Oleh Kundang K Juman

Unified Modeling Language (UML)

Konsepsi dasar UML (*Unified Modelling Language*)

UML (*Unified Modelling Language*) adalah suatu bahasa *visual* serba guna yang digunakan untuk menjelaskan, memvisualisasikan, membangun, dan mendokumentasikan suatu sistem. UML digunakan untuk memahami, merancang, mengkonfigurasi, me*maintance*, dan mengontrol informasi tentang suatu sistem.

UML terdiri dari beberapa elemen yang membentuk diagram dengan aturan tertentu. Diagram ini bertujuan untuk menggambarkan sistem dari berbagai sudut pandang. Relasi *Unified Modelling Language* menggambarkan hubungan antar *object* didalam suatu sistem. Beberapa relasi dalam UML yaitu :

1. *Dependency*, yaitu suatu relasi dimana suatu objek tergantung dengan pada objek yang lain. Perubahan pada objek tersebut berpengaruh terhadap objek lain. Relasi digambarkan sebagai garis terputus-putus yang memiliki arah menunjuk pada suatu objek.
2. *Generalization*, yaitu relasi antar objek yang umum (*parent*) dengan sesuatu yang lebih spesifik (*child*) dimana objek yang lebih khusus dapat digantikan dengan objek yang lebih umum. Generalisasi digambarkan sebagai garis dengan arah menunjuk ke objek yang lebih umum (*parent*).
3. *Association*, yaitu relasi hubungan *structural* yang menggambarkan sekumpulan *link*, dimana *link* adalah hubungan antar objek. *Association* digambarkan sebagai garis, kadang disertai dengan keterangan dan sering mengandung *multiciply*.

4. *Realization,* yaitu hubungan dimana suatu *classifiers* mengspesifikasikan sebuah kontrak sedangkan *classifiers* lain menyelesaikannya. Kita akan menemukan hubungan *realization* pada 2 buah tempat yaitu diantara *interface* dan *class*/komponen yang tercangkup di dalamnya. Juga diantaranya *use case* dan kolaborasi yang mengerjakannya. Secara grafik hubungan *realization* dinyatakan sebagai gabungan antara *generalization* dan *dependency*.

Kegunaan UML

Kegunaan dari UML, antara lain :

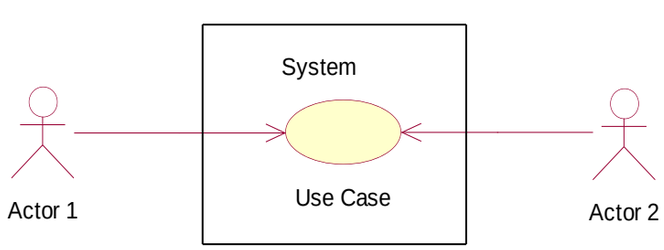
- Mempresentasikan elemen suatu sistem atau suatu domain dan *Relationship*-nya pada suatu *Static Structure* menggunakan *class* dan diagram *object.*
- Memodelkan *Behavior object* dengan *state transition diagrams.*
- Menampilkan Arsitektur Implementasi Fisik (*Physical Implementasi Architecture*) dengan Diagram Komponen dan Diagram Penyebaran (*Deployment*).
- Menampilkan batas suatu sistem dan fungsi utamanya menggunakan *use case* dan *actors.*
- Mengilustrasikan realisasi *Use Case* dengan *interaction* diagram di sisi lain.

Tipe Diagram Dalam *Unified Modelling Language*

1. *Use Case* Diagram

Menurut Munawar (2005, p63), menyatakan bahwa *use case* adalah deskripsi fungsi dari sebuah *system* dari perspektif pengguna. *Use case* bekerja dengan cara mendeskripsikan tipikal interaksi antara *user* (pengguna) sebuah sistem dengan sistemnya sendiri melalui sebuah cerita bagaimana sebuah sistem dipakai. Urutan langkah-langkah yang menerangkan antara pengguna dan system yang disebut *scenario*. Setiap *scenario* mendeskripsikan urutan kejadian. Setiap urutan diinisialisasi oleh orang, *system* yang lain, perangkat keras atau urutan waktu. *Use case* adalah serangkaian *scenario* yang digabungkan bersama-sama oleh tujuan umum pengguna. *Use case* biasanya menggunakan actor. *Actor* adalah sebuah peran yang bisa dimainkan oleh pengguna dalam interaksinya dengan sistem.

