BAB II
LANDASAN TEORI

2.1. Teori Umum

2.1.1. Pengertian *E-Learning*

E-learning terdiri dari dua bagian, yaitu ‘e’ yang merupakan singkatan dari *elektronic* dan learning yang berarti pembelajaran. Jadi *e-learning* berarti pembelajaran dengan menggunakan jasa bantuan perangkat elektronika, khususnya perangkat komputer. Karena itu, maka *e-learning* sering disebut pula dengan *online course*. Dalam berbagai literatur, *e-learning* didefinisikan sebagai berikut :

E-learning is a generic term for all technologically supported learning using an array of teaching and learning tools as phone bridging, audio and videotapes, teleconferencing, satellite transmissions, and the more recognized web-based training or computer aided instruction also commonly referred to as online courses (Soekartawi, 2003).

Dengan demikian maka *e-learning* atau pembelajaran melalui *online* adalah pembelajaran yang pelaksaanya didukung oleh jasa teknologi seperti telepon, *audio, videotape,* transmisi satelit atau komputer.

Karena ada bermacam penggunaan *e-learning* saat ini, maka ada pembagian atau pembedaan e-learning. Pada dasarnya, *e-learning* mempunyai dua tipe, yaitu *synchronous* and *asynchronous*.

1. *Synchronous Training*

   Synchronous berarti “pada waktu yang sama”. Jadi, *synchronous* training adalah tipe pelatihan, dimana proses sedang mengajar dan murid sedang belajar. Hal tersebut memungkinkannya interaksi langsung antara guru dan murid, baik melalui *internet* maupun *intranet*. *Synchronous training* mengharuskan guru dan murid mengakses internet bersamaan. Pengajar memberikan makalah dengan *slide* presentasi dan peserta *web conference* dapat mendengarkan presentasi melalui hubungan *internet*. Jadi *synchronous training* sifatnya mirip pelatihan di ruang kelas. Namun, kelasnya bersifat

maya (*virtual*) dan peserta tersebar di seluruh dunia dan terhubung melalui *internet*. Oleh karena itu, *synchronous training* dinamakan *virtual classroom*.

2. *Asynchronous Training*

   Asynchronous training berarti "tidak pada waktu yang bersamaan". Jadi, seseorang dapat mengambil pelatihan pada waktu yang berbeda dengan pengajar memberikan pelatihan. Pelatihan ini lebih popular di dunia *e-learning* karena memberikan keuntungan lebih bagi peserta pelatihan karena dapat mengakses pelatihan kapanpun dan dimanapun. Pelatihan berupa paket pelajaran yang dapat dijalankan di komputer manapun dan tidak melibatkan interaksi dengan pengajar atau pelajar lainnya. Oleh karena itu, pelajar dapat memulai pelajaran dan menyelesaikannya setiap saat. Paket pelajaran berbentuk bacaan dengan animasi, simulasi, permintaan edukatif, maupun latihan ata tes dengan jawabannya.

   Akan tetapi, ada pelatihan *Asynchronous training* yang terpimpin, dimana pengajar memberikan materi pelajaran lewat *internet* dan peserta pelatihan mengakses materi pada tugas atau latihan dan peserta mengumpulkan tugas lewat *e-mail*. Peserta dapat berdiskusi atau berkomentar dan bertanya melalui *bulletin board.*

2.1.2. *Object-oriented Programming*

Object-oriented Programming atau pemrograman berorientasi objek adalah paradigma pemrograman yang memandang perangkat lunak sebagai kumpulan objek yang saling berinteraksi di dalam suatu sistem. (Azis, 2005, p. 2) Beberapa objek berinteraksi dengan saling memberikan informasi satu terhadap yang lainnya. Masing-masing objek harus berisikan informasi mengenai dirinya sendiri (e*ncapsulation*) dan objek yang dapat dikaitkan (*inheritance*). (Febrian, 2004)

