BAB II

TINJAUAN PUSTAKA

II.1. Aplikasi

Aplikasi adalah suatu *sub* kelas perangkat lunak komputer yang memanfaatkan kemampuan komputer langsung untuk melakukan suatu tugas yagn diinginkan pengguna. Contoh utama aplikasi adalah pengolah kata, lembar kerja, memanipulasi foto, merancang rumah dan pemutar media. Beberapa aplikasi yang digabung bersama menjadi sutau pake disebut sebagai suatu paket atau *suite* aplikasi (*application suite*). Contohnya adalah *Microsoft Office* dan *OpenOffice.org*, yang menggabungkan suatu aplikasi pengolah kata, lembar kerja dan beberapa aplikasi lainnya. Aplikasi-aplikasi dalam suatu paket biasanya memiliki atarmuka pengguna yang memiliki kesamaan sehingga memudahkan pengguna untuk mempelajari dan menggunakan tiap aplikasi. Sering kali, mereka memiliki kemampuan untuk saling berinteraksi satu sama lain sehingga menguntungkan pengguna. Contohnya, suatu lembar kerja dapat dibenamkan dalam suatu dokumen pengolah kata walaupun dibuat pada aplikasi lembar kerja yang terpisah. (*Dahlan Abdullah ; 2013 : 152*)

II.2. Keamanan

Masalah keamanan merupakan salah satu aspek penting dari sebuah sistem Sayang sekali masalah keamanan ini sering kali kurang mendapat perhatian dari para pemilik dan pengelola sistem informasi. Seringkali masalah keamanan

berada di urutan kedua, atau bahkan di urutan terakhir dalam daftar hal-hal yang dianggap penting. Apabila menggangu performansi dari sistem, seringkali keamanan dikurangi atau ditiadakan *(Dowd & McHenry,* 1998: 24-28) yang dikutip oleh Rahardjo. Keamanan itu tidak dapat muncul demikian saja. Dia harus direncanakan. Ambil contoh berikut. Jika kita membangun sebuah rumah, maka pintu rumah kita harus dilengkapi dengan kunci pintu. Jika kita terlupa memasukkan kunci pintu pada budget perencanaan rumah, maka kita akan dikagetkan bahwa ternyata harus keluar dana untuk menjaga keamanan. (*Ilham Alamsyah ; 2013 :2*)

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, 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. Pengirim dan Penerima : Komunikasi data melibatkan pertukaran pesan antara dua entitas. Pengirim (*sender*) adalah entitas yang mengirim pesan kepada entitas lainnya. 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-79*)

II.3.2. Kriptografi Kunci Simetri (Kriptografi Kunci-Privat)

Pada sistem kriptografi kunci-simetri, kunci untuk enkripsi sama dengan kunci untuk dekripsi, oleh karena itulah dinamakan kriptografi simetri. Keamanan sistem kriptografi simetri terletak pada kerahasiaan kuncinya. Ada banyak algoritma kriptografi modern yang termasuk ke dalam sistem kriptografi simetri,

diantaranya adalah DES (*Data Encryption Standard*), Blowfish, Twofish, Triple-DES, IDEA, Serpent, AES (*Advanced Encryption Standard*). (*Suriski Sitinjak ; 2010 : C-79*)

Algoritma kriptografi (*cipher*) simetri dapat dikelompokkan menjadi dua kategori, yaitu (*Suriski Sitinjak ; 2010 : C-79*) :

1. Cipher aliran (*stream cipher*)

   Algoritma kriptografi beroperasi pada plainteks/cipherteks dalam bentuk bit tunggal, yang dalam hal ini rangkaian bit dienkripsikan/didekripsikan bit per bit.

2. Cipher blok (*block cipher*)

   Algoritma kriptografi beroperasi pada plainteks/cipherteks dalam bentuk blok *bit*, yang dalam hal ini rangkaian *bit* dibagi menjadi blok-blok *bit* yang panjangnya sudah ditentukan sebelumnya.

