BAB II

LANDASAN TEORI

II.1. Perancangan

Perancangan menurut Hanik Mujiati mempunyai 2 maksud, yaitu untuk memenuhi kebutuhan kepada pemakai sistem dan untuk memberikan gambaran yang jelas kepada pemogram komputer dan ahli-ahli teknik lainnya yang terlibat (Hanik Mujiati, 2012).

II.2. Pengertian Aplikasi

Aplikasi adalah suatu program yang dibuat oleh pemakai yang ditujukan untuk melakukan suatu tugas khusus. Berdasarkan definisi di atas maka dapat disimpulkan bahwa aplikasi adalah program yang dibuat untuk melakukan tugas khusus dalam perusahaan (Neti, 2010).

Beberapa aplikasi yang digabung bersama menjadi suatu paket kadang disebut sebagai suatu paket atau suite aplikasi *(application suite)*. Aplikasi-aplikasi dalam suatu paket biasanya memiliki antar muka pengguna yang memiliki kesamaan sehingga memudahkan pengguna untuk mempelajari dan menggunakan tiap aplikasi. Sering kali, mereka memiliki kemampuan untuk saling berinteraksi satu sama lain sehingga menguntungkan pengguna.

II.3. Pengenalan Algoritma

Istilah algoritma berasal dari nama seorang pengarang berkebangsaan Arab bernama Ja'fat Mohammed bin Musa al Khowarizmi (tahun 790 – 840), yang sangat terkenal dengan sebutan bapak Aljabar. Secara defenisi algoritma adalah alur pemikiran yang logis yang dapat dituangkan ke dalam bentuk tulisan (Antonius Rachmat C, 2010, 4). Sebuah algoritma dikatakan benar (*correct*) jika algoritma tersebut berhasil mengeluarkan output yang benar untuk semua kemungkinan input (Ihsan Dedy Boy Marpaung, 2013).

Pertimbangan dalam pemilihan algoritma adalah algoritma haruslah benar. Artinya algoritma akan memberikan keluaran yang dikehendaki dari sejumlah masukan yang diberikan. Tidak peduli serumit apapun algoritma, jika memberikan keluaran yang salah, pastilah algoritma tersebut bukanlah algoritma yang baik. Pertimbangan lain yang harus diperhatikan adalah seberapa baik hasil yang dicapai oleh algoritma tersebut. Hal ini penting terutama pada algoritma untuk menyelesaikan masalah yang memerlukan aproksimasi hasil. Algoritma yang baik harus mampu memberikan hasil yang sedekat mungkin dengan nilai yang sebenarnya.

Selanjutnya adalah efisiensi algoritma. Efisiensi algoritma dapat ditinjau dari 2 hal yaitu efisiensi waktu dan memori. Meskipun algoritma memberikan keluaran yang benar, tetapi jika harus menunggu lama untuk mendapatkan keluarannya, algoritma tersebut bukanlah algoritma yang baik, setiap orang menginginkan keluaran yang cepat. Dalam kenyataannya, setiap orang bisa membuat algoritma yang berbeda untuk menyelesaikan suatu permasalahan, walaupun terjadi perbedaan dalam menyusun algoritma, tentunya diharapkan keluaran yang sama (Diana Efendi, 2011).

II.4. Pengertian *Keylogger*

Keylogger merupakan sebuah perangkat baik perangkat keras atau perangkat lunak yang digunakan untuk memantau penekanan tombol *keyboard*. Sebuah *keylogger* biasanya akan menyimpan hasil pemantauan penekanan tombol *keyboard* tersebut ke dalam sebuah berkas log/catatan/rekaman. Beberapa *keylogger* tertentu bahkan dapat mengirimkan hasil rekamannya ke e-mail tertentu secara periodik (Binson Butar-Butar, 2015).

Keylogger dapat digunakan untuk kepentingan yang baik atau bahkan bisa digunakan untuk kepentingan yang jahat. Kepentingan yang baik antara lain untuk memantau produktivitas karyawan, untuk penegakan hukum dan pencarian bukti kejahatan. Kepentingan yang buruk antara lain pencurian data dan *password* (Binson Butar-Butar, 2015).

Penggunaan *keylogger* sangat banyak sekarang ini terutama untuk perbuatan yang tidak baik seperti pencurian *password* dan identitas lain, selain untuk perbuatan negatif *keylogger* juga bisa digunakan untuk memantau aktifitas pada komputer kantor yang dilakukan oleh pimpinan (Binson Butar-Butar, 2015).

II.4.1 Jenis *Keylogger*

Keylogger sendiri memiliki 2 jenis yaitu *software keylogger* dan *hardware keylogger*, berikut adalah penjelasannya:

1. *Software Keylogger*

Invisible keylogger, KGB *Keylogger* dan *Stealth Keylogger* adalah contoh *keylogger* berbentuk *software, software* ini bisa diinstal ke komputer korban dan secara otomatis, program ini akan menyembunyikan dirinya sehingga tidak diketahui oleh pengguna.

Korban tidak akan bisa melihat program ini sedang berjalan karena semua *software* menawarkan modus sembunyi yang tidak akan menampilkan *icon*, nama program di task manager dan lainnya.