Dalam *Object-oriented Programming*, *Class* merupakan sekumpulan objek yang memiliki atribut atau *properties* dan *method*. (Lethbridge & Laganiere, 2002, pp. 29-31) *Class* merupakan deskripsi dari satu atau lebih objek yang memiliki kesamaan atribut, layanan, metode, hubungan, dan semantik, termasuk

deskripsi cara membuat objek *base* dalam *class*. Ada juga yang disebut dengan *super class*, sebuah *class* induk yang nantinya mempunyai *class-class* yang terdiri dari *class* dan *subclass*. (Lethbridge & Laganiere, 2002, p. 50)

Objek dalam *Object-oriented Programming* adalah sebuah benda atau unit atau sifat kerja yang memiliki atribut-atribut. (Lethbridge & Laganiere, 2002, p. 30) Objek adalah sebuah abstraksi dari sesuatu pada domain masalah, menggambarkan kemampuan untuk menyimpan informasi mengenai hal tersebut, berinteraksi dengan hal tersebut atau keduanya.

Abtraksi prosedural dalam OOP disebut dengan operasi, yang menspesifikasi tipe dari perilaku dan terdiri dari fungsi-fungsi. (Lethbridge & Laganiere, 2002, p. 28)

Istilah lain terdapat *encapsulation* atau pengkapsulan, yang merupakan pembatasan ruang lingkup program terhadap data yang diproses supaya data terlindungi oleh prosedur atau objek lain, kecuali prosedur yang berada di objek itu sendiri. (Lethbridge & Laganiere, 2002, p. 50)

Polymorphism adalah konsep yang menyatakan bahwa sesuatu yang sama dapat mempunyai bentuk dan perilaku yang berbeda, bahwa operasi yang sama mungkin memiliki perbedaan dalam *class* yang berbeda. (Lethbridge & Laganiere, 2002, p. 36)

Pada *Object-oriented Programming*, terdapat juga yang disebut dengan *inheritance* (pewarisan), yaitu kepemilikan yang bersifat *implicit* dari fitur *subclass* yang didefinisikan dalam *superclass*. Fitur tersebut mencakup *variables* dan *method*. (Lethbridge & Laganiere, 2002, p. 37).

2.1.3. Pengertian *Internet*

Internet adalah koleksi jaringan komputer yang ada di seluruh dunia yang menghubungkan jutaan komputer yang digunakan untuk bisnis, pemerintahan, lembaga pendidikan, organisasi, dan individu menggunakan *modem*, kabel telepon, kabel televisi, satelit, dan peralatan komunikasi lainnya (Shelly, Woods, dan Dorin, 2008, p2).

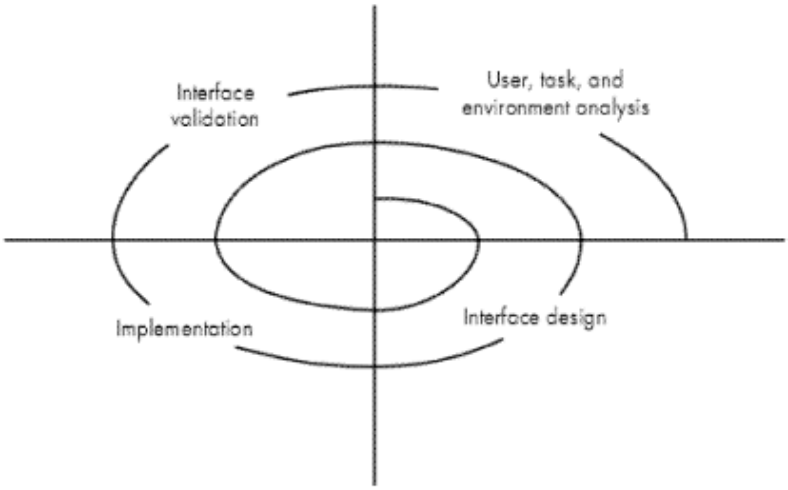
2.1.4. Teori Interaksi Manusia dan Komputer

2.1.4.1. Pengertian Interaksi Manusia dan Komputer

Interaksi manusia dan komputer merupakan sistem yang mampu menghubungkan antara user dengan komputer. Elemen-elemen yang terdap at di dalam user interface antara lain seperti menu, *window*, *keyboard*, *mouse* dan suara-suara komputer (Dastbaz, 2003, p 108).

