23.442/H/05

MILIK PERPUSTAKAAN
INSTITUT TEKNOLOGI
SEPULUH - NOPEMBER

PEMODELAN PERANGKAT LUNAK BERORIENTASI ASPEK DALAM NOTASI UML, STUDI KASUS PADA SISTEM KEAMANAN PERANGKAT LUNAK

TUGAS AKHIR

R514
005.1
Kus
p-1
2005

Disusun Oleh :

Siti Kustiarsih
5101 100 036

PERPUSTAKAAN
ITS

Tgl. Terima	5-8-2005
Terima Dari	H
No. Agenda Prp.	222896

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2005

PEMODELAN PERANGKAT LUNAK BERORIENTASI ASPEK DALAM NOTASI UML, STUDI KASUS PADA SISTEM KEAMANAN PERANGKAT LUNAK

TUGAS AKHIR

Diajukan untuk Memenuhi Sebagian Persyaratan

Memperoleh Gelar Sarjana Komputer

Pada

Jurusan Teknik Informatika

Fakultas Teknologi Informasi

Institut Teknologi Sepuluh Nopember

Surabaya

Mengetahui/Menyetujui,

Dosen Pembimbing I

Ir. Siti Rochimah, MT

NIP. 132 103 631

SURABAYA

JULI 2005

ABSTRAK

Grace Copy Center

ABSTRAK

Teknik pembangunan perangkat lunak selalu mengalami perkembangan seiring dengan kemajuan teknologi. Mulai dari teknik pembangunan terstruktur, pembangunan berorientasi objek, dan sekarang yang baru adalah pembangunan berorientasi aspek. Tahapan dalam pembangunan perangkat lunak berorientasi aspek yang sekarang banyak dibahas baru pada tingkat pemrogramannya saja, sedangkan tahap pemodelan belum banyak diteliti dan didiskusikan. Notasi pemodelan untuk pendekatan aspek ini belum ada, khususnya pada tahap analisis dan perancangan.

Karakteristik permasalahan yang sering dihadapi dalam proses pembangunan perangkat lunak adalah penanganan crosscutting concerns. Crosscutting concern muncul pada kebutuhan system-level yang berupa: logging, authentication, security, performance, error checking, multythread savety, dll. Kebutuhan sistem-level kadang saling mempengaruhi satu sama lain, mempengaruhi beberapa subsistem, atau bahkan ada kemungkinan untuk mempengaruhi jalannya sistem secara keseluruhan. Crosscutting concern menyebabkan penyebaran fungsi-fungsi umum di seluruh sistem.

Dalam Tugas Akhir ini dilakukan pemodelan terhadap studi kasus sistem keamanan login, karena sistem keamanan merupakan salah satu contoh dari kebutuhan system-level dengan memberikan usulan untuk notasi-notasi dalam pemodelan aspek. Dengan demikian diharapkan adanya gambaran yang jelas mengenai tahap pemodelan pada teknik pembangunan perangkat lunak berorientasi aspek serta perbedaannya dengan teknik pembangunan perangkat lunak berorientasi objek.

Dari hasil proses analisis dan perancangan disimpulkan bahwa UML merupakan alat yang extendible dimana bisa ditambahkan notasi-notasi baru melalui mekanisme stereotype. Hasil implementasi dari OOD dan menunjukkan bahwa hasil rancangan dengan menggunakan AOD memiliki rancangan yang lebih bersih karena tidak ada crosscutting concerns.

Kata kunci: *berorientasi aspek, crosscutting concerns, system-level.*

KATA PENGANTAR

Grace Copy Center

KATA PENGANTAR

Alhamdulillah, segala puja dan puji hanyalah bagi Allah Yang SWT, Yang telah melimpahkan hidayah dan kasih sayang-Nya. Atas kemudahan yang diberikan-Nya lah penulis dapat menyelesaikan Tugas Akhir ini. Shalawat serta salam tetap kita curahkan kepada Rasulullah Muhammad SAW.

Penulis memberi judul pada Tugas Akhir ini: **PEMBANGUNAN PERANGKAT LUNAK BERORIENTASI ASPEK KE DALAM NOTASI UML, STUDI KASUS PADA SISTEM KEAMANAN PERANGKAT LUNAK.**

Penulis berharap semoga Tugas Akhir ini dapat membantu pengembangan teknologi informasi di jurusan Teknik Informatika ITS. Tugas akhir ini masih jauh dari sempurna, oleh karena itu sangat diharapkan segala masukan bagi pengembangan selanjutnya.

Akhirnya, penulis mengucapkan terima kasih yang sebesar-besarnya kepada semua pihak yang telah membantu penyelesaian tugas akhir ini.

Surabaya, Juli 2005

Siti Kustiarsih

UCAPAN TERIMAKASIH

Grace Copy Center

UCAPAN TERIMA KASIH

Alhamdulillah, penulis bisa menyelesaikan Tugas Akhir berkat pertolongan Allah SWT. Selain itu penulis juga mendapatkan bantuan dari berbagai pihak. Untuk itulah, melalui kesempatan ini penulis mengucapkan terima kasih yang sebesar-besarnya kepada:

1. Allah SWT, Rabbul izzati yang telah memberikan limpahan rahmat dan hidayah-Nya, serta segala nikmat yang tak terhitung jumlahnya.
2. Keluarga, yang tak henti-hentinya memberikan dukungan dan nasihat kepada penulis agar selalu bersemangat dan bekerja keras dalam menjalani setiap detik kehidupan ini. Semoga Allah selalu melimpahkan ampunan, kasih sayang dan hidayah-Nya kepada kami semua.
3. Bapak Prof. Dr. Ir. Arif Djunaidy, M.Sc, selaku Ketua Dekan Fakultas Teknologi Informasi ITS dan Bapak Yudhi Purwananto, M.Kom, selaku Ketua Jurusan Teknik Informatika ITS.
4. Ibu Ir. Siti Rochimah, MT selaku dosen pembimbing, yang telah memberikan beribu-ribu ilmu dan wawasan yang bermanfaat kepada penulis selama mengerjakan Tugas Akhir.
5. Bapak Muhammad Husni selaku dosen wali, Bapak Ryanarto Sarno, Bapak Daniel O. Siahaan dan Ibu Umi Laili Yuhana, selaku dosen penguji serta seluruh dosen di Jurusan Teknik Informatika ITS yang telah memberikan ilmu yang sangat bermanfaat bagi penulis.
6. Ria, Dwi, Nina, sama Dipa.....f5 never dies, friendship never ends!! Luv u guys!!

7. Teman-teman dan kakak-kakak di kos Wisma Permai 1 No. 163. Mbak Astin yang selalu bersemangat, Mbak Ikan yang mengaku namanya Fatin yang suka konser musik di pagi, siang, sore, dan malam hari, Ria yang suka pulang ke Samarinda (lho?!), Dhewy yang mirip anak smp :P, Rossy yang meski paling kecil tapi pemberani (jadi inget pas mati lampu malam itu.:P), trus juga...bapak kos yang aneh! Aja Aja Fighting!!!:)
8. Sempal my nice nephew ☺ lengkap dengan smsnya yang aneh, Rolly si Babon PM(thanks laptopnya!), Galih Ndut yang suka bantuin ngabisin makanan, Fanni(sasaran Battle Royal 2005), Phe yang selalu tabah karena request friendsternya enggak ku-approve :D. Ntar maen-maen ke Parahyangan lagi yah! Qta pecahin rekor Mesin Cuci Elextrolux Samsung Automatic!
9. Anak2 seperjuangan bimbingan Bu Siti. Dhewy (namamu uda muncul dua kali dhew!), Sigit Bulu, Sigit Pram, Mas Hendra, Deaddie Mucil, Choy, Pandu, Miko, Dhoma, Harris.
10. Anak2 lab RPL sama adminnya yang baek-baek yang uda bikin suasana lab jadi nyaman bwt ngerjain TA. Dwi, yang kompinya sebelahan ma kompiku, Mas Junaidy yang siap sedia benerin kompi yang kena virus, Dewa yang ngasih installer yang aku butuhin, Taqi thanks Eclipse-nya, dan Yovi.
11. Kent yang suka aku ganggu pake imel-imelku saat aku pusing ngerjain TA, tapi jarang marah. "jarang" loh ya..bukannya enggak pernah....:D

DAFTAR ISI

Grace Copy Center

DAFTAR ISI

ABSTRAK iii
KATA PENGANTAR iii
UCAPAN TERIMA KASIH iv
DAFTAR ISI vii
DAFTAR GAMBAR ix
DAFTAR TABEL xi
BAB I 1
PENDAHULUAN 1
1.1. Latar Belakang 1
1.2. Tujuan dan Manfaat 2
1.3. Permasalahan 2
1.4. Metodologi 3
1.5. Sistematika Laporan 4
BAB II 6
DASAR TEORI 6
2.1. UML Rational Rose 6
2.2. Pemodelan berorientasi objek 15
2.3. Pemrograman Berorientasi Aspek 17
2.4. Pemodelan Berorientasi Aspek 24
2.5. Uji Coba yang Pernah Dilakukan Terhadap Keunggulan AOD 28
BAB III 35
ANALISIS DAN PERANCANGAN SISTEM KEAMANAN 35
3.1 Notasi untuk Istilah-istilah dalam AOD 35
3.2 Studi Kasus pada Sistem login 40
3.2.1 Use Case Diagram 43
3.2.2 Sequence Diagram 46
3.2.3 Class Diagram 49
3.3 Pemodelan perangkat lunak berorientasi aspek pada sistem keamanan login pada aplikasi SOP. 52

3.3.1 Sequence Diagram 54
3.3.2 Class Diagram 55
3.4 Perbedaan antara pemodelan AOD dengan OOD pada sistem keamanan login pada aplikasi SOP. 57
BAB IV 59
IMPLEMENTASI DAN UJI COBA MODEL SISTEM KEAMANAN 59
4.1. Lingkungan Pembangunan Perangkat Lunak 59
4.2. Implementasi Kelas Perangkat Lunak 60
4.2.1. Kelas DBManager 60
4.2.2. Kelas Authen 61
4.2.3. Kelas UserList 62
4.2.4. Kelas UA 62
4.2.5. Kelas UAList 63
4.2.6. Kelas Credentials 63
4.2.7. Kelas LoginForm 64
4.2.8. Kelas MainForm 66
4.3. Implementasi Proses 68
4.4. Implementasi dan Uji Coba AntarMuka 69
BAB V 71
KESIMPULAN DAN SARAN 71
5.1. Kesimpulan 71
5.2. Saran 71
DAFTAR PUSTAKA 72

DAFTAR GAMBAR

Grace Copy Center

DAFTAR GAMBAR

Gambar 2.1. Contoh *Use case diagram* 7
Gambar 2.2. Contoh *activity diagram* 7
Gambar 2.3. Contoh *sequence diagram* 8
Gambar 2.4. Contoh *collaboration diagram* 8
Gambar 2.5. Contoh *class diagram* 9
Gambar 2.6. Contoh *deployment diagram* 9
Gambar 2.7. Contoh *browser* 11
Gambar 2.8. Contoh *window diagram* 12
Gambar 2.9. Contoh *toolbar diagram* 12
Gambar 2.10. Contoh *documentation window* 13
Gambar 2.11. Contoh *log window* 13
Gambar 2.12. Contoh *option window* 14
Gambar 2.13. Contoh jenis-jenis *stereotype* 14
Gambar 2.14. Implementasi modul sebagai sekumpulan *concerns* 19
Gambar 2.15. Dekomposisi *concerns* 19
Gambar 2.16. Pembangunan program berorientasi aspek 20
Gambar 2.17. Enkapsulasi nama, atribut, dan layanan dalam kelas 21
Gambar 2.18. Pola umum deklarasi kelas 21
Gambar 2.19. *Weaver* 22
Gambar 2.20. Contoh OOD untuk editor gambar sederhana 29
Gambar 2.21. Contoh AOD untuk editor gambar sederhana 30
Gambar 3.1. Contoh notasi aspek pada *class diagram* 36
Gambar 3.2. Notasi *join point* untuk *method* yang melakukan *crosscutting* terhadap pada *sequence diagram* 36
Gambar 3.3. Notasi *join point* untuk penambahan *method* pada *sequence diagram* 36
Gambar 3.4. Contoh penerapan notasi *join point* untuk *method* yang melakukan *crosscutting* terhadap sistem, pada *sequence diagram* 37

MILIK PERPUSTAKAAN
INSTITUT TEKNOLOGI
SEPULUH - NOPEMBER

Gambar 3.5. Contoh penerapan notasi *join point* untuk penambahan *method*, pada *sequence diagram* 37
Gambar 3.6. Contoh penerapan notasi *join point*, *pointcut*, *aspek*, *advice*, dan *intertype declaration* dalam *class diagram* 39
Gambar 3.7. Use Case untuk login 43
Gambar 3.8. Sequence diagram pada OOD untuk aksi normal 46
Gambar 3.9. Sequence diagram untuk aksi alternative 48
Gambar 3.10. Class diagram pada OOD 49
Gambar 3.11. Sequence diagram pada AOD untuk aksi normal 55
Gambar 3.12. Class diagram pada sistem keamanan login yang dilengkapi dengan notasi aspek 55
Gambar 4.1. Implementasi kelas DBManager 61
Gambar 4.2. Implementasi kelas Authen 61
Gambar 4.3. Implementasi kelas UserList 62
Gambar 4.4. Implementasi kelas UA 63
Gambar 4.5. Implementasi kelas UAList 63
Gambar 4.6. Implementasi kelas Credentials 63
Gambar 4.7. Implementasi kelas LoginForm 66
Gambar 4.8. Implementasi kelas MainForm 67
Gambar 4.9. Implementasi aspek check 68
Gambar 4.10. Implementasi aspek RoleHierarchy 68
Gambar 4.11. Implementasi antarmuka untuk halaman login 69
Gambar 4.12. Implementasi antarmuka untuk halaman utama 69
Gambar 4.13. Pesan kesalahan pada halaman login 69

DAFTAR TABEL

Grace Copy Center

DAFTAR TABEL

Tabel 3.1. Skenario login pada aplikasi web SOP 44
Tabel 3.2. Operasi dan atribut pada sistem login OOD serta keterangannya..... 49
Tabel 3.3. Resume OOD dan AOD..... 58
Tabel 4.1 Lingkungan pembangunan perangkat lunak 59

BAB I

PENDAHULUAN

Grace Copy Center

BAB I

PENDAHULUAN

1.1. Latar Belakang

Sekarang ini pengembangan perangkat lunak banyak dikerjakan dengan menggunakan metode berorientasi objek, contohnya UML. Pada OOD (*Object Oriented Design*) tahap perancangan perangkat lunak dikerjakan dengan menentukan objek-objek yang penting untuk mendukung dan memenuhi kebutuhan (*requirement*) yang ada[CLI03]. Tujuannya adalah untuk mengelompokkan fungsionalitas seperti yang telah ditentukan pada saat menganalisis kebutuhan. Melalui proses ini, perancang sering lupa bahwa objek-objek ini memerlukan interaksi dengan objek lainnya untuk memenuhi kebutuhan. Hal ini terjadi karena bagaimana cara kebutuhan berinteraksi berbeda dengan objek. Akibatnya, arsitektur objek menjadi kacau / membingungkan dan modularitas sistem tidak jelas. Hal ini juga mempengaruhi *re-usability* sistem.

Tugas Akhir ini akan menggali tahap analisis dan memodelkan tahap perancangan perangkat lunak dengan menggunakan metode berorientasi aspek, dimana aspek merupakan pengembangan dari objek. Pemodelan aspek juga menggunakan UML yang sudah ada yang telah dikembangkan. Dengan menggunakan aspek akan mempermudah tahap perancangan sebuah sistem, terutama nanti pada saat sistem tersebut mengalami perubahan (misalnya karena perkembangan teknologi) karena

aspek memisahkan *concerns* ke dalam modul-modul sehingga sistem yang dibuat mempunyai modularitas yang bagus pada tahap perancangan perangkat lunak. Modularitas yang bagus pada tahap perancangan mempunyai kelebihan dalam hal *comprehensibility*, *reusability*, *adaptability*, *flexibility*, dan *maintainability*.

1.2. Tujuan dan Manfaat

Tujuan dari Tugas Akhir ini adalah untuk memodelkan suatu aspek yang terdiri dari sekumpulan *concerns* ke dalam notasi UML pada waktu tahap pemodelan perangkat lunak. Dengan adanya notasi baru pada UML yang menggambarkan aspek ini, diharapkan tahap analisis dan perancangan pada proses pembangunan perangkat lunak bisa dilakukan dengan lebih bersih dan jelas.

1.3. Permasalahan

Permasalahan yang diangkat dalam tugas akhir ini adalah sebagai berikut.

