BAB II

TINJAUAN PUSTAKA

II.1. Definisi Metode, Penyandian (Enkripsi) dan Dekripsi

II.1. 1. Metode

Metode berasal dari Bahasa Yunani “Methodos’’ yang berarti cara atau jalan yang ditempuh. Metode juga merupakan suatu cara yang dilakukan dalam mengerjakan sesuatu. Sehubungan dengan upaya ilmiah, maka metode menyangkut masalah cara kerja untuk dapat memahami objek yang menjadi sasaran ilmu yang bersangkutan. Fungsi metode berarti sebagai alat untuk mencapai tujuan. (*http://ktiptk.blogspirit.com/archive/2009/01/26/pengertian-metode.html*)

II.1. 2. Penyandian (Enkripsi)

Enkripsi adalah proses mengamankan suatu informasi dengan membuat informasi terseubt tidak dapat dibaca tanpa bantuan pengetahuan khusus. Keuntungan dari enkripsi adalah kode asli kita tidak dapat dibaca oleh orang lain. Proses utama dalam suatu algoritma kriptografi adalah enkripsi dan dekripsi.

Enkripsi mengubah sebuah *plaintext* kedalam bentuk *ciphertext*. Pada mode ECB (*Elektronik Codebook*), sebuah blok pada *plaintext* dienkripsi ke dalam sebuah blok *ciphertext* dengan panjang blok yang sama.

Blok *cipher* memiliki sifat bahwa setiap blok harus memiliki panjang yang sama (misalnya 128 bit). Namun apabila pesan yang dienkripsi memiliki panjang

blok terakhir tidak tepat 128 bit, diperlukan mekanisme padding, yaitu penambahan bit-bit *dummies* untuk menggenapi menjadi panjang blok yang sesuai. Biasanya *padding* dilakukan pada blok terakhir *plaintext*.

Padding pada blok terakhir bias dilakukan dengan berbagai macam cara, misalnya dengan penambahan bit-bit tertentu. Slah satu contoh penerapan *padding* dengan cara menambahkan jumlah total *padding* sebagai *byte* terakhir *plaintextI*. (Wahana Komputer, 2010 ; 4)

II. 1. 3. Dekripsi

Dekripsi adalah proses mengembalikan suatu informasi dengan cara tertentu dan dengan algoritma enkripsi yang dipakai. Dekripsi merupakan proses kebalikan enkripsi, mengubah *ciphertext* kembali ke dalam bentu *plaintext*. Untuk menghilangkan *padding* yang diberikan pada saat proses *enkripsi*, dilakukan berdasarkan informasi jumlah *padding*, yaitu angka pada *byte* terakhir. (Wahana Komputer, 2010 ; 5)

II.2. Kriptografi

II.2.1. Sejarah Kriptografi

Kriptografi mempunyai sejarah yang sangat menarik dan panjang. Kriptografi sudah digunakan 4000 tahun lalu. Diperkenalkan oleh orang-orang Mesir lewat *hieroglyph*. Jenis tulisan ini bukanlah bentuk standar untuk menulis pesan.

Kriptografi telah digunakan pada zaman Romawi kuno dimana Julius Caesar mengirimkan pesan rahasia yang akan disampaikan kepada jendralnya di medan perang. Yang dilakukan Julius Caesar adalah mengganti semua susunan alphabet dari a, b, c yaitu a menjadi d, b menjadi e, c menjadi f dan seterusnya. (Doni Arius:2008:14)

Dari ilustrasi tersebut, beberapa istilah kriptografi dipergunakan untuk menandai aktivitas rahasia dalam mengirim pesan. Apa yang dilakukan Julius Caesar yang mengacak pesan, disebut sebagai enkripsi. Pada saat sang jenderal merapikan pesan yang teracakitu, prose situ disebut deskripsi. Pesan awal yang belum teracak dan pesan yang telah dirapikan, disebut dengan *plaintext*, sedangkan pesan yang telah diacak disebut *ciphertext*.

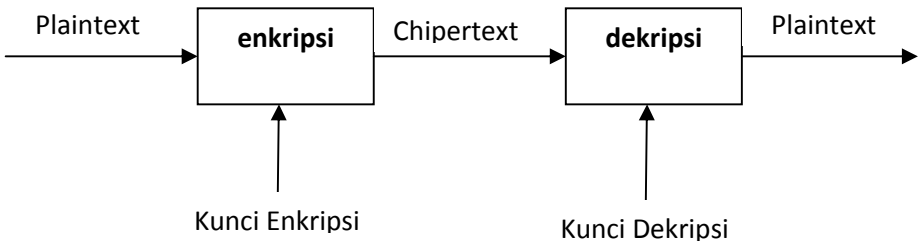
Pada zaman Romawi juga telah ada alat pembuat pesan rahasia yang disebut *Seycale* yang digunakan oleh tentara Sparta. *Seycale* merupakan suatu alat yang memiliki pita panjang dari daun papyrus dan ditambah dengan sebatang silinder. Mula-mula pengirim pesan menuliskan pesannya diatas pita papyrus yang digulung pada batang silinder. Setelah pita dilepaskan dan dikirim. Misalkan batang silinder cukup lebar untuk menulis 6 huruf dan bisa memuat 3 huruf melingkar. (Doni Arius:2008:14)

