BAB II

TINJAUAN PUSTAKA

II.1. Analisa Dan Perancangan

Analisis sistem merupakan sebuah teknik pemecahan masalah yang menguraikan sebuah sistem menjadi beberapa bagian komponen-komponen dengan tujuan mempelajari kinerja dari masing-masing komponen dan berinteraksi untuk mencapai tujuan. Analisis sistem digunakan sebagai pembelajaran sebuah sistem dan komponen-komponennya sebagai persyaratan desain sistem untuk sistem yang akan dibuat ataupun sistem yang akan diperbaharui.

Perancangan merupakan tindak lanjut dari tahapan perencanaan dimana sketsa-sketsa ide mulai ditampilkan.

Desain sistem merupakan teknik pemecahan masalah (setelah kegiatan analisis sistem) dengan menyatukan kembali bagian-bagian komponen menjadi sebuah sistem yang utuh/lengkap. Desain sistem bertujuan untuk merancang sistem yang baru atau memperbaiki sistem yang lama.

Analisis informasi merupakan fase pengembangan dalam sebuah proyek pengemangan sistem informasi yang intinya berfokus pada masalah persyaratan-persyaratan bisnis. *Repository* merupakan lokasi yang menjadi tempat para analisis sistem, desainer sistem dan pembangunan sistem dalam menyimpan semua dokumentasi yang berhubungan dengan satu atau lebih sistem/proyek. *(Fathoni Muhammad dan Sulindawati, Jurnal SAINTIKOM;2010;Hal:5)*

II.2. Keamanan

II.2.1. Dasar Pemikiran Keamanan Komputer

Agar dapat mengamankan sebuah sistem dengan benar kita harus tahu karakteristik penggangu yang mungkin akan mendatangi sistem kita. Kata aman dapat didefinisikan sebagai terhindar dari serangan atau kegagalan. Jadi ada beberapa ancaman yang dapat mengacaukan sistem, yang sering tanpa kita sadari telah dikelilingi oleh berbagai bentuk ancaman. Suatu sistem baru dapat dikatakan aman apabila dalam segala keadan, sumberdaya yang digunakan dan yang diakses adalah sesuai dengan kehendak pengguna. Sayangnya tidak ada satu sistem komputer yang memiliki sistem keamanan sempurna. Oleh sebab itu setidaknya kita harus mempunyai suatu mekanisme tertentu yang dapat mencegah terjadinya pelanggaran pada sistem kita.penggangu yang mungkin akan mendatangi sistem kita. *(Simarmata Janner,2006;Hal:1)*

II.2.2. Mengapa Kita Memerlukan Pengamanan ?

Banyak pertanyaan yang mungkin timbul di dalam fikiran kita. Mengapa kita membutuhkan keamanan, seberapa aman, atau apa yang hendak kita lindungi, seberapa penting data kita sehingga kita perlu memusingkan diri dengan masalah keamanan?

Di dalam dunia global dengan komunikasi data yang berkembang pesat dari waktu ke waktu, dengan koneksi internet yang semakin murah, masalah keamanan ternyata masih sering luput dari perhatian pemakai sebuah sistem meskipun sebenarnya masalah itu sudah menjadi isu yang sangat serius. Bagi banyak pihak, keamanan data telah menjadi kebutuhan dasar mengingat semakin

rentannya penggunaan jaringan global itu. Sebagai contoh, memindahkan data dari titik A ke titik B di *internet* ataupun pada suatu jaringan, data harus melalui beberapa titik lain selama dalam perjalanan, yang membuka kesempatan bagi orang lain untuk memotong ataupun mengubah data anda menjadi sesuatu yang tidak anda inginkan. Akses ke sistem anda, meskipun tidak anda ijinkan, mungkin dapat dilakukan oleh penyusup, yang juga dikenal sebagai *cracker*, yang kemudian menggunakan keahliannya untuk berpura-pura menjadi anda, mencuri informasi dari sistem anda, atau bahkan menolak akses anda ke sumbar daya anda sendiri. *(Simarmata Janner; 2006;Hal:2)*

II.2.3. Meminimalisasi Celah keamanan

Sekarang kita dapat mempelajari lebih jauh mengenai seberapa tinggi tingkat keamanan yang kita miliki atau pun yang kita perlukan. Satu hal yang perlu diingat bahwa tidak ada suatu sistem yang memiliki sistem keamanan yang sempurna. Hal yang dapat kita lakukan adalah mencoba meminimalisasi celah keamanan yang ada pada sistem kita. Untuk pengguna *linux* rumahan yang hanya menggunakannya untuk kerperluan pribadi di rumah, mereka mungkin tidak perlu berfikir terlalu banyak. Namun bagi pengguna *linux* skala besar, seperti *Bank* dan perushaan telekomunikasi, usaha ekstra keras harus banyak dilakukan.

Lain hal yang perlu kita ingat bahwa semakin aman sistem yang kita gunakan, sistem kita akan semakin merepotkan. Anda harus menyeimbangkan antara kenyamanan pemakai sistem dan protaksi demi alasan keamanan. Sebagai contoh, anda bisa memaksa orang lain yang ingin masuk kedalam sistem anda untuk menggunakan *call-back modem* untuk melakukan panggilan balik melalui

nomor telepon rumah mereka. Cara ini kelihatannya memang lebih aman, namun jika tidak ada seorang pun di rumah, hal itu akan menyulitkan mereka untuk login. And juga dapat mengatur konfigurasi sistem *linux* tanpa jaringan atau koneksi ke *internet*, tetapi pembatasan ini juga akan membatasi kegunaan sistem anda. Jika anda memiliki situs dengan ukuran menengah sampai besar, anda harus membangun seperangkat kebijakan sesuai dengan tingkat keamanan yang anda inginkan. *(Simarmata Janner; 2006;Hal:3)*

II.2.4. Apa Yang Akan Anda Lindungi?

