DASAR TEORI

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

Apa itu UML

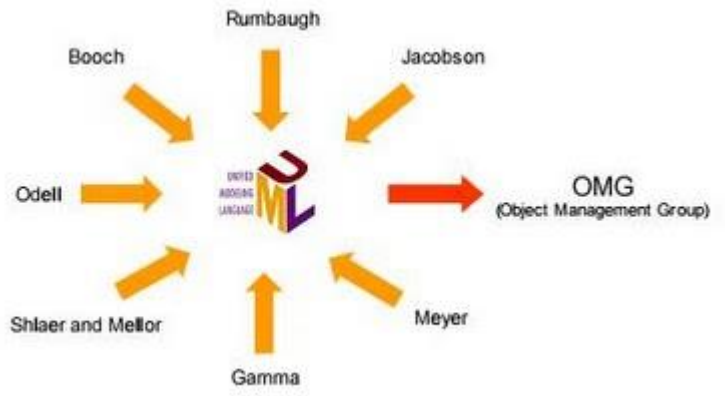
Unified Modelling Language (UML) adalah sebuah "bahasa" yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem.

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. Tetapi karena UML juga menggunakan class dan operation dalam konsep dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasa-bahasa berorientasi objek seperti C++, Java, C# atau VB.NET. Walaupun demikian, UML tetap dapat digunakan untuk modeling aplikasi prosedural dalam VB atau C.

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).

Sejarah UML sendiri cukup panjang. Sampai era tahun 1990 seperti kita ketahui puluhan metodologi pemodelan berorientasi objek telah bermunculan di dunia. Diantaranya adalah: metodologi booch [1], metodologi coad [2], metodologi OOSE [3], metodologi OMT [4], metodologi shlaer-mellor [5], metodologi wirfs-brock [6], dsb. Masa itu terkenal dengan masa perang metodologi (method war) dalam pendesainan berorientasi objek. Masing-masing metodologi membawa notasi sendiri-sendiri, yang mengakibatkan timbul masalah baru apabila

kita bekerjasama dengan group/perusahaan lain yang menggunakan metodologi yang berlainan.



Dimulai pada bulan Oktober 1994 Booch, Rumbaugh dan Jacobson,yang merupakan tiga tokoh yang boleh dikata metodologinya banyak digunakan mempelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek. Pada tahun 1995 direlease draft pertama dari UML (versi 0.8). Sejak tahun 1996 pengembangan tersebut dikoordinasikan oleh Object Management Group (OMG http://www.omg.org.Tahun 1997 UML versi 1.1 muncul, dan saat ini versi terbaru adalah versi 1.5 yang dirilis bulan Maret 2003. Booch, Rumbaugh dan Jacobson menyusun tiga buku serial tentang UML pada tahun 1999 [7] [8] [9]. Sejak saat itulah UML telah menjelma menjadi standar bahasa pemodelan untuk aplikasi berorientasi objek.

Konsepsi Dasar UML
Dari berbagai penjelasan rumit yang terdapat di dokumen dan buku-buku UML.

Abstraksi konsep dasar UML yang terdiri dari structural classification, dynamic behavior, dan model management, bisa kita pahami dengan mudah apabila kita melihat gambar diatas dari Diagrams. Main concepts bisa kita pandang sebagai term yang akan muncul pada saat kita membuat diagram. Dan view adalah kategori dari diagaram tersebut.

2.6 Notasi dalam UML

2.6.1 Actor

Gambar 2.9 Notasi Actor

Actor menggambarkan segala pengguna software aplikasi (user). Actor memberikan suatu gambaran jelas tentang apa yang harus dikerjakan software aplikasi. Sebagai contoh sebuah actor dapat memberikan input kedalam dan

menerima informasi dari software aplikasi, perlu dicatat bahwa sebuah actor berinteraksi dengan use case, tetapi tidak memiliki kontrol atas use case. Sebuah actor mungkin seorang manusia, satu device, hardware atau sistem informasi lainnya.

2.6.2 Use Case

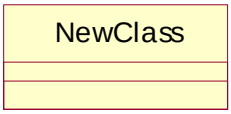
Gambar 2.10 Notasi Use Case

Use case menjelaskan urutan kegiatan yang dilakukan actor dan sistem untuk mencapai suatu tujuan tertentu. Walaupun menjelaskan kegiatan, namun use case hanya menjelaskan apa yang dilakukan oleh actor dan sistem bukan bagaimana actor dan sistem melakukan kegiatan tersebut.

- *Use-case Konkret* adalah use case yang dibuat langsung karena keperluan actor. Actor dapat melihat dan berinisiatif terhadapnya
- *Use-case Abstrak* adalah use case yang tidak pernah berdiri sendiri. Use case abstrak senantiasa termasuk didalam (include), diperluas dari (extend) atau memperumum (generalize) use case lainnya.

Untuk menggambarkannya dalam use case model biasanya digunakan association relationship yang memiliki stereotype include, extend atau generalization relationship. Hubungan *include* menggambarkan bahwa suatu use case seluruhnya meliputi fungsionalitas dari use case lainnya. Hubungan *extend* antar use case berarti bahwa satu use case merupakan tambahan fungsionalitas dari use case yang lain jika kondisi atau syarat tertentu terpenuhi.

2.6.3 Class



Gambar 2.11 Notasi Class

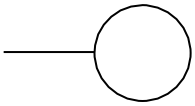
Class merupakan pembentuk utama dari sistem berorientasi obyek, karena class menunjukkan kumpulan obyek yang memiliki atribut dan operasi yang sama. Class digunakan untuk mengimplementasikan interface.

Class digunakan untuk mengabstraksikan elemen-elemen dari sistem yang sedang dibangun. Class bisa merepresentasikan baik perangkat lunak maupun perangkat keras, baik konsep maupun benda nyata.

Notasi class berbentuk persegi panjang berisi 3 bagian: persegi panjang paling atas untuk *nama class*, persegi panjang paling bawah untuk *operasi*, dan persegi panjang ditengah untuk *atribut.*

Atribut digunakan untuk menyimpan informasi. Nama atribut menggunakan kata benda yang bisa dengan jelas merepresentasikan informasi yang tersimpan didalamnya. Operasi menunjukkan sesuatu yang bisa dilakukan oleh obyek dan menggunakan kata kerja.

2.6.4 Interface



Gambar 2.12 Notasi Interface

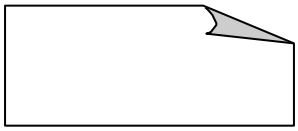
Interface merupakan kumpulan operasi tanpa implementasi dari suatu class. Implementasi operasi dalam interface dijabarkan oleh operasi didalam class. Oleh karena itu keberadaan interface selalu disertai oleh class yang mengimplementasikan operasinya. Interface ini merupakan salah satu cara mewujudkan *prinsip enkapsulasi* dalam obyek.

2.6.5 Interaction

Gambar 2.13 Notasi Interaction

Interaction digunakan untuk menunjukkan baik aliran pesan atau informasi antar obyek maupun hubungan antar obyek. Biasanya interaction ini dilengkapi juga dengan teks bernama operation signature yang tersusun dari nama operasi, parameter yang dikirim dan tipe parameter yang dikembalikan.

2.6.6 Note



Gambar 2.14 Notasi Note

Note digunakan untuk memberikan keterangan atau komentar tambahan dari suatu elemen sehingga bisa langsung terlampir dalam model. Note ini bisa disertakan ke semua elemen notasi yang lain.

2.6.7 Dependency

Gambar 2.15 Notasi Dependency

Dependency merupakan relasi yang menunjukan bahwa perubahan pada salah satu elemen memberi pengaruh pada elemen lain. Elemen yang ada di

bagian tanda panah adalah elemen yang tergantung pada elemen yang ada dibagian tanpa tanda panah.

Terdapat 2 stereotype dari dependency, yaitu include dan extend. *Include* menunjukkan bahwa suatu bagian dari elemen (yang ada digaris tanpa panah) memicu eksekusi bagian dari elemen lain (yang ada di garis dengan panah).

Extend menunjukkan bahwa suatu bagian dari elemen di garis tanpa panah bisa disisipkan kedalam elemen yang ada di garis dengan panah.

2.6.8 Association

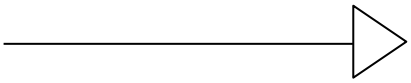
Gambar 2.16 Notasi Asociation

Association menggambarkan navigasi antar class (navigation), berapa banyak obyek lain yang bisa berhubungan dengan satu obyek (multiplicity antar class) dan apakah suatu class menjadi bagian dari class lainnya (aggregation).