II.4. Algoritma Kriptografi

Algoritma-algoritma kriptogrfi dapat dibedakan menjadi dua macam yaitu simetrik dan asimetrik. Algoritma simetrik (model enkripsi konvensional) merupakan algoritma yang menggunakan satu kunci untuk proses enkripsi dan deskripsi data. Sedangkan algoritma asimetrik (model enkripsi kunci publik) menggunakan kunci yang berbeda dalam proses enkripsi dan deskripsi pesan. (*I Gede Andika Putra ; 2012 : 30*)

II.5. Algoritma AES (*Advanced Encryption Standard*)

Dalam kriptografi, Anvanced Encryption Standard (AES), juga dikenal sebagai Rijndael, AES adalah sebuah block cipher yang dijadikan standar enkripsi oleh pemerintah Amerika Serikat. Enkripsi ini diharapkan juga digunakan secara luas di seluruh dunia dan dianalisa secara luas, seperti pada pendahulunya , Data Encryption Standard (DES). Rijndael (AES) diumumkan oleh National Institute of Standards and Technology (NIST) pada tanggal 26 Nopember 2001, setelah lima tahun proses standardisasi. Metode enkripsi ini menjadi standar secara efektif mulai tahun 2002. Pada tahun 2006, AES adalah salah satu algoritma populer yang digunakan dalam kriptografi kunci simetris.

Algoritma AES menggunakan substitusi, permutasi, dan sejumlah putaran yang dikenakan pada tiap blok yang akan dienkripsi / dekripsi. Untuk setiap putarannya, Rijndael menggunakan kunci yang berbeda. Rijndael beroperasi dalam orientasi byte sehingga memungkinkan untuk implementasi algoritma yang efisien ke dalam software dan hardware. (*I Gede Andika Putra ; 2012 : 30*)

II.5.1. Proses Enkripsi Algoritma AES

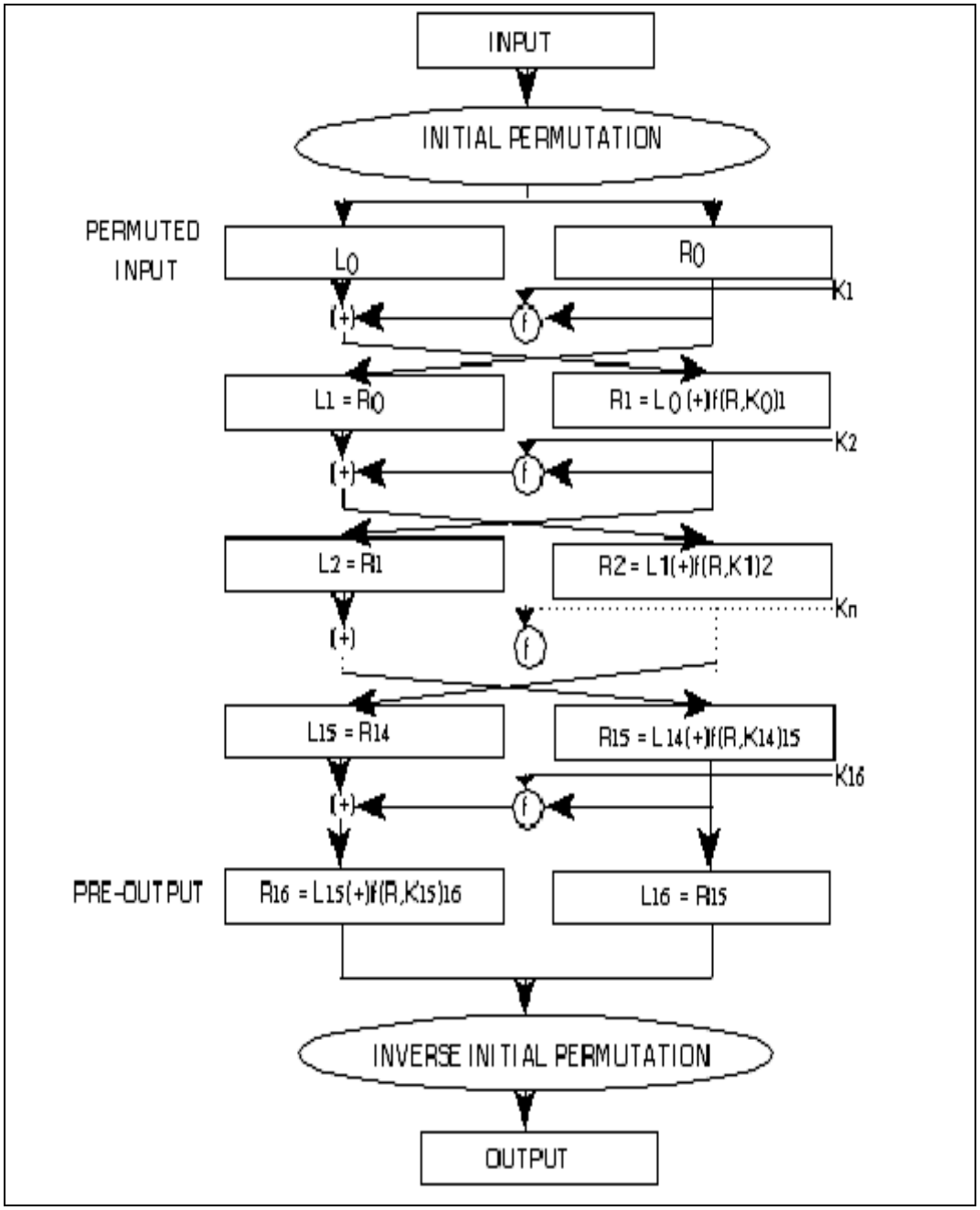
Proses yang dilakukan setiap rondenya identik (dari ronde ke-0 sampai dengan ronde ke Nr-1), kecuali untuk ronde terakhir Nr. Proses yang identik tersebut terdiri atas *SubBytes*(), *ShiftRows*(), *MixColumns*(), dan *AddRoundKey*(). Sedangkan pada ronde terakhir Nr tidak dilakukan fungsi *MixColumns*().

