Memahami Penggunaan UML (*Unified Modelling Language)*

HAVILUDDIN

Program Studi Ilmu Komputer, FMIPA Universitas Mulawarman
Jl. Barong Tongkok No. 5 Kampus Unmul Gn. Kelua Sempaja Samarinda 75119

Abstrak
Unified Modelling Language merupakan alat perancangan sistem yang berorientasi pada objek. Secara filosofi kemunculan UML diilhami oleh konsep yang telah ada yaitu konsep permodelan *Object Oriented* (OO), karena konsep ini menganalogikan sistem seperti kehidupan nyata yang didominasi oleh obyek dan digambarkan atau dinotasikan dalam simbol-simbol yang cukup spesifik maka OO memiliki proses standard dan bersifat independen.

UML diagram memiliki tujuan utama untuk membantu tim pengembangan proyek berkomunikasi, mengeksplorasi potensi desain, dan memvalidasi desain arsitektur perangkat lunak atau pembuat program. 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*).

UML mempunyai tiga kategori utama yaitu struktur diagram, *behaviour* diagram dan *interaction* diagram. Dimana masing-masing kategori tersebut memiliki diagram yang menjelaskan arsitektur sistem dan saling terintegrasi.

Kata Kunci : OOP, UML, OOD, OMT, OOSE, OMG

PENDAHULUAN

Unified Modelling Language (UML) adalah suatu alat untuk memvisualisasikan dan mendokumentasikan hasil analisa dan desain yang berisi sintak dalam memodelkan sistem secara visual (Braun, *et. al*. 2001). Juga merupakan satu kumpulan konvensi pemodelan yang digunakan untuk menentukan atau menggambarkan sebuah sistem *software* yang terkait dengan objek (Whitten, *et. al.* 2004).

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).

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.

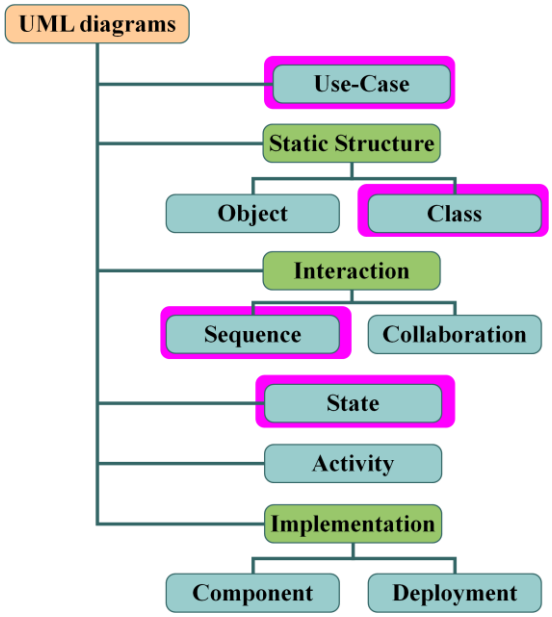
Perusahaan-perusahaan ini menyumbangkan ide-ide mereka, dan bersama para mitra menghasilkan UML revisi 1.1. Fokus dari UML versi rilis 1.1 ini adalah untuk meningkatkan kejelasan UML Semantik versi rilis 1.0. Hingga saat ini UML versi terbaru adalah versi 2.0 (http://www.uml.org/).

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.

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.

Berikut gambar dari diagram UML



Gambar 1. Diagram UML
(Sumber : http://www.uml.org/).

Tujuan Pemanfaatan 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.

***Object Oriented Program* (OOP)**

Object Oriented Program (OOP) merupakan paradigma baru dalam rekayasa *software* yang didasarkan pada obyek dan kelas. (Ronald J.N., 1996). Diakui para ahli bahwa *object-oriented* merupakan metodologi terbaik yang ada saat ini dalam rekayasa *software*. *Object-oriented* memandang *software* bagian per bagian dan menggambarkan satu bagian tersebut dalam satu obyek.

Teknologi obyek menganalogikan sistem aplikasi seperti kehidupan nyata yang didominasi oleh obyek. Dengan demikian keunggulan teknologi obyek adalah bahwa model yang dibuat akan sangat mendekati dunia nyata yang masalahnya akan dipecahkan oleh sistem yang dibangun. Model obyek, atribut dan perlakuannya bisa langsung diambil dari obyek yang ada di dunia nyata.

Ada 4 (empat) prinsip dasar dari pemrograman berorientasi obyek yang menjadi dasar kemunculan UML, yaitu abstraksi, enkapsulasi, modularitas dan hirarki. Berikut dijelaskan satu persatu secara singkat.

1. Abstraksi memfokuskan perhatian pada karakteristik obyek yang paling penting dan paling dominan yang bisa digunakan untuk membedakan obyek tersebut dari obyek lainnya.
2. Enkapsulasi menyembunyikan banyak hal yang terdapat dalam obyek yang tidak perlu diketahui oleh obyek lain. Dalam praktek pemrograman, enkapsulasi diwujudkan dengan membuat suatu kelas *interface* yang akan dipanggil oleh obyek lain, sementara didalam obyek yang dipanggil terdapat kelas lain yang mengimplementasikan apa yang terdapat dalam kelas *interface*.
3. Modularitas membagi sistem yang rumit menjadi bagian-bagian yang lebih kecil yang bisa mempermudah *developer* memahami dan mengelola obyek tersebut.
4. Hirarki berhubungan dengan abstraksi dan modularitas, yaitu pembagian berdasarkan urutan dan pengelompokkan tertentu. Misalnya untuk menentukan obyek mana yang berada pada kelompok yang sama, obyek mana yang merupakan komponen dari obyek yang memiliki hirarki lebih tinggi. Semakin rendah hirarki obyek berarti semakin jauh abstraksi dilakukan terhadap suatu obyek.

Komponen-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 (Sugrue J. 2009).

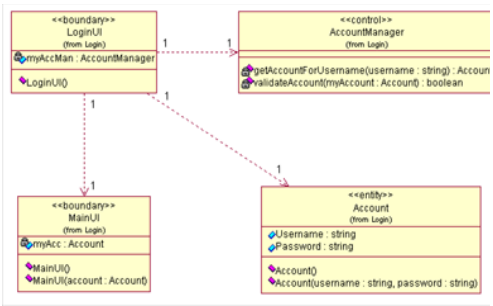
Komponen atau notasi UML diturunkan dari 3 (tiga) notasi yang telah ada sebelumnya yaitu Grady Booch, OOD (*Object-Oriented Design*), Jim Rumbaugh, OMT (*Object Modelling Technique*), dan Ivar Jacobson OOSE (*Object-Oriented Software Engineering*).

Pada UML versi 2 terdiri atas tiga kategori dan memiliki 13 jenis diagram yaitu :

1. Struktur Diagram

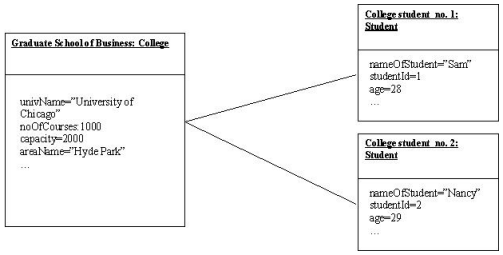
Menggambarkan elemen dari spesifikasi dimulai dengan kelas, obyek, dan hubungan mereka, dan beralih ke dokumen arsitektur logis dari suatu sistem. Struktur diagram dalam UML terdiri atas :

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.
Class memiliki tiga area pokok :
1. Nama (dan *stereotype*)
2. Atribut
3. Metoda
.