Navigation dilambangkan dengan penambahan tanda panah di akhir garis. *Bidirectional navigation* menunjukkan bahwa dengan mengetahui salah satu class bisa didapatkan informasi dari class lainnya. Sementara *UniDirectional navigation* hanya dengan mengetahui class diujung garis association tanpa panah kita bisa mendapatkan informasi dari class di ujung dengan panah, tetapi tidak sebaliknya.

Aggregation mengacu pada hubungan “has-a”, yaitu bahwa suatu class memiliki class lain, misalnya Rumah memiliki class Kamar.

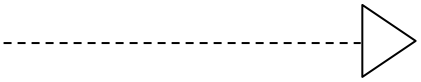
2.6.9 Generalization



Gambar 2.17 Notasi Generalization

Generalization menunjukkan hubungan antara elemen yang lebih umum ke elemen yang lebih spesifik. Dengan generalization, class yang lebih spesifik (subclass) akan menurunkan atribut dan operasi dari class yang lebih umum (superclass) atau *"subclass is superclass".* Dengan menggunakan notasi generalization ini, konsep inheritance dari prinsip hirarki dapat dimodelkan.

2.6.10 Realization



Gambar 2.18 Notasi Realization

Realization menunjukkan hubungan bahwa elemen yang ada di bagian tanpa panah akan merealisasikan apa yang dinyatakan oleh elemen yang ada di bagian dengan panah. Misalnya class merealisasikan package, component merealisasikan class atau interface.

2.7 Mengenal Rational Rose

2.7.1 Dasar-dasar Pemodelan dengan Rational Rose

2.7.1.1 Visual Modelling

Visual Modelling adalah proses menggambarkan cetak biru suatu sistem secara grafis, terdiri dari komponen-komponen, interfaces dan koneksi-koneksi yang ada dalam sistem tersebut, agar mudah dipahami dan dikomunikasikan.

Visual Modelling dapat membantu kita untuk menampilkan elemen-elemen yang penting secara detil dari suatu masalah yang kompleks dan menyaring untuk kemudian membuang elemen-elemen yang tidak penting.

Rational Rose menggunakan UML sebagai bahasa pemodelannya. Semua semantik dan notasi dalam UML dibuat untuk digunakan dalam visual modelling.

2.7.1.2 Model dalam Rekayasa Software

Sebagaimana telah diketahui dalam mendesain sebuah model dalam proses rekayasa software sangat penting sebagaimana pentingnya memiliki cetak biru untuk membangun suatu bangunan yang besar. Agar model yang dibuat memenuhi kriteria sebagai model yang bagus, maka model yang dibuat harus dapat :

- Mengidentifikassi persyaratan-persyaratan *(requirements)* dan dapat menyampaikan informasi dengan jelas.
- Berfokus pada bagaimana komponen-komponen sistem berinteraksi.
- Membantu kita untuk melihat hubungan antar komponen.

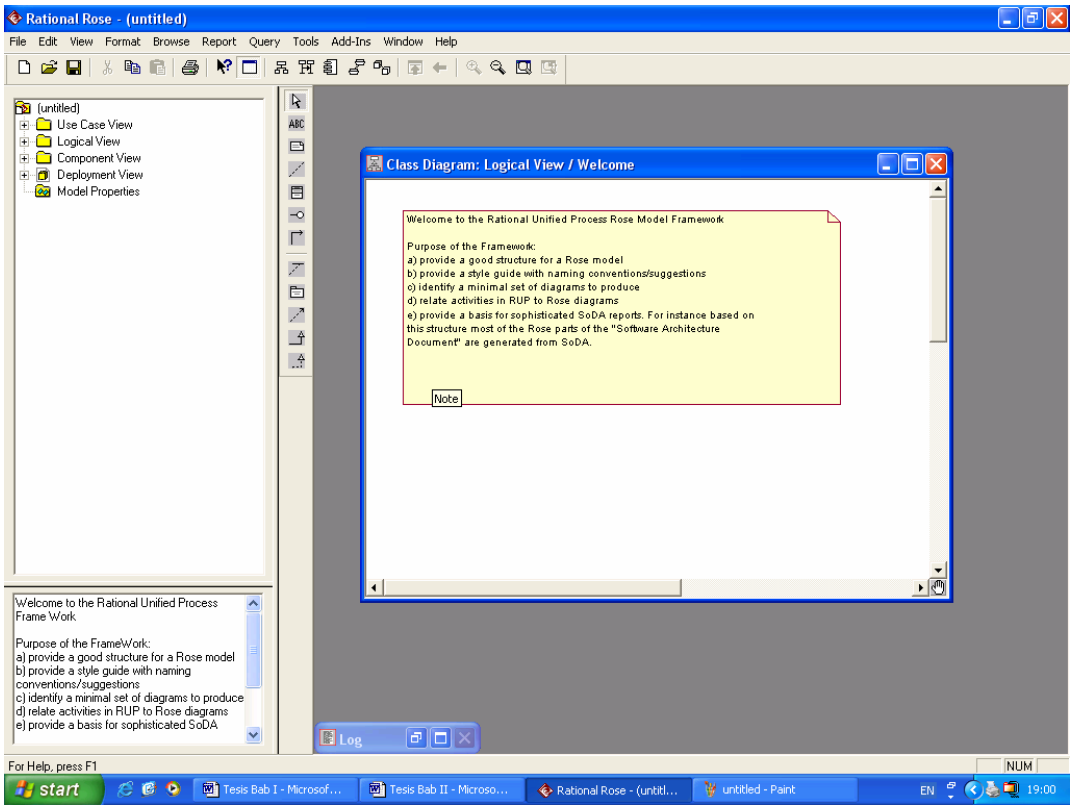
- Meningkatkan komunikasi antara anggota tim dengan menggunakan bahasa yang mudah dipahami, dalam hal ini bahasa grafis.

2.7.1.3 Edisi Rational Rose

Ada tiga edisi Rational Rose, yaitu :

- Rose Modeller, tidak mendukung bahasa pemrograman apapun.
- Rose Profesional, mendukung satu bahasa pemrograman.
- Rose Enterprise, mendukung banyak bahasa pemrograman, yaitu CORBA, VC ++, Visual Basic, Java dan sebagainya.

2.7.2 Graphical User Interface (GUI) dalam Rational Rose

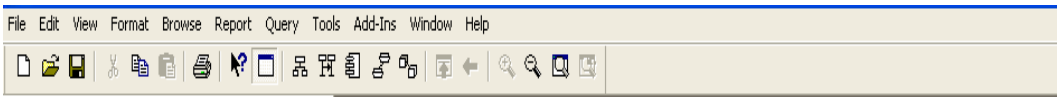


Gambar 2.19 Graphical User Interface dalam Rational Rose

Elemen-elemen dasar dalam Grapichal User Interface (GUI) dalam Rational Rose menurut A. Suhendar dan Hariman Gunadi (2002) adalah :

f Toolbar Standard

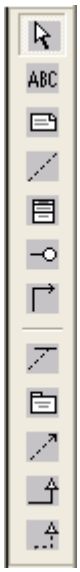
Toolbar standard pada Rational Rose terdapat pada bagian atas jendela utama dan senantiasa ditampilkan dan tidak bergantung pada tipe diagram yang aktif.



Gambar 2.20 Toolbar Standard

f Toolbox Diagram

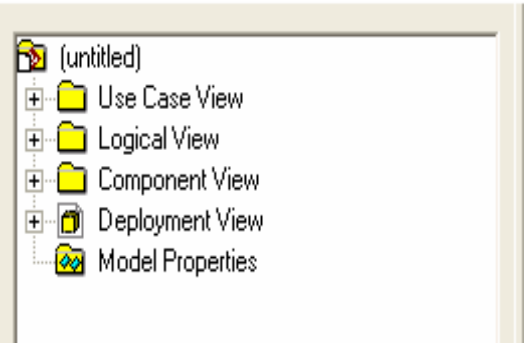
Toolbox diagram pada Rational Rose berubah sesuai dengan jenis diagram yang aktif. Diagram yang aktif ditampilkan dengan title bar warna biru.