2.1.4.2. Pengertian *User Interface*

Desain *user interface* adalah menciptakan sebuah media komunikasi yang efektif diantara manusia dan komputer (Pressman, 2010, p312). Proses design antar muka dapat dilihat pada gambar 2.1 dibawah ini:



Gambar 2.1. Proses Desain Antar Muka
Sumber: Roger Pressman (2010, p319)

Berikut langkah-langkah dalam merancang antarmuka, yaitu sebagi berikut (Pressman, 2010, p312):

1. *User, task, and environmental analysis and modelling*
   Setelah *user tasks* teridentifikasi, skenario user dibuat dan dianalisa untuk mendefinisikan satu set objek dan aksi antarmuka.

2. *Interface design*

   Melakukan desain grafis antarmuka, meliputi desain tata letak *icon*, *menu*, *text*, *image* dan lain-lain

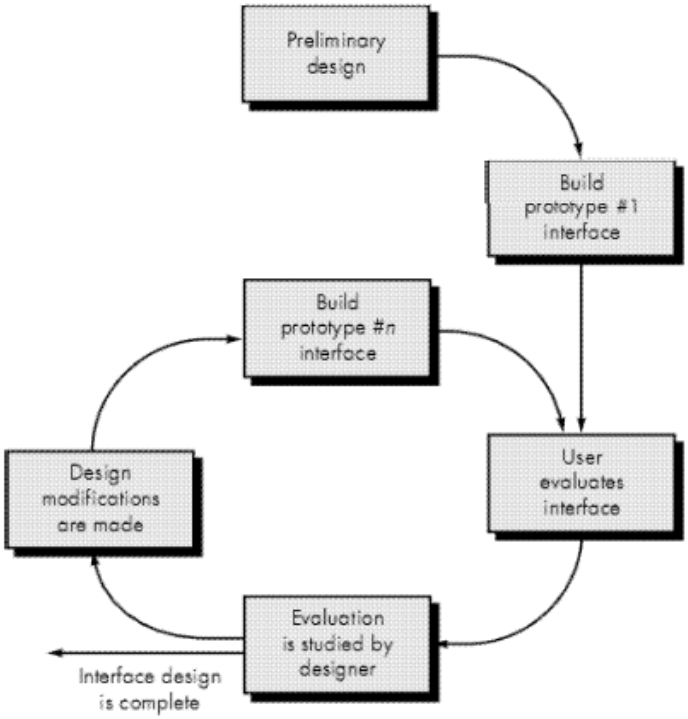
3. *Interface construction*

   Melakukan konstruksi dengan meletakkan *icon*, mendeskripsikan *screen text*, menspesifikasikan menu-menu menjadi satu kesatuan.

4. *Interface validation*

   Setelah sebuah *prototype* antarmuka pengguna diciptakan, kesemuanya harus dievaluasi untuk menentukan apakah kesemuanya itu memenuhi kebutuhan pengguna.

Langkah-langkah dalam perancangan antar muka dapat dilihat dalam gambar 2.2 dibawah ini:



Gambar 2.2. Siklus Evaluasi Desain Antar Muka
Sumber : Roger Pressman (2010, p343)

Selain itu, pengguna dapat dikategorikan sebagai berikut (Pressman, 2010, p318):

1. *Novice User* adalah pengguna awam yang tidak memiliki pengetahuan yang cukup terhadap sistem dan hanya sebatas penggunaan komputer secara umum.
2. *Knowledgeable Intermittent Users* adalah pengguna yang memiliki pengetahuan *semantic* tentang aplikasi dan sistem. Namun, kurang mengeksplorasi lebih dalam fitur-fitur yang ada.
3. *Knowledgeable Frequent Users* adalah pengguna yang memiliki kemampuan semantik dan sintatik serta menguasai sistem dan mampu mengembangkan cara-cara kreatif dalam penggunaan aplikasi.

Adapun delapan aturan emas perancangan *user interface*, yaitu sebagai berikut (Shneiderman, 2005, p74):

1. Berusaha untuk konsisten.

   Rangkaian aksi yang konsisten hams digunakan dalam keadaan seperti pada *prompts*, *menu*, dan layar *help* serta perintah yang konsisten.
2. Menyediakan *usability* universal.