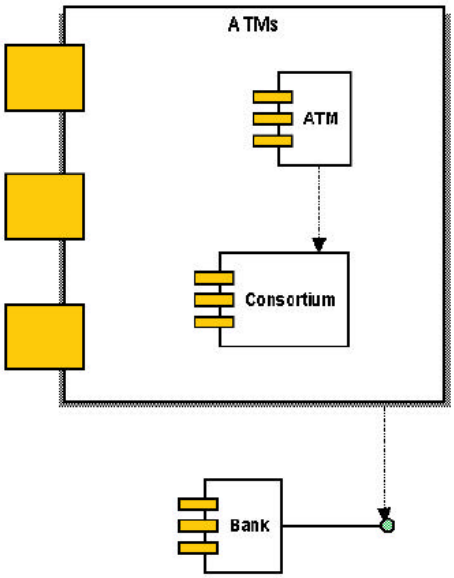


Gambar 2. Notasi *class diagram*

2) *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*.

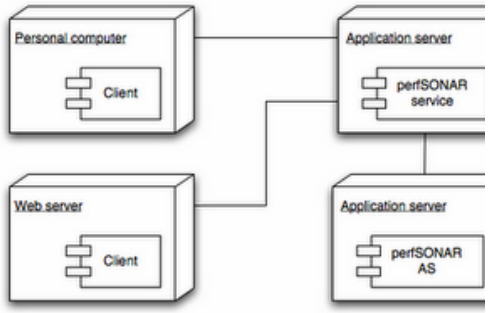


3) *Component* diagram
Component diagram menggambarkan struktur fisik dari kode, pemetaan pandangan logis dari kelas proyek untuk kode aktual di mana logika ini dilaksanakan.



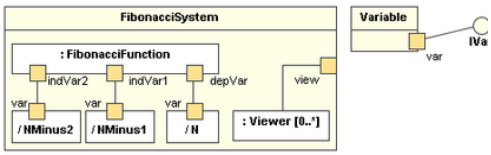
Gambar 3. Notasi *component diagram*.

4) *Deployment* diagram (*Collaboration* diagram in version 1.x)
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.



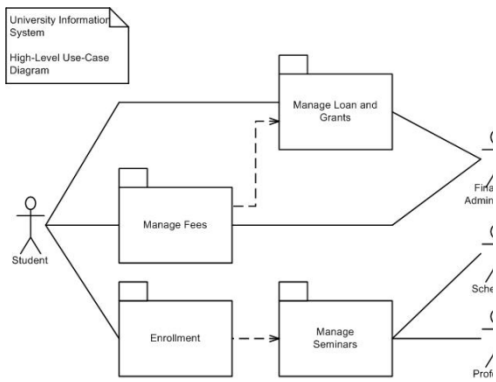
Gambar 4. Notasi *deployment* diagram.

5) *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 5. Notasi *composite diagram*.

6) *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.

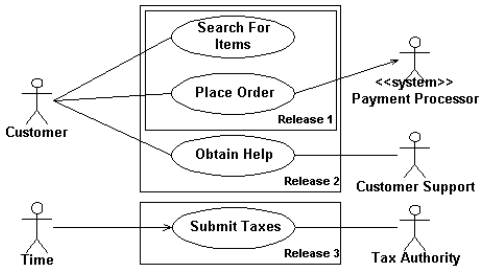


Gambar 6. Notasi *package diagram*.

2. Behavior Diagram

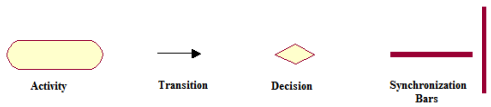
Menggambarkan ciri-ciri *behavior*/metode/ fungsi dari sebuah sistem atau *business process*. *Behavior* diagram dalam UML terdiri atas :

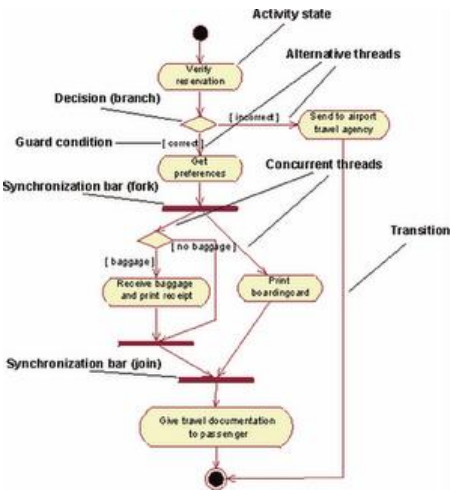
7) *Use case* diagram
Diagram yang menggambarkan *actor*, *use case* dan relasinya sebagai suatu urutan tindakan yang memberikan nilai terukur untuk aktor. Sebuah *use case* digambarkan sebagai elips horizontal dalam suatu diagram UML *use case*.
Use Case memiliki dua istilah
1. *System use case*; interaksi dengan sistem.
2. *Business use case;* interaksi bisnis dengan konsumen atau kejadian nyata



Gambar 7. Notasi *use case diagram*.

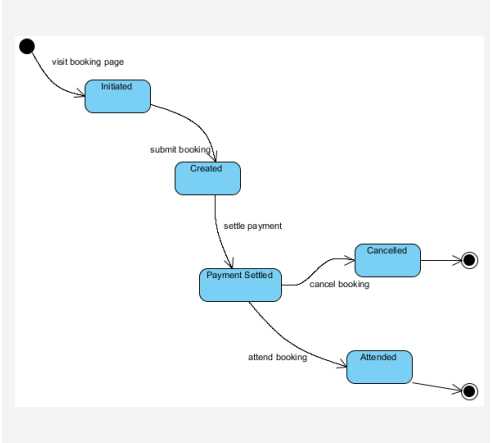
8) *Activity* diagram
Menggambarkan aktifitas-aktifitas, objek, *state*, transisi *state* dan *event*. Dengan kata lain kegiatan diagram alur kerja menggambarkan perilaku sistem untuk aktivitas





Gambar 8. Notasi *activity* diagram.

9) *State Machine* diagram (*State chart* diagram in version 1.x)
Menggambarkan *state*, transisi *state* dan *event*.

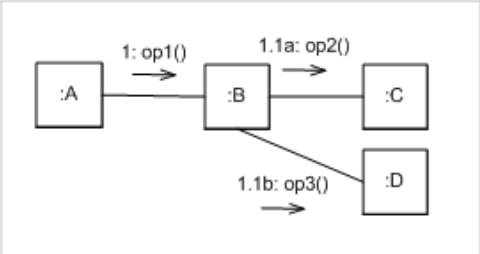


Gambar 9. Notasi *state machine* diagram.

