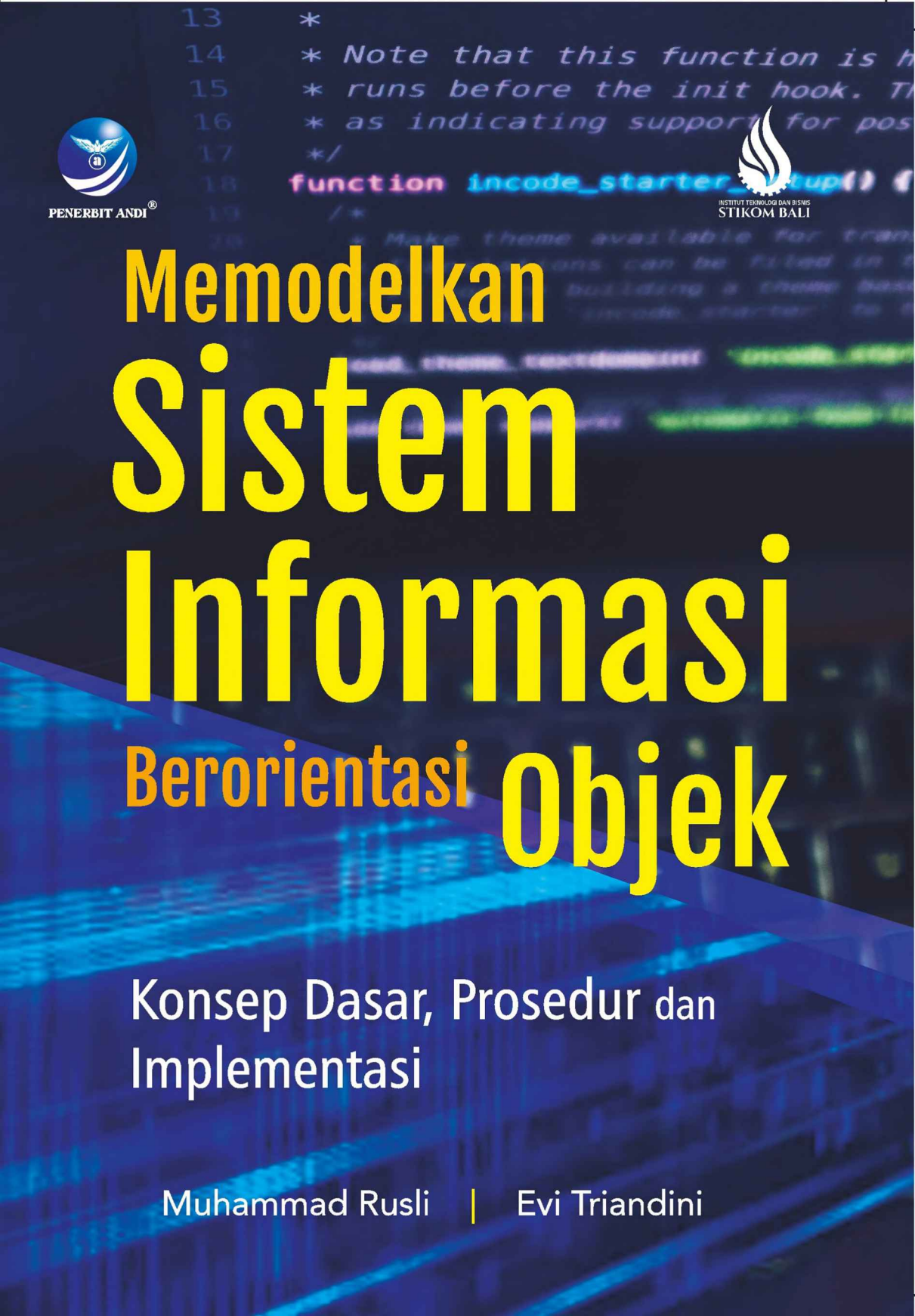

PENERBIT ANDI®
INSTITUT TEKNOLOGI DAN BISNIS
STIKOM BALI
Memodelkan
Sistem
Informasi
Berorientasi Objek
Konsep Dasar, Prosedur dan
Implementasi
Muhammad Rusli | Evi Triandini

MEMODELKAN SISTEM INFORMASI BERORIENTASI OBJEK

Konsep Dasar, Prosedur, dan Implementasi

Muhammad Rusli
Evi Triandini

Diterbitkan atas Kerja Sama:

PENERBIT ANDI®

INSTITUT TEKNOLOGI DAN BISNIS
STIKOM BALI

MEMODELKAN SISTEM INFORMASI BERORIENTASI OBJEK
Konsep Dasar, Prosedur, dan Implementasi

Oleh: Muhammad Rusli
Evi Triandini

Hak Cipta ©2022 pada Penulis.

Editor : Radhitya Indra
Desain Cover : Andang Suhana
Setter : Tabitha Sonia Ayudea
Korektor : Robertus Ari

Hak Cipta dilindungi undang-undang.
Dilarang memperbanyak atau memindahkan sebagian atau seluruh isi buku ini dalam bentuk apa pun, baik secara elektronis maupun mekanis, termasuk memfotokopi, merekam atau dengan sistem penyimpanan lainnya, tanpa izin tertulis dari Penulis.

Diterbitkan oleh Penerbit ANDI (Anggota IKAPI)
Jl. Beo 38-40, Telp. (0274) 561881 (Hunting), Fax. (0274) 588282 Yogyakarta 55281

Percetakan: CV ANDI OFFSET
Jl. Beo 38-40, Telp. (0274) 561881 (Hunting), Fax. (0274) 588282 Yogyakarta 55281

Rusli, Muhammad
MEMODELKAN SISTEM INFORMASI BERORIENTASI OBJEK - Konsep Dasar, Prosedur, dan Implementasi / Muhammad Rusli, Evi Triandini
– Ed. I. – Cetakan 1;
31 - 30 - 29 - 28 - 27 - 26 - 25 - 24 - 23 - 22
hlm viii + 184; 16 x 23 Cm.
10 9 8 7 6 5 4 3 2 1
ISBN : 978-623-01-2596-6
978-623-01-2597-3 (PDF)
I. Judul
1. Information System
2. Triandini, Evi

DDC'23: 658.403.801.1

PRAKATA

Dengan memanjatkan puji syukur ke hadirat Tuhan Yang Maha Esa, materi buku *Memodelkan Sistem Informasi Berorientasi Objek: Konsep Dasar, Prosedur, dan Implementasi* sebagai buku ajar guna mendukung/melengkapi referensi, pengetahuan, serta pembelajaran tentang bagaimana memodelkan atau merancang sistem informasi berorientasi objek dengan UML (*Unified Modelling Language*) dapat kami selesaikan.

Dalam membangun/mengembangkan suatu sistem perangkat lunak yang dipesan pelanggan (klien), salah satu tantangan utama yang akan dihadapi adalah mengklarifikasi apa yang sebenarnya diinginkan pelanggan, dan apakah kita telah memahami dengan tepat kebutuhan sistem yang dikehendaki pelanggan sesuai dengan prospektif sebuah sistem. Langkah tersebut sangat penting karena dapat menentukan keberhasilan atau kegagalan proyek yang akan kita kerjakan. Pertanyaannya adalah bagaimana kita sebaiknya berkomunikasi dengan pelanggan/manajer bisnis? Bahasa alami belum tentu merupakan pilihan yang baik karena tidak tepat dan ambigu. Kesalahpahaman dapat dengan mudah terjadi dan memiliki

risiko yang cukup serius ketika orang dengan latar belakang yang berbeda (misalnya, seorang ilmuwan/praktisi komputer dan pelanggan/manajer bisnis) berkomunikasi di lintas tujuan.

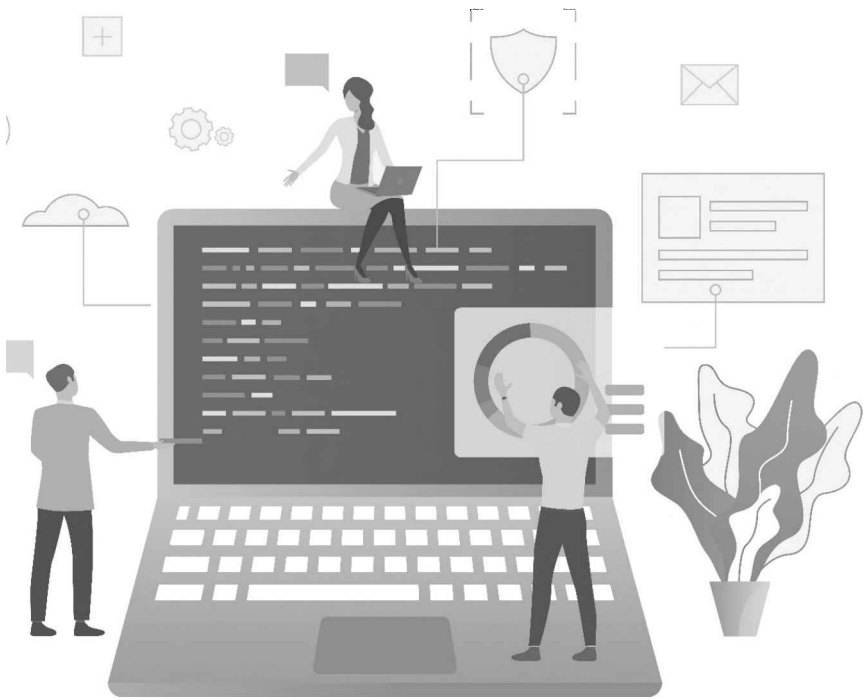
Untuk itu, diperlukan sebuah model dari perangkat lunak yang akan dibangun. Model ini fokus pada aspek-aspek penting dari perangkat lunak dalam bentuk notasi/simbol yang jelas dan sesederhana mungkin, dengan mengabaikan abstraksi detail yang tidak relevan. Model ini, sebagaimana dalam arsitektur, disebut rencana konstruksi. Rencana konstruksi untuk sebuah bangunan berisi informasi/gambaran, antara lain tentang denah ruangan dan lantai. Bahan konstruksi yang akan digunakan tidak ditentukan pada saat ini, mereka tidak relevan dan akan membuat rencana lebih rumit daripada yang diperlukan. Rencana konstruksi juga tidak berisi informasi tentang bagaimana kabel listrik akan diletakkan. Rencana terpisah dibuat untuk aspek ini guna menghindari terlalu banyak informasi tersaji sekaligus. Untuk itu, penting dalam bidang teknologi informasi bahwa orang-orang dengan latar belakang yang berbeda (pemrogram, analis dan perancang sistem informasi, pengembang, pelanggan/manajer bisnis) dapat membaca, memahami, menafsirkan, dan mengimplementasikan model.

Semoga buku ini bermanfaat bagi pembaca dan pemerhati/peminat tentang pemodelan sistem informasi berorientasi objek dengan UML. Kritik dan saran yang membangun sangat kami harapkan. Terima kasih atas segala perhatiannya.

Surabaya, Januari 2022

Penulis

DAFTAR ISI



BAB 1
PENDAHULUAN

1.1 Tentang UML

Unified Modeling Language (UML) adalah bahasa yang digunakan untuk menspesifikasikan, memvisualisasi, membangun, dan mendokumentasikan artefak dari sistem perangkat lunak, serta untuk memodelkan sistem bisnis dan non-perangkat lunak. UML telah muncul dalam bentuk notasi diagram standar "de facto dan de jure" untuk Pemodelan Berorientasi Objek (PBO).

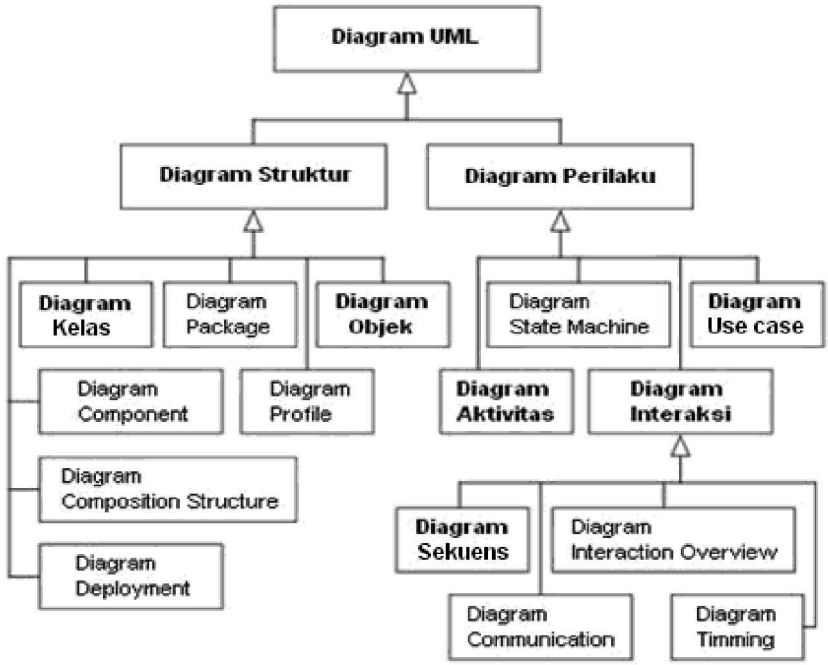
Diagram PBO terbagi menjadi dua kategori utama, yaitu untuk memodelkan (1) struktur sistem dan (2) perilaku sistem. Diagram struktur menyediakan cara untuk merepresentasikan data dan relasi/hubungan statis dalam sistem informasi. Sementara, diagram perilaku memberikan analis cara untuk menggambarkan hubungan dinamis di antara objek-objek yang mewakili sistem informasi. Di samping itu, dalam diagram perilaku memungkinkan juga memodelkan perilaku dinamis objek-objek secara individu sepanjang daur hidup mereka. Gambar 1.1 merepresentasikan diagram UML beserta kategori dan komponen-komponennya, sementara Gambar 1.2 menyajikan deskripsi dari komponen-komponen UML yang digunakan berikut fase/tahap dalam implementasinya.

Tidak semua komponen UML akan dibahas dalam buku ini, beberapa komponen yang dibahas utamanya yang terkait dengan fase analisis dan desain sistem (tercetak dengan huruf tebal/*bold*).

1.2 Motivasi

Andaikan kita ingin membangun/mengembangkan suatu sistem perangkat lunak yang dipesan pelanggan, salah satu tantangan pertama yang akan kita hadapi adalah mengklarifikasi apa yang sebenarnya diinginkan pelanggan (klien) dan apakah kita telah memahami dengan tepat kebutuhan sistem yang dikehendaki

pelanggan sesuai dengan prospektif sebuah sistem. Langkah tersebut sangat penting karena dapat menentukan keberhasilan atau kegagalan proyek yang akan kita kerjakan. Pertanyaannya adalah bagaimana kita sebaiknya berkomunikasi dengan pelanggan/manajer bisnis? Bahasa alami belum tentu merupakan pilihan yang baik karena tidak tepat dan ambigu. Kesalahpahaman dapat dengan mudah terjadi dan memiliki risiko yang cukup serius ketika orang dengan latar belakang yang berbeda (misalnya, seorang ilmuwan/praktisi komputer dan pelanggan/manajer bisnis) berkomunikasi di lintas tujuan.



Gambar 1.1 Diagram UML

Nama Diagram	Digunakan untuk...	Pada Fase/Tahap
Diagram Struktur		
Kelas	Mengilustrasikan relasi antar kelas dalam sistem	Analisis, Desain
Objek	Mengilustrasikan relasi antar objek dalam sistem	Analisis, Desain
Diagram Perilaku		
Use case	Menangkap kebutuhan bisnis dari sistem, dan mengilustrasikan interaksi antar sistem dengan lingkungannya	Analisis
Sekuens	Memodelkan perilaku objek-objek dalam sebuah use case, fokus pada urutan aktivitas sesuai sesuai waktu	Analisis, Desain
Aktivitas	Mengilustrasikan alur ketja bisnis dari kelas, alur aktivitas sebuah use case, atau desain detil dari metode	Analisis, Desain

Gambar 1.2 Deskripsi komponen-komponen UML beserta tahap/fase implementasinya

Untuk itu, yang kita butuhkan adalah sebuah model dari perangkat lunak yang akan kita bangun. Model ini fokus pada aspek-aspek penting dari perangkat lunak dalam bentuk notasi/ simbol yang jelas dan sesederhana mungkin, dengan mengabaikan abstraksi detail yang tidak relevan. Model ini, sebagaimana dalam arsitektur disebut rencana konstruksi. Rencana konstruksi untuk sebuah bangunan berisi informasi/gambaran, antara lain tentang denah ruangan dan lantai. Bahan konstruksi yang akan digunakan tidak ditentukan pada saat ini; mereka tidak relevan dan akan membuat rencana lebih rumit daripada yang diperlukan. Rencana konstruksi juga tidak berisi informasi tentang bagaimana kabel listrik akan diletakkan. Rencana terpisah dibuat untuk aspek ini guna menghindari terlalu banyak informasi tersaji sekaligus. Untuk itu, penting dalam bidang teknologi informasi bahwa orang-orang dengan latar belakang yang berbeda (pemrogram, analis dan perancang sistem informasi, pengembang, pelanggan/ manajer bisnis) dapat membaca, memahami, menafsirkan, dan mengimplementasikan model.

Bahasa pemodelan dikembangkan dengan tepat untuk skenario tersebut dan menunjukkan aturan yang jelas untuk deskripsi sistem terstruktur. Bahasa pemodelan dapat dirancang untuk domain tertentu, misalnya, untuk menghapus coretan aplikasi web. Di satu sisi, bahasa pemodelan khusus domain ini menyediakan alat dan pedoman untuk memecahkan masalah di bidang tertentu secara efisien. Di samping itu, bahasa pemodelan juga dapat dirancang untuk penggunaan tujuan umum. Bahasa UML, yang merupakan subjek buku ini, adalah bahasa pemodelan untuk tujuan umum dan digunakan untuk mengenalkan konsep pemodelan berorientasi objek.

1.3 Model

Model adalah gambaran atas sesuatu atau sebuah sistem yang nyata (atau abstrak) ke dalam bentuk yang lebih sederhana, dengan tujuan agar mudah dipelajari dan dimengerti. Sementara sistem terdiri atas komponen-komponen yang saling terintegrasi dan terkait satu sama lain serta saling memengaruhi sedemikian hingga dapat dianggap sebagai unit tunggal, berbasis tugas atau berbasis tujuan. Melalui model, sistem dimungkinkan tergambar secara efisien dan elegan. Contoh sistem adalah hal-hal yang bersifat material, seperti mobil atau pesawat terbang; lingkungan ekologi, seperti danau dan hutan; unit organisasi, seperti universitas atau perusahaan; dan juga teknologi informasi, seperti sistem perangkat lunak.

Sistem perangkat lunak bersifat abstrak (kebalikan dari nyata/konkret). Sifat abstraksinya terkait dengan pengiriman ulang fakta/realitas yang dapat diproses mesin. Dalam konteks ini, abstraksi berarti generalisasi—menyisihkan fitur yang bersifat spesifik dan individual. Dengan demikian, abstraksi sistem berarti mengenali karakteristik sistem secara umum.

Dalam membangun sistem perangkat lunak, diperlukan sarana abstraksi yang tepat/sesuai, baik untuk implementasi, maupun juga untuk penggunaan berikutnya. Memilih cara abstraksi yang tepat dapat membuat pemrograman lebih mudah. Sementara itu, jika memilih cara abstraksi yang salah dapat mengakibatkan sejumlah kejutan tak terduga selama implementasi, seperti antarmuka yang rumit dan sulit untuk menerapkan perubahan. Dengan menetapkan abstraksi yang tepat/sesuai, kebutuhan akan pengelolaan kompleksitas sistem perangkat lunak modern yang terus meningkat akan dapat terlayani/teratasi dengan baik. Di sinilah pemodelan dapat memberikan manfaat/layanan yang berharga.

Terdapat tiga karakteristik model, yaitu pemetaan, pengurangan, dan pragmatisme yang dideskripsikan sebagai berikut.

1. **Pemetaan**: model selalu merupakan gambar (pemetaan) dari sesuatu, representasi orisinal alami atau buatan yang dapat menjadi model sendiri.
2. **Reduksi**: model tidak menangkap semua atribut asli, melainkan hanya yang tampaknya relevan dengan pemodel atau pengguna model.
3. **Pragmatisme**: pragmatisme berarti berorientasi terhadap kegunaan. Model ditetapkan dari aslinya berdasarkan pertanyaan berikut: Untuk siapa? Mengapa? Untuk apa? Model digunakan oleh pemodel atau pengguna menyerupai asli dalam kerangka waktu tertentu dan untuk tujuan tertentu.

Model mendukung representasi sistem yang dikurangi menjadi hal-hal penting untuk meminimalkan kompleksitas sistem ke aspek yang dapat dipahami manusia. Sistem biasanya dijelaskan

bukan oleh satu tampilan, tetapi oleh sejumlah tampilan yang bersama-sama menghasilkan gambar keseluruhan secara terpadu.

Selanjutnya, terdapat lima karakteristik yang dapat digunakan untuk menentukan kualitas model, yakni sebagai berikut.

1. **Abstraksi**: model selalu merupakan representasi dari sistem yang diwakilinya. Namun, beberapa detail yang tidak relevan dalam konteks tertentu disembunyikan atau dihapus agar lebih mudah bagi pengguna dalam memahami esensi keseluruhan sistem.
2. **Kemampuan dimengerti**: model yang baik mengurangi upaya intelektual yang diperlukan untuk memahami konten yang digambarkan. Misalnya, bahasa pemrograman tidak terlalu ekspresif bagi pembaca manusia karena banyak upaya yang diperlukan untuk memahami isi program.
3. **Akurasi**: model harus menyoroti sifat yang relevan dari sistem nyata, mencerminkan realitas sedekat mungkin.
4. **Prediktif**: model mampu mengaktifkan prediksi sifat sistem yang menarik, tetapi kurang jelas sebagaimana yang dimodelkan. Ini dapat dilakukan melalui simulasi atau analisis properti secara formal.
5. **Efektivitas biaya**: dalam jangka panjang, harus lebih murah dalam membuat model daripada membuat sistem yang dimodelkan.

Model dapat digunakan untuk berbagai keperluan. Terdapat dua macam model, yaitu model deskriptif dan preskriptif. Model deskriptif menunjukkan bagian dari realitas untuk membuat aspek tertentu lebih mudah dipahami. Misalnya, peta kota menggambarkan kota sedemikian rupa untuk membantu orang atau penduduk

nonlokal dalam menemukan rute di dalam kota. Sebaliknya, model preskriptif digunakan untuk menawarkan manual konstruksi agar sistem dapat dikembangkan.

Berdasar uraian tersebut, terdapat tiga representasi model, yaitu (1) model sebagai sketsa, (2) model sebagai cetak biru, dan (3) model sebagai program yang dapat dieksekusi. Model digunakan sebagai sketsa untuk mengomunikasikan aspek-aspek tertentu dari sebuah sistem dengan cara yang sederhana. Di sini, model bukan pemetaan lengkap dari sistem. Sketsa sebenarnya dibedakan oleh selektivitas mereka karena mereka dikurangi menjadi aspek penting untuk memecahkan masalah. Sketsa sering membuat solusi alternatif terlihat. Ini kemudian dibahas dalam tim pengembangan. Dengan demikian, model juga digunakan sebagai dasar untuk diskusi.

Berbeda dengan penggunaan model sebagai sketsa, kelengkapan sangat penting ketika model digunakan sebagai cetak biru. Model-model ini harus berisi detail yang cukup untuk memungkinkan pengembang membuat sistem siap dijalankan tanpa harus membuat keputusan desain. Model yang digunakan sebagai cetak biru sering tidak menentukan seluruh sistem, hanya bagian tertentu. Misalnya, definisi antarmuka antara subsistem didefinisikan dalam model, di mana pengembang bebas untuk memutuskan detail implementasi internal.

Model sebagai sketsa dan cetak biru dapat digunakan untuk rekayasa maju dan rekayasa mundur. Dalam rekayasa maju, model adalah dasar untuk membuat kode, sedangkan dalam rekayasa mundur, model dihasilkan dari kode untuk mendokumentasikan kode dengan cara yang jelas dan mudah dimengerti.

Akhirnya, model dapat digunakan sebagai program yang dapat dieksekusi. Ini berarti bahwa model dapat ditentukan dengan tepat

sehingga kode dapat dihasilkan darinya secara otomatis. Dalam konteks UML, pengembangan perangkat lunak berbasis model telah menjadi sangat populer dalam beberapa tahun terakhir; menawarkan proses untuk menggunakan UML sebagai bahasa pemrograman.

1.4 Karakteristik Sistem Berorientasi Objek

Orientasi objek adalah suatu cara menggambarkan atau menyajikan sistem yang berhubungan dengan konsep-konsep tertentu, utamanya objek-pesan-dan-kelas. Sementara itu, sistem berorientasi objek merupakan himpunan objek-objek yang bekerja sama atau berinteraksi guna mencapai/merealisasi tujuan sistem.

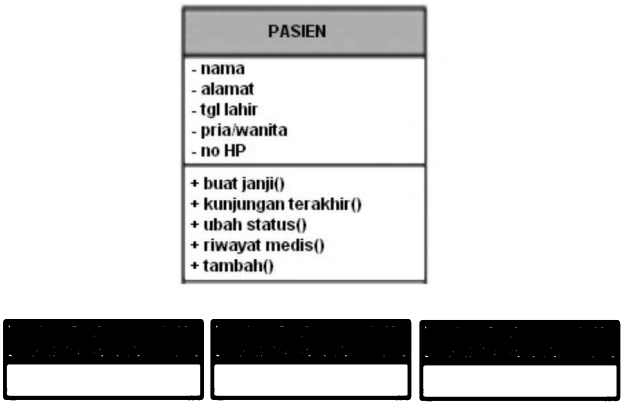
Sistem berorientasi objek fokus pada struktur dan perilaku sistem informasi dalam modul kecil yang mencakup data dan proses. Modul kecil ini dikenal sebagai objek. Pada bagian ini, akan dijelaskan karakteristik dasar dari sistem berorientasi objek, yang meliputi kelas, objek, metode, pesan, enkapsulasi, penyembunyian informasi, warisan, polimorfisme, dan pengikatan dinamis.

1.4.1 Kelas dan Objek

Kelas adalah templat umum yang digunakan untuk menentukan dan membuat instans atau objek tertentu. Setiap objek dikaitkan dengan kelas. Misalnya, semua objek yang menangkap informasi tentang pasien bisa tergolong ke dalam kelas yang disebut Pasien karena ada atribut (misalnya, nama, alamat, tanggal lahir, jenis kelamin, dan telepon) dan metode (misalnya, membuat janji, status kunjungan terakhir, mengubah status, dan memberikan riwayat medis) yang dibagikan kepada semua pasien (Gambar 1.3).

Objek adalah instansiasi dari kelas. Dengan kata lain, objek adalah orang, tempat, atau hal yang ingin kita tangkap informasinya. Andaikan kita sedang membangun sistem layanan konsultasi

dengan kantor dokter, kelas mungkin termasuk Dokter, Pasien, dan Konsultasi. Pasien tertentu, seperti Jimi Akabar, Eli Manise, dan Dedi Marhami, dianggap sebagai contoh atau objek, dari kelas pasien (Gambar 1.3).



Gambar 1.3 kelas dan objek

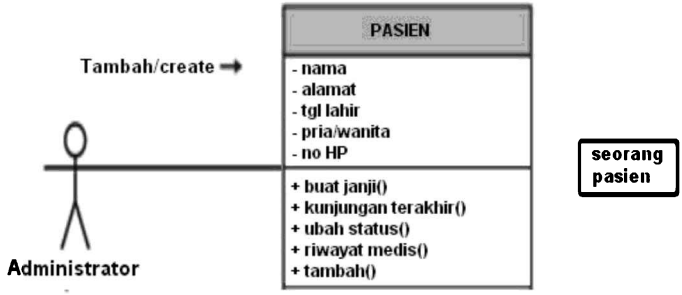
Setiap objek memiliki atribut yang menggambarkan informasi tentang objek, seperti milik (identitas) pasien, yaitu nama, tanggal lahir, alamat, dan nomor telepon. Atribut juga digunakan untuk mewakili hubungan antara objek, misalnya, mungkin ada atribut departemen dalam objek karyawan dengan nilai objek departemen yang menangkap di departemen mana karyawan tersebut objek bekerja. Kondisi atau status suatu objek didefinisikan melalui nilai atribut dan hubungannya dengan objek lain pada waktu tertentu. Misalnya, pasien mungkin memiliki kondisi baru atau saat ini atau sebelumnya.

Setiap objek juga memiliki perilaku. Perilaku menentukan apa yang dapat dilakukan objek. Misalnya, objek konsultasi mungkin dapat menjadwalkan konsultasi baru, menghapus konsultasi, dan menemukan konsultasi berikutnya yang tersedia. Dalam pemrograman berorientasi objek, perilaku diimplementasikan sebagai metode.

Salah satu aspek yang lebih membingungkan dari pengembangan sistem berorientasi objek adalah pada kenyataan bahwa dalam sebagian besar bahasa pemrograman berorientasi objek, baik kelas maupun instans/objek kelas dapat memiliki atribut dan metode. Atribut dan metode kelas cenderung digunakan untuk memodelkan atribut (atau metode) yang menangani masalah yang terkait dengan semua instans kelas/objek. Misalnya, untuk membuat objek pasien baru, pesan dikirim ke kelas Pasien untuk membuat instans baru dari dirinya sendiri.

1.4.2 Metode dan Pesan

Metode menerapkan perilaku objek. Metode tidak lebih dari tindakan yang dapat dilakukan suatu objek. Pesan adalah informasi yang dikirim ke objek untuk memicu metode. Pesan pada dasarnya adalah panggilan fungsi atau prosedur dari satu objek ke objek lain. Misalnya, kalau ada pasien baru di kantor dokter, administrator mengirim pesan *create* atau "tambah" ke aplikasi. Kelas pasien menerima pesan "tambah" dan menjalankan metode tambah(), yang kemudian membuat objek baru: seorang pasien (Gambar 1.4).



Gambar 1.4 Pesan dan metode

1.4.3 Enkapsulasi dan Penyembunyian Informasi

Konsep enkapsulasi dan penyembunyian informasi saling terkait dalam sistem berorientasi objek. Enkapsulasi merupakan gabungan dari proses dan data ke dalam satu entitas. Sementara,

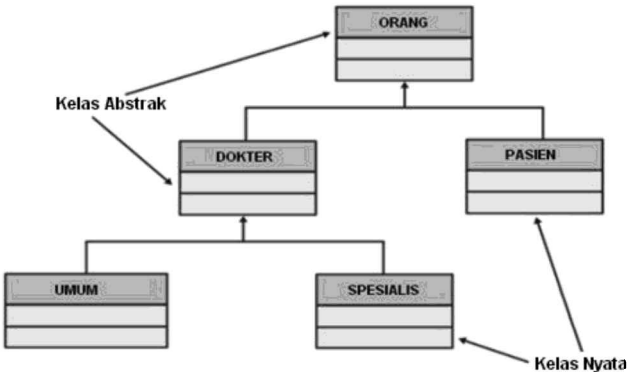
konsep penyembunyian informasi menunjukkan bahwa hanya informasi yang diperlukan dapat dipublikasikan kepada pengguna modul. Biasanya, ini menyiratkan bahwa informasi yang diperlukan untuk diteruskan ke modul dan informasi yang dikembalikan dari modul dipublikasikan. Terkait dengan hal ini, bagaimana modul mengimplementasikan fungsi yang diperlukan tidaklah relevan. Tidaklah penting, bagaimana objek melakukan fungsinya, selama fungsi terjadi. Dalam sistem berorientasi objek, menggabungkan enkapsulasi dengan prinsip menyembunyikan informasi, mendukung dalam memperlakukan objek sebagai kotak hitam.

Fakta bahwa kita dapat menggunakan objek dengan metode panggilan adalah kunci untuk reusabilitas karena melindungi cara kerja internal objek dari perubahan dalam sistem luar, dan itu membuat sistem tidak terpengaruh ketika perubahan dilakukan pada suatu objek. Pada Gambar 1.4, perhatikan bagaimana pesan tambah (*create*) dikirim ke suatu objek, tetapi algoritme internal yang diperlukan untuk menanggapi pesan disembunyikan dari bagian lain dari sistem. Satu-satunya informasi yang perlu diketahui objek adalah serangkaian operasi, atau metode, yang dapat dilakukan objek lain dan pesan apa yang perlu dikirim untuk memicunya.

1.4.4 Pewarisan

Literatur pemodelan data menunjukkan bahwa penggunaan warisan untuk mengidentifikasi kelas objek tingkat yang lebih tinggi atau lebih umum. Himpunan atribut dan metode dapat diatur menjadi superkelas. Biasanya, kelas diatur dalam hierarki di mana superkelas atau kelas umum, berada di bagian atas dan subkelas, atau kelas tertentu, berada di bagian bawah. Pada Gambar 1.5, Orang adalah superkelas untuk kelas Dokter dan Pasien. Dokter, pada gilirannya, adalah superkelas untuk Dokter Umum dan Spesialis. Perhatikan bagaimana Kelas (misalnya, Dokter) dapat berfungsi

sebagai superkelas dan subkelas secara bersamaan. Hubungan/ relasi antara kelas dan superkelasnya dikenal sebagai pewarisan (*inheritance*). Pada Gambar 1.5, seorang Dokter Umum adalah tipe Dokter, yang merupakan tipe Orang.

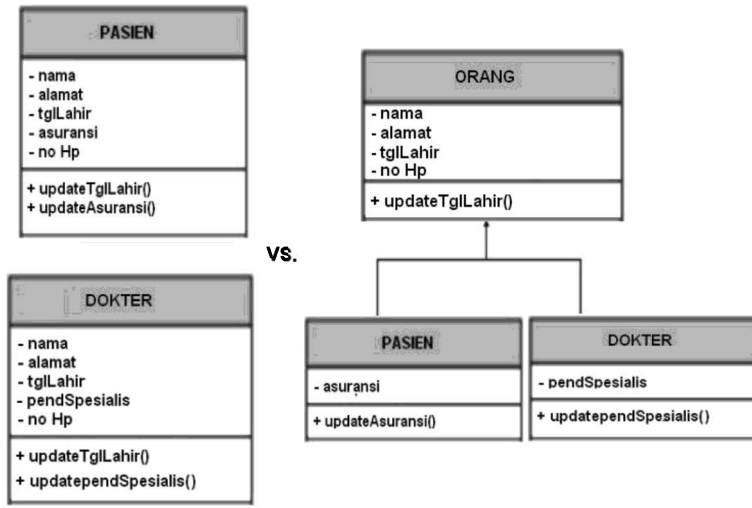


Gambar 1.5 Hierarki kelas dengan kelas abstrak dan konkret

Subkelas mewarisi atribut dan metode yang sesuai dari superkelas di atasnya. Artinya, setiap subkelas berisi atribut dan metode dari superkelas induknya. Misalnya, Gambar 1.5 menunjukkan bahwa baik Dokter dan Pasien adalah subkelas dari kelas Orang dan karenanya mewarisi atribut serta metode kelas Orang. Warisan membuatnya lebih mudah dalam mendefinisikan kelas. Sebagai pengganti mengulangi atribut dan metode di kelas Dokter dan Pasien secara terpisah, atribut serta metode yang umum untuk keduanya ditempatkan di kelas Orang dan diwarisi oleh kelas-kelas di bawahnya. Perhatikan bagaimana hierarki warisan kelas objek yang jauh lebih efisien daripada objek yang sama tanpa hierarki warisan (Gambar 1.6).

Sebagian besar kelas di seluruh hierarki mengarah pada instans; setiap kelas yang memiliki instans disebut kelas konkret. Misalnya, jika Jimi Akbar dan Eli Manise adalah instans dari kelas Pasien, Pasien akan dianggap sebagai kelas konkret (Gambar 1.3). Beberapa kelas tidak menghasilkan instans karena mereka

digunakan hanya sebagai template untuk kelas lain yang lebih spesifik (terutama kelas yang terletak lebih tinggi dalam hierarki). kelas-kelas tersebut disebut sebagai kelas abstrak. Orang adalah kelas abstrak. Sebagai pengganti membuat objek dari Orang, kita buat instans yang mewakili kelas Spesialis yang lebih spesifik dan Pasien, kedua jenis Orang (Gambar 1.5).



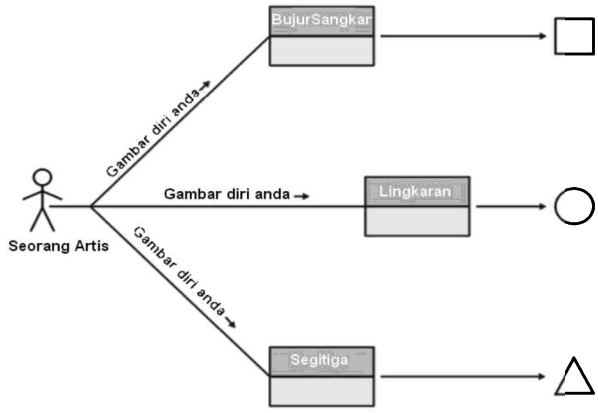
Gambar 1.6 Keuntungan generalisasi/pewarisan

1.4.5 Polimorfisme dan Pengikatan Dinamis

Polimorfisme berarti bahwa pesan yang sama dapat ditafsirkan secara berbeda oleh kelas objek yang berbeda. Untungnya, kita tidak perlu khawatir dengan bagaimana sesuatu dilakukan saat menggunakan objek. Kita hanya dapat mengirim pesan ke objek, dan objek itu akan bertanggung jawab untuk menafsirkan pesan dengan tepat. Misalnya, seorang Artis (seniman) mengirim pesan, gambarkan diri Anda ke objek persegi, objek lingkaran, dan objek segitiga, hasilnya akan sangat berbeda, meskipun pesannya sama. Perhatikan pada Gambar 1.7, bagaimana setiap objek merespons dengan tepat (dan berbeda) meskipun pesannya identik.

Polimorfisme dimungkinkan melalui pengikatan dinamis. Dinamis atau terlambat, mengikat adalah teknik yang menunda mengetik objek sampai *run-time*. Metode khusus yang sebenarnya tidak dipilih oleh sistem berorientasi objek sampai sistem berjalan. Hal ini berbeda dengan pengikatan statis. Dalam sistem yang terikat secara statis, jenis objek ditentukan pada waktu kompilasi. Oleh karena itu, pengembang harus memilih metode mana yang harus disebut, sebagai pengganti membiarkan sistem melakukannya. Itulah sebabnya sebagian besar bahasa pemrograman tradisional memiliki logika keputusan yang rumit berdasarkan berbagai jenis objek dalam suatu sistem.