Gambar 2.21 Toolbox Diagram

f Browser

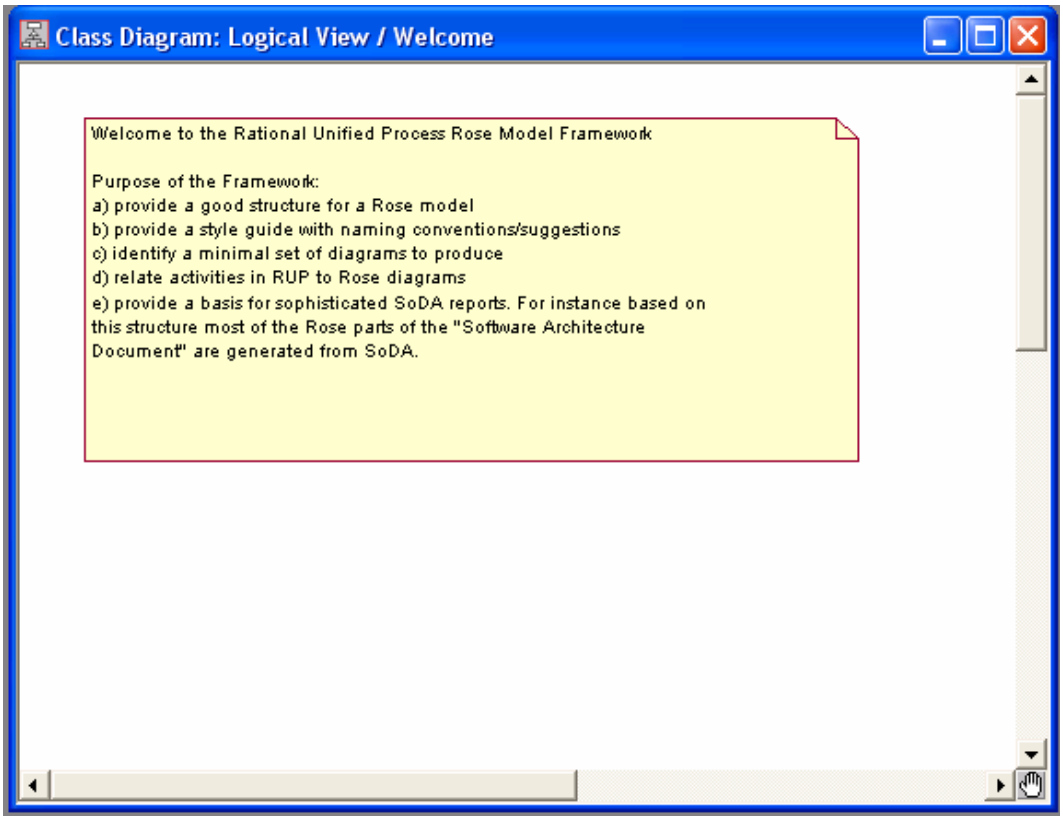
Browser pada Rational Rose membantu kita untuk melihat secara hirarkis elemen-elemen model. Tanda positif (+) menandakan bahwa elemen tersebut mengandung informasi tambahan didalamnya. Dengan menekan tanda (+), informasi tersebut diperluas. Sebaliknya tanda negatif (-) menandakan informasi terbuka secara penuh.



Gambar 2 22 Browser

ƒ Jendela Diagram

Jendela Diagram menampilkan atau memodifikasi diagram dalam jendela diagram. Jika terdapat beberapa diagram yang dibuka pada saat yang sama, dapat ditampilkan secara berlapis (cascade) atau berkotak-kotak (tiled).

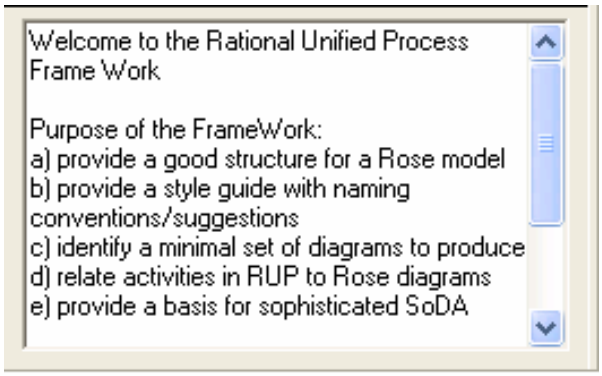


Gambar 2.23 Jendela Diagram

ƒ Jendela Dokumentasi

Jendela dokumentasi digunakan untuk membuat dokumentasi elemen-elemen model. Kita dapat membuat, melihat atau memodifikasi dokumentasi dalam jendela ini atau dalam jendela dokumentasi yang terdapat dalam spesifikasi. Jika jendela dokumentasi tidak muncul, pilih documentation dari menu view. Jika terdapat tanda ceklis pada documentation dan jendela ini belum muncul,

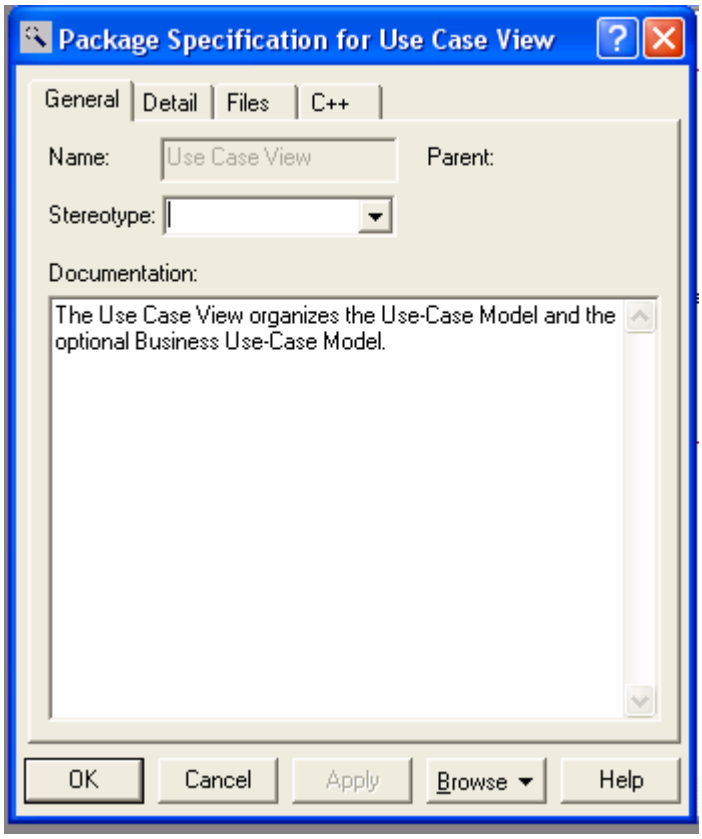
gerakkan kursor ke bagian bawah browser. Saat pointer berubah menjadi kursor pembagi (biasanya tanda panah pada kedua ujungnya) tarik keatas untuk memunculkan jendela dokumentasi. Isi dokumentasi pada jendela ini adalah dokumentasi dari elemen yang kita sorot pada jendela diagram.



Gambar 2.24 Jendela Dokumentasi

f Spesification

Spesification adalah kotak dialog yang digunakan untuk membuat atau mengubah properties elemen model.



Gambar 2.25 Spesification

2.7.3 View dalam Rational Rose

2.7.3.1 Use Case View

Use case view membantu kita untuk memahami dan menggunakan sistem yang kita modelkan. View ini melihat pada bagaimana actor[6] dan use case[7] berinteraksi.

Terdapat beberapa diagram yang digunakan dalam use case view, yaitu :

- Use Case Diagram.
- Sequence Diagram.
- Collaboration Diagram.
- Activity Diagram.

2.7.3.2 Logical View

Logical view, mengarah pada persyaratan *(requirements)* fungsional sistem. View ini melihat pada kelas-kelas dan hubungan antar kelas-kelas tersebut. Diagram dalam view ini adalah :

- Class Diagram.
- Sequence Diagram.
- Collaboration Diagram.
- Statechart Diagram.

---

[6] Actor menggambarkan pengguna (user) sistem. Actor membantu membatasi sistem dan memberi gambaran yang jelas mengenai apa yang harus dilakukan oleh sistem. Penting untuk dicatat bahwa actor berinteraksi dengan use-case, tetapi tidak mengendalikan use-case.

[7] Sebuah use-case dapat digambarkan sebagai suatu cara tertentu untuk menggunakan sistem dari sudut pandang satu pengguna (an actor)

2.7.3.3 Component View

Mengarah pada pengaturan software. View ini mengandung informasi mengenai komponen-komponen software, komponen tereksekusi (executable) dan library untuk sistem yang kita modelkan. Hanya ada satu diagram dalam view ini, yaitu component diagram.

2.7.3.4 Deployment View

Memperlihatkan pemetaan setiap proses kedalam hardware. View ini paling bermanfaat ketika kita membuat model suatu sistem yang diterapkan dalam lingkungan arsitektur yang terdistribusi dimana kita menerapkan aplikasi dan server pada lokasi yang berbeda. Hanya ada satu diagram dalam view ini, yaitu deployment diagram.

2.7.4 Diagram dalam Rational Rose

2.7.4.1 Use Case Diagram

Menjelaskan manfaat sistem jika dilihat menurut pandangan orang yang berada diluar sistem (actor). Diagram ini menunjukkan fungsionalitas suatu sistem atau kelas dan bagaimana sistem berinteraksi dengan dunia luar. Use Case diagram dapat digunakan selama *proses analisis untuk menangkap requirements sistem* dan untuk memahami bagaimana sistem seharusnya bekerja.

