BAB II

TINJAUAN PUSTAKA

II.1. Keamanan Komputer

Keamanan komputer yang dianggap tidak penting oleh banyak orang ternyata dapat membawa kerugian besar bagi orang tersebut. Jika ditelaah lebih lanjut ternyata banyak hal – hal yang dapat mengancam data – data pribadi seseorang melalui media komputer. Saat ini sudah banyak program – program yang bersifat jahat seperti virus, worm, trojan dan lainnya yang bila menginfeksi sistem komputer kita dapat mengakibatkan kerugian bagi kita. (*Tri Wahyu W ; 2008 : 70*)

Computer Security adalah bagian dari ilmu komputer yang bertugas untuk mengontrol resiko yang berhubungan dengan penggunaan komputer. *Computer Security* yang dimaksud adalah keamanan sebuah komputer yang terhubung ke dalam sebuah jaringan (*Internet*), dari akses yang tidak memiliki hak untuk mencoba masuk untuk memperoleh informasi dan *service* tertentu yang ada di dalam sistem. Usaha untuk mengakses paksa ini terdapat banyak macamnya, baik itu *intrusion* (serangan dari luar organisasi) atau *misuse* (serangan dari dalam organisasi), dengan *level hacker* (hanya mencoba masuk ke dalam sistem komputer) atau bahkan *cracker* (mencoba masuk dan merusak untuk keuntungan pribadi). (*Tri Wahyu W ; 2008 : 71*)

II.1.1. Aspek-Aspek Keamanan Komputer

- **Confidentiality**

  Informasi (data) hanya bisa diakses oleh pihak yang memiliki wewenang.

- **Integrity**

  Informasi hanya dapat diubah oleh pihak yang memiliki wewenang.

- **Availability**

Informasi tersedia untuk pihak yang memiliki wewenang ketika dibutuhkan.

- **Authentication**

  Pihak yang terlibat dengan pertukaran informasi dapat diidentifikasi dengan benar dan ada jaminan bahwa identitas yang didapat tidak palsu.

- **Nonrepudiation**

  Pengirim maupun penerima informasi tidak dapat menyangkal pengiriman dan penerimaan pesan. (*Fauziah ; 2009 : 43*)

II.2. Algoritma Kriptografi

Algoritma kriptografi merupakan langkah-langkah logis bagaimana menyembunyikan pesan dari orang-orang yang tidak berhak atas pesan tersebut. Algoritma kriptografi terdiri dari tiga fungsi dasar :

1. Enkripsi

   Merupakan hal yang sangat penting dalam kriptografi, merupakan pengamanan data yang dikirim agar terjaga kerahasiaannya. Pesan asli disebut *plaintext*, yang diubah menjadi kode-kode yang tidak dimengerti. Enkripsi bisa diartikan sebagai *cipher* atau kode dengan menggunakan algoritma yang untuk mengkodekan data yang kita inginkan.

2. Dekripsi

   Merupakan kebalikan dari proses enkripsi. Pesan yang telah dienkripsi dikembalikan ke bentuk asalnya (teks asli), disebut dengan dekripsi pesan. Algoritma yang digunakan untuk dekripsi tentu berbeda dengan algoritma yang digunakan untuk enkripsi.

3. Kunci

   Kunci adalah kunci yang dipakai untuk melakukan enkripsi dan dekripsi. Kunci terbagi menjadi duabagian, kunci rahasia (*private key*) dan kunci umum (*public key*).

II.2.1. Kriptografi

Kriptografi adalah ilmu yang mempelajari teknik-teknik matematis yang berhubungan dengan aspek keamanan informasi seperti : keabsahan, integritas data, serta autentifikasi data. Kriptografi tidak berarti hanya memberikan keamanan informasi saja, namun lebih ke arah teknik-tekniknya. (*Eko Nur Zhafar ; 2011 : 5*)

Pengamanan pesan, data, atau informasi selain bertujuan untuk meningkatkan keamanan, juga berfungsi untuk:

1. Melindungi pesan, data, atau informasi agar tidak dapat dibaca oleh orang yang tidak berhak.
2. Mencegah agar orang – orang yang tidak berhak menyisipkan atau menghapus pesan, data, atau informasi.

Informasi dibagi menjadi dua bagian yaitu informasi yang bersifat umum dan informasi yang bersifat pribadi. Pada informasi yang bersifat pribadi maksudnya informasi yang terkandung hanya untuk satu orang sedangkan informasi yang bersifat umum artinya dapat diketahui orang banyak. Adapun perjalanan informasi tersebut tidak luput dari ganguan – ganguan pihak yang tidak berhak. Salah satu ilmu untuk menjaga keamanan dan kerahasiaan data atau informasi yaitu kriptografi.