1. Menentukan *concerns* pada suatu permasalahan/studi kasus
2. Memisahkan *concerns* pada suatu permasalahan/studi kasus untuk kemudian ditempatkan pada suatu aspek
3. Menggambarkan notasi aspek ke dalam notasi UML beserta relasinya dengan kelas-kelas lain.

1.4. Metodologi

Pengerjaan tugas akhir ini terbagi menjadi beberapa tahapan, yaitu:

1. Studi literatur

   Tahap ini merupakan tahap pencarian informasi sehubungan dengan proses-proses yang dilakukan pada tahap perancangan suatu pengembangan perangkat lunak yang berorientasi aspek. Informasi-informasi tersebut didapatkan dari internet, dikarenakan buku mengenai topik tersebut masih sangat jarang.

2. Pengumpulan data

   Pada tahap ini dilakukan pengumpulan data mengenai permasalahan-permasalahan/studi kasus yang bisa diselesaikan dengan menggunakan metode pengembangan perangkat lunak yang berorientasi aspek.

3. Analisis dan perancangan data

   Pada tahap analisis dan perancangan, dilakukan analisis data-data yang telah terkumpul untuk mempelajari karakteristiknya untuk dapat dimodelkan dengan aspek. Selanjutnya merancang pembuatan notasi aspek untuk data-data tersebut. Pada tahap ni juga diusulkan notasi-notasi untuk istilah-istilah yang terdapat dalam teknik pemodelan berorientasi aspek.

4. Implementasi dan ujicoba perancangan

   Tahap ini merupakan tahap untuk mengimpementasikan /membuktikan hasil rancangan yang sudah dibuat, yaitu dengan

membuat program berorientasi aspek sesuai dengan rancangan yang telah dibuat sebelumnya. Setelah itu dilakukan tes untuk memeriksa apakah hasil implementasi yang dibuat sudah benar dan dilakukan evaluasi untuk mengetahui apa saja kekurangannya

1.5. Sistematika Laporan

Uraian sistematika penulisan pada tugas akhir ini bertujuan agar perancangan dan pembuatan perangkat lunak yang dibahas menjadi mudah dipahami, jelas dan sistematis untuk tiap-tiap bab atau sub bahasan. Secara kronologis uraian dalam tugas akhir ini disusun sebagai berikut :

Bab I Pendahuluan.

Bagian ini berisi latar belakang, permasalahan, tujuan, batasan permasalahan, metodologi, dan sistematika pembahasan.

Bab II Dasar Teori.

Bab ini berisi penjelasan mengenai dasar ilmu yang mendukung pembahasan tugas akhir ini, meliputi UML (*Unified Modeling Language*), pemodelan berorientasi aspek AOD (*Aspect Oriented Design*), dan sedikit mengenai pemrograman berorientasi aspek AOP (*Aspect Oriented Programming*).

Bab III Analisis dan Perancangan Sistem Keamanan.

Bab ini berisi penjelasan mengenai notasi untuk istilah-istilah dalam pemodelan berorientasi aspek, analisis dan pemodelan sistem keamanan login dengan menggunakan mekanisme pemodelan berorientasi objek,

memodelkan sistem keamanan login menggunakan pemodelan berorientasi aspek, serta perbedaan antara pemodelan berorientasi objek dan pemodelan berorientasi aspek.

Bab IV Implementasi dan Uji Coba Model Sistem Keamanan.

Pada bagian ini akan dibuat implementasi/pembuktian dari hasil perancangan pada bab sebelumnya. Setelah itu dilakukan uji coba dan evaluasi terhadap rancangan yang telah dibuat.

Bab V Kesimpulan dan Saran

Merupakan uraian kesimpulan dari bab-bab sebelumnya, serta saran untuk pengembangan selanjutnya.

BAB II

DASAR TEORI

Grace Copy Center

BAB II

DASAR TEORI

Pada bab ini akan dibahas mengenai teori-teori yang mendasari pembuatan Tugas Akhir yang meliputi pembahasan UML Rational Rose, Pemodelan Berorientasi Objek, Pemrograman Berorientasi Aspek, dan Pemodelan Berorientasi Aspek.

2.1. UML Rational Rose

- UML (*Unified Modeling Language*)

UML [BOO98] merupakan bahasa standar untuk visualisasi, spesifikasi, pembangunan, dan pendokumentasian artefak pada sistem perangkat lunak. UML terdiri dari:

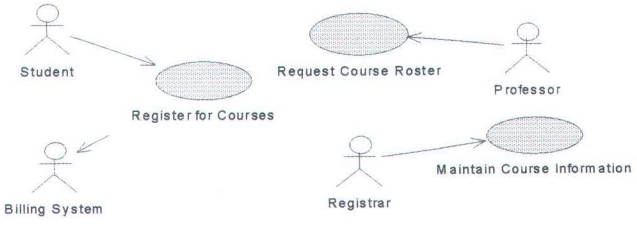
- Pemodelan data (*Data modeling*)
- Pemodelan bisnis (*Business modeling*)
- Pemodelan objek (*Object modeling*)
- Pemodelan komponen (*Component modeling*)

UML mungkin digunakan untuk memvisualisasi model:

- Perilaku (*behavior*) aplikasi
- Struktur sistem
- Arsitektur proyek yang akan dikerjakan
- Komponen sistem

Use Case Diagram

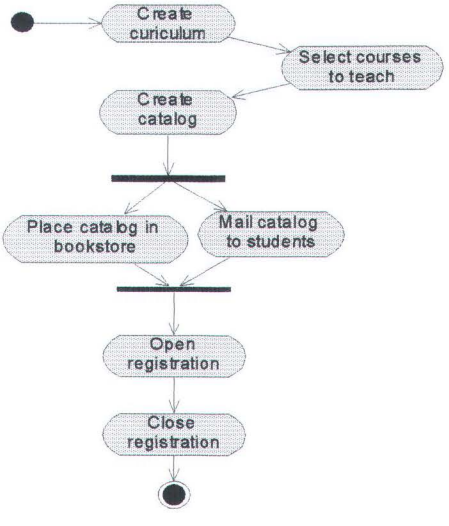
Use-case diagrams dibuat untuk menggambarkan interaksi sistem dengan luar.



Gambar 2.1. Contoh *Use case diagram*

Activity Diagram

Activity diagram menunjukkan aliran kejadian dalam sistem.

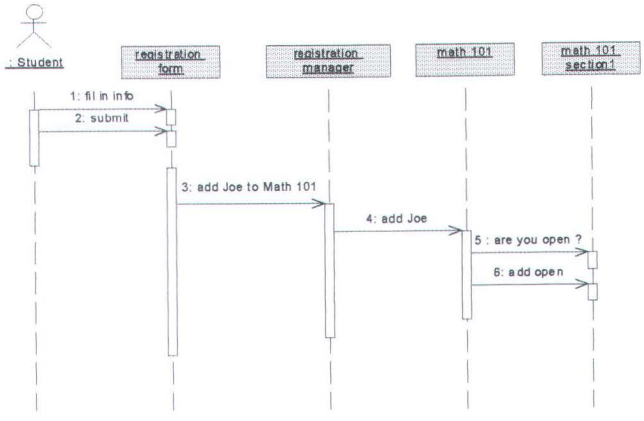


PERPUSTAKAAN INSTITUT TEKNOLOGI SEPULUH - NOPEMBER

Gambar 2.2. Contoh *activity diagram*

Sequence Diagram

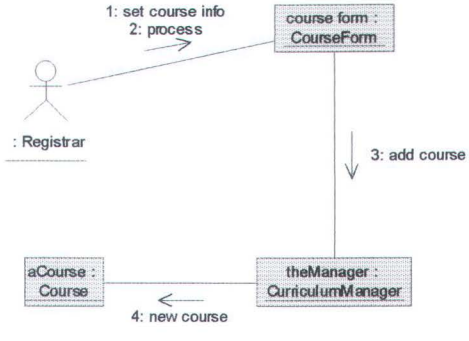
Sequence diagram menggambarkan langkah-langkah yang harus dilakukan menyelesaikan bagian fungsionalitas yang disediakan oleh sistem. untuk



Gambar 2.3. Contoh *sequence diagram*

Collaboration Diagram

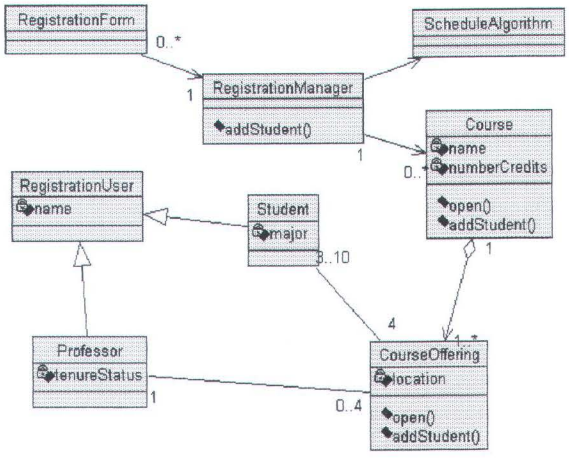
Collaboration diagram menggambarkan interaksi objek yang diatur di sekitar objek dan hubungannya satu dengan yang lain.



Gambar 2.4. Contoh *collaboration diagram*

Class Diagram

Class diagram menunjukkan struktur perangkat lunak yang dibangun.



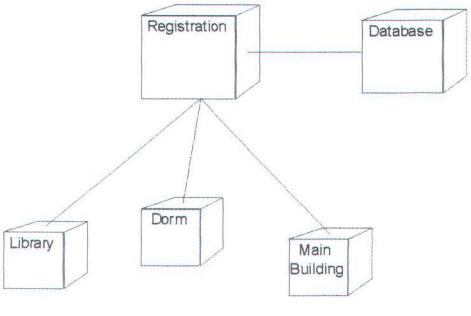
Gambar 2.5. Contoh *class diagram*

Component Diagram

Component diagrams mengilustrasikan pengaturan dan ketergantungan antar komponen perangkat lunak.

Deployment Diagram

Deployment diagram menggambarkan distribusi komponen dalam perusahaan.



Gambar 2.6. Contoh *deployment diagram*

Perluasan/pengembangan UML

- *Stereotypes* bisa digunakan untuk memperluas/mengembangkan elemen notasi UML.
- *Stereotypes* bisa digunakan untuk mengklasifikasikan dan memperluas asosiasi, hubungan pewarisan, kelas-kelas, dan komponen-komponen.
- Contoh:
  - *Stereotypes* kelas: *interface*, *exception*, *server page*
  - *Stereotypes* asosiasi: *identifying*, *non-identifying*
  - *Stereotypes* keergantungan: *include*, *extend*
  - *Stereotypes* komponen: *subsystem*

- Rational Rose

Use-Case View

Use-Case View merupakan "jantung" dari views yang lain karena menspesifikasikan mengenai APA yang harus dilakukan oleh sistem.

Logical View

Logical View mendukung kebutuhan fungsional sistem.

Component View

Component View mengatasi masalah untuk memudahkan pembangunan, pengaturan asset perangkat lunak, dan penggunaan kembali sub-kontak.

Deployment View

Deployment View mengatasi masala seperti penyebaran, instalasi, dan performasi.

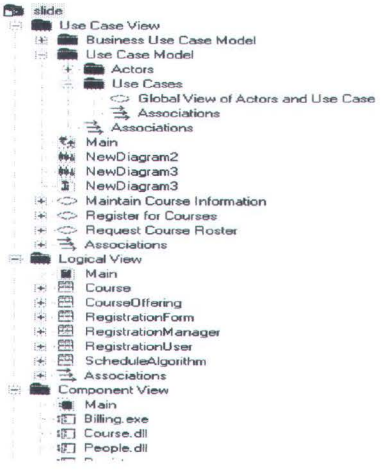
Antar muka Rational Rose.

Antar muka Rational Rose terdiri dari:

- *Browser*
- *Diagram window*
- *Diagram toolbar*
- *Documentation window*
- *Log window*
- *Options window*

Browser

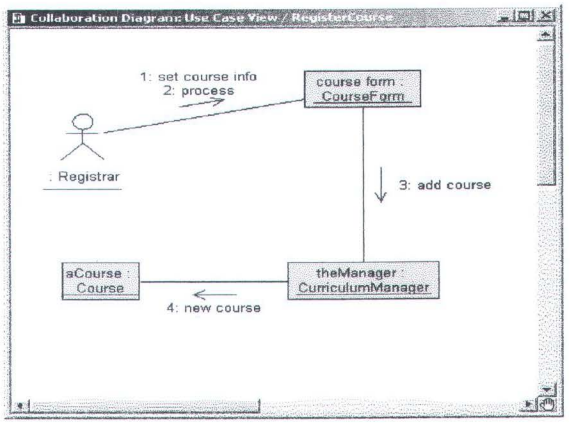
Browser memperbolehkan kita untuk melihat dan semua *views* dan diagram dalam Rational Rose.



Gambar 2.7. Contoh *browser*

Diagram window

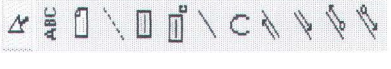
Diagram window memperbolehkan kita untuk membuat, meng-update, dan memodelkan view grafis pada model yang sedang dibuat.



Gambar 2.8. Contoh *window diagram*

Diagram Toolbar

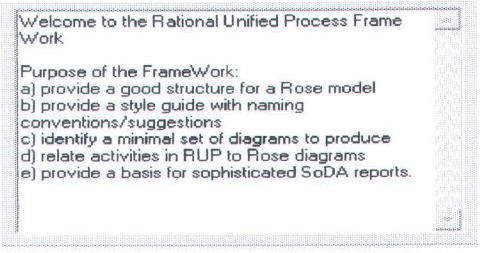
- *Diagram toolbar* berisi elemen-elemen untuk membangun diagram.
- Masing-masing toolbar bersifat unik terhadap masing-masing jenis diagram.
- Hanya akan diaktifkan ketika diagramnya ditampilkan.



Gambar 2.9. Contoh *toolbar diagram*

Documentation window

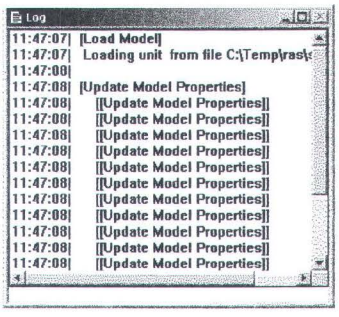
Documentation window digunakan untuk membuat, melihat, atau memodifikasi teks/tulisan yang menjelaskan bagian-bagian tertentu yang dipilih untuk dibuatkan dokumentasinya dalam suatu diagram.



Gambar 2.10. Contoh *documentation window*

Log window

Log window melaporkan kemajuan, hasil, dan kesalahan. Contoh, komen penurunan kode memberikan pesan kemajuan dan kesalahan pada *window* di bawah ini.

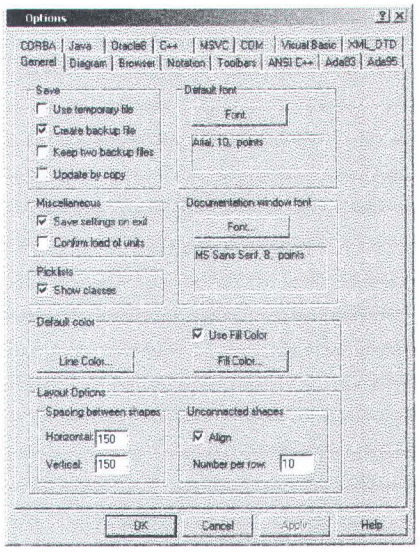


Gambar 2.11. Contoh *log window*

Option window

Options window digunakan untuk mengatur semua *default* yang digunakan selama memodelkan perangkat lunak.

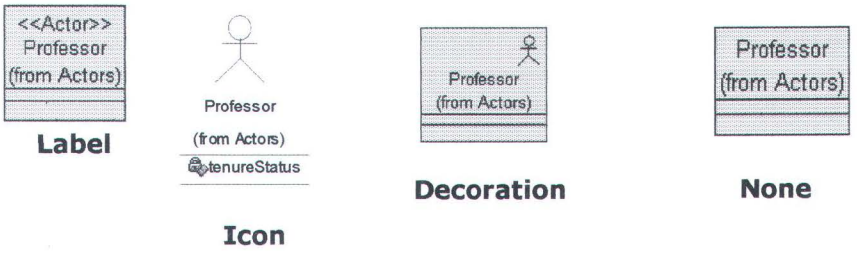
Perhatikan bahwa jika kita melakukan perubahan *defaut*, elemen model yang sudah ada tidak berubah.



Gambar 2.12. Contoh *option window*

Stereotype display

Pilihan ini memperbolehkan kita untuk memilih bagaimana *stereotype* yang kita inginkan tampak pada diagram.