   Usability universal mengacu pada desain informasi dan komunikasi produk dan layanan yang dapat digunakan oleh semua kalangan pengguna.
3. Memberikan umpan balik yang informatif. Untuk setiap aksi yang dilakukan, hendaknya selalu tersedia fasilitas umpan balik agar pengguna mengerti apa yang telah dilakukannya.
4. Merancang *dialog* yang memberikan penutupan.

   Urutan aksi hendaknya dibagi ke dalam kelompok dengan awal, tengah, dan akhir. Ketika telah mencapai bagian akhir, hendaknya pengguna diberitahu melalui umpan balik. Tanpa adanya *dialog* untuk mencapai keadaan akhir maka pengguna akan menjadi bingung.
5. Menawarkan penanganan kesalahan sederhana.

   Sebisa mungkin, desainlah sistem sehingga pengguna tidak dapat melakukan kesalahan yang serius. Jika kesalahan dibuat, sistem harus mampu mendeteksi kesalahan dan membantu memberikan solusi untuk penanganan kesalahan.
6. Memungkinkan pembalikan aksi yang mudah.

   Fitur ini mengurangi kecemasan, karena pengguna tahu bahwa kesalahan

dapat dibatalkan sehingga akan mendorong eksplorasi fungsi-fungsi lainnya.

7. Mendukung pusat kendali internal.

   Dengan pengaturan yang menyeluruh, pengguna dapat menggun akan sistem sesuai kebutuhan mereka dan menggunakan sistem lebih maksimal.

8. Mengurangi beban ingatan jangka pendek.

   Keterbatasan manusia dalam mengolah informasi dalam jangka waktu yang pendek harus diperhatikan dalam membuat tampilan sehingga tidak menyulitkan pengguna.

2.1.5. Teori *Software Development*

2.1.5.1. Metode Perancangan SDLC

System Development Life Cycle (SDLC) adalah proses keseluruhan meliputi pengembangan, implementasi, dan akhir penggunaan sistem informasi melalui proses bertingkat dari inisiasi, analisis, desain, implementasi, dan *maintenance* (Radack, 2009).

2.1.5.2. Model Proses Waterfall

Metode pengembangan perangkat lunak yang digunakan untuk membangun aplikasi *e-learning* ini menggunakan paradigma model *waterfall* (Ian Sommerville, 2010), yang meliputi beberapa proses diantaranya:

1. *Requirements analysis and definition*

   Tahap ini mengumpulkan kebutuhan secara lengkap kemudian dianalisis dan didefinisikan kebutuhan yang harus dipenuhi oleh program yang akan dibangun. Tahap ini harus dikerjakan secara lengkap untuk bisa menghasilkan desain yang lengkap.

2. *System and software design*

   Tahap ini merupakan kegiatan mengumpulkan kebutuhan secara lengkap kemudian dianalisis dan didefinisikan kebutuhan yang harus dipenuhi oleh aplikasi yang akan dibangun. Tahap ini harus dikerjakan secara lengkap untuk bisa menghasilkan desain yang lengkap.

3. *Implementation and unit testing*

   Desain program diterjemahkan ke dalam kode-kode dengan menggunakan bahasa pemrograman yang sudah ditentukan. Program yang dibangun langsung diuji baik secara unit.

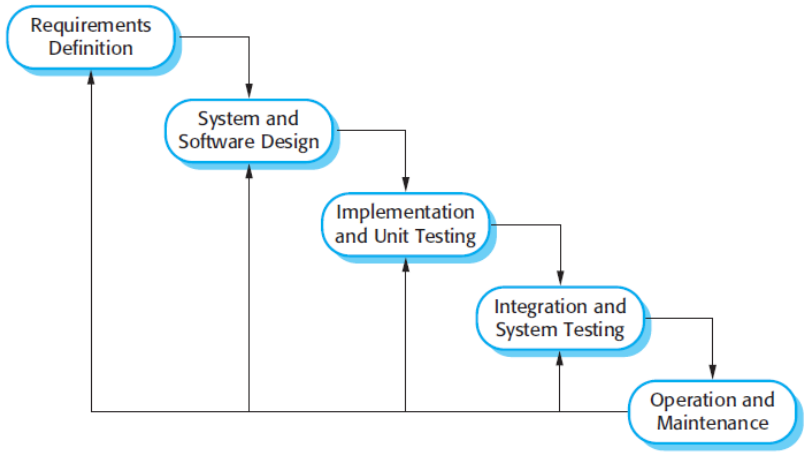
4. *Integration and system testing*

   Penyatuan unit-unit program kemudian diuji secara keseluruhan (*system testing*).