2.7.4.2 Class Diagram

Memperlihatkan hubungan antar kelas dan penjelasan detil tiap-tiap kelas didalam model desain (dalam logical view) dari suatu sistem. Selama proses analisis, class diagram memperlihatkan aturan-aturan dan tanggung jawab entitas yang menentukan perilaku sistem. Selama tahap desain, class diagram berperan dalam menangkap struktur dari semua kelas yang membentuk arsitektur sistem yang dibuat. Merupakan fondasi untuk component diagram dan deployment diagram.

2.7.4.3 Statechart Diagram

Memperlihatkan urutan keadaan sesaat yang dilalui sebuah obyek, kejadian yang menyebabkan sebuah transisi dari satu state atau aktivitas kepada yang lainnya, dan aksi yang menyebabkan perubahan satu state atau aktivitas

2.7.4.4 Activity Diagram

Memodelkan alur kerja (workflow) sebuah proses bisnis dan urutan aktivitas dalam suatu proses. Diagram ini sangat mirip dengan sebuah flowchart karena kita dapat memodelkan sebuah alur kerja dari satu aktivitas ke aktivitass lainnya atau dari satu aktivitas ke keadaan sesaat (state). Juga sangat berguna ketika ingin menggambarkan perilaku paralel atau menjelaskan bagaimana perilaku dalam berbagai use case berinteraksi.

2.7.4.5 Sequence Diagram

Menjelaskan interaksi obyek yang disusun dalam suatu urutan tertentu. Sequence diagram memperlihatkan tahap demi tahap apa yang seharusnya terjadi untuk menghasilkan sesuatu didalam use case.

2.7.4.6 Collaboration Diagram

Melihat pada interaksi dan hubungan terstruktur antar obyek. Tipe diagram ini menekankan pada hubungan *(relationship)* antar obyek, sedangkan sequence diagram menekankan pada urutan kejadian. Collaboration diagram digunakan sebagai alat untuk menggambarkan interaksi yang mengungkapkan keputusan mengenai perilaku sistem.

2.7.4.7 Component Diagram

Menggambarkan alokasi semua kelas dan obyek kedalam komponen-komponen dalam desain fisik sistem software. Diagram ini mmperlihatkan pengaturan dan kebergantungan antara komponene-komponen software seperti source code, binary code dan komponen tereksekusi.

2.7.4.8 Deployment Diagram

Diagram ini memperlihatkan pemetaan software kepada hardware.Diagram ini mengambarkan detail bagaimana komponen di-*deploy* dalam infrastruktur sistem, dimana komponen akan terletak, bagaimana kemampuan jaringan pada

lokasi tersebut dan hal lain yang bersifat fisik (Sri Dharwiyanti dan Romi satria Wahono, 2003)

2.7.5 Use-Case Modelling

Use-case Modelling adalah teknik paling sederhana dan paling efektif untuk memodelkan kebutuhan sistem berdasarkan pandangan user (Bennet, Simon, Steve Mc Robb dan Ray Farmer, 2000). Use-case modelling digunakan untuk bagaimana sistem atau kerja nyata dari suatu sistem atau bagaimana user ingin sistem itu bekerja. Use-case pada dasarnya adalah langkah awal dari analisis berdasarkan obyek dengan UML.

Use-case model terdiri dari actor dan use-case. Actor merepresentasikan user dan sistem lain yang berinteraksi dengan sistem. Use-case model sesungguhnya merepresentasikan tipe dari user, bukan suatu hal dari user. Use-case merepresentasikan karakteristik sistem, skenario dari tujuan sistem kedalam reaksi untuk menggerakkan actor.

Use-case adalah skenario untuk memahami kebutuhan user. Use-case model dapat menjadi kerangka dalam proyek pengembangan, perencanaan dan dokumentasi dari kebutuhan sistem. Use-case adalah interaksi antara user dan sistem, menggambarkan tujuan dari sistem dan tanggapan sistem untuk user. Use-case model mencoba untuk mensistematiskan identifikasi dari kegunaan sistem dan tanggapan dari sistem. Use-case model juga dapat mencakup kelas-kelas dan hubungan yang dimiliki oleh sub-sistem dari sistem.

Setiap use-case atau skenario merepresentasikan apa yang user ingin lakukan. Setiap use-case harus mempunyai nama dan deskripsi teks pendek, hanya sedikit paragraph.

2.7.5.1 Identifikasi Actor

Mengidentifikasikan actor adalah sama pentingnya dengan mengidentifikasikan kelas, struktur, hubungan, atribut dan karakteristik. Ketika menentukan actor, sangat penting untuk berfikir mengenai aturan rata-rata dari orang atau jenis pekerjaan. User mungkin memainkan lebih dari satu aturan. Actor harus merepresentasikan user tunggal.

Harus mengidentifikasi actor dan mengerti bagaimana mereka akan berguna dan berinteraksi dengan sistem. Kandidat untuk actor dapat ditemukan dengan menjawab pertanyaan-pertanyaan berikut :

- *Siapa yang menggunakan sistem ?*

  Kelompok mana yang membutuhkan pertolongan dari sistem untuk melakukan suatu pekerjaan.

- *Kelompok pengguna mana yang membutuhkan penampilan fungsi sistem ?*

  Fungsi ini dapat merupakan fungsi utama dan fungsi sekunder, seperti administrasi.

- *Apa masalah dari penyelesaian aplikasi (untuk siapa) ?*
- Dan terakhir, *bagaimana user menggunakan sistem (use-case) ?*

  Apa yang mereka lakukan dengan sistem ?

2.7.5.2 Menemukan Use-Case

Menurut Bennet, Simon, Steve Mc Robb dan Ray Farmer (2000), langkah-langkah untuk menemukan use-case adalah sebagai berikut:

- Untuk setiap actor, temukan tugas dan fungsi actor harus dapat melakukan apa yang sistem butuhkan dari actor untuk melakukannya. Use-case harus merepresentasikan kegiatan dari kejadian untuk menghasilkan tujuan.
- Beri nama untuk setiap use-case.
- Deskripsikan masing-masing use-case.

Berapa banyak use-case yang dibutuhkan ? Untuk sepuluh orang per tahun proyek, actor mungkin mendapatkan dua puluh use-case (tidak terhitung hubungan uses dan extend). Dalam penelitian lain, mungkin mencapai seratus use-case untuk sebuah proyek. Tidak ada formula tertentu, hanya dibutuhkan kedinamisan dan kerja yang menemukan kenyamanan.

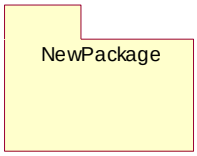
2.7.5.3 Penamaan Use-Case

Penamaan use-case harus menguntungkan deskripsi global dari fungsi use-case. Nama harus mendeskripsikan apa yang terjadi ketika bagian dari kinerja use-case bekerja. Penamaan dalam kata kerja atau kata benda, harus dilakukan dengan hati-hati karena deskripsi dari use-case harus merupakan gambaran dan sifatnya tetap.

Use-case adalah alat utama dalam menggambarkan kebutuhan. Menggambarkan use-case adalah salah satu langkah pertama yang penting untuk

melakukan sesuai dengan kebutuhan. Setiap use-case adalah kebutuhan yang berpotensi. Setiap use-case atau skenario merepresentasikan apa yang user ingin lakukan.

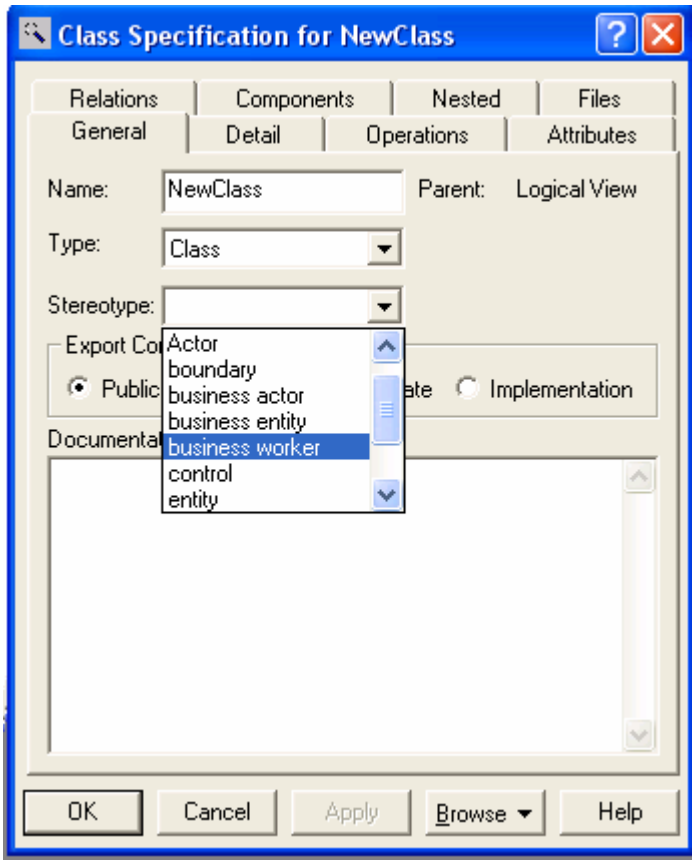
2.7.6 Package



Gambar 2.26 Notasi Package (Paket)

Paket adalah mekanisme pengelompokkan yang dgunakan untuk menandakan pengelompokkan elemen-elemen model. Sebuah paket dapat mengandung beberapa paket lain didalamnya. Paket digunakan untuk memudahkan mengorganisasikan elemen-elemen model.