Sebalum anda berusaha mengamankan sistem anda, anda harus menentukan terlebih dahulu beberapa hal. Anda perlu memikirkan tingkat ancaman yang harus anda hadapi, resiko yang harus anda ambil, dan seberapa kebal sistem anda sebagai hasil dari usaha yang telah anda lakukan. Anda harus menganalisis sistem anda untuk mengetahui apa yang anda lindungi, kenapa anda melindunginya, seberapa besar nilai data yang anda lindungi, kenapa anda melindunginya, seberapa besar nilai data yang anda lindungi, dan siapa yang bertanggung jawab terhadap data dan aset lain dalam sistem anda.

Resiko adalah kemungkinan di mana seorang penyusup berhasil dalam usahanya untuk mengakses sistem anda. Dapatkah seorang penyusup membaca, menulis berkas, ataupun mengeksekusi program yang dapat menyebabkan kerusakan? Dapatkah mereka menghapus data yang penting? Sebagai tambahan, memiliki *account* didalam sistem yang tidak aman dapat mengakibatkan anda menjadi korban pencurian. Anda harus memutuskan siapa yang anda percaya untuk mengakses sistem dan siapa yang dapat mengancam sistem anda

Ada beberapa tipe penyusup dengn karakteristik yang berbeda satu dengan yang lain, diantaranya:

1. ***The Curious***. Penyusup tipe ini pada dasarnya tertarik untuk mecari tahu tipe sistem dan data yang anda miliki.
2. ***The Malicious***. Penyusup tipe ini suka menggangu sistem anda sehingga sistem tidak dapat bekerja dengan optimal, merusak halaman situs web anda, ataupun memaksa anda untuk menghabiskan banyak uang dan waktu untuk memperbaiki kerusakan yang dibuatnya.
3. ***The High-Profile intruder***. Penyusup tipe ini mencoba menyusup ke dalam sistem anda untuk mendapatkan ketenaran dan pengakuan. Mungkin dia ingin menggunakan sistem anda yang terkenal canggih sebagai sarana untuk membuatnya terkenal.
4. ***The Competition***. Penyusup tipe ini tertarik pada data yang ada dalam sistem anda. Ia mungkin berfikir ada sesuatu yang berharga dalam sistem anda yang dapat memberikan keuntungan baginya.
5. ***The Browsers***. Penyusup tipe ini akan menggunakan sumber daya yang kita miliki untuk kepentingan mereka. Biasnya penyusup ini akan menjelaskannya sebagai *server chatting(IRC),* situs porno, atau bahkan *server DNS.*
6. ***The LeapFrogger***. Penyusup tipe ini hanya tertarik untuk menggunakan sistem yang anda miliki untuk masuk kedalam sistem lain. Jika sistem anda terhubung atau merupakan sebuah *gateway* ke sejumlah *host internal,* anda akan menyaksikan penyusup ini sedang berusha untuk berkompromi dengan sistem yang anda miliki. *(Simarmata Janner; 2006;Hal:4)*

II.3. SMS

II.3.1. Memahami Sistem Kerja SMS

Short Message Service (SMS) merupakan salah satu fitur dari *GSM* yang dikembangkan dan distandarisasi oleh *ETSI.* Pada saat kita mengirim pesan *sms* dari ponsel, maka pesan *sms* tersebut tidak langsung dikirim ke ponsel tujuan, akan tetapi terlebih dahulu dikirim ke *SMS Center (SMSC)* dengan prinsip *Store and Forward* (Simpan dan teruskan), setelah itu baru dikirimkan ke ponsel yang dituju.

Dengan adanya *SMSC* ini, kita dapat mengetahui status dari *sms* yang dikirim, apakah telah sampai atau gagal diterima oleh ponsel tujuan. Apabila ponsel tujuan dalam keadaan aktif dan menerima *sms* yang dikirim, ponsel tersebut akan mengirim kembali pesan konfirmasi ke *SMSC* yang menyatakan bahwa *sms* telah diterima. Kemudian *SMSC* mengirimkan kembali status tersebut kepada pengirim. Tetapi jika ponsel tujuan dalam keadaan mati atau di luar jangkauan, *sms* yang dikirim akan disimpan pada *SMSC* sampai periode validitas terpenuhi, jika periode validitas terlewati maka *sms* itu akan dihapus dari *SMSC* dan tidak dikirimkan ke ponsel tujuan. Di samping itu, *SMSC* akan mengirim pesan informasi ke nomor pengirim yang menyatakan pesan yang dikirim belum diterima atau gagal. *(Purnomo Adi;2007,Hal:1)*

II.4. Algoritma

II.4.1 Pengertian Algoritma

Algoritma adalah susunan langkah-langkah sistematis dan logis dalam pemecahan suatu masalah. Ada 3 cara dalam menyusun algoritma yaitu:

1. Dengan merumuskan langkah-langkah pemecahan masalah melalui kalimat yang terstruktur (tersusun secara logis)
2. Menggabungkan kalimat dengan penggalan *statements* yang ada di suatu bahasa pemrograman (mis: *Pascal*), biasanya disebut *Pseudo code* (mirip kode/perintah pemrograman)
3. menggunakan diagram alir (*flowchart*). *(Jurnal Fathoni Muhammad dan saniman, Jurnal SAINTIKOM;2008;Hal:120)*