Umumnya *use case* relatif berupa proses yang besar atau *global*, bukan suatu transaksi atau langkah-langkah individual dan mendeskripsikan aktivitas di dunia nyata. Sebagai contoh lihat gambar berikut.



Dari pendapat dan gambar diatas ditarik kesimpulan bahwa *use case* diagram menjabarkan suatu sistem yang terlihat secara *eksternal*. Terdiri dari *actors* yakni pengguna yang bisa berinteraksi dengan sistem, *use case* dimana *actor* dapat berpartisipasi, dan menggambarkan hubungan diantara mereka.

Karena *use case* ini merupakan sebuah alat bantu, maka didalamnya pun terdapat komponen-komponen penyusun alat bantu itu. Pada *use case* diagram, sebuah sistem atau proses biasanya digambarkan dengan sebuah kotak persegi dengan nama sistem di dalamnya yaitu pada bagian atas kotak persegi tersebut. Kotak persegi ini disebut *system boundary* yang merupakan batasan dari sistem yang dideskripsikan. Komponen-komponen yang merupakan bagian dari sistem digambarkan di dalam kotak, sedangkan entitas-entitas *eksternal* yang bukan merupakan bagian dari sistem dan berinteraksi dengan sistem digambarkan di luar kotak persegi tersebut.

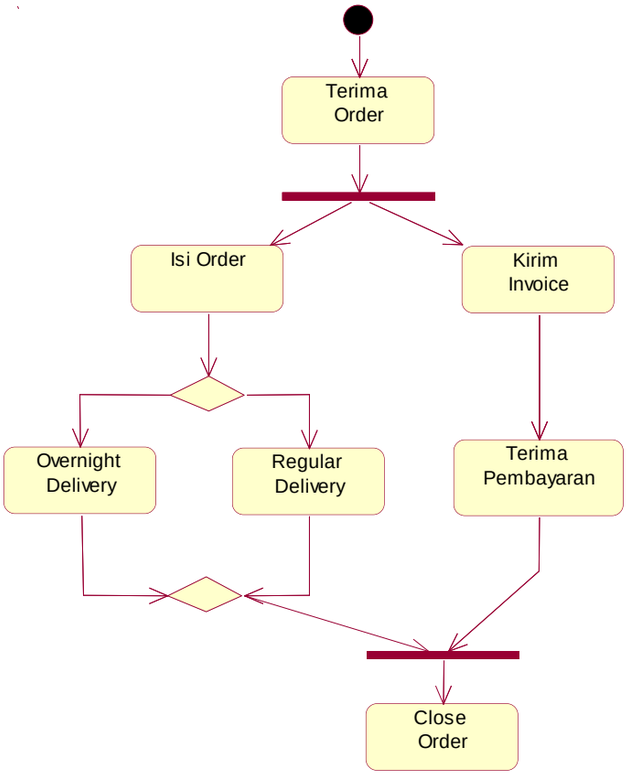
Actor adalah suatu entitas yang berinteraksi dengan sistem yang tujuannya untuk melengkapi suatu aksi atau peristiwa (Armour dan Miller, 2000, p6). *Actor* ini tidak harus seorang manusia, tetapi dapat juga berupa sistem lain, organisasi *eksternal*, peralatan *eksternal*, atau entitas *eksternal* lainnya yang ikut berinteraksi dengan sistem. Simbol yang digunakan untuk menggambarkan *actor* ini biasanya berbentuk orang dan di bawahnya ditulis nama peran yang dilakukan oleh *actor* tersebut.

Komponen penyusun lain yang digunakan adalah sebuah garis yang menghubungkan antara *use case* dengan *actor* yang berinteraksi dengannya. Garis ini disebut sebagai *association*.

2. *Activity* Diagram

Activity Diagram merupakan gambaran detail dari *use case* diagram dimana setiap *state* merupakan suatu aksi (*actin state*) dan transisinya dipicu oleh aksi (*action*) yang sudah selesai dari *state* sebelumnya dan biasanya digunakan untuk menunjukkan urutan dari *state-state* (Anonymous, 2003).

Selain sebagai gambaran detail sebuah *use case* diagram, *activity* diagram bisa juga untuk menjabarkan suatu *state* tertentu dari *statechart* diagram dimana fungsinya untuk menerangkan dan mendeskripsikan internal *behavior* suatu metode/*state* dan menunjukkan aliran *action* yang di kendalikan (*driven by*) oleh *action* sebelumnya.

