BAB II

TINJAUAN PUSTAKA

II.1. Operator Gondola

Operator merupakan SDM yang tugasnya mengoperasikan menghidupkan hardware, menjalankan *software*, berinteraksi dengan *hardware* dan *software* yang sedang beroprasi, dan menyudahi operasi (menghentikan *software* dan mematikan *hardware*). Operator tidak perlu memiliki latar belakang pendidikan IT, selain pelatihan sesuai tugasnya. Dalam melaksanakan tugasnya, tentu memerlukan otoritas yang cukup tinggi karena harus bisa menghidupkan dan mematikan sistem. Mereka memiliki akses penuh atas *console* sistem dan aplikasi untuk sistem produksi. Oleh karena itu operator harus bekerja di ruang khusus yang tidak boleh dimasuki oleh siapa saja selain yang diijinkan oleh pimpinan operasi. Sebaliknya, *console* sistem dan aplikasi untuk sistem produksi di setup sedemikian rupa supaya tidak bisa dibuka diluar ruang operator.

Gondola adalah alat penunjang atau pembantu bagi pekerja, operator, *cleaner* yang akan bekerja di luar bangunan bertingkat tinggi, tangki minyak, *tower* industri, dinding kapal, dsb. yang digerakkan dengan bantuan motor listrik atau manual dan bergerak secara vertikal maupun horisontal. Pergerakan gondola baik vertikal maupun horisontal dapat dilakukan secara manual maupun dengan bantuan motor listrik. Secara umum gondola mempunyai bagian bagian-bagian penting, diantaranya :

a. *Platform*/*cage*/*cradle*/kereta sebagai tempat pekerja melakukan pekerjaan.

b. Konstruksi penggantung yang mempunyai model sesuai dengan bentuk gedung, kegunaan dan keinginan konsumen.

c. *Wire Rope* / tali baja sebagai penggantung *platform* dengan *roof car*.

d. Mesin penggerak.

e. *Accessories* yang lain seperti :

- *Strirrup*
- *Safety Device*
- *Panel Contol* dan kabel *power*
- Roda dinding dan roda *platform*
- *Wire clip*, *thimble*, dan *shackle*. *(NCC High Rise Cleaning: 13-14)*

II.2. Sistem

Sistem adalah sebuah tatanan atau keterpaduan yang terdiri dari sejumlah komponen fungsional (dengan satuan fungsi atau tugas khusus) yang saling berhubungan dan secara bersama – sama bertujuan untuk memenuhi suatu proses atau pekerjaan tertentu. Suatu sistem terdiri dari sejumlah komponen yang saling berinteraksi, artinya saling bekerja sama membentuk satu kesatuan. Suatu sistem mempunyai karakteristik atau sifat–sifat tertentu, yaitu mempunyai komponen–komponen (*components*), batas sistem (*boundary*), lingkungan luar sistem (*environment*), penghubung (*interface*), masukan (*input*), keluaran (*output*), pengolah (*process*) dan sasaran (*objectives*) atau tujuan (*goal*). Komponen–komponen sistem atau elemen–elemen sistem dapat berupa suatu sub sistem atau bagian–bagian dari sistem. (*Hafsah ; 2011 : D-43*)

II.3. Sistem Penunjangan Keputusan

SPK (Sistem Penunjang Keputusan) Sistem Pendukung Keputusan (*Decision Support System*/DSS) merupakan sistem informasi interaktif yang menyediakan informasi, pemodelan, dan pemanipulasian data. Sistem ini digunakan untuk membantu pengambilan keputusan dalam situasi yang semiterstruktur dan situasi yang tidak terstruktur, dimana tak seorang pun tahu pasti bagaimana keputusan seharusnya dibuat. SPK biasanya dibangun untuk mendukung solusi atas suatu masalah atau untuk mengevaluasi suatu peluang. Biasanya Sistem Pendukung Keputusan lebih ditujukan untuk mendukung manajemen dalam melakukan pekerjaan yang bersifat analitis dalam *(Leni Natalia Zulita; 2013 : 95-96)*