II.4.2. Sejarah Algoritma

Sejarah kata algoritma berasal dari nama seorang ahli matematika bangsa arab yaitu *Abu Ja'far Muhammad Ibnu Musa Al-Khuwarizmi*. *Al-khuwarizmi* dibaca oleh orang barat menjadi algorism. Perubahan kata algorism menjadi *algorithm* karena kata *algorism* sering dikelirukan dengan *arithmetic*, sehingga akhiran –sm berubah menjadi –thm. Lambat laun kata algorithm dipakai sebagai metode perhitungan (komputasi) secara umum, sehingga kehilangan makna aslinya. Dalam bahasa Indonesia kaa *algorithm* diserap menjadi algoritma. *(Jurnal Fathoni Muhammad dan saniman, Jurnal SAINTIKOM;2008;Hal:120)*

II.5. Base64

II.5.1. Pengenalan MIME Base64 Encoding

Base64 atau *quadrosexagesimal* adalah penempatan notasi dengan menggunakan bilangan *radix* 64. *Base64* merupakan bilangan berbasis 2 (*power- of- two*) terbesar yang dapat direpresentasikan dengan menggunakan karakter *ASCII*. Hal ini memungkinkan untuk penggunaannya melakukan *encoding* terhadap *email* dan lainnya. *Base64* menggunakan karakter A – Z, a – z dan 0 –untuk 62 nilai pertama, sedangkan 2 nilai terakhir digunakan untuk *symbol* (+ dan /). Beberapa metode *encoding* lain seperti *unicode* dan *binhex* menggunakan 64 karakter yang berbeda untuk mewakili 6 *binary digit*, namun metode- metode tersebut tidak disebut sebagai*encoding Base64*. Untuk tabel *index* dapat dilihat di Tabel II.1

Tabel II.1. *Index Encoding Base64*

Value	Char	Value	Char	Value	Char	Value	Char
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w		
15	P	32	g	49	x		
16	Q	33	h	50	y		
(pad)	=						

(Sumber : *Teguh P. Salman*; 2007 : Hal:5)

Langkah – langkah Enkripsi menggunakan algoritma Base64

Contoh Mengubah kata " **M a n** "

1. Ubah huruf-huruf yang akan dienkripsi menjadi kode-kode *ascii*. Berikut kode *ascii* pada tabel II.2.

Tabel II.2. Kata ke *ASCII*

Text content	M	a	n
ASCII	77	97	110

(Sumber : *Teguh P. Salman*; 2007)

2. Kode-kode *ascii* tersebut diubah lagi menjadi kode *biner*. Berikut kode *biner* pada Tabel II.3.

Tabel II.3. *Ascii* ke *Biner*

Text content	M	a	n
ASCII	77	97	110
Bit pattern	0 1 0 0 1 1 0 1	0 1 1 0 0 0 0 1	0 1 1 0 1 1 1 0

(Sumber : *Teguh P. Salman*; 2007)

3. Bagi kode *biner* tersebut menjadi hanya 6 blok angka dan berjumlah kelipatan 4 blok.
4. Jika angka *biner* tidak berjumlah 6 angka dan 4 blok maka akan ditambah kode *biner* 0 sehingga mencukupi menjadi 4 blok.
5. Blok – blok tersebut ubah kembali menjadi kode desimal (data dibaca sebagai *index*). Berikut tabel *biner* 6 blok pada Tabel II.4.

Tabel II.4. *Biner* dibagi 6 blok dan diubah ke *index*

Text content	M		a		n	
ASCII	77		97		110	
Bit pattern	0 1 0 0 1 1	0 1	0 1 1 0	0 0 0 1	0 1 1 0	1 1 0
Index	19		22	5	46	

(Sumber : *Teguh P. Salman*; 2007)

6. Hasil kode index tersebut diubah menjadi huruf yang ada pada *index*. Untuk tabel *index* ke *string* dapat dilihat di Tabel II.5.

Tabel II.5. *Index* Ke *String*

Text content	M		a		n	
ASCII	77		97		110	
Bit pattern	0 1 0 0 1 1 0	1 0	1 1 0 0 0 0 1	0 1	1 0 1 1 1 0	
Index	19		22	5	46	
Base64 Encoded	T		W	F	u	

(Sumber : Teguh P. Salman; 2007)

7. Jika nilai blok adalah hasil tambahan (0) maka hasil dari *index* tersebut bernilai ‘=’. Untuk tabel enkripsi dapat dilihat pada Tabel II.6.

   Contoh

Tabel I.6. Hasil Enkripsi

Text content	M			
ASCII	77			
Bit pattern	0 1 0 0 1 1 0	1 0	0 0 0 0	0 0 0 0
Index	19	16	(kosong)	(kosong)
Base64 Encoded	T	Q	=	=

(Sumber : *Teguh P. Salman*; 2007)

Langkah – langkah Dekripsi menggunakan algoritma *Base64*

1. Hasil dari Enkripsi diubah menjadi *Index.* Untuk huruf ke index bisa dilihat pada Tabel II.7.

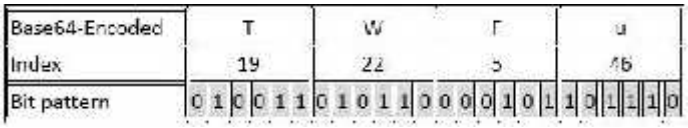
Tabel II.7. Huruf Ke Index

Base64-Encoded	T	W	F	u
Index	19	22	5	46

(Sumber : *Teguh P. Salman*; 2007)

2. Ubah kode *index* menjadi kode-kode *biner (Bit Pattern).* Untuk kode-kode *biner* bisa dilihat pada Tabel II.8.

Tabel II.8. *Index* ke *ASCII 6 blok*



Base64-Encoded	T	W	F	u
Index	19	22	5	46
Bit pattern	0 1 0 0 1 1	0 1 0 1 1 0	0 0 0 1 0 1	1 0 1 1 1 0

(Sumber : *Teguh P. Salman*; 2007)