Software keylogger mempunyai beberapa keunggulan seperti banyaknya program yang telah tersedia dan anda tinggal menggunakannya, anda juga tidak perlu mempunyai akses fisik ke komputer yang menjadi sasaran.

2. *Hardware Keylogger*

Selain *software keylogger* terdapat satu jenis lagi yaitu *hardware keylogger*, *hardware keylogger* berupa *keylogger* berbentuk *keylogger* dan untuk memasang *keylogger* ini anda memerlukan akses fisik ke komputer yang hendak dipasang.

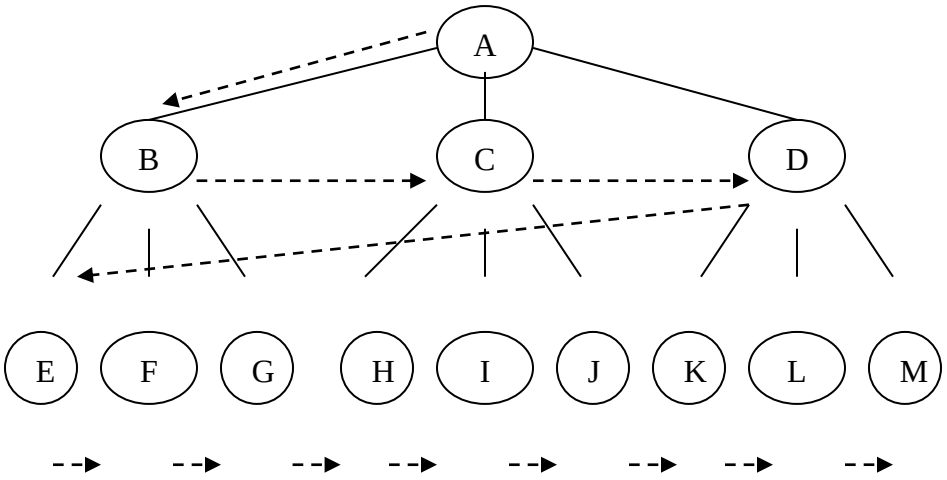
(*Sumber : http://www.howtogeek.com/180615/keyloggers-explained-what-you-need-to-know/)*

II.5. Metode *String Matching*

String Matching adalah algoritma yang melakukan pencarian secara berurut yang mengunjungi simpul secara presedor yaitu mengunjungi suatu simpul kemudian mengunjungi semua simpul yang bertetangga dengan simpul tersebut terlebih dahulu (Binson Butar-Butar, 2015).

Pencarian dilakukan pada semua simpul dalam setiap level secara berurutan. Jika pada satu level belum ditemukan solusi, maka pencarian dilanjutkan pada level berikutnya. Demikian seterusnya sampai ditemukan solusi. Dengan cara seperti ini, *String Matching* menjamin ditemukannya solusi (jika solusinya memang ada) dan solusi yang ditemukan pasti yang paling baik dikarenakan solusi sudah ada dan tinggal dicocokkan saja. Dengan kata lain, *String Matching* adalah *complete* dan optimal. Tetapi, *String Matching* harus menyimpan semua simpul

yang pernah dibangkitkan. Hal ini harus dilakukan agar *String Matching* dapat melakukan penelusuran simpul-simpul sampai di level bawah. Jika b adalah faktor percabangan (jumlah simpul anak yang dimiliki oleh suatu simpul) dan d adalah kedalaman solusi, maka jumlah simpul yang harus disimpan adalah sebanyak $O(b^d)$. Misalkan untuk b = 10 dan d = 8, maka BFS harus membangkitkan dan menyimpan sebanyak $10^0 + 10^1 + 10^2 + 10^3 + 10^4 + 10^5 + 10^6 + 10^7 + 10^8 = 111.111.111 \approx 10^8$ simpul. Jika diasumsikan bahwa dalam satu detik komputer bisa membangkitkan dan menguji 10^6 simpul, maka waktu proses yang diperlukan untuk menemukan solusi di level 8 adalah 100 detik (1,67 menit). Jika satu simpul direpresentasikan dalam struktur data sebesar 100 *bytes*, maka diperlukan memori sebesar 10^{10} *bytes* (atau 10 *gigabytes*). Dari segi kecepatan, hal ini mungkin masih bisa diterima. Tetapi dari sisi memori yang diperlukan, ini menjadi masalah yang serius. Dengan permasalahan dan komputer yang sama, waktu proses yang diperlukan untuk menemukan solusi di level 14 adalah 10^8 detik (lebih dari 3 tahun), dan diperlukan memori sebesar 10^{15} *bytes* (1000 *terabytes*) (Binson Butar-Butar, 2015).



Gambar II.1 *Metode String Matching*
Sumber : Binson Butar-Butar, 2015

II.6. *Unified Modelling Language*

Unified Modelling Language (UML) adalah suatu alat untuk memvisualisasikan dan mendokumentasikan hasil analisa dan desain yang berisi sintak dalam memodelkan sistem secara

visual dan juga merupakan satu kumpulan konvensi pemodelan yang digunakan untuk menentukan atau menggambarkan sebuah sistem *software* yang terkait dengan objek (Haviluddin, 2011:1).

