BAB III

OBJEK DAN METODE PENELITIAN

3.1 Objek Penelitian

Dalam sub bab ini, penulis akan menguraikan mengenai objek penelitian yang meliputi sejarah Hotel Royal Corner Bandung, struktur organisasi yang akan diuraikan sebagai berikut :

3.1.1 Sejarah Singkat Hotel Royal Corner

Hotel Royal Corner merupakan hotel bintang satu yang berlokasi di pusat kota Jl. Wastukencana No. 8 Bandung. Hotel ini bersebelahan dengan pusat pemerintahan dan DPRD kota Bandung, dan juga dengan pusat perbelanjaan, seperti Pasar Baru, Pusat Elektronik (BEC), *dan* pusat hiburan *lainnya*.Banyak kegiatan pemerintah daerah yang melakukan studi banding ke Pemkot Bandung ataupun ke DPRD Kota Bandung, mereka menginap di Hotel Royal Corner, karena untuk menempuh lokasi (Pemkot dan DRPD Kota Bandung) tersebut cukup dengan jalan kaki.

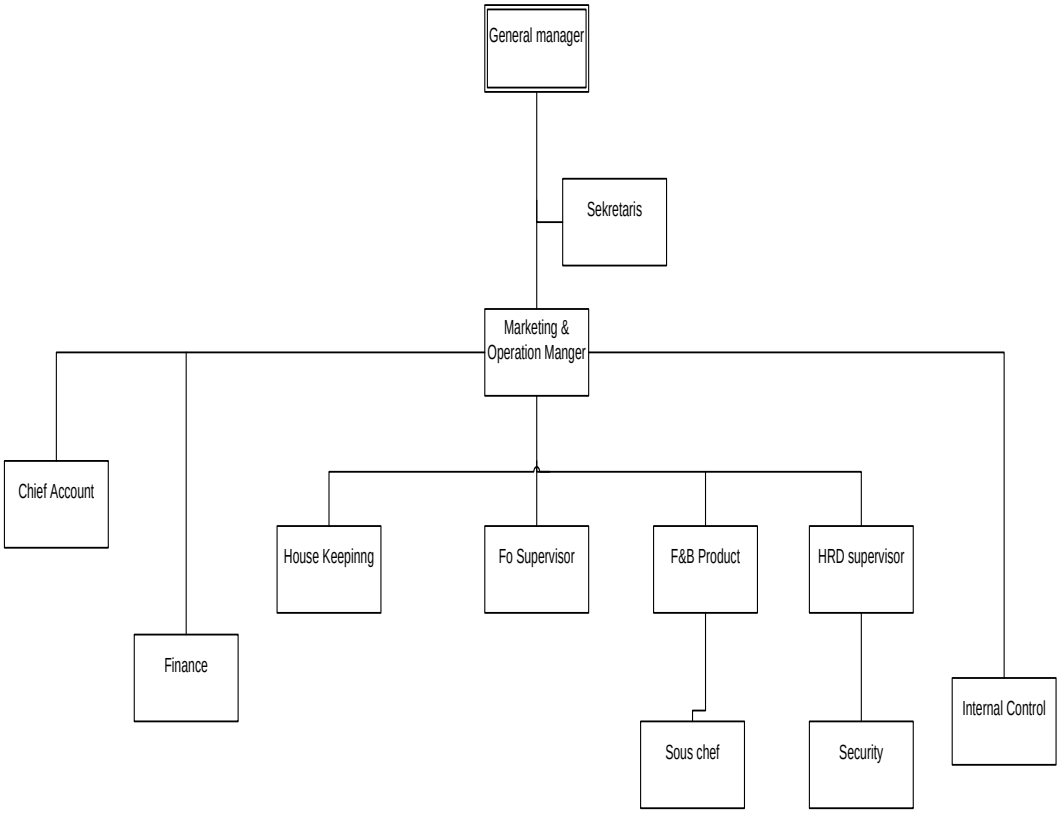
Hotel Royal Corner yang diminati oleh turis Malaysia ini memiliki 40 kamar dengan fasilitas AC, air panas, telepon, Wifi, TV Cable, ruang*meeting*, dan fasilitas parkir yang luas, dan dapat menampung Bis besar.Hotel ini sangat cocok untuk keluarga, bisnis dan yang memerlukan pelayanan *meeting* dan makan, dengan kapasitas 20 – 200 pax, karena kami memiliki 3 ruang *meeting.*

3.1.2 Visi dan Misi

Visi: Menjadikan Hotel Bisnis yang nyaman, berkualitas dan terbaik dikelasnya.