II.2.2. Konsep Dasar Kriptografi

Kriptografi adalah ilmu mengenai teknik enkripsi dimana data diacak menggunakan suatu kunci enkripsi menjadi sesuatu yang sulit dibaca oleh seseorang yang tidak memiliki kunci dekripsi. Dekripsi menggunakan kunci

dekripsi mendapatkan kembali data asli. Proses enkripsi dilakukan menggunakan suatu algoritma dengan beberapa parameter.

Kriptografi itu sendiri terdiri dari dua proses utama yakni proses enkripsi dan proses dekripsi. Seperti yang telah dijelaskan di atas, proses enkripsi mengubah *plaintext* menjadi *ciphertext* (dengan menggunakan kunci tertentu) sehingga isi informasi pada pesan tersebut sukar dimengerti.



Gambar II.1 Diagram Proses Enkripsi Dan Dekripsi

Biasanya algoritma tidak dirahasiakan,bahkan enkripsi yang mengandalkan kerahasiaan algoritma dianggap sesuatu yang tidak baik. Rahasia terletak di beberapa parameter yang digunakan, jadi kunci ditentukan oleh parameter.Parameter yang menentukan kunci dekripsi itulah yang harus dirahasiakan (parameter menjadi ekuivalen dengan kunci).

Dalam kriptogra klasik, teknik enkripsi yang digunakan adalah enkripsi simetris dimana kunci dekripsi sama dengan kunci enkripsi. Untuk *public key cryptography*, diperlukan teknik enkripsi asimetris dimana kunci dekripsi tidak sama dengan kunci enkripsi. Enkripsi, dekripsi dan pembuatan kunci untuk teknik enkripsi asimetris memerlukan komputasi yang lebih intensif dibandingkan

enkripsi simetris, karena enkripsi asimetris menggunakan bilangan-bilangan yang sangat besar. Namun, walaupun enkripsi asimetris lebih mahal dibandingkan enkripsi simetris, *public key cryptography* sangat berguna untuk *key management* dan *digital signature.*(Sentot Kromodimoelyo:2009:5)

II.2.3. Metode *Vigenere Cipher*

Vigenere Cipher merupakan jenis cipher abjad majemuk yang paling sederhana. *Vigenere Cipher* menerapkan metode substitusi poli alfabetik dan termasuk ke dalam kategori kunci simetris dimana kunci yang digunakan untuk proses enkripsi adalah sama dengan kunci yang digunakan untuk proses dekripsi. *Vigenere Cipher* ditemukan pertama kali oleh Giovan Battista Bellaso. Beliau menuliskan metode enkripsi yang kita kenal sebagai *Vigenere Cipher* ini pada bukunya yang berjudul La Cifradel. Sig. Giovan Battista Bellaso pada tahun 1553. Namun, nama "*Vigenere*" pada *Vigenere Cipher* diambil dari seorang yang bernama Blaise de Vigenere, yang juga merupakan penemu metode algoritma ini setelah Giovan Battista Bellaso.

Enkripsi dengan menggunakan algoritma *Vigenere cipher* pada dasarnya adalah menggunakan prinsip Caesar Cipher, yaitu melakukan enkripsi karakter pada plainteks menjadi karakter lain pada cipherteks. Perbedaan antara Caesar Cipher dan *Vigenere cipher* adalah huruf yang sama pada plainteks tidak selalu dienkripsi menjadi huruf yang sama pada cipherteks. Hal ini terjadi karena pada *Vigenere cipher*, pergeseran karakternya ditentukan oleh karakter yang ada pada kata kunci dan kata ini selalu diulang. Akibatnya, karakter yang sama pada

plainteks boleh jadi memiliki karakter yang berbeda pada cipherteksnya. Karena hal ini lah, *Vigenere cipher* merupakan cipher substitusi abjad-majemuk.