3. Buat blok pada tiap blok berisi 8 bit data. Untuk tabel blok 8 bit bisa dilihat pada Tabel II.9

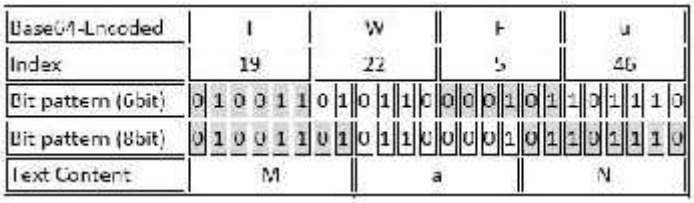
Tabel II.9. *ASCII* 6 Blok Ke *ASCII* 8 Blok

Base64-Encoded	T	W	F	u
Index	19	22	5	46
Bit pattern (6bit)	0 1 0 0 1 1	0 1 0 1 1 0	0 0 0 1 0 1	1 0 1 1 1 0
Bit pattern (8bit)	0 1 0 0 1 1	0 1 0 1 1 0	0 0 0 1 0 1	1 0 1 1 1 0

(Sumber : *Teguh P. Salman*; 2007)

4. Ubah kode *biner* menjadi *ascii* kemudian *string.* Untuk tabel *ascii* 8 blok ke *string* bisa dilihat pada Tabel II.10.

Tabel II.10. *Ascii* 8 Blok Ke *String*

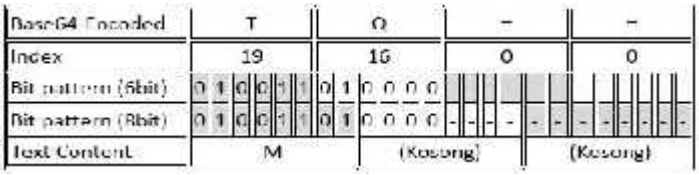


Base64-Encoded	T	W	F	u
Index	19	22	5	46
Bit pattern (6bit)	0 1 0 0 1 1	0 1 0 1 1 0	0 0 0 1 0 1	1 0 1 1 1 0
Bit pattern (8bit)	0 1 0 0 1 1	0 1 0 1 1 0	0 0 0 1 0 1	1 0 1 1 1 0
Text Content	M	a	N	

(Sumber : *Teguh P. Salman*; 2007)

5. Jika nilai sama dengan ‘=’. Untuk hasil dekripsi bisa dilihat pada Tabel II.11.

Tabel II.11. Hasil Dekripsi



Base64-Encoded	T	Q	=	=
Index	19	16	0	0
Bit pattern (6bit)	0 1 0 0 1 1	0 1 0 0 0 0		
Bit pattern (8bit)	0 1 0 0 1 1	0 1 0 0 0 0	- - - - - -	- - - - - -
Text Content	M	(Kosong)	(Kosong)	

(Sumber : *Teguh P. Salman*; 2007)

II.6. Java

II.6.1. Sekilas Tentang Java

Java adalah bahasa pemrograman yang disusun oleh *james Gosling* yang dibantu oleh rekan-rekan seperti *patrick Naugton, Chris Warth, Ed Frank* dan *Mike Sheridan* di suatu perusahaan perngkat lunak yang bernama *Sun Microsystems*, pada tahun1991. Pemrograman ini mula-mula diinisialisasi dengan nama "*Oak*" namun pada tahun 1995 diganti menjadi "*Java*".

Alasan utama pembentukan bahasa *java* adalah untuk membuat aplikasi-aplikasi yang dapat diletakkan diberbagai macam perangkat elektronik, seperti *microwave oven* dan *remote control*, sehingga *java* harus bersifat *portable* atau sering disebut dengan *platform-independent* (tidak tergantung pada *platform*). Itulah yang menyebabkan dalam dunia pemrograman *java*, dikenal adanya istilah '*white once, run everywhere'*, yang berarti kode program hanya ditulis sekali, namun dapat dijalankan dibawah *platform* manapun, tanpa harus melakukan perubahan kode program. (Rahardjo Budi;2010;Hal:1)

II.6.2. Arsitektur Java

Secara arsitektur, *java* tidak berubah sedikit pun semenjak awal mula bahasa tersebut dirilis. Compiler java (yang disebut dengan *javac* atau *java compiler*) akan mentransformasikan kode-kode dalam bahasa *java* kedalam suatu *bytecode* . apa itu *bytecode? Bytecode* adalah sekumpulan perintah hasil kompilasi yang kemudian dapat dieksekusi melalui sebuah mesin komputer abstrak, yang disebut dengan *JVM* (*Java Virtual Machine*). *JVM* juga sering dinamakan sebagai

interpreter, karenasifatnya yang selalu menerjemahkan kode-kode yang tersimpan dalam *bytecode* dengan cara baris demi baris. *(Rahardjo Budi;2010;Hal:2)*

II.6.3. Java Versi Lama (Java 1)

Pada awal perilisannya, versi *java* masih disebut dengan *JDK* (*Java Development Kit*). Dalam *JDK*, semua kebutuhan untuk pengembangan program dan eksekusi program masih tergabung jadi satu. Penamaan program ini berlaku sampai *java 1.1*. Namun sekarang, setelah *Java 1.2*, *Sun Microsystems* menamainya dengan *JSDK* (*Java Software DevelopmentKit*) dalam hal ini kebutuhan untuk pengembangan program dipisahkan dengan kebutuhan eksekusi. Bagian *Software* yang digunakan untuk kebutuhan eksekusi program disebut dengan *JRE* (*Java-Runtime Environment*). Selanjutnya, *Java 1.2* disederhanakan penamaannya menjadi *"Java 2". (Rahardjo Budi;2010;Hal:2)*