Misi: Menyediakan fasilitas kamar dan ruang meeting yang bersih dan nyaman dengan pelayanan yang profesional, yang berorientasi pada kepuasan pelanggan, dan karyawan.

3.1.3 Struktur Organisasi Hotel Royal Corner



Gambar 3.1. Struktur Ogranisasi Hotel Royal Corner

Sumber: (Dokumen Hotel Royal Corner)

3.1.4.Deskripsi Tugas

Hotel Royal Corner merupakan perusahaan yang dipimpin oleh seorang manager. Dan dibantu oleh bagian – bagian yang telah diberi tugas masing-masing. Adapun tugas masing – masing bagian pada Hotel Royal Corner adalah sebagai berikut :

1. General Manager
    a. Menyiapkan dan memantau dan mengevaluasi businees plan, policy perusahaan dan sistem operasional.
    b. Melakukan kontrol dan evaluasi terhadap target penjualan dan penggunaan biaya seluruh bagian.
    c. Membuata bahasan dan merencanakan aktivitas dengan seluruh bagian dalam mencapai Sales volume, product dan quality Service.
    d. Mengotrol masa berlaku perijinan
2. Sekretaris
    a. Membuata agenda kerja pimpinan
    b. Mengarsipkan data – data perusahaan
    c. Membuat notulen rapat
3. Operation Manager
    a. Membantu tugas – tugas GM dalam bidang marketing dan operasional serta pengawasan langsung diseluruh bagian
    b. Mengkoordinir pengawasan, pembinaan dan peningkatan sumber daya manusia diseluruh bagian
    c. Melakasanakan dan mengawasi policy dan goal hotel yang telah ditetapkan
4. HRD supervisor
    a. Melakukan pengawasan dan pembinaan terhadap Sumber daya manusia serta kegiatan di bagian HRD
    b. Membuat rencana kerja action plan dalam mendukung Goal hotel

c. Memimpin tersedianya data pegawai

d. Memimpin tersedianya dta perusahaan

5. Chief Accounting

a. Melakukan pengawasan dan pembinaan terhadap Sumber daya manusia serta kegiatan di bagian accounting

b. Membuat program kerja diaccounting

6. Finance

a. Membuat laporan kas mutasi

b. Membuat laporan Cash flow

c. Mengecek rekening koran dan transaksi Bank

d. Menyiapkan pembayaran Gaji Karyawan

e. Membuat laporan pajak

7. Internal Control

a. Memeriksa kesesuaian harga dan jumlah barang

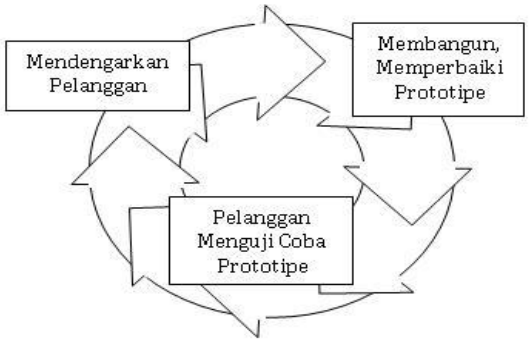
b. Menolak barang yang tidak sesuai dan spesifikasi yang telah ditentukan

c. Memeriksa kesesuaian bill

d. Menyusun standard resep bersama Sous chef

3.2 Metode Pengembangan Sistem

Metode yang digunakan penulis dalam melakukan pengembangan sistem Informasi ini yaitu menggunakan model *prototype.* Model *prototype* adalah metode pengembangan sistem yang digunakan penulis dalam penelitian ini. Karena penulis memulai penelitian ini dengan mengumpulkan kebutuhan yang diperlukan dari pada sistem atau perangkat lunak yang akan dibuat.



Gambar 3.2 prototipe Model

(Sumber: Roger S. Pressman,Ph.D., 2002, Rekayasa Perangkat Lunak, Andi)

Proses-proses tersebut dapat dijelaskan sebagai berikut:

1. Pengumpulan kebutuhan: *developer* dan klien bertemu dan menentukan tujuan umum, kebutuhan yang diketahui dan gambaran bagian-bagian yang akan dibutuhkan berikutnya;

2. Perancangan: perancangan dilakukan cepat dan rancangan mewakili semua aspek software yang diketahui, dan rancangan ini menjadi dasar pembuatan prototype;

3. Evaluasi Prototype: klien mengevaluasi prototype yang dibuat dan digunakan untuk memperjelas kebutuhan software

Untuk memodelkan sebuah perangkat lunak, metode prototyping memiliki tahapan-tahapan di dalam proses pengembangannya. Tahapan inilah yang

menentukan keberhasilan dari sebuah *software.*Pengembang perangkat lunak harus memperhatikan tahapan dalam metode prototyping agar *software final*nya dapat diterima oleh pemakai. Dan tahapan-tahapan dalam prototyping tersebut adalah sebagai berikut :

1. Pengumpulan kebutuhan

   Pelanggan dan pengembang bersama-sama mendefinisikan format seluruh perangkat lunak, mengidentifikasikan semua kebutuhan, dan garis besar sistem yang akan dibuat.