Gambar 2.13. Contoh jenis-jenis *stereotype*

Basic tool techniques

Ada dua teknik dasar yang akan kita diskusikan:

- Menghapus elemen diagram
- Menambah elemen diagram

Menghapus elemen diagram

Langkah-langkah:

- Pindahkan elemen yang dipilih yang akan dihapus dari model
- Pindahkan semua *icon* yang merepresentasikan elemen, dari semua diagram dimana elemen itu ada.
- Hapus spesifikasi untuk elemen.
- Jika kita langsung menghapus elemen dari diagram maka Rose akan memindahan elemen tersebut dari diagram, tetapi tidak merubah model kecuali jika *icon*-nya tidak diberi nama dan muncu hanya pada diagram tersebut dan tidak pada diagram yang lain.

Menambah elemen diagram

Untuk menambah elemen dalam diagram, kita bisa melakukannya dengan cara menambah melalui toolbar maupun melalui *browser*.

2.2. Pemodelan berorientasi objek

Prinsip desain/pemodelan berorientasi objek[SHI01].

Desain berorientasi objek berdasarkan pada tiga prinsip utama, yaitu enkapsulasi, inheritan, dan polimorfisme. Sebuah bahasa pemrograman dikatakan mendukung desain berorientasi objek jika memenuhi tiga konsep tersebut dalam sintak-sintaknya. Beberapa bahasa pemrograman menyediakan alat untuk memudahkan pendefinisian dan penggunaan konsep diatas.

Enkapsulasi mengacu pada konsep pembuatan sebuah objek atau yang biasa disebut dengan "Black Box". Ketika menggunakan objek, kita tidak perlu tahu apa yang terjadi di dalamnya. Kita tidak perlu mengerti bagaimana sebuah objek bekerja. Sebuah objek hanya perlu menunjukkan informasi-informasi yang diperlukan untuk berinteraksi dengan objek lain. Sebuah objek harus menyediakan antarmuka yang mudah dimengerti mengenai sekumpulan method dan propertisnya yang mana mungkin menurut seorang desainer akan berguna bagi pengguna lain. Enkapsulasi juga berarti bahwa sebuah objek berisi semua hal yang dibutuhkannya, yaitu data dan operasi-operasi(*methods*) pada data tersebut. Konsep enkapsulasi merupakan konsep yang sangat kuat karena memperbolehkan pembagian pekerjaan yang efisien pada proyek perangkat lunak yang besar. Masing-masing anggota tim bisa bekerja pada objeknya tanpa harus selalu berinteraksi dengan anggota lain. Pengeluaran tambahan pada pembangunan proyek berjalan secara oksponensial dengan jumlah antar muka antar anggota grup. Enkapsulasi merupakan kontributor utama bagi desain berorientasi objek dalam hal penanganan "Krisis perangkat lunak".

Penggunaan kembali perangkat lunak merupakan salah satu hal yang ingin dicapai oleh desain berorientasi objek. Salah satu cara untuk mencapainya adalah dengan menggunakan inheritan/pewarisan. Sebuah kelas merupakan fungsi yang mendefinisikan sebuah objek. Sebuah *superclass* merupakan kelas dimana kelas baru, *subclasses* dibuat. *Subclass* mewarisi semua *method* dan properti dari *superclass*-nya. Pada prakteknya, semua *subclasses* diturunkan secara otomatis, dan menyebabkan penyimpanan yang sangat besar. Kita tidak perlu

mendefinisikan *subclasses* ini satu per satu. Kita bisa mengabaikan *methods* dan operasi-operasi yang diwariskan, tidak ada keharusan untuk membuat *subclass* yang merupakan duplikasi 100% dari *superclass*-nya, kecuali jika kita mengabaikan paling tidak satu properties atau satu *method*.

Polimorfisme mungkin merupakan komponen yang paling sukar yang harus dimiliki. Inti dari konsep ini adalah bahwa setiap kelas harus menangani tipe data yang berbeda. Kita tidak boleh membuat kelas yang berbeda untuk menangani tipe data yang berbeda. Contohnya adalah kelas drawing. Kita tidak boleh menuliskan kelas yang berbeda untuk menggambar lingkaran, persegi, dan oval. Seharusnya ada satu kelas yang cukup pintar untuk memanggil *method* yang sesuai untuk mengoperasikan bentuk yang benar.

2.3. Pemrograman Berorientasi Aspek

Pemrograman berorientasi aspek merupakan metode pemrograman yang diperkenalkan sebagai hasil penelitian Gregor Kiczales di Xerox's Palo Alto Research Center (PARC). Paragdima ini dikembangkan sebagai pilihan solusi dari persoalan dekomposisi fungsional kode program dalam pembangunan perangkat lunak[LAD02].

Yang termasuk persoalan dekomposisi fungsional adalah sebagai berikut:

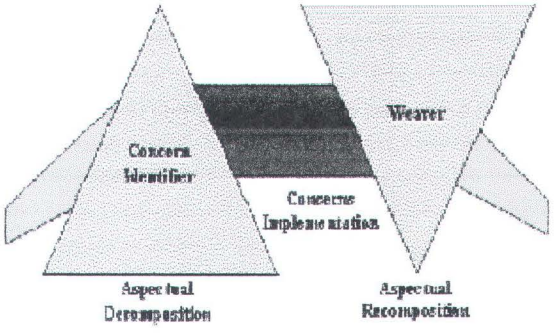
1. integritas sistem (*system integrity*)
2. adaptibilitas program (*adaptability*)
3. maintainabilitas program (*maintainability*)
4. guna ulang program (*reusability*)

Dalam pembangunan sebuah perangkat lunak yang kompleks, masalah yang harus diimplementasikan pada umumnya akan terbagi menjadi dua bagian besar. Bagian pertama adalah kumpulan masalah yang dapat dikelompokkan ke dalam struktur kelas, sedangkan bagian kedua adalah kumpulan aspek khusus yang tidak dapat dimodularisasi sebuah kelas, namun memiliki peran yang penting dalam keseluruhan bangunan program, dimana apabila diimplementasikan akan berupa baris-baris kode yang berulang dan tersebar di banyak tempat dalam kode program.

Aspek merupakan unit modular yang merepresentasikan permasalahan dengan karakteristik sebagai berikut:

1. sinkronisasi proses
2. penanganan eksepsi/asersi
3. manajemen memori
4. pencatatan kejadian (*logging*)
5. komunikasi antar proses
6. performasi sistem

Gambar di bawah ini menggambarkan sebuah sistem sebagai sekumpulan *concerns* yang diimplementasikan oleh berbagai jenis modul[LAD02].



Gambar 2.16. Pembangunan program berorientasi aspek

Pemrograman berorientasi aspek merupakan pengembangan dari pemrograman berorientasi objek[LAD02]. Aktivitas pengembangan AOP secara umum terdiri dari langkah-langkah sebagai berikut:

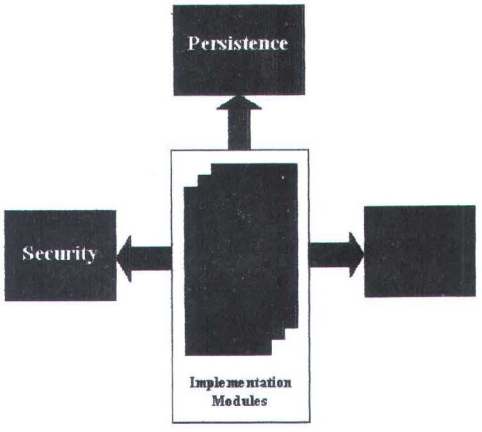
1. Dekomposisi fungsional kelas dan aspek

   Pada tahap ini semua kebutuhan (*requirements*) didekomposisi, dimana *concerns* umum (*concerns* yang mewakili komponen fungsional yang sudah modular) menjadi kelas, sedangkan *concerns* yang sifatnya khusus (*crosscutting concerns*) menjadi aspek.

2. Implementasi kelas dan aspek

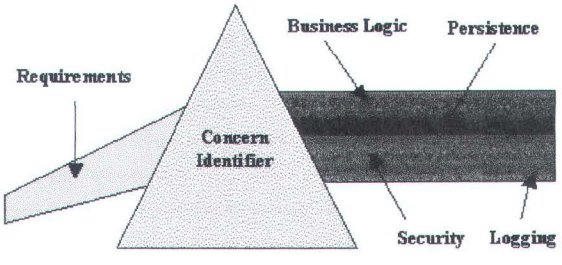
   Semua *concerns* hasil dekomposisi diimplementasikan menjadi unit-unit individual. Sebuah kelas akan diimplementasikan sebagai enkapsulasi entitas tertentu yang terdiri dari komponan nama, layanan(*method/service*), dan atribut(*property/fields*).

MILIK PERPUSTAKAAN
INSTITUT TEKNOLOGI
SEPULUH - NOPEMBER



Gambar 2.14. Implementasi modul sebagai sekumpulan *concerns*

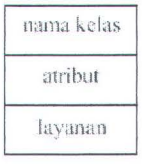
Gambar dibawah ini menunjukkan sekumpulan kebutuhan yang digambarkan sebagai sinar *beam* yang melalui prisma penentu *concerns*. Selanjutnya semua kebutuhan tadi dilewatkan melalui prisma yang akan memisahkan *concerns* [LAD02].



Gambar 2.15. Dekomposisi *concerns*

Sedangkan gambaran mengenai langkah umum dalam pembangunan program berorientasi aspek adalah seperti di bawah ini:

Enkapsulasi tersebut bisa digambarkan seperti dibawah ini:



Gambar 2.17. Enkapsulasi nama, atribut, dan layanan dalam kelas

dan dideklarasikan dengan pola umum:

```
Class X          // nama kelas
// feature : atribut

// feature : method
End Class X;
```

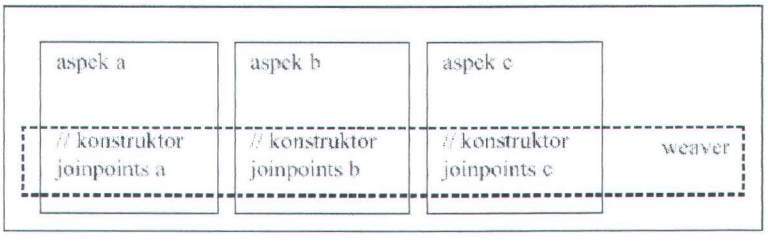
Gambar 2.18. Pola umum deklarasi kelas

Sebuah aspek diimplementasikan sebagai enkapsulasi beberapa komponen. Aspek dapat memiliki komponen atribut dan *method* seperti halnya kelas. Satu komponen semantic tambahan untuk aspek yang tidak dimiliki oleh kelas adalah fasilitas pendefinisian *joinpoint*. Fasilitas ini memungkinan pada saat eksekusi, sebuah aspek menjalankan perannya dalam beberapa fragmen-logis-kode-program yang melibatkan kelas atau aspek lainnya.

3. Rekomposisi aspek

Pada langkah ini didefinisikan aturan rekomposisi untuk semua aspek dan kelas yang telah dibangun. Proses rekomposisi (disebut juga dengan *weaving* atau *integrating*) akan membentuk sistem final dari

perangkat lunak. Sebuah *weaver* dibangun dari kumpulan konstruktor *joinpoints* yang terdapat dalam semua unit aspek yang telah didefinisikan.



Gambar 2.19. *Weaver*

Aturan yang didefinisikan dalam *weaver* akan menentukan kapan komponen dari sebuah aspek dieksekusi pada fragmen-logis tertentu. Beberapa istilah yang digunakan dalam AOP[GAO04].

1. Joinpoints

Joinpoints merupakan titik logik eksekusi yang didefinisikan pada saat pembuatan program. Pada titik-titik logis tersebut, suatu aspek dieksekusi sesuai dengan aturan yang telah ditetapkan pada saat penyusunan program. Contoh tipe *joinpoints*: pemanggilan *method*(*method call*), eksekusi *method*(*method execution*), instantiations, pengaksesan nilai sebuah *field*(*field reference*).

2. Pointcuts

Konstruktor yang digunakan program untuk menunjuk *join point* tertentu pada saat eksekusi. Sebuah poincuts dideklarasikan dengan menggunakan kata kunci "pointcut".

3. Advice

Merupakan perilaku untuk mengeksekusi *join point*. Beberapa jenis advise yang menentukan bagaimana advice berinteraksi dengan *join point* : before (sebelum *join point*), after (setelah *join point*), dan around (pada *join point*).

4. Intertype declaration

Mekanisme untuk menambah atribut dan method untuk kelas yang sudah ada sebelumnya. Penambahan atribut dan atau *method* dilakukan di aspek untuk menghindari adanya kemungkinan rancangan yang kacau akibat penambahan tadi. Karena jika atribut atau *method* langsung ditambahkan di kelas, bisa saja mempengaruhi relasi kelas tersebut dengan kelas yang lain.

5. Aspect

Aspect merupakan sebuah unit modular yang didesain untuk mengimplementasikan sebuah *concern*. Definisi *aspect* mungkin berisi beberapa kode (atau *advice*) dan instruksi mengenai dimana, kapan, dan bagaimana memanggilnya. Tergantung pada bahasa aspect, aspect bisa dibangun secara hirarki, dan bahasa mungkin menyediakan mekanisme untuk menentukan sebuah aspect dan menspesifikasikan interaksinya dengan sistem dasar.

2.4. Pemodelan Berorientasi Aspek

Pembangunan Perangkat Lunak Berorientasi Aspek (AOSD) merupakan suatu paradigma yang muncul yang menyediakan abstraksi yang jelas mengenai *concerns* yang cenderung untuk melakukan *crosscutting* berbagai komponen sistem dan mengatasi komponen individual yang membingungkan[TEK03] sehingga menghasilkan suatu modularitas yang bersih dan lebih baik. Dengan menyediakan *crosscutting concerns* atau aspek sebagai abstrak terbaik, dan dengan menyediakan teknik komposisi baru untuk mengkombinasikan aspek dengan komponen, modularitas sistem bisa ditingkatkan untuk mengurangi kompleksitas sistem serta untuk memudahkan pada saat proses *maintenance*.

Alasan utama mengapa menggunakan Pembangunan Perangkat Lunak Berorientasi Aspek adalah untuk membantu menyelesaikan isu mengenai arsitektur objek yang kacau [CLI03]. Para perintis AOSD menyatakan bahwa pemrograman berorientasi objek mengalami kesulitas saat berhadapan dengan perkembangan informasi global. Selain itu, fungsionalitas yang membutuhkan keterlibatan beberapa objek yang berbeda (kemungkinan terkumpul dalam satu komponen) menghasilkan ketergantungan antar objek-objek atau komponen-komponen tersebut. Hal ini membuat aplikasi menjadi rentan terhadap perubahan yang terjadi pada objek atau komponen yang tergantung (*dependent*). *Maintenance* dan *enhancement* juga merupakan suatu masalah, karena interaksi antar objek atau komponen (dengan mengabaikan apakah ketentuan-ketentuan interaksi dikodekan secara formal atau secara informal) biasanya sulit untuk dikodekan dalam objek.

Arsitektur objek yang kacau terjadi ketika sesuatu dalam aplikasi harus lakukan memerlukan keterlibatan dari banyak objek berbeda [CLI03]. Pada saat diberikan spesifikasi kebutuhan, Pemodelan Berorientasi Objek (OOD) seringkali memprosesnya dengan cara menentukan objek-objek yang diperlukan untuk mendukung terpenuhinya kebutuhan. Tujuannya adalah untuk membagi fungsionalitas menjadi beberapa bagian sebagaimana yang sudah ditentukan oleh analisis suatu kebutuhan sampai didapatkan abstraksi yang mencukupi. Melalui proses ini, desainer/perancang seringkali melupakan kenyataan bahwa pada akhirnya nanti, objek-objek ini harus saling berhubungan satu sama lain dalam rangka memenuhi kebutuhan dengan fungsionalitas nyata. Hal ini terjadi dengan mudah dan tanpa disadari karena bagaimana cara suatu kebutuhan berinteraksi seringkali sangat berbeda dengan bagaimana cara suatu objek berinteraksi. Konsep yang diuraikan dalam pola rancangan merupakan suatu usaha untuk memahami isu-isu pada interaksi objek dan untuk menyediakan desainer/perancang, suatu pemahaman yang baik dan cara yang bagus untuk menyelesaikan masalah ketergantungan objek. Pada kenyataannya, implementasi utama pada bahasa AOP, dalam hal ini adalah AspectJ, sangat mempercayakannya pada Observer pola desain (*Observer design pattern*). Namun, salah satu kritikan yang dibuat oleh pada pelopor AOSD adalah, disamping ada banyak pola model (dan akan bertambah), pola-pola ini seringkali dimodelkan terlalu awal pada saat proses desain. Akibatnya, desain harus dikerjakan berulang-ulang, dan yang lebih buruk lagi, implementasi harus dikerjakan lagi ketika pola desain yang berbeda diperlukan.