5. *Operation and maintenance*

   Mengoperasikan program dilingkungannya dan melakukan pemeliharaan, seperti penyesuaian atau perubahan karena adaptasi dengan situasi sebenarnya.

Secara umum tahapan pada model *waterfall* dapat dilihat pada gambar 2.3 dibawah ini:



Gambar 2.3. Model Waterfall

2.1.5.3. *Unified Modeling Language* (UML)

UML adalah bahasa pemodelan standar yang terdiri dari serangkaian diagram, dikembangkan untuk membantu sistem dan pengembang perangkat lunak menyelesaikan tugas-tugas berikut: Spesifikasi, Visualisasi, Desain arsitektur, Konstruksi, Simulasi dan Pengujian serta Dokumentasi

UML pada awalnya dikembangkan dengan ide mempromosikan komunikasi dan produktivitas antara pengembang sistem berorientasi objek, tetapi kekuatan tampak jelas dari UML telah menyebabkannya untuk membuat terobosan ke dalam setiap jenis sistem dan pengembangan perangkat lunak.

2.1.5.3.1. Konsep Dasar UML

Dari berbagai penjelasan rumit yang terdapat di dokumen dan buku-buku UML. Sebenarnya konsepsi dasar UML bisa kita rangkumkan dalam gambar 2.4 dibawah ini:

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

Gambar 2.4. Konsep Dasar UML

2.1.5.3.2. *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

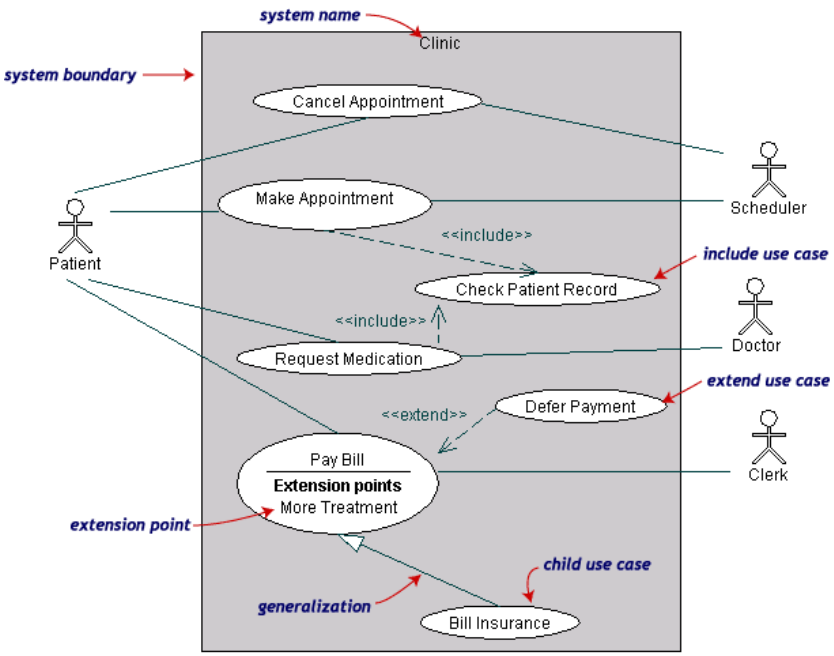
dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya *login* ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.

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 us*e case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain.