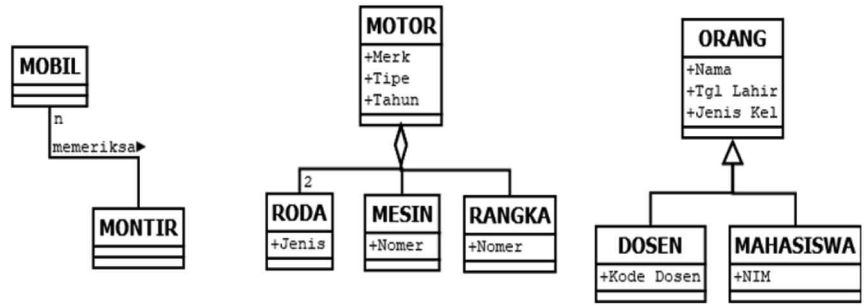
Misalnya, dalam bahasa pemrograman tradisional, sebagai pengganti mengirim pesan "gambarkan diri Anda" untuk berbagai jenis objek grafis pada Gambar 1.7. Kita harus menulis Logika keputusan menggunakan pernyataan kasus atau seperangkat pernyataan untuk menentukan jenis objek grafis apa yang ingin kita gambar dan kita harus menyebutkan setiap fungsi tersebut secara berbeda (misalnya, menggambar persegi, menggambar lingkaran, atau menggambar segitiga). Hal ini jelas membuat sistem jauh lebih rumit dan tidak menyenangkan untuk dipahami.



Gambar 1.7 Polimorfisme

1.4.6 Relasi Antarkelas/Objek

Terdapat 3 tipe relasi dalam sistem berorientasi objek, yaitu asosiasi, agregasi, dan *inheritance*/pewarisan. Asosiasi merupakan representasi relasi antarkelas-kelas yang berbeda, hubungan/*link* di antara objek-objek kelas adalah instans dari asosiasi. Agregasi merupakan relasi bagian dari. Relasi ini menyiratkan relasi yang kuat di antara sebuah objek dengan objek-objek lain yang secara nyata merupakan bagian-bagiannya. *Inheritance*/pewarisan merupakan relasi yang memiliki sifat-sifat, yaitu satu atau beberapa kelas dapat mewarisi atribut-atribut dan operasi kelas yang lain. Relasi tipe ini terdapat kelas yang bersifat umum (generalisasi) dan sementara kelas yang lain bersifat khusus (spesialisasi). Kelas yang khusus (subkelas) mewarisi atribut dan operasi kelas umum/super (atau superkelas). Relasi pewarisan antarkelas disebut juga relasi adalah atau relasi macam dari. Contoh relasi asosiasi, agregasi, dan pewarisan tersaji pada Gambar 1.8 (a), (b), (c).



(a) Asosiasi **(b) Agregasi** **(c) Pewarisan**

Gambar 1.8 Tipe Relasi antarkelas/objek

BAB 2
DIAGRAM USE CASE

Diagram use case menjelaskan semua skenario penggunaan (use case) yang akan dikembangkan sistem, yaitu tentang fungsi-fungsi apa yang sistem bisa/harus dilakukan, tetapi tidak membahas aspek detail implementasi. Aspek detail akan dicakup dalam diagram lain, seperti diagram kelas atau diagram interaksi/ sekuens. Diagram use case mengekspresikan siapa pengguna yang benar-benar akan bekerja dengan sistem yang akan dibangun.

Adapun use case merupakan konsep dasar dari banyak metode pengembangan berorientasi objek dan dapat diterapkan selama proses analisis serta desain. Use case merupakan representasi kebutuhan pelanggan terhadap sistem yang akan dibangun.

Secara khusus, penggunaan diagram use case untuk menjawab pertanyaan-pertanyaan berikut.

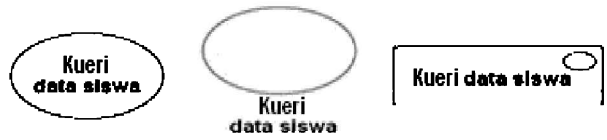
1. Apa yang dijelaskan? (sistem)
2. Siapa yang berinteraksi dengan sistem? (para aktor)
3. Apa yang aktor bisa lakukan? (use case)

Diagram use case mendokumentasikan persyaratan atau kebutuhan yang harus dipenuhi sistem. Kondisi tersebut akan sangat berguna dalam menyusun aspek desain teknis yang lebih terperinci. Apabila use case ditentukan secara tidak tepat atau salah, dalam beberapa keadaan, dapat berdampak pada biaya pengembangan dan pemeliharaan yang meningkat, pengguna tidak puas, dan lainnya. Sebagai konsekuensinya, sistem yang digunakan akan dinilai kurang berhasil dan investasi yang dilakukan dalam pengembangan sistem tidak membawa pengembalian (*return of investment*) yang diharapkan. Untuk itu, aspek kehati-hatian dan pendekatan yang sistematis dalam membuat use case sangatlah penting.

2.1 Use Case

Sebuah use case merupakan serangkaian langkah demi langkah (*step-by-step*) interaksi aktor (pengguna) dalam berinteraksi atau berkomunikasi dengan sistem dalam rangka menyelesaikan sebuah problem/masalah/tujuan (fungsi). Use case menjelaskan fungsionalitas sistem yang akan dikembangkan. Hal ini mencakup sejumlah fungsi yang dijalankan saat menggunakan sistem ini. Use case memberikan manfaat nyata bagi satu atau beberapa aktor yang berkomunikasi dengan use case ini. Diagram use case tidak mencakup struktur internal dan implementasi aktual dari use case. Secara umum, use case dipicu baik oleh aktor atau oleh sebuah kejadian/peristiwa.

Use case ditentukan berdasar keinginan pelanggan dan menganalisis masalah yang ditentukan dalam bahasa alami sebagai dasar dalam menganalisis kebutuhan/persyaratan sistem. Use case biasanya dinotasikan sebagai elips, dengan nama use case berada langsung di dalam atau di bawah elips. Use case dapat dinotasikan juga melalui kotak persegi panjang yang berisi nama di tengah dan elips kecil di sudut kanan atas. Alternatif notasi yang berbeda untuk use case "Kueri data siswa" diilustrasikan dalam Gambar 2.1. Alternatif pertama, berupa elips yang berisi nama use case adalah yang paling umum digunakan.



Gambar 2.1 Notasi alternatif untuk use case

Koleksi semua use case secara bersama-sama menjelaskan fungsionalitas yang disediakan sistem (perangkat lunak sistem). Use case umumnya dikelompokkan dalam sebuah kotak persegi panjang yang melambangkan batas-batas sistem. Gambar 2.2 memperlihatkan sistem Administrasi Siswa dengan tiga use case, yaitu (1) Kueri data siswa, (2) Isu sertifikat, dan (3) Informasi ujian. Use case ini dipicu oleh aktor Dosen.

Gambar 2.2 Representasi batasan sistem

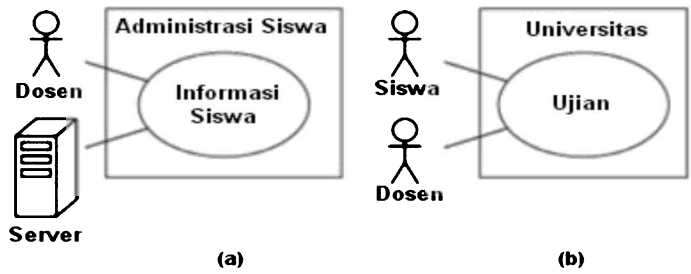
2.2 Aktor

Aktor merupakan pengguna yang berinteraksi dengan sistem, dalam konteks use case yang terlibat dengannya. Perhatikan Gambar 2.2, aktor Dosen dapat meng-kueri data siswa, mengumumkan ujian, dan menerbitkan sertifikat (isu sertifikat). Alternatif notasi aktor ditunjukkan pada Gambar 2.3. Aktor dapat berupa manusia (siswa atau dosen) atau device/nonmanusia (server atau komputer).

Gambar 2.3 Notasi alternatif bagi aktor

Aktor yang berinteraksi secara langsung dalam penggunaan sistem disebut aktor aktif, yang berarti bahwa aktor tersebut sebagai pemicu dalam eksekusi use case. Apabila sebuah aktor dalam interaksinya dalam menggunakan sistem melibatkan aktor lain maka aktor tersebut bertindak sebagai aktor pasif. Sebuah aktor yang pasif, hanya menyediakan fungsionalitas dalam pelaksanaan use case dari aktor lain yang aktif. Pada Gambar 2.4(a), Dosen sebagai aktor aktif (terkadang disebut aktor primer/utama), sedangkan Server (e-mail) sebagai aktor pasif (terkadang disebut aktor sekunder). Kedua aktor tersebut berperan dalam eksekusi use case informasi siswa.

Pada Gambar 2.4(b), aktor sekunder tidak harus pasif, baik Dosen maupun Siswa terlibat aktif dalam eksekusi use case ujian, dengan Siswa sebagai penerima manfaat utama dan Dosen sebagai penerima manfaat yang lebih rendah, tetapi diperlukan dalam eksekusi use case. Secara grafis, tidak ada perbedaan antara aktor primer dan sekunder, antara aktor aktif dan pasif, serta antara aktor manusia dan nonmanusia.

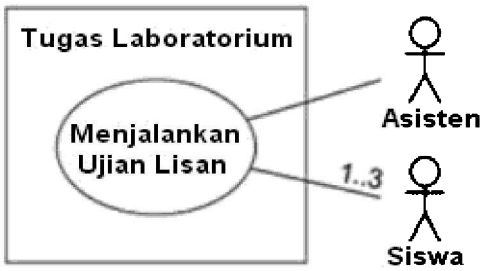


Gambar 2.4 Contoh aktor

Pada dasarnya, aktor (pengguna) harus berada di luar sistem, bukan bagian dari sistem dan oleh karenanya tidak pernah diimplementasikan. Namun, data pengguna tersedia dalam sistem dan dapat diwakili (misal, oleh kelas). Bagaimanapun, terkadang sulit untuk memutuskan apakah elemen (aktor) tersebut merupakan bagian dari sistem yang akan diimplementasikan atau berfungsi sebagai aktor (di luar sistem). Pada Gambar 2.4(a), Server sebagai aktor — bukan bagian dari sistem, tetapi diperlukan untuk eksekusi use case informasi siswa.

2.3 Asosiasi

Seorang aktor terhubung dengan use case melalui asosiasi (dinotasikan dengan garis padat/solid), yang menyatakan bahwa aktor berinteraksi/berkomunikasi dengan sistem melalui penggunaan fungsi tertentu. Setiap aktor harus berinteraksi dengan setidaknya satu use case, dan sebaliknya setiap use case harus menjalin hubungan dengan setidaknya satu aktor. Dengan demikian, asosiasi bersifat biner, yang berarti minimal ditentukan oleh satu use case dan satu aktor. Asosiasi memiliki kardinalitas atau multiplisitas. Apabila multiplisitas yang lebih besar dari 1 ditentukan di akhir asosiasi aktor, ini berarti bahwa lebih dari satu aktor terlibat dalam eksekusi use case. Pada Gambar 2.5, menunjukkan bahwa terdapat satu hingga tiga Siswa dan hanya satu Asisten yang terlibat dalam eksekusi use case Ujian Lisan. Jika tidak ada multiplisitas yang ditentukan di akhir asosiasi aktor, nilai 1 diasumsikan sebagai nilai default. Multiplisitas di akhir asosiasi use case sebagian besar tidak dibatasi. Oleh karena itu, hanya jarang ditentukan secara eksplisit.



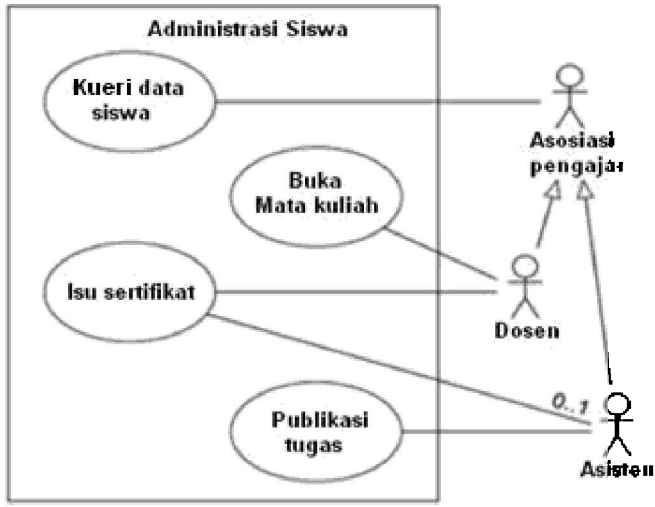
Gambar 2.5 Multiplisitas pada asosiasi

2.4 Relasi Antaraktor

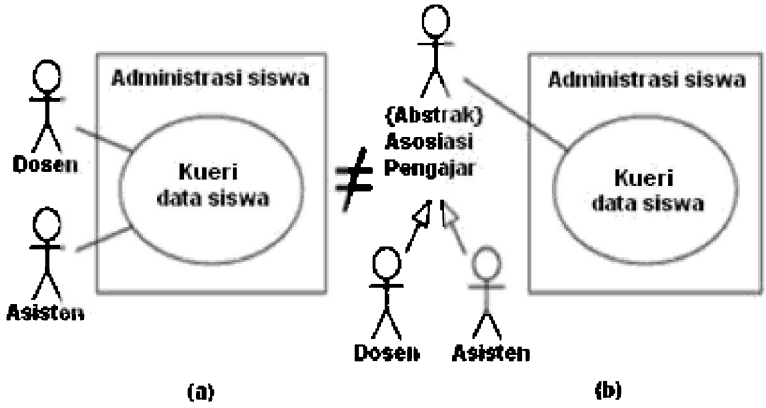
Aktor dapat berperan secara umum dan beberapa use case dapat digunakan oleh berbagai aktor. Kondisi dimungkinkan ketika, misalnya, tidak hanya Dosen, tetapi juga Asisten (atau staf Pengajar) diizinkan untuk mengakses data siswa. Untuk mengekspresikan hal ini, aktor dapat digambarkan dalam hubungan warisan (generalisasi) satu sama lain. Ketika seorang aktor Y (subaktor) mewarisi aktor X (aktor super) maka Y dapat mengeksekusi semua use case yang melibatkan X. Secara sederhana, generalisasi mengekspresikan relasi "adalah" (dengan simbol anak panah menuju aktor super). Contoh pada Gambar 2.6, aktor Dosen dan Asisten mewarisi aktor Asosiasi (staf) Pengajar, yang berarti bahwa setiap Dosen dan Asisten adalah Asosiasi Pengajar. Dengan demikian, Asosiasi Pengajar (Dosen dan Asisten) dapat mengeksekusi use case Kueri data siswa. Di samping itu, Dosen juga dapat mengeksekusi use case buka mata kuliah dan isu sertifikat, sementara Asisten mengeksekusi use case publikasi tugas serta isu sertifikat (opsional, sebagaimana sifat multiplisitas 0..1).

Terdapat perbedaan mendasar ketika dua aktor berpartisipasi dalam use case tunggal dan ketika dua aktor memiliki aktor super yang berpartisipasi dalam use case tersebut, sebagaimana Gambar 2.7(a) serta Gambar 2.7(b). Untuk contoh (a), kedua aktor harus

berpartisipasi secara bersama-sama dalam eksekusi use case Kueri data siswa. Sementara itu, contoh (b), kedua aktor masing-masing mewarisi aktor asosiasi Pengajar sehingga setiap aktor dapat berpartisipasi dalam eksekusi use case secara individu.



Gambar 2.6 Contoh generalisasi untuk aktor



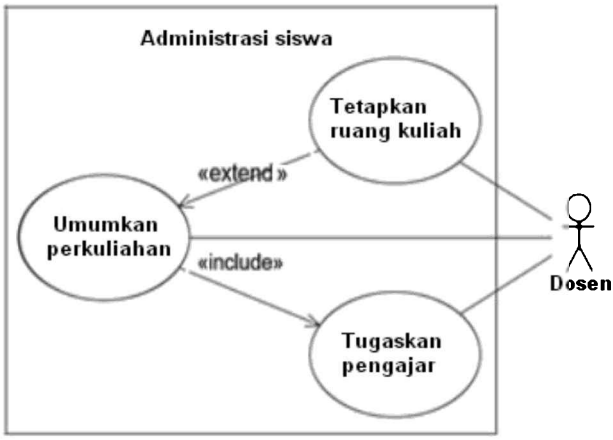
Gambar 2.7 Contoh dengan dan tanpa generalisasi

Dalam sebuah hal, aktor tidak memiliki ada instans aktor, aktor dapat dilabeli dengan kata kunci {abstrak}. Aktor staf Pengajar pada Gambar 2.7(b) adalah contoh aktor abstrak. Aktor

abstrak diperlukan guna mengekspresikan bahwa Dosen atau Asisten dapat terlibat dalam eksekusi use case kueri data siswa secara individu karena adanya relasi warisan (generalisasi). Relasi warisan karena adanya sifat umum (atau yang sama) dari subaktor di bawahnya yang dikelompokkan dan dideskripsikan pada satu titik aktor yang levelnya lebih tinggi, yaitu aktor super. Generalisasi merupakan konsep dasar orientasi objek yang memiliki sifat dari khusus ke umum (subaktor ke aktor super), dan sebaliknya dari umum ke khusus (aktor super ke subaktor) disebut spesialisasi.

2.5 Relasi Antar-use Case

Antar-use case juga dapat berelasi. Beberapa tipe relasi antar-use case meliputi: «include», «extend», dan generalisasi use case.



Gambar 2.8 Contoh relasi <<include>> dan <<extend>>

Gambar 2.9 Contoh titik-titik ekstensi dan kondisi

Apabila use case A menyertakan use case B dalam eksekusinya maka perilaku B diintegrasikan ke dalam perilaku A (dinotasikan sebagai panah putus-putus dari A ke B berlabel «include»). Use case A, dalam hal ini disebut sebagai use case dasar dan B sebagai use case yang penyerta. Use case A memerlukan perilaku use case B terkait dengan fungsionalitasnya, sementara use case B dapat dieksekusi sendiri. Penggunaan «include» dapat dianalogikan sebagai pemanggilan subroutine dalam bahasa pemrograman prosedural. Diagram use case pada Gambar 2.8, use case Umumkan Perkuliahan memiliki relasi «include» dengan use case Tugaskan Pengajar (Asisten) sehingga ketika perkuliahan baru diumumkan, use case Tugaskan Pengajar juga harus dieksekusi. Aktor Dosen terlibat dalam eksekusi kedua use case tersebut. Karena use case Tugaskan Pengajar sebagai use case penyerta dan eksekusinya dapat dilakukan secara independen maka Pengajar dapat juga ditugaskan

pada perkuliahan yang ada. Satu use case dapat mencakup beberapa use case lainnya. Satu use case juga dapat disertakan oleh beberapa use case yang berbeda. Namun, perlu diperhatikan bahwa tidak akan ada siklus yang muncul di antaranya.

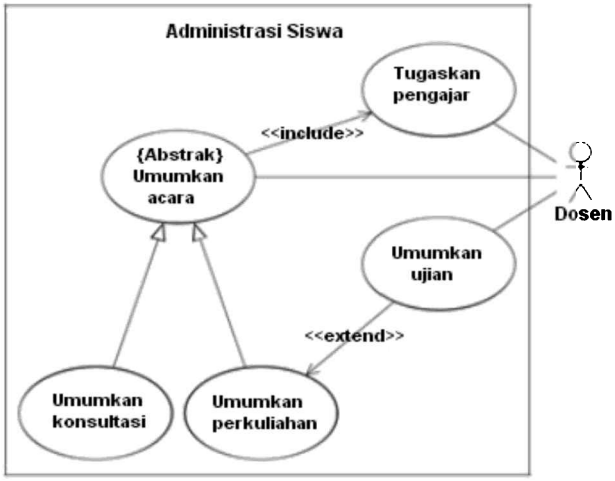
Di samping relasi antar-use case berupa «include», terdapat juga bentuk relasi lain, yaitu relasi «extend» (dinotasikan sebagai panah putus-putus dari use case B ke A). Apabila use case B berada dalam relasi «extend» dengan use case A maka A dapat menggunakan perilaku B, tetapi bersifat opsional (bila perlu). Use case B dapat diaktifkan oleh A untuk mengintegrasikan perilaku B di A. Dalam kondisi ini, A kembali disebut sebagai use case dasar dan B sebagai use case ekstensi serta kedua use case juga dapat dieksekusi secara independen satu sama lain. Sebagaimana contoh pada Gambar 2.8, dua use case Umumkan Perkuliahan dan Pesan Ruang Kuliah berada dalam relasi «extend». Ketika perkuliahan baru diumumkan, dapat diikuti dengan eksekusi pemesanan atau penetapan ruang kuliah (bersifat opsional). Use case dapat bertindak sebagai use case yang diekstensi beberapa kali atau dapat diekstensi oleh beberapa use case.

Kondisi yang harus dipenuhi use case dasar dalam menyisipkan perilaku use case yang diekstensi dapat ditentukan untuk setiap relasi «extend». Kondisi tersebut dinyatakan dalam kurung kurawal, dengan catatan yang terhubung dengan relasi «extend» yang sesuai. Dua contoh relasi ditampilkan pada Gambar 2.9. Use case Umumkan Perkuliahan, ruang kuliah hanya dapat dipesan jika ruangan kosong atau *free*. Sementara itu, use case Umumkan Ujian hanya dapat dibuat jika data yang diperlukan telah diinputkan.

Titik ekstensi bertindak sebagai titik/saat ketika perilaku use case ekstensi harus dimasukkan dalam use case dasar. Titik ekstensi ditulis langsung dalam use case, sebagaimana yang diilustrasikan dalam use case Umumkan Acara pada Gambar 2.9.

Generalisasi dapat juga terjadi dalam relasi antar-use case. Apabila use case A menggeneralisasi use case B maka B mewarisi perilaku A, dan perilaku B dapat mengekstensi atau menimpanya. Selanjutnya, B juga mewarisi semua hubungan/relasi dari A. Oleh karenanya, kalau B mengadopsi fungsi dasar A, B dapat memutuskan sendiri bagian A yang mana yang diadopsi atau diubah. Apabila sebuah use case diberi label {abstract}, kasus tersebut tidak dapat dieksekusi secara langsung; hanya use case spesifik yang mewarisi dari use case abstrak yang dapat dieksekusi.

Diagram use case Gambar 2.10 memperlihatkan contoh generalisasi use case. Use case abstrak Umumkan Acara, meneruskan properti dan perilakunya ke use case Umumkan Perkuliahan dan Umumkan Konsultasi. Sebagai hasil dari relasi «include», kedua use case harus menjalankan perilaku use case Tugaskan Pengajar. Ketika perkuliahan diumumkan, ujian juga dapat diumumkan pada saat yang sama. Kedua use case mewarisi hubungan dari use case Umumkan Acara kepada aktor Dosen. Dengan demikian, semua use case terhubung ke setidaknya satu aktor (prasyarat yang diagram use case yang benar).



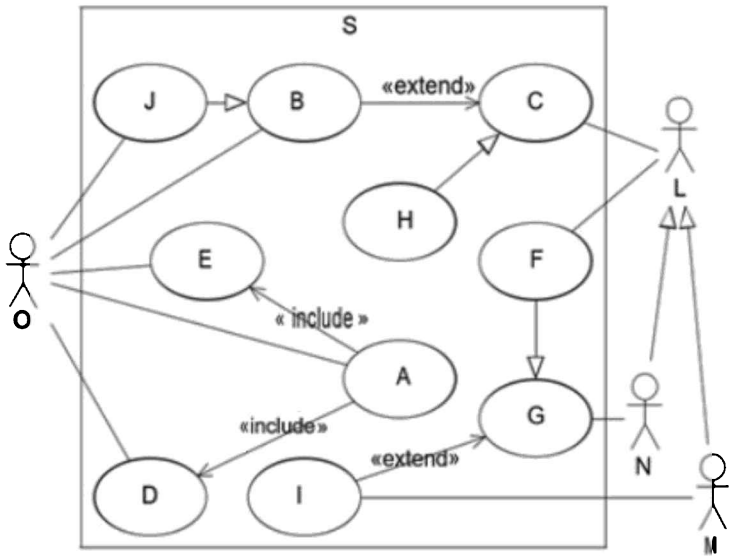
Gambar 2.10 Contoh generalisasi dari use case

Generalisasi memungkinkan kita untuk mengelompokkan fitur umum dari dua use case Umumkan Perkuliahan dan Umumkan Konsultasi. Ini berarti bahwa kita tidak perlu mencontoh relasi «include» dan asosiasi dengan Dosen dua kali.

2.6 Relasi dalam Diagram Use Case

Relasi dalam diagram use case, yang use case-nya berinteraksi satu sama lain, diilustrasikan pada Gambar 2.11.

1. Use case A memiliki relasi <<include>> dengan use case E dan D. Seorang aktor O terlibat dalam eksekusi ketiga use case tersebut.
2. Use case H mewarisi use case C. Karena use case C dieksekusi oleh aktor L, aktor L juga harus terlibat dalam eksekusi use case H. Sementara itu, aktor N dan M mewarisi aktor L. Oleh karenanya, baik use case C maupun H juga dapat dieksekusi oleh aktor M atau N.



Gambar 2.11 Contoh relasi dalam diagram use case

3. Use case J mewarisi use case B. Akibat relasi warisan tersebut, aktor O terlibat dalam eksekusi use case J. Namun, relasi dengan O juga dimodelkan untuk J secara langsung. Konsekuensi dari hal ini adalah terdapat **dua aktor dalam peran O** terlibat dalam eksekusi J. Kedua aktor ini dapat bersesuaian (identik).

4. Use case F mewarisi use case G. Akibat relasi pewarisan, aktor N terlibat dalam eksekusi use case F. Use case F dapat juga dieksekusi secara langsung oleh aktor L. Di samping itu, karena relasi warisan dari aktor L, maka aktor N, dan M juga terlibat dalam eksekusi use case F. Keterlibatan aktor N dalam eksekusi use case F (yang juga mewarisi use case G), dan use case G, bisa jadi aktor N tersebut identik.

5. Use case I dapat mengekstensi use case F via use case G (akibat relasi warisan).
6. Use case J dapat mengekstensi use case H (via relasi warisan dari B ke J dan dari C ke H).

2.7 Membangun Diagram Use Case

Selanjutnya, bagaimana kita membangun diagram use case? Berikut ini dijelaskan prinsip-prinsip bagaimana membangun diagram use case secara garis besar.

2.7.1 Identifikasi Aktor dan Use Case

Ada dua cara dalam mengidentifikasi use case dalam desain sistem prospektif, yaitu:

1. analisis dokumen kebutuhan, dan
2. analisis keinginan pengguna.

Pada cara pertama, dokumen kebutuhan merupakan spesifikasi bahasa alami yang menjelaskan apa yang diinginkan pelanggan dari sebuah sistem. Mereka harus mendokumentasikan dengan relatif tepat siapa yang akan menggunakan sistem dan bagaimana mereka akan menggunakannya. Untuk cara kedua, dalam menemukan use case, kita perlu terlebih dahulu mengidentifikasi pengguna—yaitu para aktor. Dalam mengidentifikasi para aktor yang muncul dalam diagram use case, beberapa pertanyaan berikut perlu dijawab, yaitu:

1. Siapa pengguna use case utama?
2. Siapa yang membutuhkan dukungan untuk pekerjaan sehari-hari mereka?

3. Siapa yang bertanggung jawab dalam administrasi sistem?
4. Apa sistem perangkat eksternal/(perangkat lunak) yang harus dikomunikasikan oleh sistem?
5. Siapa yang memiliki minat pada hasil sistem?

Setelah aktor teridentifikasi, kemudian identifikasi use case dengan mengajukan beberapa pertanyaan berikut tentang aktor:

1. Apa tugas utama yang harus dilakukan seorang aktor?
2. Adakah keinginan seorang aktor untuk meminta atau memodifikasi informasi yang terkandung dalam sistem?
3. Adakah keinginan seorang aktor untuk menginformasikan sistem tentang perubahan-perubahan dalam sistem lain?
4. Haruskah seorang aktor diberi tahu tentang kejadian tak terduga dalam sistem?

Dalam banyak hal, pemodelan use case dilakukan secara berulang dan bertahap. Dengan demikian, perlunya memulai dengan kebutuhan "level atas" yang mencerminkan tujuan bisnis yang harus diupayakan dengan perangkat lunak. Selanjutnya, perlu merevisi/memperbaikinya sampai pada level teknis, yaitu menentukan apa yang sistem dapat dilakukan. Sebagai contoh, persyaratan "level atas" untuk sistem administrasi universitas, bisa jadi sistem tersebut dapat digunakan untuk administrasi kemahasiswaan sebuah Universitas (level atas). Selanjutnya, spesifikasi kebutuhan tersebut diperbaiki/dikembangkan dengan lebih detail, bahwa sistem mampu mengelola pendaftaran siswa baru untuk studi, mengelola studi mereka serta menyimpan nilai-nilai hasil studinya, dan lainnya (level teknis).

2.7.2 Mendeskripsikan Use Case

Diagram use case yang dibuat harus cukup jelas/detail, termasuk pemilihan nama use case yang pendek/ringkas. Penggambaran diagram use case dan penamaannya yang jelas dan ringkas, sangatlah penting guna menghindari adanya risiko salah interpretasi dari pembaca/pengguna.

Pedoman umum dalam mendeskripsikan use case adalah sekitar 1-2 halaman per use case. Pendekatan terstruktur untuk deskripsi use case berisi informasi berikut (Tabel 2.1 sebagai contoh).

1. Nama.
2. Deskripsi singkat.
3. Prakondisi: prasyarat untuk keberhasilan eksekusi.
4. Pascakondisi: status sistem setelah eksekusi berhasil.
5. Situasi kesalahan: kesalahan yang relevan dengan domain masalah.
6. Status sistem tentang terjadinya kesalahan.
7. Aktor yang berkomunikasi dengan use case.
8. Pemicu: peristiwa yang memulai/memulai use case.
9. Proses standar: langkah individu yang harus diambil.
10. Proses alternatif: penyimpangan dari proses standar.

Tabel 2.1 berisi deskripsi use case Pemesanan Ruang Kuliah dalam sistem administrasi Siswa. Deskripsinya dapat disederhanakan tetapi diharapkan cukup memenuhi tujuan. Proses standar dan proses alternatif dapat disempurnakan lebih lanjut atau situasi kesalahan lainnya dan proses alternatif dapat

dipertimbangkan. Dalam proyek nyata, perincian use case dapat berasal dari kebutuhan dan keinginan pelanggan.

Tabel 2.1 Contoh Deskripsi Use Case Pemesanan Ruang Kuliah

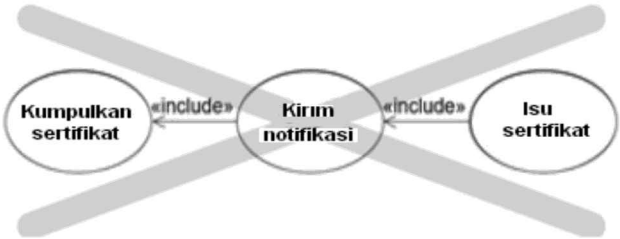
Nama:	Pemesanan ruang kuliah
Deskripsi:	Seorang karyawan memesan ruang kuliah suatu event
Prakondisi:	Karyawan diotorisasi memesan ruang kuliah Karyawan log-in ke sistem
Pascakondisi:	Ruang Kuliah telah dipesan
Kondisi error	Tidak tersedia ruang kuliah yang kosong
Status sistem ketika terjadi error	Karyawan gagal/belum berhasil memesan ruang kuliah
Aktor	Karyawan
Pemicu	Karyawan memerlukan ruang kuliah
Proses standar	1> Karyawan memilih ruang kuliah 2> Karyawan memilih tanggal 3> Sistem mengonfirmasi bahwa ruang kuliah kosong 4> Karyawan konfirmasi pesanan
Proses Alternatif	3'> Ruang kuliah tidak kosong 4'> Sistem mengajukan sebuah alternatif ruang kuliah 5'> Karyawan memilih alternatif ruang kuliah dan konfirmasi pesanan

2.8 Bentuk Kesalahan yang Bisa Terjadi dalam Pembangunan

Beberapa bentuk kesalahan yang sering terjadi dalam membangun diagram use case, yakni sebagai berikut.

2.8.1 Proses Pemodelan

Adanya keinginan untuk memodelkan seluruh proses (bisnis) atau alur kerja dalam diagram use case secara umum sering terjadi dan itu merupakan hal yang tidak tepat. Contoh pada Gambar 2.12 menunjukkan fungsionalitas yang ditawarkan salah satu use case ini bukan bagian dari fungsionalitas yang ditawarkan use case lain karenanya use case harus dapat digunakan secara independen satu sama lain.



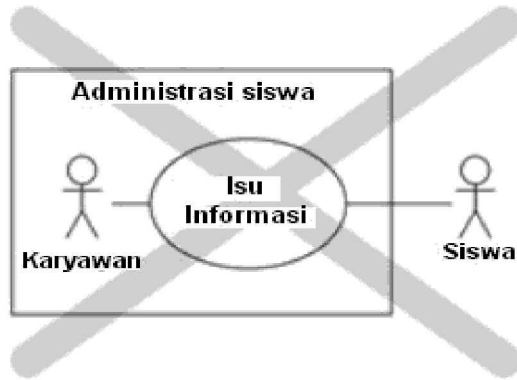
Gambar 2.12 Memodelkan proses yang tidak tepat

2.8.2 Menentukan Batas Sistem

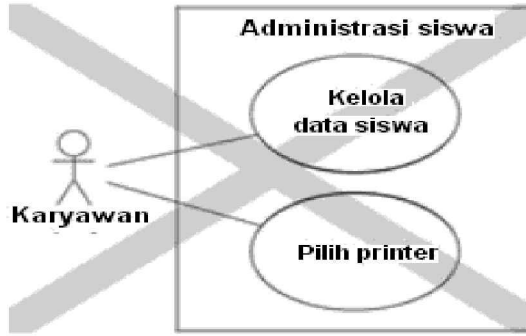
Dalam memodelkan diagram use case, harus dipertimbangkan dengan saksama dalam penggambaran batas diagram. Aktor harus selalu berada di luar batas sistem, dan jika aktor berada di dalam sistem, mereka adalah bagian dari sistem dan oleh karena itu mereka tidak boleh dimodelkan sebagai aktor. Contoh pada Gambar 2.13 menunjukkan bahwa aktor Karyawan digambarkan dalam batas sistem Administrasi siswa. Dalam hal Karyawan akan dipertimbangkan sebagai bagian dari sistem, mereka tidak boleh dimodelkan sebagai aktor.

2.8.3 Mencampur Tingkat Abstraksi yang Berbeda

Dalam mengidentifikasi use case, harus dipastikan bahwa use case berada pada tingkat abstraksi yang sama. Hindari use case berorientasi "tingkat atas" dengan use case berorientasi "teknis" dalam diagram yang sama. Contoh pada Gambar 2.14, pengelolaan data siswa (tingkat atas) dan pemilihan printer (teknis), ditampilkan bersama-sama.



Gambar 2.13 Batasan sistem yang salah



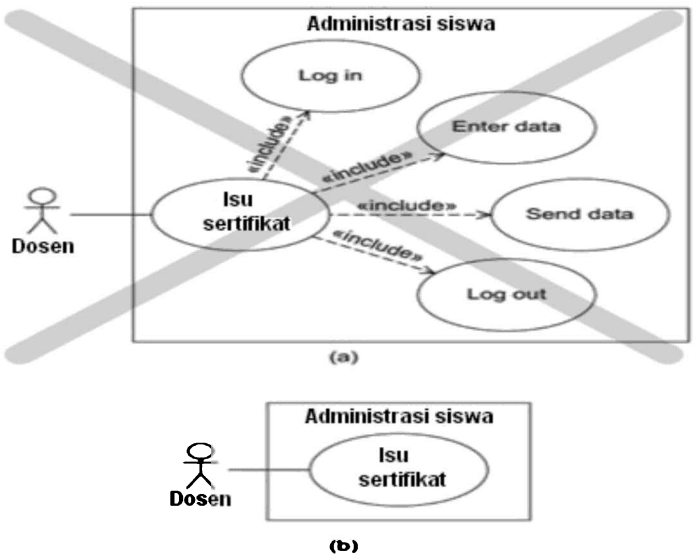
Gambar 2.14 Mencampur tingkat abstraksi yang berbeda

2.8.4 Dekomposisi Fungsional

Use case—dengan relasi <<include>> atau <<extend>>—selalu dapat dieksekusi secara independen. Bila mereka hanya dapat dieksekusi dalam lingkup use case lain dan tidak secara independen, mereka bukan use case dan tidak boleh digambarkan seperti itu. Fungsionalitas mereka kemudian harus dicakup dalam deskripsi use case yang menggunakannya. Contoh pada Gambar 2.15(a), use case Isu Sertifikat dipecah menjadi subfungsi individual yang diperlukan untuk menjalankan use case. Subfungsi ini dimodelkan sebagai use case meskipun terkadang tidak berarti use case independen (seperti enter data). Oleh karena itu, cukup dimodelkan sebagaimana Gambar 2.15(b). Informasi lain yang ditentukan dalam Gambar 2.15(a) harus dijelaskan dalam deskripsi use case.

2.8.5 Asosiasi yang Salah

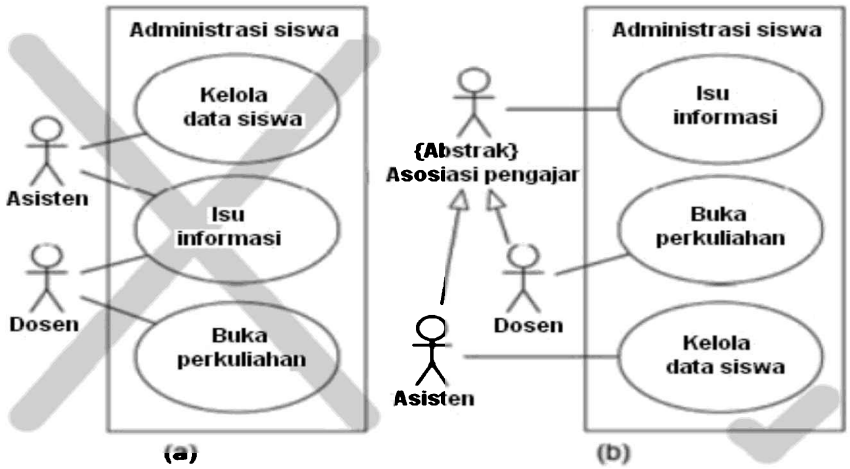
Dalam hal use case dikaitkan dengan dua aktor, tidaklah berarti bahwa salah satu aktor atau aktor lain terlibat dalam eksekusi use case. Hal itu berarti bahwa keduanya diperlukan dalam eksekusinya. Pada diagram use case Gambar 2.16(a), Asisten dan Dosen terlibat dalam eksekusi use case Isu Informasi, tetapi maksudnya tidaklah demikian. Untuk mengatasinya, dapat dikembangkan aktor baru yang abstrak, yaitu Asosiasi Pengajar, yang kedua aktor Asisten dan Dosen mewarisinya. Aktor Asosiasi Pengajar sekarang terhubung dengan use case Isu Informasi (Gambar 2.16(b)).