Tujuan utama dari *Vigenere cipher* ini adalah menyembunyikan keterhubungan antara plainteks dan cipherteks dengan menggunakan kata kunci sebagai penentu pergeseran karakternya.(Rinaldi Munir:2010:79)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Gambar II.2 Tabel Vigenere

Untuk menyandikan suatu pesan, digunakan sebuah tabel alfabet yang disebut tabel Vigenere. Tabel Vigenere berisi alfabet yang dituliskan dalam 26 baris, masing-masing baris digeser satu urutan ke kiri dari baris sebelumnya, membentuk ke-26 kemungkinan sandi Caesar. Setiap huruf disandikan dengan menggunakan baris yang berbeda-beda, sesuai kata kunci yang diulang

Misalnya, *plaintext* yang hendak disandikan adalah perintah "STMIK POTENSI UTAMA". Sedangkan sandi yang digunakan adalah "MAHASISWA", "MAHASISWA" diulang sehingga jumlah hurufnya sama banyak dengan *plaintext.*

Huruf pertama pada *plaintext* “S” disandikan dengan menggunakan baris pertama pada sandi yaitu “M”. Pada baris “M” dan kolom “S” di tabel Vigenere, terdapat huruf “E”. Demikian pula untuk huruf kedua dan seterusnya. Proses ini dijalankan terus sehingga mendapatkan hasil sebagai berikut:

Plaintext : STMIK POTENSI UTAMA

Sandi : MAHAS ISWAMAH ASISW

Ciphertext : ETTIC XGPEZSP ULIEW

Secara matematis rumus vigenere cipher dapat ditulis sebagai berikut:

Rumus enkripsi *vigenere cipher* :

$$Ci = (Pi + Ki) \bmod 26$$

atau $Ci = (Pi + Ki) - 26$ kalau hasil $Ci > 26$

Rumus dekripsi *vigenere cipher* :

$$Pi = (Ci - Ki) \bmod 26$$

atau $Pi = (Ci - Ki) + 26$ kalau hasil $Pi < 0$

Dimana:

Ci = karakter *ciphertext* ke-i

Pi = karakter *plaintext* ke-i

Ki = karakter sandi ke-i

Pada contoh diatas kata sandi "MAHASISWA" diulang sedemikian rupa hingga panjang sandi sama dengan panjang plainteksnya. Jika dihitung dengan rumus enkripsi *vigenere cipher*, *plaintext* huruf pertama **S** (yang memiliki nilai **Pi=18**) akan dilakukan pergeseran dengan huruf **M** (yang memiliki **Ki=12**) maka prosesnya sebagai berikut:

Ci = (Pi + Ki) mod 26

= (18 + 12) mod 26

= 30 mod 26

= 4

Ci=4 maka huruf *ciphertext* dengan nilai **4** adalah **E**. Begitu seterusnya dilakukan pergeseran sesuai dengan kunci pada setiap huruf hingga semua *plaintext* telah terenkripsi menjadi *ciphertext*. Setelah semua huruf terenkripsi maka proses dekripsinya dapat dihitung sebagai berikut:

Pi = (Ci – Ki) mod 26

= (4 – 12) mod 26

= –8 mod 26

= -8

Sehingga :

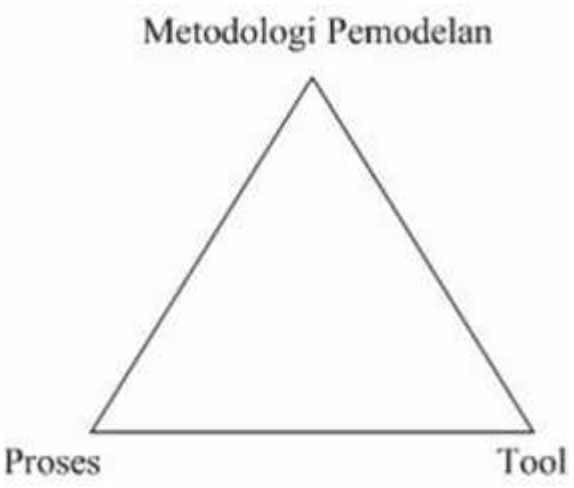
Pi = -8 + 26

= 18

Pi=18 maka huruf *plaintext* dengan nilai **18** adalah **S**. Begitu seterusnya dilakukan pergeseran sesuai dengan kunci pada setiap huruf hingga semua *ciphertext* telah terdekripsi menjadi *plaintext.*

II.3. UML *(Unified Modelling Language)*

Pemodelan (*modeling*) adalah proses merancang piranti lunak sebelum melakukan pengkodean (*coding*). Model piranti lunak dapat dianalogikan seperti pembuatan blueprint pada pembangunan gedung. Membuat model dari sebuah sistem yang kompleks sangatlah penting karena kita tidak dapat memahami sistem semacam itu secara menyeluruh. Semakin komplek sebuah sistem, semakin penting pula penggunaan teknik pemodelan yang baik. Dengan menggunakan model, diharapkan pengembangan piranti lunak dapat memenuhi semua kebutuhan pengguna dengan lengkap dan tepat, termasuk faktor-faktor seperti *scalability, robustness, security*, dan sebagainya. Kesuksesan suatu pemodelan piranti lunak ditentukan oleh tiga unsur, yang kemudian terkenal dengan sebuan segitiga sukses (*the triangle for success*). Ketiga unsur tersebut adalah metode pemodelan (*notation*), proses (*process*) dan *tool* yang digunakan. Memahami notasi pemodelan tanpa mengetahui cara pemakaian yang sebenarnya (proses) akan membuat proyek gagal. Dan pemahaman terhadap metode pemodelan dan proses disempurnakan dengan penggunaan tool yang tepat (Sri Dharwiyanti, 2006 ; 1-2)