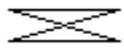
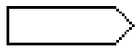
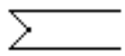


Menurut Munawar (2005, p109) *Activity diagram* adalah teknik untuk mendeskripsikan logika procedural, proses bisnis dan aliran kerja dalam banyak kasus. *Activity* Diagram mempunyai peran seperti halnya *flowchart*, akan tetapi perbedaannya dengan *flowchart* adalah *activity* diagram bisa mendukung perilaku paralel sedangkan *flowchart* tidak bisa. Simbol-simbol yang dipakai dalam pembuatan *activity* diagram :

Tabel 2.1

Simbol-simbol yang sering dipakai *activity* diagram

Symbol	Keterangan
	Titik awal
	Titik akhir
	*Activity*

	Pilihan untuk mengambil keputusan
	Fork : Digunakan untuk menunjukan kegiatan yang dilakukan secara paralel atau untuk menggabungkan dua kegiatan paralel menjadi satu.
	Rake : Menunjukan adanya dekomposisi
	Tanda waktu
	Tanda pengiriman dan ururtan aktifitas dalam suatu proses
	Tanda penerimaan
	Aliran Akhir (*Flow Final*)

3. *Class* Diagram

class adalah deskripsi kelompok obyek-obyek dengan *property*, perilaku (operasi) dan relasi yang sama. *Class* diagram bisa memberikan pandangan *global* atas sebuah sistem. Hal tersebut tercermin dari *class-class* yang ada dan relasinya satu dengan lainnya.

Class diagram menggambarkan struktur dan deskripsi *class*, *package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain.

Notasi-notasi yang terdapat dalam *class* diagram :

a. *Class*

Class menggambarkan sekumpulan *object* yang memiliki atribut dan operasi yang dikerjakan oleh *object* tersebut.

b. *Aggregation*

Aggregation menggambarkan hubungan antara dua atau lebih *object*, dimana salah satu *object* merupakan bagian dari *object* lainnya.

c. *Attribute*

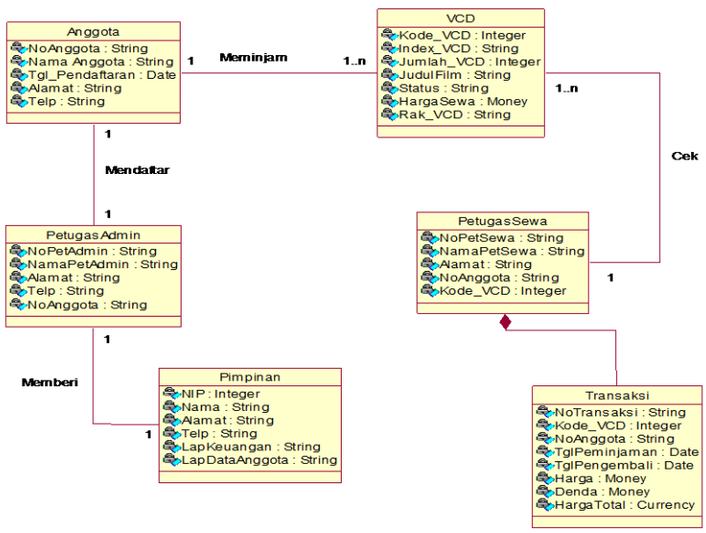
Attribute merupakan bagian dari *class* yang berisi tipe data yang dimiliki oleh *instance* dari suatu *class.*

d. *Operation*

Operation merupakan kegiatan-kegiatan yang akan dilakukan oleh suatu *class.*

e. *Association*

Association menggambarkan relasi antar *object/instance* dari *class.* Biasanya *association* digambarkan sebagai garis antara dua *class* di mana pada salah satu ujung diletakkan tanda panah yang menunjukan *navigability.*