Sejarah UML sendiri terbagi dalam dua *fase*; sebelum dan sesudah munculnya UML. Dalam fase sebelum, UML sebenarnya sudah mulai diperkenalkan sejak tahun 1990an namun notasi yang dikembangkan oleh para ahli analisis dan desain berbeda-beda, sehingga dapat dikatakan belum memiliki standarisasi. Fase kedua; dilandasi dengan pemikiran untuk mempersatukan metode tersebut dan dimotori oleh *Object Management Group* (OMG) maka pengembangan UML dimulai pada akhir tahun 1994 ketika Grady Booch dengan metode OOD (*Object-Oriented Design*), Jim Rumbaugh dengan metode OMT (*Object Modelling Technique*) mereka ini bekerja pada *Rasional Software Corporation* dan Ivar Jacobson dengan metode OOSE (*Object-Oriented Software Engineering*) yang bekerja pada perusahaan *Objectory Rasional*. Sebagai pencetus metode-metode tersebut mereka bertiga berinisiatif untuk menciptakan bahasa pemodelan terpadu sehingga pada tahun 1996 mereka berhasil merilis UML versi 0.9 dan 0.91 melalui *Request for Proposal* (RFP) yang dikeluarkan oleh OMG (Braun, et.al. 2001) (Haviluddin, 2011:1).

Kemudian pada Januari 1997 IBM, *ObjecTime, Platinum Technology, Ptech, Taskon, Reich Technologies* dan *Softeam* juga menanggapi *Request for* Proposal (RFP) yang dikeluarkan oleh OMG tersebut dan menyatakan kesediaan untuk bergabung. Perusahaan-perusahaan ini menyumbangkan ide-ide mereka, dan bersama para mitra menghasilkan UML revisi 1.1. Fokus dari UML versi rilis 1.1 ini adalah untuk meningkatkan kejelasan UML Semantik versi rilis 1.0. Hingga saat ini UML versi terbaru adalah versi 2.0 (http://www.uml.org/).

Saat ini sebagian besar para perancang sistem informasi dalam menggambarkan informasi dengan memanfaatkan UML diagram dengan tujuan utama untuk membantu tim proyek berkomunikasi, mengeksplorasi potensi desain, dan memvalidasi desain arsitektur perangkat lunak atau pembuat program (Haviluddin, 2011:1)..

Secara filosofi UML diilhami oleh konsep yang telah ada yaitu konsep permodelan Object Oriented karena konsep ini menganalogikan sistem seperti kehidupan nyata yang didominasi oleh obyek dan digambarkan atau dinotasikan dalam simbol-simbol yang cukup spesifik (Haviluddin, 2011:1-2).

II.6.1 Tujuan Penggunaan UML

Tujuan dari penggunaan diagram seperti diungkapan oleh Schmuller J. (2004), "*The purpose of the diagrams is to present multiple views of a system; this set of multiple views is called a model*". Berikut tujuan utama dalam desain UML adalah (Sugrue J. 2009) :

1. Menyediakan bagi pengguna (analisis dan desain sistem) suatu bahasa pemodelan visual yang ekspresif sehingga mereka dapat mengembangkan dan melakukan pertukaran model data yang bermakna.
2. Menyediakan mekanisme yang spesialisasi untuk memperluas konsep inti.
3. Karena merupakan bahasa pemodelan visual dalam proses pembangunannya maka UML bersifat independen terhadap bahasa pemrograman tertentu.
4. Memberikan dasar formal untuk pemahaman bahasa pemodelan.
5. Mendorong pertumbuhan pasar terhadap penggunaan alat desain sistem yang berorientasi objek (OO).
6. Mendukung konsep pembangunan tingkat yang lebih tinggi seperti kolaborasi, kerangka, pola dan komponen terhadap suatu sistem.

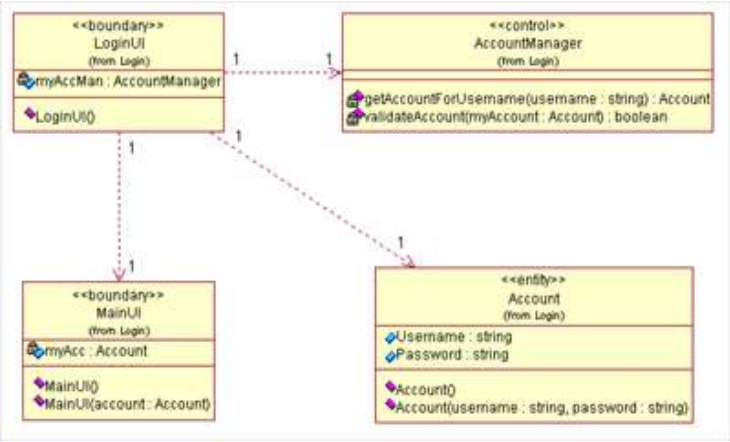
7. Memiliki integrasi praktik terbaik. (Haviluddin, 2011:2).