II.3.1. Sistem Pendukung Keputusan (*Decision Support System*)

Little (1970) mendefinisikan DSS sebagai "sekumpulan prosedur berbasis model untuk data pemrosesan dan penilaian guna membantu para manager mengambil keputusan.Jadi, mereka mendefinisikan DSS sebagai sistem yang dapat diperluas untuk mampu mendukung analisis data ad hoc dan pemodelan keputusan, berorientasi terhadap perencanaan masa depan, dan digunakan pada interval yang tidak reguler dan tak terencana. *(Efraim Turban ; 2005 : 137).*

II.3.2. Karakteristik Sistem Pendukung Keputusan

Beberapa karakteristik yang membedakan Sistem Pendukung Keputusan dengan sistem informasi lainnya menurut (Hafsah,2011) yaitu:

1. Sistem Pendukung Keputusan dirancang untuk membantu pengambilan keputusan dalam memecahkan masalah yang sifatnya semi terstruktur ataupun tidak terstruktur.
2. Dalam proses pengolahannya, sistem pendukung keputusan mengkombinasikan model-model analisis dengan teknik pemasukan dan konvensional secara fungsi-fungsi pencarian informasi.
3. Sistem Pendukung Keputusan dirancang sedemikian rupa sehingga dapat digunakan atau dioperasikan dengan mudah oleh orang-orang yang tidak memiliki dasar kemampuan pengoprasisan komputer yang tinggi. Oleh karena itu pendekatan yang digunakan biasanya model interaktif.
4. Sistem Pendukung Keputusan dirancang dengan menekankan pada aspek fleksibilitas serta kemampuan adaptasi yang tinggi. Sehingga mudah disesuaikan dengan beragai perubahan lingkungan yang terjadi pada kebutuhan pemakai. (*Hafsah ; 2011*)

II.4. Arsitektur Sistem Pendukung Keputusan

Aplikasi sistem pendukung keputusan bisa terdiri dari beberapa subsistem, yaitu :

1. Subsistem manajemen data

   Subsistem manajemen data memasukkna satu database yang berisi data yang relevan untuk suatu situasi dan dikelola oleh perangkat lunak yang disebut sisten manajemen database (DBMS/ *Data Base Management System).* Subsistem manajemen data bisa diinterkoneksikan dengan data warehouse perusahaan, suatu repositori untuk data perusahaan yang

relevan dengan pengambilan keputusan.

2. Subsistem manajemen model

   Merupakan paket perangkat lunak yang memasukkan modek keuangan, statistik, ilmu manajemen, atau model kuantitaif lain yang memberikan kapabilitas analitik dan manajemen perangkat lunak yang tepat, Bahasa-bahasa pemodelan untuk membangun model-model kustom juga dimasukkan. Perangkat lunak itu sering disebut sistem manajemen basis model (MBMS).

3. Subsistem antarmuka pengguna

   Pengguna berkomunikasi dengan dan memerintahkan sistem pendukung keputusan melalui subsistem tersebut. Pengguna adalah bagian yang dipertimbangkan dari sistem. Para peneliti menegaskan bahwa beberapa kontribusi unik dari sistem sistem pendukung keputusan berasal dari interaksi yang untensif antara komputer dan pembuat keputusan.

4. Subsistem manajemen berbasis pengetahuan

   Subsistem tersebut mendukung semua subsistem lain atau bertindak langsung sebagai suatu komponen independen dan bersifat opsional. (*Kusrini ; 2007 : 25-26*)

II.5. Struktur Dasar Algoritma

Algoritma berisi lanagkah-langkah penyelesaian suatu masalah. Langkah-langkah dapat berupa runtutan aksi, pemilihan aksi dan pengulangan aksi. Jadi sebuah algoritma dapat dibangun dengan tiga buah struktur dasar, yaitu ;

II.5.1. Runtutan

Sebuah runtutan terdiri dari satu atau lebih pernyataan. Tiap pernyataan dikerjakan secara berurutan sesuai dengan urutan penulisannya, Yakni sebuah instruksi dilaksanakan setelah instruksi sebelumnya selesai dilaksanakan. urutan instruksi menentukan keadaaan akhir algoritma. Bila urutannya dirubah, maka hasil akhirnya mungkin juga berubah.