2. Membangun prototyping

   Membangun prototyping dengan membuat perancangan sementara yang berfokus pada penyajian kepada pelanggan (misalnya dengan membuat input dan format output).

3. Evaluasi protoptyping

   Evaluasi ini dilakukan oleh pelanggan apakah prototyping yang sudah dibangun sudah sesuai dengan keinginan pelanggan. Jika sudah sesuai maka langkah keempat akan diambil. Jika tidak, maka prototyping direvisi dengan mengulang langkah 1, 2 , dan 3.

4. Mengkodekan system

   Dalam tahap ini prototyping yang sudah disepakati diterjemahkan ke dalam bahasa pemrograman yang sesuai.

5. Menguji system

   Setelah sistem sudah menjadi suatu perangkat lunak yang siap pakai, harus dites dahulu sebelum digunakan. Pengujian ini dilakukan dengan White Box, Black Box, Basis Path, pengujian arsitektur dan lain-lain.

6. Evaluasi Sistem

   Pelanggan mengevaluasi apakah sistem yang sudah jadi sudah sesuai dengan yang diharapkan . Jika sudah, maka langkah ketujuh dilakukan, jika belum maka mengulangi langkah 4 dan 5.

7. Menggunakan system

   Perangkat lunak yang telah diuji dan diterima pelanggan siap untuk digunakan.

3.3 Pemrograman Berorientasi Objek

Menurut [Nug02] Pemrograman berorientasi objek sebagia berikut :

"Pemrograman berorientasi objek atau OOP (Object Oriented Programing) adalah suatu cara baru dalam berpikir serta berlogika dalam menghadapi masalah-masalah yang akan dihadapi dengan bantuan komputer."

Seiring dengan munculnya OOP (*Object Oriented Programing*) maka telah banyak bermunculan metodologi pemodelan berorientasi objek, karena banyaknya metodologi yang ada maka tiga metodologis terkenal yaitu Grady Booch, James Rumbaugh, dan Ivar Jacobson mengkombinasikan metode-metode mereka untuk memperoleh notas-notasi yang dapat digunakan seluruh metodologi berorientasi objek yang dikemukakan para metodologis. Mereka membentuk notasi yang dapat digunakan bersama yaitu : *Unified Modeling Language (UML)*.

Menurut [Online 2] pengertian UML sebagai berikut :

"UML adalah salah satu bentuk notasi atau bahasa yang sama yang digunakan oleh professional dibidang software untuk menggambarkan atau memodelkan sebuah sistem software."

UML telah disahkan oleh OMG (*Object Management Group*) pada tahun 1997 sebagai standar *de-facto* dari pemodelan OOP dan diterima sebagai salah satu standar industri. Secara garis besar UML merupakan standard bahasa pemodelan untuk pembuatan object-oriented software dan merupakan kombinasi dari:

1. Konsep Pemodelan Data (*Entity Relationship Diagrams*)
2. Pemodelan Bisinis (*Work Flow*)

3. Pemodelan *Object*,
4. Pemodelan Komponen

Spesifikasi UML mendefinisikan sekumpulan diagram grafis sebagai tampilan dari beberapa level abstraksi dan UML dapat digunakan bersama oleh semua proses pada keseluruhan tahap siklus-hidup *(life-cycle)* pengembangan *software* serta pada implementasi ke beberapa teknologi yang berbeda.

Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan *syntax*/semantik. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram perangkat 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, yaitu: Grady Booch OOD *(Object-Oriented Design)*, Jim Rumbaugh OMT *(Object Modeling Technique)*, dan Ivar Jacobson OOSE *(Object-Oriented Software Engineering)*.

3.3.1 *Object Management Group* (OMG)

OMG adalah konsorsium yang beranggotakan lebih dari 850 perusahaan untuk mendefinisikan standar-standar teknilogi objek termasuk CORBA (*Common Object Request Broker Architectur*) yang dibentuk pada tahun 1989 yang ditujukan untuk mempromosikan penggunaan teknologi berarah obyek pada aplikasi software.