Array 4 x 4 byte *plaintext* yang disebut *state* dioperasikan XOR dengan kunci, kemudian diolah sebanyak 9 ronde dengan operasi *SubBytes*, *ShiftRows*, *MixColumns*, *AddRoundKey*. Tiap ronde akan memilki *round key* yang diturunkan dari kunci utama. Pada ronde terakhir (ronde 10) tidak dilakukan proses *MixColumns*, keseluruhan proses enkripsi ini akan mengasilkan *cipher* 4 x 4 *byte*. (*I Gede Andika Putra ; 2012 : 30*)

II.6. Algoritma DES (*Data Encryption Standard*)

Fungsi lengkap dari algoritma DES dapat dijelaskan secara singkat sebagai berikut. DES adalah blok *cipher*. Ini beroperasi pada blok dari 64-bit dalam ukuran. Sebuah masukan blok 64-bit dari *plaintext* akan dienkripsi menjadi 64-bit *output* blok teks cipher. Ini adalah sebuah Algoritma simetris, yang berarti algoritma dan kunci yang sama digunakan untuk enkripsi dan dekripsi. Keamanan DES terletak di kunci 56-bit. Blok *plaintext* diambil dan dimasukkan melalui permutasi awal. Kuncinya adalah juga diambil pada waktu yang sama.

Kuncinya disajikan dalam blok 64-bit dengan setiap bit 8 menjadi cek paritas. Kunci 56-bit kemudian diekstraksi siap digunakan. 64-bit blok *plaintext* dibagi menjadi dua bagian 32 bit, bernama kanan setengah setengah kiri. Itu dua bagian dari *plaintext* kemudian digabungkan dengan data dari kunci dalam operasi yang disebut Fungsi F. Ada 16 putaran Fungsi f, setelah itu dua bagian yang digabungkan menjadi satu 64-bit blok, yang kemudian dimasukkan melalui final permutasi untuk menyelesaikan operasi algoritma dan 64-bit teks *cipher* blok dikeluarkan. (*Mandeep Singh Narula ; 2014 ; 667*)



Gambar II.1. Proses DES *Encryption*
Sumber : (*Mandeep Singh Narula ; 2014 ; 667*)