3. Interaction diagram

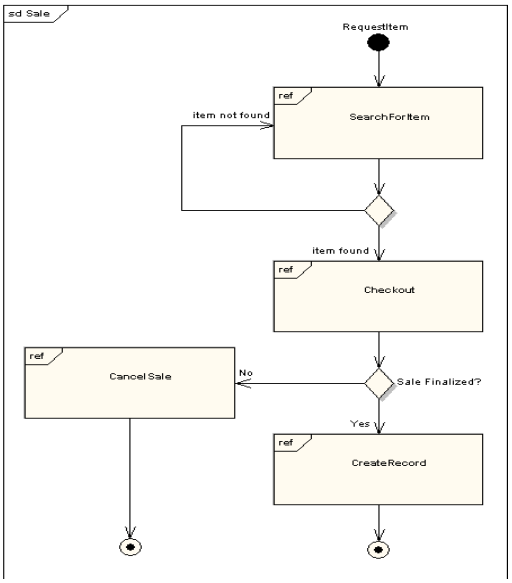
Bagian dari *behavior* diagram yang menggambarkan interaksi objek. *Interaction* diagram dalam UML terdiri atas :

10) *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.



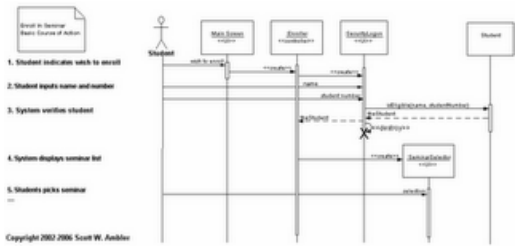
Gambar 10. Notasi *communication* diagram.

11) *Interaction Overview* diagram
Interaksi overview diagram berfokus pada gambaran aliran kendali interaksi dimana node adalah interaksi atau kejadian interaksi.



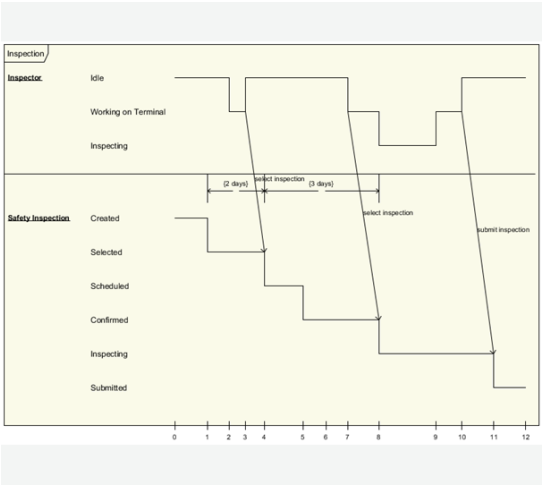
Gambar 11. notasi overview diagram.

12) *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



Gambar 12. Notasi *sequence* diagram

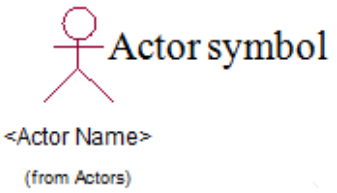
13) Timing diagram
Timing diagram di UML didasarkan pada diagram waktu *hardware* awalnya dikembangkan oleh para insinyur listrik.



Gambar 13. Notasi *timing* diagram.

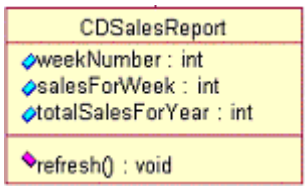
Untuk menggambarkan analisa dan desain diagram, UML memiliki seperangkat notasi yang akan digunakan ke dalam tiga kategori diatas yaitu struktur diagram, *behaviour* diagram dan *interaction* diagram. Berikut beberapa notasi dalam UML diantaranya :

1. *Actor*; menentukan peran yang dimainkan oleh *user* atau sistem lain yang berinteraksi dengan subjek. *Actor* adalah segala sesuatu yang berinteraksi langsung dengan sistem aplikasi komputer, seperti orang, benda atau lainnya. Tugas *actor* adalah memberikan informasi kepada sistem dan dapat memerintahkan sistem untuk melakukan sesuatu tugas.



Gambar 14. Notasi *actor*

2. *Class diagram*; Notasi utama dan yang paling mendasar pada diagram UML adalah notasi untuk mempresentasikan suatu *class* beserta dengan atribut dan operasinya. *Class* adalah pembentuk utama dari sistem berorientasi objek.

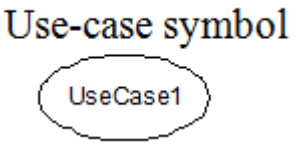


Gambar 15. Notasi *class*

3. *Use Case* dan *use case specification*; *Use case* adalah deskripsi fungsi dari sebuah sistem perspektif pengguna. *Use case* bekerja dengan cara mendeskripsikan tipikal interaksi antara *user* (pengguna) sebuah sistem dengan sistemnya sendiri melalui sebuah cerita bagaimana sebuah sistem dipakai. Urutan langkah-langkah yang menerangkan antara pengguna dan sistem disebut skenario.

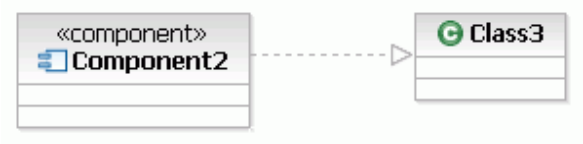
Use case merupakan awal yang sangat baik untuk setiap fase pengembangan berbasis objek, design, testing, dan dokumentasi yang menggambarkan kebutuhan sistem dari sudut pandang di luar sistem.

Perlu diingat bahwa *use case* hanya menetapkan apa yang seharusnya dikerjakan oleh sistem, yaitu kebutuhan fungsional sistem dan tidak untuk menentukan kebutuhan non-fungsional, misalnya: sasaran kinerja, bahasa pemrograman dan lain sebagainya.



Gambar 16. Notasi *use case*

4. *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.

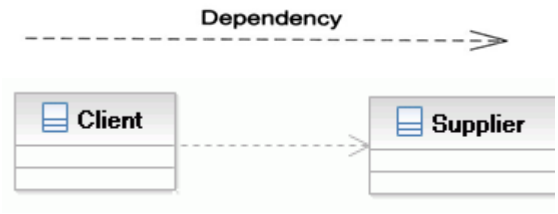


Gambar 17. Notasi *realization*

5. *Interaction*; *Interaction* digunakan untuk menunjukkan baik aliran pesan atau informasi antar obyek maupun hubungan antar obyek.

Gambar 18. Notasi *interaction*

6. *Dependency*; *Dependency* merupakan relasi yang menunjukan bahwa perubahan pada salah satu elemen memberi pengaruh pada elemen lain. 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 ke dalam elemen yang ada di garis dengan panah.

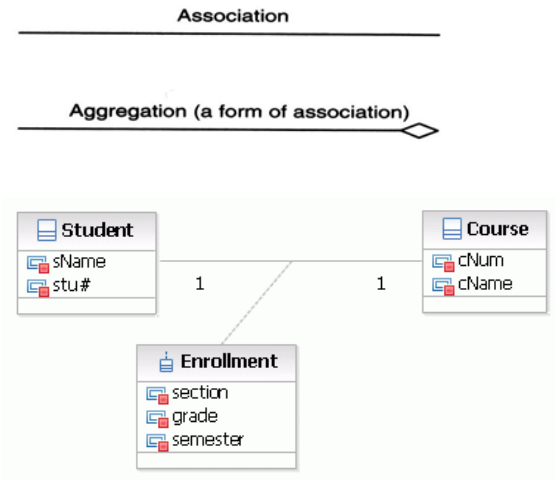


Gambar 19. Notasi *dependency*

7. *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.