Contoh *use case* diagram dapat dilihat pada gambar 2.5 dibawah ini:



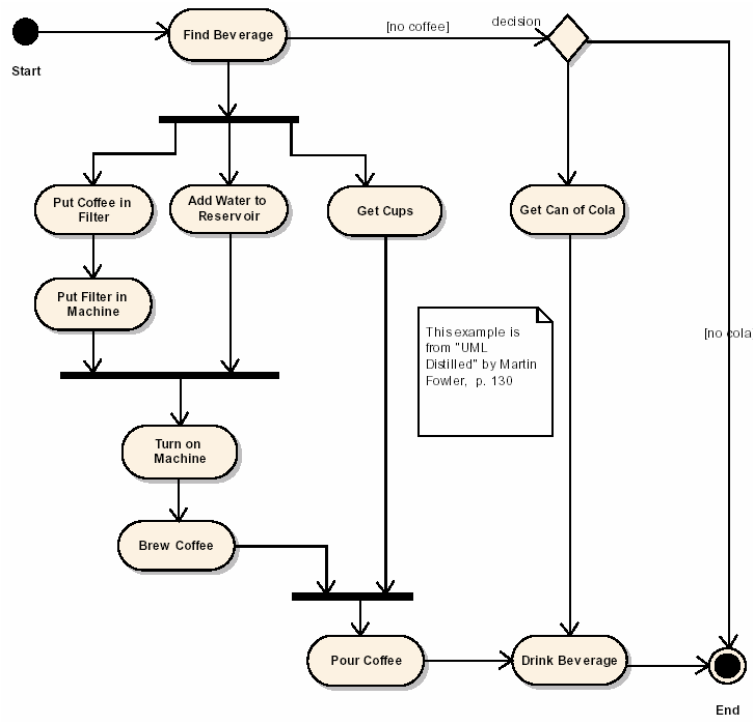
Gambar 2.5. Contoh Use Case Diagram

2.1.5.3.3. *Acivity 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. Contoh *activity diagram* tanpa *swimlane* bisa dilihat pada gambar 2.6 dibawah ini:



Gambar 2.6. Contoh Activity Diagram Tanpa Swimlane

2.1.5.3.4. *Sequence Diagram*

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

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

Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal. *Message* digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada tahap desain berikutnya, *message* akan dipetakan menjadi operasi/metoda dari *class*. *Activation* bar menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah *message*.

Untuk objek-objek yang memiliki sifat khusus, standar UML mendefinisikan *icon* khusus untuk objek *boundary*, *controller* dan *persistent entity*.

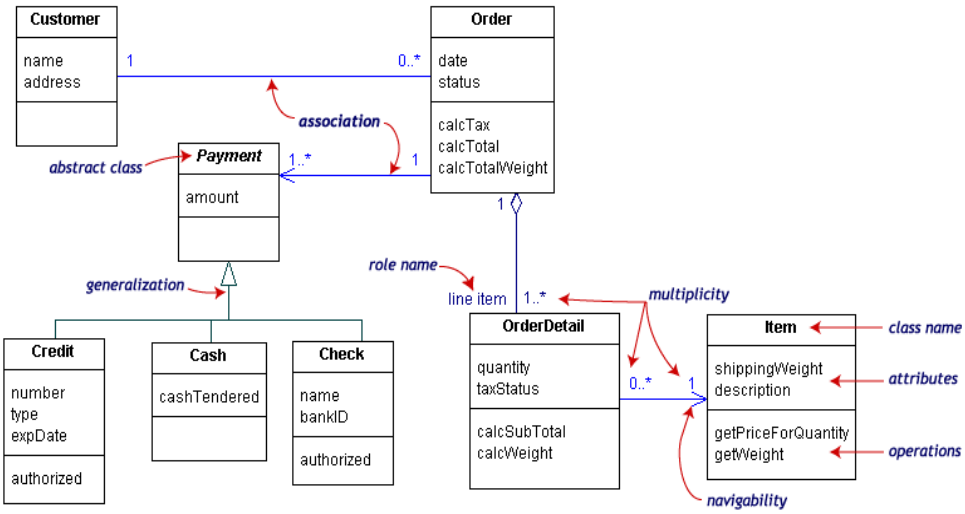
2.1.5.3.5. *Class Diagram*

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi). *Class diagram* menggambarkan struktur dan deskripsi *class*, *package* dan *objek* beserta hubungan satu sama lain seperti *containment*, *pewarisan*, *asosiasi*, dan lain-lain.