II.6.4. Java 2

Sun Microsystems telah mendefinisikan tiga buah edisi dari *java 2,* yaitu sebagai berikut :

1. ***Java 2 Standard Edition (J2SE)***, yang digunakan untuk mengembangkan aplikasi-aplikasi *desktop* dan *applet* (aplikasi *java* yang dapat dijalankan didalam *browser web*)
2. ***Java 2 Enterprise Edition (J2EE)***, merupakan *superset* dari aplikasi berskala besar (*Enterprise*), yaitu dengan melakukan pembuatan aplikasi-aplikasi di sisi *server* dengan menggunakan *servlet* dan *JSP* (*JavaServer Pages*) dan teknologi lainnya seperti *CORBA* (*Common Object Request Broker Architecture*) dan *XML*

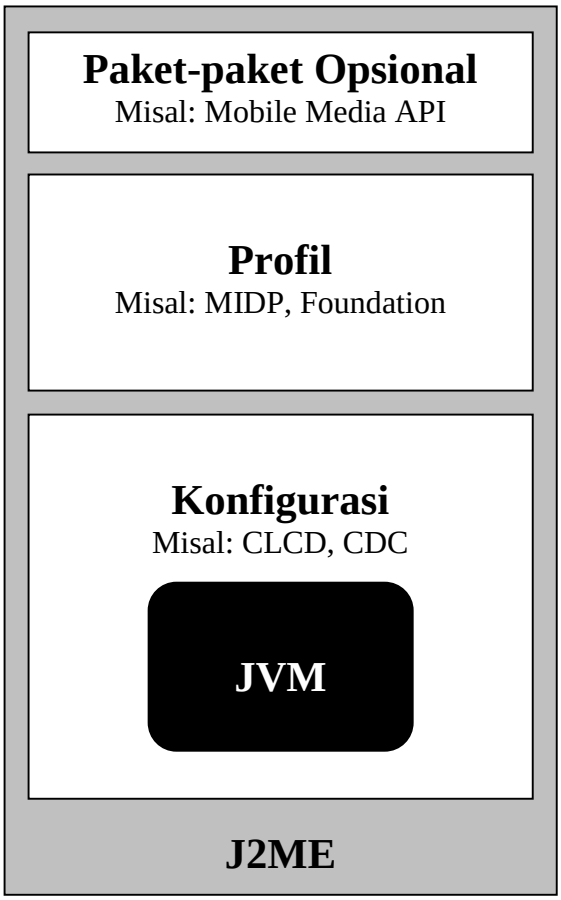
3. ***Java 2 Micro Edition* (*J2ME*)**, Merupakan *subset* dari *J2SE* yang digunakan untuk menangani pemrograman didalam perangkat-perangkat kecil, yang tidak memungkinkan untuk mendukung implementasi *J2SE* secara penuh. *(Rahardjo Budi;2010;Hal:3)*

II.6.5. Apa Itu J2ME?

J2ME merupakan sebuah kombinasi yang terbentuk antara sekumpulan *interface* java yang sering disebut dengan *java API* (*Aplication Programming Interface*) dengan *JVM* (*Java Virtual Machine*) yang dedesain khusus untuk alat, yaitu *JVM* dengan ruang yang terbatas. Kombinasi tersebut kemudian digunakan untuk melakukan pembuatan aplkasi-aplikasi yang dapat berjalan diatas alat (dalam hal ini *mobile Device*).

Mungkin anda akan bertanya, apakah kita, sebagai pengembang aplikasi (developer/programmer), harus melakukan instalasi *JVM* dan java *API* ke dalam alat yang kita gunakan? Jawabannya: "*Tidak*". Masing-masing dari perusahaan alat telah menyediakan *JVM* (dan sekumpulan java *API* yang diperlukan) di dalam alat bersangkutan. Hal ini, membuat kita sebagai *developer*, hanya perlu berkonsentrasi dalam pengembangan aplikasinya dan memasukkannya ke dalam alat tersebut.

J2ME sendiri pada dasarnya terdiri dari tiga buah bagian, yaitu konfigurasi, profil, dan paket-paket optional, seperti yang ditunjukkan oleh gambar II.1.



Gambar II.1. bagian-bagian di dalam Platform J2ME
(Sumber: *Java Untuk Handphone dan Alat Mobile*;2010;Hal:3)

1. Konfigurasi

Konfigurasi merupakan bagian yang berisi *JVM* dan beberapa *Library* kelas lainnya, perlu diperhatikan bahwa *JVM* yang dimaksud di sini bukanlah *JVM* tradisional seperti yang terdapat pada *J2SE*, melainkan *JVM* yang sudah didesain secara khusus untuk alat.

Terdapat dua buah konfigurasi yang disediakan oleh *Sun Microsystems*, yaitu *CLDC* (*Connected Limited Devide Configuration*) dan *CDC* (*Connected Device Configuration*) . Target alat dari konfigurasi *CLDC* adalah alat-alat kecil, seperti telepon selular, *PDA*, dan *pager*. Kita akan membahas lebih jauh mengenai

konfigurasi *CLDC* pada bagian tersendiri dalam bab ini. Pada sisi lain, *CDC* merupakan superset dari *CLDC* sehingga semua yang kelas yang didefinisikan di dalam *CLDC* akan ada juga di dalam *CDC*.

2. Profil

Prifil merupakan bagian perluasan dari konfigurasi. Artinya, selain sekumpulan kelas yang terdapat pada konfigurasi, terdapat juga kelas-kelas spesifik yang didefenisikan lagi di dalam profil. Dengan kata lain, profil akan membantu secara fungsioal yaitu dengan menyediakan kelas-kelas yang tidak terdapat di level konfigurasi.

