BAB III

LANDASAN TEORI

Dalam Melakukan perancangan dan pembuatan Modul web Majalah Percikan Iman. Ada beberapa dasar-dasar ilmu tentang teknologi IT yang akan mendukung yaitu UML, HTML, PHP dan MySQL.

3.1 UML (Unified Modellling Language)

Saat ini piranti lunak semakin luas dan besar lingkupnya, sehingga tidak bisa lagi dibuat asal-asalan. Piranti lunak saat ini seharusnya dirancang dengan memperhatikan hal-hal seperti *scalability*, *security*, dan eksekusi yang robust walaupun dalam kondisi yang sulit. Selain itu arsitekturnya harus didefinisikan dengan jelas, agar *bug* mudah ditemukan dan diperbaiki, bahkan oleh orang lain selain *programmer* aslinya. Keuntungan lain dari perencanaan arsitektur yang matang adalah dimungkinkannya penggunaan kembali modul atau komponen untuk aplikasi piranti lunak lain yang membutuhkan fungsionalitas yang sama.

Pemodelan (*modeling*) adalah proses merancang piranti lunak sebelum melakukan pengkodean (*coding*). Model piranti lunak dapat dianalogikan seperti pembuatan blueprint pada pembangunan gedung. Membuat model dari sebuah sistem yang kompleks sangatlah penting karena kita tidak dapat memahami sistem semacam itu secara menyeluruh. Semakin komplek sebuah sistem, semakin penting pula penggunaan teknik pemodelan yang baik.

Dengan menggunakan model, diharapkan pengembangan piranti lunak dapat memenuhi semua kebutuhan pengguna dengan lengkap dan tepat, termasuk faktor-faktor seperti *scalability, robustness, security*, dan sebagainya.

Kesuksesan suatu pemodelan piranti lunak ditentukan oleh tiga unsur, yang kemudian terkenal dengan sebuan segitiga sukses (*the triangle for success*). Ketiga unsur tersebut adalah metode pemodelan (*notation*), proses (*process*) dan *tool* yang digunakan.

Memahami notasi pemodelan tanpa mengetahui cara pemakaian yang sebenarnya (proses) akan membuat proyek gagal. Dan pemahaman terhadap metode pemodelan dan proses disempurnakan dengan penggunaan tool yang tepat.

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.

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*).

Untuk menguasai UML, sebenarnya cukup dua hal yang harus kita perhatikan:

1. Menguasai pembuatan diagram UML
2. Menguasai langkah-langkah dalam analisa dan pengembangan dengan UML

Seperti juga tercantum pada gambar diatas UML mendefinisikan diagram-diagram sebagai berikut:

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

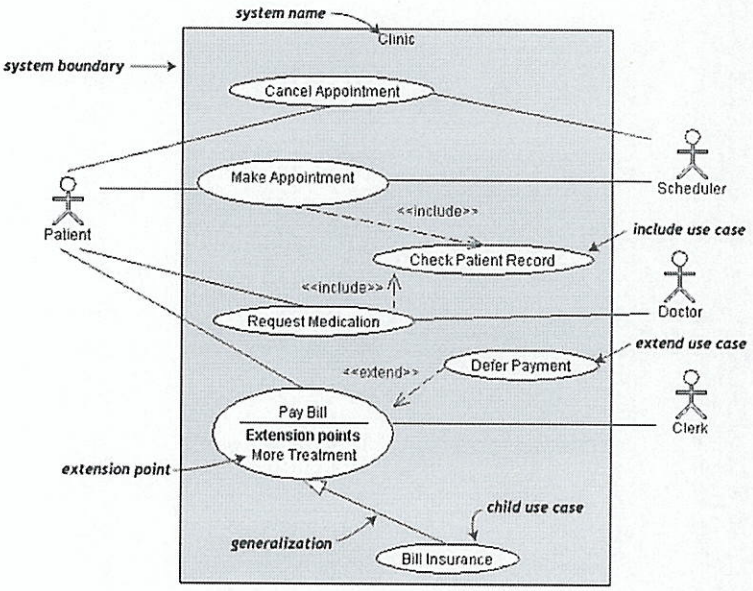
di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal.

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.

Contoh *use case diagram* :



Gambar 3.1 Contoh Use Case

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 :

- *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan
- *Protected*, dapat dipanggil oleh *class* yang bersangkutan dan yang mewarisinya
- *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.*

Dengan demikian *interface* mendukung resolusi metoda pada saat *run-time.* Sesuai dengan perkembangan *class* model, *class* dapat dikelompokkan menjadi *package.* Kita juga dapat membuat diagram yang terdiri atas *package.*

Hubungan Antar *Class*

1. *Asosiasi*, yaitu hubungan statis antar class. Panah navigability menunjukkan arah query antar class.

2. *Agregasi*, yaitu hubungan yang menyatakan bagian ("terdiri atas..").
3. Pewarisan, yaitu hubungan hirarkis antar class. Kebalikan dari pewarisan adalah generalisasi.
4. Hubungan dinamis, yaitu rangkaian pesan (message) yang di-passing dari satu class kepada class lain.

Statechart 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.

Activity Diagram

Activity diagrams 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*:

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 fase 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*.

Collaboration 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.

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.

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 ini.

Langkah-Langkah Penggunaan UML

Berikut ini adalah tips pengembangan piranti lunak dengan menggunakan UML:

1. Buatlah daftar *business process* dari level tertinggi untuk mendefinisikan aktivitas dan proses yang mungkin muncul.
2. Petakan use case untuk tiap business process untuk mendefinisikan dengan tepat fungsionalitas yang harus disediakan oleh sistem. Kemudian perhalus

use case diagram dan lengkapi dengan requirement, constraints dan catatan-catatan lain.

3. Buatlah deployment diagram secara kasar untuk mendefinisikan arsitektur fisik sistem.
4. Definisikan requirement lain (non-fungsional, security dan sebagainya) yang juga harus disediakan oleh sistem.
5. Berdasarkan use case diagram, mulailah membuat activity diagram.
6. Definisikan objek-objek level atas (package atau domain) dan buatlah sequence dan/atau collaboration diagram untuk tiap alir pekerjaan. Jika sebuah use case memiliki kemungkinan alir normal dan error, buatlah satu diagram untuk masing-masing alir.
7. Buarlah rancangan user interface model yang menyediakan antarmuka bagi pengguna untuk menjalankan skenario use case.
8. Berdasarkan model-model yang sudah ada, buatlah class diagram. Setiap package atau domain dipecah menjadi hirarki class lengkap dengan atribut dan metodanya. Akan lebih baik jika untuk setiap *class* dibuat *unit test* untuk menguji fungsionalitas *class* dan interaksi dengan *class* lain.
9. Setelah class diagram dibuat, kita dapat melihat kemungkinan pengelompokan class menjadi komponen-komponen. Karena itu buatlah component diagram pada tahap ini. Juga, definisikan tes integrasi untuk setiap komponen meyakinkan ia berinteraksi dengan baik.

10. Perhalus deployment diagram yang sudah dibuat. Detilkan kemampuan dan requirement piranti lunak, sistem operasi, jaringan, dan sebagainya. Petakan komponen ke dalam node.
11. Mulailah membangun sistem. Ada dua pendekatan yang dapat digunakan :
    - Pendekatan use case, dengan meng-assign setiap use case kepada tim pengembang tertentu untuk mengembangkan unit code yang lengkap dengan tes.
    - Pendekatan komponen, yaitu meng-assign setiap komponen kepada tim pengembang tertentu.
12. Lakukan uji modul dan uji integrasi serta perbaiki model berserta codenya. Model harus selalu sesuai dengan code yang aktual.
13. Piranti lunak siap dirilis.

3.2 HTML

Hypertext markup Language (HTML) adalah bahasa yang digunakan untuk menulis halaman Web. Dalam menyampaikan ide pada web browser diperlukan tag. Tag selalu ditulis diantara tanda lebih besar dan tanda lebih kecil. (`<tag>`).

HTML terdiri atas dua bagian yaitu header dan body. Strukturnya diapit oleh tag awal (`<HTML>`) dan tag akhir (`<HTML>`).

Standar penulisannya adalah :

```
<HTML>
      <HEAD>
            <TITLE>Deskripsi file</TITLE>
      </HEAD>
```

```
<BODY>
        Isi dokumen
</BODY>
</HTML>
```

3.3 PHP (Hypertext Preprocessor)

PHP (*Hypertext Preprocessor)* adalah sebuah bahasa scripting dieksekusi di sisi Server (Server based). Fungsinya adalah membuat sebuah web yang interaktif dan dinamis. PHP dibuat pertama kali pada tahun 1995 oleh Rasmus Lerdorf, seorang software engineering anggota tim pengembangan web server Apache. Pada tahun 1996 Rasmus menulis kode script Perl untuk diterapkan pada halaman yang lebih kompleks dengan bahasa pemrograman C. Pada bulan Oktober 2000 PHP merislis versi 4.0 yang merupakan pengembangan dari versi PHP 3 Script Engine.

Untuk memahami proses atau prosedur pemrograman sisi server PHP, maka kita terlebih dahulu harus mengetahui bagaimana sebuah HTML bisa diperlakukan oleh web server. Dalam sebuah halaman HTML yang terjadi adalah ketika sebuah request ke sebuah halaman Web dating dari browser, maka web server melakukan tiga langkah :

1. Membaca request dari browser.
2. Mencari halaman yang diminta di server.
3. Mengirim halaman yang diminta melalui internet atau internet ke browser.

Sedangkan yang terjadi di sebuah halaman PHP adalah sebgai berikut :

Ketika ada request ke sebuah halaman PHP maka server melakukan hal – hal berikut :

1. Membaca request dari browser.
2. Membaca halaman yang diminta di server.
3. Melakukan instruksi yang ada di halaman PHP yang diminta.
4. Mengirim balik halaman hasil eksekusi ke browser.

Kode PHP disimpan (saved) sebagai sebuah file plain text dalam format ASCII, jadi kita bisa menulis kode program PHP di hampir semua text editor, seperti vi, emacs, dan Notepad. Penggunaan editor HTML seperti Dreamweaver atau Homesite akan sangat membantu dalam menulis program PHP.

Seperti dijelskan sebelumnya, program PHP dapat dimasukkan atau ditempel di halaman HTML, yang dieksekusi di server sebelum dikirim ke browser. Contoh berikut ini akan menjelaskannya :

```
<HTML>
<?php
     echo ("Text generated by PHP");
?>
</HTML>
```

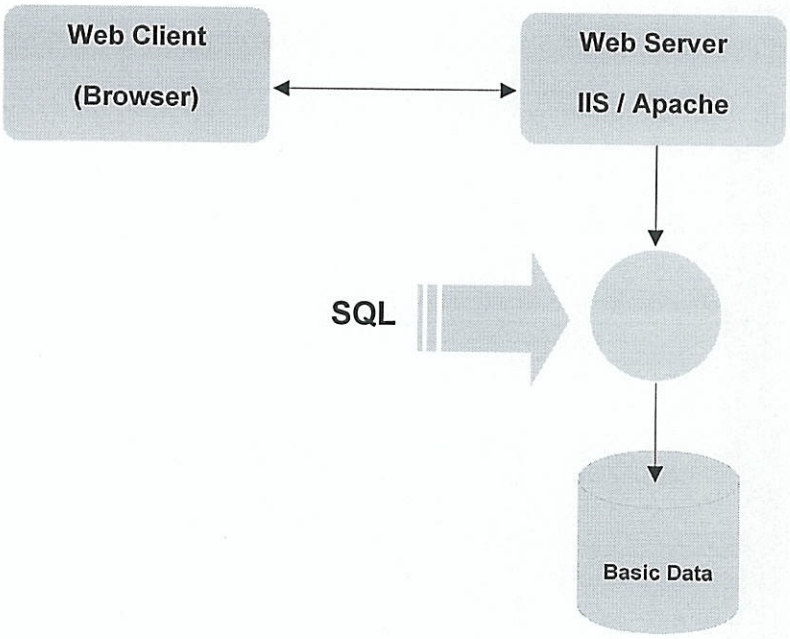
3.4 MySQL

Penyebab MySQL begitu populer dikalangan web adalah karena ia memang cocok bekerja di lingkungan tersebut.

1. MySQL tersedia di berbagai paltform Linux dan varian Unix. Sesuatu yang tidak dimiliki Acces, padahal Acces sangat populer di kalangan Window. Banyak webserver berbasiskan Unix/Linux sehingga otomatis Acces tidak dapat dipakai karena tidak memiliki kemampuan client-server/networking.

2. Fitur-fitur yang dimiliki memang banyak dibutuhkan dalam aplikasi web. Misal klausa `LIMIT` SQL-nya praktis untuk melakukan paging. Atau fungsi-fungsi built-in lainnya, mulai dari memformat dan memanipulasi tanggal, mengolah string, regex, enkripsi, dan hashing.
3. MySQL memiliki overhead koneksi yang rendah. Soal kecepatan melakukan transaksi atau kinerja di kondisi load tinggi mungkin bisa diperdebatkan dengan berbagai benchmark yang berbeda, dan MySQL adalah yang terdepan.

Namun ada beberapa hal yang kurang dan membedakan dengan Aplikasi SQL lain. Demi langsing dan cepat, fasilitas `subselect`, `view`, `stored procedure` atau `trigger` belum ada dalam versi terakhirnya.



Gambar 3.2 Skema Koneksi PHP dengan MySQL


UNIVERSITAS KOMPUTER INDONESIA
UNIKOM