Class memiliki tiga area pokok, yaitu: Nama (dan *stereotype*), Atribut dan Metoda. Atribut dan metoda dapat memiliki salah satu sifat berikut.

1. *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan.
2. *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya.
3. *Public*, dapat dipanggil oleh siapa saja.

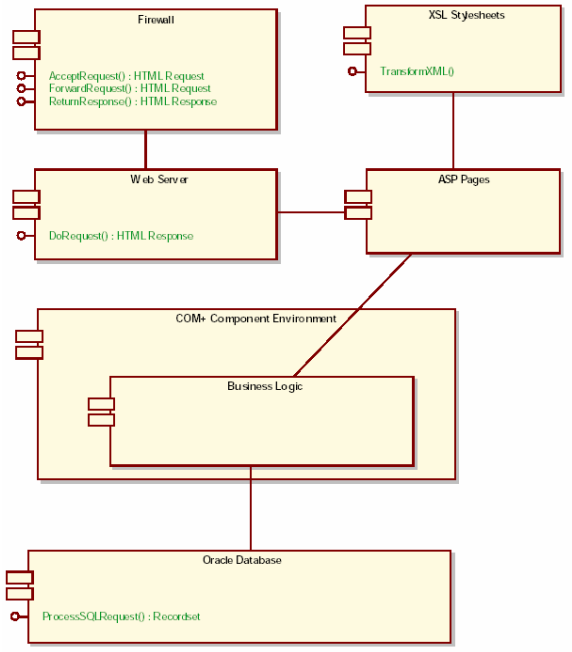
Contoh *class diagram* dapat dilihat pada gambar 2.7 dibawah ini:



Gambar 2.7. Contoh Class Diagram

2.1.5.3.6. *Component Diagram*

Component diagram menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan *(dependency)* di antaranya. Komponen piranti lunak adalah modul berisi *code*, baik berisi *source code* maupun *binary code*, baik *library* maupun *executable*, baik yang muncul pada *compile time, link time*, maupun *run time*. Umumnya komponen terbentuk dari beberapa *class* dan/atau *package*, tapi dapat juga dari komponen-komponen yang lebih kecil. Komponen dapat juga berupa *interface*, yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lain. Contoh *component diagram* dapat dilihat pada gambar 2.8 dibawah ini:

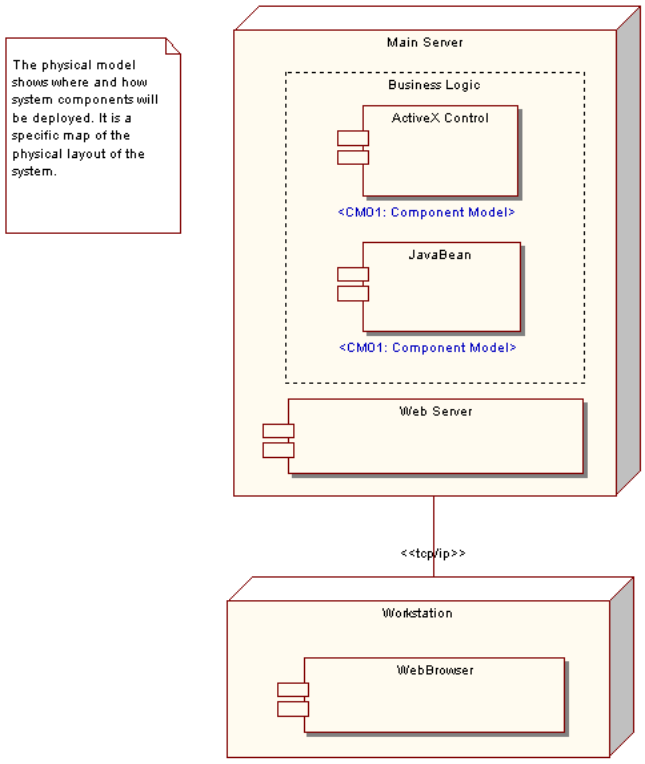


Gambar 2.8. Contoh component diagram

2.1.5.3.7. *Deployment Diagram*

Deployment/physical diagram menggambarkan detail bagaimana komponen di-*deploy* dalam infrastruktur sistem, di mana komponen akan terletak (pada mesin, *server* atau piranti keras apa), bagaimana kemampuan jaringan pada lokasi tersebut, spesifikasi *server*, dan hal-hal lain yang bersifat fisikal.