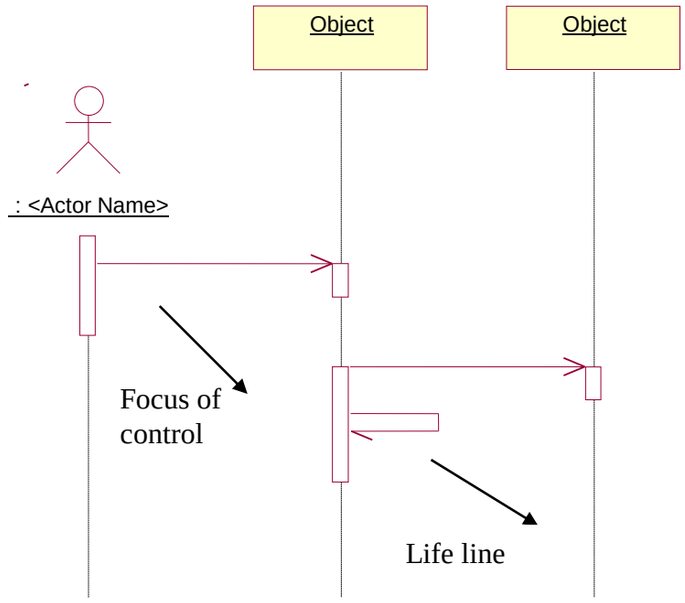


Gambar Contoh *Class* Diagram

4. *Sequence* Diagram

Sequence diagram menggambarkan pola hubungan antara sekumpulan *object* yang saling mempengaruhi menurut urutan waktu. Sebuah *object* berinteraksi dengan objek lain melalui pengiriman *message.* Diagram ini berguna untuk menggambarkan alur *event* dari *use case* dan mengidentifikasikan *object* yang terlibat dalam sebuah *use case*.

Bagian kolom dari diagram ini menggambarkan *object-object* yang terlibat dalam interaksi tersebut. Sedangkan bagian vertikalnya menggambarkan waktu interaksinya, *message* digambarkan dengan anak panah. Label anak panah ini menggambarkan nama *message. Message* yang dikirimkan juga bisa berupa kondisi ataupun berupa *message* yang sifatnya berulang. *Life line* menggambarkan lamanya *object* tersebut hidup. *Focus of control* mengambarkan lamanya waktu yang diperlukan oleh *object* untuk menyelesaikan tugasnya sesuai *message* yang diterima. *Sequence* diagram biasanya digunakan untuk mengilustrasikan sebuah *use case.*



Gambar 2.4 Contoh *Sequence* Diagram

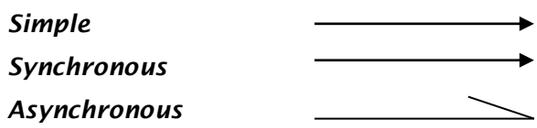
Menurut Munawar (2005, p87) *Sequence* diagram digunakan untuk menggambarkan prilaku pada sebuah *scenario.* Diagram ini menunjukkan sejumlah contoh obyek dan *message* (pesan) yang diletakkan diantara obyek-obyek ini di dalam *use case.*

- **Obyek/ *Participant***

Obyek diletakkan di dekat bagian atas diagram dengan urutan dari kiri ke kanan. Setiap *participant* terhubung dengan garis titik-titik yang disebut *lifeline.* Sepanjang *lifeline* ada kotak yang disebut *activation. Activation* mewakili sebuah eksekusi operasi dari *participant.*

- ***Message***

Sebuah *message* bergerak dari satu *participant* yang lain dan dari satu *lifeline* ke *lifeline* lain. Sebuah *participant* bisa mengirim sebuah *message* kepada diri sendiri.



Gambar 2.5 Simbol-simbol *message*

- ***Time***

Time adalah diagram yang mewakili waktu pada arah *vertical*. Waktu dimulai dari atas ke bawah, *message* yang lebih dekat dari atas akan dijalankan terlebih dahulu dibanding *message* yang lebih dekat ke bawah.

5. *State Chart* Diagram

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

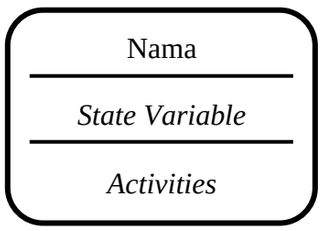
Dalam UML, *state* digambarkan berbentuk segiempat dengan sudut membulat dan memiliki nama sesuai kondisinya saat itu. Transisi antar *state* umumnya memiliki kondisi *guard* yang merupakan syarat terjadinya transisi yang bersangkutan, dituliskan dalam kurung siku. *Action* yang dilakukan sebagai akibat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

Titik awal dan akhir digambarkan berbentuk lingkaran berwarna penuh dan berwarna setengah.

