entitas yang mereka di-*subscribe.* Informasi suatu entitas disampaikan melalui elemen <show/>.

Secara *default*, elemen <show/> memiliki empat jenis tipe, antara lain:

1. *Chat* : Menunjukkan bahwa kontak tersedia dan siap untuk *chat*.
2. DND : Menunjukkan "*Do Not Disturb*" yang mengartikan bahwa kontak sedang sibuk untuk *chatting*.
3. *Away* : Menunjukkan bahwa kontak *idle* dan mungkin tidak berada di tempat.
4. Xa : Menunjukkan "*Extended Away*" yang menunjukkan bahwa kontak telah diam selama durasi yang cukup lama.

c. *IQ Semantics*

IQ (*Info* / *Query*) merupakan mekanisme *request-response*. Elemen <iq/> memungkinkan suatu entitas untuk membuat *request* dan menerima *response* dari entitas lain. Interaksi *request*-response dilacak oleh entitas yang melakukan tindakan *request* melalui penggunaan atribut "*id*". Dengan demikian, interaksi *IQ* mengikuti pola umum dari pertukaran data terstruktur seperti "*get*", "*result*", "*set*", "*error*".

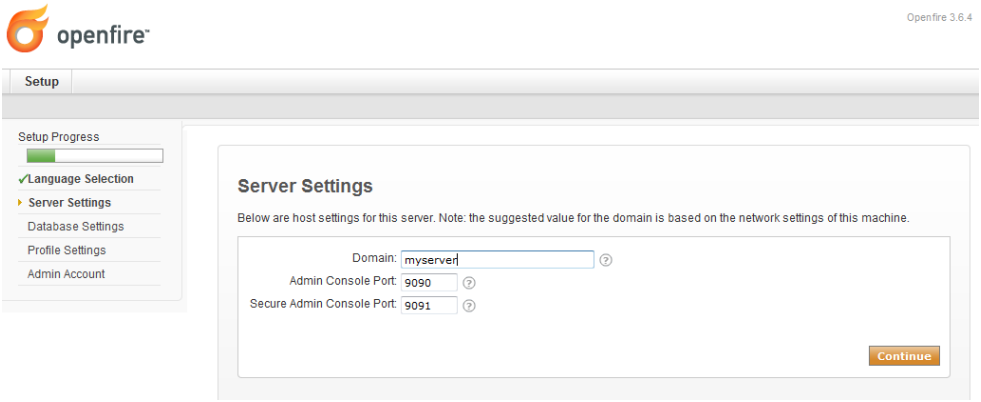


Gambar 2.13 Contoh *IQ Semantics* (Saint-Andre)

2.2.9 Openfire

Openfire adalah aplikasi *open-source* yang digunakan sebagai referensi untuk *chat* dan komunikasi internal dengan bantuan *Extensible Messaging dan Presence Protocol* (XMPP). Openfire dapat diunduh gratis dari Ignite Realtime (www.igniterealtime.org/projects/openfire), sebuah komunitas Software Jive. Openfire dapat dijalankan pada *web server*

dengan berbagai sistem operasi seperti Windows, Linux, dan Macintosh (Chan et al, 2012).



Gambar 2.14 Tampilan Instalasi Openfire (Chan et al, 2012)

2.2.10 aSmack

aSmack adalah suatu *library* XMPP yang disediakan untuk mendukung pengembangan aplikasi Android. *Library* aSmack merupakan hasil modifikasi dari *library* Smack supaya dapat digunakan pada pengembangan aplikasi Android.