Kritikan yang lain adalah interaksi antara pola desain dan struktur sistem. Suatu pola desain tidak hanya mempengaruhi arsitektur sistem, akan tetapi pola desain itu sendiri dipengaruhi oleh arsitektur sistem. Akibatnya, pola desain kehilangan modularitasnya sebagaimana modularitas tersebut juga hilang dalam tahap implementasi. Hal ini seringkali mempengaruhi re-*usability*, sebagaimana pola model (komponen yang bisa digunakan kembali) dan model objek (komponen lain yang bisa digunakan kembali) menjadi satu dalam komponen yang lebih kaku dan mempunyai tingkat re-*usability* yang rendah.

Ada dua jenis aspek, yaitu [SAU04]:

1. aspek pembangunan/pengetesan (*development/testing*), contohnya untuk masalah *debugging*, pencatatan (*logging*).
2. aspek produksi, contohnya masalah keamanan (*security*), tranksasi (*transaction*).

Kapan kita harus menggunakan aspek adalah pada waktu kita mengetahui bahwa kita mempunyai *concerns* yang melaukan *crosscutting* terhadap aplikasi/sistem secara keseluruhan dengan fungsionalitas yang mirip/serupa.

Aspek mempunyai beberapa kesamaan dengan kelas, yaitu dalam hal:

1. aspek dan kelas sama-sama bisa dimasukkan dalam suatu paket (*package*)
2. keduanya mempunyai atribut dan operasi [STE02]

Adapun perbedaan antara keduanya adalah sebagai berikut:

1. kita tidak bisa meng-instans aspek dengan menggunakan *new*. Aspek diinstans secara otomatis ketika *join point* dieksekusi [SAU04].
2. sub-aspek pada AspectJ mewarisi semua fitur dari super-aspeknya.

Beberapa hal lain yang bisa dilakukan dengan menggunakan aspek:

1. aspek bisa menambahkan *fields* ke dalam objek, hal ini dilakukan pada saat *compile time*
2. aspek bisa mendapatkan warisan/mewarisi dari satu sama lain.

Desain perangkat lunak merupakan langkah penting dalam siklus pengembangan perangkat lunak. Desain berorientasi objek bisa memodelkan perilaku (*behavior*) umum dengan baik, akan tetapi tidak bisa memodelkan perilaku (*behavior*) yang menyebar pada banyak kelas. Kebutuhan *crosscutting* seperti persistensi, keamanan (*security*) yang tersebar pada seluruh siklus pengembangan perangkat lunak menyebabkan kurang bersihnya suatu desain. Pemrograman berorientasi aspek mengatasi hal ini pada tahap pemrograman. Kurangnya dukungan pada tahap desain menyebabkan adanya ruang pemisah antara desain dan implementasi yang mengakibatkan hasil akhir suatu pengembangan perangkat lunak yang tidak begitu bagus. Oleh karena itu pemisahan *concerns* juga harus dilakukan pada tahap desain.

Crosscutting concerns pada tahap desain digunakan untuk memperbaiki pemisahan *concern* pada awal siklus pengembangan perangkat lunak. Hal ini

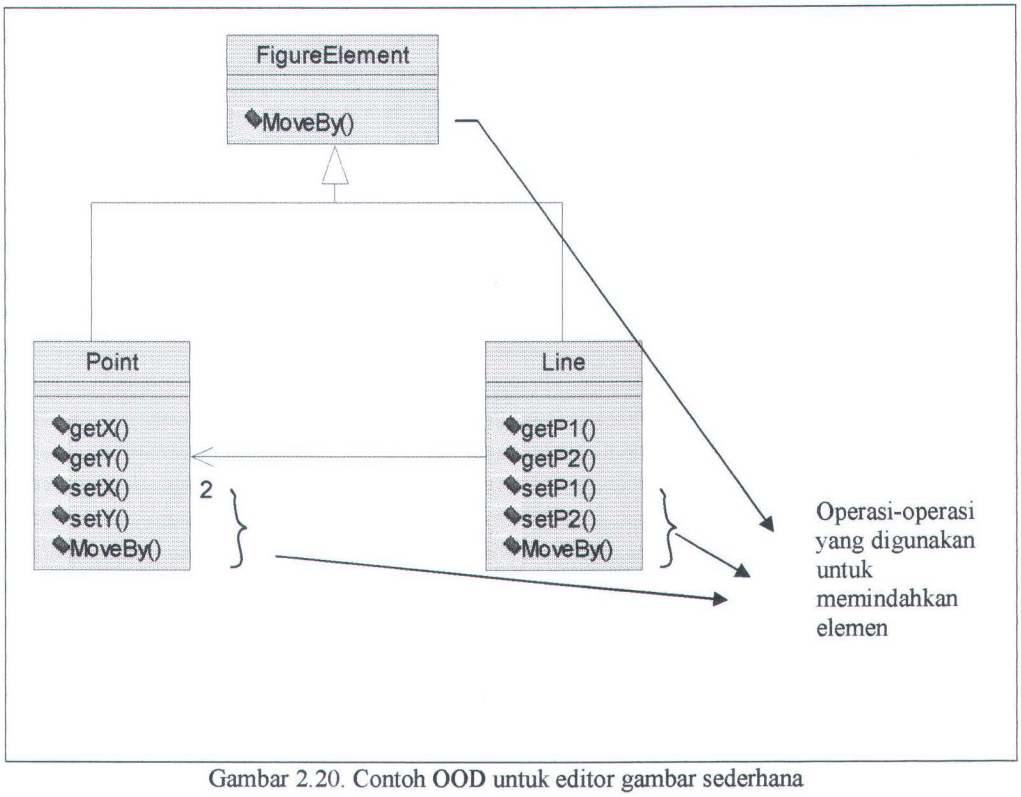
merupakan langkah menuju standarisasi UML pada bagaimana menentukan aspek pada tahap desain dalam pengembangan perangkat lunak berorientasi aspek.

Ekstensi untuk aspek pada UML tidak akan merubah spesifikasi metamodelnya, karena hanya menggunakan mekanisme ekstensi standar (seperti *stereotype*, *tagged value*, dan *constraints*) [GRO03].

2.5. Uji Coba yang Pernah Dilakukan Terhadap Keunggulan AOD

Berdasarkan uji coba yang pernah dilakukan terhadap AOD, didapatkan kesimpulan bahwa keunggulan AOD terletak pada pemisahan *concerns*[LAD02]. Pemisahan *concerns* ini bertujuan untuk menangani masalah *crosscutting concerns*, yaitu *concerns* yang menyebar pada sistem yang mengakibatkan *code tangling*(kode program yang menyebar pada satu modul) dan atau *code scattering*(kode program yang menyebar pada beberapa modul) sehingga rancangan suatu kelas menjadi tidak bersih. Suatu kelas jika terdapat *code tangling/code scattering* di dalamnya maka kelas tersebut akan kehilangan modularitasnya. Dimana suatu kelas seharusnya mencerminkan isi dari kelas tersebut, oleh karena ada *code tangling/code scattering*, kelas tersebut tersisipi oleh fungsi-fungsi/concerns yang melakukan *crosscutting* terhadap sistem.

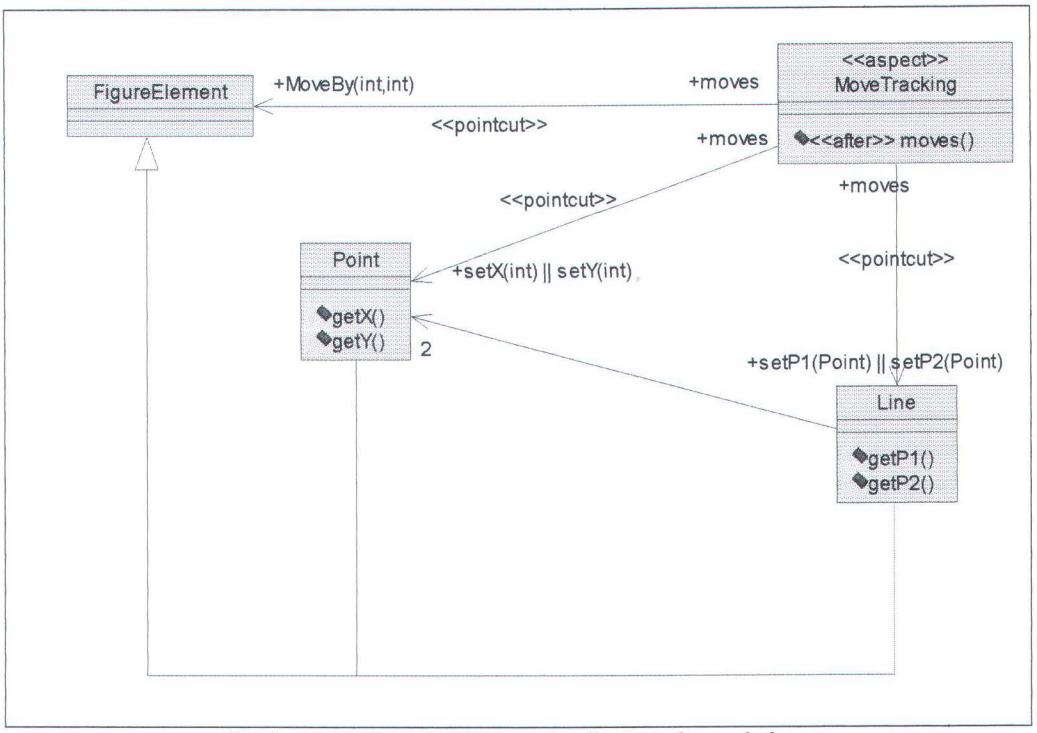
Untuk membuktikan keunggulan AOD, perlu dilakukan uji coba terhadap OOD dan AOD supaya perbedaan yang didapat bisa digunakan untuk menyimpulkan apa saja keunggulan AOD. Contoh uji coba yang dilakukan pada rancangan editor gambar sederhana untuk OOD ditunjukkan pada gambar 2.20. [CLE04].



Gambar 2.20. Contoh OOD untuk editor gambar sederhana

Pada gambar 2.20. di atas, operasi setX(int), setY(int), setP1(Point), dan setP2(Point) dikatakan saling melakukan *crosscutting* karena perubahan pada masing-masing kelas akan mempengaruhi kelas lain. Operasi-operasi tersebut digunakan untuk memindahkan/mengatur elemen.

Rancangan di atas mempunyai operasi-operasi yang melakukan *crosscut ting* terhadap sistem sehingga terjadi *code tangling/code scattering*. Untuk mengatasi hal itu, maka dibuat sebuah aspek yang menampung *concerns* yang melakukan *crosscutting* terhadap sistem. Rancangan untuk AOD terlihat pada gambar 2.21.



Gambar 2.21. Contoh AOD untuk editor gambar sederhana

Pada gambar 2.21. diatas terlihat bahwa sekarang kelas Line dan kelas Point bersih dari *code tangling*. Dan rancangan yang baru ini mempunyai modularitas dan tingkat *reusability* yang tinggi karena tidak ada *concerns* atau fungsi-fungsi yang melakukan *crosscutting* terhadapnya.

Pembangunan Perangkat Lunak Berorientasi Aspek

- Tahap analisis dan perancangan

  Pada tahap ini yang harus dilakukan adalah menganalisis dan mendekomposisi semua kebutuhan/*concerns*, dimana *concerns* yang umum(*concerns* yang mewakili komponen fungsional yang sudah modular) dikelompokkan menjadi kelas, sedangkan *concerns* yang khusus(*concerns* yang melakukan *crosscutting* terhadap sistem)

dimodulkan dalam aspek[LAD02]. Sebuah aspek digunakan untuk mengimplementasikan sebuah *concern* khusus. *Concerns* hasil dekomposisi tersebut kemudian diimplementasikan menjadi unit-unit individual, dimana implementasi kelas terdiri dari nama kelas, operasi, dan atribut, sedangkan implementasi aspek terdiri dari komponen nama, stereotype *aspect*, operasi, dan atribut. Kemudian semua aspek dan kelas direkomposisi untuk membentuk rancangan lengkap suatu sistem perangkat lunak.

- Tahap pemrograman

  Pada tahap ini AOP memudahkan programmer dalam menuliskan kode-kode program yang berulang dalam sistem, karena kode-kode program tersebut dikelompokkan dalam modul aspek, sehingga mengurangi jumlah kode yang harus ditulis. Dalam hal ini pemrograman dikerjakan dengan menggunakan AspectJ.

  Kode AOP lebih termodularisasi dan mempunyai tingkat *reusability* yang tinggi dari pada versi OOP. Kode AOP juga lebih fleksibel dalam hal penambahan method-method tanpa merubah desain atau kode, dan tidak berisi kode-kode yang menyebar di seluruh aplikasi sistem. [MIC03]

  Contoh:

  Berdasarkan desain pada gambar 1 diatas, maka kode untuk kelas2 tersebut adalah:

```
class Line {
   private Point _p1, _p2;
   Point getP1() { return _p1; }
   Point getP2() { return _p2; }
   void setP1(Point p1) {_p1 = p1;}
```

```
   void setP2(Point p2) {_p2 = p2;}
}
class Point {
   private int _x = 0, _y = 0;
   int getX() { return _x; }
   int getY() { return _y; }
   void setX(int x) {_x = x;}
   void setY(int y) {_y = y;}
}
```

Selanjutnya muncul kebutuhan untuk menampilkan setiap perubahan yang terjadi pada masing-masing elemen point maupun line.

Di bawah ini diberikan dua solusi, yang pertama pemrograman tanpa menggunakan AspectJ, dan yang kedua menggunakan AspectJ.

Tanpa AspectJ

```
class Line {
   private Point _p1, _p2;
   Point getP1() { return _p1; }
   Point getP2() { return _p2; }
   void setP1(Point p1) {
         _p1 = p1;
         MoveTracking.collectOne(this);
   }
   void setP2(Point p2) {
         _p2 = p2;
         MoveTracking.collectOne(this);
   }
}
class Point {
   private int _x = 0, _y = 0;
   int getX() { return _x; }
   int getY() { return _y; }
   void setX(int x) {
         _x = x;
         MoveTracking.collectOne(this);
   }
   void setY(int y) {
         _y = y;
         MoveTracking.collectOne(this);
   }
}

class MoveTracking {
   private static Set _movees = new HashSet();
   public static void collectOne(Object o) {
         _movees.add(o);
   }
   public static Set getmovees() {
         Set result = _movees;
```

```
        _movees = new HashSet();
        return result;
    }
}
```

Dengan AspectJ

```
class Line {
    private Point _p1, _p2;
    Point getP1() { return _p1; }
    Point getP2() { return _p2; }
    void setP1(Point p1) {
        _p1 = p1;
    }
    void setP2(Point p2) {
        _p2 = p2;
    }
}

class Point {
    private int _x = 0, _y = 0;
    int getX() { return _x; }
    int getY() { return _y; }
    void setX(int x) {
        _x = x;
    }
    void setY(int y) {
        _y = y;
    }
}

aspect MoveTracking {
    private Set _movees = new HashSet();
    public Set getmovees() {
        Set result = _movees;
        _movees = new HashSet();
        return result;
    }

    pointcut moves(FigureElement figElt):
    instanceof(figElt) &&
    (receptions(void Line.setP1(Point)) ||
    receptions(void Line.setP2(Point)) ||
    receptions(void Point.setX(int)) ||
    receptions(void Point.setY(int)));

    after(FigureElement fe): moves(fe) {
        _movees.add(fe);
    }
}
```

Pada kode program di atas terlihat bahwa pada kode program yang tidak menggunakan AspectJ terdapat penulisan kode yang berulang pada kelas Point

BAB III

ANALISIS DAN PERANCANGAN SISTEM KEAMANAN

Grace Copy Center

BAB III

ANALISIS DAN PERANCANGAN SISTEM KEAMANAN

Pada Bab ini akan dibahas mengenai notasi untuk istilah-istilah dalam pemodelan berorientasi aspek, analisis dan perancangan pada studi kasus sistem keamanan login dengan menggunakan mekanisme pemodelan berorientasi objek, memodelkan sistem keamanan login menggunakan pemodelan berorientasi aspek, serta perbandingan antara pemodelan berorientasi objek dan pemodelan berorientasi aspek.

3.1 Notasi untuk Istilah-istilah dalam AOD

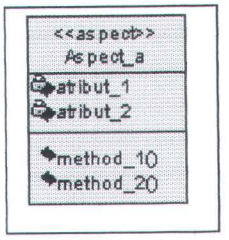
Istilah-istilah yang terdapat pada pemodelan berorientasi aspek meliputi *concern*, *crosscutting concerns*, aspek, *join point*, *pointcut*, *advice*, dan *intertype declaration* seperti yang telah dijelaskan pada bab sebelumnya. Dari ketujuh istilah terssebut, tidak semuanya yang memiliki notasi untuk dapat dimodelkan ke dalam UML.