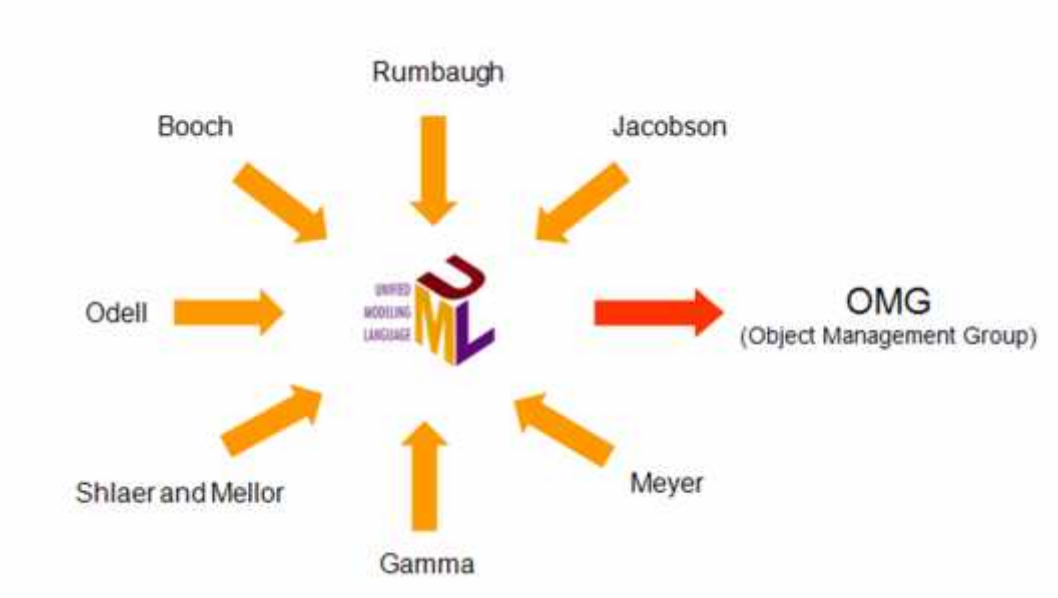


Gambar II.3. Proses Pemodelan dalam sistem

Sumber : **(Sri Dharwiyanti: 2006 ; 2)**

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

bentuk khusus untuk menggambarkan berbagai diagram piranti lunak. Setiap bentuk memiliki makna tertentu, dan UML *syntax* mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan. Notasi UML terutama diturunkan dari 3 notasi yang telah ada sebelumnya: Grady Booch OOD (*Object-Oriented Design*), Jim Rumbaugh OMT (*Object Modeling Technique*), dan Ivar Jacobson OOSE (*Object-Oriented Software Engineering*). Sejarah UML sendiri cukup panjang. Sampai era tahun 1990 seperti kita ketahui puluhan metodologi pemodelan berorientasi objek telah bermunculan di dunia. Diantaranya adalah: *metodologi booch, metodologi coad, metodologi OOSE, metodologi OMT, metodologi shlaer-mellor, metodologi wirfs-brock,* dsb. Masa itu terkenal dengan masa perang metodologi (*method war*) dalam pendesainan berorientasi objek. Masing-masing metodologi membawa notasi sendiri-sendiri, yang mengakibatkan timbul masalah baru apabila kita bekerjasama dengan group/perusahaan lain yang menggunakan metodologi yang berlainan. Dimulai pada bulan Oktober 1994 *Booch, Rumbaugh dan Jacobson,* yang merupakan tiga tokoh yang boleh dikata metodologinya banyak digunakan mempelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek. Pada tahun 1995 direlease draft pertama dari UML (versi 0.8). Sejak tahun 1996 pengembangan tersebut dikoordinasikan oleh Object Management Group (OMG – http://www.omg.org). Tahun 1997 UML versi 1.1 muncul, dan saat ini versi terbaru adalah versi 1.5 yang dirilis bulan Maret 2003. Booch, Rumbaugh dan Jacobson menyusun tiga buku serial tentang UML pada tahun 1999. Sejak saat itulah UML telah menjelma menjadi standar bahasa pemodelan untuk aplikasi berorientasi objek.



Gambar II.4. UML Sebagai Bahasa Standar Pemodelan Aplikasi OOP

Sumber : **(Sri Dharwiyanti: 2006 ; 3)**

II.3.1. Konsepsi Dasar UML

Dari berbagai penjelasan rumit yang terdapat di dokumen dan buku-buku UML. Sebenarnya konsepsi dasar UML bisa kita rangkumkan dalam gambar dibawah.