Unified Modeling Language (UML) merupakan sistem arsitektur yang bekerja dalam OOAD (*Object-Oriented Analysis/Design*) dengan satu bahasa

yang konsisten untuk menentukan, visualisasi, mengkontruksi, dan mendokumentasikan *artifact* (sepotong informasi yang digunakan atau dihasilkan dalam suatu proses rekayasa software, dapat berupa model, deskripsi, atau software) yang terdapat dalam sistem software. UML merupakan bahasa pemodelan yang paling sukses dari tiga metode OO yang telah ada sebelumnya, yaitu Booch, OMT (*Object Modeling Technique*), dan OOSE (*Object-Oriented Software Engineering*).

UML merupakan kesatuan dari dari ketiga pemodelan tersebut dan ditambah kemampuan lebih karena mengandung metode tambahan untuk mengatasi masalah pemodelan yang tidak dapat ditangani ketiga metode tersebut. UML dikeluarkan oleh OMG (Object Management Group, Inc) yaitu organisasi internasional yang dibentuk pada 1989, terdiri dari perusahaan sistem informasi, *software developer*, dan para *user* sistem komputer.

3.3.2 Artifact UML

Diagram – diagram yang digunakan untuk mendefinisikan *UML* adalah sebagai berikut:

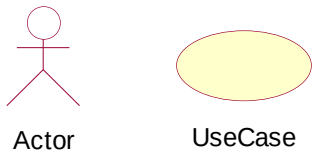
a. Use Case Diagram

Sebuah *use case* menggambarkan suatu urutan interaksi antara satu atau lebih aktor dan sistem. Dalam fase *requirements*, model *use case* mengambarkan sistem sebagai sebuah kotak hitam dan interaksi antara aktor dan sistem dalam suatu bentuk naratif, yang terdiri dari input user dan respon-respon sistem. Setiap

use case menggambarkan perilaku sejumlah aspek sistem, tanpa mengurangi struktur internalnya. Selama pembuatan model *use case* secara pararel juga harus ditetapkan objek-objek yang terlibat dalam setiap *use case*.

Use case Menggambarkan sejumlah *external actors* dan hubungannya ke *use case* yang diberikan oleh sistem. *Use case* adalah deskripsi fungsi yang disediakan oleh sistem kedalam bentuk teks sebagai dokumentasi dari *use case symbol* namun dapat juga dilakukan dalam *activity diagrams*.

Use case digambarkan hanya yang dilihat dari luar oleh *actor* (keadaan lingkungan sistem yang dilihat user) dan bukan bagaimana fungsi yang ada di dalam sistem. Simbol yang digunakan yaitu:



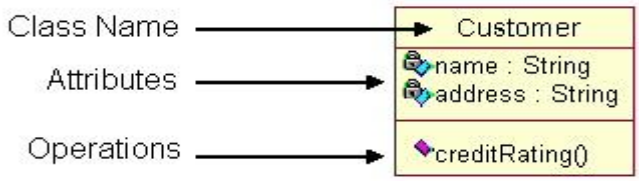
Gambar 3.3 *Use Case* Model

(Sumber: [Lar05])

b. Class Diagram

Menggambarkan struktur statis *class* di dalam sistem. *Class* merepresentasikan sesuatu yang ditangani oleh sistem. *Class* dapat berhubungan dengan yang lain melalui berbagai cara: *associated* (terhubung satu sama lain), *dependent* (satu *class* tergantung/menggunakan *class* yang lain), *specialed* (satu *class* merupakan spesialisasi dari *class* lainnya) atau *package* (grup bersama sebagai satu unit). Sebuah sistem biasanya mempunyai beberapa *class diagram*.

Suatu *class* biasanya terdiri dari 3 bagian, yaitu nama, atribut, dan operasi. Berikut adalah contoh dari suatu *class*:



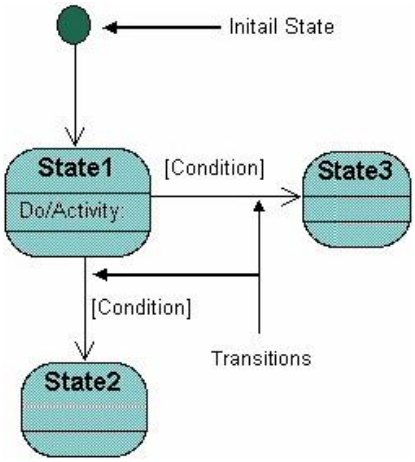
Gambar 3.4 Bagian-bagian dari c*lass*

(Sumber: [Lar05])

c. Statechart Diagram

Statechart Diagram merupakan 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 dengan 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 titik akhir digambarkan berbentuk lingkaran berwarna penuh dan berwarna setengah.