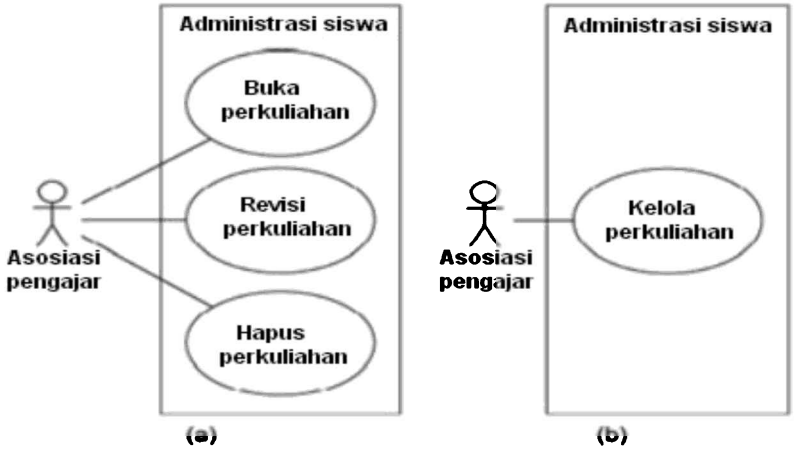
Gambar 2.15 Dekomposisional yang salah

2.8.6 Pemodelan Use Case Berlebihan

Sering adanya godaan untuk membuat use case terpisah untuk setiap situasi yang mungkin. Contoh pada Gambar 2.17(a), telah dimodelkan use case terpisah untuk membuat, memperbarui, dan menghapus perkuliahan (atau kursus). Ini menunjukkan berbagai opsi yang tersedia untuk mengedit perkuliahan dalam sistem. Dalam diagram use case seperti ini, cukup dimodelkan sebagaimana ditunjukkan pada Gambar 2.17(b). Langkah-langkah individual dapat dijelaskan dalam deskripsi prosesnya.



Gambar 2.16 Asosiasi yang salah



Gambar 2.17 Model use case redundant

2.9 Contoh Terapan

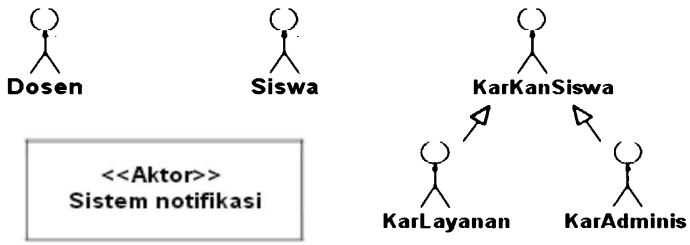
Membangun diagram use case sistem informasi kantor siswa dengan spesifikasi sebagai berikut.

1. Kegiatan administrasi sebuah universitas diproses oleh kantor siswa. Siswa dapat mendaftar untuk studi (matrikulasi) dan menarik diri (mundur) dari studi di sini. Matrikulasi melibatkan pendaftaran, yaitu mendaftar untuk studi.
2. Siswa menerima sertifikat dari kantor siswa. Sertifikat dicetak oleh karyawan. Dosen mengirimkan hasil penilaian ke kantor siswa. Sistem pemberitahuan, kemudian memberi tahu siswa secara otomatis bahwa sertifikat telah diterbitkan.
3. Ada dua jenis karyawan di kantor siswa, yaitu (a) yang secara eksklusif menangani administrasi data siswa (karyawan layanan) dan (b) yang melakukan tugas sisanya (karyawan administrasi), sedangkan semua karyawan (dari kedua tipe karyawan) dapat mengeluarkan informasi.
4. Karyawan administrasi mengeluarkan sertifikat ketika siswa datang untuk mengumpulkannya. Mereka juga membuka perkuliahan (atau membuat kursus). Ketika membuka perkuliahan, mereka dapat memesan ruang kuliah.

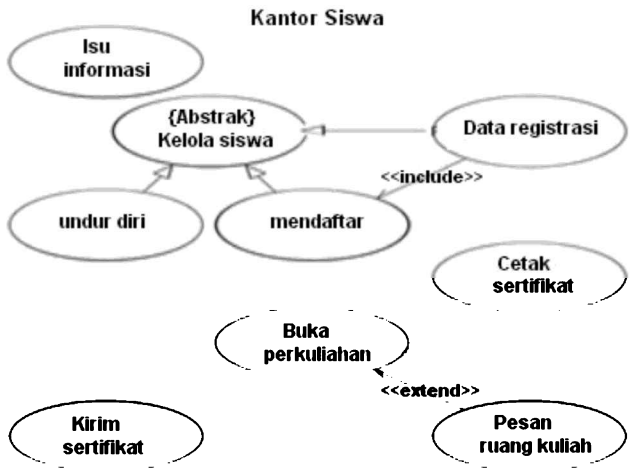
Berdasar spesifikasi kebutuhan tersebut, selanjutnya mengidentifikasi aktor dan relasinya satu sama lain. Kemudian menentukan use case, relasinya satu sama lain. Langkah berikutnya, mengaitkan aktor dengan use case-nya.

2.9.1 Mengidentifikasi Aktor

Berdasar spesifikasi tekstual, teridentifikasi kandidat aktor-aktor yang potensial, yaitu dosen, siswa, karyawan dari jenis karyawan layanan (karlayanan) dan karyawan administrasi (karadminis), serta sistem notifikasi. Dari kedua jenis karyawan menunjukkan perilaku umum, yaitu isu informasi, akan logis bila memperkenalkan karyawan kantor siswa (karkansiswa) sebagai superaktor dari karyawan layanan dan karyawan administrasi. Dengan mengasumsikan bahwa sistem notifikasi bukan bagian dari kantor siswa maka sistem notifikasi termasuk dalam daftar aktor. Hasil identifikasi para aktor ada pada Gambar 2.18.



Gambar 2.18 Hasil identifikasi para aktor



Gambar 2.19 Hasil identifikasi use case

2.9.2 Mengidentifikasi Use Case

Berdasar spesifikasi tekstual, beberapa kandidat use case dapat teridentifikasi. Di sini dibutuhkan use case mengumpulkan sertifikat, dengan seorang siswa akan terlibat. Use case tersebut tidak termasuk dalam sistem informasi. Namun, use case Cetak sertifikat yang berfungsi mencetak sertifikat dibutuhkan, termasuk di dalamnya adanya kebutuhan printer, sebagai bagian integral dari sistem yang akan dimodelkan.

Sistem juga memiliki fungsi registrasi, daftar, dan mundur/ batal. Ketiga fungsi tersebut dapat dikelompokkan dalam satu use case, yaitu Kelola siswa dan dilakukan oleh aktor karlayanan. Dari pengelompokan ketiga use case dapat berdampak atas hilangnya informasi bahwa matrikulasi termasuk pendaftaran untuk studi. Oleh karenanya, pengelompokan ketiga use case menjadi satu use case dibatalkan. Di sini, diekspresikan hubungan antara registrasi dan mendaftar dengan tipe relasi «include». Karena ketiga use case memiliki asosiasi dengan karlayanan secara umum, diperkenalkan

use case Kelola siswa (bersifat abstrak), dengan use case registrasi, mendaftar, dan mundur/batal mewarisi Kelola siswa.

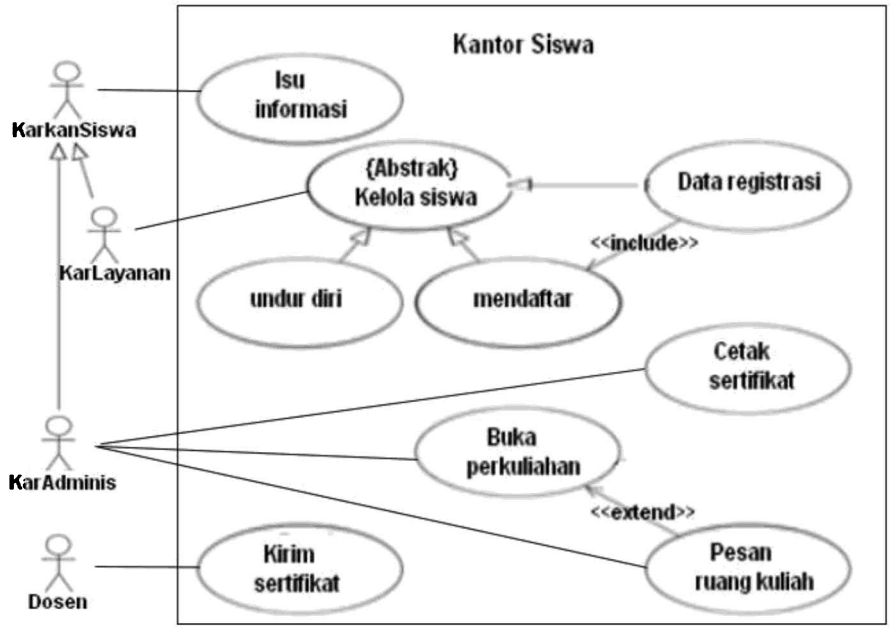
Dosen dapat mengeksekusi use case Kirim sertifikat. Ketika sertifikat dikirim ke kantor siswa, siswa yang akan menerima diberi tahu. Oleh karenanya, siswa hanya diberi tahu dalam konteks use case Kirim sertifikat. Apabila pemberitahuan kepada siswa tidak dapat dieksekusi secara independen, maka kegiatan ini bukanlah sebagai bagian dari use case sistem informasi. Di samping itu, sistem memiliki use case Isu informasi, reservasi ruang kuliah, dan buka perkuliahan/kursus, di mana reservasi ruang kuliah memperluas/mengekstensi use case Buka perkuliahan. Hasil identifikasi use case ada pada Gambar 2.19.

2.9.3 Mengidentifikasi Asosiasi

Setelah tahapan tersebut, langkah berikut kita harus mengaitkan aktor dan use case (Gambar 2.20). Tampak sekarang hanya terdapat dua aktor yang lebih sedikit daripada kandidat potensial yang teridentifikasi (Gambar 2.18). Tidak ada lagi siswa-siswa yang boleh menggunakan sistem informasi dalam bentuk sebagaimana yang telah dimodelkan dan tidak ada lagi sistem pemberitahuan karena ini dianggap sebagai bagian dari kantor siswa. Langkah berikutnya mendeskripsikan use case-use case yang telah diperoleh.

2.9.4 Mendeskripsikan Use Case

Tabel 2.2 menampilkan deskripsi salah satu use case, yaitu use case Cetak sertifikat. Deskripsi use case dalam Tabel 2.2 terkadang disebut sebagai use case perluasan (*extended use case*).



Gambar 2.20 Diagram use case sistem informasi kantor siswa

Tabel 2.2 Deskripsi Use Case Cetak Sertifikat

Nama:	Cetak sertifikat
Deskripsi singkat:	Atas permintaan seorang siswa, seorang karyawan mencetak sertifikat hasil perkuliahan
Prakondisi:	Semua data yang relevan ttg sertifikat telah dikirim, dan siswa telah memperoleh nilai
Pascakondisi:	Sertifikat dalam format tercetak telah tersedia
Situasi error:	Printer tidak bekerja
Status sistem dlm sbh kejadian error:	Sertifikat tidak tercetak
Aktor:	Karyawan administrasi
Pemicu:	Siswa meminta sertifikat tercetak
Proses standar:	(1) Siswa masuk ke kantor siswa dan meminta sertifikat (2) Karyawan admin menginput nomor matrikulasi siswa (3) Karyawan admin memilih sertifikat (4) Karyawanadmin meng"enter" perintah cetak (5) Sistem menginformasi perintah cetak sertifikat (6) Sertifikat diserahkan ke siswa
Proses Alternatif	(1') Siswa meminta sertifikat via e-mail (6') Sertifikat dikirim via pos

Contoh terapan sebuah use case dengan deskripsi atau format yang berbeda (adanya respons sistem) untuk use case pengambilan uang tunai di ATM (Anjungan Tunai Mandiri) dari sebuah sistem ATM, sebagaimana ditunjukkan pada Tabel 2.3.

Tabel 2.3 Deskripsi Use Case Pengambilan Uang Tunai dari ATM

Nama use case	Mengambil uang tunai dari ATM
Tujuan/sasaran	Mengizinkan nasabah mengambil uang tunai dari ATM
Aktor utama **Aktor tambahan**	Nasabah Bank -
Pemicu	Kartu ATM telah dimasukkan
Rangkaian kejadian khusus **Aksi aktor**	**Respons sistem**
1. Use case mulai ketika nasabah tiba di ATM dan memasukkan kartu ATM.	2. Memerlukan identifikasi.
3. Nasabah memberikan identifikasi.	4. Verifikasi identifikasi nasabah.
5. Nasabah memilih opsi mengambil uang tunai.	
6. Nasabah memasukkan jumlah yang diperlukan.	7. Mengeluarkan sejumlah uang yang diminta dan mengeluarkan kartu ATM.
8. Nasabah menerima uang dan kartu ATM serta meninggalkan ATM.	
Rangkaian kejadian alternatif	
No. 3: Apabila identifikasi tidak dapat diverifikasi, sediakan peluang tolak/error. No. 6: Apabila saldo tidak cukup, informasikan kesalahan ("saldo tidak cukup").	

BAB 3
DIAGRAM KELAS

Apabila use case dalam pembahasan sebelumnya merepresentasikan salah satu aspek perilaku sistem (atau aspek dinamis) maka diagram kelas merepresentasikan aspek/struktur statis dari sistem (Gambar 1.1 dan Gambar 1.2). Diagram kelas disebut statis karena elemen-elemen diagram kelas dan relasi di antaranya tidak berubah dari waktu ke waktu. Misalnya, elemen siswa memiliki nama, elemen matrikulasi memiliki nomor, dan relasi siswa mengikuti kursus/perkuliahan/pembelajaran.

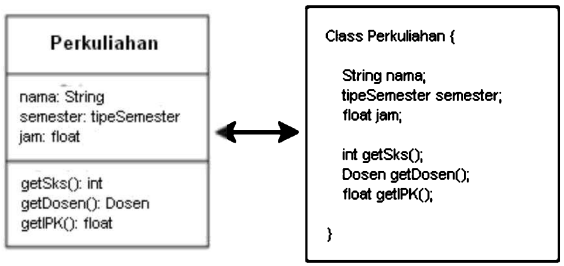
Diagram kelas diterapkan dalam berbagai fase proses pengembangan perangkat lunak (Gambar 1.2). Tingkat abstrak/ detail diagram kelas berbeda pada setiap fase (analisis, desain, dan implementasi). Pada tahap/fase awal proyek pengembangan, diagram kelas dipandang sebagai model konseptual dari sistem dan untuk menentukan kosakata/istilah/atribut yang digunakan. Selanjutnya, kosakata ini dapat diperbaiki menjadi bahasa pemrograman hingga fase implementasi.

Dalam konteks pemrograman berorientasi objek, diagram kelas memvisualisasikan kelas yang terdiri dari sistem perangkat lunak dan hubungan/relasi antara kelas-kelas ini. Diagram kelas umumnya digunakan sebagai sketsa/gambaran cepat atas struktur statis kebutuhan sistem, untuk tujuan dokumentasi, dan juga untuk membangkitkan kode program secara otomatis (dalam fase implementasi).

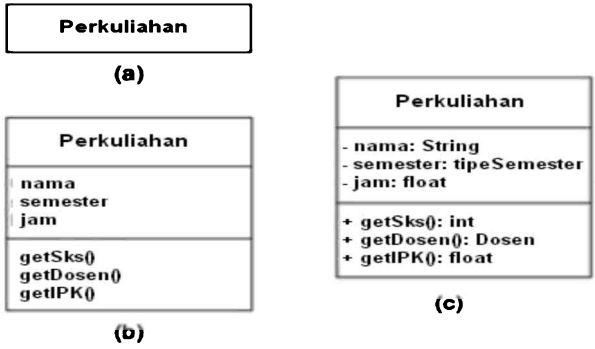
Konsep-konsep yang terkait dengan diagram kelas, antara lain Kelas, Objek, Atribut, Operasi, dan relasi di antaranya akan dideskripsikan berikut ini.

3.1 Kelas dan Objek

Kelas merupakan template bagi sekumpulan objek serupa yang muncul dalam sistem yang akan dibangun. Kelas dapat berupa individu, seperti orang, gedung, organisasi, hewan, mobil, motor, meja, dosen, siswa; atau kejadian seperti kedatangan, pernikahan, perkuliahan, ujian, dan lainnya. Kelas memiliki atribut dan operasi atau metode. Gambar 3.1 menunjukkan perbandingan definisi/ notasi kelas dalam diagram kelas UML dengan kelas dalam Java. Sementara itu, Gambar 3.2 menunjukkan representasi kelas beserta karakteristiknya.



Gambar 3.1 Definisi sebuah kelas dalam UML dan Java



Gambar 3.2 Representasi sebuah kelas dan karakteristiknya

Pada Gambar 3.1, notasi/simbol kelas berupa kotak persegi panjang dengan 3 subkotak. Subkotak teratas digunakan sebagai nama kelas, subkotak di tengah memuat nama-nama atribut, dan subkotak terakhir memuat nama-nama operasi/metode.

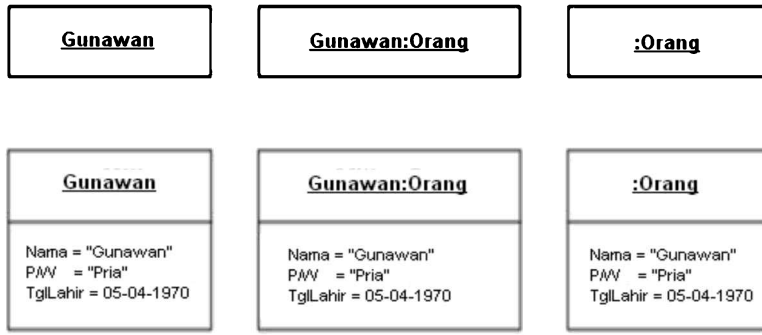
Objek merepresentasikan bentuk konkret sebuah kelas dan disebut sebagai instans kelas. Karakteristik instans kelas (objek) dijelaskan melalui definisi karakteristik struktural (atribut) dan perilaku (operasi) kelas. Dengan adanya karakteristik perilaku kelas/objek, memungkinkan objek-saling berkomunikasi satu sama lain, bertindak dan bereaksi.

Atribut kelas merupakan tempat menyimpan informasi bagi semua objek, tetapi bernilai spesifik dan berbeda bagi setiap instans/objek. Operasi menentukan bagaimana perilaku spesifik dapat dipicu oleh individu objek. Sebagai contoh, kelas mata kuliah (perkuliahan) pada Gambar 3.2 memiliki atribut nama, semester, dan jam. Sementara itu, dua operasi kelas getIPK() dan getDosen() dapat mengembalikan nilai rata-rata (IPK) serta nama dosen untuk mata kuliah (perkuliahan) masing-masing.

Guna memastikan bahwa model tetap jelas dan dapat dimengerti, secara umum tidak dimodelkan semua detail konten, melainkan hanya menyertakan informasi yang relevan bagi sistem yang saat ini dibutuhkan. Hal Ini dimaksudkan agar hasil abstraksi kebutuhan sistem ke dalam model menjadi cukup sederhana, tidak kompleks dan berguna dalam menghindari informasi berlebih yang tidak perlu.

Terkait dengan objek sebagai instans dari kelas, Gambar 3.3 menunjukkan notasi alternatif untuk salah satu Objek Siswa dengan atribut nama bernilai "Gunawan", berjenis kelamin (P/W) "Pria" dan tanggal lahir (TglLahir) "05-04-1970". Sebuah objek memiliki identitas unik dan sejumlah karakteristik yang menjelaskannya secara lebih terperinci. Objek bisa berinteraksi dan berkomunikasi dengan objek lain. Relasi antara objek disebut sebagai *link*. Karakteristik suatu objek meliputi karakteristik struktural (atribut) dan perilakunya (dalam bentuk operasi). Sementara itu, nilai konkret ditetapkan ke atribut dalam diagram objek, perilaku/

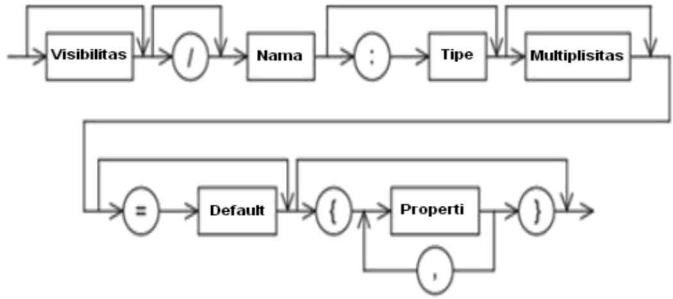
operasi umumnya tidak digambarkan. Operasi objek identik dengan operasi kelas dan karenanya biasanya dijelaskan secara eksklusif dalam kelas.



Gambar 3.3 Notasi alternatif untuk objek

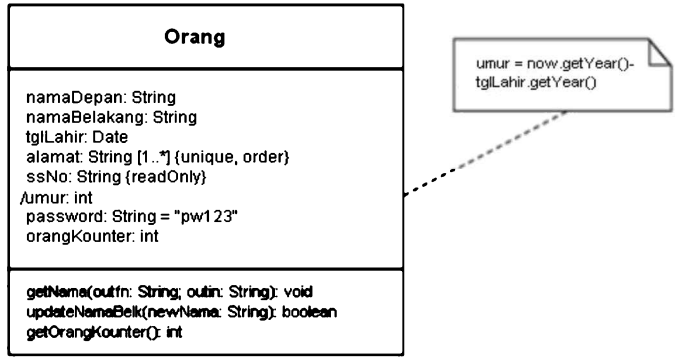
Nilai atribut umumnya dapat berubah dari waktu ke waktu (kecuali atribut kunci yang nilainya harus berbeda antar-objek). Oleh karenanya, diagram objek selalu hanya mewakili snapshot objek pada saat tertentu dan objek dapat berkembang lebih jauh dan berubah seiring berjalannya waktu. Apabila objek tertentu tidak diwakili dalam diagram objek, ini tidak berarti bahwa objek tersebut tidak ada; itu hanya mengungkapkan bahwa objek tersebut tidak penting untuk saat ini.

Penulisan atribut kelas memiliki sintaks. Gambar 3.4 menunjukkan sintaks atribut. Atribut memiliki setidaknya nama. Jenis atau tipe atribut dapat ditentukan setelah nama menggunakan ":". Tipe atribut yang mungkin, termasuk tipe data primitif, seperti integer dan string, tipe data komposit/majemuk, misalnya, tanggal, enumerasi, atau kelas yang didefinisikan pengguna. Dengan menentukan nama: String, misalnya, dapat didefinisikan nama atribut dengan tipe String. Gambar 3.5 menunjukkan contoh lebih lanjut dari tipe atribut.



Gambar 3.4 Sintaks dari spesifikasi atribut

Atribut dapat bernilai *default*, yang berarti bahwa sistem menggunakan nilai default jika nilai atribut tidak diatur secara eksplisit oleh pengguna. Dengan demikian, tidak mungkin bahwa pada suatu waktu, atribut tidak memiliki nilai. Misalnya, kalau dalam sistem, seseorang harus selalu memiliki kata sandi, kata sandi default pw123 ditentukan ketika orang baru dimasukkan dalam sistem (Gambar 3.5). Sandi ini valid/sahih hingga direset.

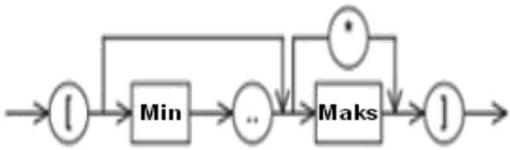


Gambar 3.5 Sifat-sifat (properti) dari atribut

Apabila properti tambahan atribut diperlukan maka cukup dinyatakan dalam kurung kurawal "{}". Misalnya, properti {readOnly} berarti atribut nilai tidak dapat diubah setelah diinisialisasi. Contoh pada Gambar 3.5, nomor jaminan sosial ssNo adalah atribut yang tidak boleh diubah. Properti lebih lanjut akan diperkenalkan di bagian berikutnya dalam deskripsi spesifikasi multiplisitas.

Spesifikasi garis miring "/" sebelum nama atribut menunjukkan bahwa nilai atribut ini berasal dari atribut lain. Contoh atribut yang diturunkan adalah usia seseorang, yang dapat dihitung sejak tanggal lahir. Pada Gambar 3.5, terdapat catatan berisi aturan perhitungan untuk menentukan usia seseorang. Bergantung pada alat pengembangan yang digunakan, catatan tersebut diformulasikan dalam bahasa alami, dalam bahasa pemrograman, atau dalam pseudocode.

Multiplisitas pada Gambar 3.4 memiliki nilai yang diapit oleh kurung siku dalam bentuk [minimum... maksimum], dengan minimum sebagai batas bawah dan maksimum sebagai batas atas interval. Nilai minimum harus lebih kecil dari atau sama dengan nilai maksimum. Apabila tidak ada batas atas untuk interval, ditandai dengan bintang "*". Kelas Orang pada Gambar 3.5 berisi atribut alamat: String [1..*], menunjukkan bahwa seseorang memiliki setidaknya satu dan mungkin beberapa alamat. Jika minimum dan maksimum identik/sama, maka tidak perlu menentukan minimum dan dua titik. Misalnya, [5] berarti bahwa atribut mengadopsi tepat lima nilai. Ekspresi [*] setara dengan [0,*]. Nilai default multiplisitas untuk atribut adalah 1 dan berarti atribut bernilai tunggal. Notasi multiplisitas sebagaimana Gambar 3.6.

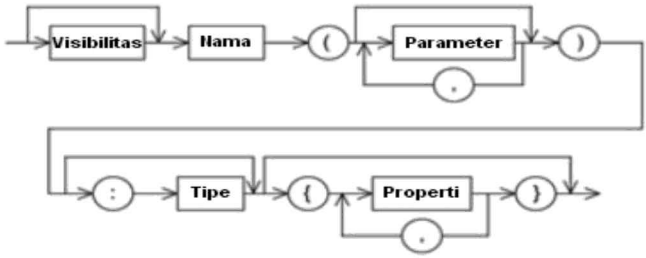


Gambar 3.6 Sintaks untuk multiplisitas

Operasi ditandai dengan nama, parameter, dan jenis nilai pengembaliannya (Gambar 3.7). Ketika operasi dipanggil dalam sebuah program, perilaku yang ditugaskan pada operasi ini dieksekusi. Dalam bahasa pemrograman, operasi sesuai dengan deklarasi metode/fungsi yang didefinisikan, tetapi

tidak diimplementasikan. Diagram kelas tidak cocok dalam menggambarkan perilaku objek secara terperinci karena ia bersifat hanya menspesifikasikan operasi yang ada pada objek dan tidak memodelkan bagaimana operasi diimplementasikan. Diagram perilaku sistem yang menggambarkan implementasi ada pada bab berikutnya.

Dalam diagram kelas, nama operasi diikuti oleh daftar parameter dalam tanda kurung. Daftar itu sendiri bisa kosong. Parameter digambarkan serupa dengan atribut. Satu-satunya informasi yang wajib adalah nama parameter. Penambahan tipe, multiplisitas, nilai default, dan properti lain, seperti terurut, unik, atau yang lain adalah opsional (Gambar 3.7).



Gambar 3.7 Sintaks untuk spesifikasi operasi

Nilai pengembalian operasi secara opsional tergantung pada tipe nilai yang dikembalikan. Pada Gambar 3.5, kelas Orang memiliki operasi updateNamaBelk(newName: String): boolean. Hanya parameter, newName, memiliki tipe String dan menspesifikasikan nama baru untuk seseorang. Nilai pengembalian operasi berjenis boolean. Apabila bernilai **true** dikembalikan maka operasi updateNamaBelk dinyatakan *berhasil* jika tidak maka nilai **false** dikembalikan.

Visibilitas atribut dan operasi menentukan siapa saja yang diizinkan/tidak diizinkan untuk mengaksesnya. Apabila atribut atau operasi tidak memiliki visibilitas yang ditentukan, tidak ada

visibilitas default yang diasumsikan. Tabel 3.1 mencantumkan jenis-jenis visibilitas dan maknanya dalam UML. Hanya objek itu sendiri yang tahu nilai atribut yang ditandai sebagai privat (*private*). Sebaliknya, siapa pun dapat melihat atribut yang ditandai sebagai publik (*public*). Akses ke atribut yang diproteksi disediakan untuk kelas itu sendiri dan subkelasnya. Dengan demikian, visibilitas operasi menentukan siapa yang diizinkan untuk menggunakan fungsionalitas operasi. Contoh diberikan dalam Gambar 3.2(c). Perhatikan bahwa makna dari visibilitas dapat bervariasi dalam bahasa pemrograman dan pemodelan yang berbeda.

Visibilitas digunakan untuk mewujudkan penyembunyian informasi, sebuah konsep yang penting dalam komputasi. Menandai atribut yang mewakili status objek sebagai privat, dapat melindungi status objek terhadap akses yang tidak sah. Oleh karena itu, akses hanya dimungkinkan melalui antarmuka yang didefinisikan dengan jelas, seperti melalui operasi yang dinyatakan secara publik.

Dalam beberapa kasus, diagram kelas hanya berisi atribut dan operasi yang terlihat secara eksternal. Atribut dan operasi kelas yang ditandai sebagai privat sering dihilangkan karena penting untuk realisasi, yaitu implementasi kelas, tetapi tidak untuk penggunaannya. Oleh karena itu, atribut dan operasi yang ditandai sebagai privat ditentukan tergantung pada tujuan serta saat pembuatan diagram kelas.

Tabel 3.1 Visibilitas Beserta Simbol dan Deskripsi

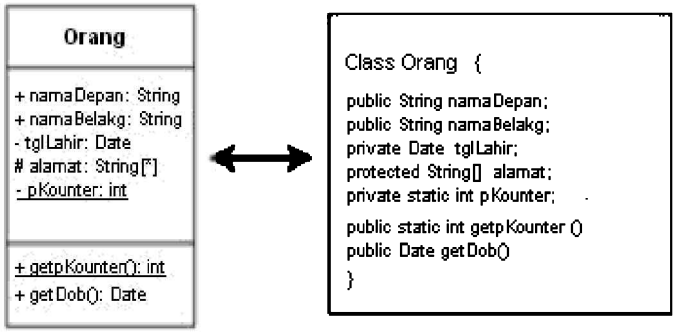
Nama	Simbol	Deskripsi
public	+	Akses objek-objek sebarang Kelas diizinkan
private	-	Akses objek-objek Kelas sendiri diizinkan
protected	#	Akses objek-objek dalam Kelas yang sama dan subkelasnya diizinkan

3.2 Variabel Kelas dan Operasi

Atribut biasanya didefinisikan pada tingkat instans. Apabila kelas diwujudkan dalam bahasa pemrograman maka memori disediakan untuk setiap atribut objek saat dibuat (di-*create*). Atribut tersebut juga disebut sebagai variabel instans atau atribut *instance*. Pada Gambar 3.8, namaBelakg dan tglLahir adalah variabel instans. Dari diagram kelas ini, andaikan orang1 adalah objek kelas Orang maka orang1.namaBelakg dapat digunakan untuk merujuk ke nama belakang orang tersebut. Adapun akses ke tanggal lahir orang ini (atribut tglLahir) tidak dimungkinkan karena visibilitas atribut tglLahir bersifat privat (*private*). Untuk mengetahui tanggal lahir person1, fungsi orang1.getDob() harus dipanggil. Operasi kelas, seperti getDob() hanya dapat dijalankan jika instans terkait yang menawarkan operasi ini dibuat sebelumnya. Dalam kasus ini adalah instans orang1. Operasi dapat menggunakan semua variabel instans yang terlihat.

Berbeda dengan variabel instans, variabel kelas hanya dibuat sekali untuk kelas daripada secara terpisah untuk setiap instans kelas ini. Variabel-variabel ini juga disebut sebagai atribut statis atau atribut kelas. Penghitung (pCounter) untuk jumlah instans kelas (Gambar 3.8) atau konstanta, seperti π sering diwujudkan sebagai atribut statis. Dalam diagram kelas, atribut statis yang digarisbawahi, bertindak seperti operasi statis. Operasi statis, juga disebut operasi kelas, dapat digunakan jika tidak ada instans kelas yang sesuai yang dibuat. Contoh operasi statis adalah fungsi matematika seperti sin(x) atau konstruktor. Konstruktor adalah fungsi khusus yang dipanggil untuk membuat instans baru kelas. Metode pemanggilan Orang.getpKounter() menggunakan operasi statis getpKounter() yang didefinisikan dalam Gambar 3.8; operasi dipanggil langsung melalui kelas dan bukan melalui instans. Kecuali dinyatakan sebaliknya, atribut dan operasi menunjukkan

atribut instans dan operasi instans dalam sebagian besar bahasa pemrograman berorientasi objek.



Gambar 3.8 Translasi kelas dari UML ke Java

3.3 Asosiasi

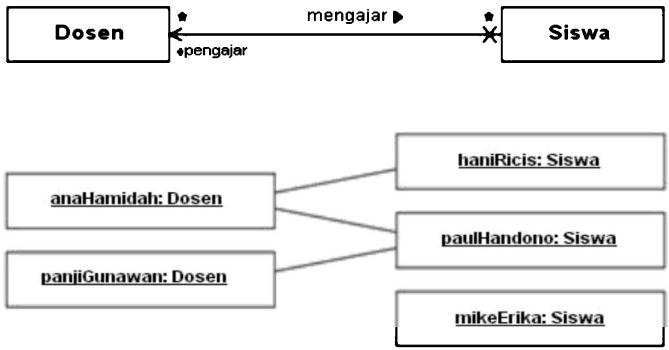
Asosiasi memodelkan hubungan atau relasi antarkelas. Asosiasi menggambarkan kelas mana yang merupakan mitra komunikasi yang potensial. Apabila atribut dan operasi mereka memiliki visibilitas yang sesuai, mitra komunikasi dapat mengakses atribut dan operasi masing-masing. Gambar 3.9 menggambarkan diagram kelas dan diagram objek yang valid. Diagram kelas menunjukkan bahwa kelas Dosen dan Siswa berelasi melalui asosiasi mengajar. Dalam peran sebagai pengajar, seorang Dosen memiliki Siswa sebanyak nol atau lebih dan satu siswa memiliki Dosen sebanyak nol atau lebih dalam perannya sebagai pengajar. Diagram objek membuat model skenario menjadi konkret/nyata.

3.3.1 Asosiasi Biner

Asosiasi biner memungkinkan terjadinya relasi antardua kelas satu sama lain. Relasi ditampilkan sebagai garis solid (disebut garis saja) antara kelas mitra yang terlibat. Garis dapat dilabeli dengan nama asosiasi secara opsional diikuti oleh arah bacaan (segitiga hitam kecil). Arah pembacaan ke salah satu ujung

asosiasi dan hanya menunjukkan ke arah mana pembaca diagram harus "membaca" nama asosiasi. Dalam diagram Gambar 3.9, arah membaca menunjukkan bahwa Dosen "mengajar" siswa dan bukan sebaliknya.

Apabila garis diarahkan, setidaknya salah satu dari dua ujung memiliki kepala panah terbuka, navigasi dari objek ke objek mitranya dimungkinkan. Navigasi menunjukkan bahwa objek mengetahui objek mitranya dan karenanya dapat mengakses atribut serta operasi yang terlihat. Navigasi tidak ada hubungannya dengan arah bacaan, seperti contoh Gambar 3.9 menunjukkan bahwa arah membaca menunjukkan bahwa Dosen memberikan kuliah (atau mengajar) siswa. Namun, navigasi yang ditentukan menunjukkan bahwa siswa dapat mengakses karakteristik yang terlihat dari Dosen pada perkuliahan yang mereka hadiri. Sebaliknya, seorang Dosen tidak dapat mengakses karakteristik yang terlihat dari siswa yang menghadiri perkuliahan karena Dosen tidak mengenal mereka.

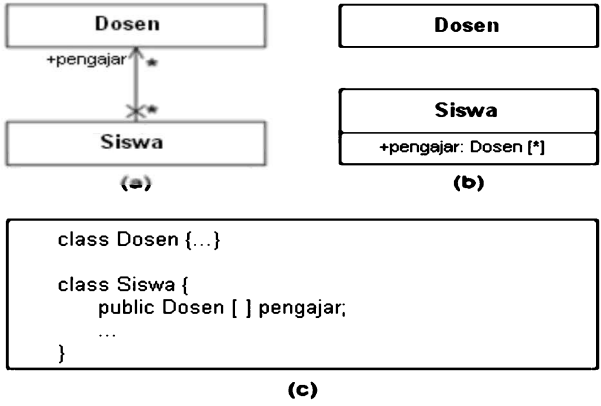


Gambar 3.9 Contoh asosiasi biner antarkelas dan diagram objek

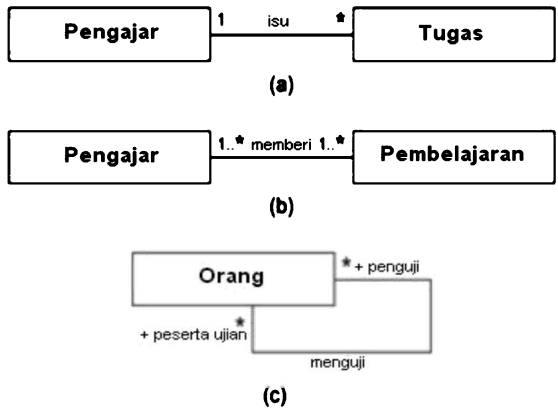
Asosiasi yang tidak dapat dinavigasi ditunjukkan melalui tanda "X" di ujung asosiasi yang bersangkutan. Misalnya, kalau tanda X seperti itu muncul di akhir asosiasi A untuk relasi antara kelas A dan

B, ini berarti bahwa B tidak dapat mengakses atribut dan operasi A —bahkan yang publik. Garis dua arah tanpa kepala panah atau X di ujungnya tidak memberikan informasi apa pun tentang arah navigasi, tetapi dalam praktiknya, biasanya diasumsikan navigasi dwiarah. Arah navigasi mewakili petunjuk untuk implementasi berikutnya karena dalam bahasa pemrograman berorientasi objek, asosiasi diwujudkan sebagai referensi ke objek terkait.

Gambar 3.10 memperlihatkan (a) diagram kelas, di mana relasi Siswa-Dosen dimodelkan secara eksplisit sebagai asosiasi, dan bentuk lain yang setara; (b) diagram kelas, di mana relasi diwakili oleh atribut di kelas Siswa; dan (c) terjemahan ke dalam Java. Diagram kelas pada Gambar 3.10(a) lebih disukai karena di sini hubungan antara kelas divisualisasikan secara eksplisit dan terlihat/tampak, sementara Gambar 3.10(b), sebagai alternatif relasi antara Siswa serta Dosen hanya dapat dikenali dengan membaca informasi jenis atribut Dosen.