Adapun profil yang sangat popular penggunanya adalah profil yang disediakan oleh *Sun Microsystems*, yaitu yang dinamakan dengan *MIDP* (*Mobile Information Device Profile*) .

3. Paket-Paket Opsional

Paket-paket opsional merupakan paket-paket tambahan yang dibutuhkan oleh aplikasi shingga pada saat proses *deployment* paket-peket tersebut perlu didistribusikan juga sebagai bagian dari aplikasi bersangkutan. Sebagai catatan bahwa peket-peket opsional ini bukan merupakan paket yang dibuat oleh perusahaan alat yang digunakan. *(Rahardjo Budi;2010;Hal:3)*

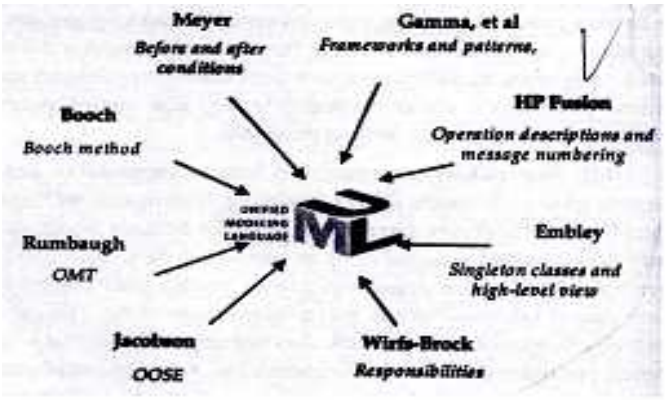
II.7 UML (*Unified Modelling Language*)

UML (*Unified Modelling Language*) adalah salah satu alat bantu yang sangat handal di dunia perkembangan sistem yang berorientasi objek. Hal ini disebabkan karena *UML* menyediakan bahasa pemodelan visual yang memungkinkan bagi perkembangan sistem untuk membuat cetak biru atas visi mereka dalam bentuk yang baku, mudah dimengerti serta dilengkapi dengan mekanisme yang efektif untuk berbagi (sharing) dan mengkomunikasikan rancangan dengan baik (Munawar ; 2005 : 17).

UML merupakan kesatuan bahasa pemodelan yang dikembangkan oleh Booch, *Object Modeling Technique* (*OMT*) dan *Object Oriented Engineering* (OOSE). Metode Booch dari Grady Booch sangat terkenal dengan nama metode *Design Object Oriented*. Metode ini menjadikan proses analisis dan design ke dalam empat tahapan interatif, yaitu: identifikasi kelas-kelas dan objek-objek, identifikasi semantik dari hubungan objek dan kelas tersebut, perincian *interface* dan implementasi.Keunggulan metode Booch adalah pada detail dan kayanya dengan notasi dan elemen. Pemodelan *OMT* yang dikembangkan oleh Rumbaugh didasarkan pada analisis terstuktur dan pemodelan *entity-relationship*. Tahapan utama dalam metodologi ini adalah analisis, disain sistem, desain objek dan implementasi. Keunggulan metode ini adalah dalam penotasian yang mendukung semua konsep *OO*. Metode *OOSE* dari Jacobson lebih memberi penekanan dan *use case*. *OOSE* memiliki tiga tahapan yaitu membuat model *requirement* dan analisis, desain dan implementasi dan model pengujian (test Model). Keunggulan

metode ini adalah mudah dipelajari karena memiliki notaasi sederhana namun mencakup seluruh tahapan dalam rekayasa perangkat lunak.

Dengan *UML*, metode *Booch*, *OMT* dan *OOSE* digabungkan dengan elemen-elemen dari metode lain yang lebih efektif dan elemen-elemen baru yang belum ada pada metode terdahulu sehingga *UML* lebih ekspresif dan seragam dari pada metode lainnya. Unsur-unsur yang membentuk *UML* ditunjukkan dalam Gambar II.2



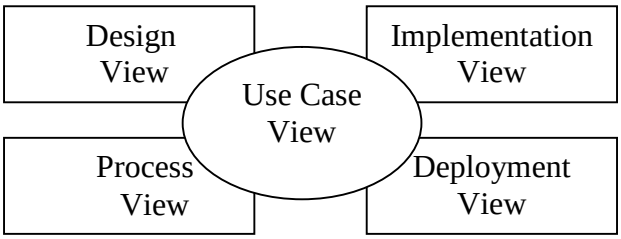
Gambar II.2. Unsur-unsur yang membentuk UML

(Sumber : *Munawar, Pemodelan Visual dengan UML,* 2005 : 18)

UML adalah hasil kerja dari konsorsium berbagai organisasi yang berhasil dijadikan sebagai standar baku dalam *OOAD* (*Object Oriented Analysis* dan *Design*). UML tidak hanya domain dalam penotasian dilingkungan *OO* tetapi juga populer di luar lingkungan *OO*. Ada tiga karakter penting yang melekat di *UML* yaitu sketsa, cetak biru dan bahasa pemrograman. Sebagai sebuah sketsa *UML* bisa berfungsi sebagai sebuah cetak biru karena sangat lengkap dan detil. Dengan cetak biru ini maka akan bisa diketahui informasi detil tentang coding

program (*Forward engineering*) atau bahkan membaca program dan mengimplementasikannya kembali ke dalam diagram (*reverse engineering*). *Reverse engineering* sangat berguna pada situasi dimana kode program yang tidak terdokkumentasi asli hilang atau bahkan belum perna dibuat sama sekali. Sebagai bahasa pemrograman, *UML* dapat diterjemahkan diagram yang ada di *UML* menjadi kode program siap untuk dijalankan.