Simbol UML untuk *state chart* diagram adalah segi empat yang tiap pojoknya dibuat *rounded*. Titik awalnya menggunakan lingkaran solit yang diarsir dan diakhiri dengan mata. Berikut adalah *symbol* UML untuk *state chart*.

Gambar 2.6 Simbol *State Chart* Diagram

UML juga memberikan pilihan untuk menambahkan detil kedalam *symbol* tersebut dengan membagi menjadi 3 area yaitu nama *state*, *state variable*, dan *activity*.



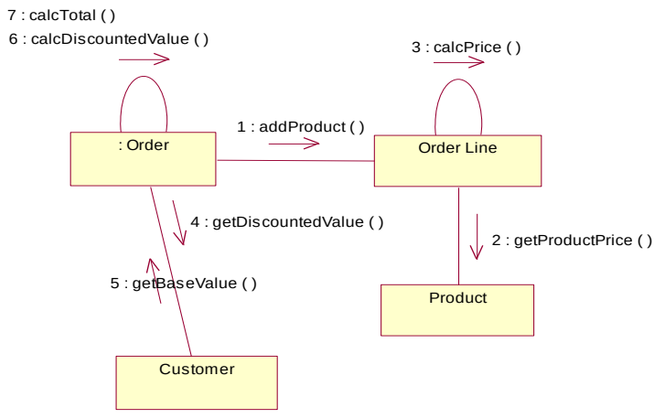
Gambar 2.7 Penambahan detil ke *state*

State variable seperti *timer* dan *counter* kadang kala sangat membantu. *Activity* terdiri atas *events* dan *action*. Tiga hal yang sering dipakai di sini adalah *entry* (apa yang terjadi ketika sistem masuk ke *state*), *exit* (apa yang terjadi ketika sistem meninggalkan *state*) dan *do* (apa yang terjadi ketika sistem ada di *state*).

6. *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. *Message* dari *level* yang sama memiliki *prefiks* yang sama.



Gambar *Collaboration* Diagram

2.11 Pemahaman Dasar *Object Oriented*

a. Pengertian obyek

Sebuah obyek memiliki keadaan sesaat (*state*) dan perilaku (*behaviour*). *State* sebuah obyek adalah kondisi obyek tersebut yang dinyatakan dalam *attribute/properties. State* sebuah obyek adalah kondisi obyek tersebut yang dinyatakan dalam *attribute*. Himpunan obyek-obyek yang sejenis disebut *class.* Obyek adalah contoh*/instance* dari sebuah *class.*

b. Abstraksi

Abstraksi bertujuan untuk mem*filter properties* dan *operation* pada sebuah obyek, sehingga hanya tinggal *properties* dan *operation* yang dibutuhkan saja.

c. Asosiasi

Asosiasi adalah hubungan antar obyek yang saling membutuhkan. Hubungan ini bisa satu arah ataupun lebih dari satu arah.

d. Agregasi

Adalah bentuk khusus dari asosiasi yang menggambarkan seluruh bagian suatu obyek merupakan bagian dari obyek yang lain.

Konsepsi Dasar UML

Menurut Nugroho (2010:10), Sesungguhnya tidak ada batasan yag tegas diantara berbagai konsep dan konstruksi dalam UML, tetapi untuk menyederhanakannya, kita membagi sejumlah besar konsep dan dalam UML menjadi beberapa *view*. Suatu *view* sendiri pada dasarnya merupakan sejumlah konstruksi pemodelan UML yang merepresentasikan suatu aspek tertentu dari sistem atau perangkat lunak yang sedang kita kembangkan. Pada peringkat paling atas, *view-view* sesungguhnya dapat dibagi menjadi tiga area utama, yaitu: klasifikasi struktural (*structural classification*), perilaku dinamis (*dinamic behaviour*), serta pengolahan atau manajemen model (*model management*).

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

Diagram-diagram dalam UML, diantaranya :

- *use case diagram*
- *class diagram*
- *statechart diagram*
- *activity diagram*
- *sequence diagram*
- *collaboration diagram*
- *component diagram*
- *deployment diagram*

Use Case Diagram

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem.

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

Use case diagram dapat digunakan untuk :

1. Menyusun *requirement* sebuah sistem,
2. Mengkomunikasikan rancangan dengan klien, dan
3. Merancang *test case* untuk semua *feature* yang ada pada sistem.