Gambar 3.10 Asosiasi dalam UML dan Java



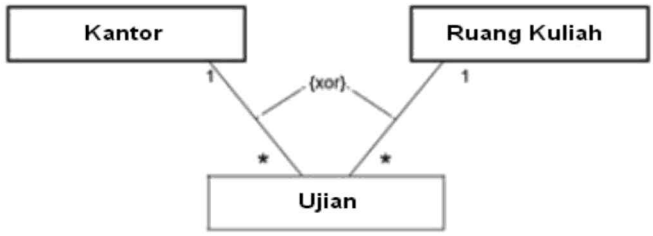
Gambar 3.11 Contoh spesifikasi multiplisitas dalam asosiasi Biner

Dengan cara yang sama bahwa multiplisitas atribut dan parameter ditentukan, multiplisitas asosiasi memiliki nilai dalam selang/interval dalam bentuk minimum.. maksimum. Apabila minimum dan maksimum identik, satu nilai serta titik dapat dihilangkan (cukup nilai saja). Untuk multiplisitas 0..∗ berarti sama/identik dengan ∗. Gambar 3.11 menunjukkan contoh spesifikasi multiplisitas untuk asosiasi biner. Gambar 3.11(a) merepresentasikan bahwa pengajar (dosen) dapat mengeluarkan satu atau beberapa tugas dan bahwa 1 tugas dikeluarkan oleh satu pengajar (dosen) yang sesuai. Tidak ada penugasan yang mungkin ada tanpa asosiasi kepada pengajar/dosen. Gambar 3.11(b) merepresentasikan bahwa seorang pengajar (Dosen) memberikan setidaknya satu pembelajaran (ceramah) dan pembelajaran (ceramah) diberikan oleh setidaknya satu pengajar (Dosen).

Gambar 3.11(c) merepresentasikan bahwa seseorang dalam peran sebagai penguji dapat menguji beberapa orang (≥ 0) dan seseorang dalam peran sebagai yang diuji dapat diuji oleh beberapa penguji. Contoh dalam Gambar 3.11(c), model tidak mengecualikan kasus bahwa orang dapat menguji diri mereka sendiri. Apabila hal ini tidak diizinkan, batasan tambahan perlu ditentukan.

Label pada asosiasi dapat berakhir dengan nama peran. Sebuah peran menggambarkan cara, di mana suatu objek terlibat dalam relasi asosiasi, yaitu peran apa yang dimainkannya dalam relasi tersebut. Asosiasi pada Gambar 3.11(c), kelas Orang berperan sebagai penguji atau yang diuji.

Untuk mengekspresikan bahwa objek kelas A akan dikaitkan dengan objek kelas B atau objek kelas C, tetapi tidak dengan keduanya, dapat ditentukan melalui batasan xor (atau yang bersifat eksklusif). Contoh pada Gambar 3.12 merepresentasikan dua asosiasi dari kelas yang saling eksklusif (dihubungkan oleh garis putus-putus berlabel {xor}) dengan kelas Ujian. Kondisi tersebut menyatakan bahwa Ujian dapat berlangsung baik di Kantor atau di Ruang Kuliah tetapi tidak di keduanya.

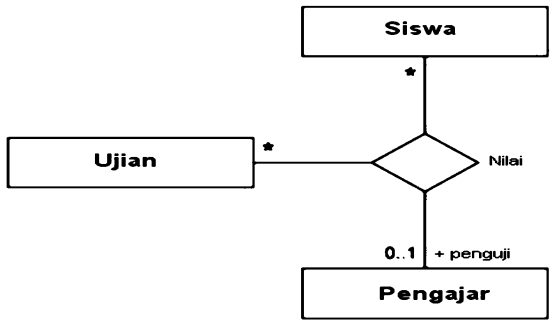


Gambar 3.12 Contoh asosiasi dengan batasan xor

3.3.2 Asosiasi N-Ary

Dalam beberapa hal (lebih dari dua) objek mitra terlibat dalam relasi, asosiasi n-ary dapat digunakan. Asosiasi n-ary direpresentasikan dengan berlian berongga di tengah. Berlian terhubung dengan semua mitra hubungan melalui garis. Nama asosiasi ditentukan di samping berlian. Tidak ada arah navigasi untuk asosiasi n-ary; tetapi, multiplisitas dan nama peran dapat dituliskan. Multiplisitas mendefinisikan berapa banyak objek sebuah kelas yang dapat ditetapkan pada (n – 1) pasangan objek-objek dari kelas/peran lainnya.

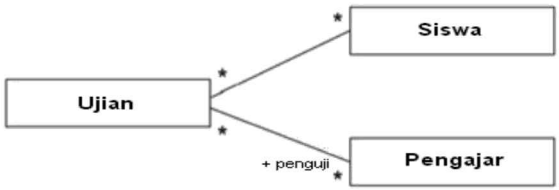
Gambar 3.13 menunjukkan model asosiasi n-ary untuk kelas Pengajar, Siswa, dan Ujian. Multiplisitas didefinisikan sebagai berikut: satu siswa tertentu mengambil satu ujian tertentu tanpa Pengajar (yaitu, tidak mengikuti ujian ini sama sekali) atau dengan tepat satu Pengajar, ditunjukkan dengan 0..1 pada kelas Pengajar. Satu ujian khusus dengan satu Pengajar tertentu dapat diambil oleh sejumlah Siswa dan satu Siswa tertentu dapat dinilai oleh satu Pengajar tertentu untuk sejumlah Ujian. Dalam model ini, tidak mungkin terdapat dua atau lebih Pengajar dengan satu Siswa untuk Ujian yang sama.



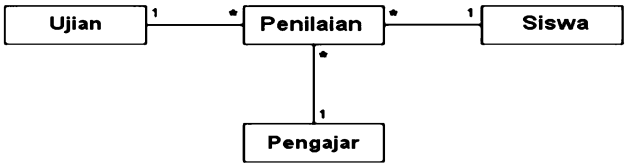
Gambar 3.13 Contoh asosiasi n-ary

Dalam representasi model pada Gambar 3.14, ujian dapat dinilai oleh beberapa pengajar. Asosiasi ternary di Gambar 3.13 dengan jelas menunjukkan pengajar mana yang meluluskan ujian tertentu Siswa — yang ini tidak terjadi dengan diagram yang ditunjukkan pada Gambar 3.14.

Sebagai alternatif untuk asosiasi ternary di Gambar 3.13, kelas tambahan (penilaian) dapat diperkenalkan dengan menghubungkan ke kelas asli melalui asosiasi biner (Gambar 3.15). Dengan model ini, ada kemungkinan bahwa seorang siswa dinilai beberapa kali untuk satu atau lebih dari ujian yang sama, sebagaimana hal yang tidak mungkin terjadi dengan model Gambar 3.13.



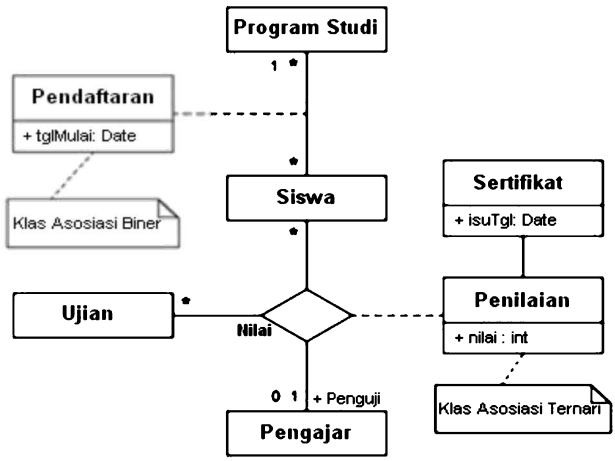
Gambar 3.14 ...versus contoh dengan dua asosiasi biner



Gambar 3.15 ...versus contoh dengan kelas tambahan

3.4 Asosiasi Kelas

Apabila diinginkan atribut atau operasi ke berhubungan dengan satu atau beberapa kelas daripada ke kelas itu sendiri, kita dapat melakukannya dengan menggunakan kelas asosiasi. Kelas asosiasi direpresentasikan oleh kelas dan asosiasi yang terhubung melalui garis putus-putus. Asosiasi dapat biner atau n-ary. Meskipun representasi mencakup beberapa komponen, kelas asosiasi adalah satu konstruksi bahasa yang memiliki properti kelas dan properti asosiasi. Oleh karena itu, dalam diagram, kelas, dan asosiasi dari kelas asosiasi harus memiliki nama yang sama, meskipun keduanya tidak perlu disebutkan (kelas asosiasi pendaftaran dan penilaian pada Gambar 3.16). Kelas asosiasi juga dapat memiliki asosiasi dengan kelas lain. Pada Gambar 3.16, kelas asosiasi, yang berisi informasi tentang nilai siswa untuk ujian tertentu, dikaitkan dengan kelas Sertifikat.

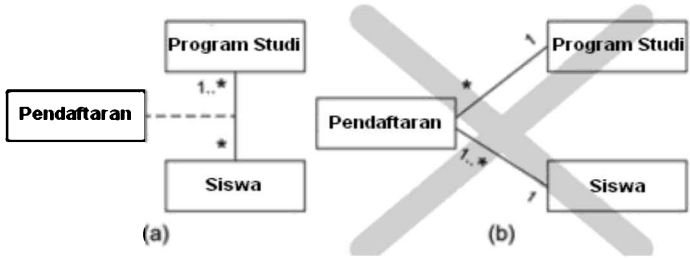


Gambar 3.16 Contoh kelas asosiasi

Secara umum, kelas asosiasi tidak dapat digantikan dengan kelas normal, sebagaimana Gambar 3.17(a) dan (b). Diasumsikan bahwa seorang siswa mendaftar untuk setidaknya satu program studi dan memiliki tepat satu pendaftaran untuk setiap program studi yang dipilih. Pada gilirannya, sejumlah (≥ 0) siswa dapat mendaftar untuk satu program studi tertentu sebagaimana Gambar 3.17(a).

Gambar 3.17(b) menunjukkan upaya untuk mencontoh situasi ini hanya dengan kelas "normal". Pendaftaran ditugaskan untuk tepat satu siswa dan tepatnya satu program studi, sementara satu program studi terkait dengan sejumlah objek pendaftaran. Seorang siswa memiliki setidaknya satu pendaftaran.

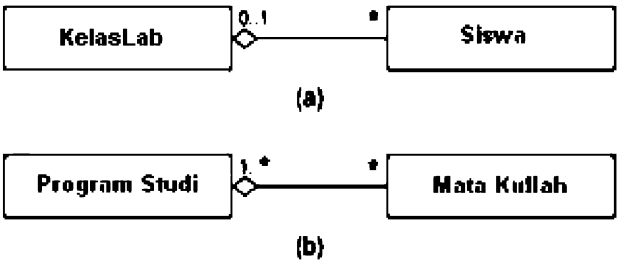
Namun, kalau kita memeriksa diagram lebih dekat, kita melihat bahwa dalam Gambar 3.17 (b), seorang siswa dapat memiliki beberapa pendaftaran untuk satu dan program studi yang sama, yang bukan tujuannya. Sebaliknya, pada Gambar 3.17(a), seorang siswa dapat mendaftar untuk program studi tertentu hanya sekali.



Gambar 3.17 Perbandingan kelas asosiasi dengan kelas normal

3.5 Agregasi

Agregasi adalah bentuk khusus asosiasi yang digunakan untuk mengekspresikan bahwa instans/objek dari satu kelas adalah bagian dari instans/objek kelas lain. Terdapat 2 tipe agregasi, yaitu agregasi berbagi dan agregasi komposisi. Keduanya disimbolkan dengan berlian di ujung asosiasi kelas, dengan sedikit perbedaan. Agregasi komposisi disimbolkan dengan berlian padat, dan agregasi bersama disimbolkan dengan berlian berongga. Keduanya bersifat transitif dan asosiasi asimetris. Dalam hal ini, transitivitas berarti bahwa jika B adalah bagian dari A dan C adalah bagian dari B, C juga merupakan bagian dari A. Asimetri menyatakan bahwa tidak mungkin A menjadi bagian dari B dan B untuk menjadi bagian dari A secara bersamaan.



Gambar 3.18 Contoh agregasi berbagi (*shared*)

3.5.1 Agregasi Berbagi

Agregasi berbagi memiliki ikatan yang lemah atas bagian-bagian (atau elemen-elemen) terhadap keseluruhan, yang berarti bahwa elemen-elemen tersebut ada secara independen dari keseluruhan. Multiplisitas di ujung agregasi bisa lebih besar dari 1, yang berarti bahwa sebuah elemen dapat menjadi bagian dari beberapa elemen lain secara bersamaan. Gambar 3.18 memperlihatkan dua contoh penggunaan agregasi berbagi. Pada Gambar 3.18(a), KelasLab terdiri dari sejumlah siswa. Namun, seorang siswa dapat berpartisipasi dalam maksimal satu KelasLab. Pada Gambar 3.18(b), program studi terdiri dari sejumlah (≥ 0) mata kuliah. Sebuah mata kuliah ditugaskan untuk setidaknya satu (≥ 1) program studi.

3.5.2 Agregasi Komposisi

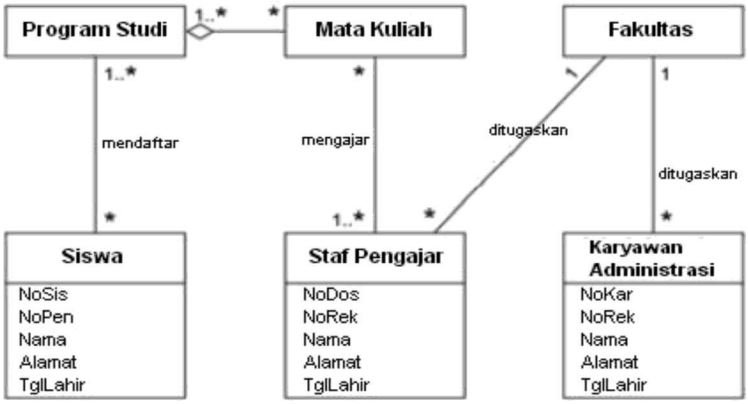
Penggunaan agregasi komposisi menyatakan bahwa bagian/elemen tertentu hanya dapat terkandung/termuat dalam (paling banyak) satu objek komposit pada satu titik waktu tertentu. Hal itu menghasilkan multiplisitas maksimum 1 di ujung agregasi (objek komposit). Oleh karenanya, objek komposit membentuk hutan pohon yang menunjukkan adanya ketergantungan antara objek komposit dan bagian-bagiannya. Apabila objek komposit dihapus, bagian-bagiannya juga terhapus. Gambar 3.19 menunjukkan contoh agregasi komposisi. Ruang kuliah adalah bagian dari sebuah bangunan. Karena multiplisitasnya 1 maka ada ketergantungan eksistensi antara elemen kedua kelas ini. Ruang kuliah tidak mungkin ada tanpa Gedung. Situasinya akan berbeda untuk proyektor jika dikaitkan dengan ruang kuliah dalam relasi agregasi komposisi. Dengan multiplisitas 0..1 pada akhir agregasi (Ruang Kuliah) maka berarti proyektor bisa tetap ada tanpa ruang kuliah.

Agregasi berbagi berbeda dari asosiasi hanya dengan melalui fakta bahwa relasi tersebut secara eksplisit memvisualisasikan hubungan/relasi "bagian dari". Dalam agregasi komposisi, ketergantungan eksistensi menandakan ikatan yang jauh lebih kuat antara eksistensi objek komposit dan bagian-bagiannya, yang berarti bahwa komposisi serta objek komposit asosiasi tidak dapat dipertukarkan. Agregasi komposisi biasanya digunakan jika bagian-bagiannya secara fisik tertanam dalam objek komposit atau hanya terlihat untuk objek komposit. Apabila objek komposit dihapus atau disalin, bagian-bagiannya juga dihapus atau disalin ketika komposisi digunakan.

Gambar 3.19 Contoh agregasi komposisi (komposisi)

3.6 Generalisasi

Generalisasi digunakan untuk menyoroti kemiripan/ keserupaan antarkelas sehingga karakteristik umum antarkelas tidak perlu disebutkan secara berulang. Dengan demikian, generalisasi dapat dipakai untuk mendapatkan kelas yang lebih spesifik dari kelas yang ada. Apabila diinginkan kelas Dosen ditambah, sebagai subkelas dari kelas staf Pengajar (Gambar 3.20) maka relasi generalisasi digunakan untuk menghindari kopi/salin karakteristik yang sama dari kelas asosiasi Pengajar ke kelas Dosen.



Gambar 3.20 Diagram kelas tanpa generalisasi

Inheritance/Pewarisan

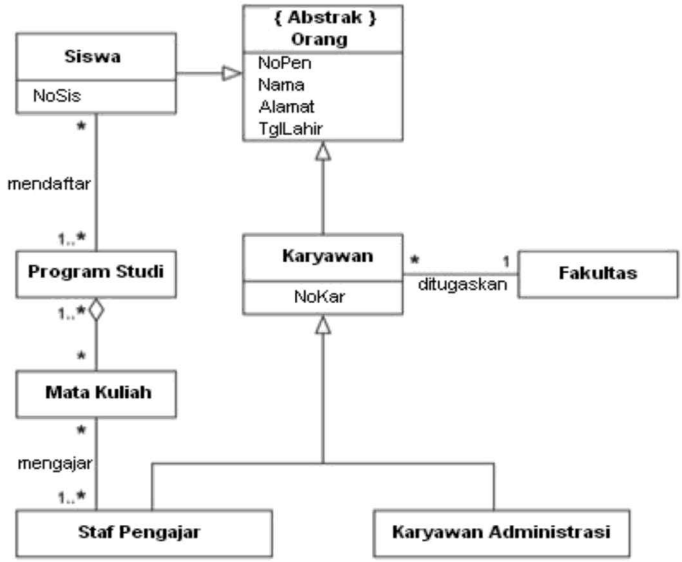
Relasi generalisasi menyatakan bahwa karakteristik (atribut dan operasi) dan asosiasi yang ditentukan pada kelas umum (superkelas) dapat diwariskan ke subkelasnya. Oleh karenanya, relasi generalisasi juga disebut sebagai pewarisan. Ini berarti bahwa setiap instans/objek subkelas secara bersamaan merupakan instans/objek tidak langsung dari superkelas. Subkelas "memiliki" semua atribut instans/objek dan atribut kelas serta semua operasi instans/objek dan operasi superkelas, asalkan belum ada tanda terkait dengan visibilitas privat. Subkelas juga dapat memiliki atribut dan operasi lebih lanjut atau masuk ke dalam hubungan lain secara independen dari superkelasnya. Dengan demikian, operasi yang berasal dari subkelas atau superclass dapat dieksekusi langsung pada instans/objek subkelas.

Relasi generalisasi ditandai dengan panah segitiga berongga dari subkelas ke superkelas, misalnya, dari Siswa ke Orang (Gambar 3.21). Nama superkelas harus dipilih sedemikian rupa sehingga merepresentasikan konsep umum/payung untuk nama-nama subkelasnya. Untuk memastikan bahwa tidak ada instans/objek langsung dari kelas Orang, kelas ini dilabeli dengan kata kunci

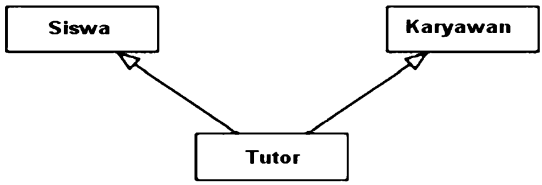
{abstrak}. Untuk itu, kelas Orang menjadi kelas abstrak dan hanya subkelas nonabstrak yang dapat diinstansiasi.

Relasi generalisasi juga disebut sebagai relasi "adalah". Misalnya, setiap siswa adalah seseorang (Gambar 3.21). Setiap staf Pengajar dan setiap Karyawan administrasi adalah karyawan dan karena transitivitas relasi generalisasi, setiap karyawan administrasi juga merupakan seseorang. Dalam bahasa pemrograman berorientasi objek jika kelas dianggap sebagai tipe maka subkelas dan superkelas setara dengan subtipe serta supertipe.

Dalam UML sebuah kelas bisa memiliki beberapa superkelas. Sebagai contoh, kelas tutor memiliki warisan dari kelas Siswa dan Karyawan (Gambar 3.22), kondisi tersebut disebut pewarisan berganda (*multiple inheritance*).



Gambar 3.21 Diagram kelas dengan generalisasi



Gambar 3.22 Contoh pewarisan berganda

3.7 Kelas Abstrak vs. Antarmuka

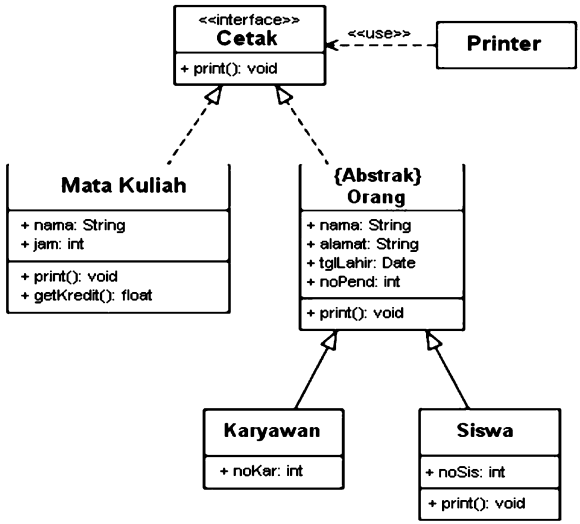
Kelas yang tidak dapat diinstansiasi sendiri dimodelkan sebagai kelas abstrak. Kelas abstrak tidak memiliki objek — hanya subkelas mereka yang dapat diinstansiasi. Kelas abstrak digunakan secara eksklusif untuk menyoroti karakteristik umum dari subkelas mereka dan karenanya hanya berguna dalam konteks relasi generalisasi. Operasi kelas abstrak juga dapat dilabeli sebagai abstrak. Operasi kelas abstrak tidak memiliki implementasi apa pun, tetapi implementasinya dapat dilakukan pada semua subkelas konkretnya.

Kelas abstrak dan operasi abstrak ditulis dengan huruf *italic* atau ditunjukkan dengan spesifikasi kata kunci {abstrak} sebelum nama kelasnya (Gambar 3.23). Penggunaan alternatif notasi kedua direkomendasikan.

Gambar 3.23 Notasi untuk kelas abstrak

Contoh pada Gambar 3.24, kelas Orang sebagai kelas abstrak. Tidak boleh ada instans dari kelas Orang, tetapi instans (objek) subkelas tertentu, yaitu Karyawan dan Siswa bisa memiliki instans (objek).

Sebagaimana kelas abstrak, kelas antarmuka (*interface*) juga tidak memiliki implementasi atau instans langsung apa pun. Kelas antarmuka mewakili kontrak. Kelas yang masuk ke dalam kontrak ini, yaitu kelas yang mengimplementasikan antarmuka, mewajibkan diri untuk menyediakan perilaku yang ditentukan oleh antarmuka. Berbeda dengan relasi antara kelas abstrak dan subkelasnya, relasi "adalah" antara antarmuka serta kelas yang mengimplementasikannya tidak diperlukan. Operasi antarmuka tidak pernah memiliki implementasi.



Gambar 3.24 Contoh sebuah antarmuka

Kelas antarmuka ditandai seperti kelas, tetapi dengan kata kunci tambahan «interface» sebelum nama kelasnya. Panah pewarisan putus-putus dengan panah segitiga berongga dari kelas ke antarmuka menandakan bahwa kelas ini mengimplementasikan antarmuka. Panah putus-putus dengan kepala terbuka dengan kata kunci «use» menyatakan bahwa kelas menggunakan antarmuka.

Contoh pada Gambar 3.24, kelas Orang dan Mata Kuliah mengimplementasikan antarmuka yang dapat dicetak. Kelas yang mengimplementasikan cetak harus menyediakan operasi print(). Operasi print ini bisa berbeda untuk setiap kelas. Untuk kelas Mata Kuliah, nama dan jumlah jam dicetak; untuk kelas Orang, nama dan alamat dicetak. Di kelas Siswa, operasi print() ditentukan lagi. Ini menunjukkan bahwa Siswa memperluas perilaku operasi print() yang diwarisi dari kelas Orang.

Metode print() ditimpa, yang berarti bahwa nomor noSis juga dicetak. Untuk Karyawan ini tidak perlu, dengan asumsi bahwa perilaku yang ditentukan untuk print() secara langsung sudah cukup. Kelas Printer sekarang dapat memproses setiap kelas yang mengimplementasikan kelas antarmuka Cetak. Dengan demikian, print() tertentu dapat diwujudkan untuk setiap kelas dan kelas Printer tetap tidak berubah.

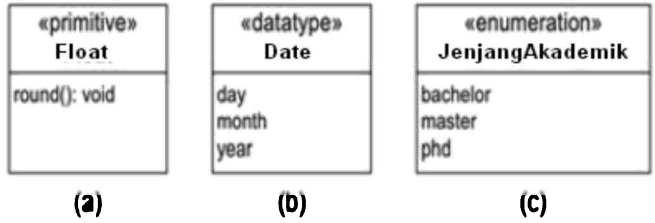
3.8 Tipe Data

Atribut, parameter, dan nilai pengembalian operasi memiliki tipe yang menentukan bentuk konkret mana yang mungkin mereka ambil. Misalnya, nama seseorang memiliki tipe String. Tipe dapat berupa kelas atau tipe data. Tipe data divisualisasikan dengan cara yang sama, seperti kelas, dengan perbedaan bahwa nama tipe data dinotasikan dengan kata kunci tambahan «data type» (Gambar 3.25(b)). Contoh pada Gambar 3.25(b) menunjukkan tipe data dapat memiliki struktur internal dalam bentuk atribut. Ada juga dua bentuk khusus tipe data, yaitu tipe data *primitive* dan *enumeration*.

Tipe data primitif (*primitive*) tidak memiliki struktur internal. Terdapat empat tipe data primitif yang telah ditentukan sebelumnya: boolean, integer, unlimitedNatural, dan String.

Tipe data primitif yang ditentukan pengguna diidentifikasi oleh spesifikasi kata kunci «primitive». Tipe data primitif bisa memiliki operasi yang dieksekusi pada nilainya (Gambar 3.25(a)).

Enumerasi (*enumeration*) adalah tipe data yang nilainya ditentukan dalam daftar. Notasinya sama dengan kelas dengan identifikasi spesifik «enumeration». Pada Gambar 3.25(c), enumerasi JenjangAkademik didefinisikan. Enumerasi ini mencantumkan semua gelar akademik yang dikenal dalam sistem. Oleh karena itu, atribut tipe JenjangAkademik yaitu bachelor, master, dan phd. Nilai-nilai ini disebut literal.



Gambar 3.25 Contoh tipe data

Tipe data yang ditentukan pengguna digunakan sebagaimana yang dispesifikasikan dalam deskripsi sintaks atribut dan operasi pada Gambar 3.5 dan pada Gambar 3.8. Perhatikan tipe data pada Gambar 3.25, dapat digunakan dalam definisi atribut berikut, yaitu berat: Float; tglLahir: Date, dan gelar: JenjangAkademik.

3.9 Prosedur Membangun Diagram Kelas

Berikut ini disampaikan panduan/prosedur dalam membangun diagram kelas atau mengembangkan model konseptual dari bahasa alami (bisa berasal dari statemen problem/spesifikasi kebutuhan sistem atau use case).

1. Kenali (*identify*) kandidat kelas dengan memilih kata benda atau ungkapan kata benda.
2. Keluarkan atau hapus kandidat-kandidat kelas yang tidak relevan, seperti adanya:
    a. Nama-nama kandidat kelas yang rangkap/redundant atau sama/serupa (dengan menetapkan salah satu).
    b. Nama kelas yang masih samar atau masih diragukan (*vague*).
    c. Nama kelas yang berupa kejadian (*event*) atau operasi kelas, terkecuali jika kandidat kelas tersebut memiliki status, identitas, atribut, dan perilaku.
3. Identifikasi relasi antarkelas (asosiasi/agregasi/ *inheritance*), dengan mengacu adanya "kata kerja" atau aktivitas antarkelas tersebut.
4. Tetapkan multiplisitas (kardinalitas) di antara relasi antarkelas tersebut (asosiasi/agregasi).
5. Identifikasi atribut dan operasi/metode kelas yang penting untuk setiap kelas.
6. Integrasikan diagram kelas yang telah diperoleh dari setiap use case dan/atau statemen problem.
7. Verifikasi hasilnya sesuai dengan domain problem atau kebutuhan sistem.

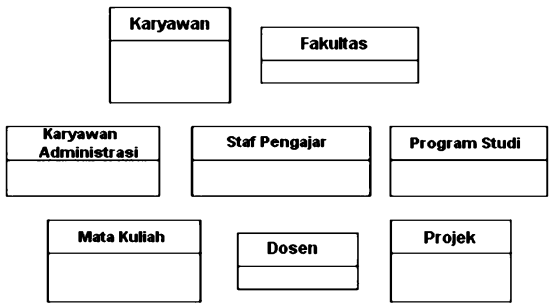
3.10 Contoh Terapan

1. Sistem Informasi Sebuah Universitas

Contoh berikut merupakan perolehan diagram kelas dari spesifikasi tekstual kebutuhan sistem informasi sebuah universitas, yaitu (1) Sebuah universitas terdiri dari beberapa fakultas, dan masing-masing fakultas memiliki berbagai program studi. Setiap fakultas dan program studi memiliki nama. Alamat dikenal untuk setiap program studi; (2) Setiap fakultas dipimpin oleh seorang dekan, yang merupakan karyawan universitas; (3) Jumlah total karyawan diketahui. Karyawan memiliki nomor karyawan, nama, dan alamat e-mail. Ada perbedaan antara asosiasi (staf/tenaga) pengajar dan tenaga administrasi; (4) Staf pengajar ditugaskan ke setidaknya satu program studi. Selain itu, staf pengajar memiliki bidang studi dan dapat terlibat dalam proyek selama beberapa jam tertentu, dengan nama, tanggal mulai, dan tanggal akhir proyek diketahui. Beberapa staf pengajar mengajar mata kuliah. Mereka disebut Dosen; dan (5) Mata kuliah memiliki kode (ID) yang unik, nama, dan durasi mingguan dalam jam.

a. Identifikasi Kelas

Pertama, diidentifikasi unsur-unsur kata benda yang ada dalam sistem universitas yang berpotensi sebagai kelas-kelas. Hasilnya ditunjukkan pada Gambar 3.26.



Gambar 3.26 Kandidat kelas

b. Identifikasi Atribut

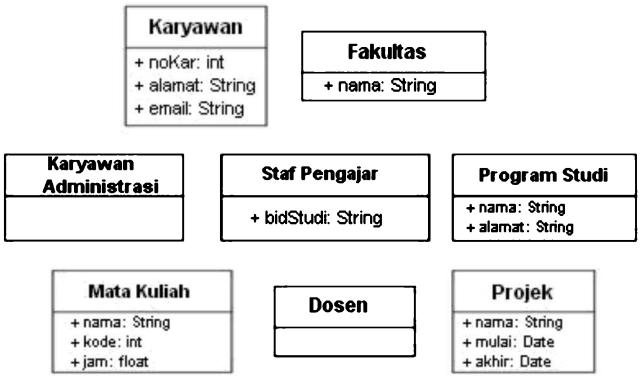
Langkah berikut setelah kelas diperoleh, atribut dari tiap kelas diidentifikasi dan hasilnya ada pada Gambar 3.27. Tipe data yang bermakna/penting untuk tiap atribut juga diidentifikasi, meskipun tidak termasuk dalam spesifikasi kebutuhan. Dalam fase ini, untuk sementara visibilitas semua atribut diset ke publik, tanpa perlu memikirkan atribut mana yang terlihat dari luar dan mana yang tidak.

c. Identifikasi Relasi Antarkelas

Terdapat tiga macam relasi antarkelas, yaitu relasi asosiasi, agregasi, dan generalisasi.

1) Generalisasi

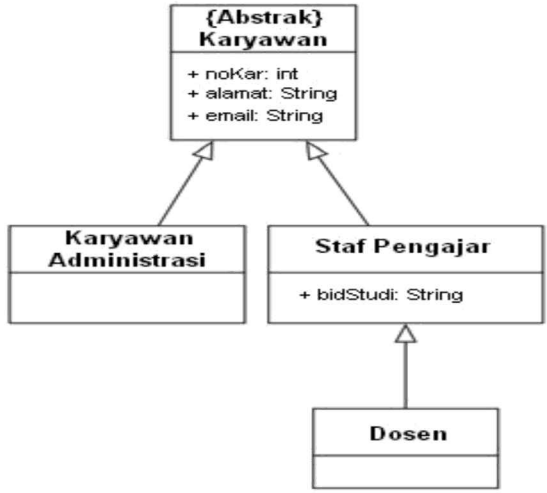
Relasi generalisasi atau pewarisan (*inheritance*) ditunjukkan sebagaimana Gambar 3.28. Pada Gambar tampak ada perbedaan antara Staf Pengajar dan Karyawan Administrasi, dan beberapa staf pengajar sebagai Dosen. Gambar 3.28 juga menunjukkan bahwa setiap karyawan universitas ada yang berperan sebagai Staf Pengajar dan ada yang sebagai Karyawan Administrasi. Kelas Karyawan dalam hal ini berperan sebagai kelas abstrak.



Gambar 3.27 Kelas dan atributnya

2) Relasi asosiasi, agregasi, dan multiplisitasnya

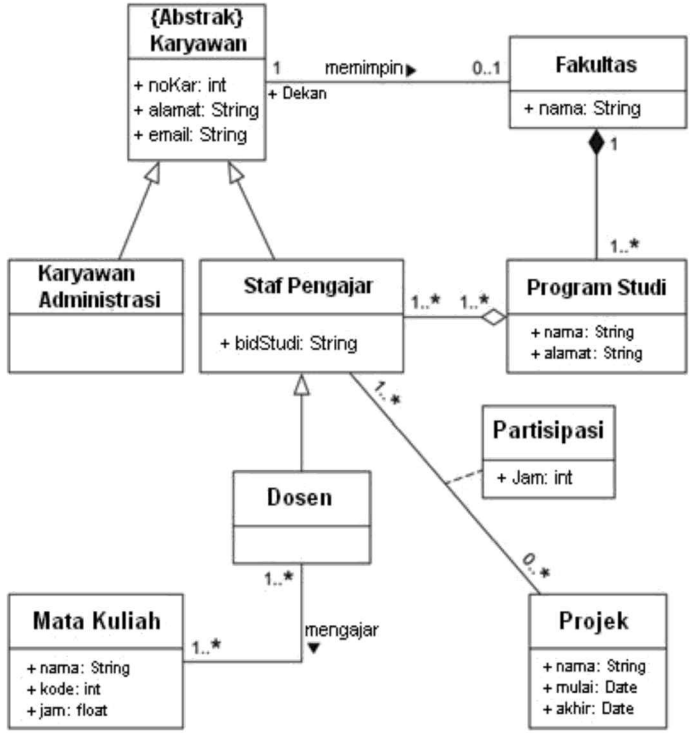
Dalam diagram kelas, terdapat dua macam relasi yang lain, yaitu asosiasi dan agregasi. Pembahasan di sini tentang relasi asosiasi dan agregasi akan dikaitkan juga dengan multiplisitas yang sesuai. Sebagaimana Gambar 3.29, kelas-kelas Dosen dan Mata Kuliah dikaitkan dengan relasi asosiasi mengajar. Seorang karyawan memimpin Fakultas. Di sini karyawan mengambil peran sebagai Dekan. Fakultas terdiri dari beberapa Program Studi, yang dimodelkan sebagai agregasi komposisi.



Gambar 3.28 Contoh relasi generalisasi

Staf Pengajar ditugaskan ke sebuah Program Studi, yang berarti mereka adalah bagian dari Program Studi. Namun, penggunaan relasi agregasi komposisi di sini kurang tepat karena tidak ada ketergantungan keberadaan antara instans Karyawan dan Program Studi. Sebagai pengganti, agregasi berbagi dimungkinkan untuk mewakili relasi bagian-seluruh secara eksplisit. Berikutnya, terdapat relasi asosiasi antara Staf Pengajar dengan Projek. Kelas asosiasi ini memerinci lebih lanjut hubungan antara Projek dan Staf Pengajar terkait dengan jumlah jam. Gambar 3.29 memperlihatkan diagram kelas lengkap untuk tugas yang diberikan.

Perhatikan bahwa model yang dihasilkan tidaklah unik bahkan untuk contoh kecil/sederhana seperti ini. Hal itu bisa tergantung pada sisi aplikasi yang dimaksudkan dan di sisi lain pada gaya pemodel. Sebagai contoh, kalau diinginkan membuat model dengan maksud menghasilkan kode (*source code*) darinya maka dimungkinkan akan dirancang antarmuka dengan lebih hati-hati dan penentuan visi yang bisa berbeda. Hal tersebut tinggal masalah rasa bahwa Dosen adalah kelas yang terpisah, tetapi dekan adalah peran. Di samping itu, Dosen bisa ditetapkan sebagai peran di akhir asosiasi mengajar yang akan didefinisikan antara kelas Staf Pengajar dan Mata Kuliah.



Gambar 3.29 Diagram kelas dari sistem informasi universitas

2. Sistem Informasi Sebuah Rumah Sakit

Contoh berikut merupakan perolehan diagram kelas dari sebuah use case (menambah pasien) kebutuhan sistem informasi Rumah Sakit.

Use case	Menambah Pasien
Tujuan/sasaran	Data identitas pasien baru dengan benar dimasukkan ke sistem.
Aktor utama **Aktor tambahan**	Petugas
Rangkaian kejadian utama **Aksi aktor**	 Respons sistem

1. Use case dimulai ketika ada pasien baru. 2. Petugas atau administrator RS memasukkan nama pasien, tanggal lahir, dan alamat.	3. Nomor pasien yang unik dihasilkan oleh sistem.
Kejadian alternatif:	

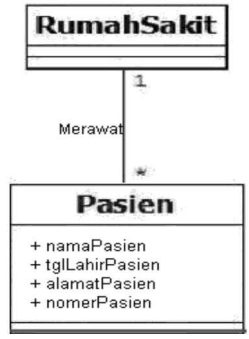
a. Identifikasi kandidat kelas

Kandidat Kelas		Status	Keterangan
Administrator RS			Dibuang tidak diperlukan
Pasien		Kelas	Pasien → Kelas
Sistem		Kelas	Sistem→ Rumah Sakit→ Kelas
Nama pasien		Atribut Kelas	Pasien
Tanggal lahir pasien		Atribut Kelas	Pasien
Alamat pasien		Atribut Kelas	Pasien
Nomor pasien		Atribut Kelas	Pasien

b. Identifikasi relasi antarkelas

Kelas	Nama Relasi	Kelas	Keterangan
Rumah Sakit	Merawat	Pasien	Asosiasi

c. Diagram kelas (untuk satu use case)



d. Integrasi diagram kelas (jika ada lebih dari satu use case, nama kelas yang sama hanya muncul sekali)

BAB 4
PERILAKU SISTEM

Perilaku sistem merupakan muara dari pembahasan materi sebelumnya yang terkait dengan use case (dengan diagram use case) dan kelas (dengan diagram kelas). Apabila kelas menjelaskan aspek statis (struktur statis) dari sistem maka perilaku sistem menjelaskan tentang aspek dinamisnya (struktur dinamis).