Gambar 20. Notasi *note*

8. *Association*; *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*).

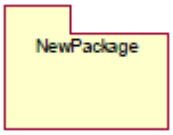


Gambar 21. Notasi *association*

9. *Generalization*; *Generalization* menunjukkan hubungan antara elemen yang lebih umum ke elemen yang lebih spesifik.

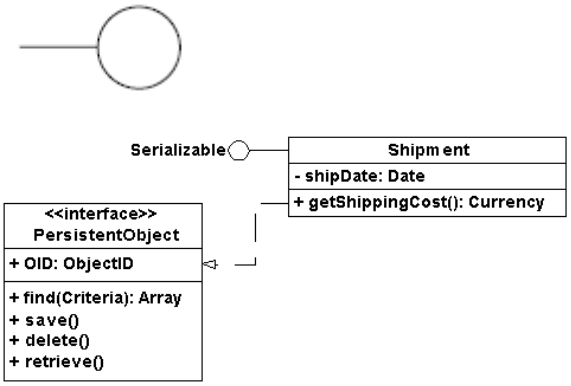
Gambar 22. Notasi *generalization*

10. *Package; package* adalah mekanisme pengelompokkan yang dgunakan untuk menandakan pengelompokkan elemen-elemen model.



Gambar 23. Notasi *package*

11. *Interface*; *Interface* merupakan kumpulan operasi berupa implementasi dari suatu *class*. Atau dengan kata lain implementasi operasi dalam *interface* dijabarkan oleh operasi di dalam *class*.



Gambar 24. Notasi *interface*

Beberapa *tools* bantu UML yang berbayar dan *open source* :

Kelompok berbayar

1. AndroMDA (http://www.andromda.org) versi *licence*.
2. AgroUML (http://argouml.tigris.org/) versi *licence*.

Kelompok *open source*

1. AgroUML (http://argouml.tigris.org/)
2. AndroMDA (http://www.andromda.org)
3. UMLGraph (http://www.umlgraph.org)
4. StarUML (http://staruml.sourceforge.net/en/)
5. UniMod (http://java-source.net/open-source/uml-modeling/unimod)
6. Alma (http://java-source.net/open-source/uml-modeling/alma)
7. UMLet (http://java-source.net/open-source/uml-modeling/umlet)
8. UML/Dot (http://java-source.net/open-source/uml-modeling/uml-dot)
9. JUG (http://java-source.net/open-source/uml-modeling/jug)
10. Violet (http://java-source.net/open-source/uml-modeling/violet)
11. MetaBoss (http://java-source.net/open-source/uml-modeling/metaboss)
12. fujaba (http://java-source.net/open-source/uml-modeling/fujaba)

13. Linguine Maps (http://java-source.net/open-source/uml-modeling/linguine-maps)
14. Visual Paradigm for UML 8.1 (http://www.visual-paradigm.com/product/vpuml/)

PROSES MEMBANGUN UML

Langkah-Langkah Penggunaan UML

Batasan sistem harus ditentukan terlebih dahulu, tujuannya agar pemakai mengetahui dengan lingkungan mana saja sistem mereka berhubungan, untuk itu setiap komponen *actor*, (sumber atau tujuan) ini harus diberi nama sesuai dengan lingkungan luar yang mempengaruhi sistem ini.

Berikut ini tahapan penggunaan UML (Dharwiyanti. 2008) :

1. Buatlah daftar *business process* dari level tertinggi untuk mendefinisikan aktivitas dan proses yang mungkin muncul dengan menentukan item-item data apa saja yang akan ditempatkan dalam sistem.
2. Petakan *use case* untuk tiap *business process* untuk mendefinisikan dengan tepat fungsionalitas yang harus disediakan oleh sistem. Kemudian perhalus *use case* diagram dan lengkapi dengan *requirement*, *constraints* dan catatan-catatan lain.
3. Buatlah *deployment diagram* secara kasar untuk mendefinisikan arsitektur fisik sistem.
4. Definisikan *requirement* lain (non-fungsional, *security* dan sebagainya) yang juga harus disediakan oleh sistem.
5. Berdasarkan *use case diagram*, mulailah membuat *activity diagram*.
6. Definisikan objek-objek level atas (*package* atau *domain*) dan buatlah *sequence* dan/atau *collaboration diagram* untuk tiap alir pekerjaan. Jika sebuah *use case* memiliki kemungkinan alir normal dan *error*, buatlah satu diagram untuk masing-masing alir.
7. Buatlah rancangan *user interface* model yang menyediakan antarmuka bagi pengguna untuk menjalankan skenario *use case*.
8. Berdasarkan model-model yang sudah ada, buatlah *class diagram*. Setiap *package* atau *domain* dipecah menjadi hirarki *class* lengkap dengan atribut dan metodenya. Akan lebih baik jika untuk setiap *class* dibuat *unit test* untuk menguji fungsionalitas *class* dan interaksi dengan *class* lain.
9. Setelah *class diagram* dibuat, kita dapat melihat kemungkinan pengelompokan *class* menjadi komponen-komponen. Karena itu buatlah *component* diagram pada tahap ini. Juga, definisikan tes integrasi untuk setiap komponen meyakinkan ia berinteraksi dengan baik.
10. Perhalus *deployment diagram* yang sudah dibuat. Detilkan kemampuan dan *requirement* piranti lunak, sistem operasi, jaringan, dan sebagainya. Petakan komponen ke dalam *node*.
11. Mulailah membangun sistem. Ada dua pendekatan yang dapat digunakan :
    - Pendekatan *use case*, dengan meng-*assign* setiap *use case* kepada tim pengembang tertentu untuk mengembangkan *unit code* yang lengkap dengan tes.
    - Pendekatan komponen, yaitu meng-*assign* setiap komponen kepada tim pengembang tertentu.
12. Lakukan uji modul dan uji integrasi serta perbaiki model berserta *code*-nya. Model harus selalu sesuai dengan *code* yang aktual.
13. Piranti lunak siap dirilis.

MENGUKUR VALIDITAS UML

Hal yang terpenting dalam membangun UML adalah validitas data yang terjamin. Untuk itu perlu lebih mengetahui seluk beluk permasalahan sistem secara spesifik.

Menurut Gordon B. Davis nilai informasi dikatakan sempurna apabila perbedaan antara kebijakan optimal, tanpa informasi yang sempurna dan kebijakan optimal menggunakan informasi yang sempurna dapat dinyatakan dengan jelas dalam bentuk notasi-notasi.

Tidak kalah pentingnya faktor *gathering* data dimana muara suatu analisa dikerjakan. Ada banyak metode yang digunakan dalam mendapatkan data tersebut seperti menggunakan metode TOE, yaitu sebagai berikut:

1. T artinya *Technical*, maksudnya bagaimana tata cara/teknik pembuatan *requirement* tersebut dalam sistem yang diusulkan.
2. O artinya Operational, maksudnya bagaimana tata cara penggunaan *requirement* tersebut dalam sistem yang akan dikembangkan.
3. E artinya *Economy*, maksudnya berapakah biaya yang diperlukan guna membangun *requirement* tersebut di dalam sistem.