Major Area	*View*	*Diagrams*	*Main Concepts*
structural	static view	class diagram	class, association, generalization, dependency, realization, interface
	use case view	use case diagram	use case, actor, association, extend, include, use case generalization
	implementation view	component diagram	component, interface, dependency, realization
	deployment view	deployment diagram	node, component, dependency, location
dynamic	state machine view	statechart diagram	state, event, transition, action
	activity view	activity diagram	state, activity, completion transition, fork, join
	interaction view	sequence diagram	interaction, object, message, activation
		collaboration diagram	collaboration, interaction, collaboration role, message
model management	model management view	class diagram	package, subsystem, model
extensibility	all	all	constraint, stereotype, tagged values

Gambar.II.5. Konsep Dasar UML

Sumber : **(Sri Dharwiyanti, 2006 ; 3)**

Seperti juga tercantum pada gambar diatas UML mendefinisikan diagram-diagram sebagai berikut:

1. use case diagram
2. class diagram
3. statechart diagram
4. activity diagram
5. sequence diagram
6. collaboration diagram
7. component diagram
8. deployment diagram

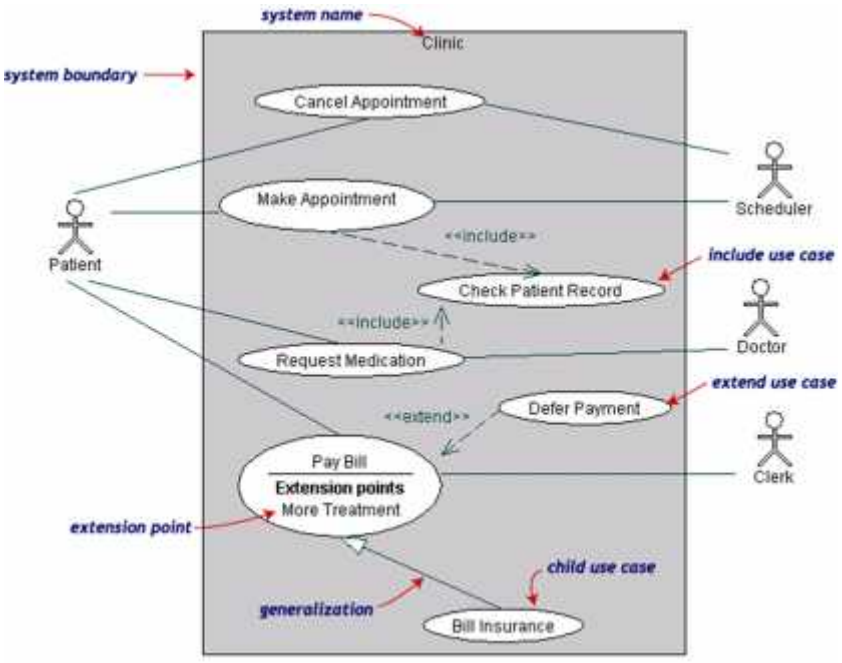
Dalam pembuatan skripsi ini penulis menggunakan diagram Use Case yang terdapat di dalam UML. Adapun maksud dari Use Case Diagram diterangkan dibawah ini.

1. *Use Case Diagram*

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah “apa” yang diperbuat sistem, dan bukan “bagaimana”. Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. *Use case diagram* dapat sangat membantu bila kita sedang menyusun *requirement* sebuah sistem,

mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem. Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal. Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*. Sebuah *use case* juga dapat meng-*extend* *use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain.

Contoh *use case diagram* :



Gambar.II.6.Use Diagram

Sumber : **(Sri Dharwiyanti, 2006 ; 5)**

2. *Class Diagram*

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi).

Class diagram menggambarkan struktur dan deskripsi *class, package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain.

Class memiliki tiga area pokok :

1. Nama (dan stereotype)
2. Atribut
3. Metoda

Atribut dan metoda dapat memiliki salah satu sifat berikut :

a. *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan
b. *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya
c. *Public*, dapat dipanggil oleh siapa saja

Gambar.II.7. Class Diagram

***Sumber :* (Sri Dharwiyanti, 2006 ; 5)**

Class dapat merupakan implementasi dari sebuah *interface*, yaitu *class* abstrak yang hanya memiliki metoda. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class.* Dengan demikian *interface* mendukung resolusi metoda pada saat *run-time*. (Sri Dharwiyanti, 2006; 5)