Apa yang akan dibangun/dihasilkan dalam bab ini adalah berupa Diagram Sekuens Sistem (DSS). DSS merupakan gambaran atau visualisasi interaksi antara pengguna/aktor dengan sistem. DSS bersama dengan diagram use case dan diagram kelas merupakan artefak-artefak hasil analisis sistem dan yang selanjutnya akan menjadi dasar dalam pengembangan fase berikutnya, yaitu fase/tahap desain sistem (*logical design*).

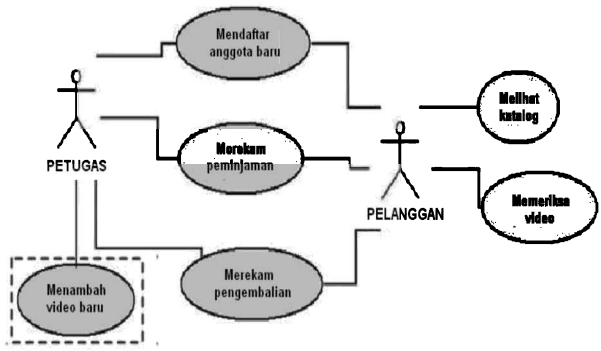
Sebagaimana diketahui bersama bahwa dalam fase analisis, adanya keinginan untuk menjawab pertanyaan tentang apa yang akan dapat sistem lakukan (sebagai respons terkait dengan aksi pengguna/aktor) sehingga sistem dalam hal ini bertindak sebagai "black box" atau kotak hitam. Adapun tentang bagaimana nantinya sistem (secara internal sistem atau kotak hitam) melaksanakan tanggung jawabnya dalam penyelesaian tugas-tugas yang lebih detail tidak dibahas di sini dan akan dibahas dalam bab berikutnya, yaitu tentang Diagram Sekuens (DS).

4.1 Membangun Diagram Sekuens Sistem (DSS)

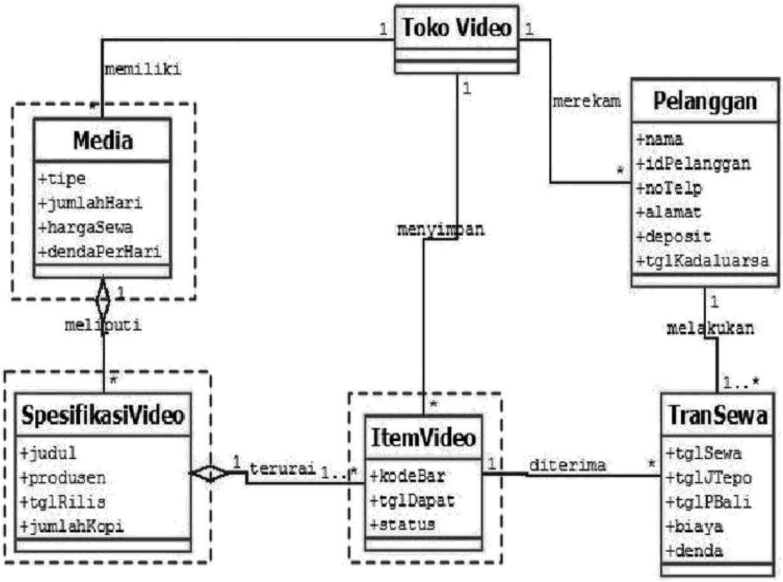
Pada dasarnya setiap DSS dibangun dari setiap use case. Andaikan dalam sebuah sistem informasi persewaan video Bahagia terdapat diagram use case berikut (Gambar 4.1), dengan diagram kelas pada Gambar 4.2. Tampak bahwa dalam Gambar 4.1 terdapat 2 aktor, yaitu petugas dan pelanggan, serta 6 use case: (1) menambah video baru, (2) melihat katalog, (3) memeriksa video, (4) menambah anggota baru, (5) merekam peminjaman, dan (6) merekam pengembalian.

Berdasar pada use case menambah video baru, akan dapat bangun DSS menambah video baru. Untuk itu, diperlukan juga kelas-kelas mana saja (dalam diagram kelas Gambar 4.2) yang sekiranya berkontribusi pada penyusunan DSS menambah video baru. Tampak bahwa terdapat 3 kelas yang relevan, yaitu kelas ItemVideo, SpesifikasiVideo, dan Media. Setelah itu, kita pilih statemen-statemen atau kejadian-kejadian (aksi-aksi) aktor/ pengguna/petugas yang mana dari deskripsi use case menambah video baru pada Tabel 4.1 yang relevan dalam berinteraksi dengan sistem. Tampak bahwa aksi aktor dari statemen no. 2, 4, dan 2a (kejadian alternatif) yang terpilih. Berdasar pada ke-3 statemen terpilih itulah bersama dengan ke-3 kelas yang relevan akan dapat digunakan menyusun DSS menambah video baru.

Dari ke-3 statemen tersebut (setelah diurut sesuai dengan kejadian, yaitu no. 2; no. 2a; no. 4) disusun di sebelah kiri gambar Diagram Sekuens Sistem atau DSS (Gambar 4.3). Hal itu diperlukan guna memudahkan dalam membangun DSS menambah video baru, mengingat dari ke-3 statemen tersebut akan dapat diperoleh 3 operasi sistem, yaitu (sesuai urutan), (1) temuVideoSpes(tipe, judul, produsen), (2) tambahVideoSpes(tipe, judul, produsen, tglRilis), dan (3) tambahVideoItem(tglDapat), sebagaimana tertera pada Gambar 4.3.



Gambar 4.1 Diagram use case sistem persewaan video Bahagia



Gambar 4.2 Diagram kelas sistem persewaan video Bahagia

Tabel 4.1 Deskripsi Use Case Menambah Video Baru

Nama use case	Menambah video baru
Tujuan/sasaran	Merekam sebuah item video baru yang diperoleh Toko
Aktor utama **Aktor tambahan**	Petugas -
Pemicu	Item video baru telah didapat/diperoleh
Rangkaian kejadian khusus **Aksi aktor**	 Respons sistem
1. Use case mulai ketika petugas ingin merekam video baru yang telah didapat.	
2. Petugas memverifikasi catatan tentang kebaruan video (media, judul, produsen).	3. Sistem memeriksa dan mengonfirmasi bahwa status video baru belum pernah terekam (masih baru).

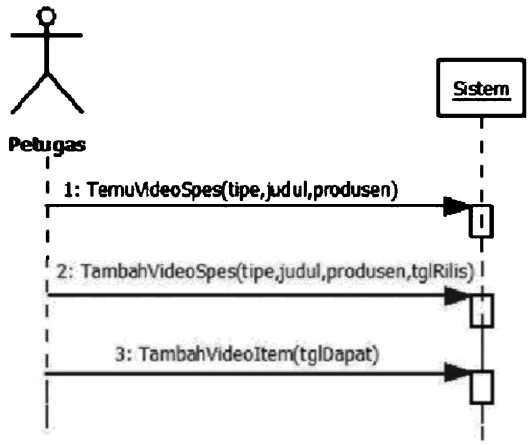
4. Petugas merekam tanggal diperolehnya item video.	5. Sistem membangkitkan nomor kode barang bagi item video baru dan menginputkan jumlah kopi dengan 1 untuk spesifikasi video.
Rangkaian kejadian alternatif	
No. 2a: Apabila status video adalah baru, rekam nilai-nilai atribut video baru (tipe media, judul, produsen, tanggalRilis) untuk spesifikasi video.	

Dari use case:

2: Petugas memverifikasi catatan tentang kebaruan video (media, judul, produsen).

2a. Jika status video adalah baru, rekam nilai-nilai atribut video baru (tipe media, judul, produsen, tanggalRilis) untuk spesifikasi video.

4: Petugas merekam tanggal didapatnya item video.



Gambar 4.3 DSS menambah item video baru

Berdasar DSS pada Gambar 4.3 diperoleh 3 operasi sistem, yaitu sebagai berikut.

1. TemuVideoSpes(tipe, judul, produsen).
2. TambahVideoSpes(tipe, judul, produsen, tglRilis).
3. TambahVideoItem(tglDapat).

Untuk mengetahui selanjutnya tentang bagaimana setiap operasi bekerja dalam sistem (terkait dengan objek-objek yang ada

dalam sistem), akan dijabarkan lebih lanjut dalam Diagram Sekuens (DS) pada bab berikutnya. Dalam hal ini, untuk satu operasi akan dideskripsikan lebih detail dalam 1 DS.

4.2 Contoh Terapan

Studi Kasus: Manajemen Proyek Perusahaan Tanpa Nama (TN)

Sebuah perusahaan, TN memiliki sejumlah pekerja. Pekerja teridentifikasi via nomor/id pekerja yang unik, nama dan tanggal lahir. Terdapat pekerja yang dibayar secara bulanan (pekerja tetap) dan secara per jam (paruh waktu). Setiap pekerja paruh waktu, untuk setiap harinya, menyampaikan lembar kerja sebagai representasi jumlah jam kerja yang dilakukan setiap hari. Beberapa pekerja mengelola/memimpin pekerja lain. TN mengelola beberapa proyek. Setiap proyek memiliki nama proyek yang unik, tanggal mulai dan tanggal berakhir. Setiap pekerja dapat ditugaskan pada satu atau lebih dari satu proyek atau tanpa proyek. Sebuah proyek harus memiliki paling sedikit satu pekerja.

Use case-use case berikut (1-6) mendeskripsikan kebutuhan sistem:

Use case-1:	**Menambahkan Pekerja Baru**	
Tujuan:	Data seorang pekerja baru dimasukkan dengan tepat ke sistem.	
Aktor Utama:	Manajer SDM	
Deskripsi Aksi Utama	**Step**	**Aksi**
	1	Manajer SDM menginput nama pekerja, alamat, dan tanggal lahir.
	2	Nomor pekerja yang unik dialokasikan oleh Sistem.
Deskripsi Aksi Alternatif	**Step**	**Aksi Percabangan**

	1a	Manajer SDM menginput nama pekerja tetap, alamat, tanggal lahir, dan gaji per bulan.
	1b	Manajer SDM menginput nama pekerja honorer, alamat, tanggal lahir, dan honor per jam.

Use case-2:	**Tambahkan Proyek Baru**	
Tujuan:	Sebuah proyek baru secara tepat diinput ke dalam Sistem.	
Aktor Utama:	Administrator	
Deskripsi Aksi Utama	**Step**	**Aksi**
	1	Administrator menginput nama proyek dan tanggal mulai. Sistem mengatur tanggal akhir dengan nilai null.
Deskripsi Aksi Alternatif	**Step**	**Aksi Percabangan**

Use case-3:	**Menugaskan Pekerja ke Proyek**	
Tujuan:	Seorang pekerja ditugaskan pada sebuah proyek.	
Aktor Utama:	Administrator	
Deskripsi Aksi Utama	**Step**	**Aksi**
	1	Administrator menampilkan detail proyek melalui input nama proyek.
	2	Administrator menampilkan setiap anggota Tim melalui input nomor (id) pekerja.
	3	Administrator menugaskan pekerja ke proyek.
Deskripsi Aksi Alternatif	**Step**	**Aksi Percabangan**

Use case-4:	**Menyampaikan (submit) Lembar Kerja**	
Tujuan:	Membangun/mengisi lembar kerja baru untuk pekerja paruh waktu.	
Aktor Utama:	Administrator	
Deskripsi Aksi Utama	**Step**	**Aksi**

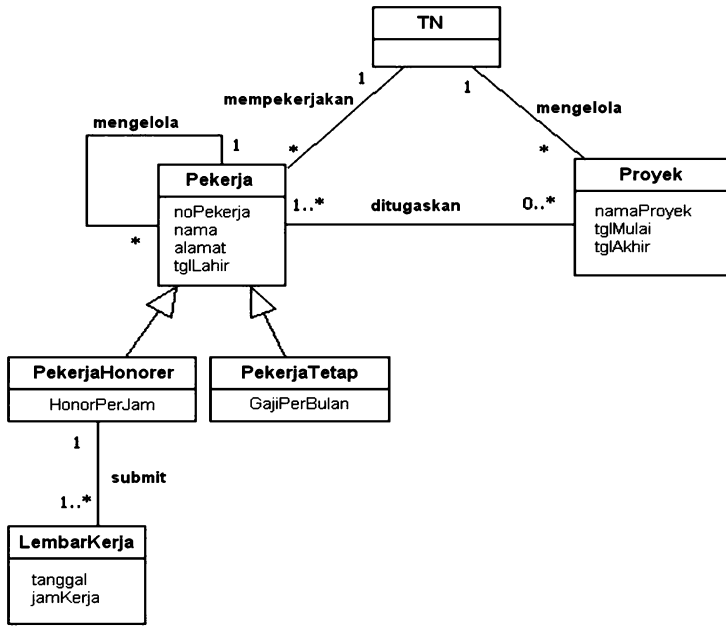
	1	Administrator menampilkan detail proyek melalui input nomor (id) pekerja.
	2	Administrator menginput total jam kerja dari pekerja honorer untuk suatu hari kerja tertentu.
Deskripsi Aksi Alternatif	**Step**	**Aksi Percabangan**

Use case-5:	**Menugaskan Manajer**	
Tujuan:	Menugaskan seorang pekerja untuk mengelola/ memimpin pekerja lain.	
Aktor Utama:	Administrator	
Deskripsi Aksi Utama	**Step**	**Aksi**
	1	Use case ini dimulai bila administrator menginput nomor pekerja manajer.
	2	Administrator menugaskan pekerja yang perlu dikelola/dipimpin melalui input nomor (id) pekerja. Langkah ini dilanjutkan sampai dengan administrator telah menyelesaikan penugasan semua pekerja yang dikelola manajer.
Deskripsi Aksi Alternatif	**Step**	**Aksi Percabanga**n

Use case-6:	**Menghitung Gaji Pekerja Paruh-Waktu**	
Tujuan:	Menghitung gaji/honor pekerja paruh waktu pada suatu periode tertentu.	
Aktor Utama	Administrator	
Deskripsi Aksi Utama	**Step**	**Aksi**
	1	Use-case ini dimulai bila administrator menghendaki sistem mengalkulasi honor pekerja paruh waktu pada suatu periode tertentu.
	2	Administrator menampilkan detail pekerja paruh waktu dengan menginput nomor (id) pekerja.

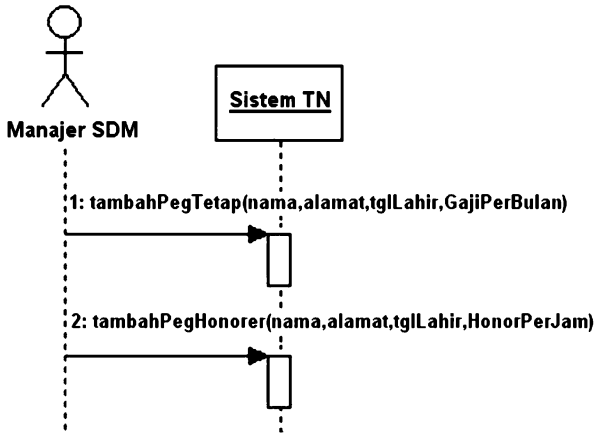
	3	Administrator menginput bulan dan tahun dari periode waktu yang diperlukan.
	4	Semua lembar kerja pekerja paruh-waktu dari periode yang diperlukan ditampilkan.
	5	Total jam kerja dikalkulasi dan dikalikan dengan rate/jam pekerja paruh waktu.
	6	Total honor hasil kalkulasi ditampilkan.
Deskripsi Aksi Alternatif	**Step**	**Aksi Percabangan**
	2a	Use case dibatalkan jika nomor pekerja invalid.
	4a	Use case dibatalkan jika tidak ditemukan lembar kerja pekerja.

Berdasarkan statemen problem dan ke-6 use case kebutuhan sistem tersebut, dapat dibangun diagram kelas (hasil integrasi) sistem manajemen perusahaan TN (Gambar 4.4) melalui langkah-langkah yang telah ditetapkan dalam bab sebelumnya tentang bagaimana membangun diagram kelas.

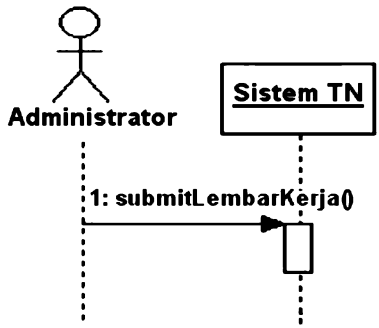


Gambar 4.4 Diagram kelas manajemen proyek perusahaan TN

Dengan memperhatikan use case-1 (menambahkan pekerja baru), use case-4 (menyampaikan/submit lembar kerja) dan diagram kelas, dapat disusun DSS menambahkan pekerja baru (Gambar 4.5) dan DSS submit lembar kerja (Gambar 4.6).



Gambar 4.5 DSS menambah pekerja baru



Gambar 4.6 DSS submit lembar kerja

BAB 5
DIAGRAM SEKUENS

Diagram sekuens merupakan salah satu bentuk diagram interaksi yang memodelkan interaksi antara objek dalam sebuah use case. Mereka menggambarkan bagaimana bagian-bagian yang berbeda dari sebuah sistem berinteraksi satu sama lain untuk melaksanakan fungsi dan urutan ketika use case tertentu dieksekusi. Dengan kata lain, diagram sekuens merupakan aspek detail dari perilaku sistem (interaksi aktor dengan sistem, sebagaimana telah dibahas pada bab sebelumnya).

Interaksi menentukan bagaimana pesan dan data dipertukarkan antara mitra interaksi (objek-objek). Mitra interaksi bisa orang, dosen, siswa, atau nonmanusia, seperti server, printer, atau perangkat lunak yang dapat dieksekusi. Interaksi dapat menjadi percakapan antara beberapa orang—misalnya, ujian lisan; atau memodelkan protokol komunikasi, seperti HTTP atau mewakili pertukaran pesan antara manusia dan sistem perangkat lunak—misalnya, antara dosen dan sistem administrasi siswa-dosen dalam menginformasikan hasil ujian; atau berupa urutan panggilan metode dalam program atau sinyal, seperti alarm kebakaran dan proses komunikasi yang dihasilkan.

Terdapat skenario konkret dalam memodelkan diagram interaksi yang berarti bahwa pertukaran pesan terjadi dalam konteks tertentu untuk memenuhi tugas/fungsi tertentu. Interaksi biasanya hanya menggambarkan bagian tertentu dari sebuah situasi. Sering kali ada jalur eksekusi lain yang valid yang tidak dicakup dalam diagram interaksi.

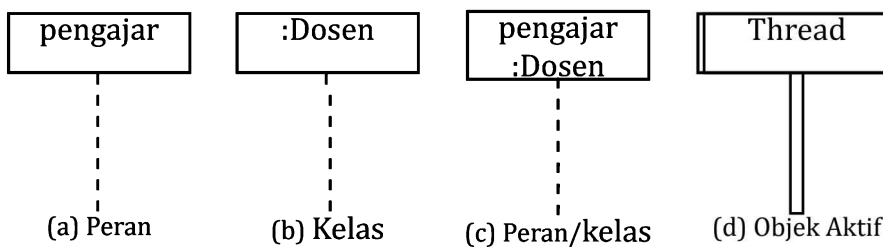
Interaksi menggambarkan mekanisme urutan komunikasi pada tingkat detail yang berbeda, baik bagi pakar komputer maupun untuk pengguna dan pembuat keputusan. Oleh karenanya, diagram interaksi digunakan dalam berbagai situasi, baik sebagai representasi interaksi sistem (sebagai kotak hitam) yang lengkap dengan lingkungannya atau untuk memodelkan interaksi

antarbagian sistem, guna menunjukkan bagaimana use case tertentu dapat diimplementasikan.

Terdapat empat macam diagram interaksi (diagram komunikasi, diagram waktu, diagram interaksi ringkasan, dan diagram sekuens), tetapi diagram sekuens/urutan adalah diagram yang paling sering digunakan —menyajikan urutan interaksi antarobjek dengan cepat, dan selanjutnya akan dibahas lebih lanjut secara lebih detail.

5.1 Mitra Interaksi

Dalam diagram sekuens/urutan, mitra interaksi digambarkan sebagai garis hidup. Garis hidup ditampilkan sebagai garis vertikal dan biasanya berupa garis putus-putus yang mewakili masa pakai objek yang terkait dengannya (Gambar 5.1). Di ujung atas garis adalah kepala garis hidup, persegi panjang yang berisi ekspresi dalam bentuk namaPeran:kelas (Gambar 5.1(c)). Ekspresi ini menunjukkan nama peran dan kelas objek yang terkait dengan garis kehidupan. Dengan cara yang sama seperti diagram objek (Bab sebelumnya), salah satu dari dua nama dapat dihilangkan. Apabila Kelas dihilangkan maka titik dua dapat dihilangkan (Gambar 5.1 (a)); tetapi jika hanya Kelas yang ditentukan, titik dua harus mendahului nama Kelas (Gambar 5.1 (b)). Dengan demikian, diagram sekuens dapat ditentukan pada tingkat instans dan tingkat Kelas.



Gambar 5.1 Garis hidup

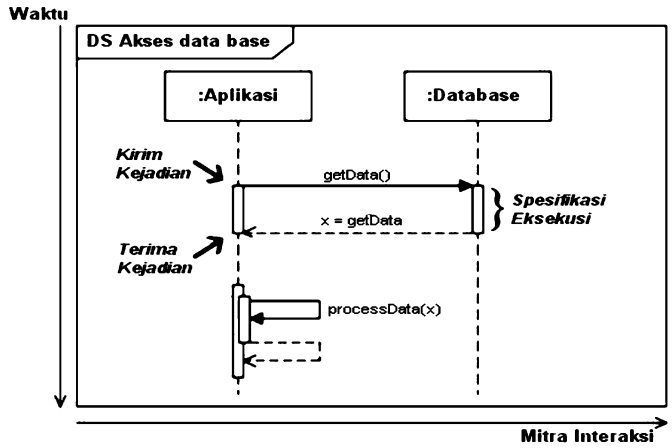
Garis hidup dapat mewakili objek aktif yang digunakan untuk memodelkan proses dan *thread*. Objek aktif memiliki aliran kontrolnya sendiri, yang berarti dapat beroperasi secara independen dari objek lain. Kepala garis hidup yang mewakili objek aktif memiliki batas ganda di sebelah kiri dan kanan. Bilah berkelanjutan sering digunakan sebagai pengganti garis putus-putus (Gambar 5.1(d)).

5.2 Pertukaran Pesan

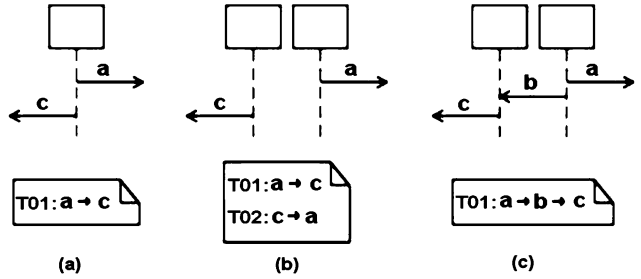
Diagram sekuens merupakan diagram dua dimensi (Gambar 5.2). Mitra interaksi yang terlibat dalam interaksi disajikan pada sumbu horizontal dan harus diatur dalam sekuens/urutan yang jelas. Sumbu vertikal merupakan urutan kronologis interaksi yang menspesifikasikan urutan kejadian-kejadian. Spesifikasi kejadian mencakup pengiriman dan penerimaan pesan atau terjadinya kejadian berbasis waktu seperti titik waktu.

Sumbu waktu vertikal menentukan urutan kejadian pada garis hidup, meskipun ini tidak menentukan urutan kejadian pada garis hidup yang berbeda. Perintah/pesan di beberapa garis hidup hanya dipaksakan jika pesan dipertukarkan antara garis hidup yang berbeda, kecuali ditentukan sebaliknya. Berikut ini diasumsikan bahwa transmisi pesan tidak memerlukan waktu, yang berarti bahwa kejadian pengiriman di pengirim dan kejadian penerimaan di penerima berlangsung pada saat yang sama. Ini memungkinkan kita untuk menyajikan jejak lebih kompak karena kita tidak perlu mempertimbangkan urutan kirim dan terima kejadian serta dapat berkonsentrasi pada urutan pesan. Hubungan kronologis antara pesan a dan pesan b dinyatakan oleh simbol →. Sebagai contoh, kalau diketahui pesan a → b, berarti pesan a dikirim sebelum pesan b.

Perhatikan Gambar 5.3 dengan urutan pesan yang dimungkinkan. Apabila kejadian kirim dan terima dua pesan terjadi di sepanjang garis hidup yang sama, urutan kronologis kejadian ini menentukan urutan pesan. Pada Gambar 5.3(a), pesan a harus selalu terjadi sebelum pesan c karena kejadian pengiriman berlangsung sebelum peristiwa pengiriman c. Apabila dua pesan tidak memiliki mitra interaksi yang sama (Gambar 5.3(b)) dan urutan pesan a serta c ini tidak ditentukan maka ada dua kemungkinan pelacakan/jejak: a → c dan c → a. Apabila ada pesan b disisipkan antara a dan c dan pesan ini memaksa a serta c ke dalam urutan kronologis, satu-satunya jejak yang mungkin adalah a → b → c (lihat Gambar 5.3(c)).



Gambar 5.2 Struktur Diagram Sekuens (DS)



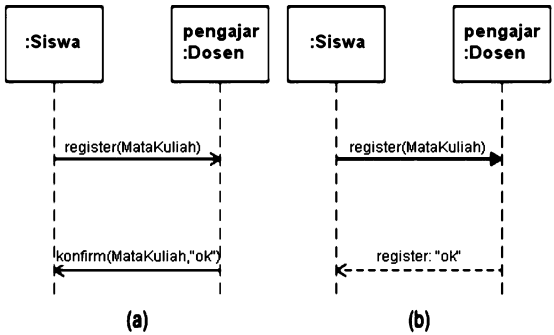
Gambar 5.3 Urutan pesan-pesan dan pelacakan/jejak

Eksekusi perilaku ditunjukkan oleh dua kejadian, yaitu yang memulai dan yang menyelesaikan eksekusi ini pada garis hidup yang sama (Gambar 5.2). Perilaku ini divisualisasikan dengan batang/balok dan disebut sebagai spesifikasi eksekusi. Terdapat perbedaan antara perilaku eksekusi langsung dan tidak langsung. Dalam kasus eksekusi langsung, mitra interaksi yang memengaruhi eksekusi perilaku yang ditentukan itu sendiri; sementara untuk eksekusi tidak langsung, mitra interaksi melimpahkan eksekusi kepada mitra interaksi lainnya. Spesifikasi eksekusi yang tumpang-tindih ditunjukkan dengan tumpang-tindihnya balok. Apabila mitra interaksi mengirim pesan ke dirinya sendiri dan spesifikasi eksekusi terkait akan dimodelkan secara eksplisit, dua batang spesifikasi eksekusi ditampilkan tumpang-tindih, di mana panah pesan menunjuk ke bilah kedua, serta dalam kasus pesan yang sinkron (bersamaan), panah respons putus-putus di akhir eksekusi mengarah kembali ke balok asli (lihat processData(x) pada Gambar 5.2). Pemodelan spesifikasi eksekusi bersifat opsional, khususnya digunakan untuk memvisualisasikan ketika mitra interaksi menjalankan beberapa perilaku.

5.3 Pesan-Pesan

Dalam diagram sekuens, pesan digambarkan sebagai anak panah (panah) dari pengirim ke penerima. Jenis panah mengekspresikan jenis komunikasi yang ditransmisikan. Pesan sinkron disimbolkan dengan panah bergaris solid/padat dan kepala panah segitiga yang terisi. Pesan asinkron disimbolkan dengan panah bergaris solid/padat dan kepala panah terbuka. Dalam kasus pesan sinkron, pengirim menunggu sampai menerima pesan respons sebelum melanjutkan. Pesan respons disimbolkan oleh garis putus-putus dengan kepala panah terbuka. Apabila konten pesan respons dan titik di mana pesan respons dikirim serta diterima dengan jelas sesuai konteks maka pesan respons dapat

dihilangkan dalam diagram. Dalam komunikasi asinkron, pengirim berproses lanjut setelah mengirim pesan. Dua contoh ditampilkan dalam Gambar 5.4.



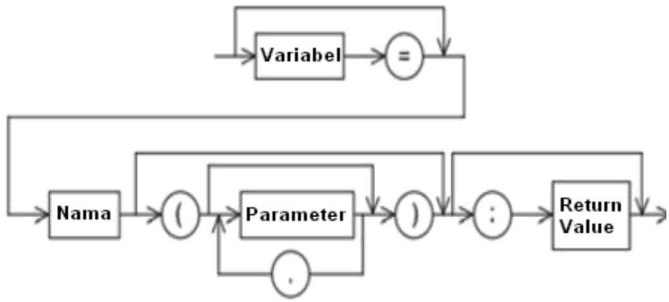
Gambar 5.4 Contoh komunikasi asinkron (a), dan sinkron (b)

Gambar 5.4 menunjukkan contoh kasus seorang siswa berkomunikasi dengan seorang Dosen dalam mendaftar/mengikuti mata kuliah. Untuk kasus (a), pendaftaran dilakukan via e-mail sehingga bersifat asinkron. Siswa tidak secara eksplisit menunggu penerimaan pesan konfirmasi. Untuk kasus (b), siswa mendaftar via Dosen secara pribadi sehingga bersifat sinkron. Dalam hal ini, siswa menunggu sampai menerima pesan respons hasilnya.

Pesan diidentifikasi dengan nama, dengan spesifikasi parameter opsional dan nilai yang dikembalikan atau *return value* (Gambar 5.5). Parameter dipisahkan oleh koma dan dimasukkan ke dalam tanda kurung. Nilai yang dikembalikan juga dapat ditetapkan secara opsional ke variabel. Untuk itu, pesan dapat diberi label var=m1:value, dengan var adalah variabel yang akan ditetapkan oleh value/nilai yang dikembalikan, m1 menentukan nama pesan, dan value mewakili nilai pengembalian aktual.

Penerimaan pesan oleh objek umumnya memanggil operasi terkait yang ditentukan dalam diagram kelas (Bab 3). Pada dasarnya, argumen yang dilewatkan harus kompatibel dengan

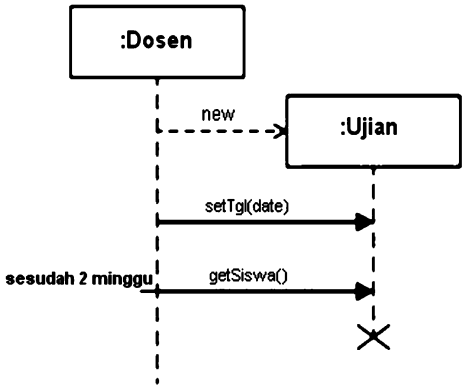
parameter spesifikasi operasi dalam diagram kelas. Namun, kalau nama parameter digunakan untuk menetapkan nilai ke parameter yang sesuai, baik angka maupun urutan argumen, tidak harus sesuai dengan parameter dalam spesifikasi operasi.



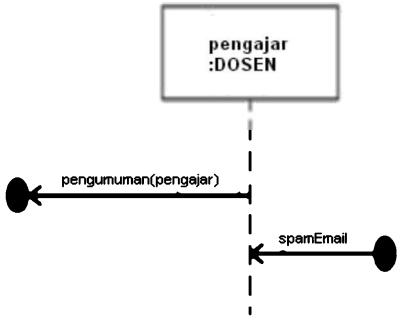
Gambar 5.5 Sintaks spesifikasi pesan

Pesan untuk membuat objek bertipe pesan khusus. Ini digambarkan oleh panah putus-putus dengan kepala panah terbuka yang berakhir di kepala garis hidup yang terkait dengan objek yang akan dibuat. Panah diberi label kata kunci baru dan sesuai dengan memanggil konstruktor dalam bahasa pemrograman berorientasi objek. Contoh pada Gambar 5.6, seorang Dosen membuat tanggal ujian baru dengan setTgl(date).

Apabila objek dihapus selama interaksi, yaitu peristiwa penghancuran terjadi, akhir garis hidup ditandai dengan X besar (lihat Gambar 5.6). Jika tidak, garis hidup membentang ke ujung bawah diagram urutan.



Gambar 5.6 Membangun sebuah objek

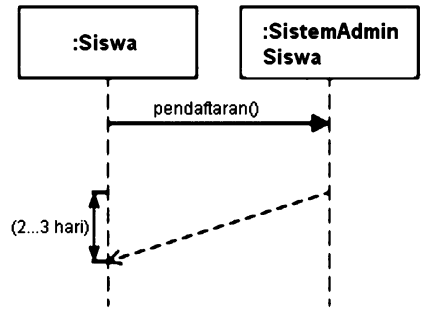


Gambar 5.7 Contoh pesan-pesan yang hilang dan ditemukan

Apabila pengirim pesan tidak diketahui atau tidak relevan, kita dapat mengekspresikannya dengan pesan yang ditemukan (*found*). Dalam hal ini, notasi lingkaran hitam sebagai sumber pengganti menentukan mitra interaksi yang mengirim pesan. Kondisi yang berbeda bisa terjadi jika penerima pesan tidak diketahui, dalam hal ini tipe pesan disebut pesan yang hilang (*lost*). Gambar 5.7, menunjukkan tipe pesan yang *found* maupun yang *lost*.

Gambar 5.8 menunjukkan contoh skenario. Seorang siswa mendaftar untuk program studi dalam sistem administrasi siswa. Dalam dua hingga tiga hari ke depan, siswa menerima pesan konfirmasi yang menegaskan bahwa pendaftaran berhasil.

Konfirmasi ini dikirim sebagai surat tradisional dan karenanya ada transit selama beberapa hari sebelum siswa menerimanya.



Gambar 5.8 Contoh pesan dengan konsumsi waktu

5.4 Fragmen Kombinasi

Dalam diagram sekuens, dimungkinkan penggunaan fragmen gabungan/kombinasi (operator) untuk memodelkan berbagai struktur kontrol secara eksplisit sehingga dapat menggambarkan sejumlah jalur eksekusi yang ringkas dan tepat. Fragmen kombinasi disimbolkan/dinotasikan dengan persegi panjang, dengan tipe operator yang ditentukan oleh kata kunci masing-masing dalam segilima kecil di sudut kiri atas persegi panjang ini.

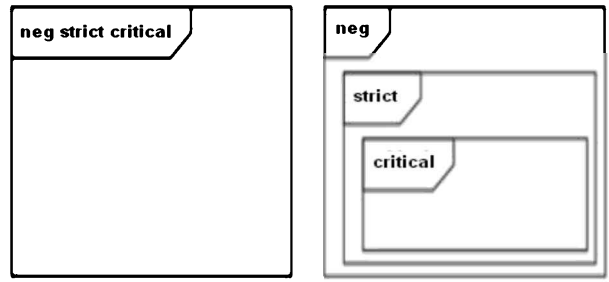
Terdapat 12 jenis operator yang berbeda (pembahasannya di sini dibatasi 6 operator yang dipandang cukup sering digunakan). Operator bisa berisi satu atau beberapa operan yang pada gilirannya dapat berisi interaksi, fragmen gabungan, atau referensi ke diagram sekuens lainnya. Operan yang berbeda dari operator dipisahkan satu sama lain dengan garis horizontal dan putus-putus. Ke-6 jenis operator yang berbeda tersebut terbagi atas dua grup:

1. Percabangan dan Perulangan
2. Konkurensi dan Urutan

Tabel 5.1 menunjukkan ke-6 operator yang tersedia dengan kata kunci yang sesuai dan semantiknya. Fragmen kombinasi dapat ditumpuk secara bersarang, dengan bingkai ditentukan untuk setiap fragmen. Fragmen bersarang dapat berbagi bingkai. Operator terkiri ditugaskan ke fragmen terluar dan operator terkanan ditugaskan ke fragmen terdalam (Gambar 5.9).

Tabel 5.1 Operator bagi Fragmen Kombinasi

	Operator	Tujuan
Percabangan dan Perulangan	alt	Interaksi alternatif
	opt	Interaksi opsional
	loop	Interaksi iteratif
	break	Interaksi pengecualian
Konkurensi dan Urutan	seq	Urutan lemah
	strict	Urutan kuat



Gambar 5.9 Notasi alternatif fragmen kombinasi tersarang

5.4.1 Percabangan dan Perulangan

Sebagaimana Tabel 5.1, operator **alt** mewakili urutan alternatif. Operator ini memiliki setidaknya dua operan. Setiap operan mewakili jalur alternatif dalam eksekusi, yang terkait kira-kira dengan beberapa kasus dalam bahasa pemrograman, misalnya, pernyataan **switch** di Java. Penjaga digunakan untuk memilih jalur yang akan dieksekusi.

Setiap operan memiliki penjaga. Sebuah penjaga adalah ekspresi boolean tertutup dalam kurung siku. Apabila tidak ada penjaga maka **[true]** diasumsikan sebagai nilai default. Jika beberapa penjaga benar secara bersamaan, ini menghasilkan ketidaktentuan. Dalam hal ini, tidak ada prediksi mengenai operan mana yang dipilih. Kondisi tersebut kontras dengan semantik pernyataan **switch** dalam bahasa pemrograman, di mana alternatif biasanya diproses dari atas ke bawah. Penjaga khusus adalah **[else]** yang dievaluasi sebagai benar jika tidak ada kondisi lain yang terpenuhi.

Apabila tidak ada penjaga yang mengevaluasi ke **true**, tidak ada operan yang dieksekusi dan eksekusi fragmen di sekitarnya berlanjut. Gambar 5.10 memperlihatkan contoh fragmen **alt**. Ketika siswa ingin mendaftar (*register*) ujian, kasus-kasus berikut dapat terjadi (1) Masih ada tempat yang tersedia dan siswa dapat mendaftar; (2) Ada tempat yang tersedia di daftar tunggu (*waiting list* atau WL). Selanjutnya, siswa harus memutuskan apakah akan masuk (*enter*) ke daftar tunggu (WL); dan (3) Apabila tidak ada tempat yang tersedia untuk ujian atau di daftar tunggu ujian, siswa menerima pesan kesalahan (*error*) dan tidak terdaftar untuk mengikuti perkuliahan.