KEUNTUNGAN DAN KELEMAHAN UML

Unified Modeling Language (*UML*) merupakan alat bantu, bahasa pemodelan yang dapat digunakan untuk rancang bangun berorientsi-

objek. *UML* dapat digunakan untuk spesifikasi, visualisasi dan dokumentasi sistem pada *fase* pengembangan (Eriksson dan Penker,1998).

Keuntungan

Karena merupakan bahasa pemodelan visual dalam proses pembangunannya maka UML bersifat independen terhadap bahasa pemrograman tertentu.

Taylor (1992) menyatakan bahwa membangun *software* menggunakan pendekatan teknologi objek memberikan beberapa keuntungan, antara lain: memungkinkan penggunaan kembali objek yang ada (*reusable*), memungkinkan *software* yang baru dengan konstruksi yang lebih besar, *software* berorientasi objek secara umum lebih mudah dimodifikasi dan dirawat karena sebuah objek dapat dimodifikasi tanpa banyak berpengaruh pada objek yang lain.

Kelemahan

UML dipandang masih mempunyai kekurangan terutama dalam meng-*generate* kode program secara komplit. Hal ini karena kurangnya cara memodelkan aspek kelakuan *internal* perangkat lunak untuk dipetakan ke dalam kode program.

Seperti yang kita ketahui, diagram UML yang dapat menghasilkan kode hanyalah *diagram class*, namun itupun hanya sebatas kerangka kodenya saja dan tidak bisa meng-*generate* badan program-nya.

Perbedaan DFD (*Data Flow Diagram*) dan UML

UML biasa digunakan untuk mempresentasikan sistem kepada orang-orang yang tidak mengerti tata-cara pemrograman, seperti orang-orang awam pada umumnya.

Hal ini dikarenakan UML memakai penggambaran alur sistem dan logika algoritma suatu program. Sedangkan DFD (*Data Flow Diagram*) kebalikannya, biasa digunakan untuk mempresentasikan sistem kepada orang-orang yang mengerti tata cara pemrograman (*programmer*, analis programmer).

Hal ini dikarenakan DFD memakai penggambaran sistem secara umum. Dari alur memperoleh data, proses data, basis data, dan entitas.

KESIMPULAN

UML merupakan penggabungan konsep dari Grady Booch dengan metode OOD (*Object-Oriented Design*), Jim Rumbaugh dengan metode OMT (*Object Modelling Technique*) dan Ivar Jacobson dengan metode OOSE (*Object-Oriented Software Engineering*), sehingga UML merupakan suatu bahasa pemodelan tunggal yang umum dan dapat digunakan secara luas oleh para user ketiga metode tersebut dan bahkan para *user* metode lainnya.

Penekanan pada UML adalah pada apa yang dapat dikerjakan dengan metode-metode tersebut.

UML berfokus pada suatu bahasa pemodelan standar, bahkan pada proses standar dan bersifat independen.

DAFTAR PUSTAKA

Bernd Oestereich. 1999. *Developing Software with UML*. Addison-Wesley

Boogs W., and Boogs M. 2002. *Mastering UML with Rational Rose 2002.* SYBEX Inc., 1151 Marina Village Parkway, Alameda, CA 94501.

Braun D., Sivils J., Shapiro A., Versteegh J. 2001. *Object Oriented Analysis and Design Team*. Kennesaw State University CSIS 4650 - Spring 2001

Bruce E. Wampler, Ph.D. 2003. *The Essence of Object Oriented Programming with Java and UML***.** Addison-Wesley.

Eriksson H-E and Penker M. 1998. *UML Toolkit*, John Wiley & Son Inc.

Gordon B. Davis. 1998. *Kerangka dasar Sistem Informasi Manajemen: Pengantar Seri Manajemen No: 90. A*, PT. Pustaka Binaman Pressindo.

Gordon B. Davis. **1999.** *A Research Perspective for Information Systems and Example of Emerging Area of Research.* Information Systems Frontiers 1(3): 195-203 (1999)

Grady Booch, Ivar Jacobson, James Rumbaugh. 1996. *The Unified Modelling Language for Object-Oriented Development; Documentation Set Version 0.91Addendum UML Update*. Copyright ©1996 Rational Software Corporation URL: http://www.rational.com

Hans-Erik Eriksson, "*UML2 Toolkit*", 2004 www.OMG.org."UML *2.0 Superstructure Specification*",2004

Jeffrey L. Whitten, Lonnie D. Bentley, Kevin C. Dittman,. 2004. "*System Analysis and Design Methods", 5th edition*, McGraw-Hill.

Kunle Odutola, Anthony Oguntimehin, Linus Tolke, Michiel van der Wulp. 2008. *ArgoUML Quick Guide Get started with ArgoUML 0.30*

Martin Fowler, Kendall Scott. 2000. *UML Distilled.* Addison Wesley.

Nurokhim dan Ratnasari Nur Rohmah. 2002. *Case Tool* Pengembangan Perangkat Lunak Berorientasi-objek menggunakan *Unified Modeling Language* (*UML*). JURNAL TEKNIK ELEKTRO EMITOR Vol. 2, No. 1, Maret 2002

Ronald J. Norman. 1996. "*Object Oriented Systems Analysis and Design*",Prentice Hall.

Schmuller J. 2004. *Sams Teach Yourself UML in 24 Hours, Third Edition.* Sams Publishing. http://www.omg.org

Scott W. Ambler,"*The elements of UML style*",Cambridge University Press 2003

Sugrue J. 2009. *Getting Started with UML.* http://www.dzone.com/links/index.html

Watson A. 2009. *Visual Modelling: past, present and future.* Whitepaper. http://www.uml.org/Visual_Modeling.pdf.Vice-President and Technical Director Object Management Group™

http://argouml.tigris.org/

http://argouml-stats.tigris.org/documentation/

http://blog.syabac.web.id/2009/03/27/unified-modeling-language-uml-diagrams.html

http://publib.boulder.ibm.com/infocenter/rsmhelp/v7r0m0/index.jsp?topic=/com.ibm.xtools.modeler.doc/topics/cdepend.html