Beberapa istilah yang diusulkan bentuk notasinya antara lain:

1. Aspek

   Notasi aspek dimunculkan pada saat pembuatan *class diagram* dengan cara mengisi *stereotype*-nya dengan "aspect". Aspek muncul karena adanya *concern*. Saat ditemukan adanya *concern* dalam suatu sistem, maka saat itu pula suatu aspek dibangun.

   Contoh notasi aspek pada *class diagram*:



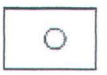
Gambar 3.1. Contoh notasi aspek pada *class diagram*

2. Join point

Notasi untuk *join point* muncul pada saat pembuatan *sequence diagram* dan *class diagram*.

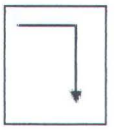
Terdapat dua notasi yang berbeda untuk *join point*, yaitu *join point* untuk penambahan *method*/operasi pada kelas lain dan *join point* untuk *method* yang melakukan *crosscutting* terhadap sistem.

Notasi *join point* untuk *method* yang melakukan *crosscutting* terhadap sistem pada *sequence diagram*:



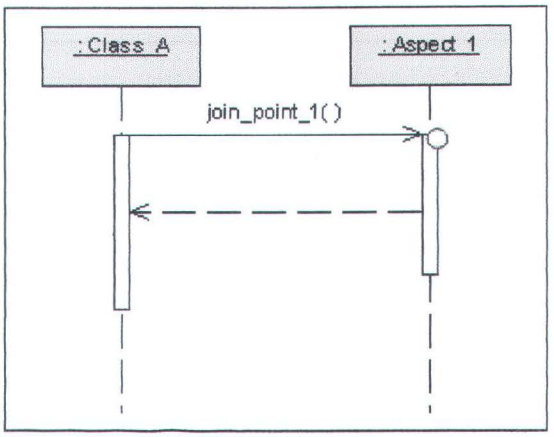
Gambar 3.2. Notasi *join point* untuk *method* yang melakukan *crosscutting* terhadap pada *sequence diagram*

Notasi *join point* untuk penambahan *method* pada kelas kelas lain pada *sequence diagram*, notasinya adalah:



Gambar 3.3. Notasi *join point* untuk penambahan *method* pada *sequence diagram*

Contoh notasi *join point* untuk *method* yang melakukan *crosscutting* terhadap sistem pada *sequence diagram*:

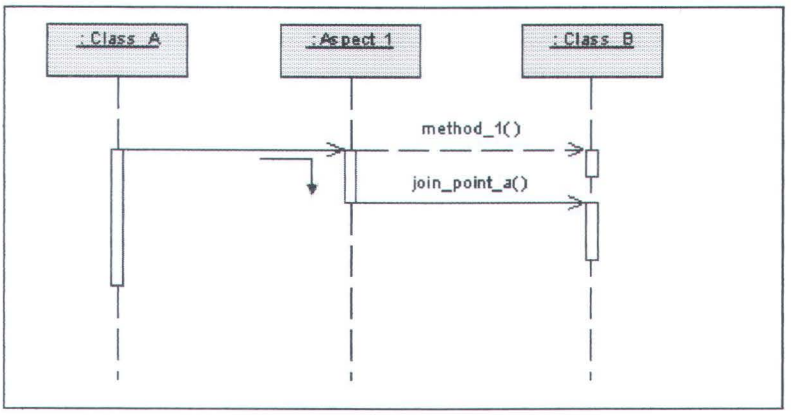


Gambar 3.4. Contoh penerapan notasi *join point* untuk *method* yang melakukan *crosscutting* terhadap sistem, pada *sequence diagram*

Keterangan:

Pada *sequence diagram* di atas, *join point* join_point_1() merupakan *concern* yang melakukan *crosscutting* terhadap sistem, yang kemudian ditempatkan dalam suatu aspek Aspect_1.

Contoh notasi *join point* penambahan *method* pada kelas lain, pada *sequence diagram*:



Gambar 3.5. Contoh penerapan notasi *join point* untuk penambahan *method*, pada *sequence diagram*

Keterangan:

Pada *sequence diagram* di atas, pemanggilan *method* method_1() pada kelas Class_B ditentukan oleh *join point* join_point_a pada aspek Aspect_1.

Pada class diagram, *join point* muncul pada detail relasi antara aspek dan kelas. Disini, notasi untuk *join point*, *pointcut*, *advice*, dan *intertype declaration* merupakan satu kesatuan, sehingga untuk contohnya akan diberikan setelah penjelasan *pointcut*, *advice*, dan *intertype declaration* di bawah ini.

3. Pointcut

   Notasi untuk *pointcut* muncul pada saat pembuatan *class diagram* sebagai *stereotype* relasi antara aspek dan kelas. Untuk contohnya, akan diberikan setelah penjelasan *advice* dan *intertype declaration*.

4. Advice

   Notasi untuk *advice* muncul pada saat pembuatan *class diagram* dengan cara mengisi *stereotype*: "before", "after", atau "around". *Advice* muncul untuk memberikan jenis-jenis perilaku *pointcut* untuk mengeksekusi *join point*. Contoh untuk notasi *advice* pada *class diagram* akan diberikan setelah penjelasan *intertype declaration*.

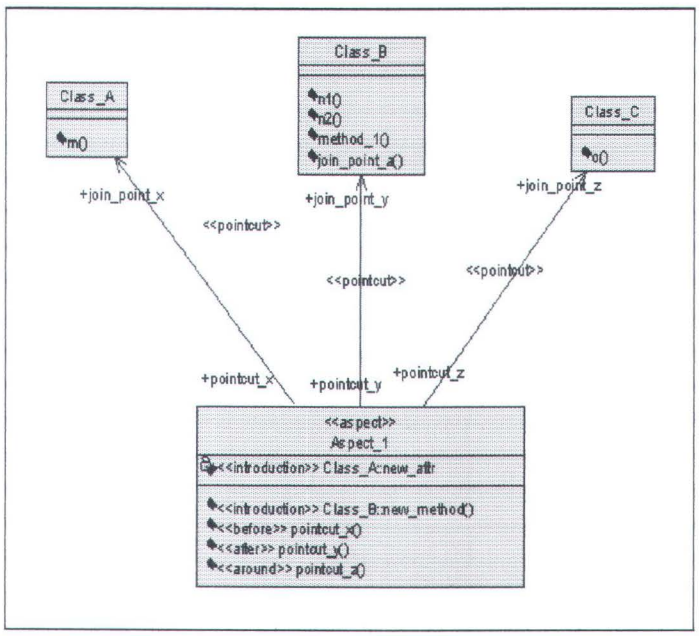
5. Intertype declaration

   Notasi untuk *intertype declaration* muncul pada saat pembuatan *class diagram* dengan cara mengisi *stereotype* "introduction" pada setiap *method* yang akan ditambahkan pada kelas tertentu. *Intertype declaration* harus dituliskan di dalam modul aspek.

Sintaknya adalah:

<<introduction>>[nama_kelas][nama_method/nama_atribut]

Contoh notasi untuk *join point*, *pointcut*, *advice*, dan *intertype declaration* pada *class diagram*.



Gambar 3.6. Contoh penerapan notasi *join point*, *pointcut*, *aspek*, *advice*, dan *intertype declaration* dalam *class diagram*

Keterangan:

- Pointcut *pointcut_x* pada aspek *Aspect_1* menunjukkan tempat eksekusi pada *join_point_x* pada kelas *class_A* dengan tipe advice *<<before>>*.
- Pointcut *pointcut_y* pada aspek *Aspect_1* menunjukkan tempat eksekusi pada *join_point_y* pada kelas *class_B* dengan tipe advice *<<after>>*.
- Pointcut *pointcut_z* pada aspek *Aspect_1* menunjukkan tempat eksekusi pada *join_point_z* pada kelas *class_C* dengan tipe advice *<<around>>*.
- Inter-type declaration *<<introduction>> Class_A::new_attr* pada aspek *Aspect_1* untuk penambahan atribut baru pada kelas Class_A.

MILIK PERPUSTAKAAN
INSTITUT TEKNOLOGI
SEPULUH - NOPEMBER

- Inter-type declaration *<<introduction>> Class_B::new_method* pada aspek *Aspect_1* untuk penambahan method baru pada kelas *Class_B*.

Untuk istilah *concern* dan *crosscutting concern* tidak ada notasinya. Kedua istilah tersebut hanya sebatas pendefinisian saja, karena sebenarnya secara tidak langsung, keduanya sudah tercerminkan/diakui keberadaannya saat pembuatan pemodelan berorientasi aspek. Aspek muncul jika ada *concern* dan *crosscuting concern*.

3.2 Studi Kasus pada Sistem login

Kemajuan teknologi semakin lama berkembang semakin pesat. Seiring dengan hal itu, ada isu penting yang harus diselesaikan untuk membuat proses pembangunan perangkat lunak menjadi lebih praktis, yaitu masalah tingkat *re-usability* suatu model. Model yang bagus adalah model yang mempunyai tingkat *re-usability* tinggi, sehingga bisa digunakan dalam sistem/aplikasi lain yang mempunyai karakteristik yang mirip.

Dalam membuat model yang bagus, ada permasalahan yang harus diselesaikan yaitu masalah penanganan *crosscutting concerns*. *Crosscutting concern* muncul pada kebutuhan *system-level* yang berupa: *logging*, *authentication*, *security*, *performance*, *error checking*, *multythread savety*, dll. Kebutuhan sistem-level kadang saling mempengaruhi satu sama lain, mempengaruhi beberapa subsistem, atau bahkan ada kemungkinan untuk mempengaruhi jalannya sistem secara keseluruhan.

Pemodelan berorientasi objek kurang bisa menangani permasalahan pemodelan dengan adanya kasus *crosscutting concern*. Pemodelan terhadap kebutuhan *system-level* biasanya disisipkan pada kelas-kelas yang membutuhkan saja, akibatnya pemodelan kelas-kelas tersebut menjadi tidak bersih, atau kadang dengan adanya sisipan tersebut malah merusak relasi antar kelas.

Salah satu alternatifnya adalah melakukan pemodelan berorientasi aspek. Disini diambil studi kasus sistem keamanan login, karena sistem keamanan merupakan salah satu *concern* pada kebutuhan *system-level* yang melakukan *crosscutting* terhadap sistem. Selain itu karena hampir semua aplikasi membutuhkan sistem keamanan login, maka muncul suatu kebutuhan adanya model sistem keamanan login yang mempunyai tingkat *re-usability* yang tinggi.

Karakteristik lain pada sistem keamanan yang membuat sistem keamanan ini dijadikan studi kasus yaitu karena sistem keamanan selalu mengikuti perubahan kebijakan keamanan akibat dari perkembangan teknologi. Perubahan kebijakan ini dimaksudkan untuk membuat sistem keamanan yang lebih handal lagi. Dengan adanya perubahan kebijakan tersebut, memungkinkan adanya penambahan fungsi-fungsi pada sistem, dan otomatis, akan dilakukan perubahan terhadap modelnya untuk menambah operasi (*method*) dan atribut pada kelas-kelas yang bersangkutan.

Sistem keamanan pada perangkat lunak mempunyai arti yang sangat luas dan jenisnya bisa bermacam-macam. Pada Tugas Akhir ini akan dibahas mengenai sistem keamanan perangkat lunak yang diterapkan pada sistem login pada aplikasi web SOP (*Standard Operation Procedure*).

SOP (*Standard Operation Procedure*) merupakan struktur normal pada sebuah perusahaan untuk menangani kegiatan operasional yang spesifik. SOP adalah elemen pendukung yang penting pada pembentukan kontrol dan perencanaan yang bagus di dalam sebuah perkembangan perusahaan. SOP yang dihubungkan dengan sebuah proyek tidak hanya mengacu pada prosedur untuk mengoperasikan fasilitas yang telah dibangun, dengan satu jenis operasi saja, melainkan SOP juga berisi prosedur untuk pembayaran, pengetesan, pemasangan pada peralatan-peralatan tertentu dan sejenisnya.

SOP akan membantu membuat perusahaan/organisasi lebih efisien dan produktif secara substansial, namun SOP memerlukan evaluasi dan pengembangan yang berkelanjutan untuk menjamin bahwa SOP dan perusahaan terintegrasi dengan semua operasi yang ada didalamnya, akan tetapi ingat juga relevansinya dan kegunaannya untuk semua karyawan perusahaan.

Segala sesuatu atau pekerjaan yang dilakukan masing-masing karyawan dalam perusahaan/organisasi/institusi harus berdasarkan SOP yang telah dibuat. *Standard Operation Procedures* (SOPs) yang telah dibuat templatenya, dimasukkan ke situs web dengan tujuan bisa diakses oleh semua staff, khususnya staff-staff yang bertugas menjalankan SOPs tersebut. Dalam melaksanakan perkerjaannya, masing-masing staff tinggal mengikuti prosedur yang ada dalam SOP.

Dalam SOP, user dibedakan menjadi tiga role, yaitu administrator, *owner* (pemilik/pembuat SOP), dan *viewer*(user lainnya). Satu user bisa mempunyai

Owner merupakan user yang mempunyai hak untuk membuat satu/lebih SOP (tergantung kebutuhan). Satu SOP bisa dimiliki/dibuat oleh satu atau beberapa orang *owner*. Seorang *owner* memiliki hak penuh terhadap SOP, tetapi terbatas hanya pada SOP yang dibuatnya saja. Hak-hak tersebut antara lain, hak untuk membuat SOP yang baru, menghapus SOP yang pernah dibuatnya, melakukan perubahan terhadap SOPnya. Seorang *owner* biasanya adalah seorang kepala bidang, dimana dia harus membuat suatu prosedur pengerjaan tugas yang harus dilakukan oleh semua anak buahnya.

Viewer merupakan user yang hanya mempunyai hak untuk melihat SOP yang sesuai dengan pekerjaannya dan prosedur pengerjaan yang harus dilakukannya.

Skenario sistem login pada aplikasi web SOP.

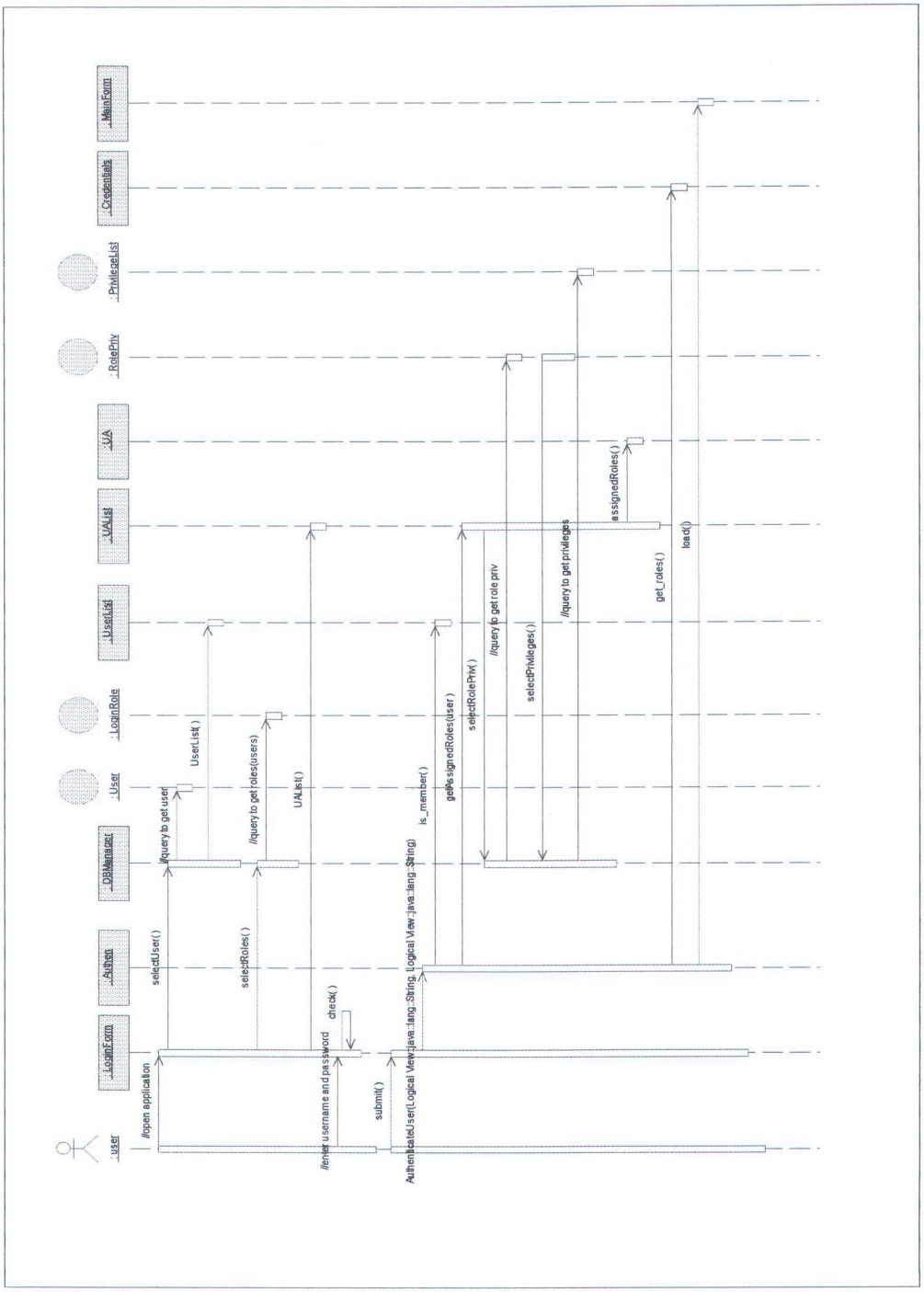
Tabel 3.1. Skenario login pada aplikasi web SOP