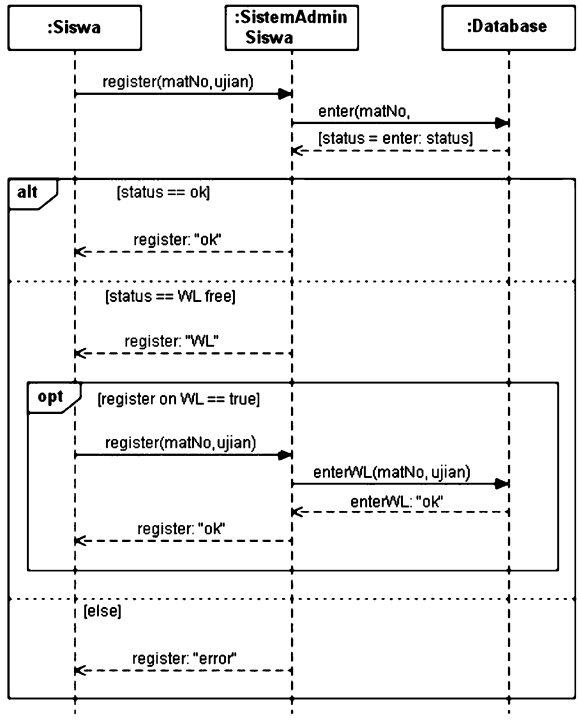
Fragmen **opt** sesuai dengan fragmen **alt** dengan dua operan, salah satunya kosong. Operator **opt** dengan demikian mewakili urutan interaksi yang eksekusi aktualnya pada waktu proses tergantung pada penjaga. Dalam bahasa pemrograman, operator ini akan ditentukan sebagai pernyataan **if** tanpa cabang lain. Gambar 5.10 menggambarkan penggunaan fragmen **opt**. Apabila ada tempat yang tersedia di daftar tunggu, saat mendaftar tugas, siswa dapat memutuskan apakah akan mengambil tempat di daftar tunggu. Jika siswa ingin berada di daftar tunggu, siswa harus mendaftar untuk itu.

Fragmen **loop** digunakan untuk mengekspresikan bahwa urutan akan dieksekusi berulang kali. Fragmen gabungan ini memiliki satu operan. Kata kunci **loop** diikuti oleh spesifikasi opsional dari jumlah iterasi **loop**. Spesifikasi ini mengambil bentuk (min.. max) atau (min,max), di mana min menentukan jumlah minimum iterasi yang harus dilalui **loop** dan max menunjukkan jumlah maksimum iterasi. Apabila min dan max identik, salah satu dari dua angka dan titik-titik dapat dihilangkan. Semisal tidak ada batas atas jumlah iterasi loop maka dapat diganti dengan tanda bintang *. Dalam hal ini, jumlah minimum iterasi diasumsikan nol. Seumpama loop kata kunci tidak diikuti oleh spesifikasi lebih lanjut dari jumlah iterasi, * diasumsikan sebagai nilai default. Jika diperlukan, penjaga dapat ditentukan, yang kemudian memeriksa untuk setiap iterasi dalam batas (min,max). Dalam kondisi tersebut, penjaga mengevaluasi segera setelah jumlah minimum iterasi telah terjadi. Semisal kondisi yang mendasarinya tidak terpenuhi, eksekusi loop dihentikan meskipun jumlah maksimum eksekusi belum tercapai.

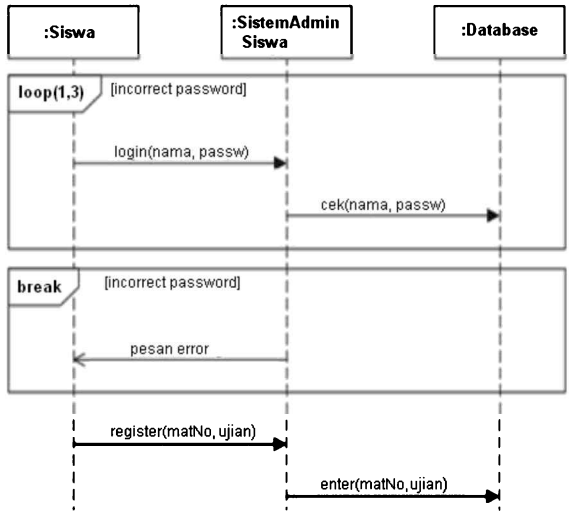
Gambar 5.11 memperluas contoh dari Gambar 5.10 untuk menyertakan login sistem yang diperlukan sebelum siswa dapat mendaftar untuk tugas. Kata sandi harus dimasukkan setidaknya sekali dan paling banyak tiga kali, seperti yang tecermin oleh argumen loop. Setelah upaya pertama, sistem memeriksa apakah kata sandi dapat divalidasi. Semisal bisa, yaitu, kondisi sandi yang salah tidak lagi benar, eksekusi interaksi dalam **loop** berhenti. Sistem juga keluar dari **loop** jika siswa salah memasukkan kata sandi tiga kali. Kasus ini kemudian ditangani lebih lanjut dalam fragmen **break** berikutnya.

Fragmen **break** memiliki struktur yang sama dengan operator **opt**, yaitu terdiri dari satu operan ditambah penjaga. Jika penjaga benar, interaksi dalam operan ini dijalankan, operasi sisa fragmen

di sekitarnya diabaikan, dan interaksi berlanjut di fragmen tingkat yang lebih tinggi berikutnya. Operator **break** dengan demikian menawarkan bentuk penanganan pengecualian yang sederhana. Contoh pada Gambar 5.11 kalau kata sandi (*password*) dimasukkan secara tidak benar tiga kali maka kondisi *incorrect password* adalah benar (*true*). Dengan demikian konten fragmen **break** dieksekusi sehingga pesan kesalahan (*error*) dikirim ke siswa dan siswa tidak diizinkan untuk mendaftar tugas. Sisa interaksi setelah akhir fragmen **break** dilewati atau skip. Setelah keluar dari operator **break**, posisi eksekusi lanjutan berada di fragmen terluar diagram sekuens sehingga eksekusi diagram sekuens ini berakhir. Dalam hal posisi eksekusi lanjutan tidak berada dalam fragmen terluar, diagram sekuens akan berlanjut pada fragmen dengan tingkat yang lebih tinggi berikutnya.



Gambar 5.10 Contoh fragmen Alt dan Opt



Gambar 5.11 Contoh fragmen Break dan Loop

5.4.2 Konkurensi dan Urutan

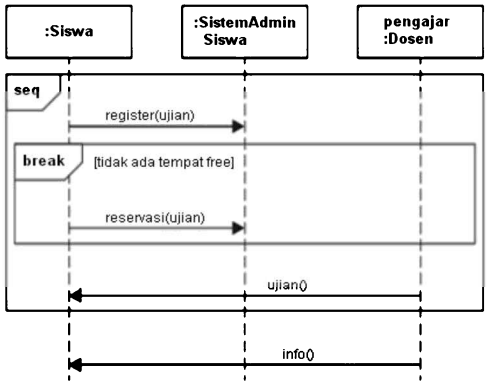
Sebagaimana diketahui, pengaturan kejadian/peristiwa pada sumbu vertikal merepresentasikan urutan kronologis kejadian, asalkan ada pertukaran pesan antara mitra interaksi yang terlibat. Fragmen gabungan yang dijelaskan berikut ini secara eksplisit dapat mengontrol urutan kejadian/peristiwa.

Fragmen **seq** mewakili urutan secara **default**. Operator **seq** setidaknya punya satu operan dan mengekspresikan urutan yang lemah sebagai berikut.

1. Pemesanan kejadian dalam setiap operan dipertahankan hasilnya.
2. Kejadian pada garis hidup yang berbeda dari operan yang berbeda dapat datang dalam urutan apa pun.
3. Kejadian pada garis hidup yang sama dari operan yang berbeda diperintahkan sedemikian rupa sehingga kejadian operan pertama datang sebelum operan kedua.

Fragmen **seq** bersama dengan fragmen **break** dapat digunakan untuk mengelompokkan pesan. Apabila kondisi fragmen **break** bernilai benar, pesan dari fragmen **seq** yang belum dieksekusi dilewati, dan eksekusi diagram sekuens berlanjut ke fragmen sekitarnya. Gambar 5.12 menunjukkan contoh tersebut. Seorang siswa ingin mendaftar untuk ujian. Jika tidak ada lagi tempat yang tersedia untuk tanggal yang diinginkan, siswa membuat reservasi untuk tanggal berikutnya (fragmen **break**). Dalam hal ini, siswa tidak diperiksa oleh pengajar dan eksekusi diagram sekuens berlanjut di luar fragmen **seq**. Terlepas dari apakah pendaftaran berhasil atau tidak, pengajar mengirimkan pesan info() kepada siswa.

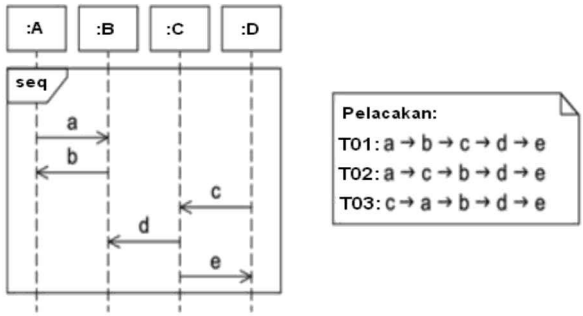
Perhatikan bahwa operator **seq** adalah urutan default dan umumnya tidak harus dimodelkan secara eksplisit. Namun, dalam kasus ini, tanpa pemodelan secara eksplisit fragmen **seq**, eksekusi akan berakhir setelah fragmen **break** jika *incorrect password* adalah benar. Hal ini disebabkan oleh fakta bahwa setelah mengeksekusi konten fragmen **break**, operasi fragmen di sekitarnya diabaikan. Tanpa menggunakan **seq**, fragmen di sekitarnya akan menjadi struktur terluar diagram dan dengan demikian seluruh eksekusi akan berakhir.



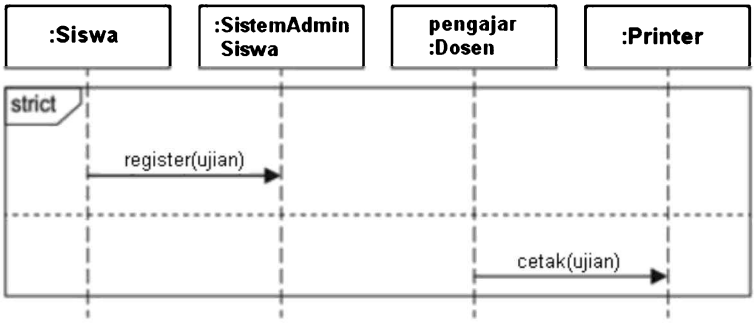
Gambar 5.12 Contoh fragmen Break dan Seq

Gambar 5.13 memperlihatkan diagram sekuens lain bersama dengan semua pelacakan/jejak yang mungkin. Karena diagram ini menunjukkan urutan yang lemah, pesan c tidak terhubung secara kronologis ke pesan a dan b, serta dapat disisipi (*interleaved*) dengan pesan-pesan ini. Karena b dikirim oleh mitra interaksi B dan d juga diterima oleh B, ada urutan kronologis antara kedua pesan ini. Bagaimanapun e adalah pesan terakhir.

Fragmen **strict** menggambarkan interaksi berurutan sesuai perintah. Urutan kejadian/peristiwa pada garis hidup yang berbeda antara operan yang berbeda sangat signifikan, yang berarti bahwa jika tidak ada pertukaran pesan antara mitra interaksi, pesan dalam operan yang lebih tinggi pada sumbu vertikal selalu dipertukarkan sebelum pesan dalam operan yang lebih rendah pada sumbu yang sama.



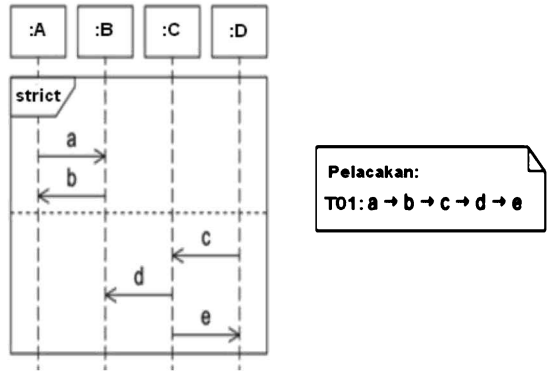
Gambar 5.13 Pelacakan sebuah fragmen Seq



Gambar 5.14 Contoh sebuah fragmen Strict

Dalam contoh pada Gambar 5.14, seorang dosen hanya mencetak ujian ketika seorang siswa telah mendaftar ujian. Jika fragmen Strict tidak ditentukan, akan ada kemungkinan bagi dosen untuk mencetak ujian sebelum mahasiswa mendaftar.

Gambar 5.15 berisi urutan pesan yang sama sebagaimana Gambar 5.13, tetapi kali ini berdasarkan urutan yang ketat. Ini berarti bahwa pesan berada dalam urutan tetap dan hanya ada satu jejak.



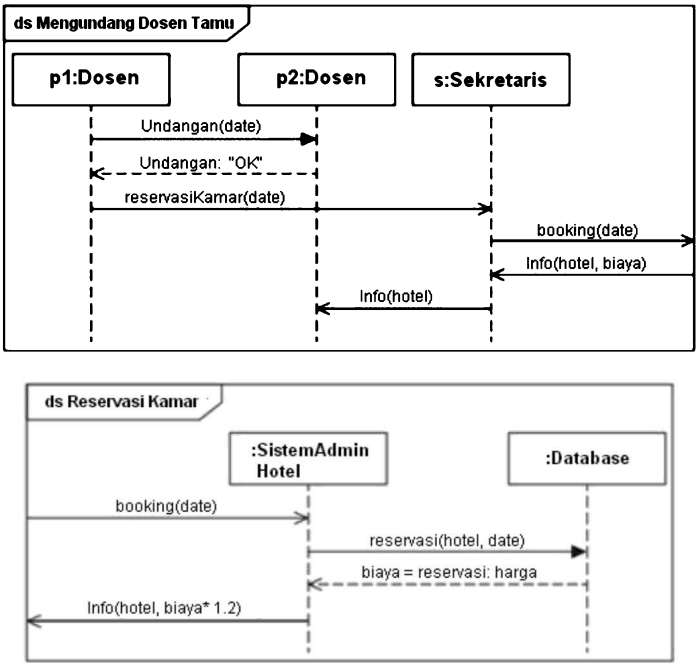
Gambar 5.15 Pelacakan pada fragmen Strict

5.4.3 Gate

Sebagaimana yang telah dibahas sebelumnya tentang operator atau fragmen kombinasi, pertukaran pesan pada dasarnya tidak boleh melampaui batas interaksi atau fragmen kombinasi yang berarti bahwa pesan-pesan tersebut tidak boleh melebihi bingkai secara sewenang-wenang. Dalam hal ini, **Gate** sebagai salah satu elemen bahasa lainnya di luar yang tertera di Tabel 5.1, memungkinkan pertukaran pesan dilakukan di luar batas tersebut. **Gate** (gerbang) divisualisasikan oleh ujung atau akhir panah pesan—bergantung pada apakah pesan tersebut merupakan pesan masuk atau keluar—menyentuh batas bingkai yang mewakili diagram sekuens atau fragmen gabungan. **Gate** diidentifikasi baik dengan nama atau dengan nama pesan yang menggunakan **Gate**

secara opsional bersama-sama dengan arah pesan (contoh kasus reservasi pada Gambar 5.16).

Melalui **Gate**, penentuan pengirim tertentu dan penerima tertentu untuk setiap pesan dapat terjadi, bahkan jika pengirim atau penerima berada di luar interaksi masing-masing atau di luar fragmen. Penyertaan **Gate** secara eksplisit untuk fragmen gabungan, bisa berarti bahwa pesan dapat menunjuk langsung ke penerima.



Gambar 5.16 Contoh sebuah Gate

5.5 Membangun Diagram Sekuens

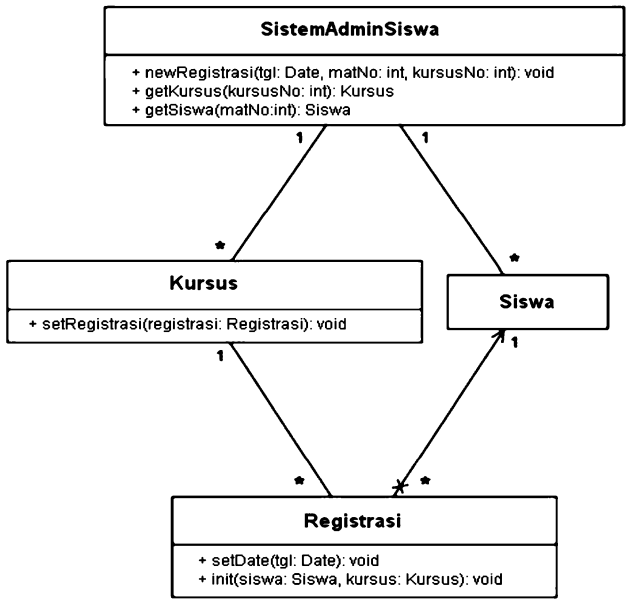
Bagian ini akan menyampaikan contoh dua skenario dalam membangun diagram sekuens. Skenario pertama mengilustrasikan hubungan antara diagram kelas dan diagram sekuens. Skenario berikutnya, menyimpulkan bagian dengan aplikasi khas untuk diagram sekuens, yaitu deskripsi pola desain.

5.5.1 Hubungan Diagram Kelas dengan Diagram Sekuens

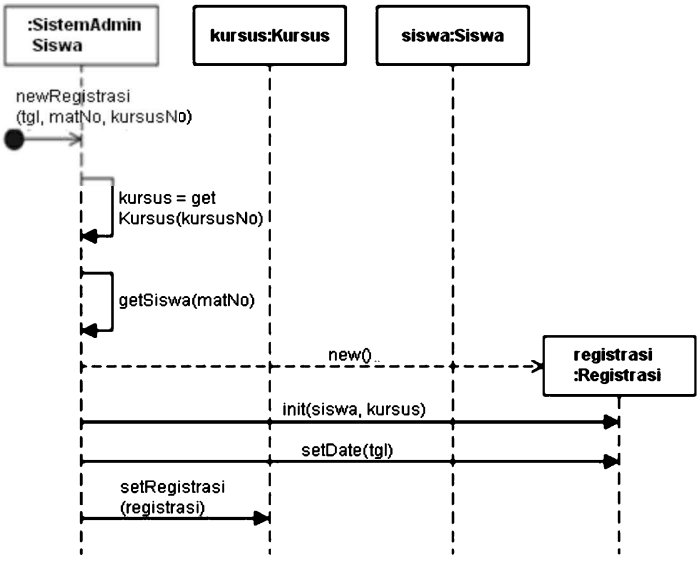
Perlu diketahui bahwa diagram UML yang berbeda tidak boleh dianggap independen satu sama lain; mereka hanya menawarkan pandangan yang berbeda dari konten tertentu. Misalnya, diagram kelas yang ditampilkan dalam model Gambar 5.17, sebagai bagian dari sistem universitas yang juga mencakup sistem administrasi siswa.

Sistem administrasi siswa memiliki akses langsung ke semua siswa dan kursus. Sistem ini mengetahui data pendaftaran siswa dan kursus yang disimpan di kelas Pendaftaran/Registrasi. Selanjutnya gambaran komunikasi diperlukan untuk membuat registrasi siswa baru tertentu untuk kursus tertentu. Untuk itu, metode newRegistrasi dari kelas SistemAdminSiswa harus dipanggil.

Untuk membuat pendaftaran baru, perlu mengetahui objek siswa yang termasuk dalam nomor matrikulasi masing-masing dan objek kursus yang termasuk dalam nomor kursus yang diberikan. Hal itu bisa didapat melalui pemanggilan operasi getKursus dan getSiswa. Segera setelah informasi ini diperoleh, objek baru dari kelas Registrasi dibangun dan memanggil operasi init yang menetapkan siswa dan kursus untuk objek registrasi. Berikutnya perlu dibangun hubungan antara registrasi dan kursus karena navigasi di kedua arah diasumsikan. Untuk itu, dilakukan pemanggilan metode setRegistrasi. Kondisi tersebut, tidak perlu dilakukan pada objek-objek registrasi dan siswa, mengingat navigasi dari Siswa ke Registrasi tidak dimungkinkan (adanya tanda X pada diagram kelas). Diagram sekuens yang diperoleh tersaji pada Gambar 5.18.



Gambar 5.17 Diagram Kelas



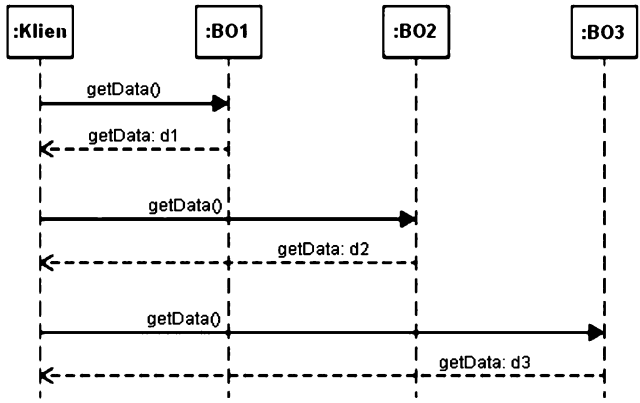
Gambar 5.18 Diagram Sekuens berdasar Diagram Kelas

5.5.2 Mendeskripsikan Pola Desain

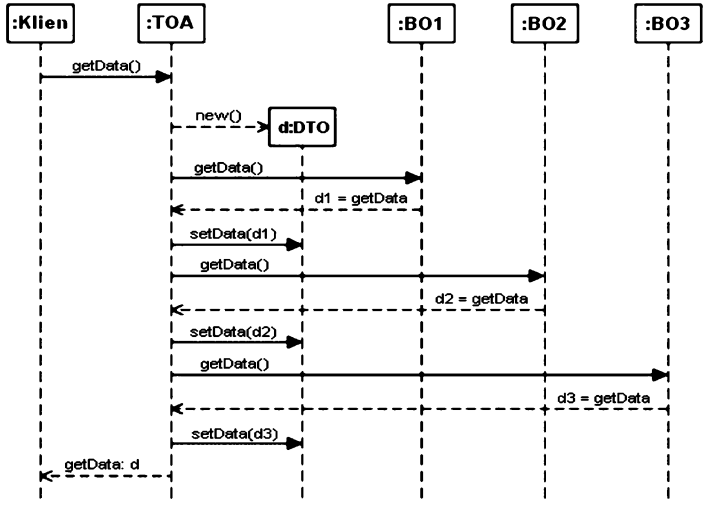
Diagram sekuens sering digunakan untuk menggambarkan pola desain. Pola desain menawarkan solusi untuk menggambarkan masalah berulang. Hal berikut ini tentang pola *Transfer Object Assembler* (TOA) yang menjelaskan apa yang terjadi ketika klien (pengguna) di lingkungan terdistribusi memerlukan informasi dari ke-3 (tiga) objek bisnis.

Gambar 5.19 memperlihatkan sebuah solusi ketika pengguna membutuhkan informasi tentang tiga objek bisnis untuk mengakses data yang diperlukan. Oleh karenanya, klien sangat ingin dihubungkan dengan objek bisnis tersebut — yang umumnya tidak diinginkan. Kita dapat menggunakan pola TOA untuk memecahkan masalah dependensi ini. Dalam pola ini, perakit menggabungkan data dari beberapa objek bisnis menjadi satu objek transfer yang kemudian ditransfer ke klien. Klien kemudian menerima data yang diperlukan dalam bentuk terenkapsulasi. Dalam kondisi nyata, pola diimplementasikan seperti yang dijelaskan dalam Gambar 5.20.

Dengan menggunakan getData(), klien meminta informasi yang diperlukan dari sebuah objek tipe TOA. Objek TOA membangun objek **d** dengan tipe *Data Transfer Object* (DTO) dan mengisinya dengan data dari tiga objek bisnis yang berbeda (BO1, BO2, BO3). Data dari sebuah objek bisnis dapat dikueri dengan menggunakan getData(). Objek **d** menawarkan metode setData yang mengizinkan memasukkan data. Akhirnya, objek tipe TOA mengembalikan **d** ke klien.



Gambar 5.19 Skenario aplikasi pola TOA

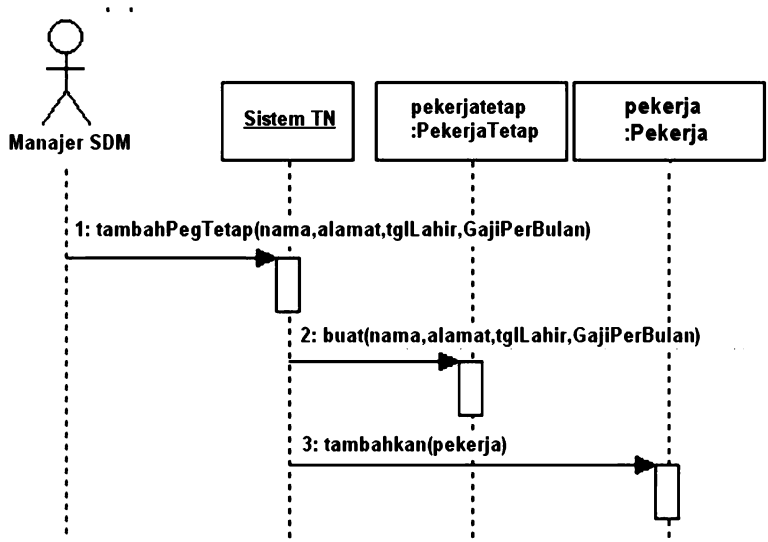


Gambar 5.20 Deskripsi pola TOA menggunakan Diagram Sekuens

5.6 Contoh Terapan

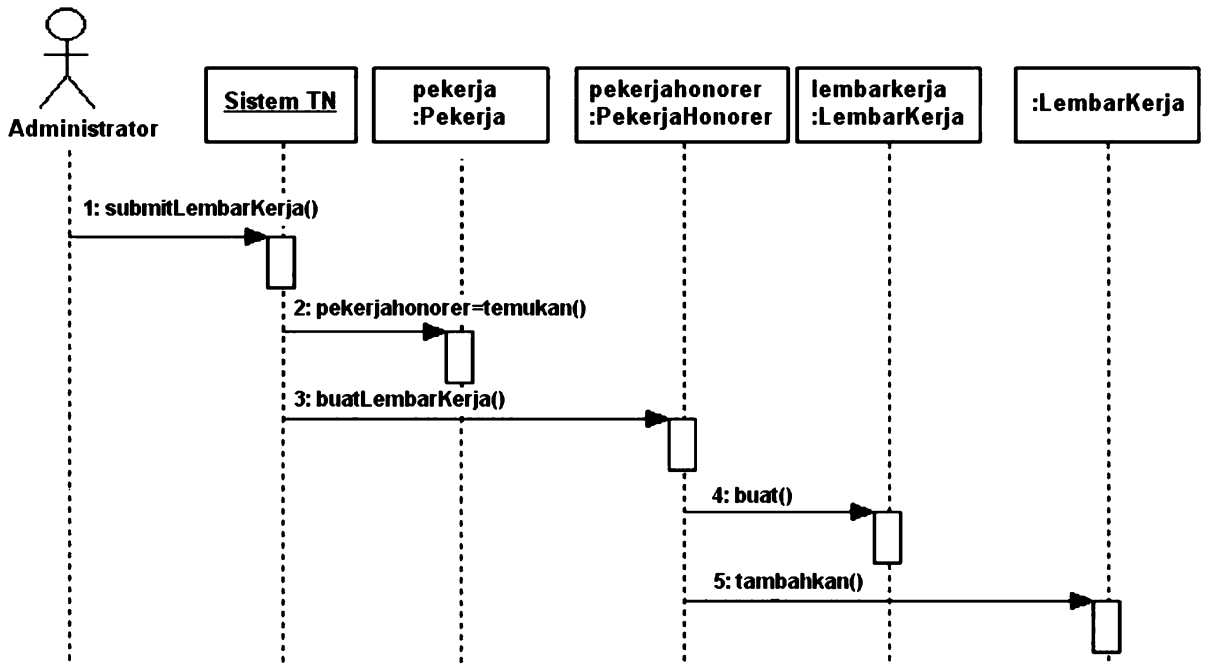
Berikut ini contoh terapan pembangunan diagram sekuens, sebagai langkah detail yang menunjukkan interaksi antar objek-objek dari perilaku sistem Manajemen Proyek Perusahaan TN (Bab 4) untuk contoh 1 (Gambar 5.21) dan 2 (Gambar 5.22). Sementara itu, contoh 3 merupakan DS sistem ATM sebuah Bank untuk kasus Kartu Invalid (Gambar 5.23).

1. Diagram Sekuens (DS) untuk operasi tambahPegTetap()



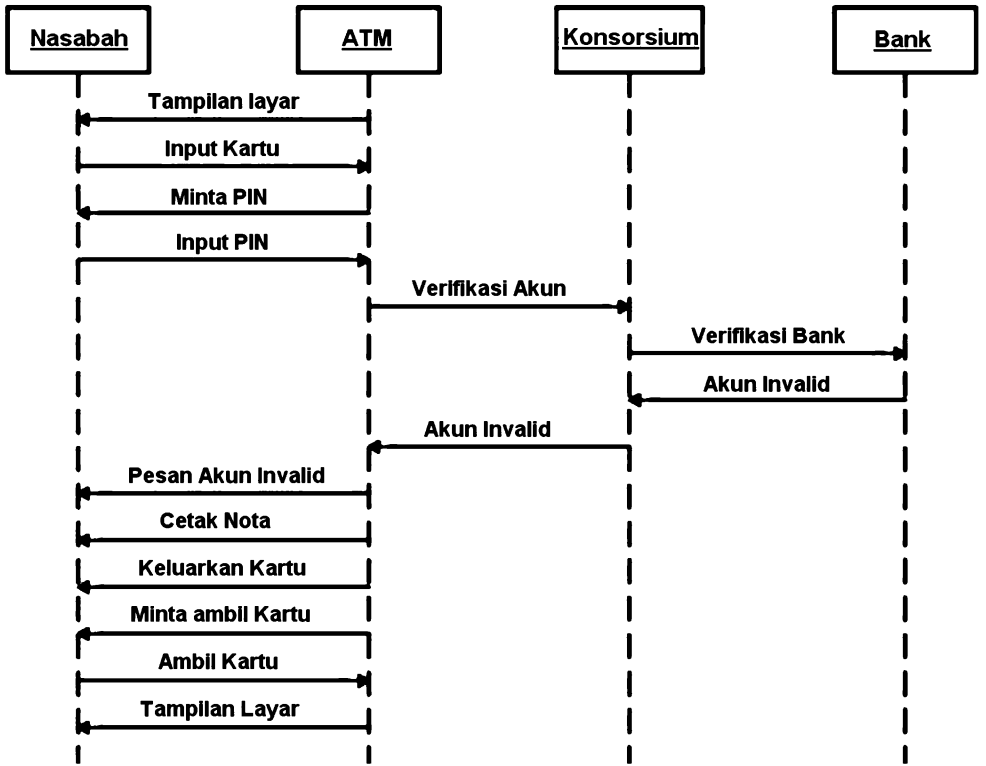
Gambar 5.21 DS operasi tambahPegTetap()

2. Diagram Sekuens (DS) untuk operasi submitLembarKerja()



Gambar 5.22 DS operasi submitLembarKerja()

3. DS Sistem ATM sebuah Bank untuk kasus Kartu Invalid



Gambar 5.23 DS Sistem ATM sebuah Bank untuk kasus Kartu Invalid

BAB 6
DIAGRAM AKTIVITAS

Diagram aktivitas memfokuskan pada pemodelan proses bisnis dari sistem atau prosedur pemrosesan sebuah sistem. Artefak diagram aktivitas berupa gambar-gambar visualisasi aliran kontrol dan data di antara berbagai langkah/aksi yang diperlukan dalam mengimplementasikan aktivitas-aktivitas sebuah sistem. Dalam bab ini, tidak semua contoh dalam pemodelan proses diberikan secara lengkap—beberapa model dibatasi dalam mengekstrak proses sehingga pada beberapa diagram aktivitas ada yang tidak memuat node awal dan akhir. Namun demikian, secara umum, diagram aktivitas yang lengkap harus memiliki node/titik awal dan akhir yang ditentukan dengan jelas.

6.1 Aktivitas-Aktivitas

Diagram aktivitas merupakan visualisasi perilaku pengguna dalam bentuk aktivitas-aktivitas ketika berinteraksi dengan sistem. Visualisasi perilaku/aktivitas tersebut merupakan gambaran operasional/pelaksanaan satu atau beberapa use case. Pada level konseptual, diagram aktivitas dapat memodelkan fungsi proses bisnis sistem, baik antarmitra bisnis maupun proses internal perusahaan/organisasi guna mencapai tujuan mereka. Sementara pada level detail/operasional, diagram aktivitas dapat menggambarkan perilaku operasi dalam bentuk instruksi individu. Dengan demikian, perilaku interaksi pengguna dengan sistem, dapat didefinisikan pada tingkat granularitas yang berbeda. Aktivitas dapat ditugaskan ke operasi kelas tetapi juga dapat bersifat otonom.

Konten sebuah aktivitas berupa grafik berarah, yang node-nodenya merepresentasikan komponen aktivitas, seperti aksi/tindakan; penyimpanan data; elemen kontrol dan aliran objek, yang bisa jadi sebagai jalur eksekusi aktivitas tersebut.

Aktivitas digambarkan sebagai persegi panjang dengan sudut bulat, sebagai representasi sebuah operasi yang bisa memiliki parameter. Ini ditunjukkan sebagai persegi panjang yang disusun tumpang-tindih di batas kegiatan. Untuk membuat diagram aktivitas lebih mudah dibaca, parameter input diposisikan di batas sebelah kiri atau atas aktivitas, dan parameter output di batas sebelah kanan atau bawah aktivitas sehingga aktivitas dapat dibaca dari kiri ke kanan dan/atau dari atas ke bawah. Nilai yang ditransfer ke aktivitas melalui parameter input tersedia untuk aksi yang terhubung ke parameter input dengan tepi yang diarahkan.

Dengan cara yang sama, parameter output menerima nilainya melalui tepi yang diarahkan dari aksi dalam aktivitas. Gambar 6.1 menunjukkan langkah-langkah yang diperlukan untuk melaksanakan aktivitas mengikuti ujian. Parameter input adalah nomor matrikulasi (noMat) dan ID program studi seorang siswa. Aksi daftar/registrasi, tulis/kerjakan ujian, dan menilai/mengoreksi dieksekusi dalam aktivitas ini. Hasil dari aktivitas tersebut berupa nilai. Diagram aktivitas (Gambar 6.1) tidak memperlihatkan siapa yang melakukan aksi mana. Untuk memungkinkan aksi dilakukan oleh aktor tertentu, diagram aktivitas menawarkan konsep partisi, yang akan dibahas pada seksi berikutnya.

Dalam sebuah aktivitas, berlaku prakondisi (kondisi sebelumnya) dan pascakondisi (kondisi sesudahnya) sebelum/sesudah aktivitas tersebut dieksekusi. Kata kunci «precondition» dan «postcondition» digunakan untuk mengidentifikasi kondisi masing-masing. Pada Gambar 6.1, siswa yang ingin mengikuti ujian harus terdaftar/registrasi, mengikuti ujian/tulis ujian, dan hasilnya harus diperiksa/dikoreksi/dinilai.

Gambar 6.1 Contoh sebuah aktivitas

6.2 Aksi-Aksi

Elemen dasar kegiatan/aktivitas adalah tindakan/aksi. Aksi digambarkan sebagai persegi panjang dengan sudut bulat, di mana nama aksi di tengah-tengahnya. Aksi dapat digunakan untuk menentukan perilaku yang ditetapkan pengguna. Dalam mendeskripsikan aksi tidak ada persyaratan bahasa khusus, bisa dengan bahasa alami atau bahasa pemrograman. Aksi memproses nilai input untuk menghasilkan nilai output. Gambar 6.1 menunjukkan aksi-aksi: mendaftar/registrasi, tulis/kerjakan ujian, dan periksa ujian/koreksi dari sebuah aktivitas ikut ujian.

Dalam lingkup aktivitas, sebuah aksi bersifat atomik (tidak dapat dipecah/diurai lagi), meskipun mungkin terdiri dari beberapa langkah individual. Mendaftar untuk ujian biasanya memerlukan beberapa langkah, seperti masuk ke sistem dan memilih kursus/ objek ujian/mata kuliah yang sesuai dan tanggal ujian. Meskipun demikian, pada Gambar 6.1, mendaftar sebagai aksi dari sebuah aktivitas, mengingat eksekusi pendaftaran dianggap sebagai langkah tunggal; adapun perincian internal langkah ini dianggap tidak menarik/penting sehingga aksi ini tidak dipecah/diurai.

Sebagai mana Gambar 6.1, aksi dan parameter terhubung satu sama lain melalui tepi yang terarah. Tepi ini mengekspresikan urutan aksi dieksekusi dari sebuah aktivitas. Di sini dibedakan antara tepi aliran kontrol dan tepi aliran objek. Tepi aliran kontrol menentukan urutan aksi, sedangkan tepi aliran objek dapat digunakan untuk bertukar data atau objek. Kondisi tersebut, memungkinkan dependensi data antara aksi sebelum dan sesudahnya dapat diekspresikan. Namun demikian, sebuah aksi hanya dapat dieksekusi jika semua aksi sebelumnya telah berhasil diselesaikan, semua penjaga (yang bertugas memfilter/menyeleksi) yang relevan mengevaluasi ke true dan semua parameter input memiliki nilai. Penjaga merupakan kondisi yang harus dipenuhi guna memungkinkan transisi dari satu aktivitas atau aksi ke aktivitas atau aksi yang lain. Mereka biasanya terjadi terkait dengan adanya cabang alternatif.

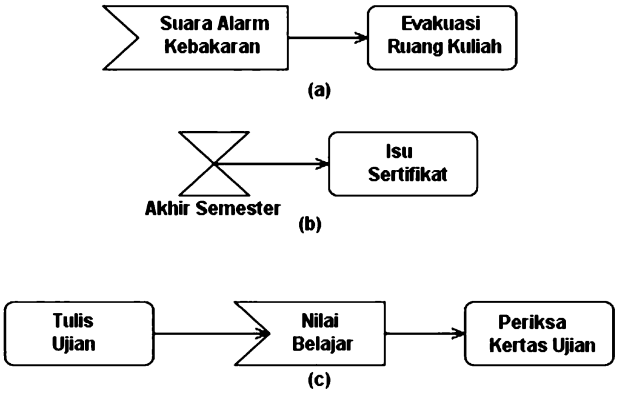
Terdapat dua kategori aksi standar, yaitu aksi berbasis kejadian (*event-based actions*) dan aksi panggilan perilaku (*call behaviour actions*). Seksi berikut ini mendeskripsikannya.