http://umlforstudents.blogspot.com/

http://www.agilemodeling.com/artifacts/communicationDiagram.htm

http://www.agilemodeling.com/artifacts/interactionOverviewDiagram.htm

http://www.agilemodeling.com/artifacts/useCaseDiagram.htm

http://www.agilemodeling.com

http://www.altova.com/umodel/object-diagrams.html

http://www.dzone.com

http://www.kumaira.co.cc/2009/11/diagram-diagram-uml.html

http://www.modelingstyle.org

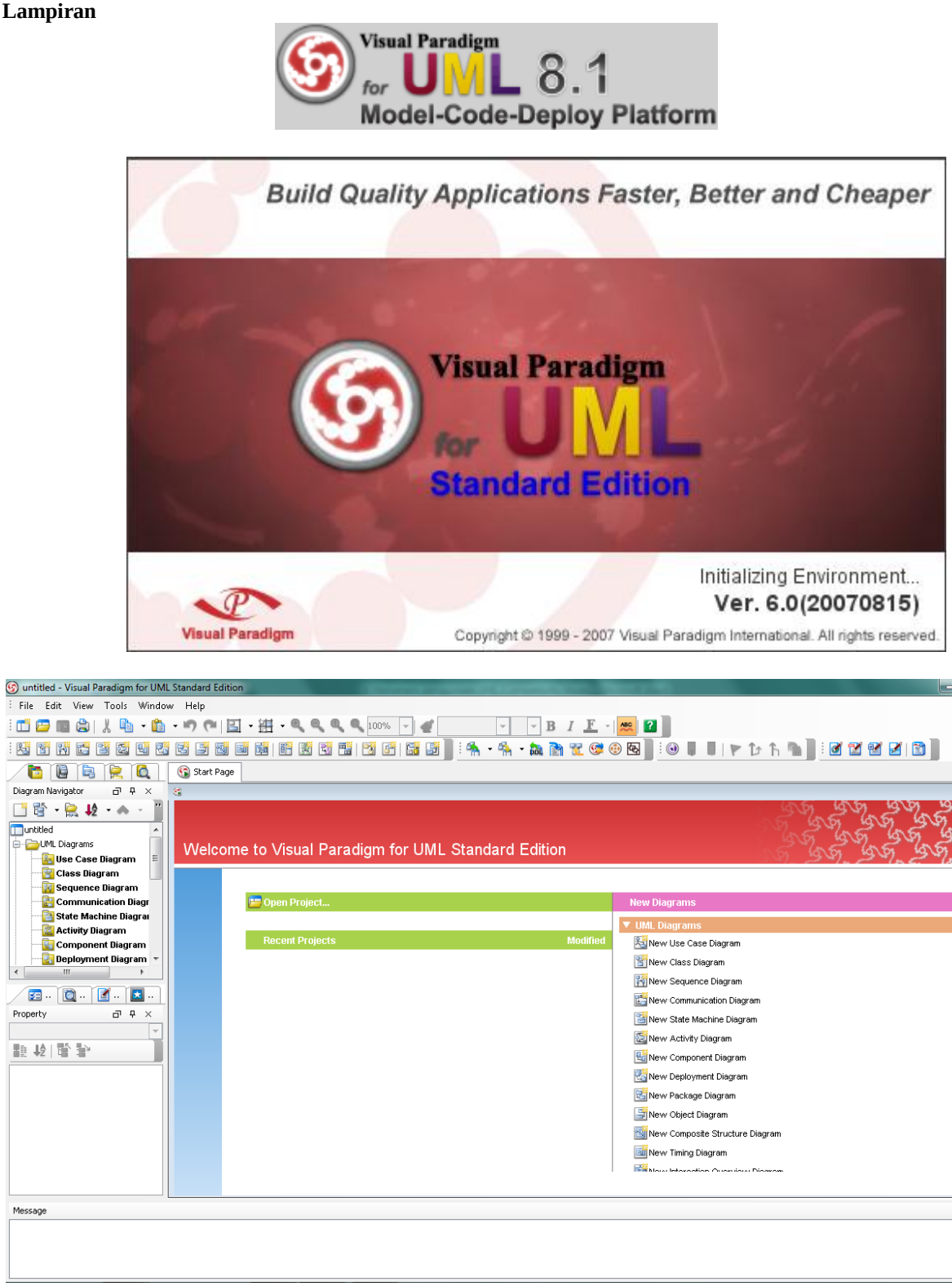