II.6.2 Komponen UML

Sejauh ini para pakar merasa lebih mudah dalam menganalisa dan mendesain atau memodelkan suatu sistem karena UML memiliki seperangkat aturan dan notasi dalam bentuk grafis yang cukup spesifik (Haviluddin, 2011:3). Komponen atau notasi UML diturunkan dari 3 (tiga) notasi yang telah ada sebelumnya yaitu *Grady Booch*, OOD (*Object-Oriented Design*), Jim Rumbaugh, OMT (*Object Modelling Technique*), dan Ivar Jacobson OOSE (*Object-Oriented Software Engineering*). Pada UML versi 2 terdiri atas tiga kategori dan memiliki 13 jenis diagram (Haviluddin, 2011:3), yaitu :

1. *Class Diagram*

   Class diagram menggambarkan struktur statis dari kelas dalam sistem anda dan menggambarkan atribut, operasi dan hubungan antara kelas. *Class diagram* membantu dalam memvisualisasikan struktur kelas-kelas dari suatu sistem dan merupakan tipe diagram yang paling banyak dipakai. Selama tahap desain, *class diagram* berperan dalam menangkap struktur dari semua kelas yang membentuk arsitektur sistem yang dibuat (Haviluddin, 2011:3), berikut contohnya



Gambar II.2 *Class Diagram*

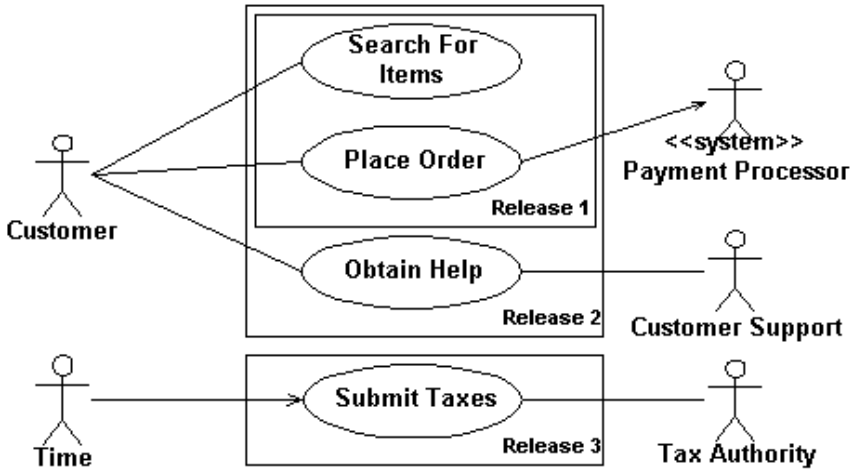
Sumber: Haviluddin, 2011:3

2. *Use Case Diagram*

Diagram yang menggambarkan *actor*, *use case* dan relasinya sebagai suatu urutan tindakan yang memberikan nilai terukur untuk aktor. Sebuah *use case* digambarkan sebagai *elips horizontal* dalam suatu diagram UML *use case*. *Use Case* memiliki dua istilah (Haviluddin, 2011:3)

a. *System use case*; interaksi dengan sistem.

b. *Business use case*; interaksi bisnis dengan konsumen atau kejadian nyata

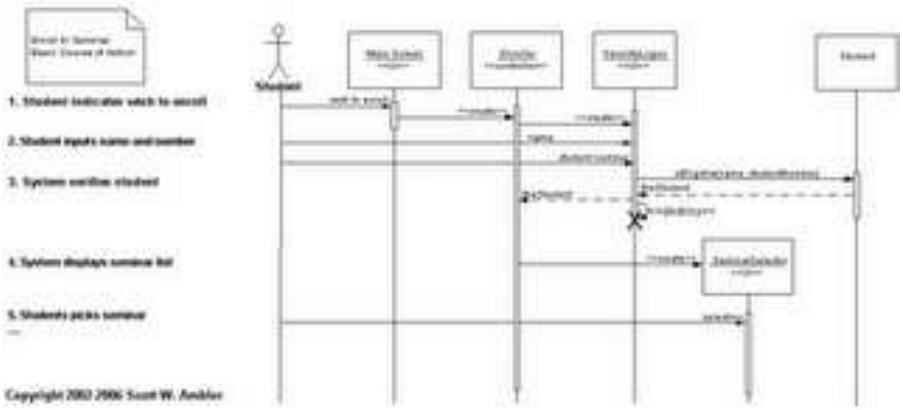


Gambar II.3 *Use Case Diagram*

Sumber: Haviluddin, 2011:4

3. *Sequence Diagram*

Sequence diagram menjelaskan interaksi objek yang disusun berdasarkan urutan waktu. Secara mudahnya *sequence* diagram adalah gambaran tahap demi tahap, termasuk kronologi (urutan) perubahan secara logis yang seharusnya dilakukan untuk menghasilkan sesuatu sesuai dengan *use case diagram* (Haviluddin, 2011:4)