6.2.1 Aksi Berbasis Kejadian

Tipe aksi ini memungkinkan objek dan sinyal ditransmisikan ke objek penerima. Dengan aksi ini berbagai jenis kejadian dapat dibedakan. Kita dapat menggunakan aksi terima kejadian untuk memodelkan aksi yang menunggu terjadinya kejadian tertentu. Elemen notasi untuk aksi terima kejadian berbentuk "segi lima cekung"—persegi panjang dengan ujung yang menunjuk ke dalam dari kiri (Gambar 6.2(a) dan 6.2(c)). Jika kejadian merupakan kejadian berbasis waktu maka kita dapat menggunakan aksi terima kejadian (waktu) yang dinotasikan berbentuk *hourglass* (Gambar 6.2(b)).

Aksi terima kejadian berbasis waktu tidak selalu memiliki tepi masuk. Semisal mereka tidak memiliki tepi masuk maka mereka mulai ketika kejadian yang sesuai terjadi. Mereka tetap aktif dan dapat menerima sinyal sampai aktivitas yang memuat mereka berakhir.

Gambar 6.2 menunjukkan tiga contoh aksi terima kejadian (waktu): setiap kali alarm kebakaran dipicu, ruang kuliah harus dievakuasi (Gambar 6.2(a)); pada akhir semester, sertifikat diterbitkan (Gambar 6.2(b)); ketika siswa telah mengikuti ujian (tulis ujian), siswa menunggu nilai dan berikutnya memeriksa naskah ujian saat menerima nilai (Gambar 6.2(c)).



Gambar 6.2 Contoh aksi terima kejadian (a+c) dan aksi terima kejadian (waktu [b])

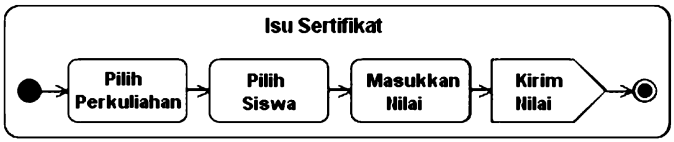
Untuk kirim sinyal, aksi kirim sinyal dapat digunakan. Aksi tersebut dinotasikan dengan "segi lima cembung"—persegi panjang dengan ujung yang menonjol ke kanan. Aksi kirim nilai pada Gambar 6.3(b) sebagai contoh aksi sinyal kirim.

6.2.2 Aksi Panggilan Perilaku

Aksi dapat memanggil aktivitas itu sendiri. Aksi ini disebut sebagai aksi panggilan perilaku dan dinotasikan dengan simbol garpu terbalik. Simbol garpu ini menunjukkan sebuah hierarki.

Hal tersebut menunjukkan bahwa eksekusi aksi ini memulai kegiatan lain yang membagi sistem menjadi berbagai bagian. Gambar 6.3(a) memperlihatkan contoh aksi panggilan perilaku. Dalam diagram tersebut, aksi **isu (cetak) sertifikat** dalam aktivitas **mengorganisasi ujian** mengacu pada aktivitas yang menspesifikasikan **isu sertifikat** secara lebih terperinci. Dalam konteks aktivitas **mengorganisasi ujian**, langkah-langkah internal yang mengarah pada isu sertifikat tidak relevan. Oleh karenanya, **isu sertifikat** dipandang sebagai sebuah unit atom, meskipun melibatkan proses dengan aksi berganda. Gambar 6.3(b) menunjukkan detail aktivitas **isu sertifikat** yang disebut dengan parameter input nilai (grade).

(a)



(b)

Gambar 6.3 Contoh aksi panggilan perilaku dan aksi pengiriman sinyal

Sebuah aksi dapat memicu panggilan operasi. Jenis aksi ini disebut sebagai aksi panggilan operasi. Ini direpresentasikan dalam bentuk persegi panjang dengan tepi bulat. Apabila nama operasi tidak sesuai dengan nama aksi, nama operasi dapat ditetapkan di bawah nama aksi dalam formulir (NamaKelas::NamaOperasi).

6.3 Aliran Kontrol

Diagram aktivitas terdiri dari aktivitas dan aksi lain yang terhubung satu sama lain melalui tepi (*edges*). Jika kita melihat grafik aktivitas, struktur statis tidak menunjukkan dengan jelas bagaimana eksekusi bekerja. Untuk mengintegrasikan aspek perilaku dinamis ke dalam diagram, kita perlu semantik eksekusi, yang berarti bahwa kita harus menentukan dengan tepat bagaimana sebuah diagram aktivitas dijalankan.

Konsep token merupakan dasar bagi semantik eksekusi diagram aktivitas. Token adalah mekanisme koordinasi virtual yang menjelaskan eksekusi dengan tepat. Dalam konteks tersebut, virtual berarti token bukan komponen fisik diagram. Mereka adalah mekanisme yang memberikan aksi izin eksekusi.

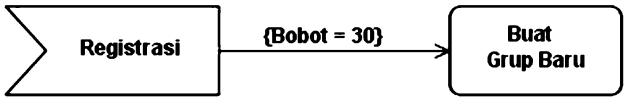
Seumpama aksi menerima token, aksi bersifat aktif dan dapat dieksekusi. Setelah aksi berakhir, ia meneruskan token ke node berikutnya melalui tepi dan memicu eksekusi aksi ini. Setelah aksi ini berakhir, ia meneruskan token ke tepi keluar atau mempertahankannya sampai kondisi tertentu terpenuhi. Lewatnya token dapat dicegah oleh penjaga yang mengevaluasi ke status false. Penjaga ditentukan dalam kurung siku. Dalam contoh pada Gambar 6.4, ujian hanya diadakan jika siswa telah terdaftar.

Gambar 6.4 Contoh sebuah penjaga

Aksi hanya dapat dijalankan jika token ada di semua tepi yang masuk. Jika aksi memiliki beberapa tepi keluar, token ditawarkan ke semua target node tepi ini sehingga menyebabkan pemisahan menjadi beberapa jalur eksekusi yang independen. Node khusus juga tersedia sebagai alternatif untuk memodelkan konkurensi ini. Ketika aksi dijalankan, biasanya satu token dari setiap tepi

yang masuk dikonsumsi atau bobot (weight) dapat ditempatkan di tepi untuk memungkinkan sejumlah token dikonsumsi di tepi itu dengan satu eksekusi. Bobot tepi ditentukan dalam kurung keriting dengan kata kunci **weight**.

Gambar 6.5 memberikan contoh penggunaan **weight** (bobot). Apabila sinyal register diterima 30 kali artinya setidaknya 30 siswa telah mendaftar. Dengan demikian, 30 token ditawarkan ke aksi berikutnya maka aksi berikutnya ini dijalankan, mengonsumsi 30 token, dan kelompok baru dibuat.



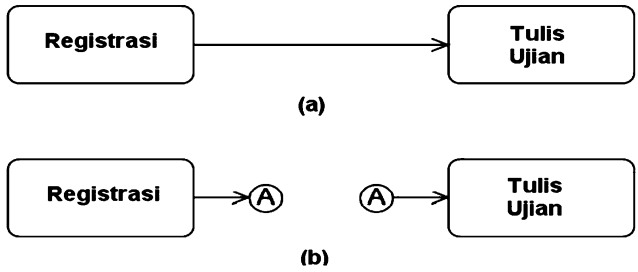
Gambar 6.5 Contoh bobot sebuah tepi

Semisal dua aksi yang akan terhubung satu sama lain melalui tepi jauh terpisah dalam diagram, kita dapat menggunakan konektor untuk membuat diagram lebih jelas. Untuk itu, penggambaran tepi sebagai garis berkelanjutan dari satu aksi ke aksi lainnya tidak diperlukan dan sebagai gantinya konektor bertindak sebagai penanda kelanjutan yang selaras dengan penanda kelanjutan dalam diagram sekuens (Bab 5). Konektor digambarkan sebagai lingkaran kecil yang berisi nama konektor. Setiap konektor harus muncul dua kali dalam sebuah aktivitas, yaitu sekali sebagai tepi masuk dan sekali dengan tepi keluar. Gambar 6.6 memodelkan relasi antara dua aksi, sekali dengan tanpa konektor (Gambar 6.6(a)) dan sekali dengan konektor (Gambar 6.6(b)).

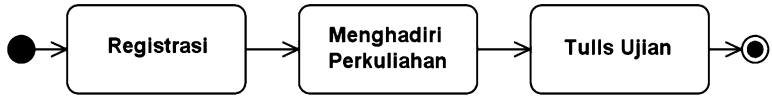
Diagram aktivitas menawarkan node khusus untuk mengontrol aliran kontrol. Node ini disebut node kontrol. Node awal menunjukkan eksekusi aktivitas dimulai. Ia tidak memiliki tepi yang masuk, tetapi memiliki setidaknya satu tepi keluar dan dinotasikan sebagai lingkaran hitam padat. Segera setelah aktivitas menjadi aktif, token disediakan di semua tepi keluar dari node awal.

Dengan demikian, aktivitas dimulai. Gambar 6.7 menunjukkan contoh node awal. Diagram juga berisi node akhir aktivitas yang mewakili akhir aktivitas.

Beberapa node awal juga diizinkan untuk setiap aktivitas. Kondisi tersebut mengekspresikan adanya konkurensi yang berarti bahwa beberapa jalur eksekusi dapat aktif secara bersamaan. Apabila sebuah aktivitas dengan beberapa node awal dipanggil, tepi keluar dari semua node awal disediakan dengan token secara bersamaan. Contoh pada Gambar 6.8 menunjukkan dua subpath serentak dari aktivitas **melakukan perkuliahan**. Jika aktivitas **melakukan perkuliahan** diaktifkan, token ditempatkan di masing-masing dari dua node awal. Dengan demikian, kedua subpath diaktifkan. Satu subpath berkaitan dengan aksi siswa (mulai **registrasi**) dan subpath lainnya mengacu pada aksi yang dilakukan oleh seorang dosen (mulai **persiapan perkuliahan**). Dalam aksi **kerjakan/tulis ujian**, kedua jalur digabungkan. Sebuah token harus ada di kedua tepi masuk agar aksi **kerjakan/tulis ujian** dapat dieksekusi.



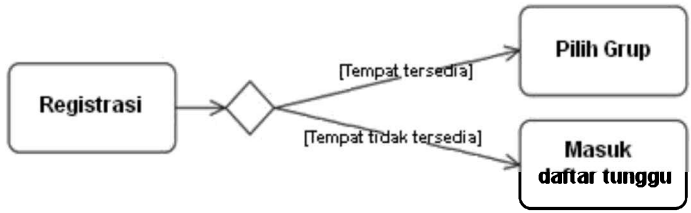
Gambar 6.6 Contoh sebuah konektor



Gambar 6.7 Contoh inisial (awal) node

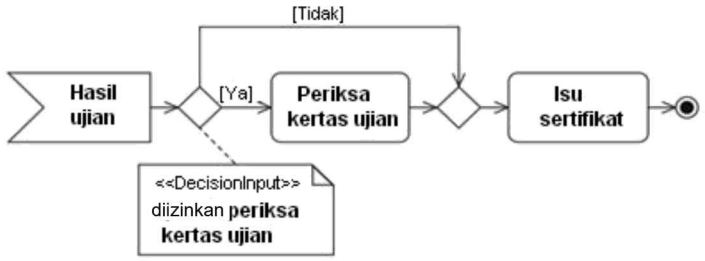
Gambar 6.8 Contoh inisial (awal) node berganda

Dalam diagram aktivitas, dapat juga dimodelkan penggunaan node keputusan dengan cabang alternatif. Node keputusan bertindak sebagai titik pengalihan untuk token dan berhubungan dengan pernyataan if dalam bahasa pemrograman konvensional. Node keputusan digambarkan sebagai berlian dengan satu tepi masuk dan beberapa tepi keluar. Tepi keluar memiliki penjaga (juga disebut sebagai kondisi). Kondisi tersebut tidak boleh tumpang-tindih yang berarti bahwa sistem harus dapat dengan jelas memutuskan tepi keluar mana yang harus diambil token dalam situasi tertentu. Misalnya, memiliki satu tepi keluar dengan kondisi [x > 1] dan tepi keluar lainnya dengan kondisi [x < 3] tidak diizinkan karena tidak akan ada pilihan unik (tunggal) tepi yang harus diambil token jika x = 2 berlaku. Apabila ada token pada node keputusan, sistem harus dapat memutuskan dengan jelas, berdasarkan konteks saat ini (misalnya, tergantung pada nilai variabel), jalur mana yang diambil token untuk keluar dari node keputusan. Gambar 6.9 menunjukkan contoh node keputusan. Dalam hal aksi Registrasi dieksekusi, akan diikuti oleh aksi Pilih grup (jika masih ada tempat tersedia). Semisal hal tersebut tidak terjadi, aksi masuk pada daftar tunggu dieksekusi.



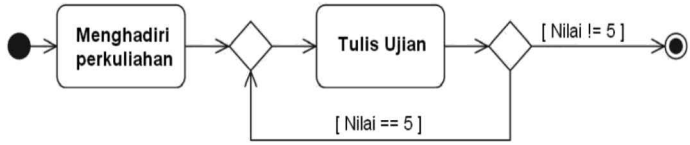
Gambar 6.9 Contoh node keputusan

Perilaku keputusan untuk sebuah node keputusan dapat ditentukan, hal ini diperlukan guna mengevaluasi penjaga. Ini dimungkinkan untuk menghindari situasi, di mana kalkulasi yang sama harus dilakukan beberapa kali untuk penjaga yang berbeda, sebagai hasil dari kalkulasi yang dapat diakses setiap penjaga. Namun, perilaku ini tidak boleh mengakibatkan efek samping, yang berarti bahwa pelaksanaan perilaku yang didefinisikan dalam node keputusan tidak boleh mengubah nilai objek dan variabel. Perilaku keputusan dilekatkan pada node keputusan sebagai komentar dengan label «decisionInput». Gambar 6.10 menunjukkan contohnya. Segera setelah hasil ujian diketahui, keputusan diambil di departemen pusat mengenai apakah akan menawarkan kepada siswa kesempatan untuk memeriksa naskah ujian mereka yang dikoreksi. Jika keputusannya adalah positif, siswa diperbolehkan untuk memeriksa naskah ujian mereka. Sertifikat mereka dikeluarkan setelahnya. Jika keputusannya negatif, sertifikat segera dikeluarkan.



Gambar 6.10 Contoh perilaku keputusan

Semisal diinginkan subjalur (subpath) alternatif diarahkan bersama kembali, penggunaan node gabungan dapat dilakukan. Node ini dinotasikan sebagai berlian tetapi dengan beberapa tepi yang masuk dan hanya satu tepi keluar. Pada khususnya, sebuah token hanya dapat ada di satu tepi yang masuk paling banyak. Penggunaan node keputusan dan gabungan dapat dimodelkan langkah-langkah eksekusi yang diulang yang disebut sebagai loop (Gambar 6.11).



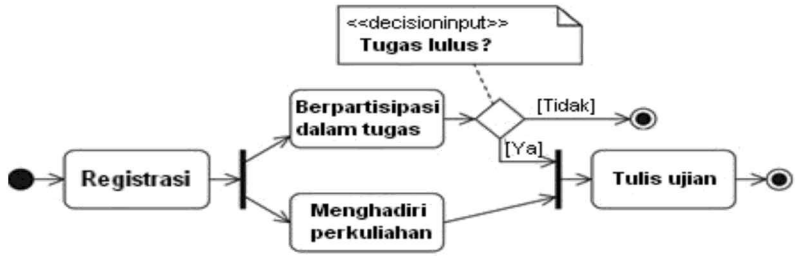
Gambar 6.11 Contoh sebuah Loop

Untuk node keputusan dan penggabungan, hanya salah satu jalur yang bisa aktif. Sebagaimana yang telah disebutkan, penggunaan beberapa node awal untuk memodelkan konkurensi pada awal aktivitas dapat dilakukan. Jika jalur eksekusi dipisahkan ke beberapa jalur eksekusi aktif secara bersamaan di kemudian hari, bentuk realisasinya dapat menggunakan **node paralelisasi (*fork*)**. Node paralelisasi digambarkan sebagai bilah hitam dengan satu tepi masuk dan beberapa tepi keluar. Gambar 6.12 menunjukkan contoh **node paralelisasi**. Setelah siswa mendaftar untuk kuliah, siswa menghadiri perkuliahan dan berpartisipasi dalam tugas secara bersamaan. Siswa hanya dapat mengerjakan ujian ketika kuliah dan tugas telah selesai. Penggabungan subpath secara bersamaan disebut **node sinkronisasi (*join*)**. Node ini adalah mitra dari node paralelisasi, dan digambarkan sebagai bar hitam dengan beberapa tepi yang masuk tetapi hanya satu tepi keluar. Segera setelah token hadir di semua tepi yang masuk, dan segera setelah semua aksi sebelumnya dieksekusi, semua token yang masuk digabungkan menjadi satu token yang diteruskan ke tepi keluar.

Gambar 6.12 Contoh penggunaan paralelisasi dan sinkronisasi

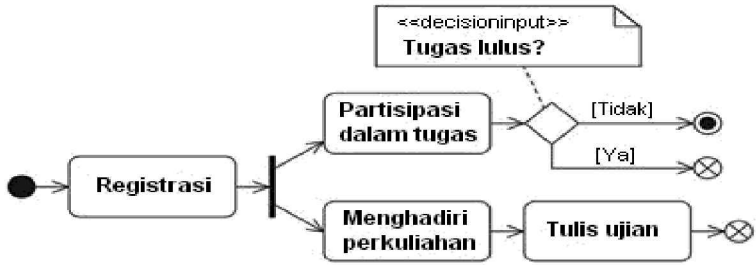
Untuk mengekspresikan akhir aktivitas, diagram aktivitas menawarkan node untuk tujuan ini, yaitu node akhir aktivitas. Node ini digambarkan sebagai lingkaran kecil berisi lingkaran padat dan sering disebut sebagai "**mata banteng**". Apabila token ada di tepi masuk dari node akhir aktivitas, seluruh aktivitas dihentikan—yaitu semua aksi aktif dari aktivitas ini dihentikan. Hal ini termasuk juga subpath konkurensi aktif dan dengan demikian semua token dalam aktivitas dihapus. Pengecualian untuk aturan ini adalah data token yang sudah ada pada parameter output aktivitas. Jika sebuah diagram berisi beberapa node akhir aktivitas, yang pertama dicapai selama eksekusi adalah mengakhiri seluruh aktivitas. Sebagai contoh, partisipasi dalam perkuliahan diakhiri jika (salah satu) tugas belum diselesaikan atau ujian telah diambil (Gambar 6.13).

Penggabungan beberapa node akhir aktivitas ke dalam satu aktivitas node akhir dengan beberapa tepi masuk juga dapat dilakukan. Segera setelah satu token mencapai node akhir aktivitas melalui tepi masuk, seluruh aktivitas berakhir.



Gambar 6.13 Contoh node akhir aktivitas berganda dalam sebuah diagram aktivitas

Ketika diinginkan hanya satu jalur eksekusi diakhiri, sementara jalur eksekusi aktif lainnya dibiarkan tidak terpengaruh maka **node akhir aliran** harus digunakan. Node ini hanya menghapus token yang mengalir ke dalamnya secara langsung sehingga hanya mengakhiri jalur masing-masing. Semua token aktivitas lainnya tetap tidak terpengaruh dan dimungkinkan tetap berlangsung/ hidup. Node akhir aliran dinotasikan dengan lingkaran kecil yang berisi **X** dan hanya memiliki tepi yang masuk.



Gambar 6.14 Contoh akhir aliran node dan akhir aktivitas node

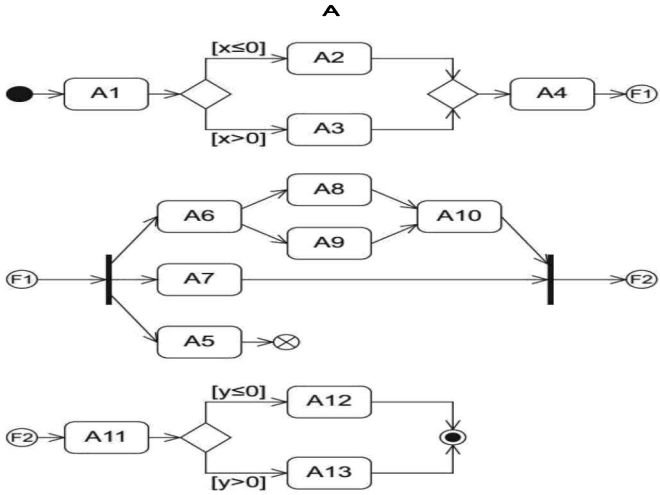
Gambar 6.14 merupakan contoh hasil modifikasi Gambar 6.13. Partisipasi yang berhasil dalam penyelesaian tugas dan kehadiran secara simultan dari kuliah terkait, dikombinasikan dengan mengikuti ujian adalah dua jalur eksekusi independen yang masing-masing berakhir dengan **node akhir aliran**. Namun, kegagalan dalam tugas akan mengakhiri seluruh kegiatan sehingga siswa yang memiliki nilai tugas negatif, siswa tidak dapat lagi mengikuti ujian akhir.

Gambar 6.15 menunjukkan semantik eksekusi **node kontrol individu** yang menerapkan konsep token. Jika aktivitas A diaktifkan, semua tepi keluar dari semua node awal menetapkan token. Dalam hal ini, aksi A1 menerima token dan memulai eksekusi. Setelah A1 berhasil diselesaikan, A1 meneruskan token ke node keputusan.

Tergantung pada nilai variabel x, node keputusan meneruskan token ke A2 jika nilai [x ≤ 0] benar atau ke A3 jika nilai [x > 0] adalah benar. Node penggabungan berikutnya meneruskan setiap token yang diterimanya ke node berikutnya. Dengan demikian, setelah eksekusi A2 atau A3, aksi A4 diaktifkan. Node paralelisasi berikutnya menduplikasi token untuk semua tepi keluar, sehingga membuat tiga token dan mengaktifkan A5, A6, dan A7. Tiga jalur eksekusi berikut diambil secara bersamaan:

1. Satu token mengaktifkan A5. Segera setelah eksekusi A5 berakhir, token lolos ke node akhir aliran yang kemudian mengakhiri jalur eksekusi ini. Tidak ada jalur eksekusi lain yang terpengaruh.
2. Token lebih lanjut mengaktifkan A6. Setelah eksekusi A6, semua yang keluar dari tepi A6 ditetapkan token A8 dan A9, kemudian dieksekusi secara bersamaan. A10 hanya dapat dieksekusi ketika token masuk di semua tepi masuk A10, yaitu ketika A8 dan A9 telah selesai. Dari aliran token dapat dilihat bahwa beberapa tepi keluar dari aksi setara dengan node paralelisasi. Apabila beberapa tepi menjadi node aksi, semua jalur eksekusi yang masuk harus selesai sebelum pelaksanaan aksi ini. Perilaku ini bisa juga dimodelkan menggunakan node sinkronisasi sebagai alternatif.

3. Token ketiga mengaktifkan A7



Gambar 6.15 Contoh konsep Token

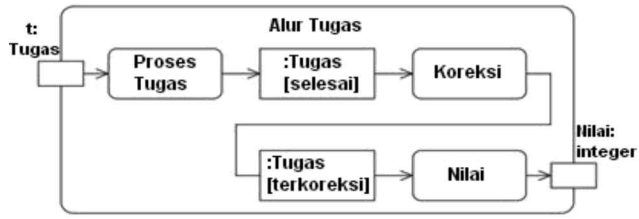
Ketika A7 dan A10 keduanya telah berhasil dieksekusi, dengan demikian token di kedua tepi node sinkronisasi ada yang masuk, token ini digabungkan menjadi satu token dan A11 diaktifkan. Tergantung pada nilai variabel y, node keputusan meneruskan token ke A12 jika nilai [y ≤ 0] benar atau ke A13 jika nilai [y > 0] adalah benar. Dalam kedua kasus, setelah eksekusi aktivitas masing-masing, token memasuki node akhir aktivitas. Ketika token mencapai node akhir aktivitas, seluruh aktivitas berakhir. Dan selanjutnya token yang tersisa ditarik dari semua aksi, sehingga misalnya, eksekusi A5 dihentikan meskipun aksi ini belum berakhir pada saat ini.

6.4 Aliran Objek-Objek

Pembahasan sampai saat ini, berfokus pada aliran kontrol, berkonsentrasi pada urutan aksi yang logis. Konsep token yang disampaikan digunakan secara eksklusif untuk mengontrol token itu sendiri. Untuk tujuan penyederhanaan, sejauh ini hanya disebut sebagai **token**. Namun, mungkin atau bahkan sangat mungkin, aksi itu bertukar data menggunakan token data. Sama seperti **token kontrol**, ini tidak pernah dideskripsikan dalam diagram dan juga digunakan hanya untuk menggambarkan semantik eksekusi diagram aktivitas. **Token data** secara implisit juga **mengontrol token** karena pertukaran token ini memengaruhi aliran aktivitas. Data dapat menjadi hasil dari aksi dan juga dapat berfungsi sebagai input untuk aksi berikutnya. Akan tetapi, data juga dapat diterima melalui parameter input aktivitas dan diteruskan ke parameter output, seperti yang telah dijelaskan sebelumnya (Gambar 6.1). Parameter input biasanya hanya dibaca satu kali pada awal aktivitas dan parameter output ditulis sekali di akhir aktivitas. Apabila diizinkan, parameter dibaca dan ditulis terus selama pelaksanaan aktivitas, parameter input atau output dapat dilabeli dengan **kata kunci {stream}**. Gambar 6.16 memperlihatkan contoh aliran untuk aktivitas atau aksi parameter input dan output. **Parameter streaming** untuk aksi dapat diuraikan dengan persegi panjang yang berisi.

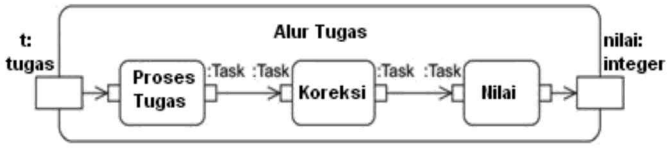
Gambar 6.16 Contoh Streams

Ketika aktivitas berakhir, parameter output apa pun yang tidak memiliki token ditetapkan token bernilai null. Dalam suatu aktivitas, penggunaan node objek secara eksplisit mewakili pertukaran data. **Node objek** dapat digambarkan dengan berbagai cara. Node objek ditampilkan sebagai node terpisah, seperti dalam diagram objek atau dilampirkan ke aksi secara langsung sebagai **pin input** atau **output**. Gambar 6.17 memberikan **contoh node objek sebagai node independen**. Ini ditambahkan sebagai persegi panjang antara aksi yang menyampaikan data dan aksi yang mengonsumsi data. Persegi panjang berisi nama objek yang diwakilinya. Secara opsional tipe objek juga dapat ditentukan. Status/eksistensi objek dapat ditetapkan dalam kurung siku.



Gambar 6.17 Contoh sebuah Node Objek

Notasi pin untuk aksi berhubungan dengan notasi parameter sebuah aktivitas. Ini digunakan dengan cara yang sama untuk mewakili objek yang berfungsi sebagai aksi input dan output. Persegi kecil ditentukan pada awal atau akhir tepi di batas yang sesuai. Pin dapat dinotasikan dengan informasi yang sama dengan yang digunakan ketika secara eksplisit mewakili node objek sebagai persegi panjang. Gambar 6.18 memperlihatkan contohnya.



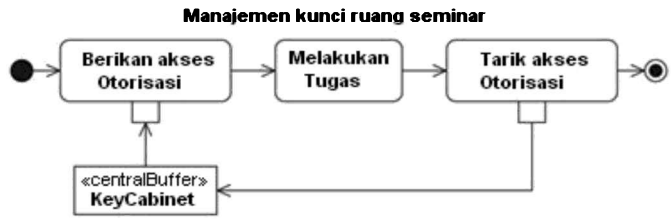
Gambar 6.18 Contoh sebuah Node Objek dalam notasi Pin

Diagram aktivitas menawarkan node objek khusus untuk menyimpan dan meneruskan token data, yaitu penyangga pusat serta penyimpanan data. Penyangga pusat adalah node objek khusus yang mengelola aliran data antara beberapa sumber dan beberapa penerima. Ia menerima token data masuk dari node objek dan meneruskannya ke node objek lainnya. Berbeda dengan aktivitas pin dan parameter, penyangga pusat tidak terikat dengan aksi atau aktivitas. Ketika token data dibaca dari penyangga pusat, kondisi tersebut dihapus dan tidak dapat dikonsumsi lagi. Gambar 6.19 menunjukkan contoh penggunaan penyangga pusat.

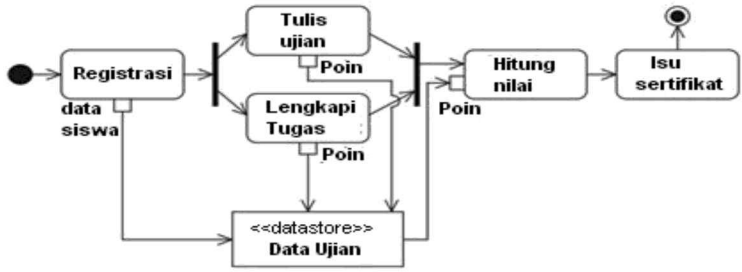
Untuk menjalankan aksi **berikan otorisasi akses**, kunci harus diambil dari ***KeyCabinet***. Si kunci kemudian tidak lagi berada di ***KeyCabinet*** sampai dikembalikan dalam aksi **tarik otorisasi akses**. Di penyimpanan data (**data store**), semua token data yang mengalir ke penyimpanan data disimpan secara permanen yang berarti bahwa mereka disalin sebelum mereka meninggalkan penyimpanan data lagi. Konten penyimpanan data di tepi keluar yang mengarah dari penyimpanan data dapat ditentukan. Kueri ini melekat pada tepi keluar menggunakan simbol catatan. Oleh karena itu, penyimpanan data dapat memodelkan fungsionalitas basis data.

Gambar 6.20 memperlihatkan contoh penggunaan penyimpanan data. Kinerja peserta kursus dikelola di penyimpanan data DataUjian. Data ini mencakup penilaian tugas dari penugasan dan hasil ujian yang diperlukan untuk menghitung nilai keseluruhan. **Penyangga pusat mewakili memori sementara, sedangkan penyimpanan data mewakili memori permanen**. Untuk yang pertama, informasi dapat hanya digunakan sekali artinya setelah dibaca dari penyangga pusat, diteruskan, kemudian hilang. Sementara itu, penyimpanan data, informasi dapat

digunakan sesering yang diperlukan, asalkan telah disimpan sekali di penyimpanan data.



Gambar 6.19 Contoh sebuah Sentral Buffer



Gambar 6.20 Contoh sebuah Data Store

6.5 Partisi

Dengan partisi, node dan tepi aktivitas dapat dikelompokkan berdasarkan properti/sifat umum. Dalam proses bisnis, partisi dapat digunakan untuk mengelompokkan semua aksi yang bertanggung jawab terhadap pelaksanaan entitas tertentu. Tidak ada aturan ketat mengenai kriteria pengelompokan yang dapat digunakan. Umumnya, partisi mencerminkan unit organisasi atau peran yang bertanggung jawab atas pelaksanaan aksi dalam partisi. Partisi dapat ditetapkan pada tingkat detail yang berbeda hingga tingkat kelas individu. Partisi dapat tumpang-tindih dan bersarang dengan cara apa pun yang diperlukan. Mereka tidak mengubah arti dari semantik eksekusi diagram aktivitas yang didefinisikan oleh token. Ini berarti bahwa partisi tidak memengaruhi aliran token, tetapi hanya mewakili tampilan logika komponennya. Partisi

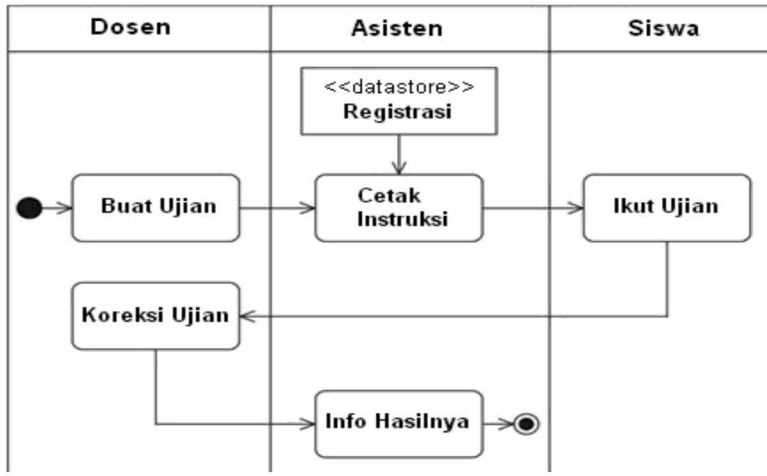
membuat diagram lebih jelas, memungkinkan area tanggung jawab dapat dilihat dengan cepat sehingga memperkenalkan informasi lebih terperinci ke dalam model.

Partisi dapat digambarkan, baik secara grafis atau dalam bentuk tekstual. Dalam bentuk grafis, partisi ditempatkan di atas diagram aktivitas sebagai persegi panjang "terbuka". Semua elemen yang terletak dalam persegi panjang "terbuka" termasuk dalam kelompok umum. Nama partisi ditentukan pada salah satu ujung persegi panjang. Karena penampilannya, partisi juga disebut sebagai **swimlanes**. Gambar 6.21 menunjukkan contoh penggunaan partisi dalam diagram aktivitas yang mencontohkan pelaksanaan ujian. Pihak-pihak (aktor-aktor) yang terlibat (berperan) adalah seorang mahasiswa, asisten, dan seorang dosen. Penggunaan partisi memungkinkan masing-masing aktor ini memiliki aksi apa yang harus dilakukan.

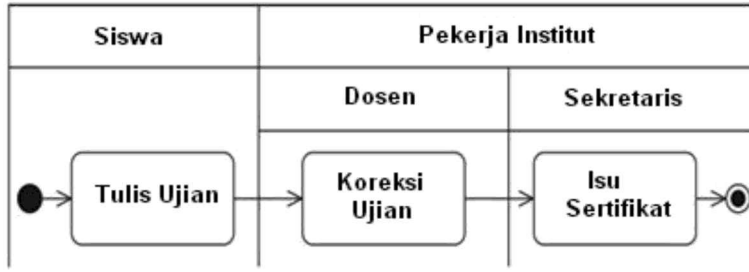
Partisi itu sendiri dapat dibagi menjadi beberapa subpartisi. Gambar 6.22 menunjukkan contoh ini, dengan Dosen dan Sekretaris terlibat dalam pelaksanaan ujian. Sebagaimana Gambar 6.23, partisi juga dapat multidimensi. Dalam contoh ini, dimodelkan korespondensi antar Dosen dari Perguruan Tinggi PT A dan PT B. Dosen dari PT B menulis surat kepada Dosen PT A. Dosen memberikan surat kepada Sekretaris, yang selanjutnya membawa surat itu ke kantor pos. Sekretaris di PT A mengambil surat dari kotak surat segera setelah surat tiba dan mengirimkannya kepada Dosen PT A, yang kemudian membacanya. Ini menunjukkan bagaimana kita membutuhkan partisi multidimensi ketika berbagai kelompok aktor dapat muncul dalam berbagai bentuk.

Aksi ke partisi juga dapat ditetapkan atau—dalam kasus partisi multidimensi—ke sekumpulan partisi dalam bentuk teks. Dalam situasi ini, partisi ditentukan dalam tanda kurung di atas nama aksi. Kalau aksi milik beberapa partisi, partisi ini

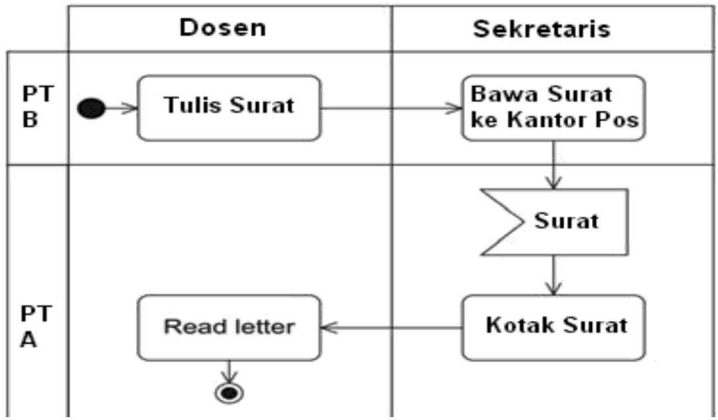
dicantumkan dan dipisahkan oleh koma, misalnya (Partisi 1, Partisi 2). Ketika menentukan subpartisi, penggunaan titik dua—(Partisi::Subpartisi)—untuk mengekspresikan partisi hierarkis. Diagram aktivitas dari Gambar 6.24 menunjukkan contoh dari Gambar 6.23 dalam notasi ini.