Login System	
Aktor	User, yang terdiri dari administrator, viewer, dan owner
Tujuan	Menampilkan menu utama
Ringkasan	User memasukkan Username dan Password dan ia akan masuk ke menu utama
Tipe	Primer
Masukan	Username & Password
Keluaran	Menu Utama Tampil
Kondisi Awal	User berada pada halaman login
Kondisi Akhir	User Berada pada Halaman utama
Urutan Aksi normal	
Aksi User	Reaksi Sistem
1. User membuka aplikasi yang berupa halaman login.	2. Halaman login meminta DBManajer untuk mendapatkan semua nama user yang telah terdaftar. 3. DBManajer melakukan query untuk mendapatkan daftar user
4. User memasukkan username dan password.	5. Sistem memeriksa apakah input yang dimasukkan user sudah sesuai dengan kebutuhan

	(*requirement*), misalnya untuk input username tidak boleh ada karakter special seperti (! @ # $ % ^ & * < > ? , . /)
6. User menekan tombol Submit.	7. Sistem memeriksa apakah input yang dimasukkan user yang berupa username dan password sesuai dengan data yang terdapat pada sistem. 8. Jika input yang dimasukkan sesuai, maka sistem akan membuka MainForm.
Urutan aksi alternatif	
Aksi User	Reaksi Sistem
8. Jika user tersebut tidak terdaftar dalam data sistem, maka akan muncul pesan error. 9. Kembali ke Aksi Normal No.1 10. Kembali ke halaman login.	

Untuk lebih jelasnya mengenai alur diatas, bisa dilihat pada *sequence diagram* di bawah ini.

3.2.2 Sequence Diagram



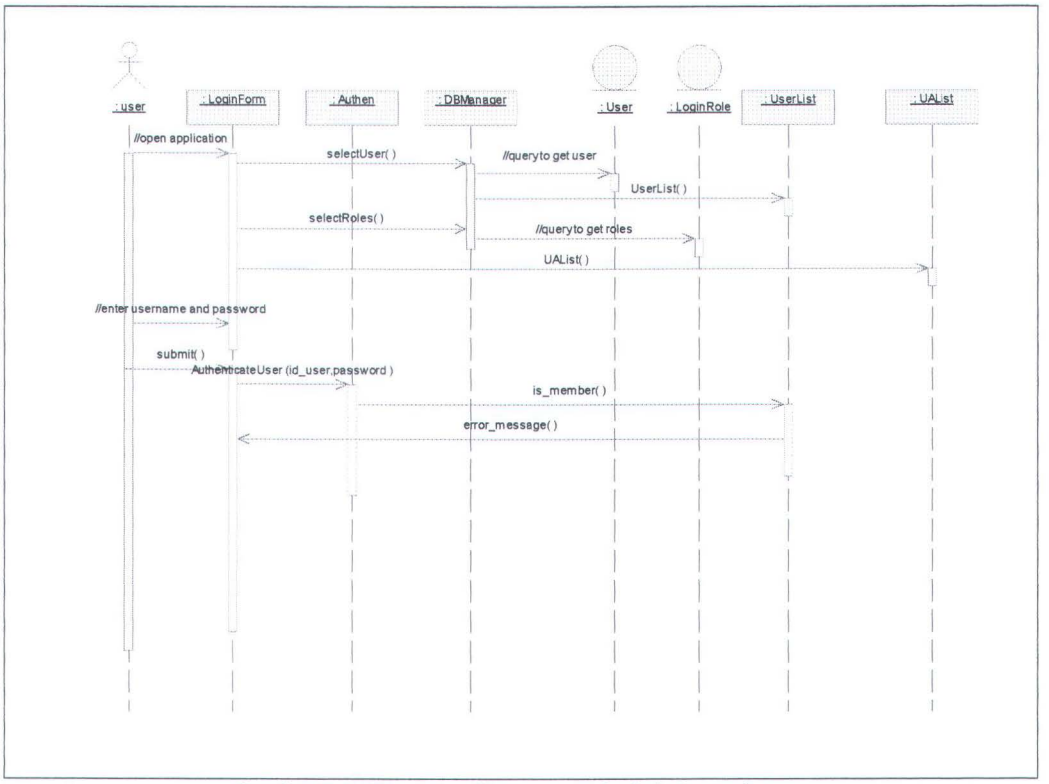
Gambar 3.8. Sequence diagram pada OOD untuk aksi normal

Penjelasan untuk sequence diagram di atas:

1. User membuka LoginForm
2. Halaman login meminta DBManajer untuk mendapatkan semua nama user yang telah terdaftar.
3. DBManajer melakukan query pada database User untuk mendapatkan daftar semua user.
4. Daftar yang didapat akan disimpan dalam class UserList.
5. Halaman login meminta DBManajer untuk mendapatkan daftar roles yang ada pada database LoginRole.
6. Daftar yang didapat akan disimpan dalam class UAList.
7. User memasukkan username dan password.
8. Sistem memeriksa input yang dimasukkan user dengan fungsi check().
9. User menekan tombol Submit.
10. Class authen memeriksa apakah user tersebut ada dalam daftar class UserList atau tidak.
11. User meminta privilege melalui class UAList getAssignedRoles() terlebih dahulu untuk kemudian DBManajer melakukan query pada RolePriv untuk mendapatkan id_role dan id_privileges nya. Setelah itu meng-query lagi PrivilegeList untuk mendapatkan daftar privilege sesuai dengan id_privilege yang ada pada RolePriv.
12. Setelah user selesai diautentikasi, akan dibuat object Credential. Object credentials terdiri dari atribut-atribut privilege yang diminta user. User bisa mempunyai lebih dari satu object credential.

13. Setelah didapatkan privilegenya, sistem akan membuka MainForm.

Di bawah ini merupakan sequence diagram yang menggambarkan aksi alternatif.

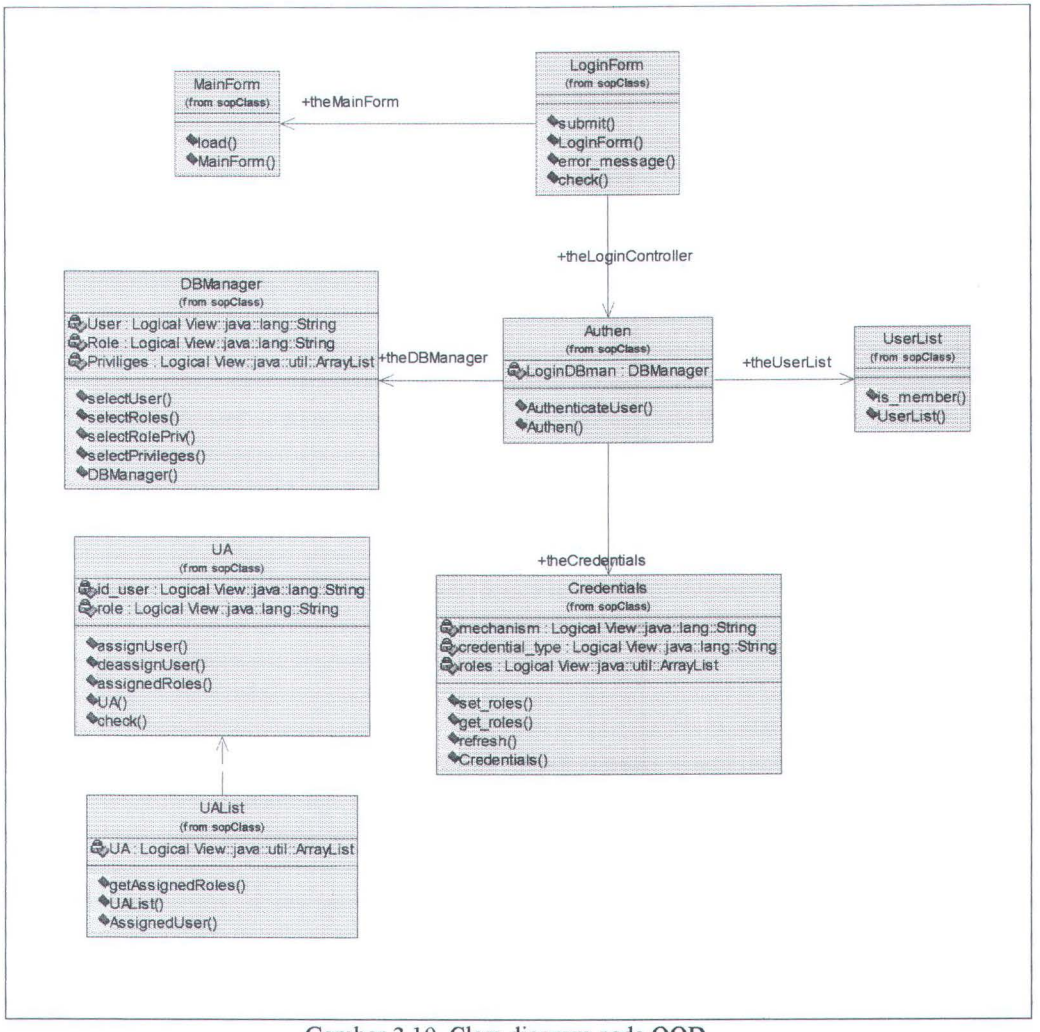


Gambar 3.9. Sequence diagram untuk aksi alternative

Penjelasan untuk *sequence diagram* di atas:

Untuk langkah pertama sampai dengan langkah kesepuluh sama seperti penjelasan pada *sequence diagram* untuk aksi normal. Setelah itu apabila pada kelas UserList, input yang dimasukkan user tidak sesuai dengan data yang ada, maka akan muncul pesan error pada halaman login tersebut.

3.2.3 Class Diagram



Gambar 3.10. Class diagram pada OOD

Penjelasan properti masing-masing kelas pada *class diagram* di atas.

Tabel 3.2. Operasi dan atribut pada sistem login OOD serta keterangannya

No.	Nama Kelas	Operasi	Atribut	Keterangan
1.	MainForm	Load()		Fungsi load() ini digunakan untuk membuka/meng-load halaman MainForm setelah tombol submit pada halaman LoginForm ditekan. Fungsi load() hanya akan dijalankan apabila user memasukkan inputan username dan password dengan benar.
2.	LoginForm	submit()		Fungsi submit() ini digunakan

				untuk menjalankan perintah terhadap aplikasi untuk memeriksa apakah username dan password yang dimasukkan user sesuai dengan data yang ada atau tidak. Jika sesuai maka user bisa masuk dalam halaman MainForm. Jika tidak, user akan tetap berada pada halaman LoginForm.
		error_message( )		Fungsi ini digunakan untuk menangkap inputan username dan atau password yang salah.
		check()		Fungsi ini digunakan untuk memeriksa apakah input yang dimasukkan user sudah sesuai dengan requirement yang dibutuhkan atau tidak.
3.	DBManajer	selectUser()		Fungsi selectUser() ini digunakan untuk memberikan perintah kepada kelas DBManajer untuk melakukan query pada tabel User sehingga didapatkan daftar seluruh user yang ada pada aplikasi web SOP.
		selectRoles()		Fungsi selectRoles() ini digunakan untuk memberikan perintah kepada kelas DBManajer untuk melakukan query pada tabel LoginRole sehingga didapatkan daftar seluruh role yang ada pada aplikasi web SOP.
		selectRolePriv		Fungsi selectRolePriv() ini digunakan untuk memberikan perintah kepada kelas DBManajer untuk melakukan query pada tabel RolePriv sehingga didapatkan daftar privilege apa saja yang ada pada suatu role.
		selectPrivileges		Fungsi selectPrivileges() ini digunakan untuk memberikan perintah kepada kelas DBManajer untuk melakukan query pada tabel PrivilegeList sehingga didapatkan daftar privilege yang dimiliki oleh user yang login pada saat itu.
			user	Atribut user ini digunakan untuk menyimpan seluruh data user yang ada, pada saat user membuka halaman LoginForm.
			role	Atribut role ini digunakan untuk

				menyimpan seluruh daftar role yang ada pada aplikasi web SOP, pada saat user membuka halaman LoginForm.
			privilege	Atribut privilege ini digunakan untuk menyimpan semua daftar privilege user tertentu yang login pada saat itu.
4.	Authen	AuthenticateUser()		Fungsi AuthenticateUser() ini digunakan untuk memeriksa apakah user yang login pada saat itu memang benar-benar user yang berhak.
5.	UserList	is_member()		Fungsi is_member() ini digunakan untuk memeriksa apakah username dan password yang dimasukkan user ada pada daftar user yang terdapat pada kelas UserList.
6.	UA	assignedRoles()		Fungsi ini digunakan untuk mendapatkan semua *roles* yang diberikan pada user yang login pada saat itu.
		assignUser()		Fungsi ini digunakan untuk memberikan hak pada user.
		deassignUser()		Fungsi ini digunakan untuk menghapus hak pada user.
		check()		Fungsi ini digunakan untuk memeriksa apakah input yang dimasukkan user sudah sesuai dengan *requirement* atau tidak.
			id_user	Atribut id_user digunakan untuk menyimpan daftar id_user yang ada pada aplikasi web SOP.
			role	Atribut role ini digunakan untuk menyimpan daftar role yang ada pada aplikasi web SOP.
7.	UAList	getAssignedRoles()		Fungsi getAssignedRoles() ini digunakan untuk mendapatkan semua role yang dimiliki user yang login pada saat itu dengan cara meminta DBManajer untuk melakukan query untuk mendapatkan data-data yang diinginkan.
			UA	-
8.	Credential	set_roles()		Fungsi set_roles() ini digunakan untuk mengembalikan semua role yang dimiliki user yang login pada saat itu.
		get_roles()		Fungsi get_roles() ini digunakan untuk mendapatkan semua role

				user yang login pada saat itu.
		refresh()		Fungsi refresh() ini digunakan untuk mengambalikan semua nilai yang dimiliki user pada saat setelah user melakukan refresh pada halaman tertentu pada aplikasi web SOP.
			mechanism	Atribut mechanism ini dimaksudkan untuk menyimpan jenis-jenis mekanisme login yang dilakukan oleh user. Misalnya, apakah user login dengan menggunakan password atau smart card.
			credential_type	-
			roles	Atribut role ini digunakan untuk menampilkan semua role yang dimiliki user yang login pada saat itu.

3.3 Pemodelan perangkat lunak berorientasi aspek pada sistem keamanan login pada aplikasi SOP.

Desain sistem keamanan diharapkan mempunyai tingkat fleksibilitas yang tinggi untuk mendukung kebijakan-kebijakan baru yang mungkin muncul nantinya seiring dengan perkembangan teknologi, sehingga memudahkan proses *maintenance* karena tidak perlu merubah desain secara keseluruhan. Pemodelan berorientasi aspek mengatasi permasalahan ini dengan memperkenalkan sebuah *stereotype* baru pada UML yang disebut sebagai aspek. Nantinya akan dimodelkan sebuah sistem keamanan login pada aplikasi SOP (*Standard Operation Procedure*) untuk menunjukkan peran penting suatu aspek dalam pemodelan menggunakan UML.