a. Hubungan Antar *Class*

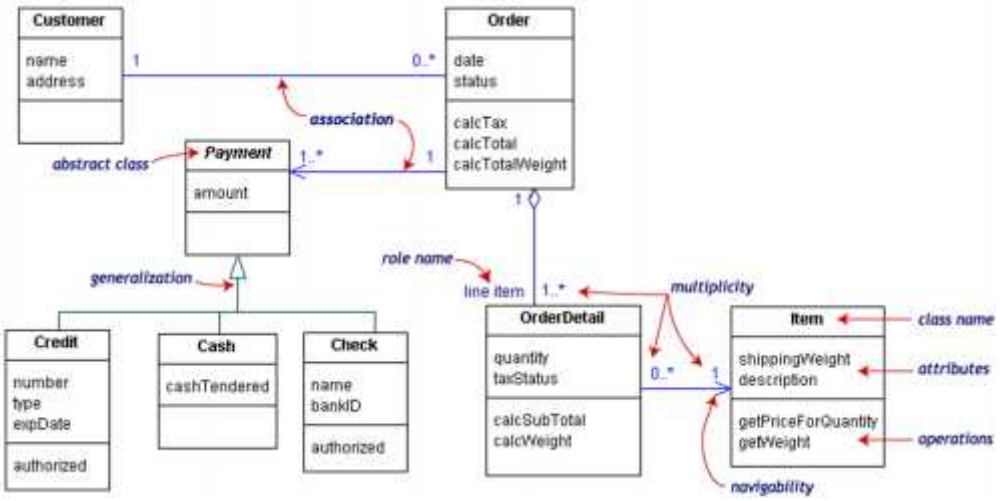
Adapun hubungan atar *class* yang terjadi adalah sebagai berikut :

1. Asosiasi, yaitu hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain. Panah *navigability* menunjukkan arah *query* antar *class*.
2. Agregasi, yaitu hubungan yang menyatakan bagian (“terdiri atas..”).
3. Pewarisan, yaitu hubungan hirarkis antar *class*. *Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metoda *class* asalnya dan

menambahkan fungsionalitas baru, sehingga ia disebut anak dari *class* yang diwarisinya. Kebalikan dari pewarisan adalah generalisasi.

4. Hubungan dinamis, yaitu rangkaian pesan (*message*) yang di-*passing* dari satu *class* kepada *class* lain. Hubungan dinamis dapat digambarkan dengan menggunakan *sequence diagram* yang akan dijelaskan kemudian.

5.



Gambar.II.8.Hubungan antar Class

***Sumber :* (Sri Dharwiyanti, 2006 ; 6)**

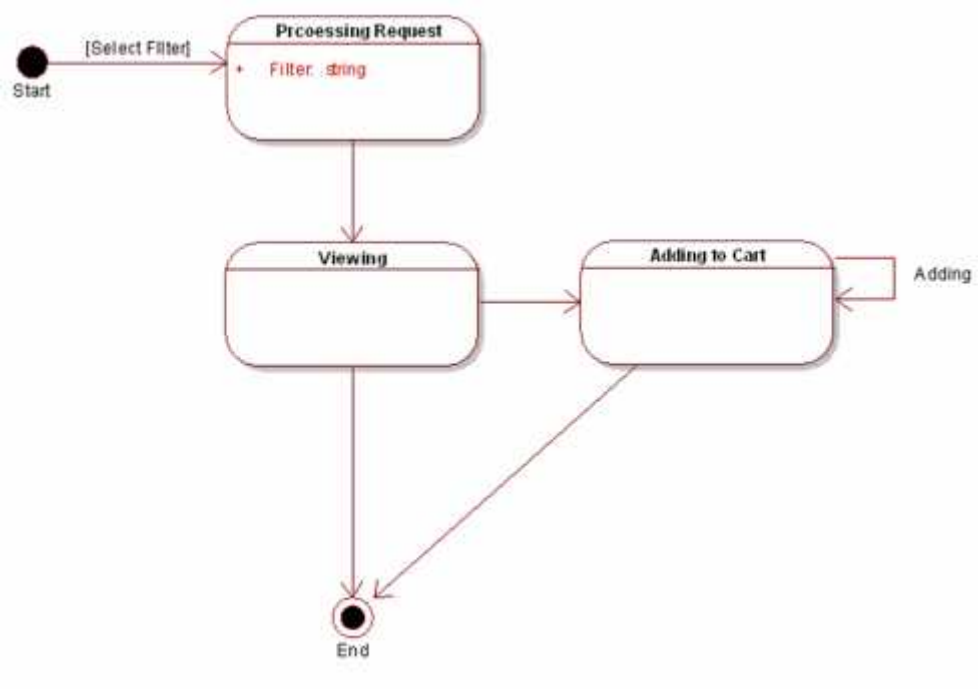
3. *Statechart Diagram*

Statechart diagram menggambarkan transisi dan perubahan keadaan (dari satu *state* ke *state* lainnya) suatu objek pada sistem sebagai akibat dari *stimuli* yang diterima. Pada umumnya *statechart diagram* menggambarkan *class* tertentu (satu *class* dapat memiliki lebih dari satu *statechart diagram*).

Dalam UML, *state* digambarkan berbentuk segiempat dengan sudut membulat dan memiliki nama sesuai kondisinya saat itu. Transisi antar *state*

umumnya memiliki kondisi *guard* yang merupakan syarat terjadinya transisi yang bersangkutan, dituliskan dalam kurung siku. *Action* yang dilakukan sebagai akibat dari *event* tertentu dituliskan dengan diawali garis miring.