II.5.2. Pemilihan

Adakalanya sebuah aksi dikerjakan jika kondisi tertentu dipenuhi. Misalkan kendaraan anda tiba diperempatan yang ada *traffic light*. Jika lampu *Traffic Light* sekarang berwarna merah, maka kendaraan anda harus berhenti.

II.5.3. Pengulangan

Salah satu kelebihan komputer adalah kemampuannya untuk mengerjakan pekerjaan yang sama berulang kali tanpa mengenal lelah. Ini berbeda dengan manusia yang cepat lelah bila mengerjakan perkerjaan yang sama berulang-ulang. Tidak hanya lelah tetapi juga cepat bosan. (Rinaldi Munir ; 2005 ; 22-26)

II.6. Algoritma ID3

Iterative Dichotomicer 3 (ID3) adalah algoritma *decision tree learning* (algoritma pembelajaran pohon keputusan) yang paling dasar. Algoritma ini melakukan pencarian secara rakus /menyeluruh (*greedy*) pada semua kemungkinan pohon keputusan.

Penelitian Algoritma ID3 (Iterative Dichotomizer Three) akan digunakan sebagai perhitungan yang akan menghasilkan pohon keputusan. Data yang diperlukan dalam penelitian ini akan diambil dari database karyawan, dan data rekrut karyawan dimana data ini dimasukkan langsung oleh bagian HRD (Human Resources Department) dan semua proses perhitungan algoritma ID3 (Iterative Dichotomizer Three) dilakukan oleh sistem.(Ninik Kristiyani ; 2011 : 3)

Decision Tree adalah sebuah struktur pohon, dimana setiap *node* pohon merepresentasikan atribut yang telah diuji, setiap cabang merupakan suatu pembagian hasil uji, dan *node* daun (*leaf*) merepresentasikan kelompok kelas tertentu. Level *node* teratas dari sebuah *decision tree* adalah *node* akar (*root*) yang biasanya berupa atribut yang paling memiliki pengaruh terbesar pada suatu kelas tertentu. (*Wahyudin ; 2010 : 6*)

Sebuah obyek yang diklasifikasikan dalam pohon harus dites nilai *entropinya*. *Entropy* adalah ukuran dari teori informasi yang dapat mengetahui karakteristik dari *impuryt* ,dan *homogenity* dari kumpulan data. Dari nilai *entropy* tersebut kemudian dihitung nilai *information gain* (IG) masing-masing atribut.

- Entropy(S) = - p+ log2p + -p -log2p (*Wahyudin ; 2010 : 7*)

dimana :

- S adalah ruang (data) *sample* yang digunakan untuk *training*.

- P+ adalah jumlah yang bersolusi positif (mendukung) pada data *sample* untuk kriteria tertentu.
- P+ adalah jumlah yang bersolusi negatif (tidak mendukung) pada data *sample* untuk kriteria tertentu. Dari rumus *entropy* diatas dapat disimpulkan bahwa definisi *entropy* (S) adalah jumlah *bit* yang diperkirakan dibutuhkan untuk dapat mengekstrak suatu kelas (+ atau -) dari sejumlah data acak pada suatu ruang sampel S. *Entropy* bisa dikatakan sebagai kebutuhan bit untuk menyatakan suatu kelas. Semakin kecil nilai *entropy* maka semakin baik digunakan dalam mengekstraksi suatu kelas.(*Avia Enggar; 2015 : 6*)