2.7.7 Stereotype

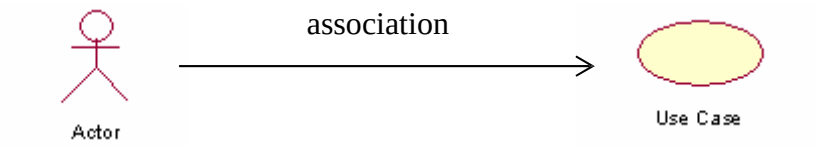


Gambar 2.27 Ruang Stereotype dalam Jendela Spesification

Sebuah stereotype menerangkan subklasifikasi dari sebuah elemen model.

2.7.8 Relationship

Adalah koneksi antar model elemen.



Gambar 2.28 Association Relationship

Association Relationship, memodelkan koneksi antar obyek dari kelas yang berbeda.

Interaksi antara actor dan use-case dalam use-case model biasanya digunakan association relationship yaitu :

- *<<uses>>*

  Hubungan uses menunjukkan bahwa prosedur dari use-case merupakan bagian dari prosedur yang menggunakan use-case. Tanda panah menunjukkan keadaan tidak mengakibatkan pemanggilan prosedur dalam menggunakan use-case. Relasi uses antara use-case ditunjukkan dengan panah generalisasi dari use-case. Use-case yang dilakukan secara berulang, digunakan untuk meminimalkan pekerjaan.

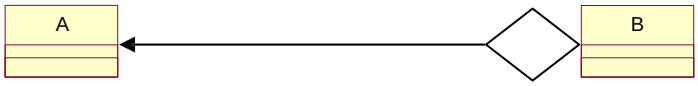
- *<<extend>>*

  Hubungan extend antar use-case berarti bahwa suatu use-case merupakan tambahan kegunaan dari use-case yang lain jika kondisi atau syarat tertentu dipenuhi. Jika prosedur dari use-case merupakan alternatif untuk menjelaskan use-case lain. Use-case akan dikerjakan apabila salah satu syarat terpenuhi.

Hubungan generalisasi antar use-case menunjukkan bahwa use-case yang satu merupakan spesialisai dari yang lain.

- *<<include>>*

  Hubungan include menggambarkan suatu use-case seluruhnya meliputi kegunaan dari use-case lainnya. 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 dieksekusi secara normal. Sebuah use-case dapat di-include oleh lebih dari use-case lain, sehingga duplikasi fungsional dapat dihindari.



Gambar 2.29 Aggregation Relationship

Aggregation Relationship, adalah bentuk khusus assosiasi yang memodelkan hubungan keanggotaan antara 2 kelas, yakni satu kelas disusun oleh kelas lainnya. Gambar diatas memperlihatkan bahwa B (kelas aggregate) yang secara fisik dibentuk oleh A atau secara logis B mengandung A.

2.8 Model Analisis

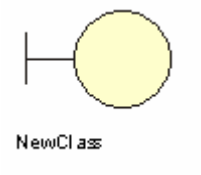
Model analisis menggambarkan realisasi dari use case - use case dalam use case model, dan bertindak sebagai abstraksi dari model desain. Tujuan akhir dari sebuah model analisis adalah untuk membuat pemetaan awal mengenai perilaku yang diisyaratkan dalam sistem aplikasi kedalam elemen-elemen pemodelan.

Model analisis merupakan transisi kedalam model desain, dan kelas-kelas analisis secara langsung berkembang menjadi elemen-elemen dalam model desain.

2.8.1 Kelas dalam Model Analisis

Elemen model yang terdapat dalam model analisis disebut kelas analisis. Kelas analisis adalah kelas berstereotype *boundary, control atau entity* yang menggambarkan sebuah konsep awal mengenai benda dalam sistem aplikasi yang memiliki tanggung jawab dan perilaku. Kelas analisis akhirnya berkembang menjadi kelas didalam model desain.

2.8.1.1 Boundary Class



Gambar 2.30 Boundary Class

Kelas boundary adalah kelas yang memodelkan interaksi antara satu atau lebih actor dengan sistem. Kelas boundary memodelkan bagian dari sistem yang bergantung pihak lain disekitarnya dan merupakan pembatas sistem dengan dunia luar. Lebih lanjut lagi user interface class sering disamakan dengan form yang digunakan sebagai interface antara sistem dengan user.

Kelas boundary dapat berupa :

- *User interface,* yang merupakan sarana komunikasi antara sistem dengan user, misalnya jendela (window) dalam GUI.
- *Sistem interface,* yang merupakan sarana komunikasi antara sistem dengan sistem informasi lainnya misalnya communication protocol.
- *Device interface,* yang merupakan sarana komunikasi antara sistem dengan device *(alat),* seperti printer, sensor dan sebagainya.

2.8.1.2 Control Class

Gambar 2.31 Control Class

Kelas control adalah kelas yang mengkoordinasikan aktivitas dalam sistem. Kelas ini menghubungkan kelas boundary dengan kelas entity. Kelas *control* digunakan untuk memodelkan *"perilaku mengatur"*, khusus untuk satu atau beberapa *use-case* saja. Kelas *control* tidak dipengaruhi perubahan disekelilingnya. Kelas ini menggunakan atau membuat isi dari kelas *entity* dan biasanya memasangkan kelas *boundary* dengan kelas *entity*.

2.8.1.3 Entity Class

Gambar 2.32 Entity Class

Kelas entity adalah kelas yang berhubungan data dan informasi yang digunakan oleh sistem. Kelas entity ini adalah kelas yang menyimpan dan mengolah data. Kelas entity memodelkan informasi yang harus disimpan oleh sistem. Kelas *entity* memperlihatkan data dari sebuah sistem. Oleh karena itu, kelas *entity* membantu untuk memahami apa yang kira-kira ditawarkan oleh sistem kepada user.

Entity object (instance dari kelas entity) biasanya bersifat pasif dan tetap (tidak berubah-ubah). Tanggung jawab utama objek ini adalah untuk menyimpan dan mengatur informasi dalam sistem.

2.8.1.4 Aturan yang Well-formed dalam Model Analisis

Berikut ini tabel mengenai *association relationship* beserta *stereotype*-nya yang diperbolehkan dalam model analisis.[8]

TO FROM	***ACTOR***	***BOUNDARY***	***ENTITY***	***CONTROL***
Actor		communicate		
Boundary	communicate	communicate	communicate subscribe	communicate
Entity			communicate subscribe	
Control		Communicate	communicate subscribe	communicate

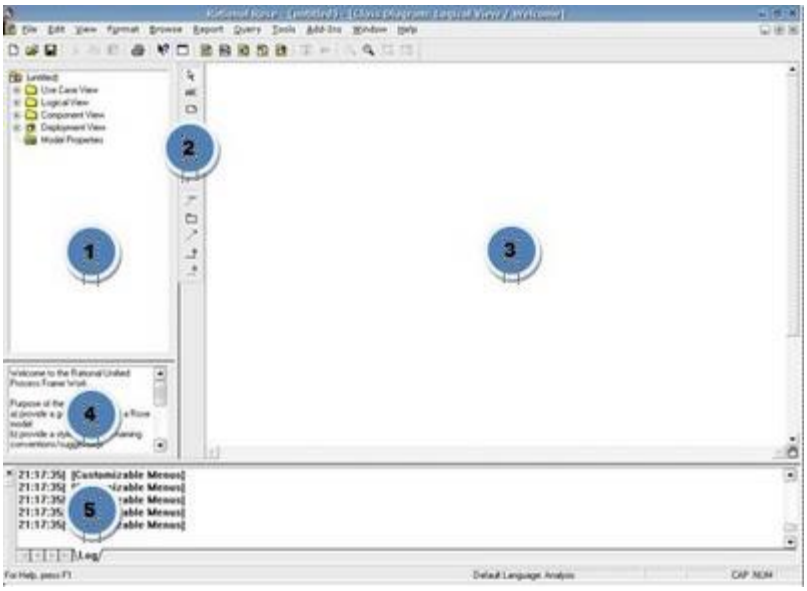
Tabel 2.2 Aturan dalam Model Analisis

Mengenal Rational Rose

Rational Rose adalah tools pemodelan visual untuk pengembangan system berbasis objek yang handal untuk digunakan sebagai bantuan bagi para pengembang dalam melakukan analisis dan perancangan system. *Rational rose* mendukung permodelan bisnis yang membantu para pengembang memahami system secara komprehensif. Ia juga membantu analisis system dengan cara pengembang membuat diagram use case untuk melihat fungsionalitas system secara keseluruhan sesuai dengan harapan dan keinginan pengguna. Kemudian, ia juga menuntut pengembang untuk mengambangkan Interaction Diagram untuk melihat bagaimana objek-objek saling bekerjasama dalam menyediakan fungsionalitas yang diperlukan.