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 diantaranya :

1. Atribut/properti suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi).
2. 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 :

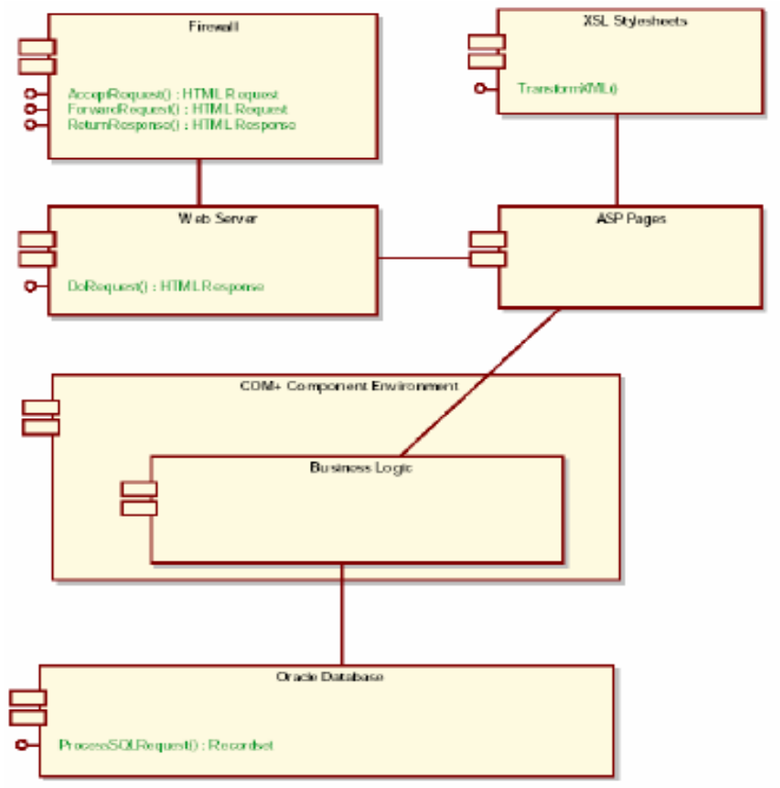
- *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan
- *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya

- *Public*, dapat dipanggil oleh siapa saja

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*:

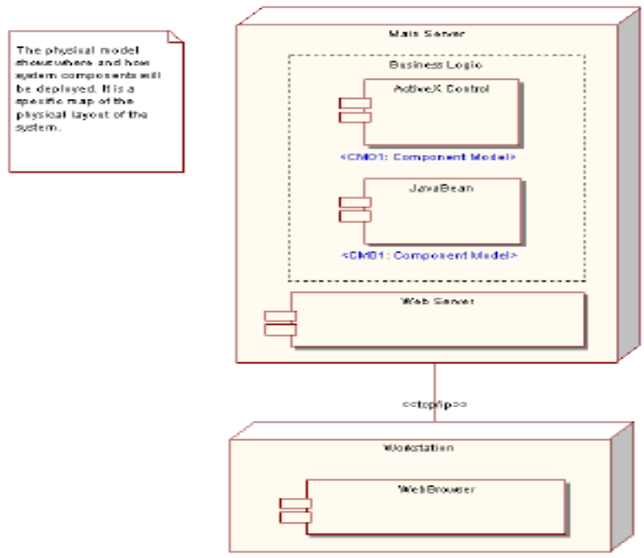


Deployment Diagram

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.

Contoh *deployment diagram* :



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. Buatlah 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 *code*nya. Model harus selalu sesuai dengan *code* yang aktual.
13. Piranti lunak siap dirilis.

Tool Yang Mendukung UML

Saat ini banyak sekali tool pendesainan yang mendukung UML, baik itu tool komersial maupun opensource. Beberapa diantaranya adalah:

• Rational Rose (www.rational.com)

• Together (www.togethersoft.com)

• Object Domain (www.objectdomain.com)

• Jvision (www.object-insight.com)

• Objecteering (www.objecteering.com)

• MagicDraw (www.nomagic.com/magicdrawuml)

• Visual Object Modeller (www.visualobject.com)

Referensi :

1. Pengantar Unified ModellingLanguage (UML), Sri Dharwiyanti (dharwiyanti@rnd.inti.co.id) Romi Satria Wahono (romi@romisatriawahono.net, http://romisatriawahono.net)
2. https://www.uml.edu/docs/Intellectual-Property-Policy_tcm18-88078.pdf
3. Munawar, UML 2018