Gambar 3.5 *State Diagram*

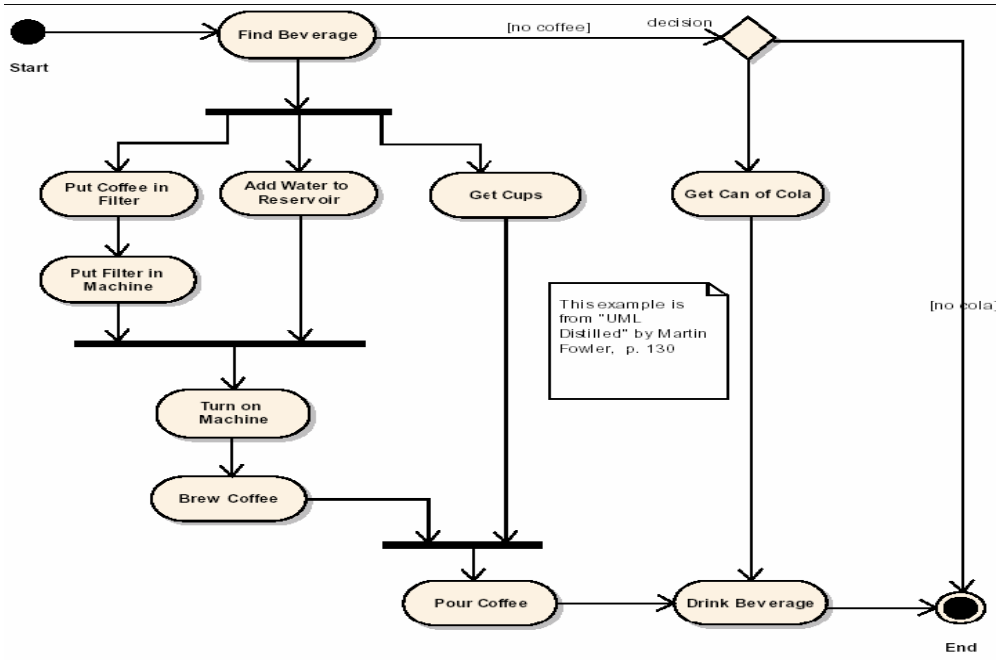
(Sumber: [Lar05])

d. Activity Diagram

Activity Diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing - masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana aktivitas itu berakhir. *Activity diagram* juga dapat menggambarkan proses pararel yang mungkin terjadi pada beberapa eksekusi. *Activity diagram* merupakan *state diagram* khusus,dimana sebagian besar *state* adalah *action* dan sebagian besar transisi di *triger* 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 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 dengan *state*, standar *UML* menggunakan segi empat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan *behaviour* pada kondisi tertentu. Untuk mengilustrasikan proses-proses pararel (*fork on 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.



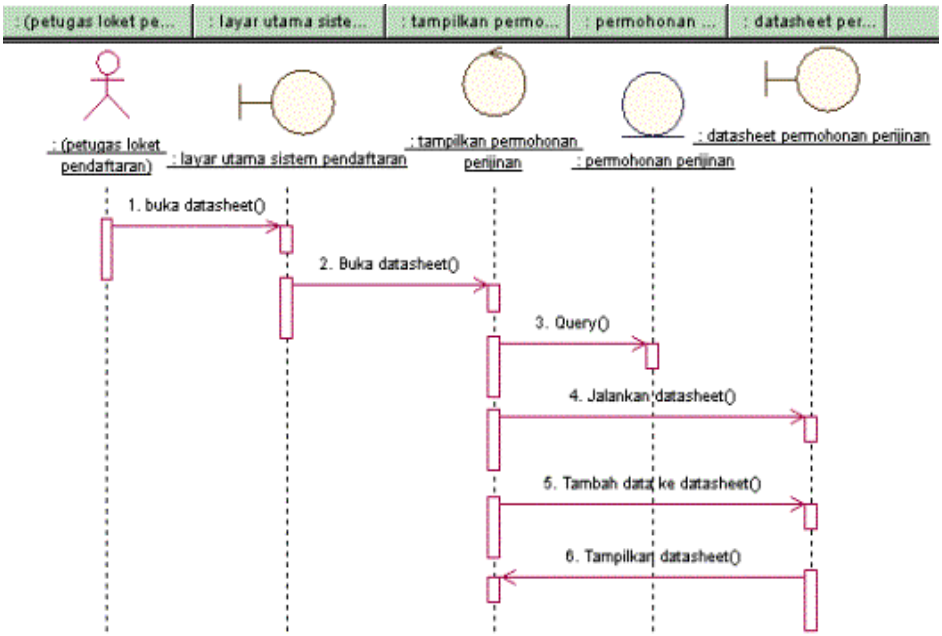
Gambar 3.6 *Activity Diagram*

(Sumber: [Lar05])