II.7. Java

Java merupakan bahasa pemrograman berorientasi objek daan bebas platform, dikembangkan oleh SUN Micro System dengan jumlah keunggulan yang memungkinkan java dijadikan sebagai bahasa pengembang entreprise. Java merupakan bahasa yang powerfull yang bisa digunakan dalam hampir semua bentuk pengembangan software. Anda dapat menggunakan java untuk membuat game, aplikasi desktop, aplikasi web, aplikasi enterprise, aplikasi jaringan, dan lain-lain. Yang menarik adalah bahwa java bias digunakan untuk membuat laporan yang dapat berjalan di atas HP, PDA, dan peralatan lain yang dilengkapi dengan Java Virtual Machine(JVM). (*Atik Rusmayanti ; 2014 : 2*)

II.8. Database

Database adalah sekumpul’an file data yang saling berkaitan dan

berorganisasi sedemikian rupa sehingga memudahkan untuk mendapat dan memproses data. Lingkungan databse menekan data yang tidak tergantung (independent data) pada aplikasi yang akan menggunakan data. (Andi, 2006).

II.8.1. Jenis-Jenis *Database*

1. *Database* Hirarki

Yaitu suatu data yang tersusun dengan bentuk hirarki pohon. Susunan yang seperti ini terdiri dari beberapa unsur komponen yang saling mempengaruhi dan tidak dapat dipisahkan, jenis *database* ini merupakan hubungan satu komponen dengan banyak komponen. (*Indra Warman ; 2012 : 46*).

2. *Database* Relasi

Adalah suatu data yang disusun dalam bentuk tabel yang terdiri dari dua definisi dan tersusun secara terstruktur. Bentuk susunan dua dimensi ini terdiri dari beberapa kolom dan *record* yang tersusun berbentuk baris dari kiri kekanan. Data- data yang susunannya berbentuk barus adalah susunan yang menurun kebawah. Dimana pada setiap baris berisikan data- data yang saling berkaitan satu sama lainnya. Artinya setiap pemasukan data yang tersimpan pada *field* merupakan kesatuan dalam bentuk satu baris.(*Indra Warman ; 2012 : 46*)

II.8.2. Prinsip-Prinsip *Database*

Sistem database manajemen dibentuk untuk mengurangi masalah-masalah dalam organisasi. Misalnya data/informasi tidak tersedia atau saling tumpang tindih. Berikut prinsip manajemen database menurut *Anis nurhanafi* (2013 : 3) adalah :

1. Ketersediaan

   Data mudah diakses oleh suatu program dan pemakai (*user*) dimanapun dan kapanpun diperlukan.

2. Pemakaian bersama

   Struktur data disusun sedemikian hingga dapat digunakan oleh beberapa pemakai bersama-sama untuk mengurangi redudansi data.

3. Pengembangan

   Databases dapat dikembangkan sesuai dengan perkembangan kebutuhan pemakai. *Databases* dapat dimodifikasi untuk pengembangan selanjutnya dan dapat beradaptasi dengan lingkungan.

4. Kesatuan

   Database dibentuk dalam satu kesatuan untuk memudahkan pengotrolannya (pemeliharaan dan pengawasan) mudah dilakukan.

II.8.3. Normalisasi

Ada beberapa aturan di dalam perancangan database, yang disebut dengan aturan normalisasi. Aturan-aturan ini akan merancanga database yang normal, atau setidaknya memverifikasi rancangan anda. Database dianggap noemal jika ia tidak mengulangi informasi atau tidak menimbulkan keanehan pada proses update atau penghapusan. Walaupun jumlah aturan ini bervariasi aturan dasar normalisasi sebenarnya ada tiga (3) : aturan normalisasi pertama, kedua dan ketiga.

II.8.3.1. Bentuk Normalisasi Pertama

Bentuk ini sangat sederhana. Aturannya, sebuah tabel tidak boleh mengandung kelompok yang berulang. Dibawah ini adalah sebuah tabel yang

mengandung sebuah kesalahan dari rancangan sebelumnya.

II.8.3.2. Bentuk Normalisasi Kedua

Aturan normalisasi yang kedua berbunyi bahwa setiap field yang tidak bergantung sepenuhnya pada kunci premier harus dipindahkan ketabel lain. Field Topic pada struktur tabel yang terakhir tidak bergantung secara fungsional adalah sebuah istilah matematika, yang berarti bahwa sebuahfield sangat ditentukan oleh kunci.