Skenario sistem keamanan login pada aplikasi web SOP (*Standard Operation Procedure*) yang menggunakan pemodelan berorientasi aspek ini sama

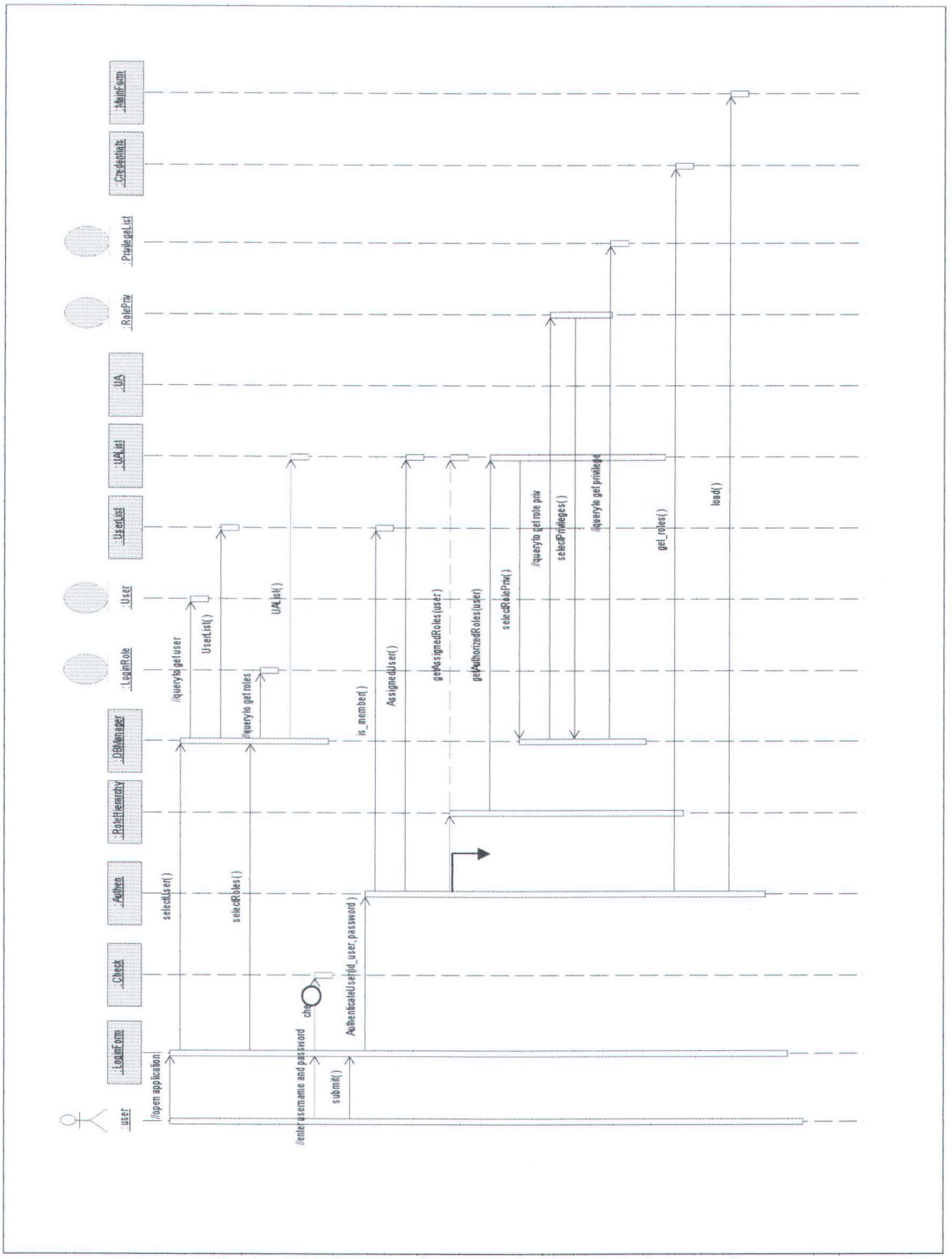
seperti skenario yang menggunakan mekanisme pemodelan berorientasi objek. Perancangan sistem keamanan login berorientasi aspek ini meliputi *sequence diagram*, dan *class diagram* karena untuk *use case diagram* sama seperti *use case diagram* untuk pemodelan berorientasi objek.

Analisis *concerns* pada desain login.

Beberapa criteria dalam menentukan sebuah aspek yang akan digunakan untuk mengenkapsulasi *crosscutting concerns* adalah:

1. sebuah aspek mengimplementasikan hanya satu concern, yang harus melakukan *crosscutting* terhadap concern utama (*main concern*) atau aspek yang lain.
2. aspek harus menangani fungsi-fungsi atau inter-type declaration yang akan melakukan *crosscutting* terhadap concern utama (*main concern*) atau aspek yang lain. Atribut-atribut dan operasi-operasi yang khusus pada kelas tertentu dan tidak melakukan *crosscutting* terhadap kelas-kelas yang lain harus didefinisikan pada kelas itu sendiri sebagai pengganti aspek.

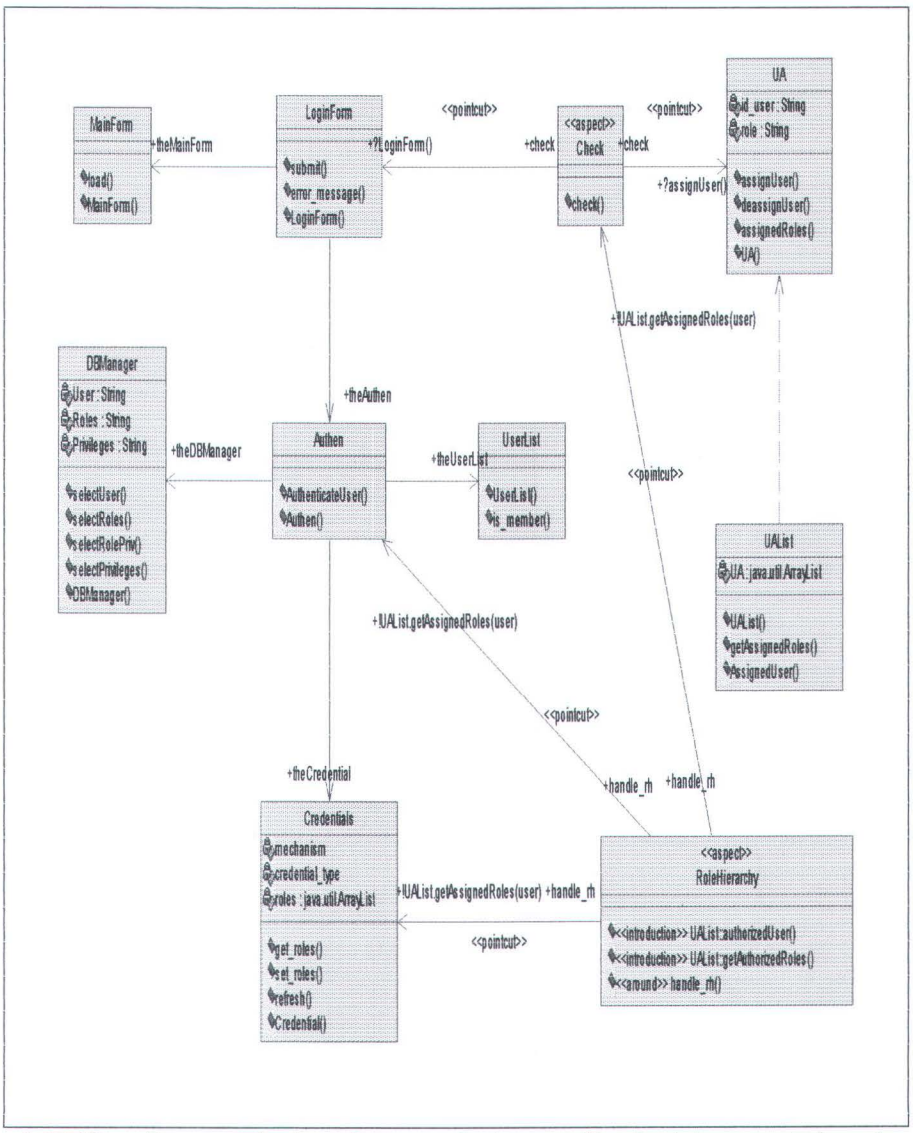
3.3.1 Sequence Diagram



Gambar 3.11. Sequence diagram pada AOD untuk aksi normal

Untuk sequence diagram aksi alternative pada pemodelan berorientasi aspek sama seperti pada pemodelan berorientasi objek.

3.3.2 Class Diagram



Gambar 3.12. Class diagram pada sistem keamanan login yang dilengkapi dengan notasi aspek

Pada gambar *class diagram* di atas terdapat dua aspek, yaitu aspek RoleHierarchy dan aspek Check. Aspek RoleHierarchy muncul untuk mendukung

pengaturan tingkatan *role* (*role hierarchy*) yang terdapat pada aplikasi SOP. Pada desain dasarnya terdapat method getAssignedRoles(user) pada kelas UAList, yang mengembalikan semua *roles* yang diberikan kepada user yang digunakan untuk menentukan hak akses user terhadap *resource*. Saat tingkatan *role* muncul, getAssignedRoles(user) tidak bisa mengembalikan semua *roles* yang dimiliki user, karena beberapa *roles* yang tidak diberikan bisa diwariskan.

Aspek RoleHierarchy pada gambar di atas berisi implementasi dari *concerns* RoleHierarchy. Di dalam aspek RoleHierarchy, beberapa *inter-type declaration* didefinisikan untuk menambahkan/memasukkan atribut dan method yang baru ke dalam kelas-kelas yang sudah ada sebelumnya. Hanya ada satu pointcut handle_rh dan satu *join point* !UAList.getAssignedRoles(user) yang didefinisikan. Pada saat *runtime*, pemanggilan method terhadap UAList.getAssignedRoles(user) diturunkan oleh Authen atau Credential akan men-trigger *pointcut* handle_rh. Kode tipe *advice* <<around>> yang didefinisikan untuk handle_rh akan dieksekusi pada method UAList.getAssignedRoles(user). Pada desain ini, kode advice akan memanggil UAList.getAuthorizedRoles(user) yang didefinisikan pada aspek yang sama.

Aspek yang kedua yaitu aspek Check. Aspek ini digunakan untuk mengumpulkan semua fungsi check() yang tersebar pada seluruh desain. Hal ini memberikan tingkatan modularitas yang lebih baik pada desain model.

Pada gambar *class diagram* di atas, aspek Check terdiri dari pointcut check dengan tipe *advice* <<before>>, dimana aspek tersebut direlasikan dengan dua kelas dan satu aspek yaitu aspek RoleHierarchy. Yang pertama direlasikan

dengan kelas LoginForm dengan join point LoginForm(). Yang kedua, aspek Check direlasikan dengan kelas UA dengan *join point* assignUser(). Eksekusi pada UA.assignUser() akan men-trigger pointcut Check untuk memeriksa apakah input yang dimasukkan untuk menjalankan fungsi UA.AssignUser() sudah sesuai dengan *requirement* yang ada atau belum.

3.4 Perbedaan antara pemodelan AOD dengan OOD pada sistem keamanan login pada aplikasi SOP.

Desain berorientasi objek kadang bisa memisahkan *concerns* dengan cara memisahkan *concerns* tersebut ke dalam objek. Akan tetapi, desain berorientasi objek hanya bisa mengatasi pemisahan di luar konsep yang mudah dipetakan ke dalam objek nyata. Tidak bisa memetakan ke dalam konsep yang abstrak.

Contohnya, sulit untuk memodelkan sistem keamanan pada desain berorientasi objek. Fungsi check() mrupakan fungsi umum yang penting pada sistem keamanan dan tersebar pada seluruh desain, karena fungsi ini mempengaruhi jalannya sistem secara keseluruhan. Apabila ada perubahan terhadap fungsi check() ini, maka seluruh desain harus diperiksa satu per satu untuk memperbaikinya. Hal ini tentu saja membutuhkan waktu yang lama serta biaya yang mahal. Dan apabila pada salah satu kelas saja fungsi ini lupa diperiksa, maka sistem tidak akan berjalan dengan baik.

Karena fungsi check() merupakan fungsi yang universal, fungsi ini perlu dikeluarkan dari masing-masing kelas yang memilikinya untuk kemudian ditempatkan dalam sebuah aspek. Aspek inilah nantinya yang akan direlasikan

dengan kelas-kelas yang sebelumnya mempunyai fungsi check(). Sehingga apabila diadakan perubahan terhadap fungsi ini, hanya tinggal memeriksa aspeknya. Hal ini akan mempercepat waktu pengerjaannya serta biaya yang dikeluarkan sehubungan dengan adanya perubahan itu akan menjadi lebih sedikit.

Resume mengenai pemodelan berorientasi objek dan pemodelan berorientasi aspek pada sistem keamanan login.

Tabel 1.3. Resume OOD dan AOD

Pemodelan berorientasi objek			**Pemodelan berorientasi aspek**		
Kelas	**Operasi**	**Atribut**	**Kelas**	**Operasi**	**Atribut**
LoginForm	submit()	-	LoginForm	submit()	-
	check()			error_message()	
	error_message()				
Authen	AuthenticateUser()	-	Authen	AuthenticateUser()	-
DBManajer	selectUser()	User	DBManajer	selectUser()	user
	selectRoles()	Roles		selectRoles()	roles
	selectRolePriv()	Privileges		selectRolePriv()	privileges
	selectPrivileges()			selectPrivileges()	
UserList	is_member()	-	UserList	is_member()	-
UAList	getAssignedRoles()	-	UAList	getAssignedRoles()	-
	assignedUser()			AssignedUser()	
UA	assignUser()	id_user	UA	assignUser()	id_user
	deassignUser()	role		deassignUser()	role
	assignedRoles			assignedRoles()	
	check()				
Credential	set_roles	mechanism	Credential	set_roles	mechanism
	get_roles	credential_type		get_roles	credential_type
	refresh()	roles		Refresh	roles
MainForm	load()	-	MainForm	load()	-
			RoleHierarchy	UAList::AuthorizedUser() <<introduction>>	-
			<<aspek>>	UAList::getAuthorizedRoles() <<introduction>>	
				Handle_rh() <<pointcut>>	
			Check <<aspek>>	check() <<pointcut>>	-

BAB IV

IMPLEMENTASI DAN UJI COBA MODEL SISTEM KEAMANAN

Grace Copy Center

BAB IV

IMPLEMENTASI DAN UJI COBA MODEL SISTEM KEAMANAN

Bab ini berisi implementasi model sistem keamanan sesuai dengan perancangannya yang ada di Bab III. Pembahasan meliputi lingkungan pembangunan perangkat lunak, implementasi kelas perangkat lunak, implementasi proses, dan implementasi antar muka.

4.1. Lingkungan Pembangunan Perangkat Lunak

Lingkungan pembangunan perangkat lunak ini meliputi perangkat keras dan perangkat lunak yang digunakan. Spesifikasi perangkat keras dan perangkat lunak yang digunakan dalam pembangunan perangkat lunak sistem keamanan login ini dapat dilihat pada tabel dibawah ini.

Tabel 4.1 Lingkungan pembangunan perangkat lunak

Perangkat Keras	Prosesor : Intel Pentium IV 2400 MHz Memory : 512 MB
Perangkat Lunak	Sistem Operasi : Microsoft Windows XP Pro. Bahasa Pemrograman : Java. Compiler & Tools : Rational Rose Enterprise Edition 2001, Eclipse 3.0 , Java J2SE 1.4. Component : Aspect J.1.1 .

4.2. Implementasi Kelas Perangkat Lunak

Pada sub bab ini akan dijelaskan mengenai implementasi dari kelas-kelas sesuai dengan perancangannya yang sudah disampaikan pada bab III.