e. Sequence Diagram

Menggambarkan kolaborasi dinamis antara sejumlah *object*. Kegunaannya untuk menunjukkan rangkaian pesan yang dikirim antara *object* juga interaksi antara *object*, sesuatu yang terjadi pada titik tertentu dalam eksekusi sistem.

Komponen utama *sequence diagram* terdiri atas objek yang dituliskan dengan kotak segiempat bernama pesan diwakili oleh garis dengan tanda panah dan waktu yang ditunjukkan dengan proses vertikal.



Gambar 3.7 *Sequence Diagram*

(Sumber: [Lar05])

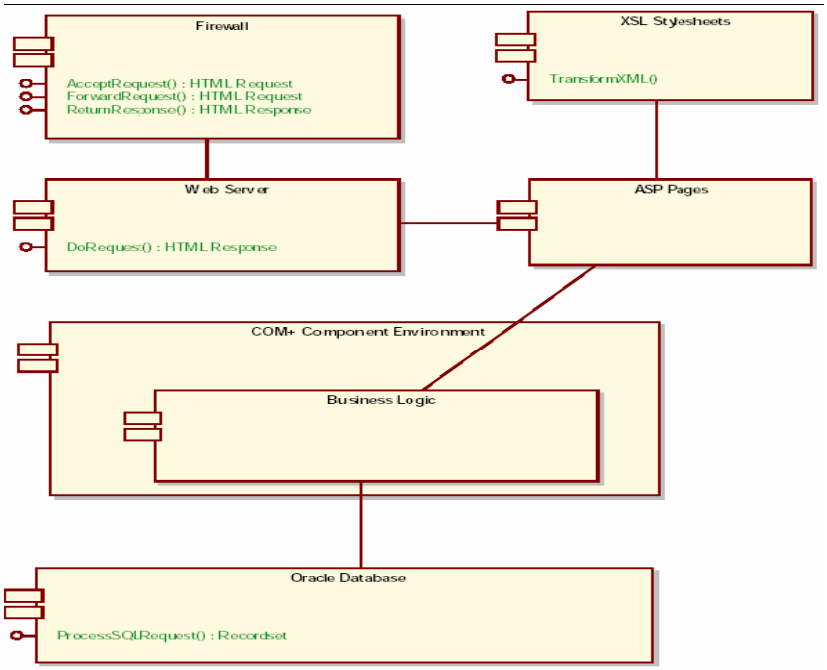
f. Colaboration Diagram

Menggambarkan kolaborasi dinamis seperti *sequence diagrams*. Dalam menunjukkan pertukaran pesan, *collaboration diagrams* menggambarkan object dan hubungannya (mengacu ke konteks). Jika penekanannya pada waktu atau urutan maka gunakan *sequence diagrams*, tapi jika penekanannya pada konteks gunakan *collaboration diagrams*.

g. Component Diagram

Component diagram menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan (*dependency)* diantaranya.

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 komponent terbentuk dari beberapa *class* atau *package* tapi juga dari komponen - komponen yang lebih kecil. Komponen dapat juga berupa *interface*, yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lain.



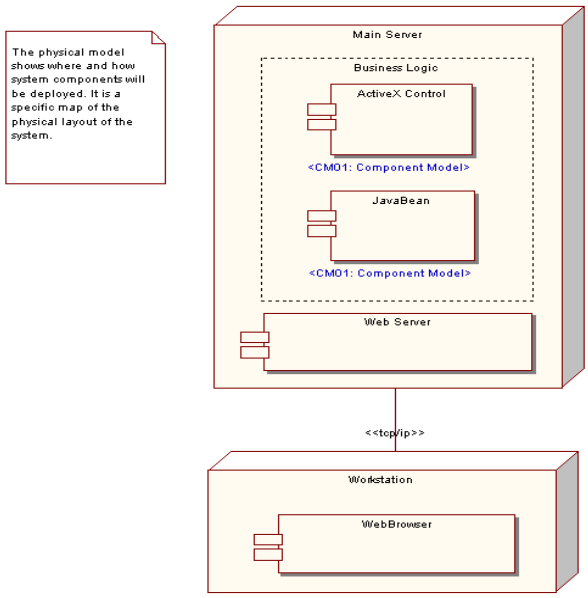
Gambar 3.8 *Component Diagram*

(Sumber: [Lar05])

h. Deployment Diagram

Deployment / phisycal diagram menggambarkan detail bagaimana komponen di-*deploy* dalam infrastruktur sistem, dimana 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 ini.