II.7. *Java*

Java adalah nama untuk sekumpulan teknologi untuk membuat dan menjalankan perangkat lunak pada komputer *standalone* ataupun pada lingkungan jaringan *Java* 2 adalah generasi kedua dari *Java platform* (generasi awalnya adalah JDK atau *Java Development Kit*). *Java* inilah yang berdiri diatas mesin *interpreter* yang diberi nama *Java Virtual Machine*(JVM). JVM inilah yang akan membaca *bytecode* dalam *file .class* dari suatu program sebagai representasi langsung program yang berisi bahasa mesin". Oleh karena itu bahasa java disebut juga sebagai bahasa pemrograman yang *portable* karena dapat dijalankan sebagai sistem operasi, asalkan pada sistem operasi tersebut terdapat JVM. (*Utomo Budiyanto ; 2011 : 27*)

Sun Microsystems telah mendefinisikan tiga *platform java* menurut *Utomo Budiyanto 2011* yang masing – masing diarahkan untuk tujuan tertentu dan untuk lingkungan komputasi yang berbeda-beda:

1. *Java Standard Edition* (J2SE), adalah inti dari bahasa pemrograman *java*. JDK adalah salah satu *tool* dari J2SE untuk mengkompilasi program *java* pada JRE.
2. *Java Enterprise Edition* (J2EE), dengan *built-in* mendukung untuk *servlets*, JSP, dan XML, edisi ini ditujukan untuk aplikasi berbasis *server*.
3. *Java Micro Edition* (J2ME), didesain untuk meletakkan perangkat lunak *java* pada barang elektronik beserta perangkat pendukungnya.

Teknologi *Java* mencangkup 2 elemen penting yaitu bahasa pemrograman (*programming language*) dan lingkungan aplikasi (*application environment*). *Java* sebagai bahasa pemrograman dapat diartikan bahwa *java* sebanding dengan

bahasa pemrograman seperti C++, *Pascal*, *Visual Basic*, dan lainya, sedangkan *Java* sebagai lingkungan aplikasi berarti bahwa *java* dapat berjalan pada berbagai lingkungan seperti *browser*(*Applets*), *server*(*servlets* dan JSP) dan pada *mobile device*(*midlet* dan WAP). *Java* dalam hal ini mengungguli bahasa lainnya yang pernah ada jika dilihat dari sisi teknologi *mobile*. Hal ini dibuktikan dengan banyaknya jenis telepon genggam yang menggunakan *java* sebagai fitur utamanya. *Microsoft.NET mobile* pun kelihatannya belum dapat menyaingi keunggulan *Java* dalam bidang aplikasi *mobile*. Perlu diketahui bahwa *Microsoft* hanya mengandalkan solusi WAP yang mengembangkan ASP.NET untuk kebutuhan *mobile device*, sedangkan *Java* memiliki 2 solusi yaitu WAP dan MIDP (*Mobile Information Device Profile*). Solusi pertama adalah dengan mengandalkan J2EE (*Java* 2 *Enterprise Edition*) dengan produknya yang bernama JSP (*Java Server Pages*) dan *Java Servlets*. JSP dan *Servlets* ini digunakan untuk membentuk halaman WAP. Solusi kedua dengan menggunakan J2ME(*Java* 2 *Micro Edition*) MIDP dengan produknya yang bernama *Midlets*. *Midlets* inilah yang menjadi fitur andalan oleh beberapa jenis telepon genggam terbaru. (*Utomo Budiyanto ; 2011 : 27*)

II.7.1. J2ME (*Java 2 Micro Edition*)

Java Micro Editon atau yang biasa disebut J2ME adalah bagian dari J2SE, karena itu banyak pustaka yang ada pada J2SE dapat digunakan pada J2ME. Tetapi J2ME mempunyai beberapa pustaka khusus yang tidak dimiliki J2SE. Kelahiran *platform* J2ME timbul karena dibutuhkan adanya sebuah *platform* komputasi yang mengakomodasi piranti komputer elektronik dan *embedded*.