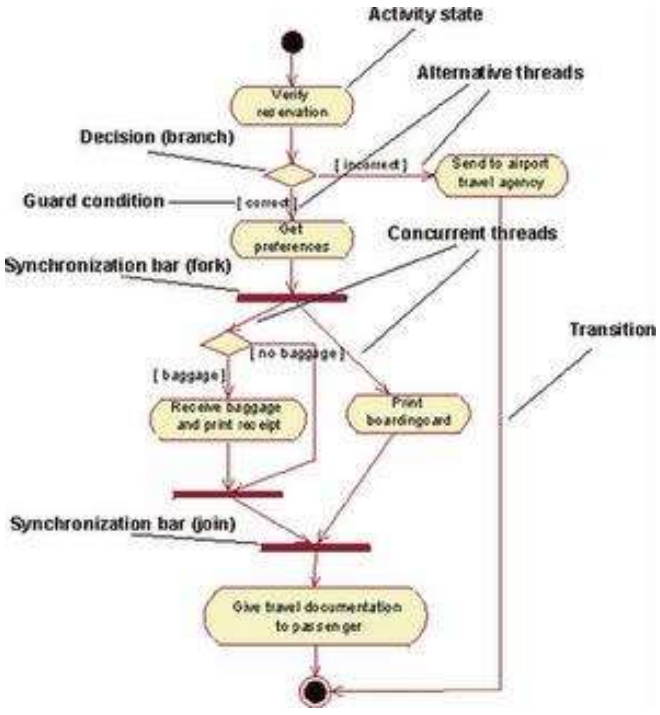


Gambar II.4 *Sequence Case Diagram*

Sumber: Haviluddin, 2011:4

4. *Activity Diagram*

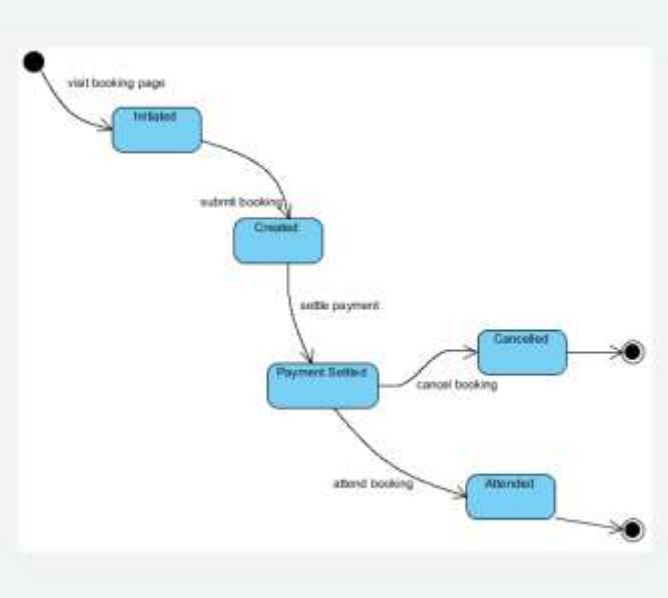
Menggambarkan aktifitas-aktifitas, objek, *state*, transisi *state* dan *event*. Dengan kata lain kegiatan diagram alur kerja menggambarkan perilaku sistem untuk aktivitas (Haviluddin, 2011:4)



Gambar II.5 *Activity Diagram*
Sumber: Haviluddin, 2011:4

5. *State Chart Diagram*

   Menggambarkan *state*, transisi *state dan event*.



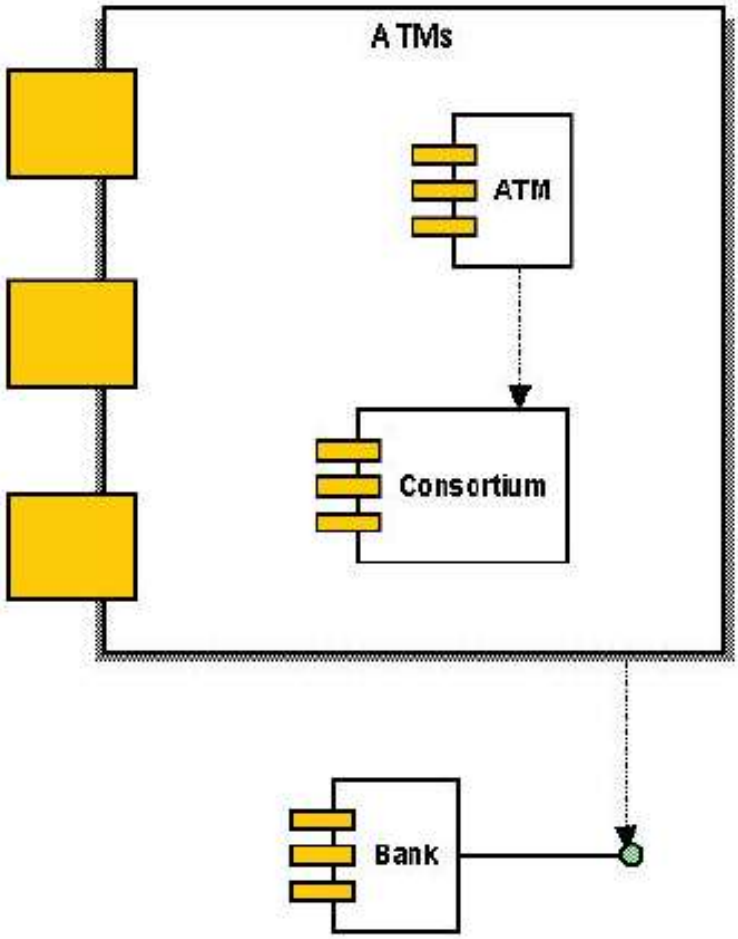
Gambar II.6 *State Chart Diagram*

Sumber: Haviluddin, 2011:4

6. *Component Diagram*

   Component diagram menggambarkan struktur fisik dari kode, pemetaan

   pandangan logis dari kelas proyek untuk kode aktual di mana logika ini dilaksanakan.