Gambar 3.9 *Deployment Diagram*

(Sumber: [Lar05])

3.3.3 Semantik Dalam UML

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 yaitu *Grady Booch OOD (Object-Oriented Design)*, *Jim Rumbaugh OMT (Object Modeling Technique)*, dan *Ivar Jacobson OOSE (Object-Oriented Software Engineering)* [Online 3].

OMG telah menetapkan semantik (makna istilah) semua notasi *UML* dalam *Model Struktural* dan *Model Behavioral.*

1. *Model Struktural* atau model statis, menekankan pada struktur objek dalam sebuah sistem, menyangkut kelas - kelas, *interface*, *atribute* dan hubungan antar komponen.
2. *Model Behavioral* atau model dinamis, menekankan pada perilaku objek dalam sebuah sistem, menyangkut metode, interaksi, kolaborasi dan *state history*.

3.3.4 Tujuan UML

Tujuan utama dari perancangan UML adalah :

1. Menyediakan bahasa pemodelan visual yang ekspresif dan siap pakai untuk mengembangkan dan pertukaran model-model yang berarti.
2. Menyediakan mekanisme perluasan dan spesialisasi untuk memperluas konsep-konsep ini.
3. Mendukung spesifikasi independen bahasa pemrograman dan proses pengembangan tertentu.
4. Menyediakan basis formal untuk pemahaman bahasa pemodelan.
5. Mendorong pertumbuhan pasar kakas berorientasi objek.
6. Mendukung konsep-konsep pengembangan level lebih tinggi seperti komponen, kolaborasi, *framework* dan *pattern*.

3.3.5 Cakupan UML

Model *UML* dapat mencakup banyak perbendaharaan. Diantaranya adalah sebagai berikut :

1. *Things*
   a. *Structural*:
      1. *Use case* : deskripsi interaksi dengan *external actor*.
      2. *Class* : deskripsi untuk objek – objek.
      3. *Interface* : kumpulan operasi yang memberikan service tertentu untuk *class / component.*
      4. *Component* : bagian sistem yang dapat diganti *(replaceable)* dan realisasinya sesuai dengan interface.
   b. *Behaviour*
      1. Interaksi *(message sequence chart)*: pertukaran messages antar objek.
      2. *State machine*: urutan state dari objek dalam berinteraksi dengan objek lain.
   c. *Grouping*

      Package: mekanisme untuk mengumpulkan elemen ke dalam satu set (group).
   d. *Anotasi*

      Catatan atau keterangan (teks) sebagai dokumentasi.
2. *Relationship*
   a. *Dependency*

      Hubungan antar element dimana perubahan pada elemen yang satu dapat mempengaruhi elemen yang lain *(dependent)*.

b. Association

Hubungan struktur antara elemen dan bertindak sebagai *link.*

c. Generalization

Hubungan dimana elemen yang special *(child)* mewarisi elemen yang umum *(parent).*

d. Realization

1. Hubungan (semantik) antara 2 elemen, dimana elemen yang satu memberikan kontrak dan elemen yang lain menjamin realisasi kontrak tersebut.
2. Dimana elemen yang special *(child)* mewarisi elemen yang umum *(parent).*

3. *Diagram*: *use case diagram, class diagram, msc diagram,* dan lain – lain.

Selain cakupan – cakupan di atas, ada tiga aspek utama dalam pemodelan sistem yang mampu didukung oleh *UML*:

a. *Functional Model*, untuk menunjukkan fungsionalitas dari suatu sistem dari sudut pandang user atau pengguna. Ini dicapai dengan menggunakan *Use Case Diagram.*

b. *Object Model,* untuk menunjukkan struktur dan substruktur dari suatu sistem dengan menggunakan object, atribut, operasi dan juga asosiasi. Ini dicapai dengan menggunakan *Class Diagram*.

c. *Dynamic Model*, menunjukkan *internal behavior* dan suatu sistem. Ini dicapai dengan menggunakan *Sequence Diagram, Activity Diagram* dan juga *Statechart Diagram*.