Piranti ini dikelompokkan menjadi dua kategori, menurut *Utomo Budiyanto ; 2011 : 27* yaitu :

1. Personal, *piranti mobile* yang dapat digunakan untuk komunikasi melalui jaringan tertentu misalkan ponsel, Personal *Digital Assistant* (PDA), *Palm*, *Pocket* PC dan *organizer*.
2. Piranti informasi yang digunakan bersama dengan jaringan tetap, koneksi jaringan yang tidak putus-putus misalnya TV, internet dan sistem navigasi.

Kategori pertama mengarahkan piranti untuk tujuan khusus atau fungsi-fungsi tertentu yang terbatas dan tidak digunakan untuk mesin komputasi yang serba guna. Kategori kedua diarahkan untuk piranti yang mempunyai kapabilitas yang lebih besar dengan fasilitas *user interface* yang lebih baik, kemampuan komputasi yang lebih besar. (*Utomo Budiyanto ; 2011 : 27*)

1. Keunggulan J2ME

Salah satu kelebihan *Java* yang paling signifikan adalah *run everywhere*. Dengan kelebihan ini, para pengembang yang sudah terbiasa mengembangkan aplikasi dalam bingkai kerja J2ME dan J2EE akan mampu bermigrasi dengan mudah untuk mengembangkan aplikasi J2ME. Selain itu, *Java* juga merupakan *platform* yang memiliki banyak keunggulan lain, keunggulan *Java* secara umum adalah :

a. *Multiplatform*, aplikasi J2ME bisa berjalan diatas banyak *platform* yang didalamnya terdapat JVM. Beberapa *platform* yang tersedia didalamnya terdapat JVM antara lain *Windows* CR, *Symbian*, *Embedded Linux* dan sebagainya.

b. *Robust*, kode-kode *Java* adalah kode-kode *robust*, karena *virtual machine* mengatur keamanan proses eksekusi aplikasi. *Java virtual machine* menyediakan *garbage collector* yang berfungsi mencegah kebocoran memory.
c. Terintegrasi dengan baik, J2ME bisa terhubung dengan *back-end* J2EE *server* dan *web services* dengan mudah karena menyediakan pustaka-pustaka API RMI dan *web services*.
d. Berorientasi obyek, *Java* merupakan salah satu bahasa pemrograman yang murni berorientasi obyek. Hal ini mempermudah dan mempercepat pengembangan sistem yang dikembangkan dengan metode analisa dan desain berorientasi obyek.

II.8. Sistem Operasi *Android*

II.8.1. *Android*

Android adalah sistem operasi untuk telepon seluler yang berbasis Linux, yang mencakup sistem operasi, *middleware* dan aplikasi. *Android* tidak terikat ke satu merek telepon seluler. *Android* menyediakan *platform* terbuka bagi para pengembang untuk menciptakan aplikasi mereka sendiri hingga dapat digunakan oleh berbagai peranti *mobile*. Beberapa fitur utama dari *Android* antara lain WiFi *hotspot*, *Multi-touch*, *Multitasking*, GPS, *support java*, mendukung banyak jaringan (GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, *Bluetooth*, Wi-Fi, LTE, and WiMAX) dan juga kemampuan dasar telepon seluler pada umumnya. (*Alicia Sinsuw ; 2013 : 2*)

II.8.2. *Android* OS

Android OS adalah sistem operasi yang berbasis *Linux*, sistem operasi *open source.* Selain *Android Software Development Kit (SDK)* untuk pengembangan aplikasi, *android* juga tersedia bebas dalam bentuk sistem operasi. Hal ini yang menyebabkan *vendor-vendor smartphone* begitu berminat untuk memproduksi *smartphone* dan komputer *tablet* berbasis *Android*. (*Alicia Sinsuw ; 2013 : 2*)

II.8.3. *Android* SDK (*Software Development Kit*)