Titik awal dan akhir digambarkan berbentuk lingkaran berwarna penuh dan berwarna setengah.



Gambar.II.9. State Diagram

Sumber : **(Sri Dharwiyanti, 2006 ; 7)**

4. *Activity Diagram*

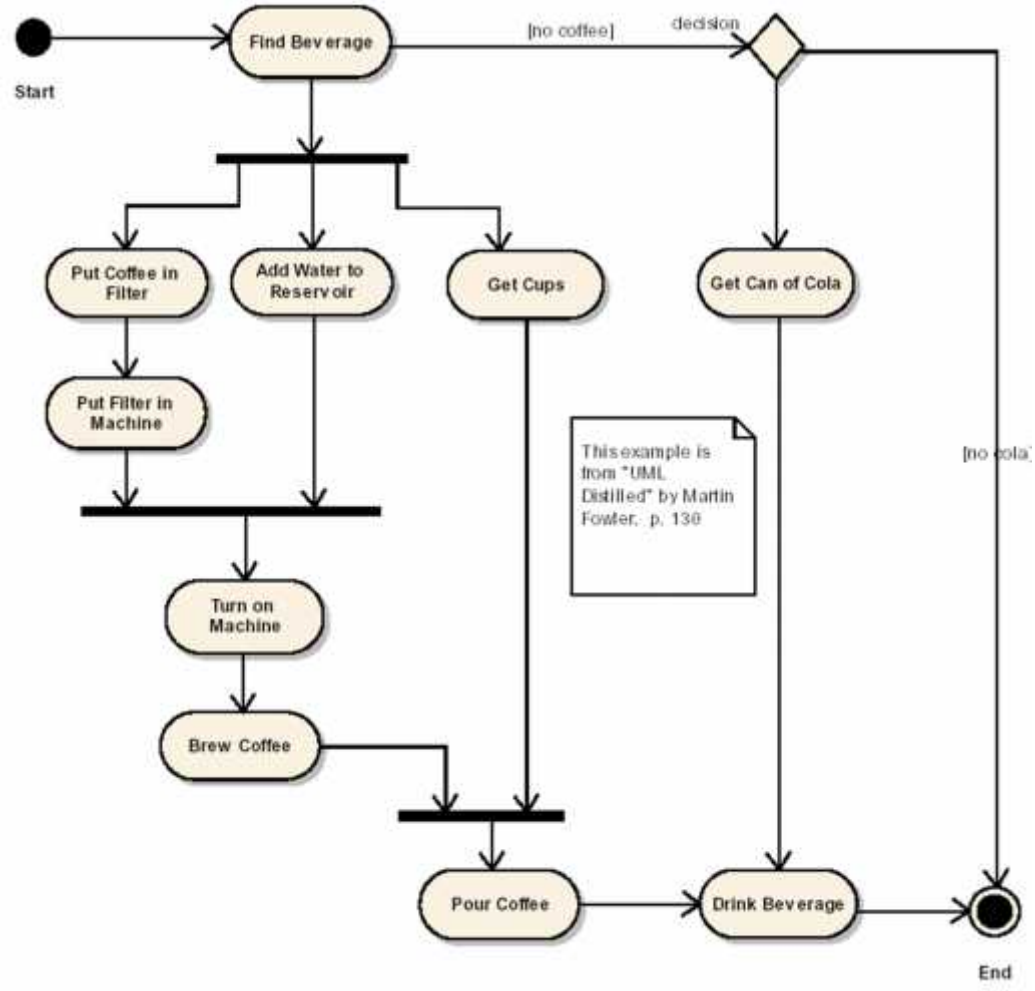
Activity diagrams menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

Activity diagram merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan behaviour internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas.

Sama seperti *state*, standar UML menggunakan segi empat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan behaviour pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal.

Activity diagram dapat dibagi menjadi beberapa *object swimlane* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu.