II.8.3.3. Bentuk Normalisasi Ketiga

Aturan normalisasi ketiga berbunyi bahwa tidak boleh ada kebergantungan antara field-field non-kunci. Pada rancangan tabel yang terakhir, anda memiliki kebergantungan semacam ini. Alamat penerbit bergantung semacam ini, anda harus memindahkan informasi penerbit ke tabel lain.

II.9. MySQL

MySQL Server 2000 adalah suatu Perangkat lunak Relational Database Mangement system (RDBMS) yang handal. Didesain untuk mendukung proses transaksi yang besar (seperti order entri yang online, inventori, akuntansi atau manufaktur). MySQL Server akan secara otomatis menginstal enam database utama, yaitu master, model, tempdb, pubs, Northwind, dan Msdb. (Anis nurhanafi ; 2013 : 5)

II.10. 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.

II.10.1. *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,

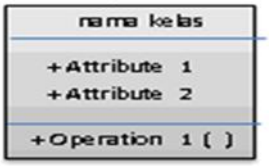
mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem. Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap 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)

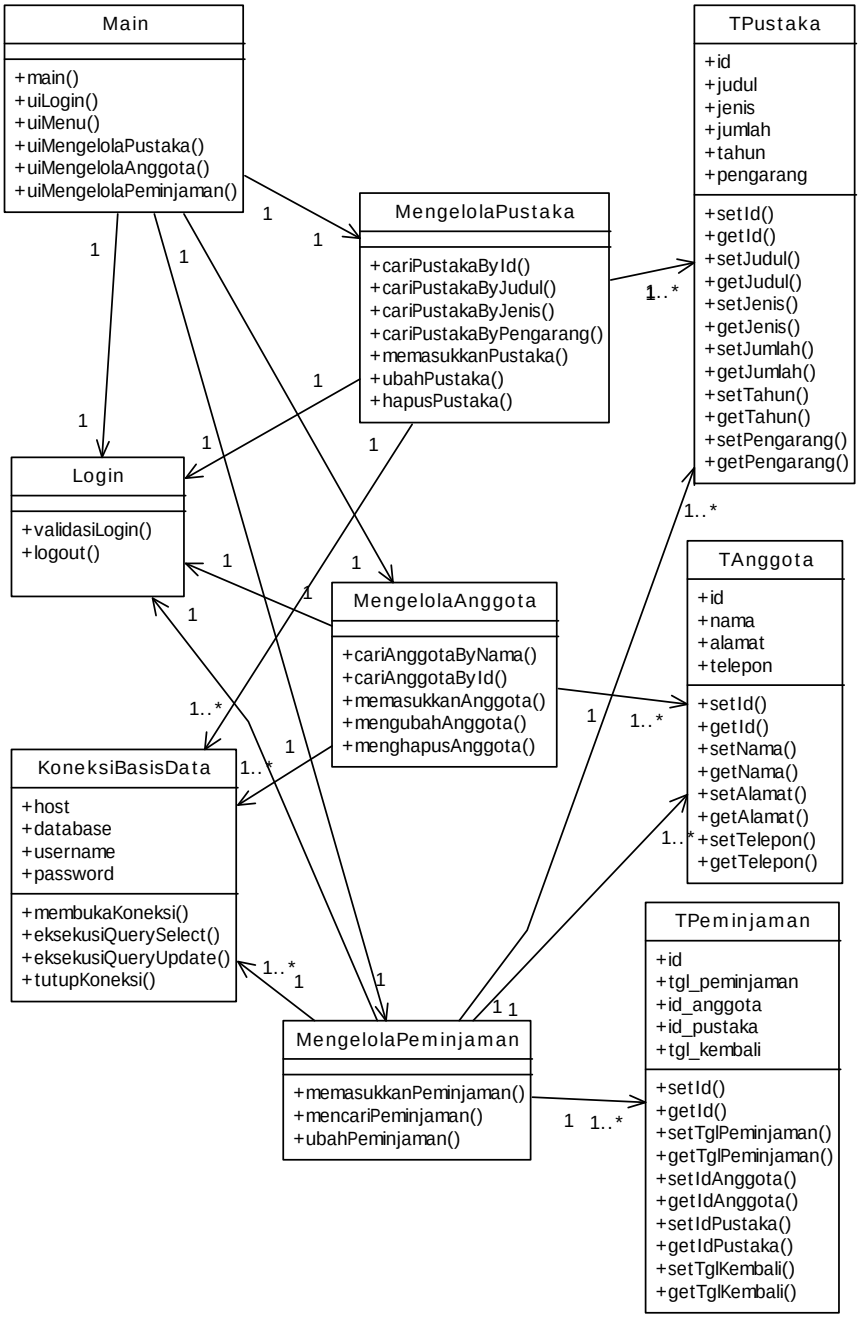
II.10.2. *Class Diagram*

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)

II.10.3. *Sequence Diagram*

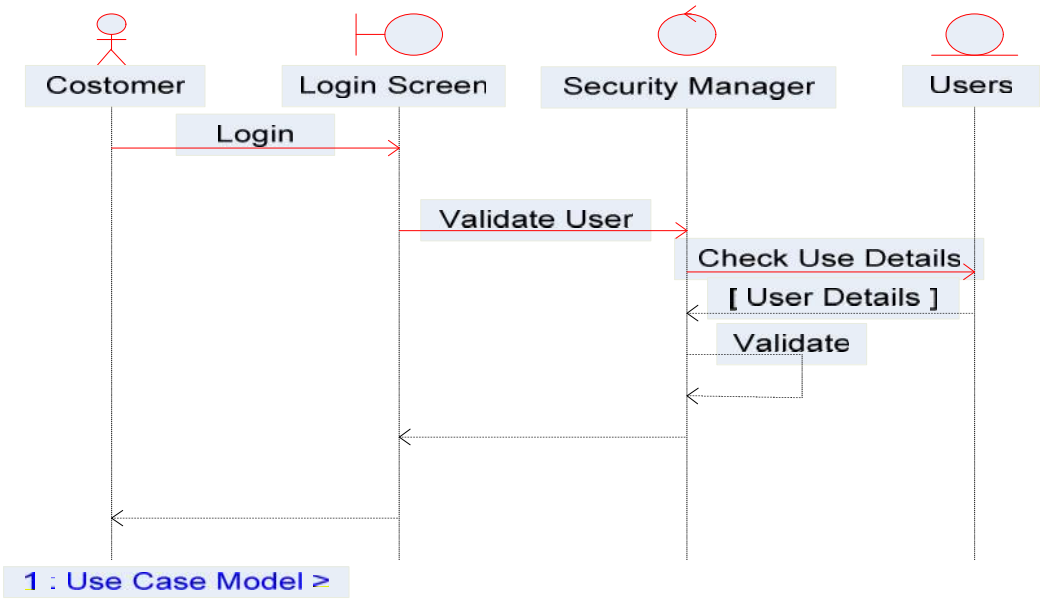
Diagram *Sequence* menggambarkan kelakuan/prilaku objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan

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.

NO	GAMBAR	NAMA	KETERANGAN
1		*LifeLine*	Objek *entity*, antarmuka yang saling berinteraksi.
		*Actor*	Digunakan untuk menggambarkan user / pemgguna.
2		*Message*	Spesifikasi dari komunikasi antar objek yang memuat informasi-informasi tentang aktifitas yang terjadi.
3		*Boundary*	Digunakan untuk menggambarkan sebuah form.
4		*Control Class*	Digunakan untuk menghubungkan *boundary* dengan tabel.
5		*Entity Clas*	Digunakan untuk menggambarkan hubungan kegiatan yang akan dilakukan.