Android SDK adalah *tools* API *(Aplication Programming Interface)* yang diperlukan untuk mengembangkan aplikasi pada *platform Android* menggunakan bahasa pemrograman *Java*. Beberapa fitur-fitur *Android* yang paling penting adalah mesin *Virtual Dalvik* yang dioptimalkan untuk perangkat *mobile, integrated browser* berdasarkan *engine open source WebKit*, Grafis yang dioptimalkan dan didukung oleh *libraries* grafis 2D, grafis 3D berdasarkan spesifikasi *opengl* ES 1.0 (Opsional akselerasi perangkat keras), kemudian *SQLite* untuk penyimpanan data (*database*). Fitur-fitur *android* lainnya termasuk media yang mendukung *audio, video*, dan gambar, juga ada *fitur bluetooth*, EDGE, 3G dan WiFi, dengan fitur kamera, GPS, dan kompas. Selanjutnya fitur yang juga turut disediakan adalah lingkungan *Development* yang lengkap dan kaya termasuk perangkat emulator, *tools* untuk *debugging*, profil dan kinerja memori, dan *plugin* untuk IDE *Eclipse*. (*Alicia Sinsuw ; 2013 : 2*)

II.8.4. *AVD (Android Virtual Device)*

Android Virtual Device merupakan emulator untuk menjalankan aplikasi *android*, yang tampilannya dapat dilihat pada gambar 1. Setiap AVD terdiri dari sebuah profil perangkat keras yang dapat mengatur pilihan untuk menentukan *fitur hardware emulator*. Misalnya, menentukan apakah menggunakan perangkat kamera, apakah menggunakan *keyboard* QWERTY fisik atau tidak, berapa banyak memori internal, dan lain-lain. AVD juga memiliki sebuah pemetaan versi *Android*, maksudnya kita menentukan versi dari *platform Android* akan berjalan pada emulator. Pilihan lain dari AVD, misalnya menentukan *skin* yang kita ingin gunakan pada emulator, yang memungkinkan untuk menentukan dimensi layar, tampilan, dan sebagainya. Kita juga dapat menentukan SD *Card virtual* untuk digunakan dengan di emulator. (*Alicia Sinsuw ; 2013 : 2*)

Gambar II.2. Tampilan AVD
Sumber : (*Alicia Sinsuw ; 2013 : 3*)

II.9. 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

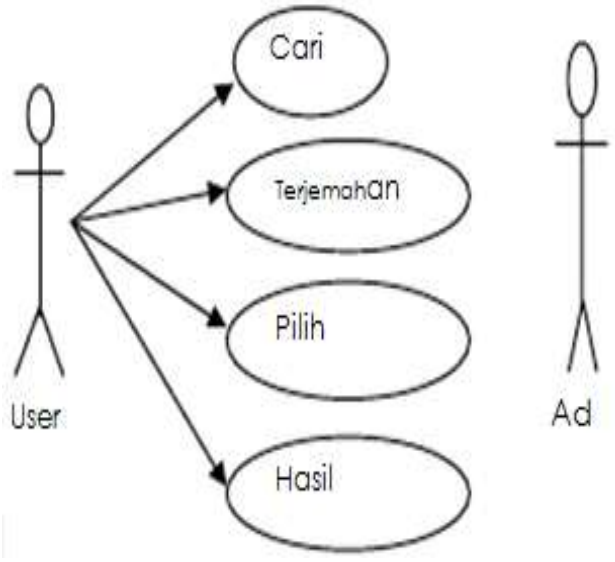
masa perang metodologi (*method war*) dalam pendesainan berorientasi objek. Masing-masing metodologi membawa notasi sendiri-sendiri, yang mengakibatkan timbul masalah baru apabila kita bekerjasama dengan group/perusahaan lain yang menggunakan metodologi yang berlainan. Dimulai pada bulan Oktober 1994 *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

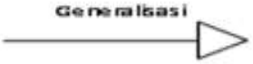
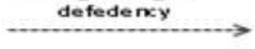
adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. *Use case diagram* dapat sangat membantu bila kita sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* 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)*



Gambar II.3. ***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 :