Dalam *Rational rose*, pemodelan adalah cara melihat system dari berbagai sudut pandang. Ia mencakup semua diagram yang dikenal dalam UML, actor-aktor yang terlibat dalam system, use-case, objek-objek, kelas-kelas, komponen-komponen, serta simpul-simpul penyebaran. Model juga mendeskripsikan rincian yang diperlukan system dan bagaimana ia akan bekerja, sehingga para pengembang dapat menggunakan model itu sebagai blue print untuk system yang akan dikembangkan.

A. ELEMENT ELEMENT DASAR GUI RATIONAL ROSE



1. BROWSER
   Browser pada Rational Rose membantu kita untuk melihat secara hirarkis elemen-elemen model. Tanda positif (+) menandakan bahwa elemen tersebut mengandung informasi tambahan didalamnya. Dengan menekan tanda (+), informasi tersebut diperluas. Sebaliknya tanda negatif (-) menandakan informasi terbuka secara penuh.

2. TOOLBOX DIAGRAM
   Berisi tool-tool yang dibutuhkan dalam membuat sebuah diagram, toolbox ini berubah sesuai jenis diagram yang aktif.

3. JENDELA DIAGRAM
   Jendela Diagram menampilkan atau memodifikasi diagram dalam jendela diagram. Jika terdapat beberapa diagram yang dibuka pada saat yang sama, dapat ditampilkan secara berlapis (cascade) atau berkotak-kotak (tiled).

4. JENDELA DOKUMENTASI
   Jendela dokumentasi digunakan untuk membuat dokumentasi elemen-elemen model. Kita dapat membuat, melihat atau memodifikasi dokumentasi dalam jendela ini atau dalam jendela dokumentasi yang terdapat dalam spesifikasi. Jika jendela dokumentasi tidak muncul, pilih documentation dari menu view. Jika terdapat tanda ceklis pada documentation dan jendela ini belum muncul, gerakkan kursor ke bagian bawah browser. Saat pointer berubah menjadi kursor pembagi (biasanya tanda panah pada kedua ujungnya) tarik keatas untuk memunculkan jendela dokumentasi. Isi dokumentasi pada jendela ini adalah dokumentasi dari elemen yang kita sorot pada jendela diagram.

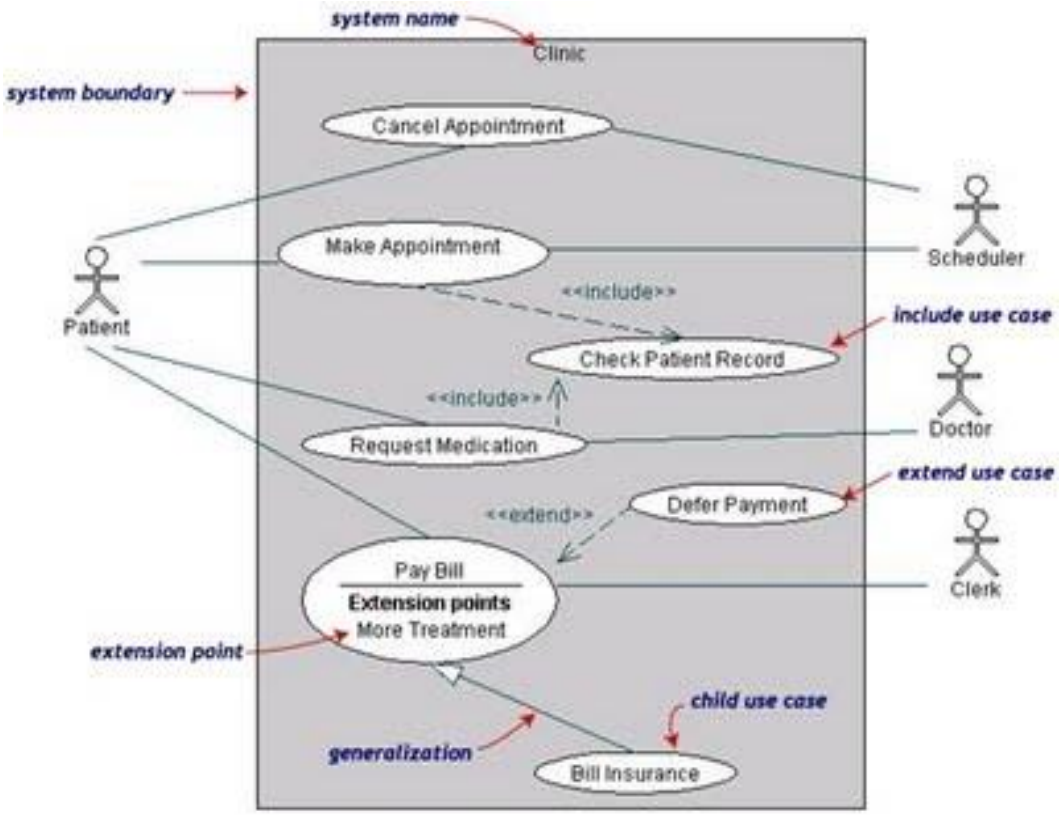
5. LOG
   Log atau log window ini di guanakan untuk menampilkan atau melaporkan kesalahan waktu

membuat sebuah diagram atau menampilkan error secara update juga menampilkan hasil dari proses yang telah kita lakukan ketika membuat diagram.

B. BERIKUT MACAM-MACAM DIAGRAM YANG ADA DI RATIONAL

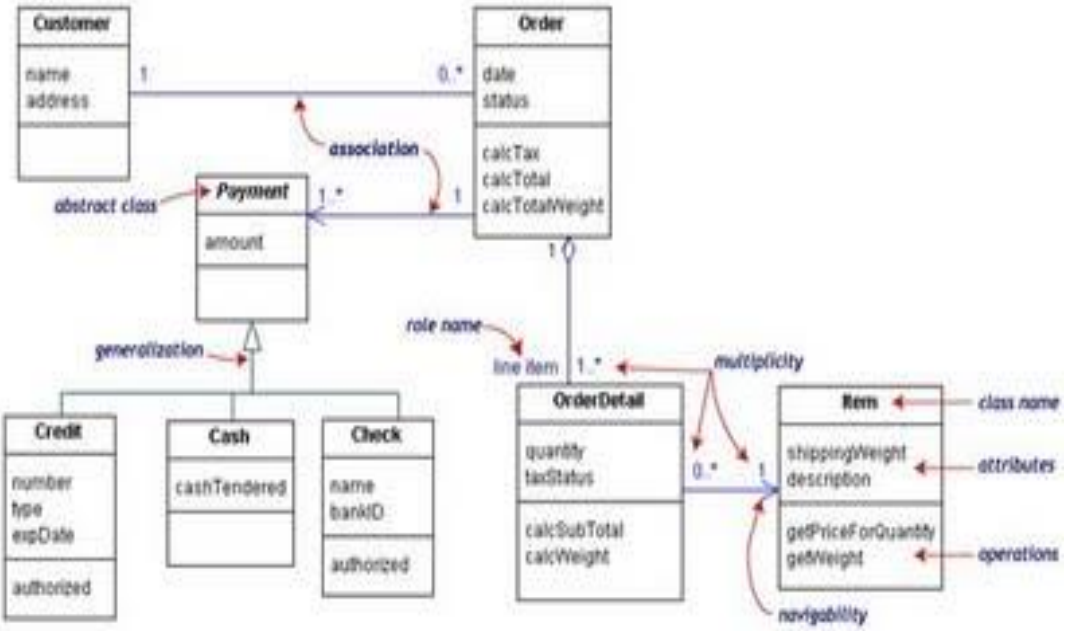
1. Use case diagram

Menjelaskan manfaat system jika dilihat menurut pandangan orang yang berada diluar system (actor). Use case diagram dapat digunakan selama proses analis untuk menangkap requirements system. Dan selama tahap design, use case diagram menetapkan perilaku system saat di implementasikan. Seperti yang ada pada screenshoot dibawah ini:



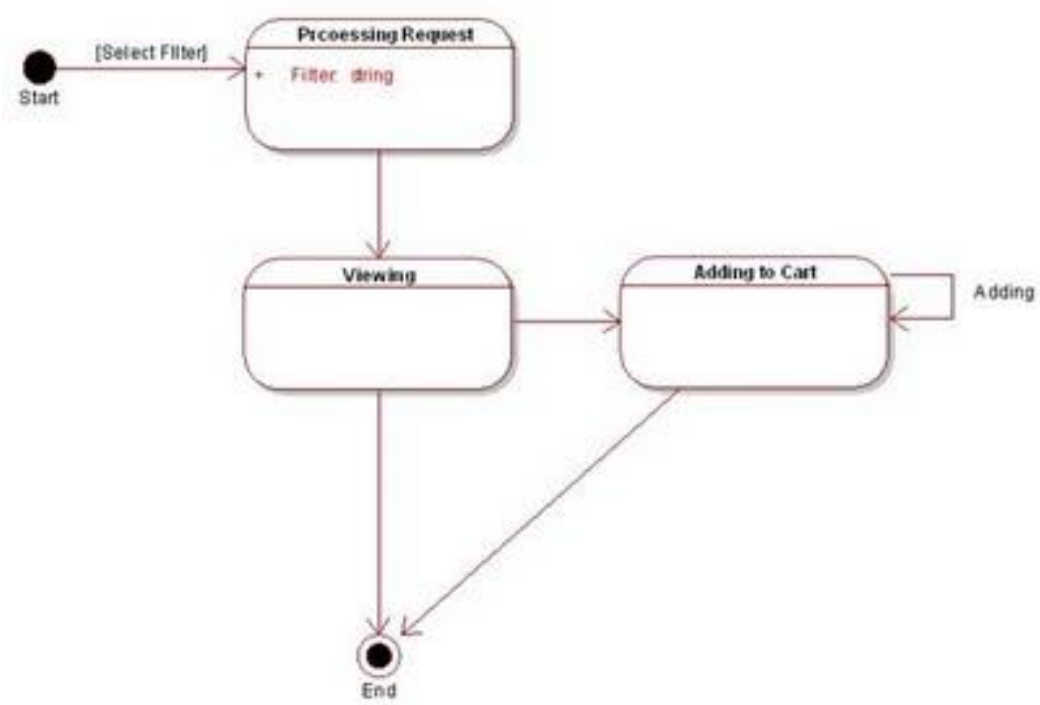
2. Class diagram

Membantu kita memvisualisasi struktur kelas-kelas dari suatu system. Class diagram memperlihatkan hubungan antar kelas dan penjelasan detail tiap-tiap kelas didalam model design (dalam logical view) dari suatu system. Seperti yang ada pada screenshoot dibawah ini:



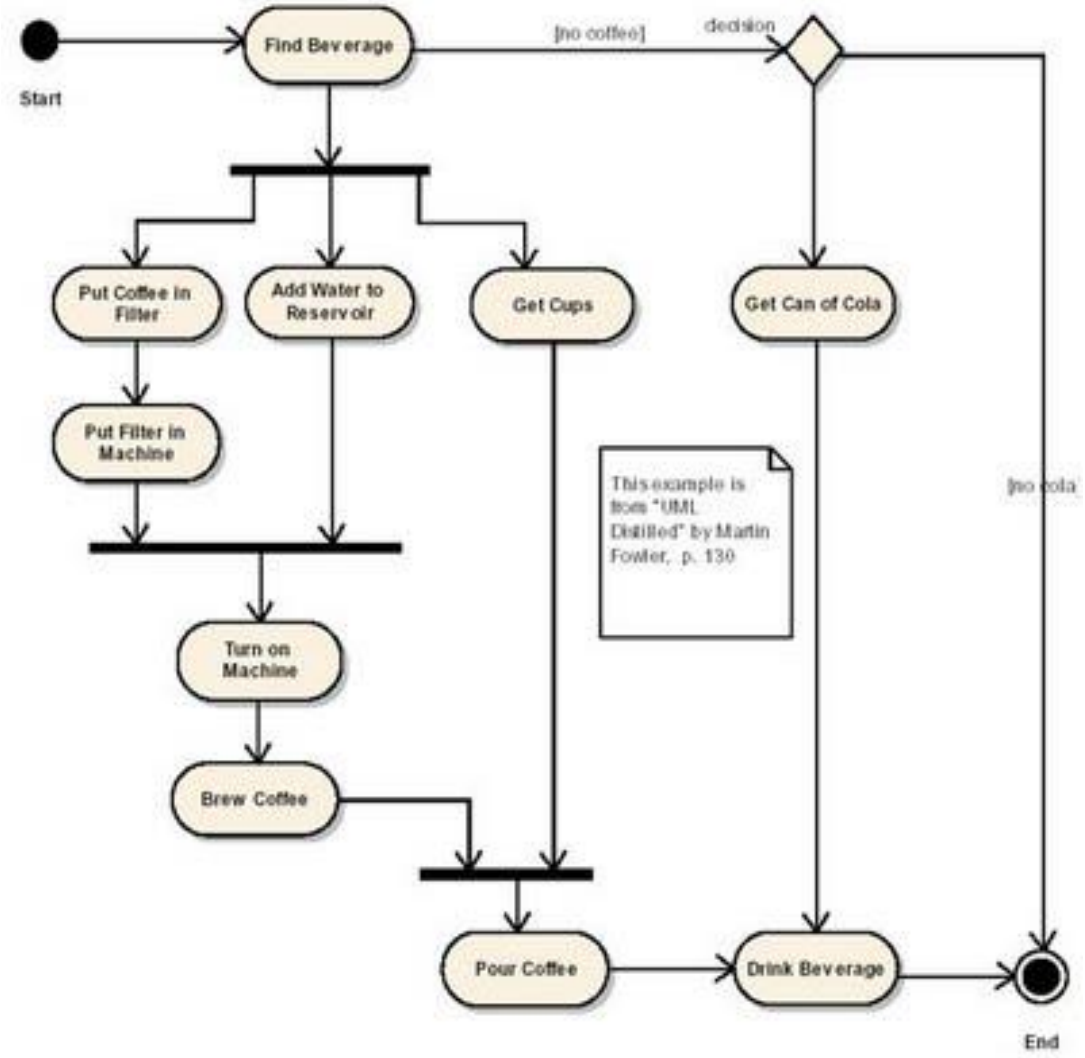
3. Statechart diagram

Kita dapat menggunakan statechart diagram untuk memodelkan perilaku dinamis satu kelas atau objek. Statechart diagram memperlihatkan urutan keadaan sesaat (state) yang dilalui sebuah objek, kejadian yang menyebabkan sebuah transisi dari satu state atau aktifitas. Statechart diagram khusunya diagunakan untuk memodelkan taraf-taraf diskrit dari sebuah siklus hidup objek. Seperti yang ada pada screenshoot dibawah ini:



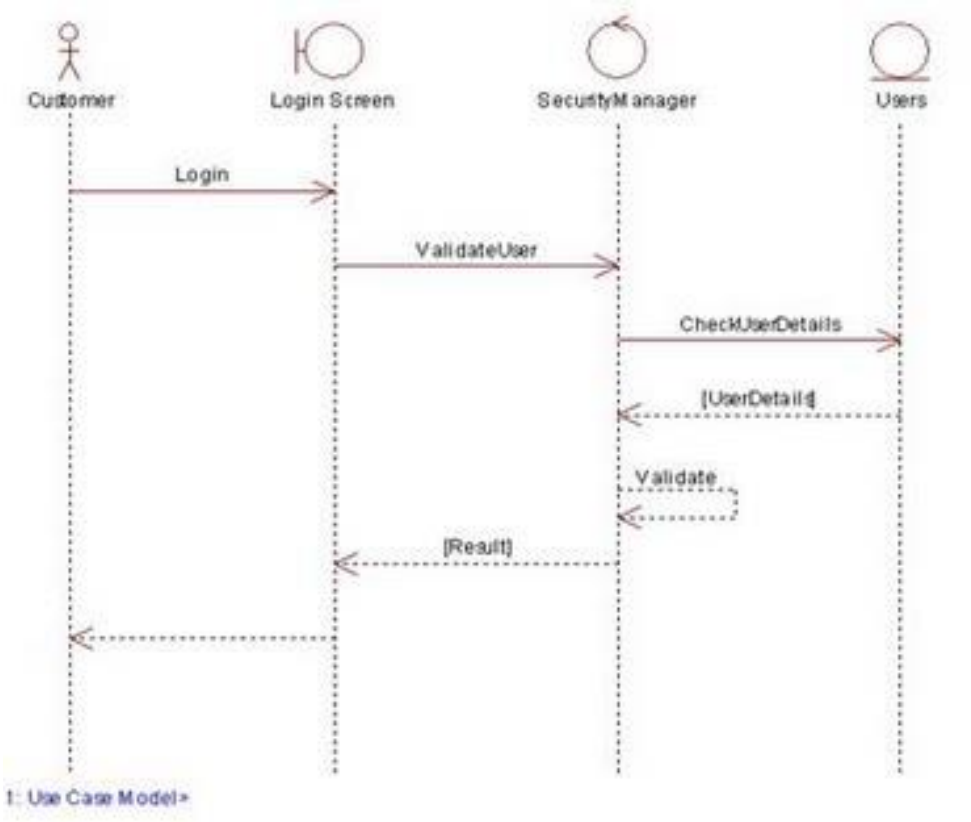
4. Aktifiti diagram

Memodelkan alur kerja (workflow) sebuah proses bisnis dan urutan aktivitas dalam suatu proses. Diagram ini sangat mirip dengan sebuah flowchart karena kita dapat memodelkan sebuah alur kerja dari satu aktivitas ke aktivitass lainnya atau dari satu aktivitas ke keadaan sesaat (state). Juga sangat berguna ketika ingin menggambarkan perilaku paralel atau menjelaskan bagaimana perilaku dalam berbagai use case berinteraksi. Seperti yang ada pada screenshoot dibawah ini:



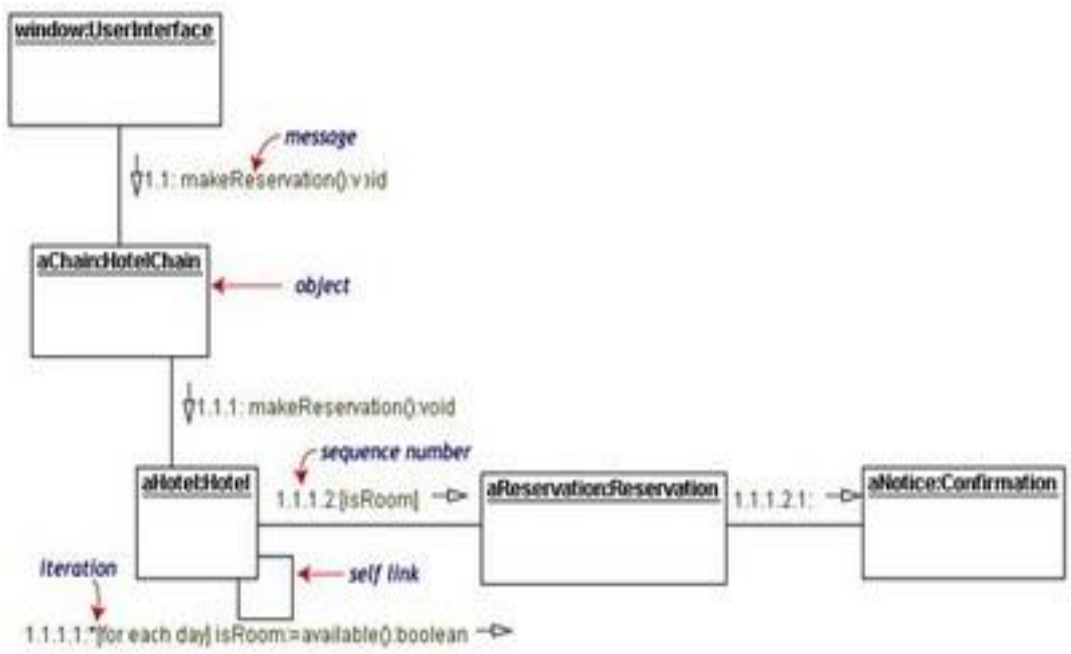
5. Sequence diagram

Menjelaskan interaksi obyek yang disusun dalam suatu urutan tertentu. Sequence diagram memperlihatkan tahap demi tahap apa yang seharusnya terjadi untuk menghasilkan sesuatu didalam use case. Seperti yang ada pada screenshoot dibawah ini: Collaboration diagram Melihat pada interaksi dan hubungan terstruktur antar obyek. Tipe diagram ini menekankan pada hubungan (relationship) antar obyek, sedangkan sequence diagram menekankan pada urutan kejadian. Collaboration diagram digunakan sebagai alat untuk menggambarkan interaksi yang mengungkapkan keputusan mengenai perilaku sistem. Seperti yang ada pada screenshoot dibawah ini:



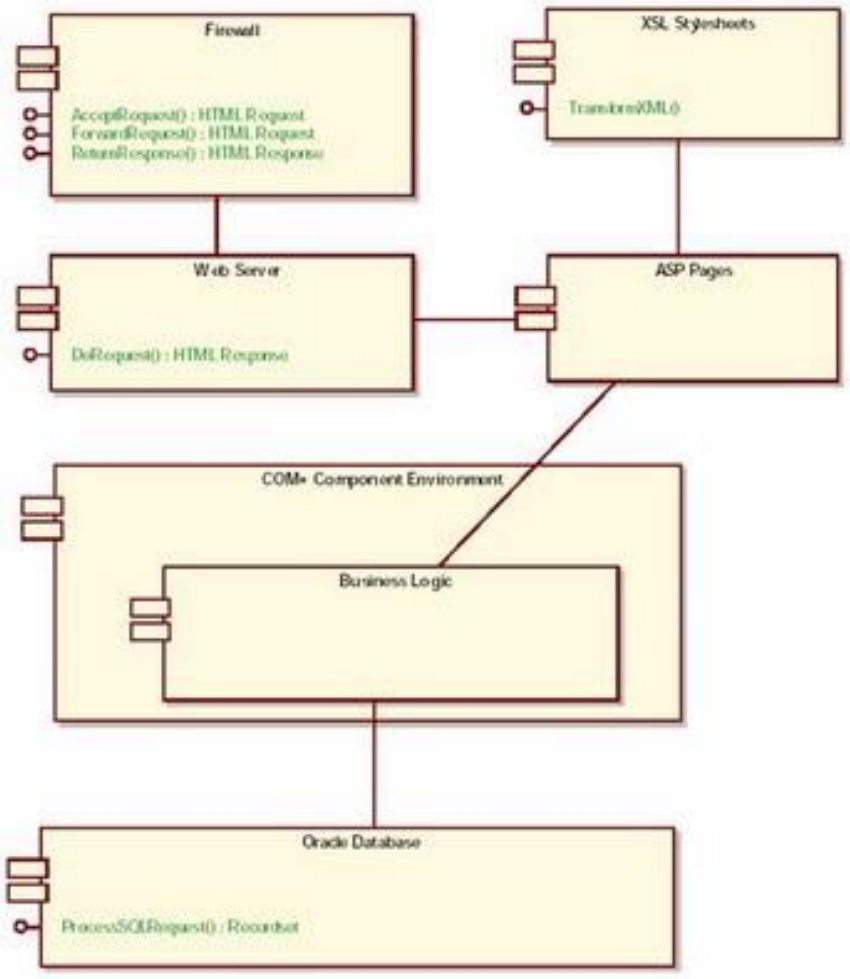
6. Collaboration diagram

Melihat pada interaksi dan hubungan terstruktur antar obyek. Tipe diagram ini menekankan pada hubungan (relationship) antar obyek, sedangkan sequence diagram menekankan pada urutan kejadian. Collaboration diagram digunakan sebagai alat untuk menggambarkan interaksi yang mengungkapkan keputusan mengenai perilaku sistem. Seperti yang ada pada screenshoot dibawah ini:



7. Component diagram

Menggambarkan alokasi semua kelas dan obyek kedalam komponenkomponen dalam desain fisik sistem software. Diagram ini mmperlihatkan pengaturan dan kebergantungan antara komponene-komponen software seperti source code, binary code dan komponen tereksekusi. Seperti yang ada pada screenshoot dibawah ini:



8. Deployment diagram

Diagram ini memperlihatkan pemetaan software kepada hardware.Diagram ini mengambarkan detail bagaimana komponen di-deploy dalam infrastruktur sistem, dimana komponen akan terletak, bagaimana kemampuan jaringan pada lokasi tersebut dan hal lain yang bersifat fisik (Sri Dharwiyanti dan Romi satria Wahono, 2003. Seperti yang ada pada screenshoot dibawah ini: