BAB II

LANDASAN TEORI

2.1 Pengertian Program Aplikasi

Program adalah rangkaian perintah yang sistematis yang disimpan dalam satu file sehingga menghasilkan satu hasil yang dikehendaki, sedangkan aplikasi adalah program yang dibuat dengan bahasa pemrograman tertentu yang bersifat umum.

Maka program aplikasi adalah sederetan kode yang digunakan untuk mengatur komputer agar melakukan sesuatu dengan keinginan yang membuatnya.

2.2 Pengertian Data Informasi

Data adalah elemen masukan sistem berupa data-data yang diperlukan sebagai bahan mentah untuk sistem tersebut yang kemudian diolah menghasilkan suatu informasi. Adapun pengertian umum dari data adalah sebagai berikut :

“Data adalah kenyataan yang menggambakan suatu kejadian – kejadian dan kesatuan nyata”. (H.M Jogianto,1993:8)

Setelah melalui suatu proses, data diolah menjadi informasi yang bermanfaat bagi pemakainya. Dalam hal ini definisi informasi adalah sebagai berikut:

“informasi adalah data yang telah diproses menjadi suatu bentuk yang yang berarti bagi penerima dan mempunyai nilai yang nyata atau bermanfaat

untuk pengambilan keputusan saat ini atau mendatang". (Gordon B. Davis,1992, Hal :28)

2.3 Teknologi JAVA

Java pertama kali diluncurkan pada tahun 1995 sebagai bahasa pemrograman umum dengan kelebihan dapat dijalankan pada web browser sebagai *applet*. Pada awalnya bahasa pemograman Java benama *Oak.* Yaitu bagian dari project *Green* yang dikembangkan khusus oleh *Sun Microsystem* untuk memprogram perangkat-perangkat elektronik rumah tangga. Bahasa Java mempunyai beberapa karakteristik yang membedakannya dengan bahasa pemograman lain, adapun karakteristik tersebut adalah :

1. Sederhana karena Java membuang fitur-fitur yang tidak diperlukan yang dimiliki oleh beberapa bahasa pemograman yang lain. Sebagai contoh, Java tidak mendukung pointer matematis, *Multiple Innheritance*, dan lain-lain.
2. Java merupakan bahasa pemograman yang murni berorientasi objek (OOP). dengan implementasi yang sangat baik. Bahasa berorientasi objek mempunyai ciri-ciri sebagai berikut:
    - Abstraksi yang baik untuk memudahkan pemahaman dan komunikasi.
    - Enkapsulasi: penyembunyian informasi dari pihak-pihak yang tidak memerlukannya sehingga kompleksitas tetap tertangani dengan baik.
    - Pewarisan dan *polymorphism*, teknik yang menyebabkan objek menjadi modular dan mudah untuk dicopot dan dipasang objek yang lain.

- Komposisi dan *Interface*, berguna untuk membuat tingkat kopling yang lebih rendah dan struktur hirarki objek-objek yang rapih sehingga bisa mengintegrasikan komponen-komponen untuk membuat sistem yang besar.

3. Program Java yang telah dibuat, dapat dijalankan di Windows, Machintos, Linux, Solaris tanpa harus mengkompilasi ulang atau menulis ulang kode, dukungannya terhadap XML membuat data menjadi portable,
4. Aplikasi-aplikasi yang dibangun dengan bahasa Java sangat handal dengan manajemen *garbage collection*. Keamanan yang bagus dengan penanganan *buffer overflow* yang umumnya menjadi lubang keamanan.
5. Java memiliki portabilitas yang tinggi. Ia dapat berada pada smart card, pager, PDA, TV, embeded device (PLC, micro controler), Laptop, PC, dan Server. Java dibagi menjadi tiga arsitektur, yaitu :
    1. Enterprise Java (J2EE) untuk aplikasi berbasis web. Merupakan superset dari Standar Java
    2. Standar Java (J2SE), ini adalah yang biasa di kenal sebagai bahasa Java, digunakan untuk aplikasi-aplikasi PC
    3. Micro Java (J2ME) merupakan subset dari J2SE dan salah satu aplikasinya yang banyak dipakai untuk *Wireless device/mobile device*.

2.3.1 Arsitektur Pemrograman JAVA

Arsitektur pemograman Java berbeda dengan pemograman lainnya secara umum. Kompiler Java akan mengkompilasi program Java menjadi Java *bytecode*,

Java *bytecode* ini merupakan instruksi mesin yang dikenali oleh JVM (Java *Virtual Machine*) yang kemudian akan dieksekusi sesuai dengan *platform* yang bersangkutan oleh JVM. JVM adalah lingkungan tempat eksekusi program Java dimana para objek saling berinteraksi satu dengan yang lainnya. *Virtual Machine* inilah yang menyebabkan Java mempunyai kemampuan penanganan memori yang lebih baik, keamanan yang lebih tinggi serta portabilitas yang besar.

Kode Java (Java *bytecode*) senantiasa mengikuti spesifikasi dan semantik yang didefinisikan oleh JVMS (Java *Virtual Machine Specification*). Kode Java yang digunakan untuk memprogram akan senantiasa mengikuti spesifikasi dan semantik yang didefinisikan dalam JLS (Java *Language Specification*).

2.3.2 JAVA untuk *platform* Android

Teknologi Java untuk Android sama seperti halnya bahasa pemrograman Java pada umumnya yang sedikit berbeda terletak pada API yang dipakai, Dalam membangun aplikasi Java Android ini bisa memakai 2 aplikasi sebagai *editor*, *Debuger* dan *Builder*, yang pertama Java Development Environment(JDE) dan Java Eclipse, Kedua *software* aplikasi ini memakai API Java yang sama.

2.4 Java Development Kit 1.6 (JDK 1.6)

JDK merupakan sekumpulan program dan *liblary* Java yang digunakan untuk membuat aplikasi Java. Software JDK versi 1.6 atau versi yang lebih tinggi dapat didapatkan secara gratis di http://Java.sun.com/j2se/1.6/download.html.

Untuk menginstal versi Win32 pada Windows 95, 98, 2000, NT 4.0, Vista Dan Windows7 dibutuhkan sistem yaitu prosesor 486/DX atau lebih dan RAM

minimum 32 Megabyte. Adapun langkah-langhkah untuk instalasi JDK adalah sebagai berikut :

1. Double klik icon j2sdk1_6_0-win,
2. Setelah installer Java 2 JDK membuka file-file instalasi, akan ditampilkan perjanjian lisensi, dan kotak untuk memilih lokasi penyimpanan file JDK 1.6
3. Pilih lokasi default atau lokasi lain utuk menyimpan file JDK 1.6

Pada JDK 1.6 terdapat beberapa program bantu yang digunakan untuk mengkompilasi dan menjalankan program seperti Java, Javac, dll.

1.5 Android

1.5.1 Pengertian Android

Android adalah sekumpulan perangkat lunak yang ditujukan bagi perangkat yang bergerak mencakup sistem Operasi *middleware*, dan aplikasi kunci, android *Standart development ki*t (SDK) menyediakan perlengkapan dan *Apllication Programming interface* (API) yang diperlukan untuk mengembangkan aplikasi pada *platform* Android menggunakan bahasa pemrograman JAVA. Jeff Lessard (2010;1-2) *Android is an operation System*(OS) *development by open Handset Alliance* (OHA) yaitu aliansi perangkat selular terbuka yang terdiri dari 47 perusahaan Hardware, Software dan perusahaan telekomunikasi ditujukan untuk mengembangkan standar terbuka bagi perangkat selular.

1.5.2 Sejarah Android

Pada mulanya terdapat berbagai macam sistem operasi pada perangkat selular, diantaranya sistem operasi Symbian, Microsoft windows Mobile, Mobile Linux, iPhone, dan Sistem operasi lainnya. Namun diantara sistem operasi yang ada belum mendukung standar dan penerbitan API yang dapat dimanfaatkan secara keseluruhan dan dengan biaya yang murah. Kemudian Google ikut berkecimpung didalamnya dengan *platform* Android, yang menjanjikan keterbukaan, keterjangkauan, *open source* dan *framework* yang berkualitas.

Pada tahun 2005, google mengakuisisi perusahaan Android Inc. Untuk memulai perkembangan platform Android. Dimana terlibat dalam pengmbangan ini Andy Rubin, Rich Miner, Nick Sears, Dan Chris White. Pada pertengahan 2007 sekelompok pemimpin industri bersama-sama membentuk aliansi perangkat selular terbuka, *Open Handset Alliance* (OHA). Bagian dari tujuan aliansi ini adalah berinovasi dengan cepat dan menanggapi kebutuhan konsumen dengan lebih baik, dengan produk awalnya adalah *platform* Android. Dimana Android dirancang untuk melayani kebutuhan operator telekomunikasi, manufaktur handset, dan pengembangan aplikasi.

OHA berkomitmen untuk membuat Android *open Source* dengan Lisensi Apache Versi 2.0, Android Pertama kali diluncurkan pada 5 november 2007, dan *Smartphone* pertama yang menggunakan sistem android dikeluarkan oleh T-Mobile dengan sebutan G1 pada bulan september 2008. Hingga Saat ini Android telah merilis beberpa versi Android untuk menyempurnakan versi sebelumnya.

Selain berdasarkan penomoran, pada setiap versi Android terdapat kode nama berdasarkan nama-nama kue. Hingga saat ini sedah terdapat beberapa versi yang telah diluncurkan, diantaranya : versi 1.5 dirilis pada 30 april 2009 diberi nama Cupcake, versi 1.6 dirilis pada 15 september 2009 diberi nama Donut, versi 2.1 dirilis pada 26 Oktober 2009 diberi na,a Eclair, versi 2.2 diberinama Frozen Yoghurt (Froyo), 2.3 diberinama GinggerBread, Versi 3.0 Diberi nama HoneyComb, Versi 4.0 Diberi nama Ice Cream Sandwich dan versi 4.2 Jelly Bean.

1.5.3 Kelebihan Android

Sudah banyak *platform* untuk perangkat seluler saat ini, termasuk didalmnya Symbian, iPhone, Windows mobile, Blackberry, Java Mobile Edition, linux mobille. Dan masih banyak lagi. Namun ada beberapa hal yang menjadi kelebihan Android. Walaupun beberapa fitur-fitur yang ada telah muncul sebelumnya pada *platform* lain, android adalah yang pertama menggabungkan hal seperti berikut:

1. Keterbukaan, bebas pengembangan tanpa dikenakan biaya terhadap sistem karena berbasis Linux dan open soure. Pembuatan perangkat menyukai hal ini karena dapat membangun *platform* yang sesuai yang diinginkan tanpa harus membayar royalty. Sementara pengembangan software menyukai karena Android dapat digunakan diperangkat manapun dan tanpa terikat oleh vendor manapun.

2. Arsitektur komponen dasar Android terinspirasi dari teknologi internet Mashup. Bagian dalam sebuah aplikasi dapat digunakan oleh aplikasi lainnya, bahkan dapat diganti dengan komponen lain yang sesuai dengan aplikasi yang dikembangkan.
3. Banyak dukungan servis, kemudian dalam menggunakan berbagai macam layanan pada aplikasi seperti penggunaan layanan pencarian lokasi, database SQL, browser dan penggunaan peta. Semua itu sudah tertanam pada Android sehingga memudahkan dalam pengembangan aplikasi.
4. Siklus hidup aplikasi diatur secara otomatis, setiap program terjaga antara satu sama lain oleh berbagai lapisan keamanan, sehingga kerja sistem menjadi lebih stabil. Pengguna tidak perlu khawatir dalam menggunakan aplikasi pada perangkat yang memorinya terbatas.
5. Dukungan grafis dan suara terbanyak, dengan adanya dukungan 2D grafis dan animasi yang diilhami oleh Flash menyatu dalam 3D menggunakan OpenGL memungkinkan membuat aplikasi maupun game yang berbeda.
6. Portabilitas aplikasi, aplikasi dapat digunakan pada perangkat yang ada saat ini maupun yang akan datang. Semua program ditulis engan menggunkan bahasa pemrograman JAVA dan dieksekusi oleh mesin virtual Dalvik, Sehingga kode program portabel antara ARM,X86, dan aristektur lainnya. Sama halnya dengan dukungan masukan seperti penggunaan Keyboard, layar sentuh, trackball dan resolusi layar semua dapat disesuaikan dengan program.

1.6 Eclipse

Eclipse adalah sebuah IDE (*integrated Development Environment*) untuk mengembangkan perangkat lunak dan dapat dijalankan disemua *platform* (*platform independent)*. Berikut ini adalah sifat dari Eclipse:

1. *Multi-platform*: target sistem operasi Eclipse adalah Microsoft Windows, Linux, Solaris, AIX, HP-UX dan Mac OS X.
2. *Multi-language*: Eclipse dikembangkan dengan bahasa pemrograman Java, akan tetapi Eclipse mendukung pengembangan aplikasi berbasis bahasa pemrograman lainnya, seperti C/C++, Cobol, Phthon, Perl, PHP dan lain sebagainya.
3. *Multi-role*: selain sebagai IDE untuk pengembangan aplikasi, eclipse pun bisa digunakan untuk aktivitas dalam siklus pengembangan perangkat lunak, seperti dokumentasi, test perangkat lunak, pengembangan web, dan lain sebagainya. Eclipse pada saat ini merupakan salah satu IDE favorit dikeranakan gratis dan *open source*, yang berarti setiap orang boleh melihat kode pemrograman perangkat lunak ini. Selain itu, kelebihan dari eclipse yang membuatnya populer adalah kemampuannya untuk dapat dikembangkan oleh pengguna dengan komponen yang dinamakan *plugin*

1.6.1 Sejarah Eclipse

Eclipse awalnya dikembangkan oleh IBM untuk menggantikan perangkat lunak IBM *Visual Age for Java* 4.0. produk ini diluncurkan oleh IBM pada tanggal 5 November 2001, yang menginvestasikan sebanyak US$ 40 Juta untuk

pengembangannya. Semenjak itu konsumen Eclipse foundation mengambil alih untuk pengembangan Eclipse lebih lanjut dan pengaturan organisasinya.

1.6.2 Arsitektur Eclipse

Sejak versi 3.0 eclipse pada dasarnya merupakan sebuah kernel, yang mengangkat *plug-in*. Apa yang dapat digunakan didalam Eclipse sebenernya adalah fungsi dari *plug-in* yang sudah diinstal. Ini merupakan basis dari Eclipse yang dinamakan *Rich Client Platform* (RCP). Berikut ini adalah komponen yang membentuk RCP:

1. Core platform
2. OSGi
3. SWT (standar Widget Toolkit)
4. Jface
5. Eclipse Workbench

Sejarah standar Eclipse selalu dilengkapi dengan JDT (java Development Tools), *Plug-in* yang membuat Eclipse kompatibel untuk mengembangkan program Java, dan PDE (*Plug-in Development Environment*) untuk mengembangkan *plug-in* baru. Eclipse beserta *plug-in* nya diimplementasikan dalam bahasa pemrograman Java. Konsep Eclipse Adalah IDE yang terbuka (*open*), mudah diperluas (*extensible*) untuk apa saja, dan tidak untuk sesuatu yang spesifik. Jadi, eclipse tidak saja untuk mengembangkna program java, akan tetapi dapat digunakan untuk berbagai macam keperluan, cukup dengan menginstal *plug-in* yang dibutuhkan. Apabila ingin menembangkan program C/C++ terdapat *Plug-in* CDT

(C/C++ Development tools). Selain itu, pengembangan secara visual bukan hal yang tidak mungkin oleh Eclipse, *plug-in* UML tersedia untuk membuat diagram UML. Dengan menggunkan PDE setiap orang bisa membuat *Plug-in* sesuai dengan keiinginannya.

1.6.3 Perkembangan Eclipse

Sejak tahun 2006, Eclipse foundation mengkoordinasikan peluncuran eclipse secara rutin dan simultan yang dikenal dengan nama *Simultaneous release*. Setiap versi peluncuran terdiri dari eclipse *platform* dan juga sejumlah proyek yang terlibat dalam proyek Eclipse. Tujuan dari sistem ini adalah untuk menyediakan distribusi eclipse dengan fitur-fitur dan versi yang terstandarisasi. Hal ini juga dimaksudkan untuk mempermudah deployment dan *maintenance* untuk sistem enterprise. Adapun versi eclipse yang telah diluncurkan yaitu:

1. Eclipse 3.0
2. Eclipse 3.1
3. Callisto
4. Europa
5. Ganymede
6. Galileo
7. Helios
8. Juno

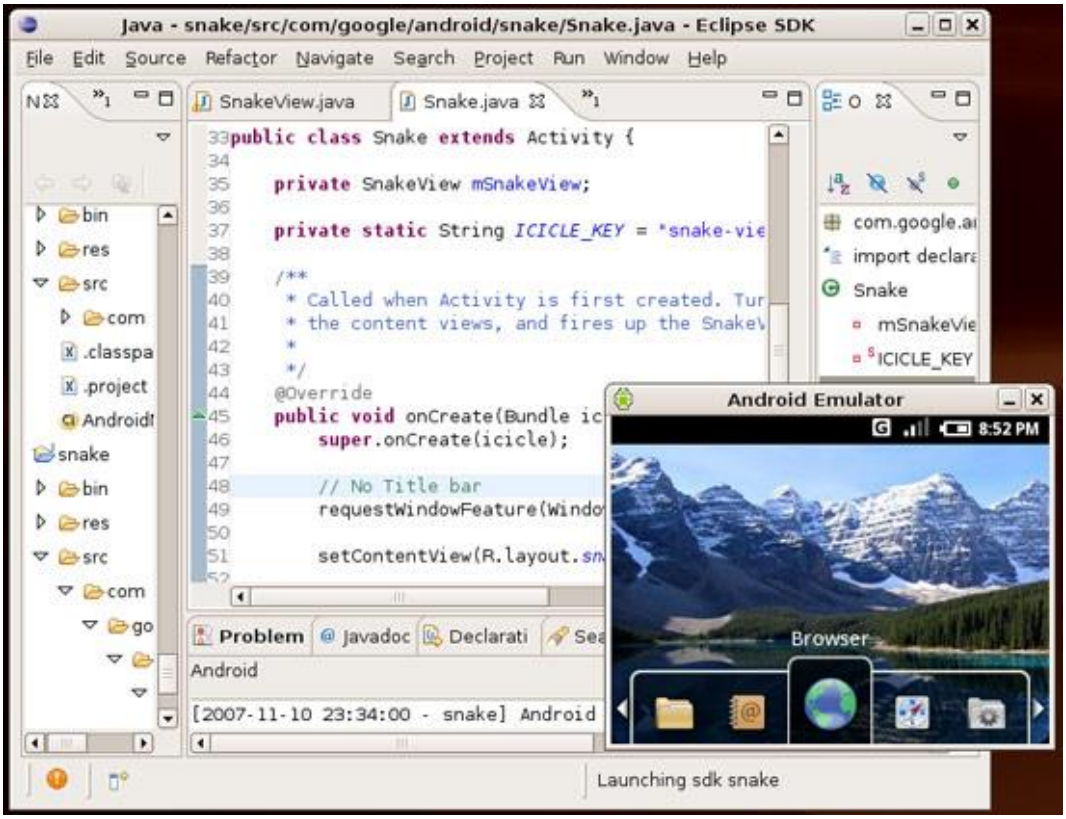
1.7 ADT

Android development tools (ADT) adalah plugin untuk eclipse *integrated development Environment (IDE)* yang dirancang untuk memberikan lingkungan yang terpadu dimana untuk membangun aplikasi Android. ADT memperluas kemampuan Eclipse untuk membiarkan para developer lebih cepat dalam membuat proyek baru.

Android, membuat aplikasi UI, manambahkan komponen berdasarkan Android *Framework* API, *Debug* aplikasi dalam penggunaan Android SDK, dan membuat *file* APK untuk mendistribusikan aplikasi. Mengembangkan aplikasi di Eclipse dengan ADT sangat dianjurkan dan merupakan cara tercepat untuk memulai Aplikasi Android, karena banyak kemudahan kemudahan sebagai *tools* yang terintegrasi seperti costum XML editor, dan *debug panel output.* Selain itu ADT memberikan dorongan luar biasa dalam mengembangkan aplikasi Android.

1.8 ANDROID SDK

Segala yang dibutuhkan dalam pembangunan Android termasuk penulisan syntak program menggunakan *editor*, untuk men-*debug* menggunakan array dari Perangkat simulator Android dapat dilakukan, juga untuk membangun dan melegalkan aplikasi kedalam Perangkat Android sebenarnya. Berikut dibawah ini adalah gambar Android SDK setalah terinstal di Komputer dan akan digunakan untuk pertama kalinya.



Gambar 2.1 Contoh ANDROID SDK, Sebagai Editor Program

1.9 Android Simulator

Desain Simulator merupakan perangkat tambahan yang terdapat dalam komponen Android (SDK), simmulator ini layaknya sebuah handphone Android nyata yang dapat mengakses internet dan segala macam hal terkait ponsel pada umumnya, fungsi utama dari simulator ini adalah untuk menjalankan, program yang telah dibuat sebelumnya. Simulator ini juga dapat diunduh secara terpisah sesuai simulator yang diinginkan,ada berbagai macam simulator,.



Gambar 2.2 Simulator Android

Simulator yang penulis gunakan adalah simulator Android, simulator ini menggunakan teknologi layar sentuh/ *Touch Pad*.

1.10 *Unifed Modelling Language* (UML)

Unified Modelling Language (UML) adalah sebuah "bahasa" yg telah menjadi standar dalam industri untuk visualisasi, merancang dan

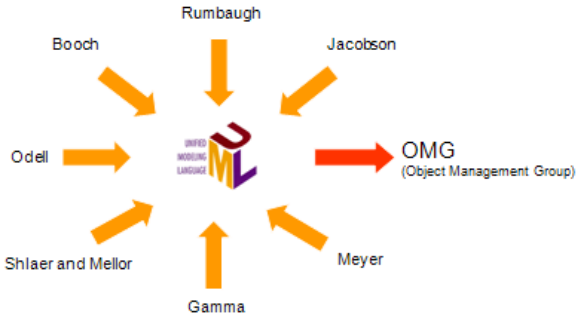
mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem.

Dengan menggunakan UML programer 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*. 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 diketahui puluhan metodologi pemodelan berorientasi objek telah bermunculan didunia. 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 bekerjasama dengan group/perusahaan lain yang menggunakan metodologi yang berlainan.



Gambar 2.3 Konsep Unified Modeling Language

(sumber : Sri dharwiyanti , 2003)

konsepsi dasar UML bisa dirangkumkan dalam gambar dibawah

Major Area	*View*	*Diagrams*	*Main Concepts*
structural	static view	class diagram	class, association, generalization, dependency, realization, interface
	use case view	use case diagram	use case, actor, association, extend, include, use case generalization
	implementation view	component diagram	component, interface, dependency, realization
	deployment view	deployment diagram	node, component, dependency, location
dynamic	state machine view	statechart diagram	state, event, transition, action
	activity view	activity diagram	state, activity, completion transition, fork, join
	interaction view	sequence diagram	interaction, object, message, activation
		collaboration diagram	collaboration, interaction, collaboration role, message
model management	model management view	class diagram	package, subsystem, model
extensibility	all	all	constraint, stereotype, tagged values

Gambar 2.4 Konsep dasar UML

(sumber : Sri dharwiyanti , 2003)

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

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

- *use case diagram*
- *class diagram*

- *statechart diagram*
- *activity diagram*
- *object diagram*
- *sequence diagram*
- *collaboration diagram*
- *component diagram*
- *deployment diagram*

1.10.1 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 analis sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem.

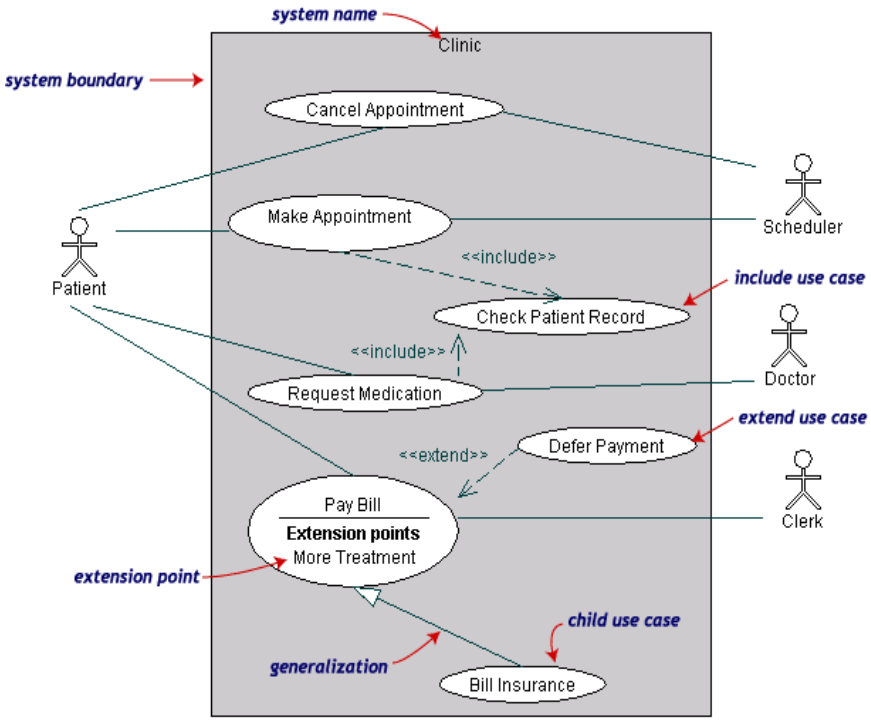
Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang

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 2.5 Contoh Diagram *Usecase*
(sumber : Sri dharwiyanti , 2003)

1.10.2 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).

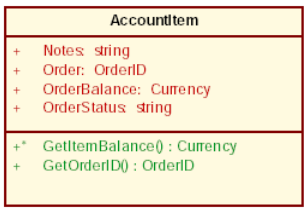
Class diagram menggambarkan struktur dan deskripsi *class, package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain.

Class memiliki tiga area pokok :

1. Nama (dan stereotype)
2. Atribut
3. Metoda

Atribut dan metoda dapat memiliki salah satu sifat berikut :

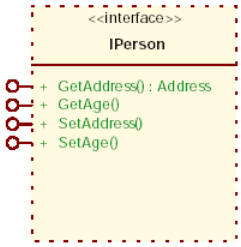
- *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan
- *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya
- *Public*, dapat dipanggil oleh siapa saja



Gambar 2.6 Contoh *Class Diagram (attribut)*

(sumber : Sri dharwiyanti , 2003)

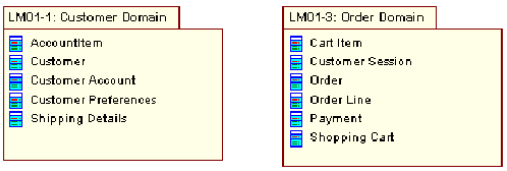
Class dapat merupakan implementasi dari sebuah *interface*, yaitu *class* abstrak yang hanya memiliki metoda. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class.* Dengan demikian *interface* mendukung resolusi metoda pada saat *run-time.*



Gambar 2.7 Contoh *Class Diagram (interface)*

(sumber : Sri dharwiyanti , Ilmu Komputer .com)

Sesuai dengan perkembangan *class* model, *class* dapat dikelompokkan menjadi *package*. programer juga dapat membuat diagram yang terdiri atas *package.*



Gambar 2.8 Contoh Hubungan antar *Class (package)*

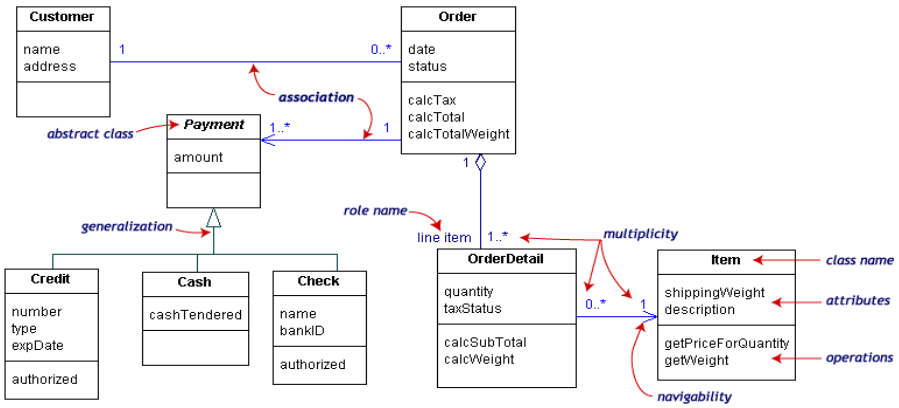
(sumber : Sri dharwiyanti , Ilmu Komputer .com)

1. Asosiasi, yaitu hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain. Panah *navigability* menunjukkan arah *query* antar *class*.

2. Agregasi, yaitu hubungan yang menyatakan bagian (“terdiri atas..”).

3. Pewarisan, yaitu hubungan hirarkis antar *class*. *Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metoda *class* asalnya dan menambahkan fungsionalitas baru, sehingga ia disebut anak dari *class* yang diwarisinya. Kebalikan dari pewarisan adalah generalisasi.

4. Hubungan dinamis, yaitu rangkaian pesan (*message*) yang di-*passing* dari satu *class* kepada *class* lain. Hubungan dinamis dapat digambarkan dengan menggunakan *sequence diagram* yang akan dijelaskan kemudian.



Gambar 2.9 Contoh *Class Diagram*
(sumber : Sri dharwiyanti , Ilmu Komputer .com)

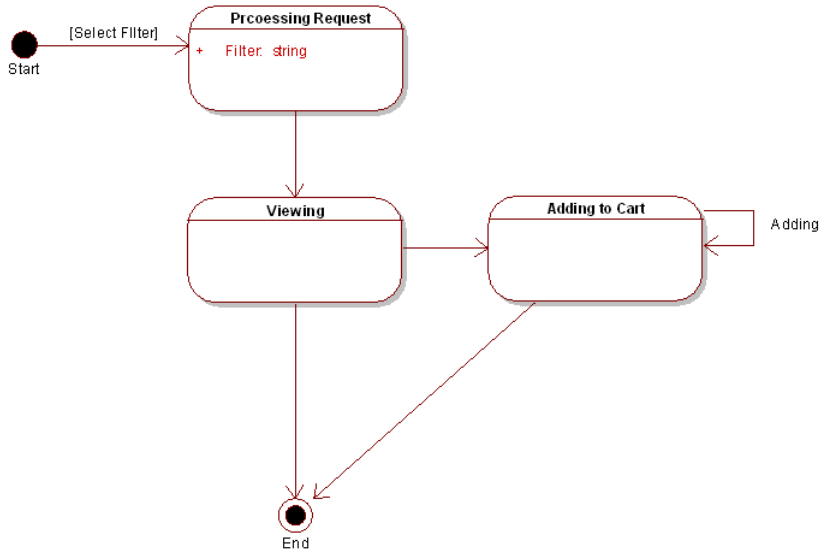
1.10.3 Statechart Diagram

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.

Contoh *statechart diagram* :



Gambar 2.10 Contoh *Statechart Diagram*
(sumber : Sri dharwiyanti , Ilmu Komputer .com)

1.10.4 Activity Diagram

Activity diagram 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.

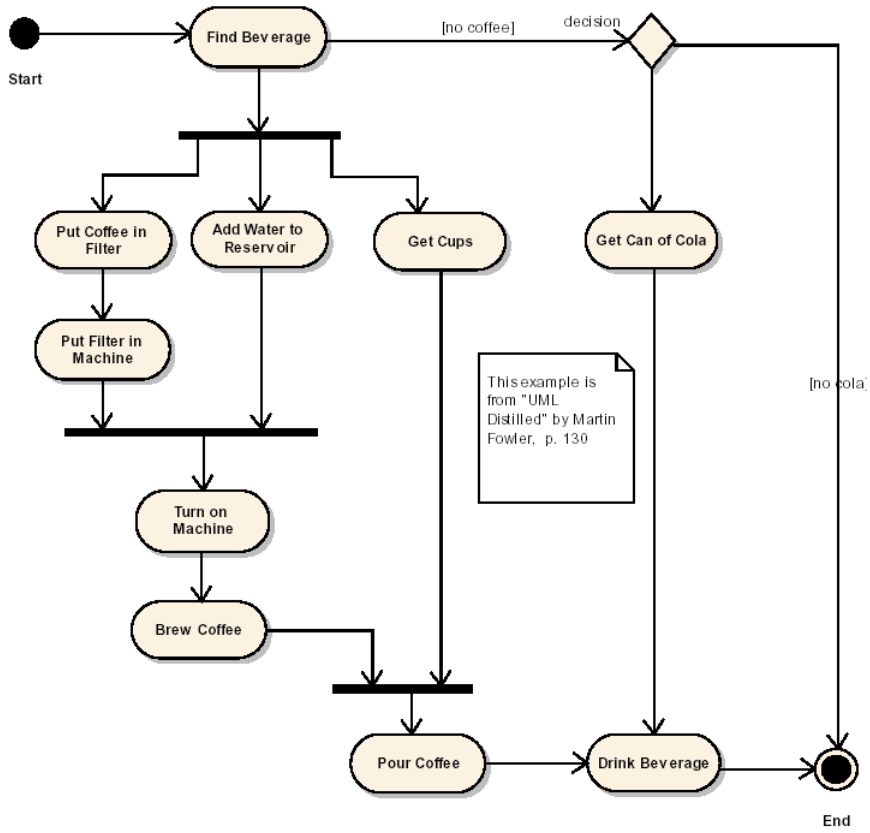
Activity diagram merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan behaviour internal sebuah sistem (dan interaksi 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.

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:*



Gambar 2.11 Contoh *Activity Diagram*

(sumber : Sri dharwiyanti , Ilmu Komputer .com)

1.10.5 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 yangterkait).

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.

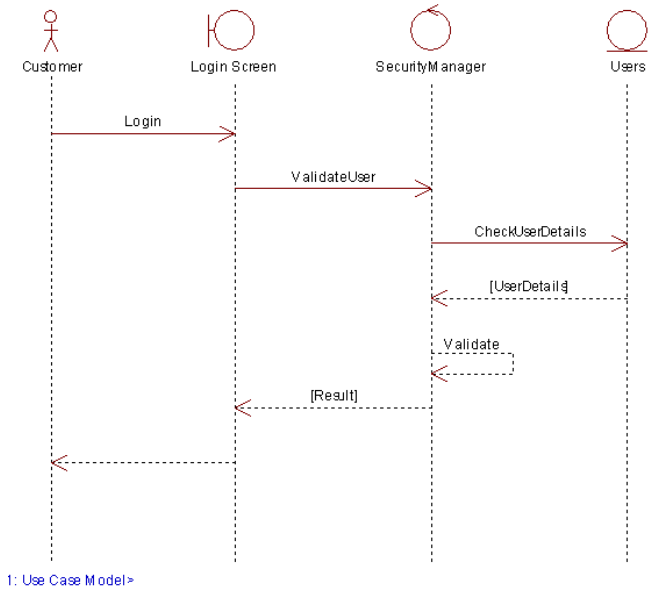
Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal.

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* :

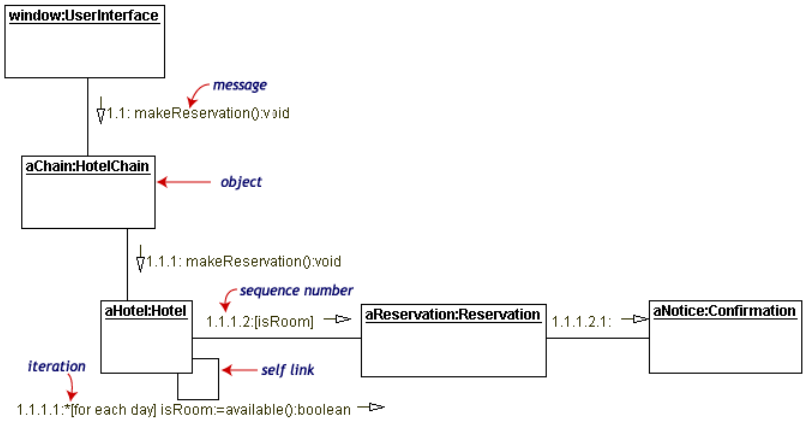


Gambar 2.12 Contoh *Sequance Diagram*
(sumber : Sri dharwiyanti , Ilmu Komputer .com)

1.10.6 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. Messages dari level yang sama memiliki prefiks yang sama.



Gambar 2.13 Contoh *Collaboration Diagram*

(sumber : Sri dharwiyanti , Ilmu Komputer .com)

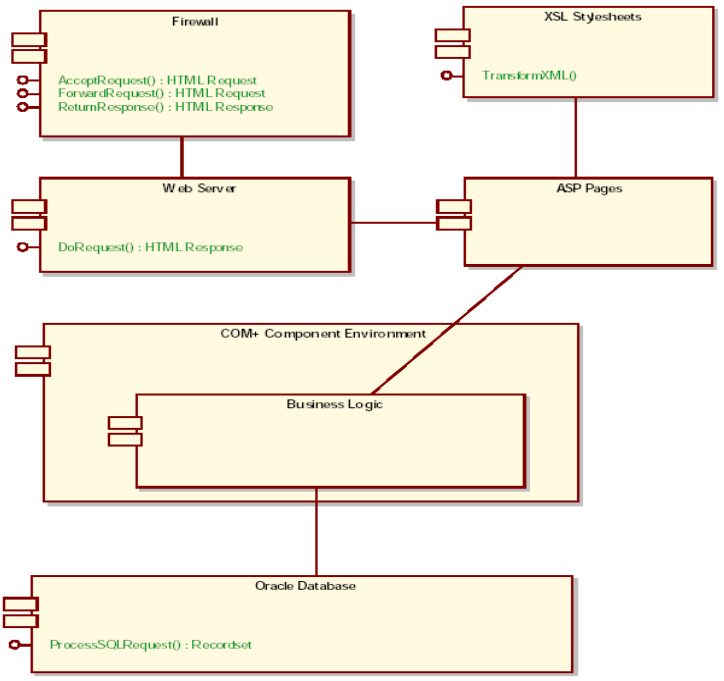
1.10.7 Component Diagram

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*:



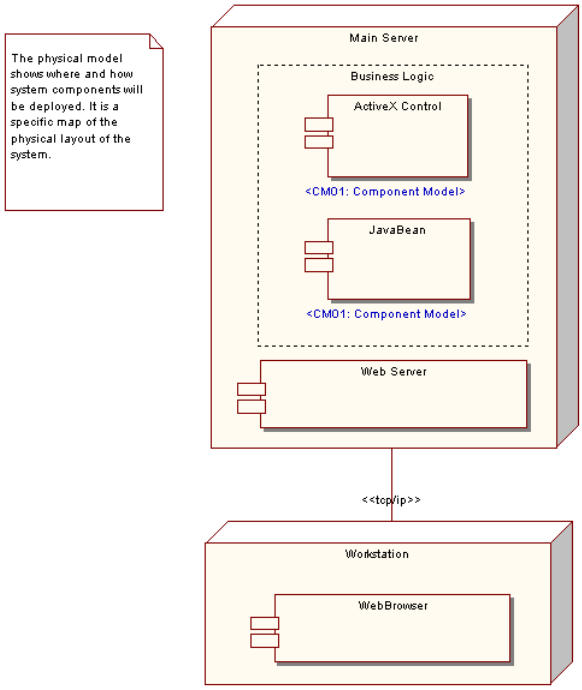
Gambar 2.14 Contoh *Component Diagram*

(sumber : Sri dharwiyanti , Ilmu Komputer .com)

1.10.8 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.



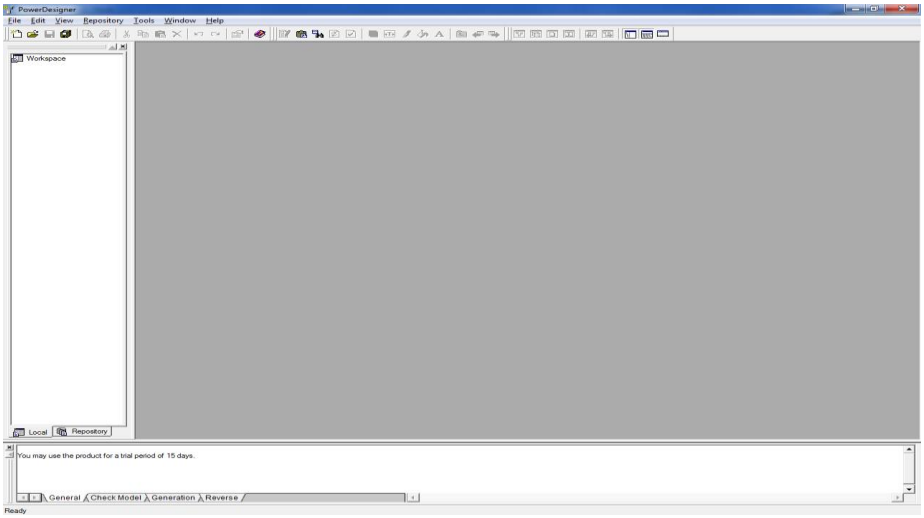
Gambar 2.15 Contoh *Deployment Diagram*
(sumber : Sri dharwiyanti , Ilmu Komputer .com)