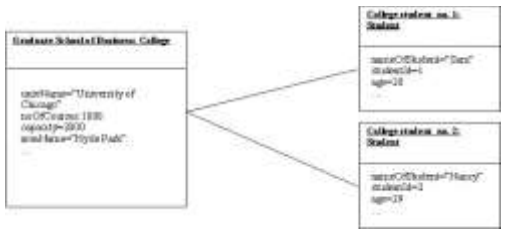


Gambar II.7 *Component Diagram*

Sumber: Haviluddin, 2011:4

7. *Object Diagram*

Object diagram menggambarkan kejelasan kelas dan warisan dan kadang-kadang diambil ketika merencanakan kelas, atau untuk membantu pemangku kepentingan non-program yang mungkin menemukan diagram kelas terlalu abstrak. Berikut notasi *object diagram*.

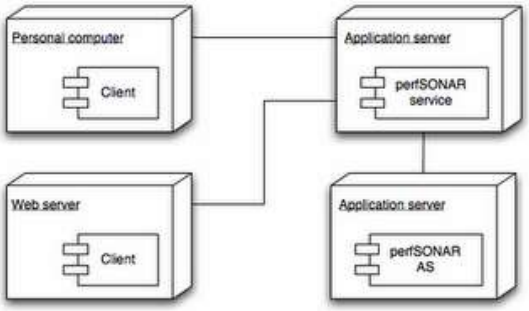


Gambar II.8 *Object Diagram*

Sumber: Haviluddin, 2011:4

8. Deployment Diagram

Deployment diagram memberikan gambaran dari arsitektur fisik perangkat lunak, perangkat keras, dan artefak dari sistem. *Deployment diagram* dapat dianggap sebagai ujung spektrum dari kasus penggunaan, menggambarkan bentuk fisik dari sistem yang bertentangan dengan gambar konseptual dari pengguna dan perangkat berinteraksi dengan sistem.

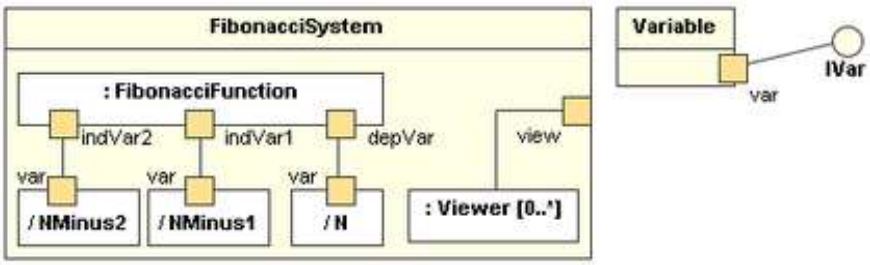


Gambar II.9 *Deployment Diagram*

Sumber: Haviluddin, 2011:4

9. Composite structure diagram

Sebuah diagram struktur komposit mirip dengan diagram kelas, tetapi menggambarkan bagian individu, bukan seluruh kelas. Kita dapat menambahkan konektor untuk menghubungkan dua atau lebih bagian dalam atau ketergantungan hubungan asosiasi.

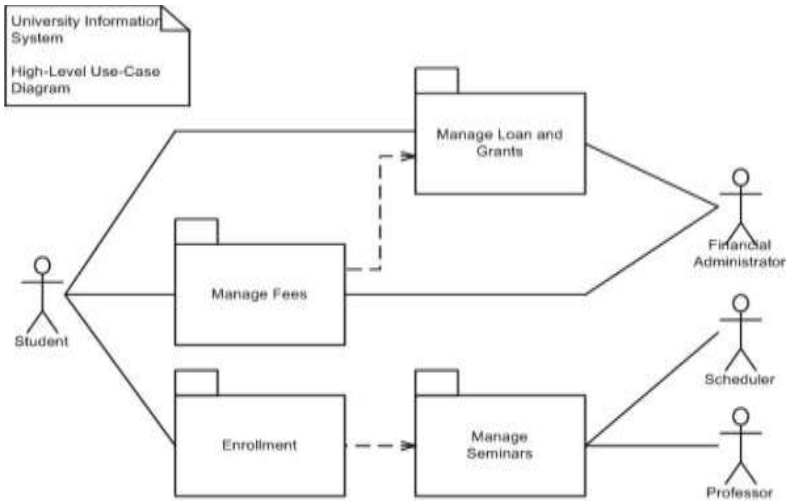


Gambar II.10 *Composite Structure Diagram*

Sumber: Haviluddin, 2011:4

10. Package Diagram

Paket diagram biasanya digunakan untuk menggambarkan tingkat organisasi yang tinggi dari suatu proyek software. Atau dengan kata lain untuk menghasilkan diagram ketergantungan paket untuk setiap paket dalam Pohon Model.