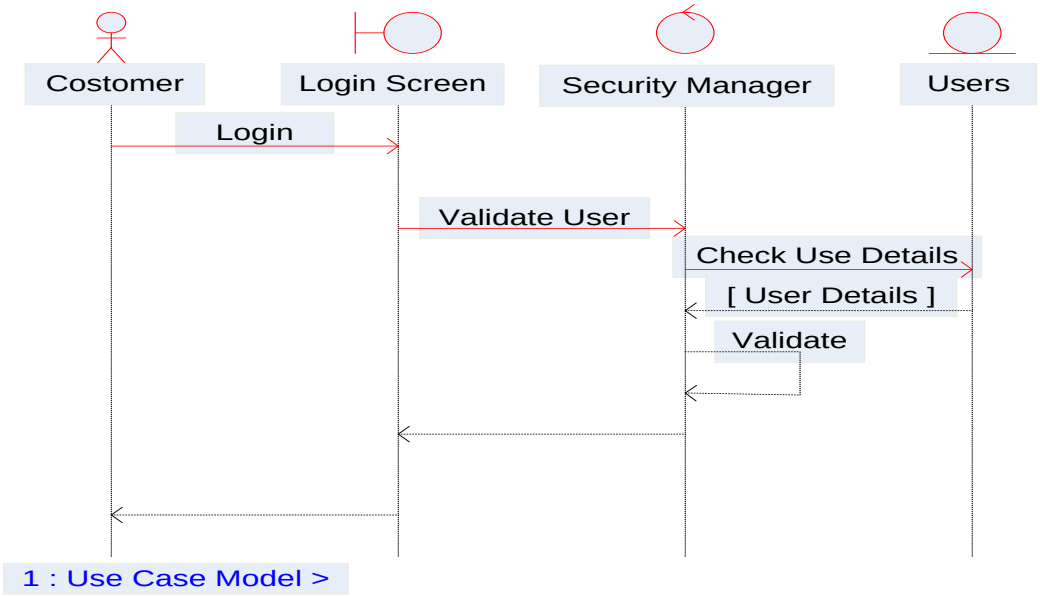
Simbol	Deskripsi
Package 	Package merupakan sebuah bungkusan dari satu atau lebih kelas
Operasi 	Kelas pada struktur sistem
Antarmuka / interface 	sama dengan konsep interface dalam pemrograman berorientasi objek
Asosiasi 	relasi antar kelas dengan makna umum, asosiasi biasanya juga disertai dengan multiplicity
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.4. *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.5. *Contoh Sequence Diagram*
Sumber : (Yuni Sugiarti ; 2013 : 63)

4. *Activity Diagram*

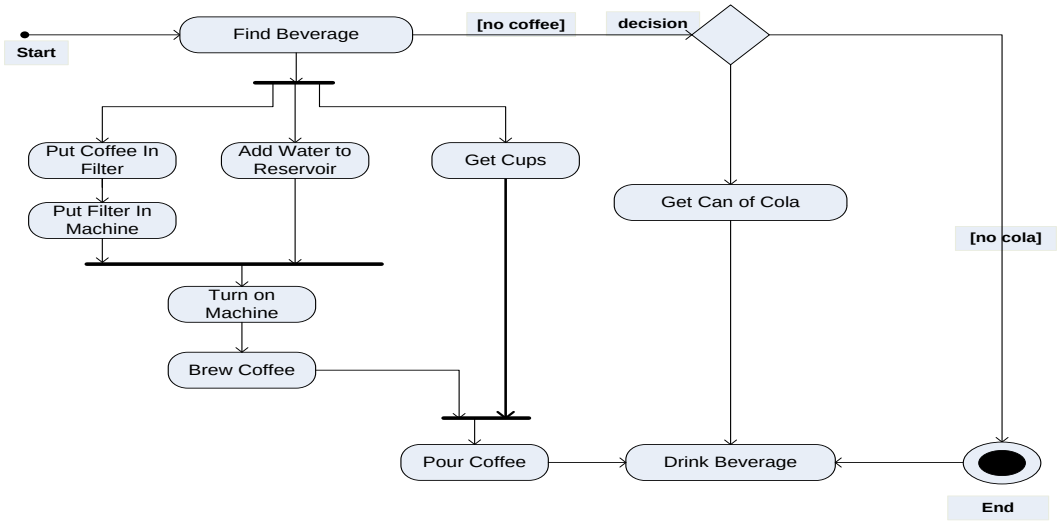
Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

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.6. *Activify* Diagram
Sumber : (Yuni Sugiarti ; 2013 : 76)