BAB II

TINJAUAN PUSTAKA

II.1. Perancangan

Perancangan adalah suatu proses pemilihan dan pemikiran yang menghubungkan fakta-fakta berdasarkan asumsi-asumsi yang berkaitan dengan masa datang dengan menggambarkan dan merumuskan kegiatan-kegiatan tertentu yang diyakini diperlukan untuk mencapai tujuan-tujuan tertentu dan menguraikan bagaimana pencapaiannya. (*Anggraheni Rukmana ; 2011 : 2*)

II.2. Aplikasi

Aplikasi berasal dari kata *application* yang artinya penerapan, lamaran, penggunaan. Secara istilah aplikasi adalah program siap pakai yang direka untuk melaksanakan suatu fungsi bagi pengguna atau aplikasi yang lain dan dapat digunakan oleh sasaran yang dituju. Perangkat lunak aplikasi adalah suatu subkelas perangkat lunak komputer yang memanfaatkan kemampuan komputer langsung untukmelakukan tugas yangdiinginkan pengguna. Contoh utama perangkat lunak aplikasi adalah pengolah kata, lembar kerja ,dan pemutar media. (*Fricles Ariwisanto Sianturi ; 2013 : 43*)

II.3. Kriptografi

Kriptografi berasal dari bahasa Yunani yaitu *cryptós* yang artinya "*secret*" (yang tersembunyi) dan *gráphein* yang artinya "*writting*" (tulisan). Jadi, kriptografi berarti "*secret writting*" (tulisan rahasia). Definisi yang

dikemukakan oleh Bruce Schneier (1996), kriptografi adalah ilmu dan seni untuk menjaga keamanan pesan (*Cryptography is the art and science of keeping messages secure*). (*Suriski Sitinjak ; 2010 : C-78*)

II.3.1. Terminologi Kriptografi

Ada beberapa istilah-istilah yang penting dalam kriptografi menurut (*Suriski Sitinjak ; 2010 : C-78*), yaitu :

1. Pesan (*Plaintext* dan *Ciphertext*)

   Pesan (*message*) adalah data atau informasi yang dapat dibaca dan dimengerti maknanya. Pesan asli disebut plainteks (*plaintext*) atau teks-jelas (*cleartext*). Sedangkan pesan yang sudah disandikan disebut cipherteks (*chipertext*).

2. Pengirimdan Penerima

   Komunikasi data melibatkan pertukaran pesan antara dua entitas. Pengirim (*sender*) adalah entitas yang mengirim pesan kepada entitaslainnya. Penerima (*receiver*) adalah entitas yang menerima pesan.

3. Penyadap (eavesdropper)

   adalah orang yang mencoba menangkap pesan selama ditransmisikan.

4. Kriptanalisis dan Kriptologi

   Kriptanalisis (*cryptanalysis*) adalah ilmu dan seni untuk memecahkan chiperteks menjadi plainteks tanpa mengetahui kunci yang digunakan. Pelakunya disebut kriptanalis. Kriptologi (*cryptology*) adalah studi mengenai kriptografi dan kriptanalisis.

5. Enkripsi dan Dekripsi

   Proses menyandikan plainteks menjadi cipherteks disebut enkripsi (*encryption*) atau *enciphering*. Sedangkan proses mengembalikan cipherteks menjadi plainteks semula dinamakan dekripsi (*decryption*) atau *deciphering*.

6. Cipher dan Kunci

   Algoritma kriptografi disebut juga *cipher* yaitu aturan untuk *enchipering* dan *dechipering*, atau fungsi matematika yang digunakan untuk enkripsi dan dekripsi. Kunci (*key*) adalah parameter yang digunakan untuk transformasi *enciphering* dan *dechipering*. Kunci biasanya berupa string atau deretan bilangan. (*Suriski Sitinjak ; 2010 : C-78*)

II.4. Algoritma *Blowfish*

Blowfish diciptakan oleh seorang *Cryptanalyst* bernama Bruce Schneier, Presiden perusahaan *Counterpane Internet Security, Inc* (Perusahaan konsultan tentang kriptografi dan keamanan komputer) dan dipublikasikan tahun 1994. Dibuat untuk digunakan pada komputer yang mempunyai *microposesor* besar (32-bit keatas dengan *cache* data yang besar). *Blowfish* merupakan algoritma yang tidak dipatenkan dan *licensefree*, dan tersedia secara gratis untuk berbagai macam kegunaan. (*Suriski Sitinjak ; 2010 : C-79*)