http://www.OMG.org

http://www.rational.com/

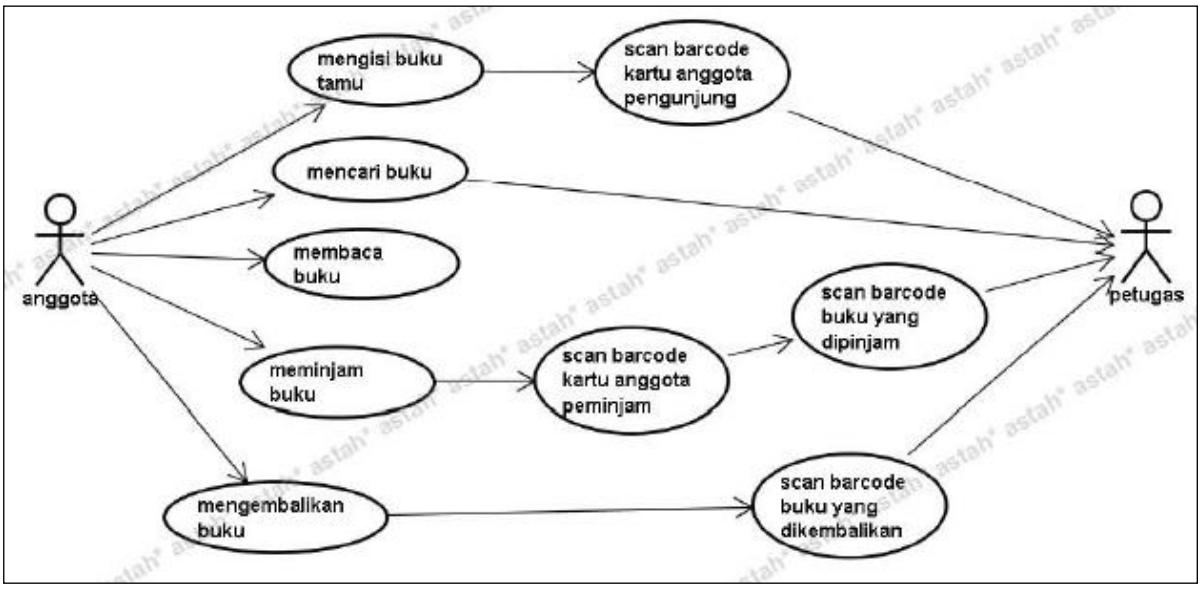
http://www.opencontent.org/openpub/

http://www.visual-paradigm.com/VPGallery/diagrams/TimingDiagram.html

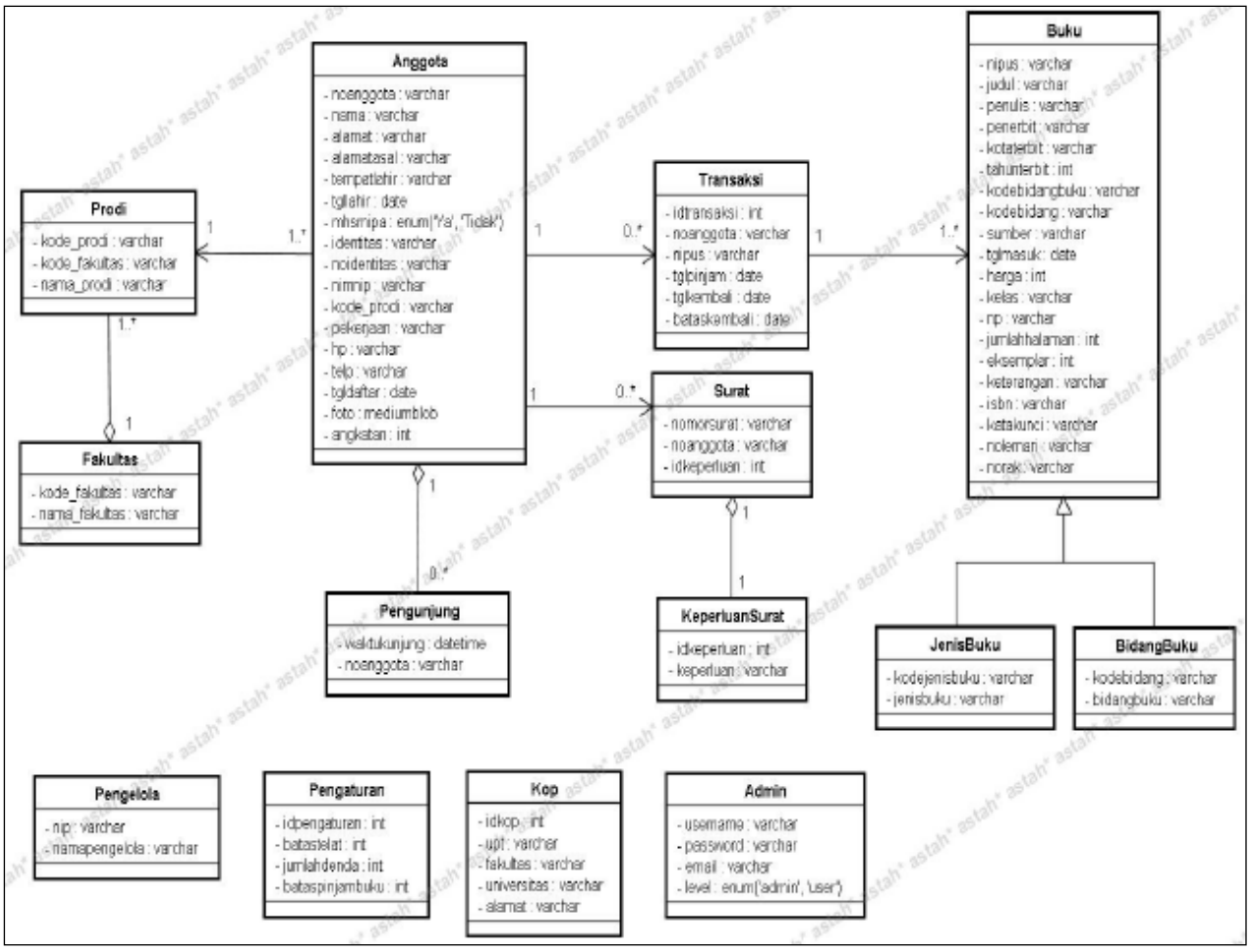
Lampiran



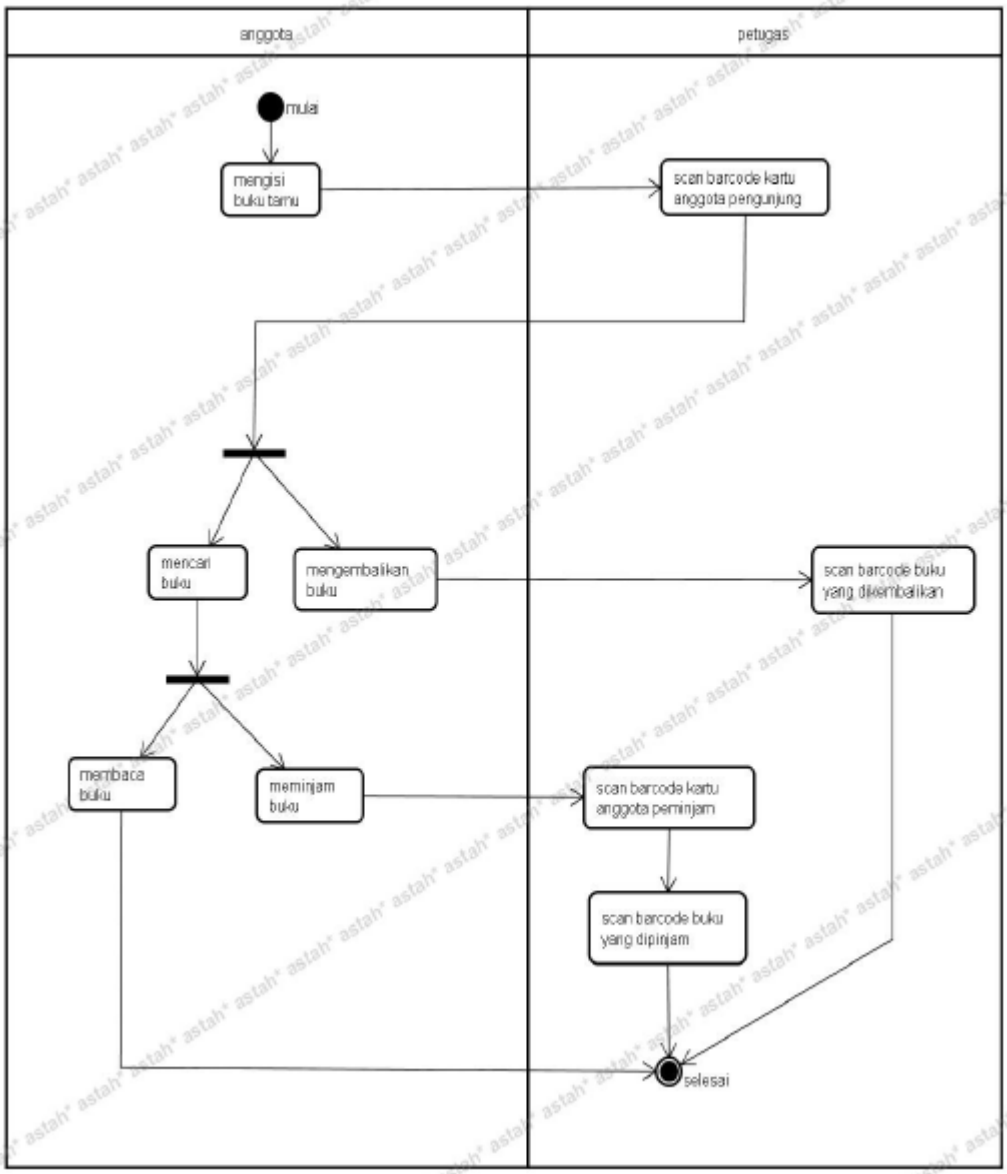
Gambar 26. Contoh pemanfaatan UML dalam kasus sistem informasi perpustakaan dengan menggunakan software Visual Paradigm for UML Standard Edition Ver. 6.0(20070815)