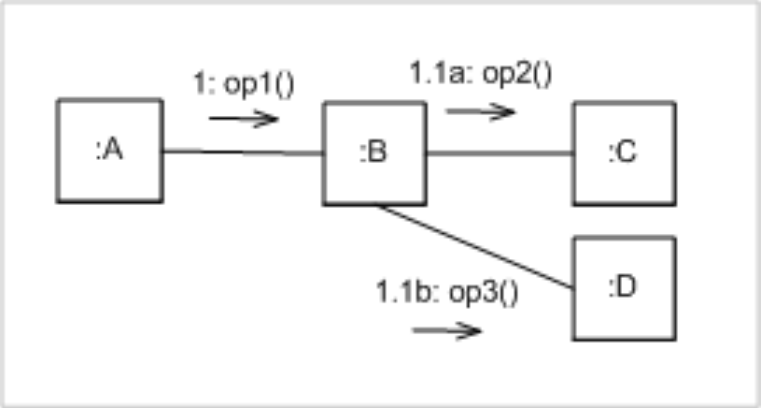


Gambar II.11 *Package Diagram*

Sumber: Haviluddin, 2011:4

11. Communication diagram

Serupa dengan *sequence diagram*, tetapi diagram komunikasi juga digunakan untuk memodelkan perilaku dinamis dari *use case*. Bila dibandingkan dengan *Sequence diagram*, diagram komunikasi lebih terfokus pada menampilkan kolaborasi benda daripada urutan waktu.

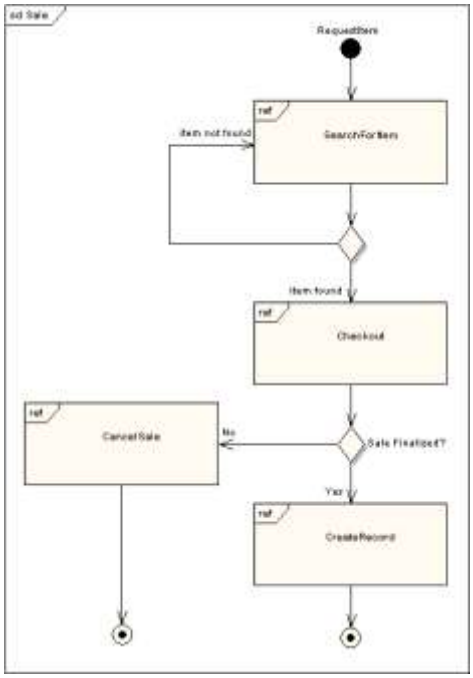


Gambar II.12 ***Communication Diagram***

Sumber: Haviluddin, 2011:5

12. Interaction Overview diagram

Interaksi overview diagram berfokus pada gambaran aliran kendali interaksi dimana node adalah interaksi atau kejadian interaksi.

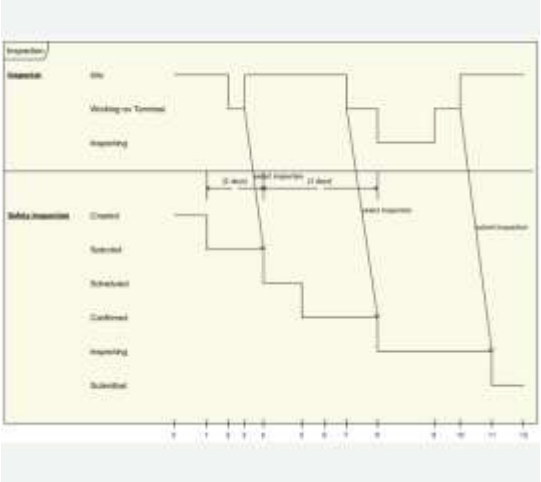


Gambar II.13 ***Interaction Overview Diagram***

Sumber: Haviluddin, 2011:5

13. Timing Diagram

Timing diagram di UML didasarkan pada diagram waktu *hardware* awalnya dikembangkan oleh para insinyur listrik.



Gambar II.14 ***Interaction Overview Diagram***

Sumber: Haviluddin, 2011:5

II.7. *Zoho Mail*

Zoho Mail merupakan sebuah *mail client* yang telah dirancang untuk memecahkan gagasan bahwa hanya *client* email desktop dapat menyediakan fitur yang disukai oleh pengguna bisnis. fungsionalitas kuat dan handal ditempatkan di setiap fitur untuk tetap mempertahankan kehandalan yang ramping, tampilan menarik dan memudahkan pengguna untuk menggunakannya dibandingkan dengan mail *client* lainnya (*https://www.zoho.com/mail/advanced-features.html*).