Pada saat *Blowfish* dirancang, diharapkan mempunyai kriteria perancangan sebagai berikut (*Suriski Sitinjak ; 2010 : C-79*):

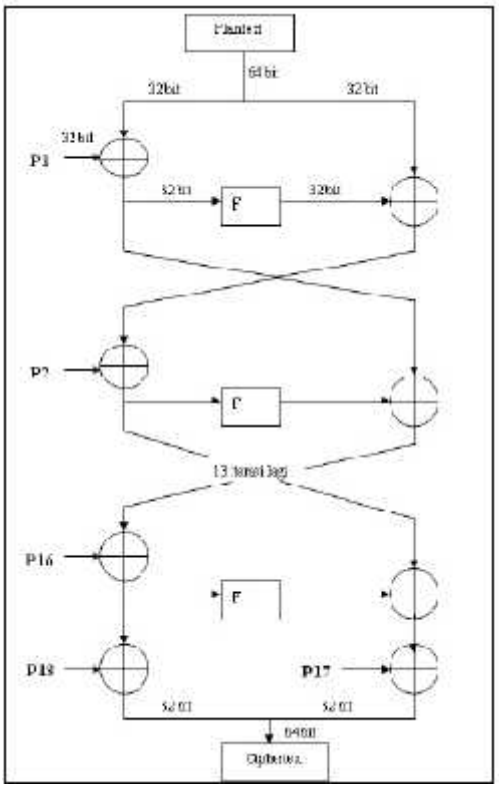
1. Cepat, *Blowfish* melakukan enkripsi data pada *microprocessors* 32-bit dengan *rate26 clock cycles* per *byte*.
2. *Compact* (ringan), *Blowfish* dapat dijalankan pada memori kurang dari 5K.

3. Sederhana, *Blowfish* hanya menggunakan operasi-operasi sederhana: penambahan, XOR, dan lookup tabel pada operan 32-bit.
4. Memiliki tingkat keamanan yang bervariasi, panjang kunci yang digunakan oleh *Blowfish* dapat bervariasi dan bisa sampai sepanjang 448 bit.

Dalam penerapannya sering kali algortima ini menjadi tidak optimal. Karena strategi implementasi yang tidak tepat. Algoritma *Blowfish* akan lebih optimal jika digunakan untuk aplikasi yang tidak sering berganti kunci, seperti jaringan komunikasi atau enkripsi *file* otomatis. Selain itu, karena algoritma ini membutuhkan memori yang besar, maka algoritma ini tidak dapat diterapkan untuk aplikasi yang memiliki memori kecil seperti *smart card*. Panjang kunci yang digunakan, juga mempengaruhi keamanan penerapan algoritma ini. Algoritma *Blowfish* terdiri atas dua bagian, yaitu ekspansi kunci dan enkripsi data. (*Suriski Sitinjak ; 2010 : C-79*)

II.4.1. Enkripsi *Blowfish*

Berikut ini merupakan contoh algoritma blowfish yang digambarkan pada blok diagram. Adapun gambar tersebut dapat dilihat dibawah ini.



Gambar II.1. Blok Diagram Enkripsi Algoritma *Blowfish*
Sumber : (*Suriski Sitinjak ; 2010 : C-81*)

Pada langkah kedua, telah dituliskan mengenai penggunaan fungsi F. Fungsi F adalah: bagi XL menjadi empat bagian 8-bit: a,b,c dan d.

$F(XL) = ((S1,a + S2,b \bmod 2^{32}) \text{ XOR } S3,c) + S4,d \bmod 2^{32}$.

II.4.2. Ekspansi kunci (*Key-expansion*)

Berfungsi merubah kunci (minimum 32-bit, maksimum 448-bit) menjadi beberapa *array* subkunci (*subkey*) dengan total 4168 *byte* (18x32-bit untuk P-*array* dan 4x256x32-bit untuk S-*box* sehingga totalnya 33344 bit atau 4168 *byte*). Kunci disimpan dalam K-*array* (*Suriski Sitinjak ; 2010 : C-80*):

K1, K2, ... Kj $1 \leq j \leq 14$

Kunci-kunci ini yang dibangkitkan (*generate*) dengan menggunakan subkunci yang harus dihitung terlebih dahulu sebelum enkripsi atau dekripsi data. Sub-subkunci yang digunakan terdiri dari :

P-array yang terdiri dari 18 buah 32-bit subkunci,

P1, P2, ..., P18

S-*box* yang terdiri dari 4 buah 32-bit, masing-masing memiliki 256 entri :

S1,0, S1,1, ..., S1,255

S2,0, S2,1, ..., S2,255

S3,0, S3,1, ..., S3,255

S4,0, S4,1, ..., S4,255

Langkah-langkah perhitungan atau pembangkitan subkunci tersebut adalah sebagai berikut (*Suriski Sitinjak ; 2010 : C-80*):

1. Inisialisasi P-*array* yang pertama dan juga empat S-*box*, berututan, dengan *string* yang telah pasti. *String* tersebut terdiri dari digit-digit heksadesimal dari phi, tidak termasuk angka tiga di awal.

   Contoh :

   P1= 0x243f6a88

   P2= 0x85a308d3

   P3= 0x13198a2e

   P4= 0x03707344

   dan seterusnya sampai S-*box* yang terakhir (daftar heksadesimal digit dari phi untuk P-*array* dan *Sbox* bisa lihat Lampiran).

2. XOR-kan P1 dengan 32-bit awal kunci, XOR-kan P2 dengan 32-bit berikutnya dari kunci, dan seterusnya untuk semua bit kunci. Ulangi siklus seluruh bit kunci secara berurutan sampai seluruh P-*array* ter-XOR-kan dengan bit-bit kunci. Atau jika disimbolkan : P1 = P1 ⊕K1, P2 = P2 ⊕ K2, P3 = P3 ⊕K3, . . . P14 = P14 ⊕K14, P15 = P15 ⊕K1, . . . P18 = P18 ⊕K4.
   Keterangan : ⊕adalah simbol untuk XOR.
3. Enkripsikan *string* yang seluruhnya nol (*all-zero string*) dengan algoritma *Blowfish*, menggunakan subkunci yang telah dideskripsikan pada langkah 1 dan 2.
4. Gantikan P1 dan P2 dengan keluaran dari langkah 3.
5. Enkripsikan keluaran langkah 3 menggunakan algoritma *Blowfish* dengan subkunci yang telah dimodifikasi.
6. Gantikan P3 dan P4 dengan keluaran dari langkah 5.
7. Lanjutkan langkah-langkah di atas, gantikan seluruhelemen P-*array* dan kemudian keempat S-*box* secara berurutan, dengan hasil keluaran algoritma *Blowfish* yang terus-menerus berubah.

Total keseluruhan, terdapat 521 iterasi untuk menghasilkan subkunci-subkunci dan membutuhkan memori sebesar 4KB. (*Suriski Sitinjak ; 2010 : C-80*)

II.4.3. Enkripsi Data

Terdiri dari iterasi fungsi sederhana (*Feistel Network*) sebanyak 16 kali putaran (iterasi), masukannya adalah 64-bit elemen data X. Setiap putaran terdiri dari permutasi kunci-*dependent* dan substitusi kunci- dan data

dependent. Semua operasi adalah penambahan (*addition*) dan XOR pada variabel 32-bit. Operasi tambahan lainnya hanyalah empat penelusuran tabel *array* berindeks untuk setiap putaran. Langkahnya adalah seperti berikut. (*Suriski Sitinjak ; 2010 : C-80*)

1. Bagi X menjadi dua bagian yang masing-masing terdiri dari 32-bit: XL, XR.
2. Lakukan langkah berikut

   For i = 1 to 16:

   XL= XL ⊕Pi

   XR= F(XL) ⊕XR

   Tukar XLdan XR
3. Setelah iterasi ke-16, tukar XLdan XR lagi untuk melakukan membatalkan pertukaran terakhir.
4. Lalu lakukan

   XR= XR ⊕P17

   XL= XL ⊕P18
5. Terakhir, gabungkan kembali XLdan XR untuk mendapatkan cipherteks. (*Suriski Sitinjak ; 2010 : C-80*)

II.5. PHP

PHP adalah bahasa *server side scripting* yang menyatu dengan HTML untuk membuat halaman *web* yang dinamis. Maksud dari istilah *server side scripting* adalah sintaks dan perintah yang diberikan akan sepenuhnya dijalankan di *server* tetapi disertakan pada dokumen HTML. Pembuatan *web* sendiri merupakan kombinasi antara PHP sebagai bahasa pemrograman dan HTML

sebagai pembangun halaman *web*. Ketika seorang pengguna *internet* akan membuka suatu situs yang menggunakan fasilitas *server side scripting* PHP, terlebih dahulu *server* yang bersangkutan akan memproses semua perintah PHP di server lalu mengirimkan hasilnya dalam format HTML ke *web browser* pengguna *internet* tadi. Seorang pengguna internet tidak dapat melihat kode program yang ditulis dalam PHP sehingga keamanan dari halaman *web* menjadi lebih terjamin.

PHP bersifat *open source*, sehingga untuk mendapatkannya kita tidak perlu membayar lisensi. Dasar pertimbangan untuk mengembangkan kemampuan pemrograman berorientasi objek pada PHP adalah dengan melihat perkembangan aplikasi *web* sebagai sebuah *platform* yang terus meluas dengan cepat sehinggga aplikasi *web* yang dibangun juga menjadi semakin besar, rumit, dan kompleks. Aplikasi *web* telah diimplementasikan mulai dari tingkatan yang paling sederhana seperti berita online hingga ke tingkatan *enterprise* seperti aplikasi *online banking*. PHP dapat mengirim HTTP *header*, dapat mengeset *cookies*, mengatur *authentication*. PHP menawarkan koneksitas yang baik dengan beberapa basis data, antara lain *Oracle*, *Sybase*, *mSQL*, *MySQL*, *Solid*, *PostgreSQL*, *Adabas*, *Velocis*, *dBase* dan semua *database berinterface* ODBC. PHP juga berintegrasi dengan beberapa external *library* sehingga pengguna dapat membuat dokumen PDF. (*R. Arum ; 2012 : 3*)

II.6. UML (*Unifed Modelling Language*)

Unified Modelling Language (UML) adalah sebuah "bahasa" yang 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

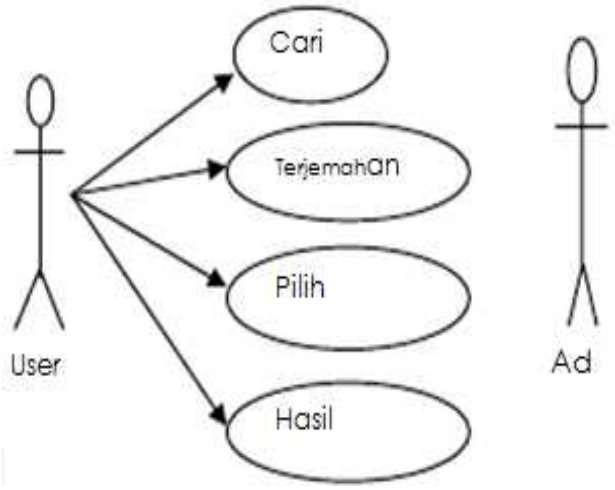
Booch, Rumbaugh dan Jacobson, yang merupakan tiga tokoh yang boleh dikata metodologinya banyak digunakan mempelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek. Pada tahun 1995 direlease *draft* pertama dari UML (versi 0.8). Sejak tahun 1996 pengembangan tersebut dikoordinasikan 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. *(Yuni Sugiarti ; 2013 : 33)*

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

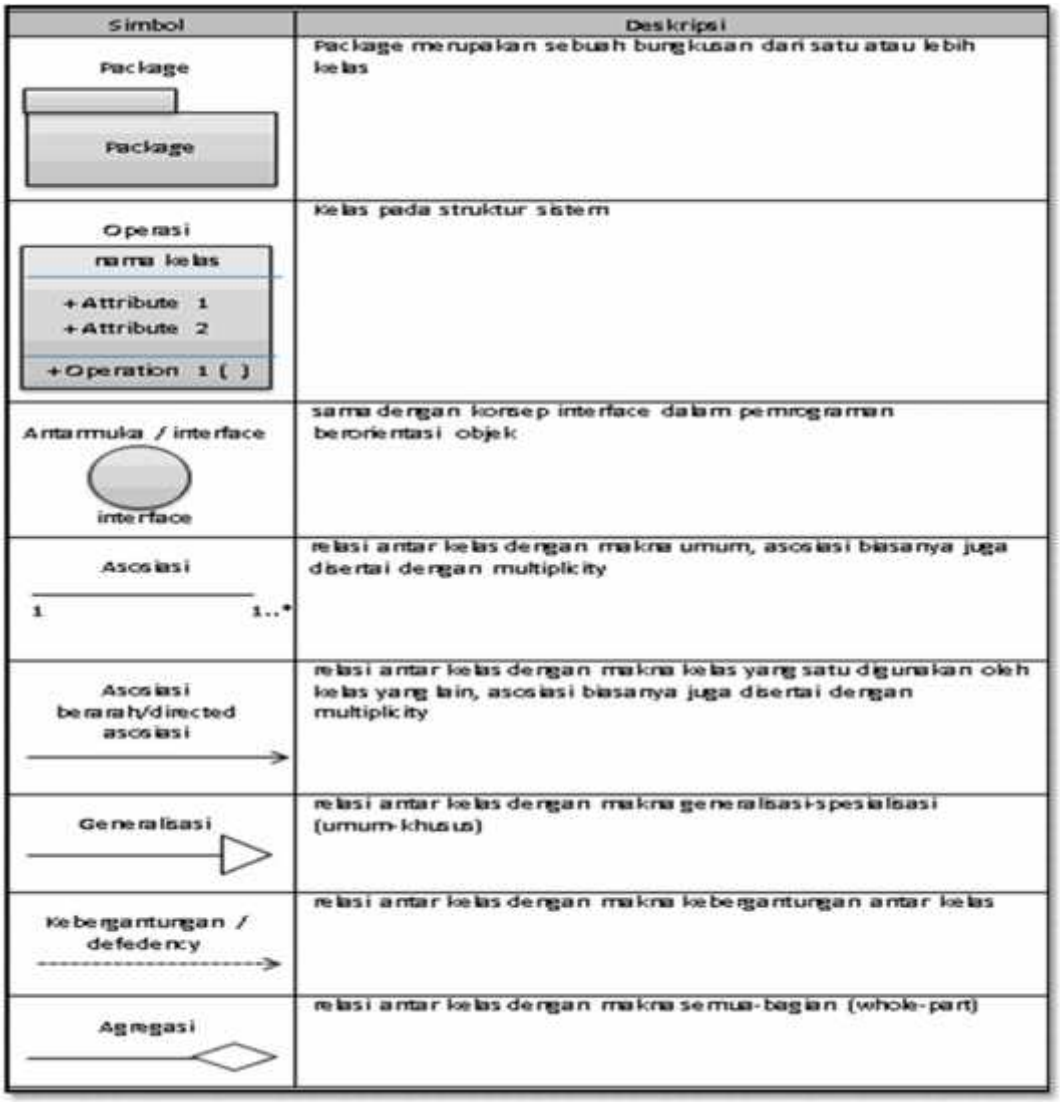
semua *feature* yang ada pada sistem. Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal. Sebuah *use 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. *(Yuni Sugiarti ; 2013 : 41)*



Gambar II.2. Contoh *Use Case* Diagram
Sumber : (*Junaedi Siregar ; 2013 : 76*)

2. *Class Diagram*

Diagram kelas atau *class* diagram menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi. Berikut adalah simbol-simbol pada diagram kelas :



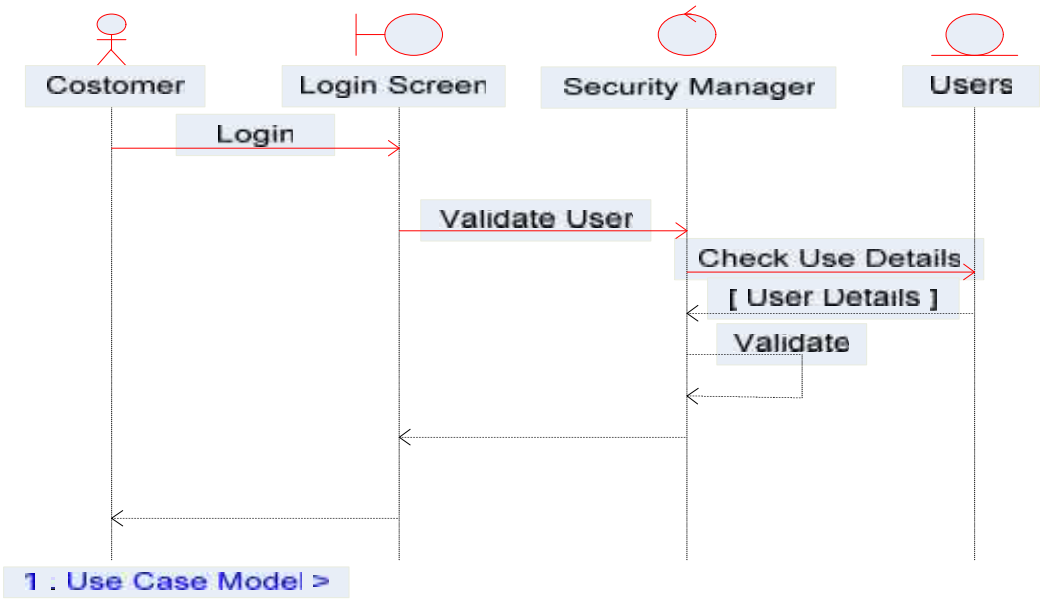
Simbol	Deskripsi
Package Package	Package merupakan sebuah bungkusan dari satu atau lebih kelas
Operasi nama kelas +Attribute 1 +Attribute 2 +Operation 1 ( )	Kelas pada struktur sistem
Antarmuka / interface interface	sama dengan konsep interface dalam pemrograman berorientasi objek
Asosiasi 1 1..*	relasi antar kelas dengan makna umum, asosiasi biasanya juga disertai dengan multiplicity
Asosiasi berarah/directed asosiasi	relasi antar kelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai dengan multiplicity
Generalisasi	relasi antar kelas dengan makna generalisasi-spesialisasi (umum-khusus)
Kebergantungan / defedency	relasi antar kelas dengan makna kebergantungan antar kelas
Agregasi	relasi antar kelas dengan makna semua-bagian (whole-part)

Gambar II.3. *Class* Diagram
Sumber : (Yuni Sugiarti ; 2013 : 59)

3. *Sequence Diagram*

Diagram *Sequence* menggambarkan kelakuan/prilaku objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambarkan diagram *sequence* maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu.

Banyaknya diagram *sequence* yang harus digambar adalah sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram *sequence* sehingga semakin banyak *use case* yang didefinisikan maka diagram *sequence* yang harus dibuat juga semakin banyak.



Gambar II.4. *Contoh Sequence Diagram*
Sumber : (Yuni Sugiarti ; 2013 : 63)

4. *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.

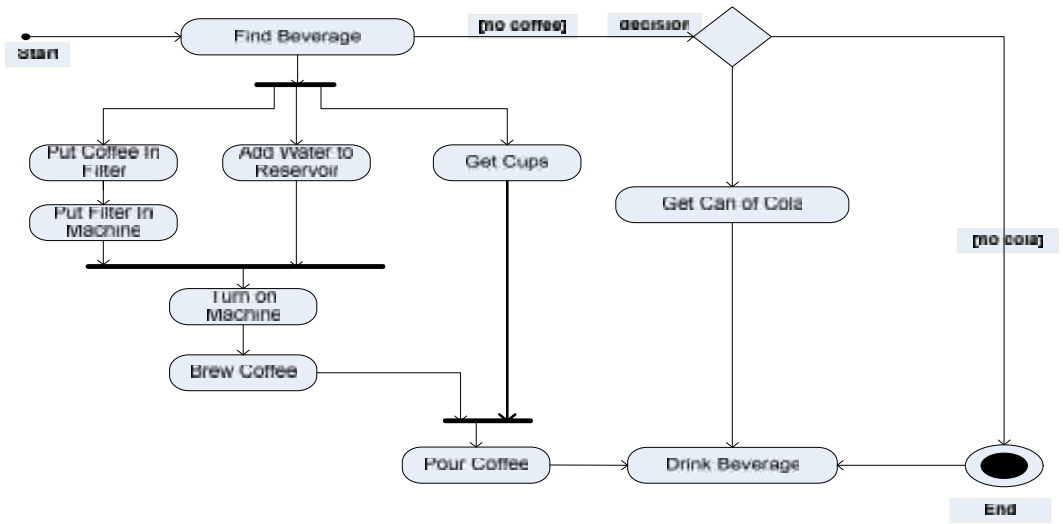
Activity diagram merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak

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 segiempat 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.5. *Activity* Diagram
Sumber : (Yuni Sugiarti ; 2013 : 76)