UML dibangun atas model 4+1 *view*. Model ini didasarkan pada fakta bahhwa struktur sebuah sistem dideskripsikan dalam *view* dimana salah satu diantaranya *use case view*. *use case view* ini memegang peran khusus untuk mengintegrasikan *content* ke *view* yang lain. Model 4+1 *view* ditunjukkan pada gambar II.3



Gambar II.3. Model 4+1 *View*

(Sumber : *Munawar, Pemodelan Visual dengan UML,* 2005 : 20)

Kelima *view* tersebut tidak berhubungan dengan diagram yang dideskripsikan di *UML*. Setiap *view* berhubungan dengan perspektif tertentu dimana sistem akan diuji. *View* yang berbeda akan menekankan pada aspek yang berbeda dari sistem yang mewakili tentang sistem bisa dibentuk dengan menggabungkan informasi-informasi yang ada pada kelima *view* tersebut.

Use case view mendefinisikan perilaku eksternal sistem. Hal ini menjadi daya tarik bagi *end user*, analis dan tester. Pandangan ini mendefenisikan kebutuhan sistem karena mengandung semua *view* yang lain yang mendeskripsikan aspek-aspek tertentu dari peran dan sering dikatakan yang mendrive proses perkembangan perangkat lunak.

Design view mendeskripsikan struktur logika yang mendukung fungsi-fungsi yang dibutuhkan di *use case*. *Design view* ini berisi definisi komponen program, class-class utama bersama-sama dengan spesifikasi data, perilaku dan interaksinya. Informasi yang terkandung di *view* menjadi pergantian para progremer karena menjelaskan secara detil bagaimana fungsionalitas sistem akan diimplementasikan.

Implementasi *view* menjelaskan komponen-komponen visi yang akan dibangun. Hal ini berbeda dengan komponen logic yang dideskripsikan pada *design view*. Termasuk disini diantaranya *file exe*, *library* dan *database*. Informasi yang ada di *view* dan integrasi siistem.

Proses *view* berhubungan dengan hal-hal yang berkaitan dengan *concurrency do* dalam sistem. Sedangkan *deployment view* menjelaskan bagaimana komponen-komponen fisik didistribusikan ke lingkungan fisik seperti jaringan komputer dimana sistem akan dijalankan. Kedua view ini menunjukan kebutuhan non fungsional dari sistem seperti toleransi kesalahan dan hal-hal yang berhubungan dengan kinerja *(Munawar;2005:17-21).*

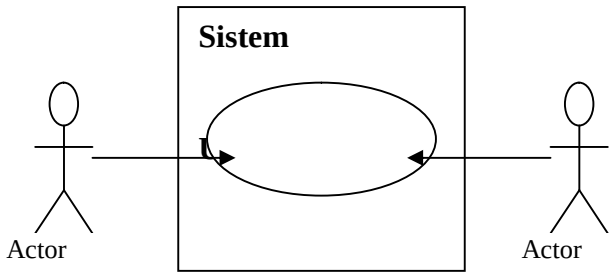
II.7.1 Use Case Diagram

Use case adalah deskripsi fungsi dari sebuah sistem dari perspektif pengguna. *Use case* bekerja dengan cara deskripsikan tipikal interaksi antara *user* (pengguna) sebuah sistem dengan sistemnya sendiri melalui sebuah cerita bagaimana sebuah sistem dipakai. Urutan langkah-langkah yang menerangkan antara pengguna dan sistem yang disebut *scenario*. Setiap *scenario* mendeskripsikan urutan kejadian. Setiap urutan diinisialisasi oleh orang, sistem yang lain, perangkat keras dan urutan waktu. Dengan demikian secara singkat bisa dikatakan *use case* adalah serangkaian *scenario* yang digabungkan bersama-sama oleh pengguna tujuan umum pengguna.

Dalam pembicaraan tentang *use case*, pengguna biasanya disebut dengan *actor*. *Actor* adalah sebuah peran yang bisa dimainkan oleh pengguna dalam interaksinya dengan sistem.

Model *use case* adalah bagai dari model *requirement*. Termasuk disini adalah problem domain object dan penjelasan tentang *user interface*. *Use case* memberikan spesifikasi fungsi-fungsi yang ditawarkan oleh sistem dari *persfectif user*.

Notasi *use case* menunjukkan 3 aspek dari sistem yaitu *actor use case* dan *system / sub system boundary. Actor* mewakili peran orang, *system* yang lain atau alat ketika berkomunikasi dengan *use case*. Ilustrasi *actor, use case* dan *system* ditunjukkan pada gambar II.4



Gambar II.4. *Use Case* Diagram

(Sumber : *Munawar, Pemodelan Visual dengan UML,* 2005 : 64)

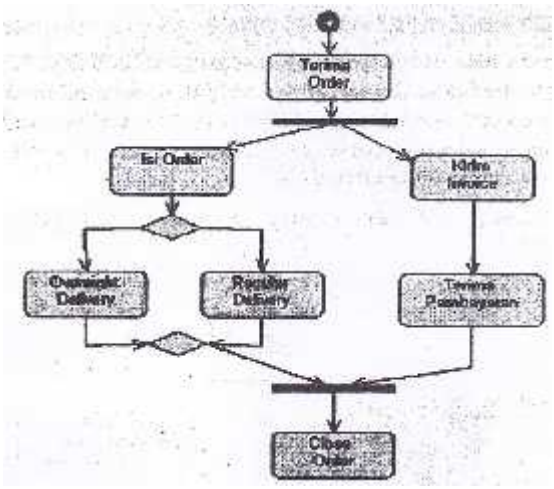
Untuk mengidentifikasi *actor*, harus ditentukan pembagian tenaga kerja dan tugas-tugas yang berkaitan dengan peran pada konteks targer sistem. *Actor* adalah *abstraction* dari orang dan sistem yang lain mengaktifkan fungsi dari target sistem. Orang atau sistem bila muncul dalam beberapa peran. Perlu dicatat bahwa *actor* berinteraksi dengan use case, tetapi tidak memiliki kontrol atas use *case*.