Ada banyak model dan metodelogi enkripsi, salah satunya adalah enkripsi dengan *Riverst Code* 6 (*RC6*). Model ini merupakan salah satu kandidat *Advenced Encryption Standart* (*AES*) yang diajukan *RSA Security Laboratories* kepada IST. Algoritma ini adalah pengembangan darialgoritma sebelumnya adalah *RC*5 dan telah memenuhi kriteria dari NIST.

Setiap kriptografi atau cryptosystem yang baik memiliki karakteristik sebagai berikut:

1. Keamanan sistem terletak pada kerahasiaan kunci.

2. pada kerahasiaan algoritma yang digunakan.
3. *Cryptosystem* yang baik memiliki ruang kunci (*keysspace*) yang besar.
4. *Cryptosystem* yang baik akan menghasilkan *chipertext* yang terlihat acak dalam seluruh tes static yang dilakukan.
5. *Cryptosystem* yang baik mampu menahan seluruh serangan yang telah dikenal sebelumnya. (*Defni ; 2014 : 66-67*)

II.2.2. Tujuan Ilmu Kriptografi

Ada empat tujuan dari ilmu kriptografi, yaitu :

1. kerahasiaan, adalah layanan yang digunakan untuk menjaga isi dari informasi dari siapapun kecuali yang memiliki otoritas
2. integritas data, adalah berhubungan dengan penjagaan dari perubahan data secara tidak sah. Untuk menjaga integritas data, sistem harus memiliki kemampuan untuk mendeteksi manipulasi data oleh pihak-pihak yang tidak berhak, antara lain menyangkut penyisipan, penghapusan, dan pensubtitusian data lain ke dalam data yang sebenarnya
3. autentikasi, adalah berhubungan dengan identifikasi, baik secara kesatuan sistem maupun informasi itu sendiri. Dua pihak yang saling berkomunikasi harus saling memperkenalkan diri. Informasi yang dikirimkan melalui kanal harus diautentikasi keaslian, isi datanya, waktu pengiriman, dan lain-lain
4. non-repudiasi, yang berarti begitu pesan terkirim, maka tidak akan dapat dibatalkan. (*Eko Nur Zhafar ; 2011 : 6*)

II.3. Algoritma RC4

RC4 merupakan jenis *stream cipher* yang mempunyai sebuah *S-Box*, S0,S1,...,S255, yang berisi permutasi dari bilangan 0 sampai 255, dan permutasi merupakan fungsi dari kunci dengan panjang yang variabel. Algoritma RC4 memiliki dua fase, yaitu *setup* kunci dan pengenkripsian. Dengan kunci yang sama maka pada proses dekripsi data kembali ke bentuk semula. Sampai saat ini RC4 masih banyak digunakan orang dalam mengenkripsi informasi berupa data teks karena algoritmanya yang sederhana namun memiliki kecepatan yang hampir sama dibandingkan algoritma yang lebih rumit. Data tersebut berupa .txt yang dienkrip per karakter dengan sebuah kunci yang mengubah plainteks menjadi cipherteks. (*Busran ; 2012 : 34*)

II.4. Algoritma *Triangle Chain*

Algoritma kriptografi *triangle chain* atau umumnya dikenal dengan sebutan rantai segitiga merupakan *cipher* yang ide awalnya dari algoritma kriptografi *One Time Pad*, yaitu kunci yang dibangkitkan secara *random* dan panjang kunci sepanjang plainteks yang akan dienkripsi. Tetapi pada algoritma kriptogarfi rantai segitiga pembangkitan kunci-kunci tersebut secara otomatis dengan teknik berantai.

Algoritma rantai segitiga ini memiliki aturan substitusi berdasar pada caesar *cipher* yaitu dengan pergeseran huruf-huruf. Kekuatan *cipher* ini terletak pada kunci yaitu nilai integer yang menunjukkan pergeseran karakter-karakter sesuai dengan operasi pada caesar *cipher*. Kekuatan kedua terletak pada barisan bilangan-bilangan yang berfungsi sebagai pengali dengan kunci. Barisan bilangan tersebut dapat berupa bilangan tertentu seperti deret bilangan ganjil, deret bilangan genap, deret *fibonancci*, deret bilangan prima, serta deret bilangan yang dapat dibuat sendiri.