4.2.1. Kelas DBManager

Kelas DBManager ini menyediakan operasi-operasi (*methods*) yang akan digunakan untuk melakukan *query* terhadap data tertentu yang selanjutnya akan ditampung pada kelas-kelas yang dituju. *Method* untuk koneksi ke basis data juga dituliskan di kelas ini. Hal ini dimaksudkan jika ada kelas lain yang memerlukan koneksi basis data atau ingin menampilkan data, tinggal memanggil kelas DBManager dan *method* yang dibutuhkan.

```
public class DBManager {
  private Connection conn;
  private Statement stmt;
  public ResultSet select(String namakol, String namatabel, String
  where) throws SQLException {
  stmt=conn.createStatement();
  return (stmt.executeQuery("select "+namakol+" from "+namatabel+"
  where "+where));
  }
  public ResultSet selectTabel(String namatabel) throws Exception {
  stmt=conn.createStatement();
  return(stmt.executeQuery("select * from "+namatabel));
  }
  public DBManager(String driver, String url, String usr, String psd)
  throws Exception {
     try{
     Class.forName(driver);
     conn=DriverManager.getConnection(url, usr, psd);
      }
     catch(SQLException ex){
     System.out.println(ex);
      }
     }
  public int getNumberData(String namaTabel) {
  int i = 0;
  try{
  PreparedStatement pStatement1 = conn.prepareStatement("Select
  Count(*) From "+namaTabel);
      try {
      ResultSet rset1 = pStatement1.executeQuery();
      if(rset1.next()) {
            i = Integer.parseInt(rset1.getString(1));
            }
```

```
            else i = 0;
            }
            catch (Exception ex) { // Trap general Errors
                  ex.printStackTrace();
                  }
                  finally {
                        pStatement1.close();
                  }
            }
            catch (Exception ex) { // Trap general Errors
                  ex.printStackTrace();
            }
            finally {
                  // Close opened statement
                  return i;
            }
      }
      public String selectUser(){
      return null;
      }
      public String selectRoles() {
       return null;
      }
      public String selectRolePriv() {
       return null;
      }
      public String selectPrivileges() {
       return null;
      }
}
```

Gambar 4.1. Implementasi kelas DBManager

4.2.2. Kelas Authen

Kelas Authen memeriksa input yang dimasukkan user dengan menggunakan menggunakan parameter id user dan password untuk dicocokkan dengan data yang ada pada kelas UserList.

```
public class Authen {
    private DBManager LoginDBman;
    public DBManager theDBManager;
    public Credentials theCredentials;
    public UserList theUserList;
    public Authen() {
    ..............................
    }
    public String AuthenticateUser(String id_user, String pass)
    {
    ..............................
    return null;
    }
}
```

Gambar 4.2. Implementasi kelas Authen

4.2.3. Kelas UserList

Kelas UserList digunakan untuk menampung data yang berupa nama seluruh user dan semua role yang ada pada saat user membuka aplikasi. Dengan demikian, pada waktu user melakukan aktivitas login, sistem akan memeriksa kelas UserList, bukan basis data.

```
public class UserList {
    private ArrayList user2, pass2;
    public UserList(ArrayList user2, ArrayList pass2) {
    this.user2=user2;
    this.pass2=pass2;
    }
    public ArrayList getUser(){
    return user2;
    }
    public ArrayList getPass(){
    return pass2;
    }
    public Boolean is_member() {
    return null;
    }
}
```

Gambar 4.3. Implementasi kelas UserList

4.2.4. Kelas UA

Kelas UA merupakan kelas yang berhubungan dengan hak yang dimiliki user. Kelas ini mempunyai method antara lain: assignUser() untuk pemberian hak terhadap user, deassignUser() untuk pencabutan/penghapusan satu atau beberapa hak user, assignedRoles() untuk mengetahui role apa saja yang telah diberikan terhadap user tertentu.

```
public class UA {
    private String id_user;
    private String role;
    public UA() {
    ..............
    }
    public void assignUser() {
    ................
    }
    public void deassignUser() {
    ................
    }
    public String assignedRoles() {
```

MILIK PERPUSTAKAAN INSTITUT TEKNOLOGI SEPULUH - NOPEMBER

```
    return null;
    }
    public void check() {
    ..................
    }
}
```

Gambar 4.4. Implementasi kelas UA

4.2.5. Kelas UAList

Kelas ini tergantung (*dependent*) pada kelas UA. Pada kelas ini terdapat *method* getAssignedRoles() untuk mendapatkan role yang diberikan kepada user tertentu, *method* AssignedUser() untuk mengetahui siapa saja user yang telah di-*assign*.

```
public class UAList {
    private ArrayList UA;
    private ArrayList role;
    public UAList(ArrayList role) {
    this.role=role;
    }
    public ArrayList getAssignedRoles() {
    return role;
    }
    public void AssignedUser() {
    ......................
    }
}
```

Gambar 4.5. Implementasi kelas UAList

4.2.6. Kelas Credentials

```
public class Credentials {
    private String mechanism;
    private String credential_type;
    private ArrayList roles;
    public Credentials() {
    }
    public String set_roles() {
    return set_roles();
    }
    public String get_roles() {
    return get_roles();
    }
    public Boolean refresh() {
    return null;
    }
}
```

Gambar 4.6. Implementasi kelas Credentials

4.2.7. Kelas LoginForm

Kelas LoginForm menyediakan halaman untuk user melakukan aktivitas login. Pada saat halaman dibuka oleh user, kelas ini akan mengambil data user serta role yang ada untuk kemudian ditampung pada kelas yang sudah ditentukan. Pada kelas ini terdapat *method* antara lain check() dan error_message() untuk memeriksa input yang dimasukkan user dan pesan error apabila input tidak sesuai dengan data.

```
public class LoginForm extends JFrame {
    private JPasswordField pass;
    private JTextField user;
    private JLabel judul, user1, pass1, kosong, label0;
    private JPanel p1, p2, p3, p4, p5;
    private JButton but, cancel;
    private DBManager d;
    private UAList ua;
    private MainForm m;
    private ResultSet rs;
    private UserList ul;

    public LoginForm() throws Exception {
    this.setTitle("Login Area");
    this.setLocation(450,300);
    this.m=m;

    DBManager x = new DBManager("sun.jdbc.odbc.JdbcOdbcDriver",
    "jdbc:odbc:sop", "usr", "psd");
    ResultSet rs = x.selectTabel("UserList");
    ResultSet rs1 = x.selectTabel("Role");
    ArrayList user2 = new ArrayList();
    ArrayList pass2 = new ArrayList();
    ArrayList role = new ArrayList();
    if (rs!=null){
    while(rs.next()){
    pass2.add(rs.getString(4));
    user2.add(rs.getString(3));
    }
    }
    if (rs1!=null){
    while(rs1.next()){
    role.add(rs1.getString(2));
    }
    }
    ul= new UserList(user2, pass2);
    ua = new UAList(role);
    try {
    UIManager.setLookAndFeel (
    UIManager.getCrossPlatformLookAndFeelClassName());
    }
    catch (Exception e) {
    // Can't change look and feel
```

```
    }
    getContentPane().setLayout(new BorderLayout());
    p4=new JPanel();
    p4.setLayout(new BorderLayout());
    getContentPane().add(p4, BorderLayout.NORTH);
    p2=new JPanel();
    p2.setLayout(new FlowLayout(FlowLayout.CENTER));
    p4.add(p2, BorderLayout.NORTH);
    judul=new JLabel("LOGIN");
    judul.setFont(new Font("Verdana", Font.BOLD, 16));
    p2.add(judul);
    p1=new JPanel();
    p1.setLayout(new GridLayout(2,2,3,3));
    getContentPane().add(p1, BorderLayout.CENTER);
    user1=new JLabel("Username");
    user1.setFont(new Font("Verdana", Font.PLAIN, 12));
    p1.add(user1);
    user=new JTextField(10);
    p1.add(user);
    pass1=new JLabel("Password");
    pass1.setFont(new Font("Verdana", Font.PLAIN, 12));
    p1.add(pass1);
    pass=new JPasswordField(10);
    p1.add(pass);
    p3=new JPanel();
    p3.setLayout(new FlowLayout(FlowLayout.CENTER));
    getContentPane().add(p3, BorderLayout.SOUTH);
    but=new JButton("    OK    ");
    but.setBorder(new BevelBorder(BevelBorder.RAISED));
    but.setFont(new Font("Verdana", Font.BOLD, 12));
    but.addActionListener(
        new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
            if(user.getText().length()<1||pass.getText().length()<1){
                error_message();}
            else if(ul.getUser().contains(user.getText()))
                if(String.valueOf(ul.getPass().get(ul.getUser()
                .indexOf(user.getText()))).equals(String.valueOf
                (pass.getPassword())))
                try{ MainForm mf=new MainForm(ul,ua,user.getText());
                }
                catch(Exception ex){}}
                }
    }
    );
    p3.add(but);
    cancel=new JButton(" Cancel ");
    cancel.setBorder(new BevelBorder(BevelBorder.RAISED));
    cancel.setFont(new Font("Verdana", Font.BOLD, 12));
    cancel.addActionListener(
        new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
            System.exit(0);
            }
        }
    );
    p3.add(cancel);
    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
        dispose();
        System.exit(0);
        }
```

```
    });
    setVisible(true);
    pack();
    this.setResizable(false);
    }
    public void submit() {

    }
    public void error_message() {
        javax.swing.JOptionPane.showMessageDialog(null,
        "Username / password tidak boleh kosong","Error",
    JOptionPane.ERROR_MESSAGE);
    }
    public void check() {
    }
    public static void main(String args[]) throws Exception{
    LoginForm lf = new LoginForm();
    lf.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
}
```

Gambar 4.7. Implementasi kelas LoginForm

4.2.8. Kelas MainForm

Kelas MainForm menyediakan halaman utama yang berisi daftar privilege user yang login pada saat itu. Halaman ini akan terbuka apabila input yang dimasukkan user pada halaman login sesuai dengan data.

```
public class MainForm extends JFrame {
    public JPasswordField pass;
    public JTextField user;
    private DisplayMode original;
    public GraphicsDevice device;
    private boolean fs = false;
    private boolean logon = false;
    private LoginForm lf;
    private JLabel ltoolbar;
    private JLabel lbl;
    private JPanel p;
    private UserList ul;
    private DBManager d;
    private JLabel label0;
    private JPanel panel2;
    Vector columnNames = new Vector();
    Vector data = new Vector();
    private JMenuBar menubar;
    private JTable table = null;
    GridBagConstraints gbc;
    public MainForm(UserList list1, UAList ua, String user) throws
    Exception {
    super("Privilege User");
    DBManager x = new DBManager("sun.jdbc.odbc.JdbcOdbcDriver",
    "jdbc:odbc:sop", "usr", "psd");
    ResultSet rs=x.select("privilege.privilege","role_priv, privilege,
    login_role", "login_role.id_user='"+user+"' and
    login_role.id_role=role_priv.id_role and
```

```
    privilege.id_privilege=role_priv.id_privilege");


    Vector row = new Vector();
    while (rs.next()) {
       try {
       ResultSetMetaData md = rs.getMetaData();
       int columns = md.getColumnCount();
       //  Get column names

       for (int i = 1; i <= columns; i++)
       {
       columnNames.addElement( md.getColumnName(1) );
       }

       //  Get row data

       for (int i = 1; i <= columns; i++)
       {
       row.addElement( rs.getString(1) );

       }
    }
    catch(SQLException se){
       System.out.println(se);
       }
    }

    data.addElement( row );
    rs.close();

    table = new JTable(data, columnNames);
    table.setPreferredScrollableViewportSize(new Dimension(500, 70));
    table.setForeground(new Color(51, 51, 255));
    table.setSelectionBackground(Color.blue);
    table.setSelectionForeground(Color.yellow);

    JTable table = new JTable(data, columnNames);
    JScrollPane scrollPane = new JScrollPane( table );
    getContentPane().add( scrollPane );

    JPanel buttonPanel = new JPanel();
    getContentPane().add( buttonPanel, BorderLayout.SOUTH );

    JLabel label = new JLabel(data.toString());
    this.device = device;
    this.setSize(400,550);
    this.setLocation(225,150);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.getContentPane().setLayout(new BorderLayout());
    this.getContentPane().add("North",  new  JLabel(user+",  privilege
    anda adalah sebagai berikut: "));
    this.getContentPane().add("Center",new JLabel(data.toString());
    this.setDefaultCloseOperation(EXIT_ON_CLOSE);
    this.pack();
    setVisible(true);
    }
    public Boolean load() {
    return null;
    }
}
```

Gambar 4.8. Implementasi kelas MainForm

4.3. Implementasi Proses

Implementasi proses ini berisi penerapan pemrograman berorientasi aspek yang ditambah dengan listing kode aspek yang merupakan kunci utama sistem berorientasi aspek. Sesuai dengan perancangan pada bab 3, ada dua listing kode aspek yaitu aspek Check() dan aspek RoleHierarchy().

- aspek Check

```
public aspect Check {
    pointcut check():execution(*UA.assignUser())||
    execution (*LoginForm.LoginForm);
    before():check(){
    keyDown();
    }
    public boolean keyDown(Event e, int key){
       if(key<123 || key>125){
              if(key<91 || key > 95){
                     if(key<58 || key >64) {
                     if(key<32 || key >47)
                     System.out.println("no " + key);
                            }
                     }
              }
               return true;
       }
}
```

Gambar 4.9. Implementasi aspek check

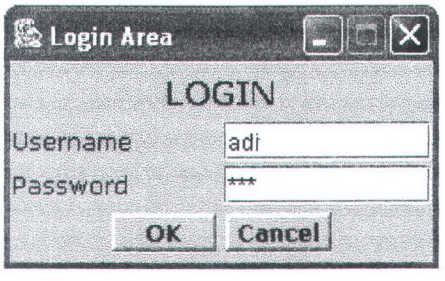
- aspek RoleHierarchy

```
public aspect RoleHierarchy {
    public pointcut handle_rh():call(*UAlist.getAssignedRoles());
    private String UAList.getAuthorizedRoles();
    void around():handle_rh(){
    UAList.getAuthorizedRoles();
    }
}
```

Gambar 4.10. Implementasi aspek RoleHierarchy

4.4. Implementasi dan Uji Coba AntarMuka

Berikut ini adalah implementasi dan uji coba antarmuka yang telah dijelaskan pada bab tiga yang merupakan halaman login dan halaman utama yang berisi nama user, role user, serta privilege user yang login pada saat itu.

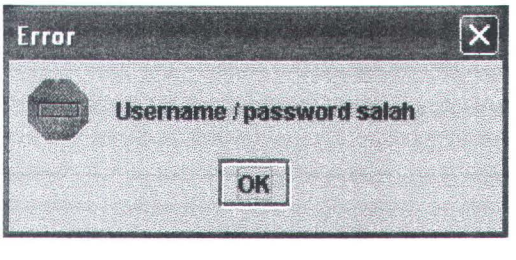


Gambar 4.11. Implementasi antarmuka untuk halaman login

Apabila login sukses, maka akan ditampilkan halaman utama seperti gambar di bawah ini:

Gambar 4.12. Implementasi antarmuka untuk halaman utama

Sedangkan jika loginnya salah, maka akan muncul pesan kesalahan seperti gambar berikut:



Gambar 4.13. Pesan kesalahan pada halaman login

PERPUSTAKAAN INSTITUT TEKNOLOGI SEPULUH NOPEMBER

Aplikasi diatas merupakan implementasi baik aplikasi yang menerapkan pemrograman berorientasi objek maupun yang menerapkan pemrograman berorientasi aspek memiliki desain antarmuka yang sama sehingga keduanya cukup diwakilkan dengan kedua tampilan antarmuka diatas.

BAB V

KESIMPULAN DAN SARAN

Grace Copy Center

BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

- UML merupakan alat yang *extendible* dimana bisa ditambahkan notasi-notasi baru melalui mekanisme *stereotype*.
- Hasil implementasi dari OOD dan AOD menunjukkan bahwa hasil rancangan dengan menggunakan AOD memiliki rancangan yang lebih bersih karena tidak ada *crosscutting concerns*.

5.2. Saran

Perlu dilakukan pemodelan secara menyeluruh pada suatu sistem yang lengkap sehingga didapatkan hasil yang jelas setelah menggunakan teknik pembangunan perangkat lunak berorientasi aspek dan didapat keuntungan menggunakan pembangunan perangkat lunak berorientasi aspek dalam hal modularitas, *reusability*, dan *maintainability* yang tinggi.

DAFTAR PUSTAKA

Grace Copy Center

DAFTAR PUSTAKA

[BOO98] Booch, Grady dkk, *Unified Modeling Language User Guide*,Lakewood, Colorado: Addison-Wesley, September 1998

[CLE04] Clement, Andy. AJDT: *getting started with aspect-oriented programming in Eclipse*. AJDT and AspectJ Committer, IBM UK, 2004.

[CLI03] Clifton, Marc, *Aspect Oriented Programming / Aspect Oriented Software Design*, http://www.codeproject.com/gen/design/aop.asp, 30 April 2003

[GAO04] Gao, Shu, *Applying Aspect-Orientation in Designing Security Systems: A Case Study,* School of Computer Science Florida International University, 2004

[GRO03] Groher, Iris, *Generating Aspect Code from UML Models,* Boston-USA, Maret 2003

[LAD02] Laddad, Ramnivas, *Separate software concerns with aspect-oriented programming*, http://www.javaworld.com/javaworld/jw-01-2002/jw-0118-aspect.html, Januari 2002

[MIC03] Mickelson, Magnus, *Aspect-Oriented Programming Compared to Object-Oriented Programming When Implementing a Distributed, Web Based Application*, Department of Information Technology Uupsala University, 2003

[SAU04] Sauer, Eileen, *Why use Aspect Oriented Technologies?*, http://www.volantec.biz/aop.htm, 30 Oktober 2004

[SHI01] Shiran, Yehuda, *OO(Object Oriented) Design principles*, http://www.webreference.com/js/tips/010211.html, Februari 2001

[STE02] Stein, Dominik. *A UML-based Aspect-Oriented Design Notation For AspectJ*.Institute for Computer Science University of Essen-Germany, 2001

[TEK03] Tekinerdogan, Bedir. *Aspect-Oriented Software Development*. http://trese.cs.utwente.nl/taosad/aosd.htm