Use case adalah abstraksi dari interaksi antara sistem dan *actor*. Oleh kerena itu sangat penting untuk memilih abstraksi yang cocok. Use case dibuat berdasarkan keperluan actor. Use case harus merupakan ’apa’ yang dikerjakan software aplikasi, bukan ’bagaimana’ software aplikasi mengerjakannya. Setiap use case harus diberi nama yang menyatakan apa hal yang dicapai dari hasil interaksinya dengan actor. Namun use case boleh terdiri dari beberapa kata dan tidak boleh ada dua use case yang memiliki nama yang sama. *(Munawar ; 2005 : 63-66).*

II.7.2 Activity Diagram

Activity diagram adalah teknik untuk mendeskripsikan logika prosedural, proses bisnis dan aliran kerja dalam banyak kasus. *Activity diagram* mempunyai peran seperti halnya *flowchart*, akan tetapi perbedaanya dengan *flowchart* adalah

activity diagram bisa mendukung perilaku paralel sedangkan flowchart tidak bisa. Contoh *activity diagram* sederhana ditunjukkan pada gambar II.5



Gambar II.5. Contoh *Activity Diagram* Sederhana

(Sumber : *Munawar, Pemodelan Visual dengan UML*, 2005 : 111)

II.7.3 Squence Diagram

Squence diagram digunakan untuk menggambarkan perilaku pada sebuah sekenario. Diagram ini menunjukkan sebuah contoh objek dan pesan yang diletakkan diantara objek-objek ini didalam *use case*.

Komponen utama *Squence diagram* terdiri dari atas objek yang dituliskan dengan kotak segiempat bernama. *Messege* diwakili oleh garis dengan tanda panah dan waktu yang ditunjukkan dengan *progress vertical* (Munawar ; 2005 : 109).

1. Objek / *participant*

Objek diletakkan di dekat bagian atas diagram dengan urutan dari kiri ke kanan. Mereka diatur dalam urutan guna menyederhanakan diagram. Setiap *participant* dihubungkan garis titik-titik yang disebut *lifeline.* Sepanjang *lifeline*

ada kotak yang disebut *activation*. *Activation* mewakili sebuah eksekusi operasi dari *participant*. Panjang kotak ini berbanding lurus dengan durasi *activation*. *Activation* mewakili sebuah eksekusi operasi dari participant. Panjang kotak ini berbanding lurus dengan durasi *activation.* Bentuk *participant* dapat dilihat pada gambar II.6

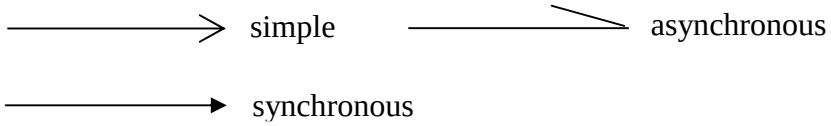
Gambar II.6. Bentuk *Participant*

(Sumber : *Munawar, Pemodelan Visual dengan UML,* 2005 : 88)

2. *Messege*

Sebuah *messege* bergerak dari suatu *participant* ke *participant* yang lain dan dari *lifeline* ke *lifeline* yang lain. Sebuah *participant* bisa mengirim sebuah *message* kepada dirinya sendiri.

Sebuah *message* bisa jadi *simple*, *synchronous* atau *asynchoronous*. *Message* yang *simple* adalah sebuah perpindahan (transfer), contoh dari satu *participant* ke *participant* yang lainnya. Jika suatu *participant* mengirimkan sebuah *message* tersebut akan ditunggu sebelum di proses dengan urusannya. Namun jika *message asynchoronous* yang dikirimkan, maka jawabannya atas *message* tersebut tidak perlu ditunggu. Simbol *message* pada *squnence diagram* dapat dilihat pada gambar II.7



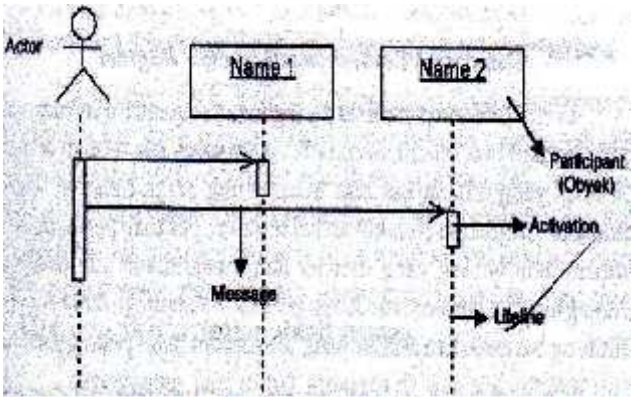
Gambar II.7. Bentuk *Messege*

(Sumber : *Munawar, Pemodelan Visual dengan UML,* 2005 : 88)

3. *Time*

Time adalah diagram yang mewakili waktu pada arah vertikal. Waktu dimulai dari atas ke bawah. *Message* yang lebih dekat dari atas akan dijalankan terlebih dahulu dibanding *message* yang lebih dekat kebawah.

Terdapat dua dimensi pada *squence diagram* yaitu dimensi dari kiri ke kanan menunjukkan tata letak participant dan dimensi dari atas ke bawah menunjukkan lintasan waktu. Simbol-simbol yang ada pada *squence diagram* ditunjukkan pada gambar II.8



Gambar II.8. Bentuk *Time*

(Sumber : *Munawar, Pemodelan Visual dengan UML,* 2005 : 89)