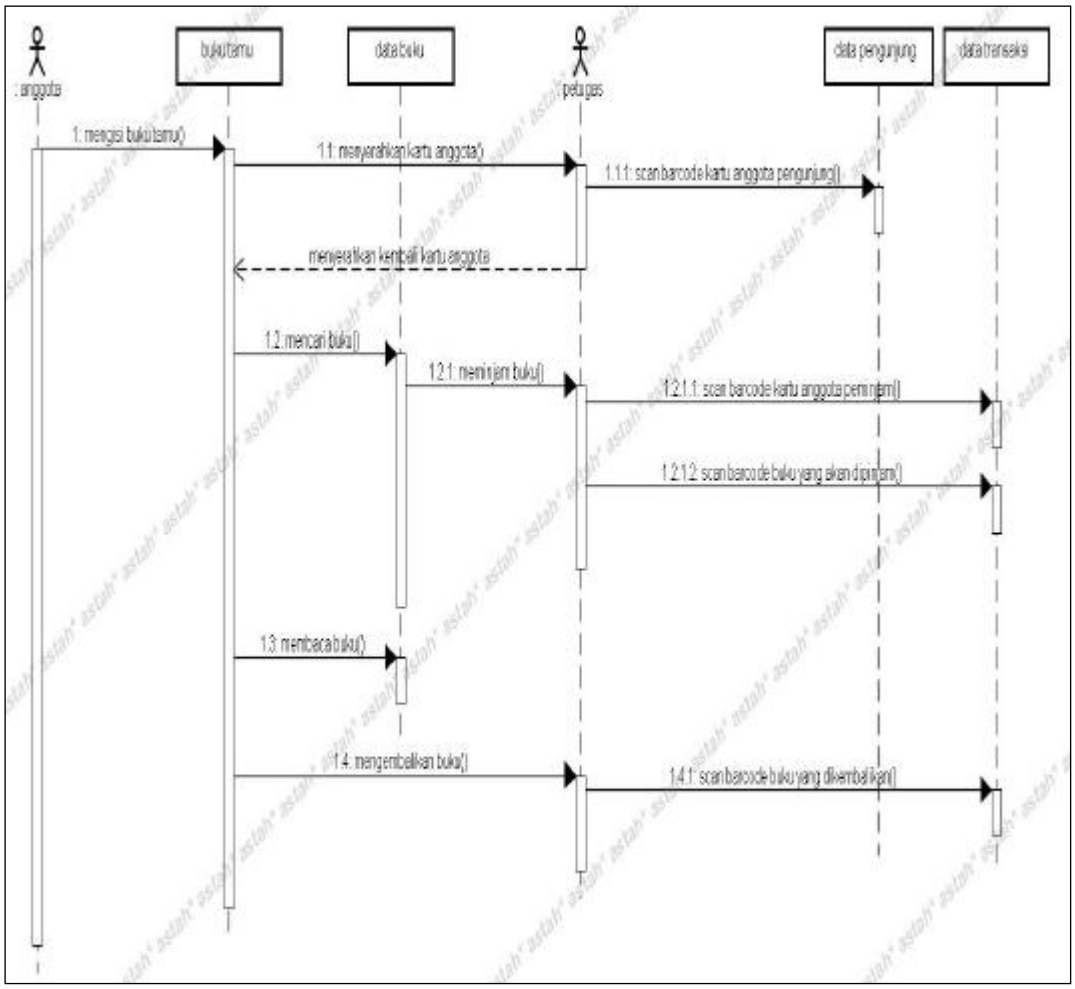
Gambar 27. Contoh *use case diagram* Sistem Informasi Perpustakaan



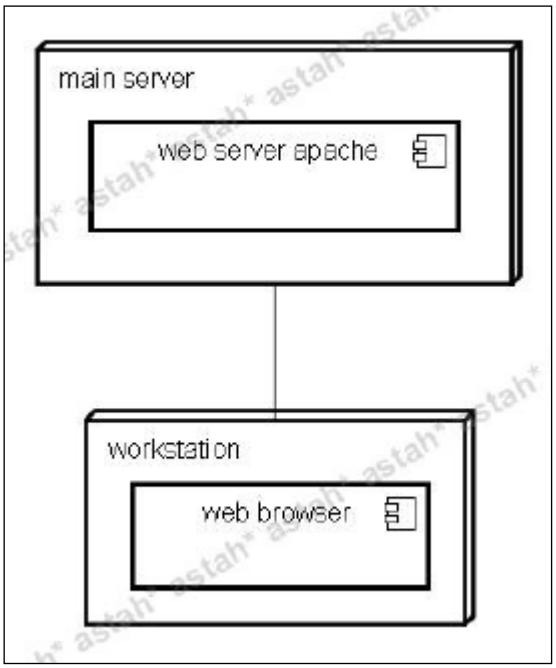
Gambar 28. Contoh *class diagram* Sistem Informasi Perpustakaan



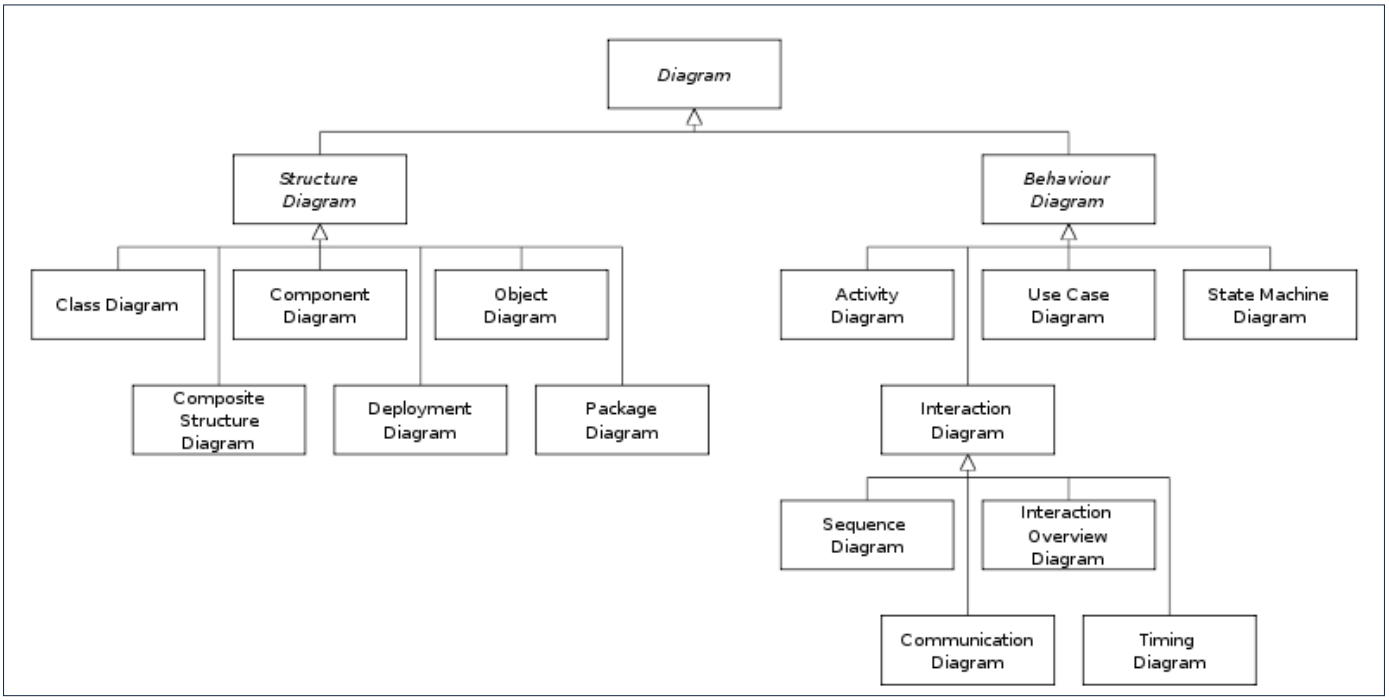
Gambar 29. Contoh *activity diagram* Sistem Informasi Perpustakaan



Gambar 30. Contoh *sequence diagram* Sistem Informasi Perpustakaan



Gambar 31. Contoh *deployment diagram* Sistem Informasi Perpustakaan



Gambar 32. Struktur pembentuk diagram UML