Gambar.II.10. Contoh *Activity Diagram* State Diagram

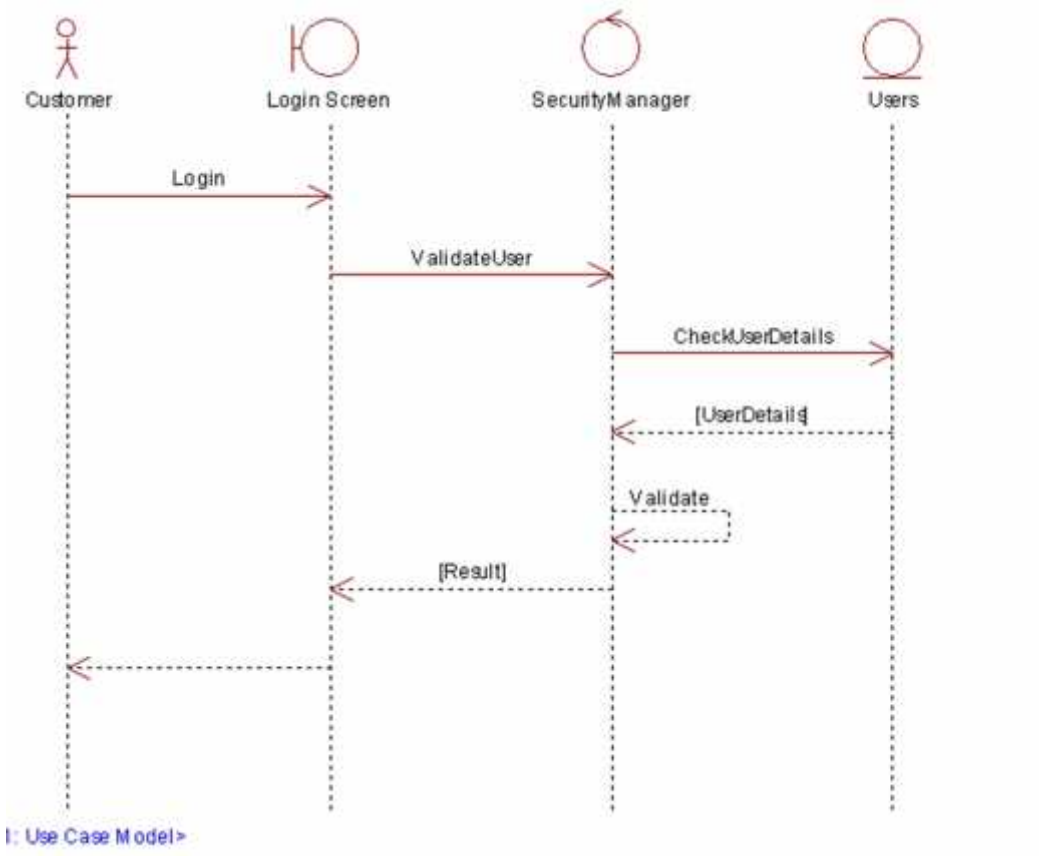
***Sumber :* (Sri Dharwiyanti, 2006 ; 7)**

5. *Sequence Diagram*

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).

Sequence diagram biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan.

Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal. *Message* digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, *message* akan dipetakan menjadi operasi/metoda dari *class*.*Activation bar* menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah message. Untuk objek-objek yang memiliki sifat khusus, standar UML mendefinisikan *icon* khusus untuk objek *boundary*, *controller* dan *persistent entity*.



Gambar.II.11. Contoh *Activity Diagram* State Diagram

***Sumber :* (Sri Dharwiyanti, 2006 ; 8)**

II.4. *Microsoft Visual Studio*

Microsoft Visual Studio merupakan aplikasi pemrograman yang menggunakan teknologi .NET Framework. Dimana teknologi .NET Framework merupakan komponen *Windows* yang terintegrasi serta mendukung pembuatan, penggunaan aplikasi dan halaman web. Teknologi *.NET Framework* mempunyai 2 komponen utama, yaitu CLS (*Common Language Runtime*) dan *Class Library*. CLR digunakan untuk menjalankan aplikasi yang berbasis .NET, sedangkan

Library adalah kelas pustaka atau perintah yang digunakan untuk membangun aplikasi. (Wahana Komputer: 2010).

Visual Studio mencakup kompiler, SDK, *Integrated Development Environment* (IDE), dan dokumentasi (umumnya berupa MSDN Library). Kompiler yang dimasukkan ke dalam paket Visual Studio antara lain Visual C++, Visual C#, Visual Basic, Visual Basic .NET, Visual InterDev, Visual J++, Visual J#, Visual FoxPro, dan Visual SourceSafe. Microsoft Visual Studio dapat digunakan untuk mengembangkan aplikasi dalam *native code* (dalam bentuk bahasa mesin yang berjalan di atas Windows) ataupun *managed code* (dalam bentuk *Microsoft Intermediate Language* di atas *.NET Framework*). Selain itu, Visual Studio juga dapat digunakan untuk mengembangkan aplikasi Silverlight, aplikasi *Windows Mobile* (yang berjalan di atas *.NET Compact Framework*).

Visual Studio kini telah menginjak versi *Visual Studio* 9.0.21022.08, atau dikenal dengan sebutan *Microsoft Visual Studio 2008* yang diluncurkan pada 19 November 2007, yang ditujukan untuk *platform Microsoft .NET Framework 3.5*. Versi sebelumnya, *Visual Studio 2005* ditujukan untuk platform *.NET Framework 2.0 dan 3.0. Visual Studio 2003* ditujukan untuk *.NET Framework 1.1*, dan *Visual Studio 2002* ditujukan untuk *.NET Framework 1.0.* Versi-versi tersebut di atas kini dikenal dengan sebutan *Visual Studio .NET*, karena memang membutuhkan *Microsoft .NET Framework*. Sementara itu, sebelum muncul *Visual Studio .NET*, terdapat *Microsoft Visual Studio 6.0* (VS1998).