II.8. *Visual Basic.Net*

Microsoft Visual Basic .NET adalah sebuah alat untuk mengembangkan dan membangun aplikasi yang bergerak di atas sistem *.NET Framework,* dengan menggunakan bahasa *BASIC*. Dengan menggunakan alat ini, para *programmer* dapat membangun aplikasi *Windows Forms,*

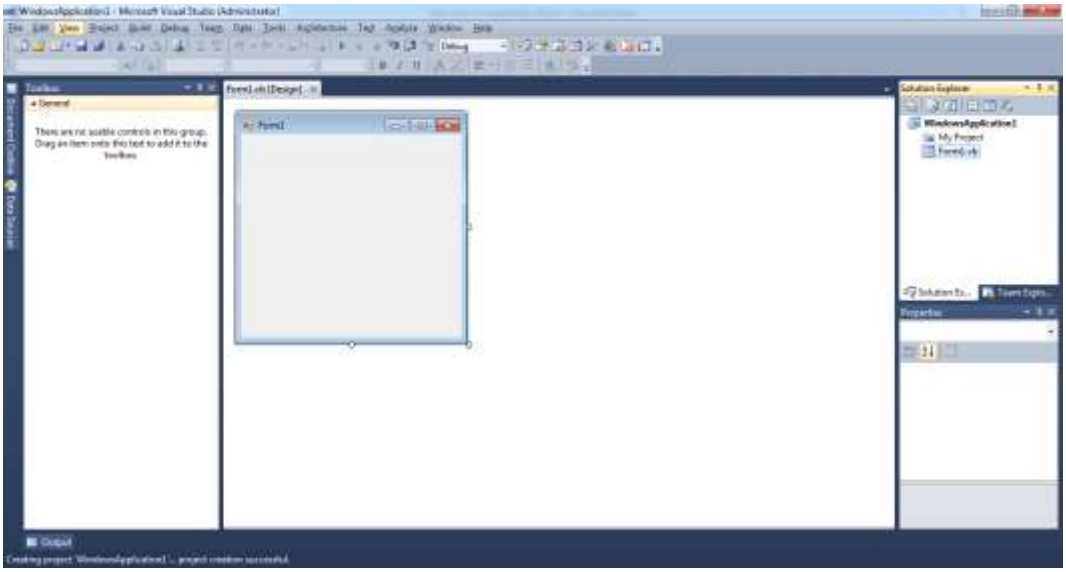
Aplikasi *web berbasis ASP.NET*, dan juga aplikasi *command-line*. Alat ini dapat diperoleh secara terpisah dari beberapa produk lainnya (seperti *Microsoft Visual C++, Visual C#, atau Visual J#*), atau juga dapat diperoleh secara terpadu dalam *Microsoft Visual Studio .NET*. Bahasa *Visual Basic .NET* sendiri menganut paradigma bahasa pemrograman berorientasi objek yang dapat dilihat sebagai evolusi dari *Microsoft Visual Basic* versi sebelumnya yang diimplementasikan di atas *.NET Framework*. Peluncurannya mengundang kontroversi, mengingat banyak sekali perubahan yang dilakukan oleh *Microsoft*, dan versi baru ini tidak kompatibel dengan versi terdahulu.

Apakah *Visual Basic .NET* dianggap sebagai sebuah versi *Visual Basic* atau benar-benar bahasa yang berbeda merupakan sebuah topik perdebatan yang hangat. Hal ini dikarenakan sintaksis bahasa *Visual Basic .NET* tidak mengalami perubahan yang sangat drastis, dan hanya menambahkan beberapa dukungan fitur baru seperti penanganan eksepsi secara terstruktur dan ekspresi yang bisa di-short-circuit-kan. Dua perubahan tipe data pun terjadi saat berpindah ke *Visual Basic .NET*. Dibandingkan dengan *Visual Basic* 6.0, tipe data *Integer* yang dimiliki oleh *Visual Basic .NET* memiliki panjang dua kali lebih panjang, dari 16 bit menjadi 32 bit. Selain itu, tipe data *Long* juga sama-sama berubah menjadi dua kali lipat lebih panjang, dari 32 bit menjadi 64 bit. Bilangan bulat 16-bit dalam *Visual Basic .NET* dinamakan dengan *Short*. Lagi pula, *desainer GUI Windows Forms* yang terdapat di dalam *Visual Studio .NET* atau *Visual Basic .NET* memiliki gaya yang sangat mirip dengan editor *form Visual Basic clasik.*

Jika sintaksis tidak banyak yang berubah, lain halnya dengan semantik, yang berubah secara signifikan. *Visual Basic .NET* merupakan sebuah bahasa pemrograman yang mendukung fitur "Bahasa Pemrograman Berorientasi Objek" secara penuh, karena memang didukung oleh arsitektur *Microsoft .NET Framework*, yang mengandung kombinasi dari *Common Language*

Runtime dan *Base Class Library*. *Visual Basic clasik*, hanya merupakan sebuah bahasa pemrogaman berbasis objek, yang berjalan di atas arsitektur *Component Object Model* (COM).

Perubahan ini telah mengubah banyak asumsi tentang hal yang benar yang harus dilakukan dengan mempertimbangkan performa dan kemudahan untuk dipelihara. Beberapa fungsi dan pustaka perangkat lunak, yang ada di dalam *Visual Basic clasik*, kini tidak terdapat di dalam *Visual Basic .NET*, mungkin masih banyak yang masih terdapat di dalam *Visual Basic .NET*, tapi tidak seefisien apa yang ditawarkan oleh *.NET Framework*. Bahkan jika program *Visual Basic clasik* bisa dikompilasi dengan benar, sebagian besar program *Visual Basic clasik* harus melalui beberapa proses *refactoring* untuk mengadopsi fitur bahasa baru secara keseluruhan. Dokumentasi untuk ini pun tersedia di situs *Microsoft*



Gambar II.15. *Visual Basic.Net*

Sumber : Wahana Komputer, 2012