**Sequence Diagram** (Berguna pada tahap awal analisa)

*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

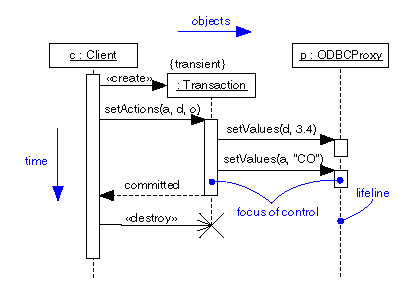
berikutnya, *message* akan dipetakan menjadi operasi/metoda dari *class*.

*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*.

Contoh *sequence diagram* :

****

**Collaboration Diagram** ( Digunakan lebih banyak pada tahap Desain)

*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.

Contoh *Collaboration Diagram*:



**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 :



**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**

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.

1. Lakukan uji modul dan uji integrasi serta perbaiki model berserta *code*nya. Model harus selalu sesuai dengan *code* yang aktual.
2. Piranti lunak siap dirilis.