Sebuah *node* adalah *server*, *workstation*, atau piranti keras lain yang digunakan untuk men-*deploy* komponen dalam lingkungan sebenarnya. Hubungan antar *node* (misalnya TCP/IP) dan *requirement* dapat juga didefinisikan dalam diagram 2.9 dibawah ini.



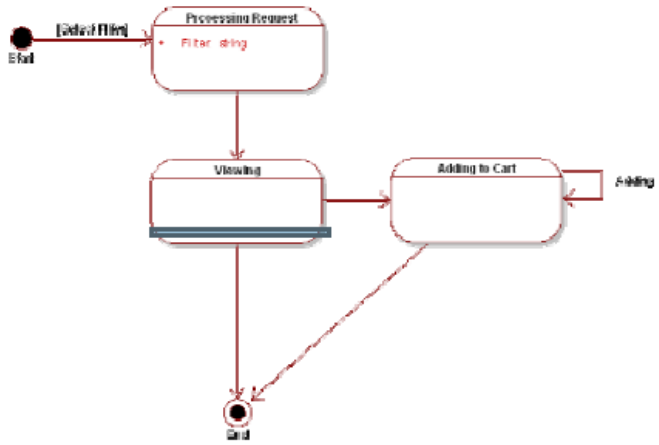
Gambar 2.9. Contoh Deployment Diagram

2.1.5.3.8. *State Diagram*

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

Dalam UML, *state* digambarkan berbentuk segiempat dengan sudut membulat dan memiliki nama sesuai kondisinya saat itu. Transisi antar *state* umumnya memiliki kondisi *guard* yang merupakan syarat terjadinya transisi yang bersangkutan, dituliskan dalam kurung siku. *Action* yang dilakukan sebagai akibat

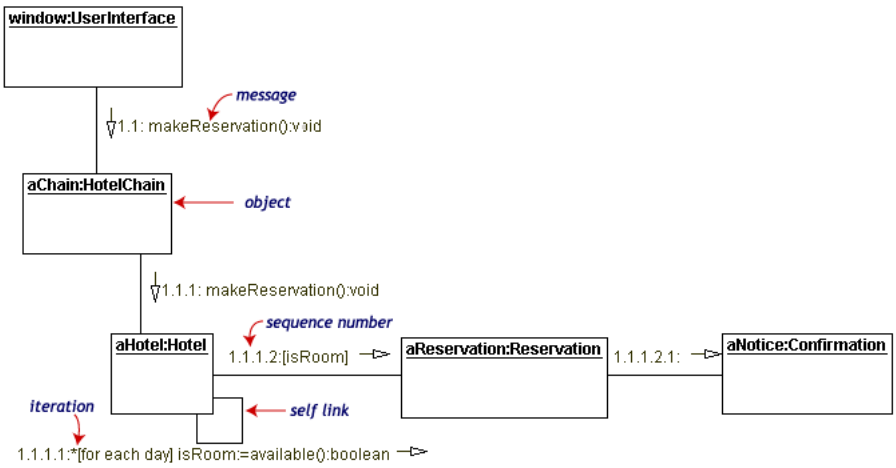
dari *event* tertentu dituliskan dengan diawali garis miring. Titik awal dan akhir digambarkan berbentuk lingkaran berwarna penuh dan berwarna setengah. Untuk lebih jelasnya dapat dilihat pada gambar 2.10 dibawah ini:



Gambar 2.10. Contoh state diagram

2.1.5.3.9. *Colaboration Diagram*

Collaboration diagram juga menggambarkan interaksi antar objek seperti *sequence diagram*, tetapi lebih menekankan pada peran masing-masing objek dan bukan pada waktu penyampaian message. Setiap message memiliki *sequence number*, di mana *message* dari level tertinggi memiliki nomor 1. *Messages* dari level yang sama memiliki *prefiks* yang sama. Untuk lebih jelasnya dapat dilihat pada gambar 2.11 dibawah ini:



Gambar 2.11. Contoh colaboration diagram