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

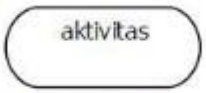
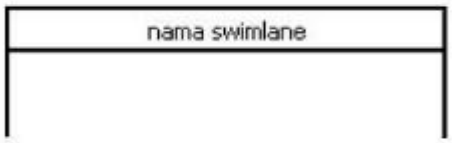
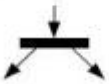
Banyaknya diag ram *sequence* yang harus digambar adalah sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram *sequence* sehingga semakin banyak *use case* yang didefinisikan maka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



Gambar II.5. Contoh *Sequence* Diagram
(Sumber : Yuni Sugiarti ; 2013 : 63)

II.10.4. *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.

Simbol	Deskripsi
status awal 	status awal aktivitas sistem, sebuah diagram aktivitas memiliki sebuah status awal
aktivitas 	aktivitas yang dilakukan sistem, aktivitas biasanya diawali dengan kata kerja
percabangan / *decision* 	asosiasi percabangan dimana jika ada pilihan aktivitas lebih dari satu
penggabungan / *join* 	asosiasi penggabungan dimana lebih dari satu aktivitas digabungkan menjadi satu
status akhir 	status akhir yang dilakukan sistem, sebuah diagram aktivitas memiliki sebuah status akhir
swimlane 	memisahkan organisasi bisnis yang bertanggung jawab terhadap aktivitas yang terjadi
fork, 	digunakan utk menunjukkan kegiatan yg dilakukan secara paralel
join, 	digunakan utk menunjukkan kegiatan yg digabungkan

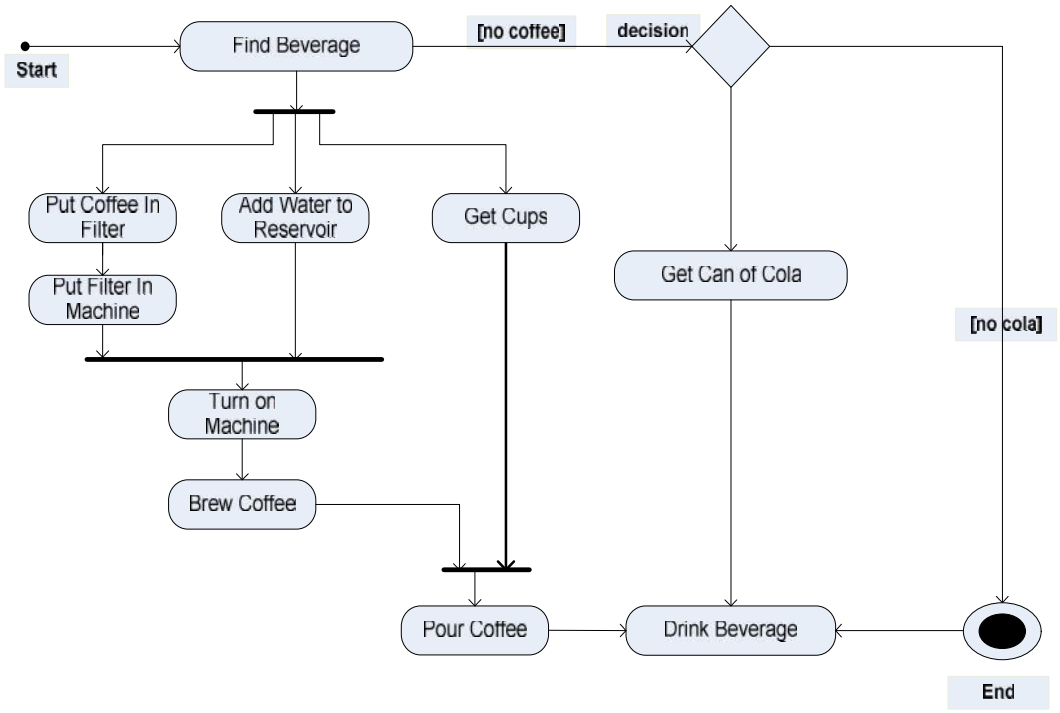
Gambar II.6. Simbol Activity Diagram
(Sumber : Yuni Sugiarti ; 2013 : 76)

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.7. Contoh Activity Diagram
(Sumber : Yuni Sugiarti ; 2013 : 76)