3.3.6 Perancangan Basis Data

Sebelum kita membuat basis data, terlebih dahulu harus dilakukan suatu perancangan. Proses perancangan ini bersifat konseptual. Kita belum menentukan DBMS apa yang akan kita gunakan untuk mengimplementasikan rancangan basis data yang dibuat. Tujuan perancangan basis data adalah untuk mendapatkan skema basis data yang meminimasi terjadinya redudansi dan duplikasi data serta menjaga integritas data. Kebanyakan metode perancangan yang ada berbasis pada model basis data relasional, yaitu struktur data diatur melalui pembuatan tabel-tabel dan keterkaitan antar tabel satu dengan yang lainnya *(*relasi*)*.

a. *Multiplicity*

Multiplicity pada kasus asosiasi menunjukkan bahwa ada sejumlah objek pada sebuah *class* yang berhubungan dengan sebuah objek pada sebuah asosiasi *class*, ada banyak multiplicity yang mungkin untuk dipakai. Sebuah class dapat berhubungan dengan yang lain dalam satu ke satu *(one-to-one)*, satu ke banyak (*one-to-many*), satu ke satu atau lebih (*one-to-one or more*), satu ke nol atau satu, satu ke sebuah interval tertentu (contoh satu ke lima sampai dengan sepuluh), satu ke sejumlah n atau satu ke serangkaian pilihan (contoh satu ke sembilan atau sepuluh). Pada notasi UML, untuk menampilkan lebih atau banyak digunakan (*). Untuk menunjukkan ‘atau’ digunakan titik dua (..) seperti contohnya 1..* (berarti satu atau lebih). Notasi ‘atau’ bisa juga digunakan tanda koma (,).

b. 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 :

1. Nama (dan *stereotype)*
2. Atribut
3. 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.

Class dapat merupakan implementasi dari sebuah *interface*, yaitu *class* abstrak yang hanya memiliki metoda. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class.* Diagram demikian *interface* mendukung resolusi metoda pada saat *run-time.*

Sesuai dengan perkembangan *class* model, *class* dapat dikelompokan menjadi *package*. *Class* dapat membuat diagram yang terdiri atas *Package.*

Hubungan antar Class :

1. Asosiasi, yaitu hubungan statis antar class. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain. Panah *navigability* menunjukan arah query antar *class.*

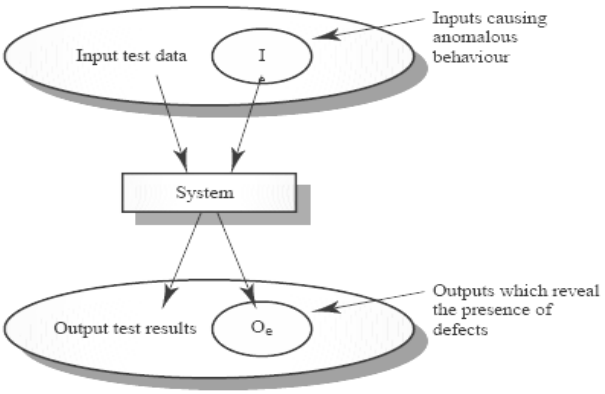
2. Agregasi, yaitu hubungan yang menyatakan bagian (“terdiri atas”).
3. Pewarisan, yaitu hubungan hirarkis antar *class. Class* dapat diturunkan dari class lain dan mewarisi semua atribut dan metoda *class* asalnya dan menambahkan fungsionalitas baru, sehingga ia disebut anak dari *class* yang diwarisinya. Kebalikan dari pewarisan adalah generalisasi.
4. Hubungan dinamis, yaitu rangkaian pesan (*message*) yang di *passing* dari satu *class* kepada *class* lain. Hubungan dinamis dapat digambarkan dengan menggunakan *sequence diagram* yang akan dijelaskan kemudian.

3.4 Pengujian *Black Box*

Pengujian *black-box* berfokus pada persyaratan fungsional perangkat lunak. Dengan demikian, pengujian *black-box* memungkinkan perekayasa perangkat lunak mendapatkan serangkaian kondisi input yang sepenuhnya menggunakan semua persyaratan fungsional untuk suatu program. Pengujian *black-box* bukan merupakan alternatif dari teknik *white-box*, tetapi merupakan pendekatan komplementer yang kemungkinan besar mampu mengungkap kelas kesalahan daripada metode *white-box*.

Pengujian *black-box* berusaha menemukan kesalahan dalam kategori sebagai berikut :

1. fungsi-fungsi yang tidak benar atau hilang.
2. kesalahan *interface.*
3. kesalahan dalam struktur data atau akses *database eksternal.*
4. kesalahan kinerja.
5. inisialisasi dan kesalahan terminasi.



Gambar 3.10 *Metode Pengujian Black-Box*