1.11 Sybase Power Designer 12

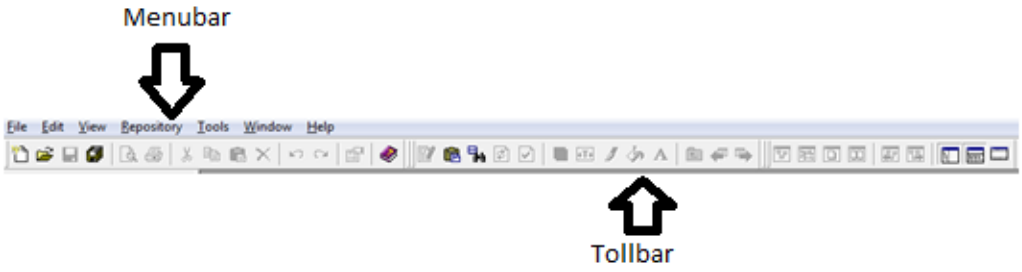
Power designer mengintegrasikan dukungan RDBMS dan pengembangan untuk Java,.NET,PowerBuilder Dan *Web Service*. Ini adalah alat penggabung beberapa model standar teknik, aplikasi pemodelan melalui UML, Business Process Modeling dan didukung solusi manajemen data.

Power Designer mendukung analisis persyaratan rinci terkait dengan semua model untuk persyratan dokumen. Dan dapat sinkronisasi dengan Microsoft Word menempatkan pengguna langsung ke dalam proses.

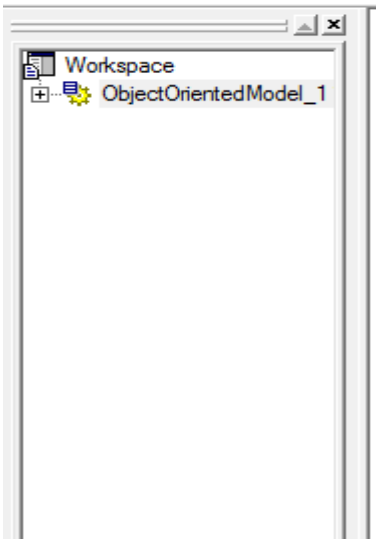
Berikut gambar aplikasi dari Power Designer 12:



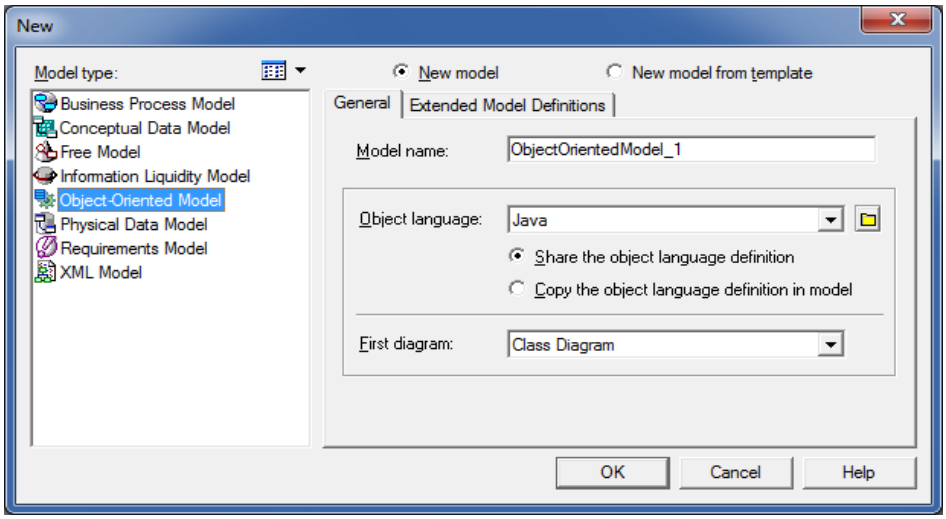
Gambar 2.16 *Field* utama dari Power Designer 12



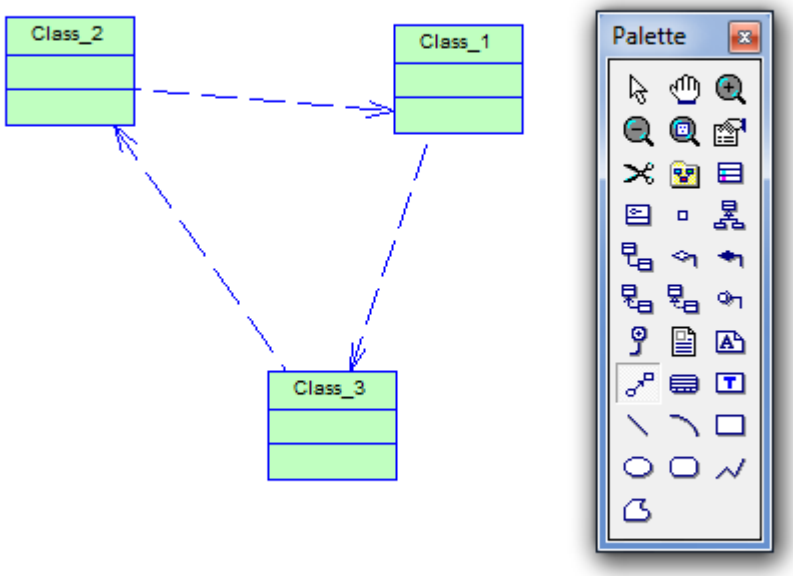
Gambar 2.17 Menu bar Dan Toolbar dari Power Designer 12



Gambar2.18 Workspace dari Power Designer 12



Gambar 2.19 Pilihan Menu Yang akan dibuat di Power Designer 12



Gambar 2.20 Contoh Pengerjaan dari Menu Object Oriented Program (OOP) dari Power Designer 12