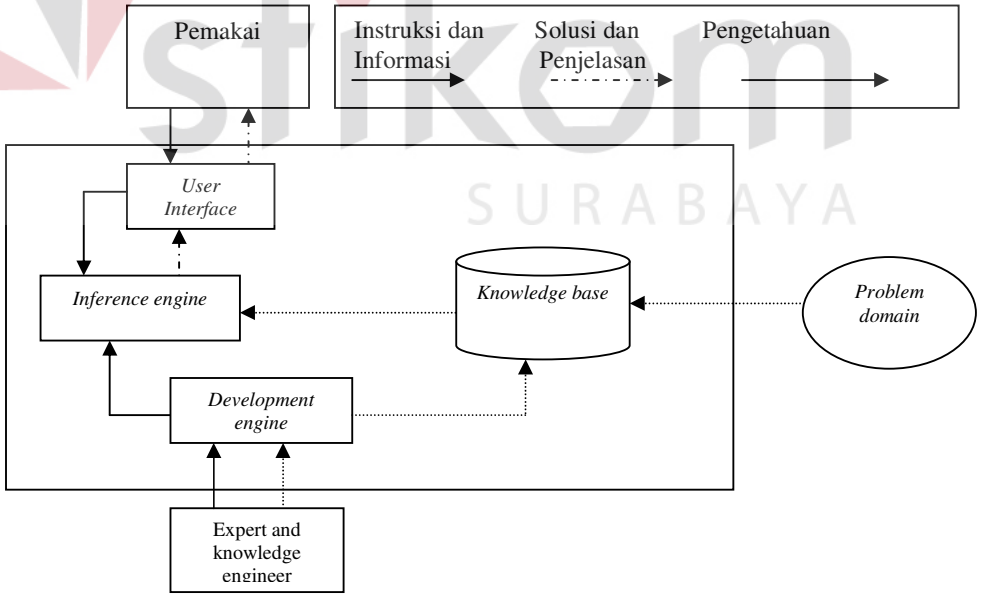
BAB II

LANDASAN TEORI

2.1 Sistem Pakar (Expert System)

Sistem Pakar (Expert System) merupakan suatu sistem perangkat lunak komputer yang memiliki basis pengetahuan untuk domain tertentu dan menggunakan penalaran inferensi menyerupai seorang pakar dalam memecahkan masalah. Digunakan untuk membantu mengambil keputusan dalam memilih berbagai alternatif keputusan yang merupakan hasil pengolahan informasi-informasi yang diperoleh nantinya akan tersedia dengan menggunakan model-model pengambilan keputusan.

Sistem pakar terdiri dari 4 bagian utama yaitu : *user interface, knowledge base, inference engine, dan development engine*.



Gambar 2.1 Model Sistem Pakar

Keterangan :

a. User Interface

Merupakan fasilitas komunikasi antara pemakai dengan sistem serta berfungsi sebagai media pemasukan informasi ke dalam basis pengetahuan.

b. Knowledge Base

Knowledge base adalah pengetahuan yang diperlukan untuk membuat suatu keputusan dimana didalam knowledge base memuat fakta-fakta dan juga teknik dalam menerangkan masalah yang menjelaskan bagaimana fakta-fakta tersebut cocok antara satu dengan yang lain dalam urutan yang logis. Knowledge Base terdiri atas 2 bagian yaitu fakta dan aturan. Dimana **Fakta** merupakan informasi tentang kenyataan atau kebenaran yang diketahui. Fakta menyatakan hubungan (relasi) antara dua objek (benda) atau lebih. Fakta dapat juga menunjukkan sifat suatu objek. Sedangkan **Aturan** adalah informasi tentang bagimanana membangkitkan fakta baru atau hipotesa dari fakta yang sudah diketahui. Sturuktur kaidah yang paling umum dipakai adalah : IF……..THEN.

c. Inference Engine

Menggunakan penalaran, yang serupa dengan manusia, dalam mengolah isi dari knowledge base berdasarkan urutan tertentu. Selama konsultasi, inference engine menguji aturan-aturan dari knowledge base satu demi satu, dan saat kondisi aturan itu benar tindakan tertentu diambil dan saat kondisi aturan itu salah akan dikesampingkan.

d. Development Engine

Terdiri dari bahasa pemograman atau prewritten inference engine yang disebut shell sistem pakar. Development engine adalah komponen utama keempat yang digunakan untuk menciptakan sistem pakar dimana didalamnya melibatkan pembuatan perangkat aturan dengan menggunakan bahasa pemrograman yang di spesialisasikan untuk sistem pakar.

Dalam memecahkan masalah dengan Sistem Pakar dapat menggunakan Rule Base System (Sistem yang berbasis pada aturan-aturan). Pada Rule Base System dapat menggunakan metode *Backward Reasoning*. Backward Reasoning dimulai dengan database dari fakta yang tidak diketahui dan dikosongi.

Known Fact Base : ( )

Sekumpulan tujuan/kesimpulan dibutuhkan untuk dapat dicapai.

Contoh :

Goals : (fruit)

Langkah-langkah pada backward reasoning adalah sebagai berikut :

1. Buat stack yang mulainya berisi semua top level goal yang didefinisikan dalam sistem.
2. Untuk goal yang pertama dari stack, kumpulkan rule-rule yang sesuai.
3. Untuk semua rule tersebut (2), kajilah premisnya :
   a. Bila semua premis untuk sebuah rule adalah cocok, kemudian eksekusi rule untuk mendapatkan kesimpulan. Jika nilai telah didapat untuk tujuan yang ada, hapus dari stack dan kembali ke langkah no. 2
   b. Bila ada sebuah premise dari rule tidak cocok, carilah rule yang memberikan parameter tertentu yang digunakan dalam premis tersebut. Bila dapat

ditemukan maka parameter tersebut dapat dijadikan subgoal dan ditempatkan sebagai top of stack, dan kembali ke nomor 2.

c. Bila pada langkah b tidak terpenuhi, minta user untuk memasukkan nilainya dan dimasukkan ke database. Bika nilainya memenuhi dengan premise yang diuji maka lanjutkan dengan premis pada rule tersebut. Jika premise tidak cocok, maka lanjutkan ke rule berikutnya.

4. Jika semua rule telah dicocokkan dengan tujuan yang ada dan semua gagal maka tujuan ini tidak dapat ditetapkan. Hapus dari stack dan kembali ke langkah no. 2. jika stack telah kosong (semua tujuan pucak yang ada telah dicoba), kemudian berhenti dan proses selesai.

2.2 Unified Modeling Language (UML)

Unified Modeling Language adalah sebuah tool yang digunakan untuk membangun, membuat, mendokumentasikan sebuah sistem. Komponen dari UML terdiri dari :

2.2.1 Use Case Diagram

Menurut Cavaness (2001:20) use case diagram mengambarkan interaksi actor (mis. user) dengan sebuah sistem software. Maksudnya untuk menggambarkan kebutuhan sistem (*system requirements*) berkenaan dengan fitur-fitur kongkrit (*concrete features*) yang dibutuhkan oleh *end user*. Dalam use case diagram terdapat 2 (dua) unsur yang saling berinteraksi, yaitu:

a. Actor

Actor merepresentasikan sebuah *external entity* yang berkomunikasi dengan sistem. Actor menyediakan input dan menerima informasi dari sistem. Bisa

berupa user, *external system* atau *physical environment.* Menurut Schmuller (1999:68) actor didefinisikan sebagai entity yang memulai (*initiate*) *sequence*.

b. Use case

Use case merepresentasikan sebuah rangkaian (*sequence*) dari interaksi fungsional. Menurut Schmuller (1999:68) use case merupakan sekumpulan skenario mengenai penggunaan sistem. Setiap skenario menggambarkan sebuah sequence dari kejadian-kejadian (*events*).

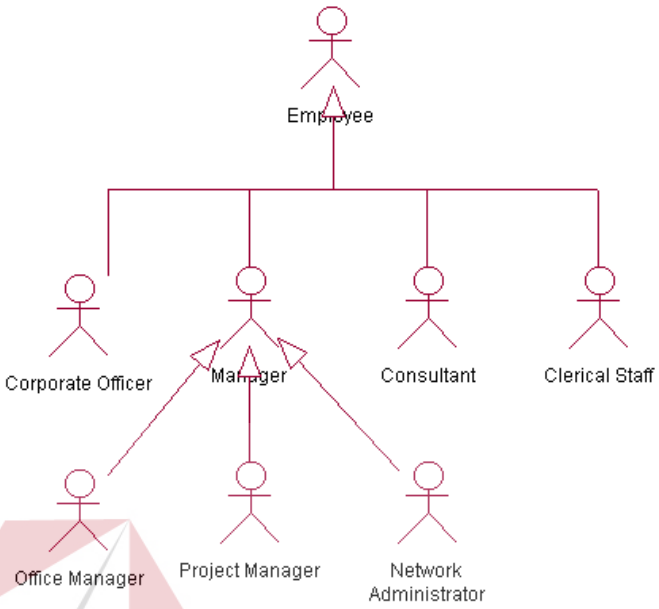
Interaksi atau hubungan (*relationship*) antara *actor* dengan use case ada 3 (tiga) jenis antara lain:

1. Asosiasi: relasi di mana dua obyek tidak saling tergantung, hanya sekadar berkomunikasi. Relasi ini merupakan hubungan yang paling umum.
2. Generalisasi (turunan): relasi di mana satu obyek merupakan generalisasi (memiliki sifat-sifat yang lebih umum) dari beberapa obyek.
3. Dependensi: relasi di mana perubahan yang terjadi pada satu obyek akan menimbulkan pengaruh pada obyek yang lain.

Stereotype yang digunakan untuk menandakan relasi antara actor dengan use case atau use case dengan use case antara lain:

a. <<communicate>>, menunjukkan actor berkomunikasi (berinterkasi) dengan use case.
b. <<include>>, artinya suatu use case akan dimasukkan secara otomatis oleh use case utamanya.
c. <<extend>>, artinya suatu use case akan diikuti oleh use case yang lain.

Contoh use case diagram dapat dilihat pada Gambar 2.1 dan Gambar 2.2.



Gambar 2.2. Contoh Use Case Diagram Yang Menggambarkan Relasi Generalisasi

Sumber: Schmuller, Joseph, 1999, Sams Teach Yourself UML in 24 Hours, Indiana, Sams Publishing, hal. 83

Dari gambar 2.1 dapat dilihat bahwa:

- actor `Employee` merupakan generalisasi dari actor `Corporate Officer`, `Manager`, `Consultant` dan `Clerical Staff`. Sedangkan actor `Manager` merupakan generalisasi dari actor `Office Manager`, `Project Manager` dan `Network Administrator`.
- actor `Corporate Officer`, `Manager`, `Consultant` dan `Clerical Staff` merupakan turunan dari actor `Employee`, sehingga

semua hak yang dimiliki `Employee` dimiliki juga oleh `Corporate Officer`, `Manager`, `Consultant` dan `Clerical Staff`. Demikian juga actor `Office Manager`, `Project Manager` dan `Network Administrator` karena merupakan turunan dari actor `Manager`, maka memiliki semua hak yang dimiliki oleh `Manager`.

Use case semacam ini biasanya digunakan untuk memodelkan hierarki user (pemakai sistem).

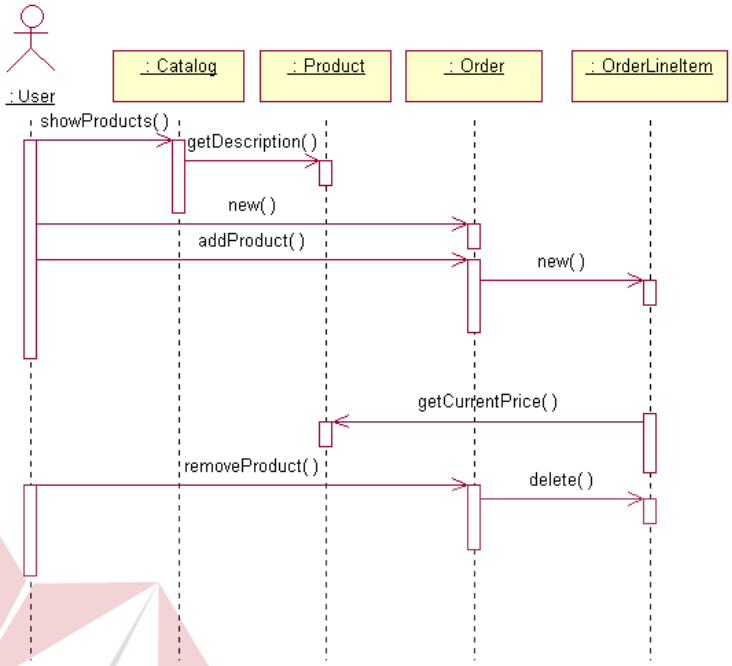
2.2.2 Sequence Diagram

Sequence diagram merupakan gambaran detail dari use case, interaksi antara actor dengan use case dijelaskan secara lebih rinci dalam sequence diagram. Obyek-obyek yang ada dalam sistem akan tampak di sequence diagram. "Dalam sequence diagram interaksi antar obyek dilakukan dengan cara mengirim message. Message digambarkan dengan anak panah dari satu *lifeline* obyek ke lifeline obyek yang lain, atau ke lifeline obyek itu sendiri" (Schmuller: 1999:104).

"Sequence diagram menggambarkan interaksi obyek yang dibutuhkan untuk menyelesaikan tugas-tugas yang dibutuhkan oleh sebuah use case" (Cavaness, 2001:20).

Menurut Schmuller (1999:104) sequence diagram terdiri dari obyek-obyek yang direpresentasikan dengan cara yang lazim: seperti kotak yang diberi nama yang diberi garis bawah, *messages* yang direpresentasikan dengan anak panah dan waktu yang direpresentasikan dengan *progress* vertikal.

Contoh sequence diagram dapat dilihat pada Gambar 2.3.



Gambar 2.3. Contoh Sequence Diagram

Sumber: Cavaness, Chuck and Friesen, Geoff, and Keeton, Brian 2001, Special Edition Using Java™ 2 Standar Edition, Indiana, Que Corporation, hal. 24

Dari Gambar 2.3 yang dimaksud dengan:

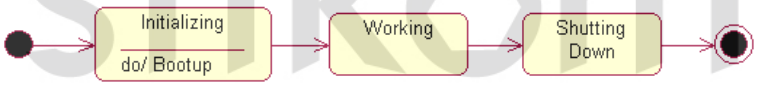
- obyek: `User`, `Catalog`, `Product`, `Order`, `OrderLineItem`
- messages: `showProducts()`, `getDescription()`, `new()`, `addProduct()`, `getCurrentPrice()`, `removeProduct()`, `delete()`,
- lifeline: garis putus-putus vertikal yang terletak di bawah obyek-obyek `User`, `Catalog`, `Product`, `Order`, `OrderLineItem`
- *focus of control*: balok pada lifeline yang menunjukkan periode berapa lama waktu yang dibutuhkan sebuah obyek melakukan suatu aksi.

Untuk mengecek keabsahan sequence diagram yang telah dibuat, obyek-obyek yang ada dalam sequence diagram perlu di-*mapping* dengan class yang ada dalam class diagram.

2.2.3 State Diagram

"*State-chart diagram* atau *state diagram* merepresentasikan keadaan-keadaan (*states*) sebuah obyek yang mungkin ada bersamaan dengan transisi antara states tersebut dan menunjukkan titik awal dan titik akhir dari rangkaian perubahan state" (Schmuller, 1999:92).

Sedangkan menurut Cavaness (2001:20) state diagram menunjukkan keadaan-keadaan (states) yang mungkin terjadi dalam sebuah obyek dan kejadian-kejadian (events) yang menyebabkan timbulnya perubahan state. Contoh state diagram dapat dilihat pada Gambar 2.4.



Gambar 2.4. Contoh State Diagram

Sumber: Schmuller, Joseph, 1999, Sams Teach Yourself UML in 24 Hours, Indiana, Sams Publishing, hal. 95

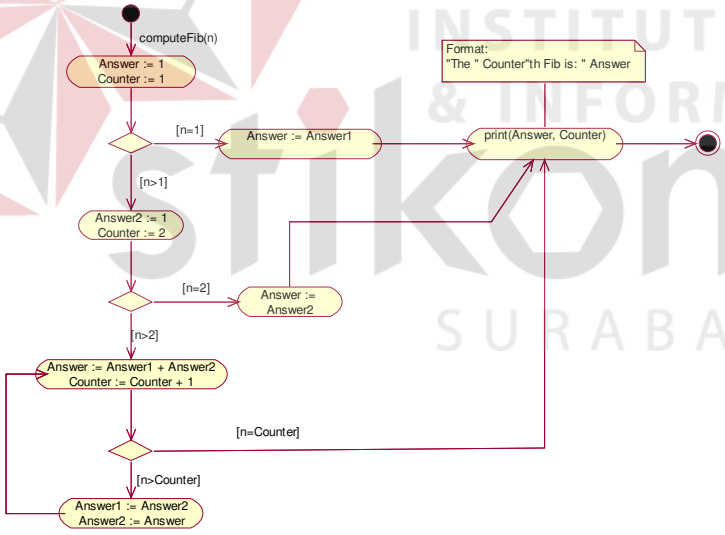
Dari gambar tersebut yang dimaksud dengan:

- State: `Initializing`, `Working`, `Shutting Down`
- Start state ditunjukkan dengan icon lingkaran hitam
- End state: ditunjukkan dengan icon lingkaran hitam di dalam lingkaran.

- Action: `do/ Bootup`

2.2.4 Activity Diagram

"Activity diagram didesain untuk menyederhanakan pandangan akan apa yang terjadi selama suatu operasi atau proses berlangsung" (Schmuller, 1999:134). Activity diagram merupakan perluasan dari state diagram, hanya saja bedanya kalau pada state diagram ditunjukkan perubahan state dari suatu object, sedangkan pada activity diagram yang ditekankan adalah aktivitas (*activity*)nya. Notasi yang digunakanpun sama hanya saja lebih banyak di activity diagram dan tidak mengenal icon *state*. Pada pemrograman konvensional diagram ini dikenal dengan flow chart. Contoh activity diagram dapat dilihat pada Gambar 2.5.



Gambar 2.5. Contoh Activity Diagram

Sumber: Schmuller, Joseph, 1999, Sams Teach Yourself UML in 24 Hours, Indiana, Sams Publishing, hal. 138

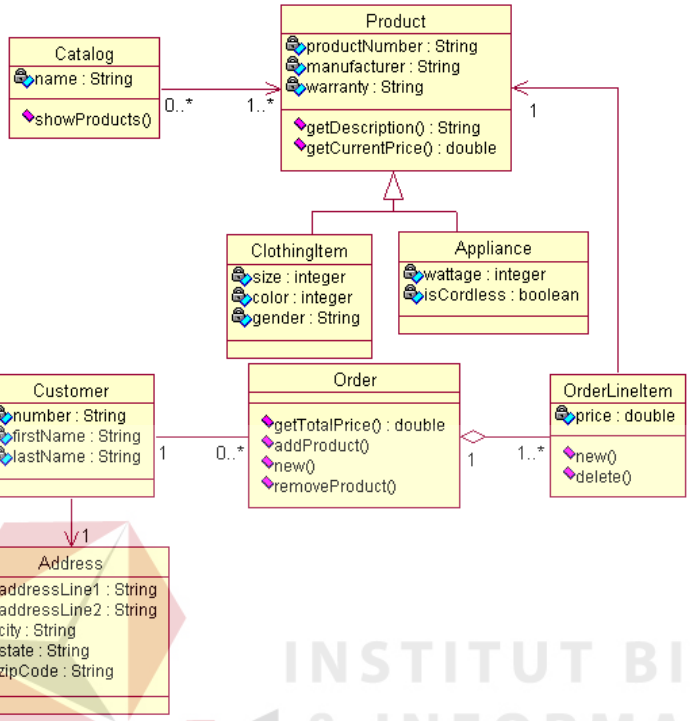
Dari Gambar 2.5. telihat contoh activity diagram dari method computeFibo:

- awal dan akhir activity ditandai dengan icon yang sama dengan pada state diagram, yaitu start state dan end state
- activity digambarkan dengan icon segiempat yang bersudut bulat setengah lingkaran dengan di dalamnya terdapat operasi yang dilakukan.
- icon condition digambarkan dengan belah ketupat kosong, sedangkan kondisinya ditulis pada masing-masing jalur percabangan.
- komentar (catatan) sama seperti diagram yang lain ditandai dengan icon note yang digambarkan dengan kertas yang terlipat salah satu ujungnya. Pemberian catatan ini hanya untuk penjelasan saja.

2.2.5 Class Diagram

"Class diagram menggambarkan kelas-kelas (classes) yang ditemukan dalam sebuah sistem dan hubungan (relationship) di antara kelas-kelas itu" (Cavaness, 2001:20). Sebuah kelas (class) adalah sekumpulan obyek yang memiliki struktur dan perilaku yang sama, dalam arti sama *attribute*, *operation* dan relationshipnya. Attribute adalah data (fields) yang dimiliki oleh sebuah class. Sedangkan operation artinya layanan (services) yang disediakan oleh suatu class, bisa berupa method untuk mengakses dan merubah nilai field/attribute dari class tersebut atau method yang mengimplementasikan karakteristik behavior dari interface. Operation dalam class di class diagram merupakan message yang masuk ke suatu obyek dalam sequence diagram.Object dalam sequence diagram merupakan instance dari class dalam class diagram. Sebuah class dapat mengimplementasi (merealisasi) sebuah interface atau lebih. Sebuah interface

menspesifikkan operation dari sebuah class yang belum diimplementasi. Gambar 2.6 menunjukkan contoh class diagram.



Gambar 2.6. Contoh Class Diagram

Sumber: Cavaness, Chuck and Friesen, Geoff, and Keeton, Brian 2001, Special Edition Using Java™ 2 Standar Edition, Indiana, Que Corporation, hal. 22

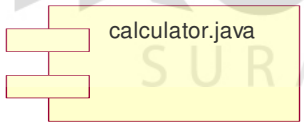
Dari Gambar 2.6 dapat dilihat bahwa:

- dalam class diagram tersebut terdapat 8 (delapan) class, yaitu: `Catalog`, `Product`, `ClothingItem`, `Appliance`, `Customer`, `Order`, `OrderLineItem` dan `Address`.
- class `Catalog` memiliki *private attribute* `name` yang bertype `String`, dan memiliki *public operation* `showProducts()`. Demikian juga class yang lain memiliki *attribute* dan *operation*nya masing-masing.

- *Class* `Product` merupakan generalisasi dari class `ClothingItem` dan `Appliance`.
- Antara class `Order` dan `OrderLineItem` terdapat relasi agregasi. Sedangkan relasi di antara class yang lain adalah asosiasi biasa.
- *Cardinality* relasi asosiasi antara class `Catalog` dan `Product` dapat diartikan object yang merupakan instance dari class `Product` boleh berasosiasi dengan banyak (*) atau tidak sama sekali (0) object yang merupakan instance dari class `Catalog`. Namun sebaliknya object yang merupakan instance dari class `Catalog` boleh berasosiasi dengan banyak (*) atau minimal 1 object yang merupakan instance dari class `Product`.

2.2.6 Component Diagram

"Component diagram berisi komponen, interface dan relationship"(Schmuller, 1999:152). Gambar 2.7 menggambarkan contoh component diagram.



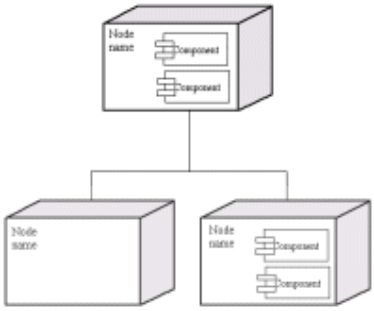
Gambar 2.7. Contoh Component Diagram

Sumber: Schmuller, Joseph, 1999, Sams Teach Yourself UML in 24 Hours, Indiana, Sams Publishing, hal. 152

Dari Gambar 2.7 terlihat component `calculator` merealisasikan class `calculator`.

2.2.7 Deployment Diagram

Deployment diagram menggambarkan arsitektur secara fisik sistem berbasis computer, dapat juga menggambarkan setiap komponen yang berada didalam komputer serta peralatan yang dibutuhkan di dalam sebuah sistem. Gambar 2.8 menggambarkan contoh deployment diagram.



Gambar 2.8 Contoh Deployment Diagram

2.3 Rational Rose

Rational Rose merupakan suatu tool pemodelan dari UML yang bersifat visual. Rational rose terdiri dari :

2.3.1 Use Case View

Use Case View digunakan untuk menggambarkan system yang ada. Use Case View menggambarkan bagaimana actor dan use case berinteraksi. Use Case View terdiri dari 4 diagram :

1. Use Case Diagram
2. Sequence Diagram
3. Collaboration Diagram
2. Activity Diagram

2.3.2 Logical View

Logical View digunakan untuk menggambarkan kebutuhan dari system serta bagaimana class-class yang ada berhubungan. View terdiri dari 2 diagram :

1. Class Diagram
2. Statechart Diagram

2.3.3 Component View

Component View menggambarkan informasi tentang software, komponen library dari system. View ini hanya terdiri dari component diagram.

2.3.4 Deployment View

Deployment View menggambarkan proses yang ada pada hardware serta kebutuhannya.

2.4 Java Server Pages (JSP)

Java Server Pages adalah suatu teknologi web berbasis bahasa pemrograman java dan berjalan di platform java, serta merupakan bagian teknologi J2EE(Java 2 Enterprise Edition) . JSP sangat sesuai dan tangguh untuk menangani presentasi di web. Sedangkan J2EE merupakan platform java untuk pengembangan system aplikasi enterprise dengan dukungan API (Application Programming Interface) yang lengkap dan portabilitas serta memberikan sarana untuk membuat suatu aplikasi yang memisahkan antara business logic (system), presentasi dan data.

JSP merupakan bagian dari J2EE dan khususnya merupakan komponen web dari aplikasi J2EE secara keseluruhan. JSP juga memerlukan JVM (Java

Virtual Machine) supaya dapat berjalan, yang berarti mengisyaratkan keharusan menginstall java virtual machine di server, dimana JSP akan dijalankan.

Kelebihan dari JSP adalah :

a. Memisahkan presentasi statis dan isi yang dinamik

   Untuk memudahkan dalam pembuatan maupun pemeliharaa situs desain presentasi harus dapat dipisahkan dengan skrip/kode pemograman untuk menghasilkan data/isi yang dinamik.

   Dengan teknologi JSP hal ini dapat dilakukan, di mana dengan JSP, maka web programmer dapat menyisipkan tag atau skriplet dengan data atau isi dinamik akan ditampilkan pada bagian-bagian dari halaman web yang telah didesain. Proses logic yang menampilkan data dinamik juga dapat terenkapsulasi menggunakan tag JSP maupun Java Bean.

b. Menekankan komponen reusable

   Teknologi JSP memerlukan komponen yang reusable dan cross platform (Java Bean atau Enterprise Java Bean) untuk melakukan pemrosesan yang lebih kompleks. Dengan komponen, developer dapat menggunakannya untuk operasi yang umum sehingga memungkinkan sharing dan distribusi komponen kepada public. Penggunaan komponen dapat mempercepat pembuatan aplikasi web karena proses logik yang diperlukan sudah tersedia dan langsung dapat digunakan.

c. Memudahkan pembuatan aplikasi dengan tag

   Teknologi JSP memungkinakan pembuatan dan pendefinisian tag-tag baru yang disebut custom tag sehingga memungkinkan adanya tag libraries, yaitu kumpulan tag yang memiliki berbagai fungsi yang mudah digunakan.

d. Berbasis Bahasa Pemrograman Java

Oleh karena JSP berbasis java, maka aplikasi yang dibuat dengna JSP juga memiliki manajemen memori dan sekuritas yang baik. Selain itu, JSP mudah dipelajari dan dapat memanfaatkan pemrograman berorientasi objek dari java.

e. Bagian dari platform Java

Oleh karena merupakan bagian dari platform java, maka JSP juga memiliki karakteristik "*Write Once, Run Anywhere*" yaitu protabilitas yang tinggi.

f. Terintegrasi dengan J2EE

Oleh karena JSP merupakan bagian integrasi J2EE, maka aplikasi JSP dapat dikembangkan ke aplikasi berskala enterprise.

2.5 JavaBeans

Salah satu kehandalan dari teknologi JSP adalah kemampuan dalam mengakses komponen javabeans. Penggunaan Java bean mampu meringkas dan mengefisiensikan pemrograman JSP, karena Java bean dapat mengenkapsulasi logic bisnis atau fungsi-fungsi tertentu.

2.5.1 Sifat dan Karakteristik Java Bean

Adapun sifat dan karakteristik Java bean adalah :

a. Memiliki properti dengan akses bukan public yang digunakan untuk menyimpan data.
b. Memiliki metode untuk set dan get.
c. Memiliki konstruktor kosong.

2.5.2 Menambahkan Java Bean dalam JSP

Menambahkan JavaBean ke dalam halaman JSP, dengan menggunakan tag <jsp:usebean>, yang memiliki sintak seperti berikut :

<jsp:usebean id=”namaBean” class=”belajarjsp.TesBean” scope=”page” />

Keterangan :

a. Atribut id berfungsi untuk memberi nama pada Java bean pada halaman JSP.
b. Atribut class merupakan program java yang sudah dikompilasi yang menyediakan logic untuk Java bean pada halaman JSP.
c. Atribut scope mendefinisikan scope Java bean dalam halaman JSP.