Pada kenyataannya *cipher* substitusi segitiga tidak dibuat secara sederhana, tetapi dengan mengenkripsi ganda (mengenkripsi dua kali), jadi plainteks dienkripsi dengan *cipher* segitiga I, kemudian hasil enkripsi pertama dienkripsi kembali dengan *cipher* segitiga II yang arah segitiga II merupakan kebalikan arah segitiga I.

Untuk itu maka standar untuk *cipher* segitiga ini adalah *cipher* segitiga ganda yaitu *cipher* rantai segitiga yang melakukan enkripsi ganda, yaitu dengan membuat pola enkripsi pertama dengan mengerucut ke arah kanan dan enkripsi kedua mengerucut ke arah kiri.

Secara matematis pola enkripsi rantai segitigadapat digambarkan dengan matriks N x N dengan N merupakan panjang plainteks yang akan dienkripsi dan operasi pada alfabet ASCII. (*Taronisoki Zebua ; 2013 : 39*)

II.5. *Java*

Java memiliki cara kerja yang unik dibandingkan dengan bahasa perograman lainya yaitu bahasa perograman java bekerja mengunakan *interptreter* dan juga *compiler* dalam proses pembuatan program, *Interpreter* java dikenal sebagai perograman *bytecode* yaitu dengan cara kerja mengubah paket class pada java dengan extensi .java menjadi .class, hal ini dikenal sebagai class *bytecode*, yaitunya class yang dihasilkan agar program dapat dijalankan pada semua jenis perangkat dan juga *platform*, sehingga program java cukup ditulis sekali namun mampu bekerja pada jenis lingkungan yang berbeda. (*Defni ; 2014 : 64*)

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

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

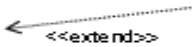
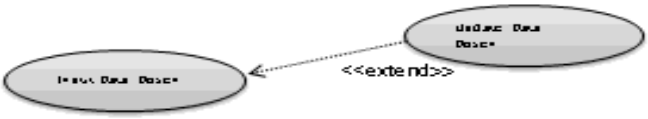
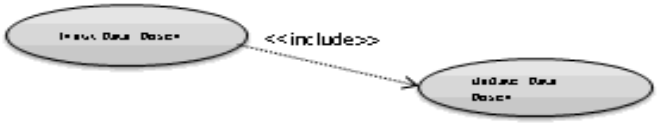
(*Object-Oriented Design*), Jim Rumbaugh OMT (*Object Modeling Technique*), dan Ivar Jacobson OOSE (*Object-Oriented Software Engineering*). Sejarah UML sendiri cukup panjang. Sampai era tahun 1990 seperti kita ketahui puluhan metodologi pemodelan berorientasi objek telah bermunculan di dunia. Diantaranya 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. *(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)*

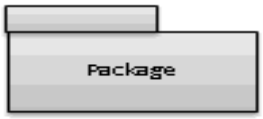
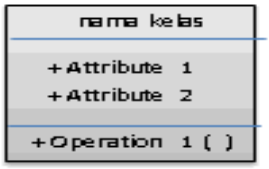
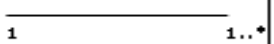
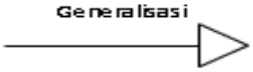
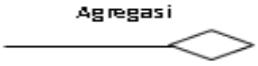
Simbol	Deskripsi
Use Case 	fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit dan aktor; biasanya dinyatakan dengan menggunakan kata kerja di awal frase nama use case
Aktor  nama aktor	orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang; biasanya dinyatakan menggunakan kata benda di awal frase nama aktor
Asosiasi / association	komunikasi antara aktor dan use case yang berpartisipasi pada use case atau use case memiliki interaksi dengan aktor
Extend  <<extend>>	relasi use case tambahan ke sebuah use case dimana use case yang ditambahkan dapat berdiri sendiri walaupun tanpa use case tambahan itu; mirip dengan prinsip inheritance pada pemrograman berorientasi objek; biasanya use case tambahan memiliki nama depan yang sama dengan use case yang ditambahkan, arah panah menunjukan pada use case yang dituju contoh : 
Include <<include>> 	relasi use case tambahan ke sebuah use case dimana use case yang ditambahkan memerlukan use case ini untuk menjalankan fungsinya atau sebagai syarat dijalankan use case ini. Ada dua sudut pandang yang cukup besar mengenai include di use case, include berarti use case yang ditambahkan akan selalu dipanggil saat use case tambahan dijalankan, contoh : 

Gambar II.1. *Use Case* Diagram

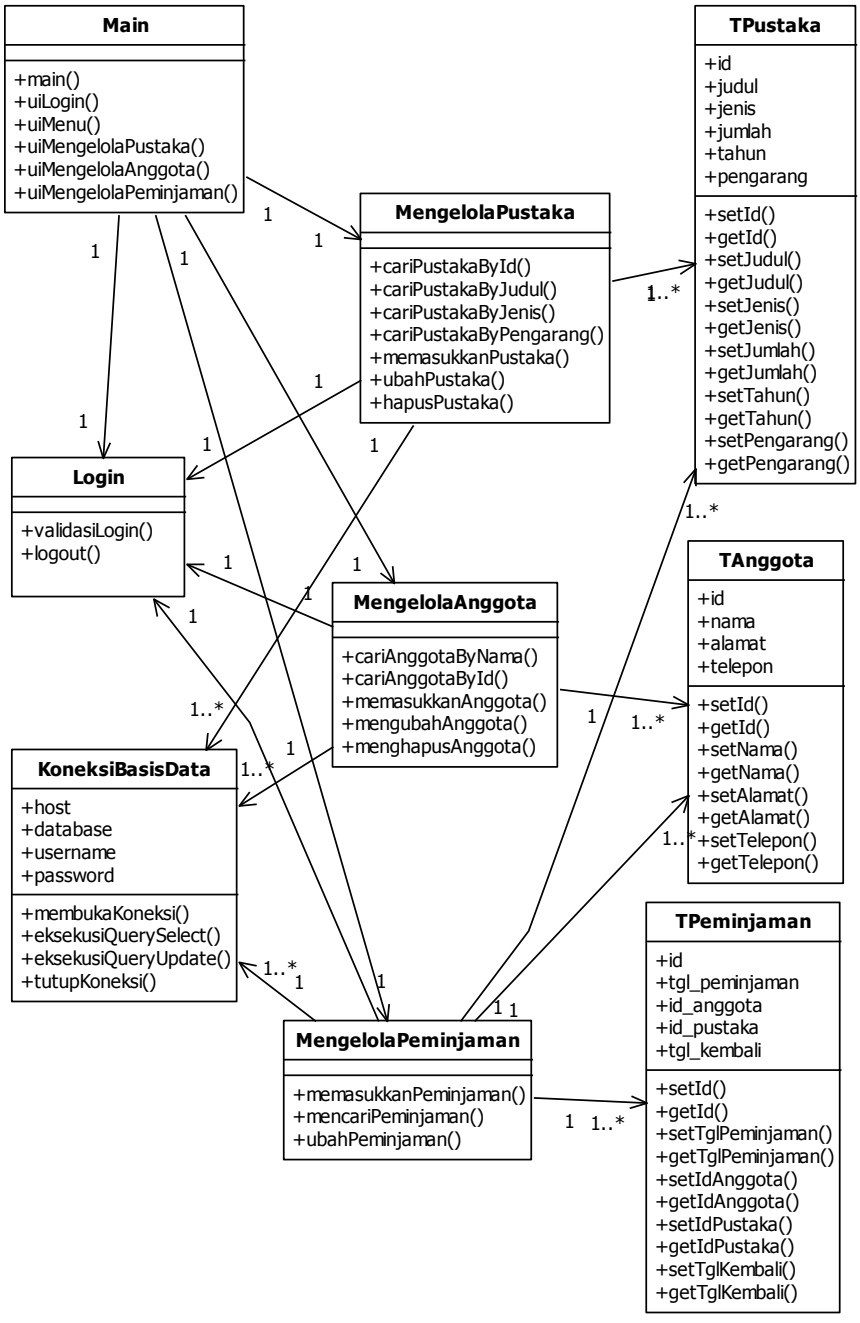
Sumber : (Yuni Sugiarti ; 2013 ; 42)

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 merupakan sebuah bungkusan dari satu atau lebih kelas
Operasi 	Kelas pada struktur sistem
Antarmuka / interface  interface	sama dengan konsep interface dalam pemrograman berorientasi objek
Asosiasi 	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.2. *Class* Diagram
Sumber : (Yuni Sugiarti ; 2013 : 59)



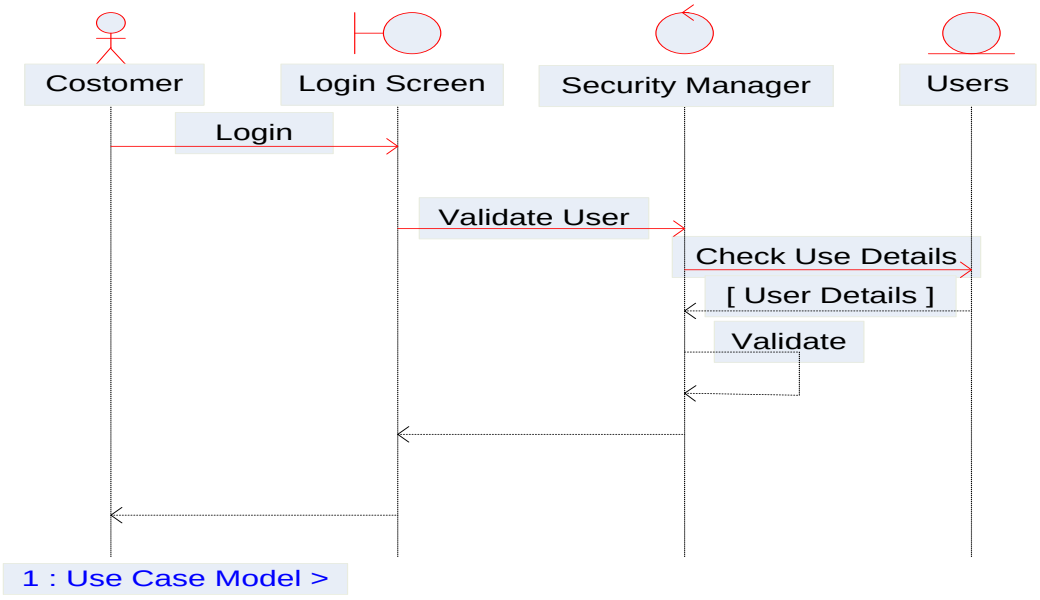
Gambar II.3. Contoh Class Diagram

Sumber : (Yuni Sugiarti ; 2013 : 63)

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*

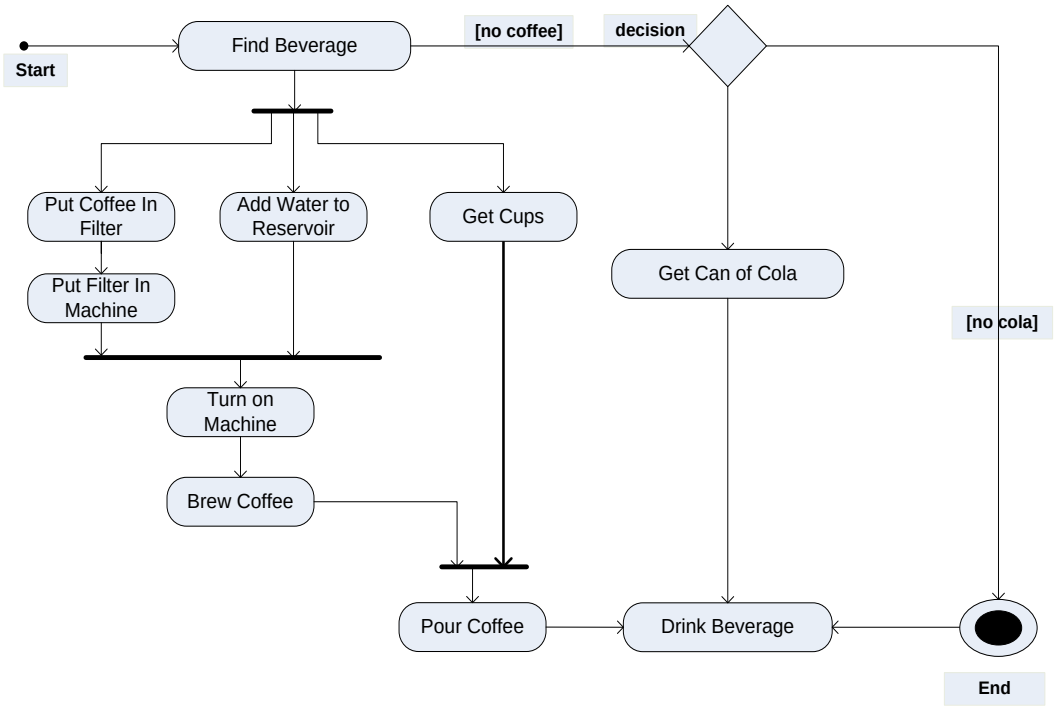
Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

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)