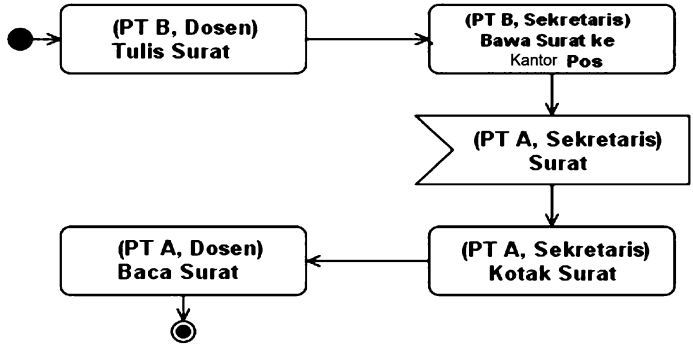
Gambar 6.21 Contoh Partisi satu Dimensi



Gambar 6.22 Contoh Subpartisi



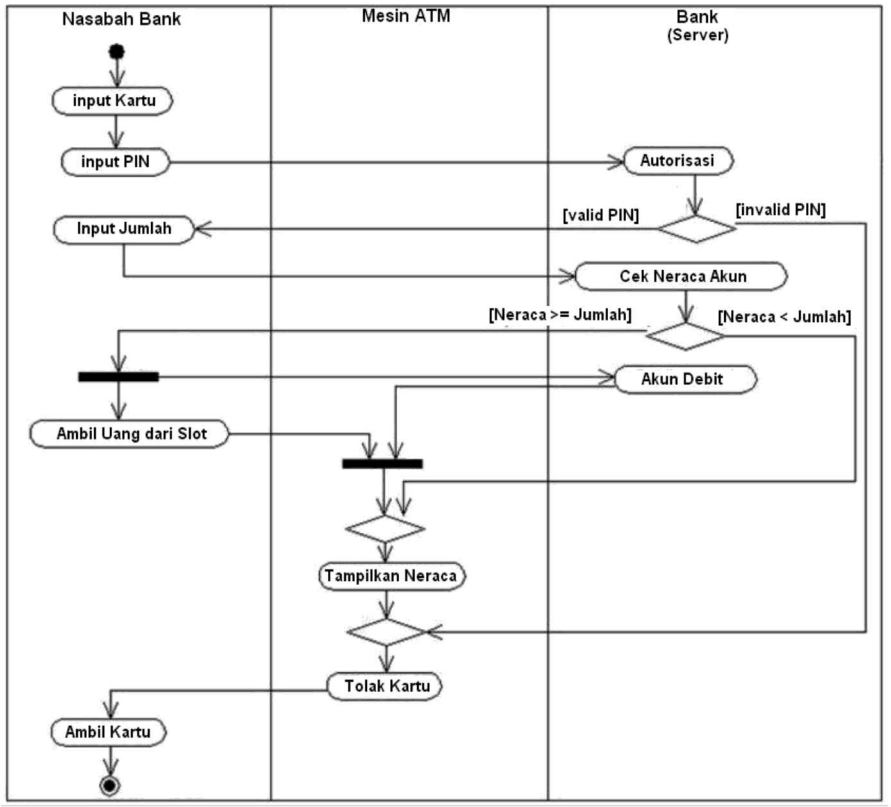
Gambar 6.23 Contoh Partisi Multidimensi



Gambar 6.24 Contoh Partisi Multidimensi dengan notasi Tekstual

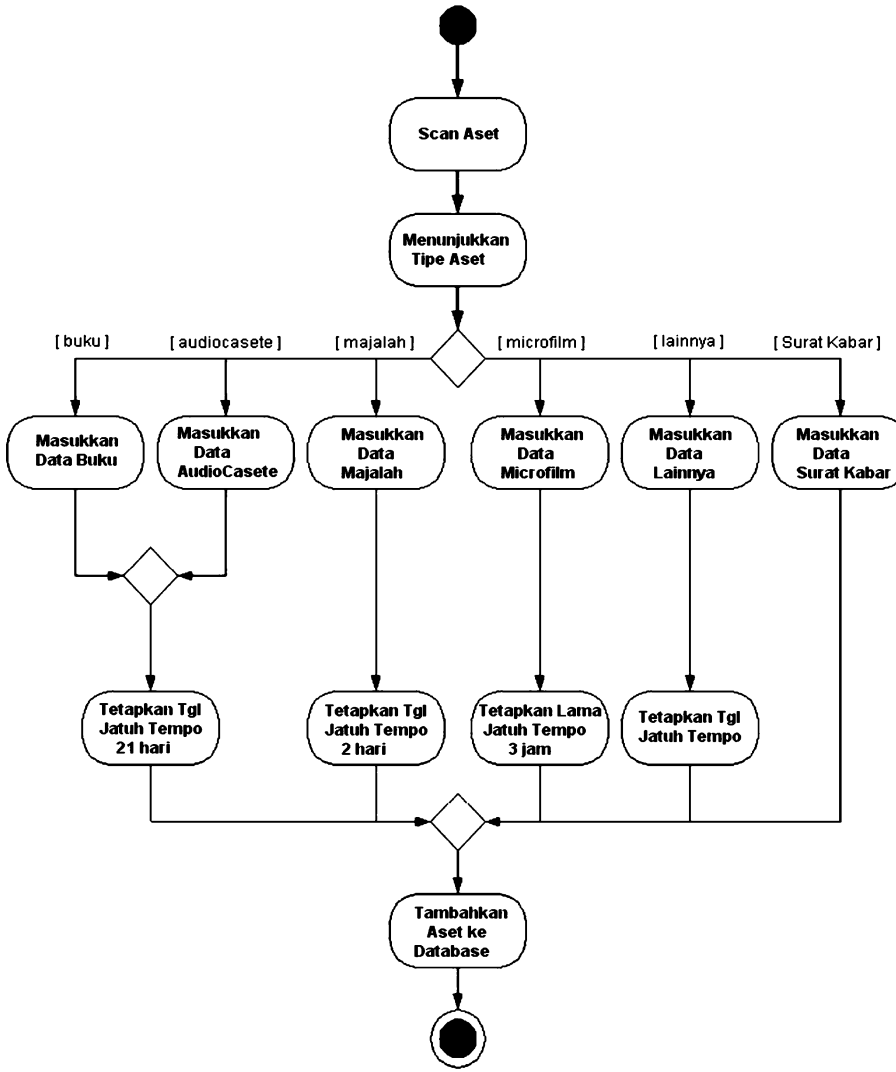
6.6 Contoh Terapan

1. Berikut ini contoh terapan diagram aktivitas (DA) pada sistem penarikan uang tunai dari ATM sebuah Bank (Gambar 6.25).

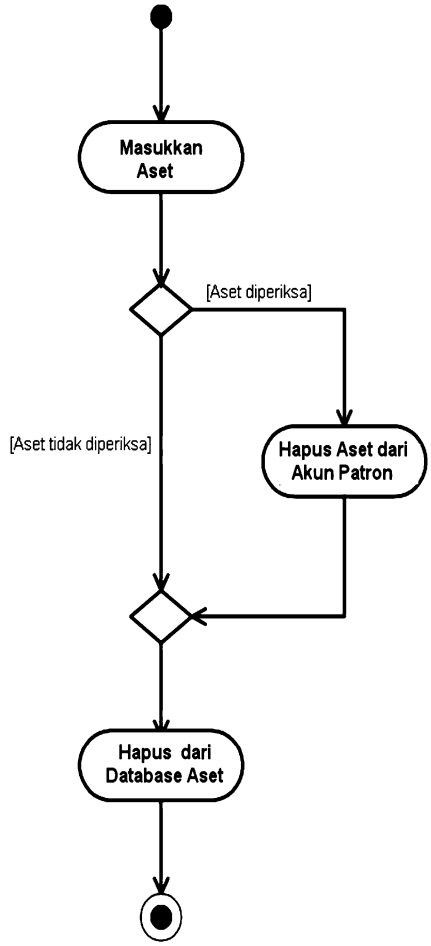


Gambar 6.25 DA Sistem Penarikan Uang Tunai dari ATM

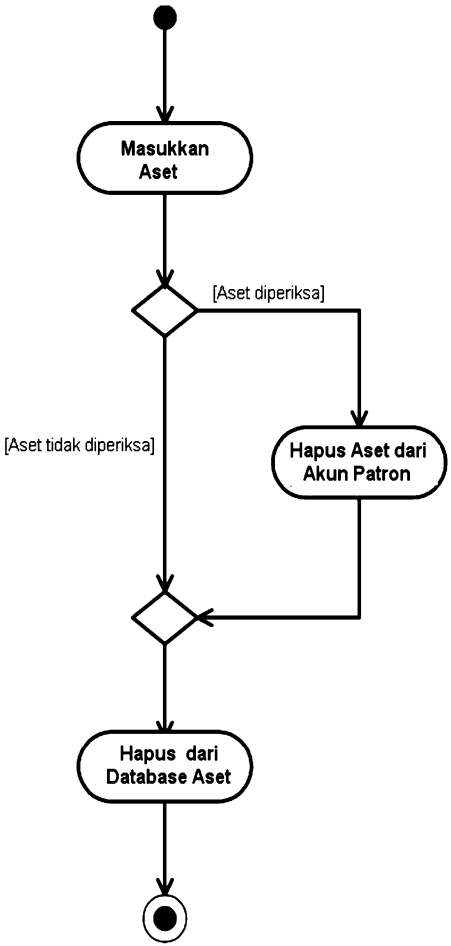
2. Gambar berikut ini (Gambar 6.26<a> sampai dengan Gambar 6.26<i>) merupakan contoh terapan pembangunan diagram aktivitas (DA) sistem Perpustakaan.



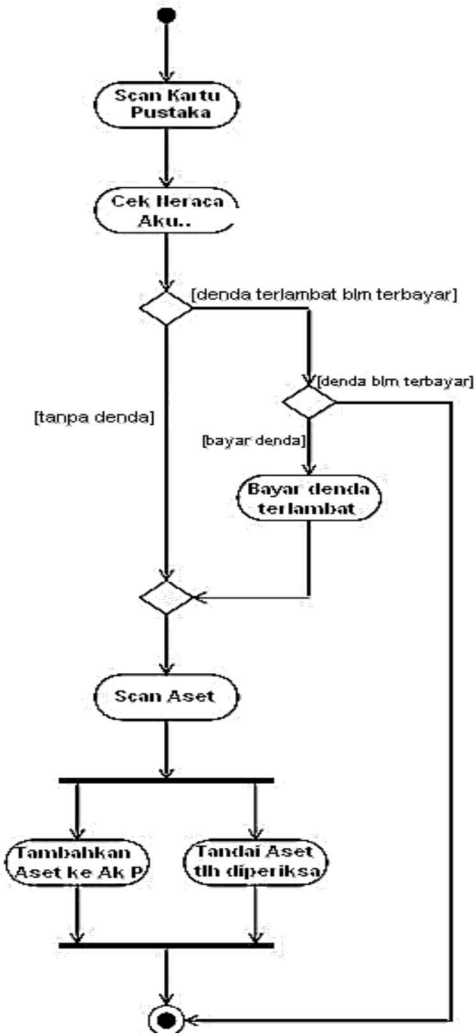
Gambar 6.26<a> DA Tambahkan aset ke database



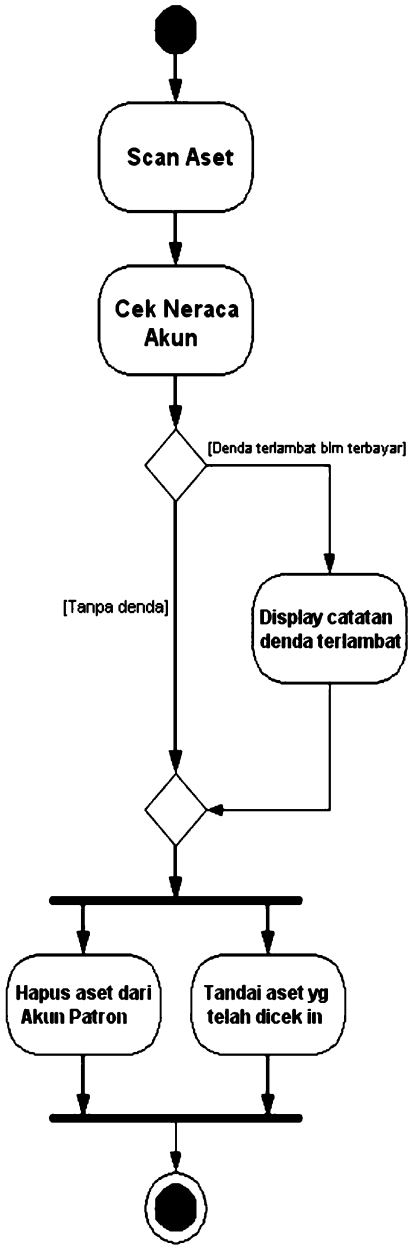
Gambar 6.26<b> DA Hapus aset dari database



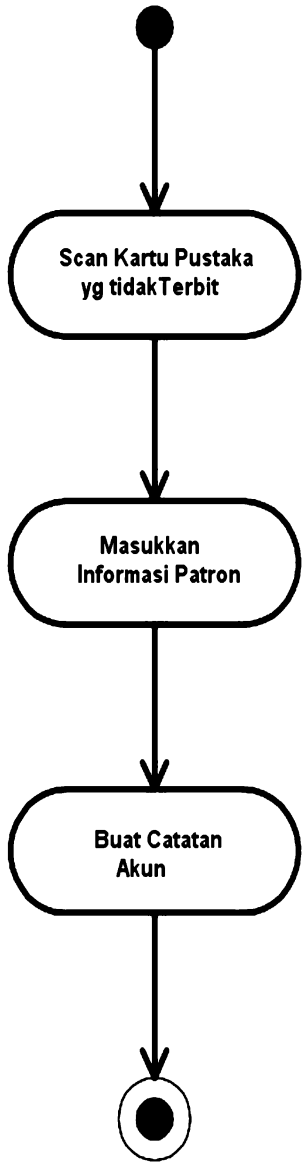
Gambar 6.26<c> DA Buat Laporan



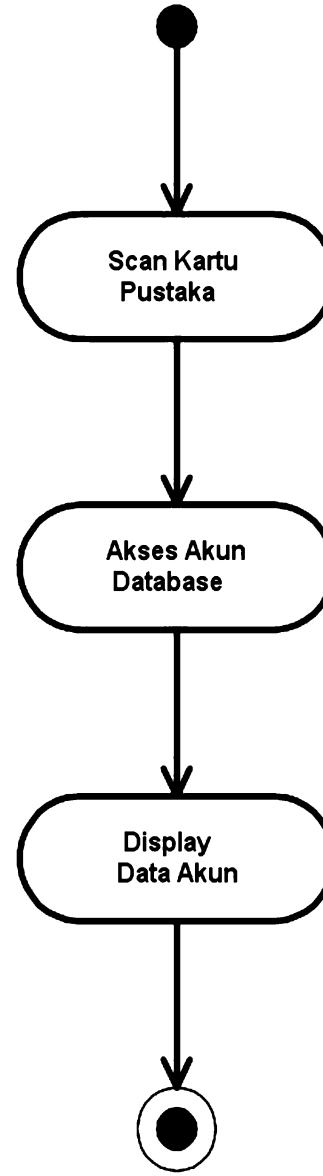
Gambar 6.26<d> DA Periksa Aset Keluar



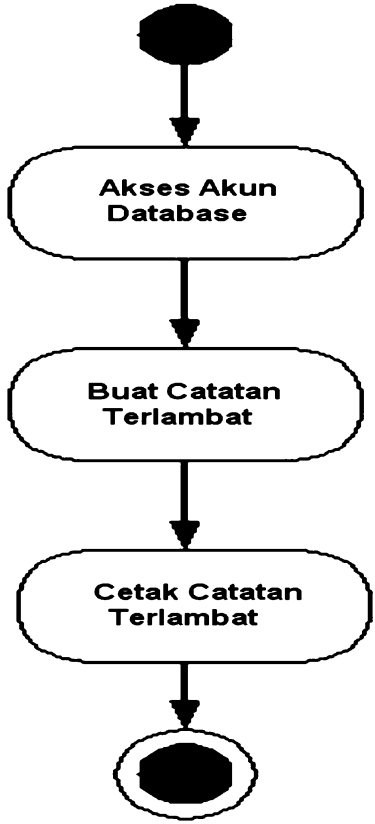
Gambar 6.26<e> DA Periksa Aset Masuk



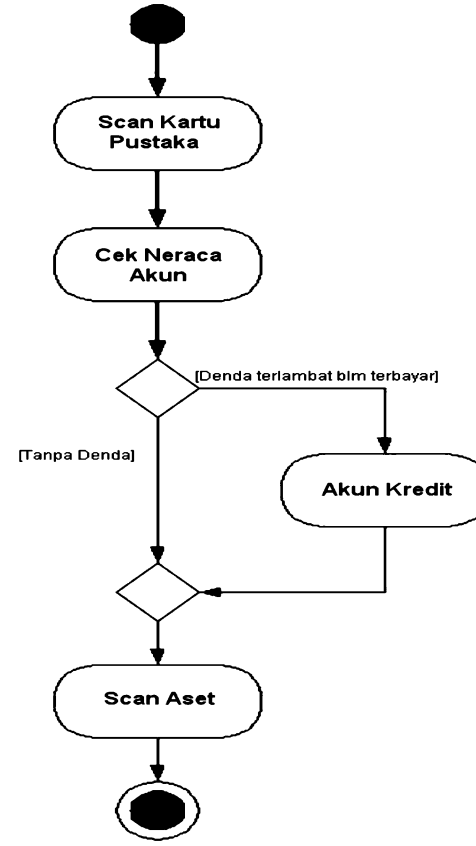
Gambar 6.26<f> DA Isu Kartu Pustaka



Gambar 6.26<g> DA Akses informasi Akun



Gambar 6.26<h> DA Buat Catatan Terlambat



Gambar 6.26<i> DA Bayar Denda keterlambatan

BAB 7
IMPLEMENTASI

Dalam bab ini, akan disajikan contoh implementasi dari 2 (dua) buah studi kasus yang akan diselesaikan dengan menggunakan materi pembahasan dari bab-bab sebelumnya.

STUDI KASUS 1

Sebuah sistem delivery/pengiriman (submission) tugas (assignment) siswa (mahasiswa), dengan kebutuhan atau spesifikasi sistem ini mencakup hal-hal sebagai berikut.

1. Setiap kursus (perkuliahan) dalam sistem memiliki dosen yang ditugaskan untuk itu. Penugasan kepada dosen dilakukan oleh koordinator kursus/perkuliahan (juga seorang dosen). Sebagai bagian dari kursus, dosen dapat membuat tugas dan menilai makalah/naskah hasil pekerjaan/jawaban siswa, termasuk pemberian saran serta apresiasi (poin) kepada siswa/mahasiswa.
2. Administrator (admin) kursus menentukan dosen mana yang menilai makalah/naskah jawaban yang mana. Pada akhir kursus, admin kursus juga mengatur tentang penerbitan sertifikat. Nilai siswa dihitung berdasarkan jumlah total poin yang dicapai atas naskah jawaban tugas yang diserahkan.
3. Siswa dapat mengambil kursus dan mengunggah makalah/naskah jawaban.
4. Semua pengguna—mahasiswa dan dosen—dapat mengelola data pengguna, kursus serta tugas yang ditetapkan, dan melihat makalah/naskah yang dikirimkan serta nilai poin. Namun demikian, siswa hanya dapat melihat makalah/naskah jawaban mereka sendiri beserta nilainya. Dosen hanya dapat melihat makalah

yang ditugaskan untuk mereka dan nilai-nilai yang telah mereka berikan. Admin kursus memiliki hak akses untuk semua data.

5. Kursus dibuat dan dihapus oleh admin.
6. Ketika kursus dibuat, setidaknya satu admin harus ditugaskan untuk itu. Admin kursus, kemudian dapat ditugaskan di tahap berikutnya (termasuk dapat menghapus kursus).
7. Informasi tentang pengguna dan admin secara otomatis ditransfer dari sistem lain. Oleh karena itu, fungsi yang memungkinkan pembuatan data pengguna tidak diperlukan.
8. Semua fungsi sistem hanya dapat digunakan/diakses oleh orang-orang yang berhak.

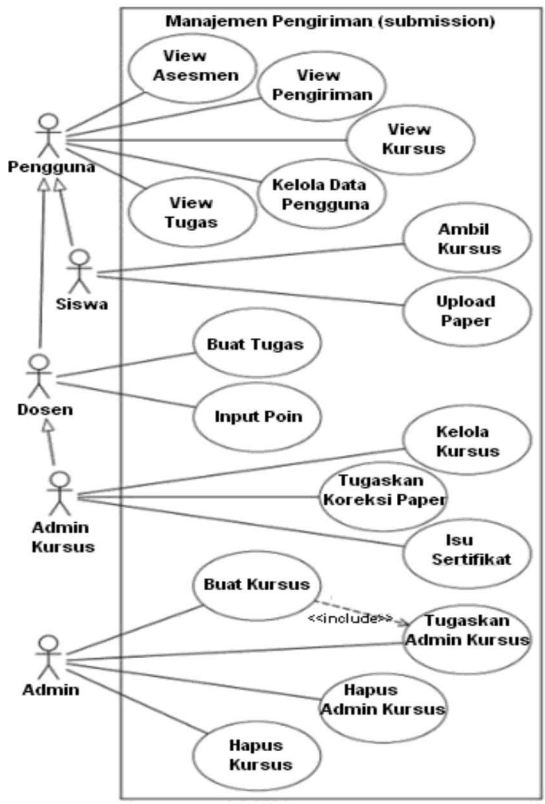
Aktor dan use case untuk spesifikasi tersebut tersaji sebagaimana diagram use case Gambar 7.1. Proses login dan logout tidak tersaji dalam diagram use case, dengan asumsi bahwa mereka bukan fungsi yang diinginkan oleh aktor, tetapi sebaliknya tetap berkontribusi sebagai prasyarat pada penggunaan sistem yang aman.

Selanjutnya, sistem dapat diperbesar dengan memodelkan struktur dan perilakunya. Gambar 7.2 menggambarkan diagram kelas struktur internal sistem pengiriman/*delivery* tugas. Semua aktor yang ada dalam diagram use case (gambar 7.1) juga dimodelkan dalam diagram kelas (Gambar 7.2), meskipun dinyatakan bahwa aktor-aktor tersebut bukan bagian dari sistem. Peran mereka dalam diagram kelas hanya bersifat mewakili data para aktor dan bukan aktor itu sendiri.

Data tersebut akan digunakan dalam pelaksanaan otorisasi, penugasan kepada siswa, dll. Informasi tentang pengguna tersimpan dalam kelas pengguna. Atribut otorisasi digunakan sebagai pembeda antara admin nilai dan semua pengguna lain.

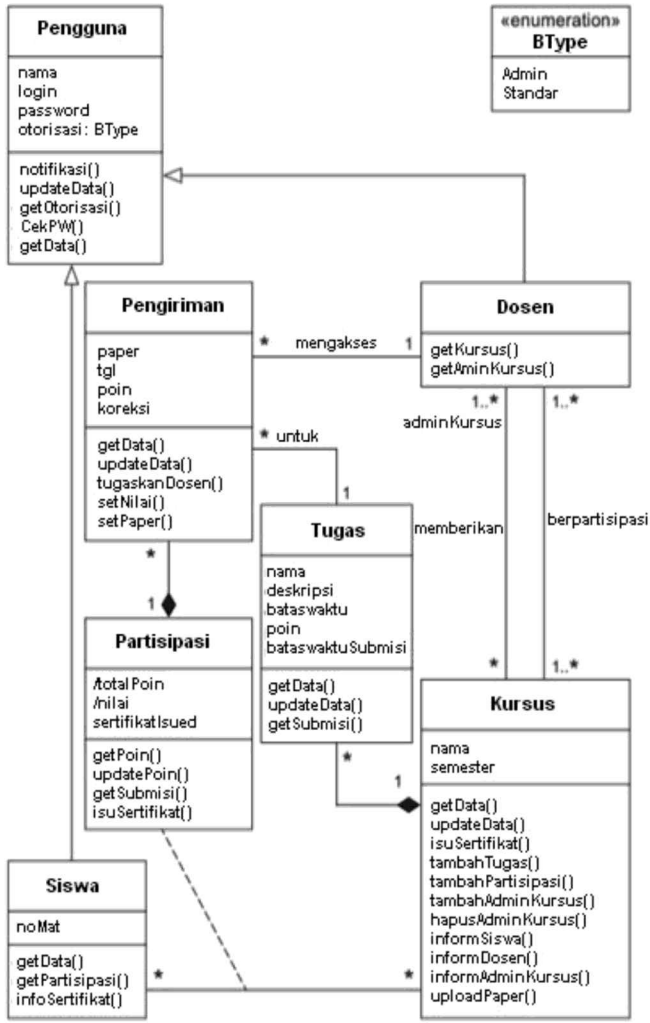
Dalam diagram kelas (Gambar 7.2), seorang dosen dapat menjadi admin kursus dalam hubungannya pemberian kursus. Tugas ditetapkan ke kursus, dengan cara yang sama seperti pengiriman ditetapkan ke tugas. Pemodelan partisipasi siswa dalam kursus menggunakan kelas asosiasi yang berisi informasi tentang jumlah total poin dan nilai siswa. Kedua nilai dihitung secara otomatis dan atribut dilabeli sebagai atribut turunan.

Diagram kelas sampai dengan saat ini tidak memuat informasi yang spesifik, masih bersifat umum (dapat dikembangkan di langkah selanjutnya). Asumsi atau skenario berikutnya sebagai hasil pengembangan use case dan diagram kelas dapat dilihat alur komunikasi pada diagram sekuens (Gambar 7.3). Sementara diagram aktivitas yang mendeskripsikan proses secara lebih detail terkait tentang sistem *delivery*/pengiriman tugas direpresentasikan sebagaimana Gambar 7.4.



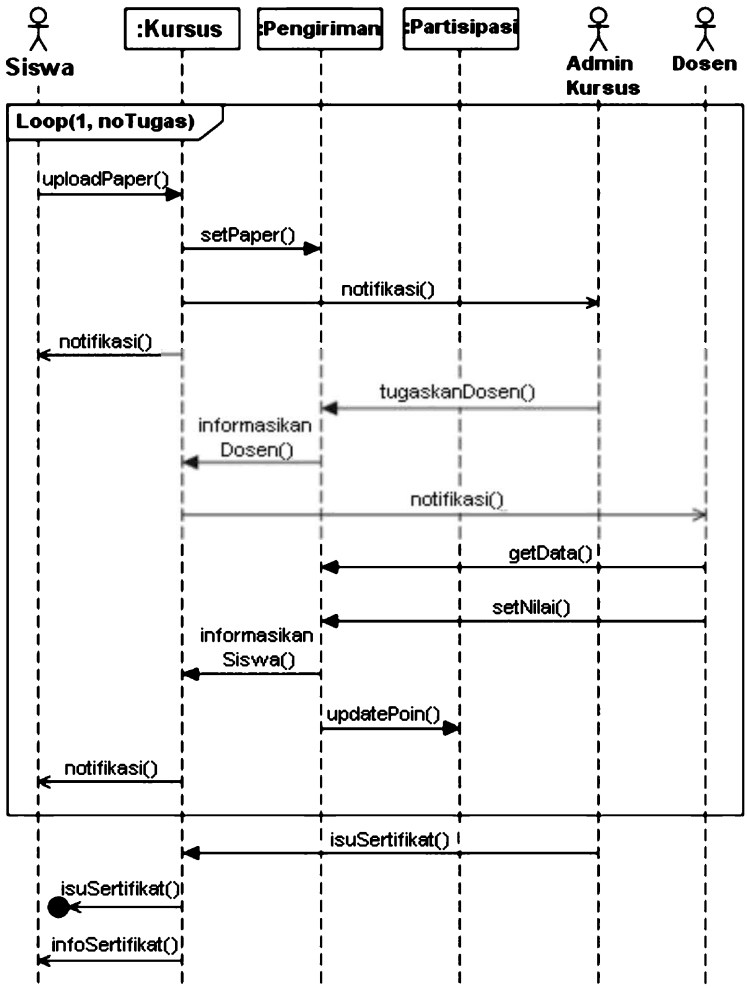
Gambar 7.1 Diagram use case sistem pengiriman (*submission*)

Diasumsikan bahwa pada antarmuka pengguna, admin kursus dapat melihat ikhtisar kursus yang ditetapkan. Pertama, administrator kursus memilih kursus untuk menerbitkan sertifikat. Para siswa yang telah mengambil diperlihatkan. Admin kursus, kemudian dapat memilih apakah akan menerbitkan sertifikat untuk semua atau hanya untuk siswa tertentu. Dalam kasus terakhir, administrator juga harus menetapkan siswa yang akan mendapatkan sertifikat. Nilai dihitung, dikirim ke kantor siswa, dan masing-masing siswa diberi tahu/diinformasikan tentang nilainya. Dalam implementasi praktis, sangat penting memodelkan semua aliran yang mungkin secara lebih detail dan mendekati kenyataan.

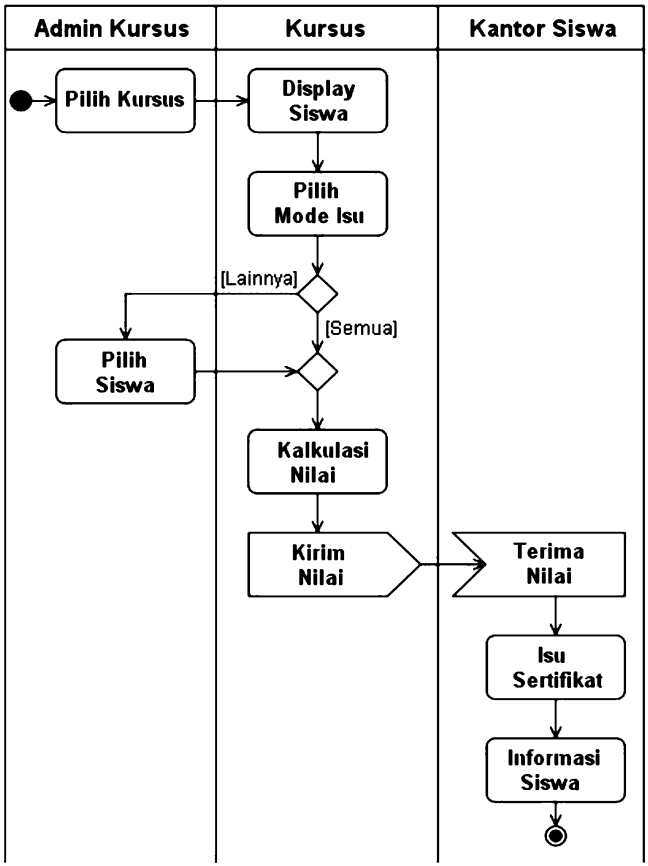


Gambar 7.2 Diagram kelas sistem pengiriman (*submission*)

Hal yang terpenting dalam memodelkan proses sistem informasi haruslah dilakukan secara tepat dan mendekati kenyataan. Mengingat ketidaktepatan atau kekurangtepatan atas hal tersebut dapat berakibat sistem tidak diterima oleh pengguna karena harapan pengguna akan diperolehnya peningkatan efisiensi atas sistem yang dibangun tidak akan dapat tercapai.



Gambar 7.3 Diagram sekuens sistem pengiriman (*submission*)



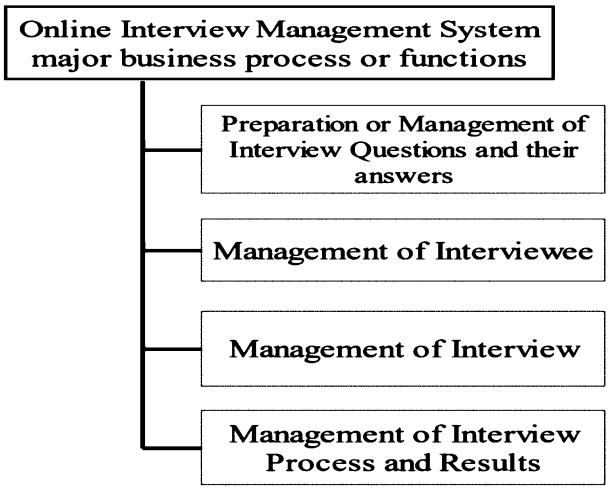
Gambar 7.4 Diagram aktivitas isu (terbit) sertifikat

STUDI KASUS 2

Sistem Manajemen Wawancara Online (Online Interview Management System [OIMS]).

Hal berikut ini merupakan Pemodelan Spesifikasi Persyaratan Perangkat Lunak OIMS dalam konteks wawancara online.

Empat kemungkinan fungsi utama dari proses bisnis ini meliputi persiapan atau manajemen wawancara pertanyaan dan jawaban, manajemen orang yang diwawancarai (*interviewee*), manajemen wawancara serta manajemen proses wawancara dan hasil (Gambar 7.5).



Gambar 7.5 Proses fungsional dasar OIMS

Untuk setiap proses utama (Gambar 7.5), masing-masing dijabarkan aktivitas-aktivitasnya sebagaimana Tabel 7.1.

Aktivitas-aktivitas yang ada pada Tabel 7.1 akan digunakan sebagai pertimbangan dalam pembuatan kebutuhan persyaratan fungsional sistem, mengingat persyaratan fungsional menggambarkan hubungan antara input dan output sistem. Tabel 7.2 menunjukkan daftar kebutuhan/persyaratan fungsional dari sistem.

Kebutuhan fungsional dapat dikategorikan sebagai fungsi yang nyata dan yang tersembunyi (sembunyi). Kebutuhan fungsional yang nyata menunjukkan apa yang sedang dilakukan pengguna. Sementara kebutuhan fungsional yang sembunyi, menunjukkan bahwa fungsi sedang dilakukan, tetapi tidak terlihat oleh pengguna.

Tabel 7.1 Proses dan Aktivitas Utama OIMS

No.	Proses Utama	Aktivitas Proses Bisnis Utama
1	Persiapan atau manajemen pertanyaan wawancara dan jawaban mereka	a. Mendaftar bidang keahlian yang akan diwawancarai. b. Menyiapkan deskripsi pekerjaan/lowongan (misalnya, jabatan, kualifikasi yang dibutuhkan, dll.). c. Mendaftar spesialis bidang para ahli yang dapat membuat pertanyaan wawancara. d. Membuat pertanyaan wawancara. e. Membuat jawaban untuk pertanyaan yang dibuat. f. Alokasikan tanda/nilai untuk jawaban setiap pertanyaan.
2	Manajemen *interviewee*	a. Memublikasikan deskripsi lowongan. b. Tamu mendaftar secara online, pada tahap ini mereka disebut pelamar. c. Pelamar mengunggah dokumen pendukung yang relevan. d. Filter pelamar berdasarkan kualifikasi. Hanya pelamar yang memenuhi syarat yang dipanggil/diwawancarai. e. Daftarkan pelamar yang diwawancarai ke dalam sistem. f. Beri tahu pelamar yang terpilih untuk diwawancarai, jenis pekerjaan yang ditawarkan/diaplikasikan, waktu dan lokasi wawancara. g. Menetapkan kredensial login ke pelamar yang diwawancarai.
3	Manajemen wawancara	a. Membuat wawancara. b. Mengalokasikan pertanyaan untuk wawancara. c. Buat jadwal waktu (waktu, tempat, program). d. Link wawancara dan yang diwawancarai.

4	Manajemen proses wawancara dan hasilnya	a. Membuat *interviewee* memberikan kredensial login. b. Membuat instruksi tampilan yang diwawancarai. c. Orang yang diwawancarai melakukan pertanyaan (yaitu menjawab pertanyaan wawancara). d. Mengontrol waktu pelaksanaan wawancara. e. Tandai/nilai setiap pertanyaan yang telah selesai. f. Hitung total nilai yang diperoleh untuk setiap orang yang diwawancarai. g. Nilai setiap tanda total yang diperoleh oleh orang yang diwawancarai. h. Mengurutkan hasil berdasarkan kinerja: total nilai, penilaian. i. Pilih hanya orang terbaik dari sejumlah orang yang diwawancarai dan diperlukan sesuai deskripsi pekerjaan.

Tabel 7.2 Kebutuhan Fungsional OIMS

No. Acuan	Deskripsi Fungsional	Kategori
F1	**Persiapan atau manajemen pertanyaan wawancara dan jawaban mereka**	
	F1.1 Sistem harus mengizinkan pendaftaran bidang keahlian yang diperlukan untuk diwawancarai.	Nyata
	F1.2 Sistem harus mengizinkan penciptaan deskripsi pekerjaan atau lowongan berdasarkan bidang keahlian yang akan mencakup jabatan, kualifikasi yang dibutuhkan, jumlah yang dibutuhkan.	Nyata
	F1.3 Sistem harus mengizinkan pendaftaran spesialis bidang para ahli yang dapat membuat pertanyaan wawancara.	Nyata
	F1.4 Sistem harus mengizinkan pembuatan banyak pertanyaan wawancara dari berbagai jenis (pilihan berganda, isi yang kosong, esai, pencocokan, dll.).	Nyata

	F1.5 Sistem harus mengizinkan untuk menciptakan jawaban untuk pertanyaan.	Nyata
	F1.6 Sistem harus mengizinkan alokasi tanda/nilai untuk setiap jawaban pertanyaan.	Nyata
F2	**Manajemen yang diwawancarai (interviewee)**	
	F2.1 Sistem harus mengizinkan deskripsi pekerjaan yang lowong/kosong untuk dipublikasikan.	Nyata
	F2.2 Sistem harus mengizinkan pelamar untuk mendaftar.	Nyata
	F2.3 Sistem harus mengizinkan pelamar untuk melamar pekerjaan yang dipublikasikan secara online.	Nyata
	F2.4 Sistem harus mengizinkan pengunggahan dokumen yang diperlukan ke dalam sistem, seperti CV dan sertifikat.	Nyata

	F2.5	Sistem harus dapat memfilter aplikasi pelamar berdasarkan himpunan kualifikasi yang dibutuhkan dan tetap dengan hanya pelamar yang memenuhi syarat yang akan dipanggil untuk diwawancarai.	Sembunyi
	F2.6	Sistem harus mendaftarkan semua orang yang diwawancarai (pelamar yang memenuhi syarat) ke dalam database untuk memisahkan mereka dari pelamar yang tidak memenuhi syarat.	Sembunyi
	F2.7	Sistem harus mengirim pemberitahuan ke semua orang yang diwawancarai tentang keterpilihan mereka, deskripsi pekerjaan dipilih, tanggal, dan lokasi atau tempat wawancara.	Nyata
	F2.8	Sistem harus menetapkan kredensial login kepada orang yang diwawancarai.	Nyata

F3	**Manajemen wawancara**	
	F3.1 Sistem harus mengizinkan pembuatan wawancara.	Nyata
	F3.2 Sistem harus mengizinkan alokasi pertanyaan untuk wawancara yang dibuat.	Nyata
	F3.3 Sistem harus mengizinkan pembuatan penjadwalan waktu wawancara, yang meliputi: waktu, tempat dan program lengkap.	Nyata
	F3.4 Sistem harus menghubungkan sebuah wawancara dan yang diwawancarai.	Nyata
F4	**Manajemen proses wawancara dan hasil**	
	F4.1 Sistem harus membatasi orang yang diwawancarai untuk memberikan kredensial login.	Nyata
	F4.2 Sistem harus menghubungkan orang yang diwawancarai dengan wawancara yang bersangkutan.	Sembunyi
	F4.3 Sistem harus menampilkan instruksi wawancara.	Nyata

	F4.4	Sistem harus memungkinkan orang yang diwawancarai untuk melakukan wawancara dengan menampilkan pertanyaan wawancara.	Nyata
	F4.5	Sistem harus dapat mengontrol jadwal waktu untuk wawancara.	Sembunyi
	F4.6	Sistem harus menandai setiap pertanyaan dan memberikan tanda/ nilai.	Sembunyi
	F4.7	Sistem harus dapat menghitung total nilai yang diperoleh untuk setiap interviewee dan per setiap wawancara.	Sembunyi
	F4.8	Sistem harus dapat menilai (mengalokasikan nilai) untuk total nilai yang diperoleh.	Sembunyi
	F4.9	Sistem harus dapat mengurutkan tanda/nilai berdasarkan total nilai dalam urutan naik atau turun, penilaian, dll.	Sembunyi
	F4.10	Sistem harus dapat memilih hanya orang terbaik dari yang diwawancarai berdasarkan jumlah yang dibutuhkan per deskripsi pekerjaan.	Nyata

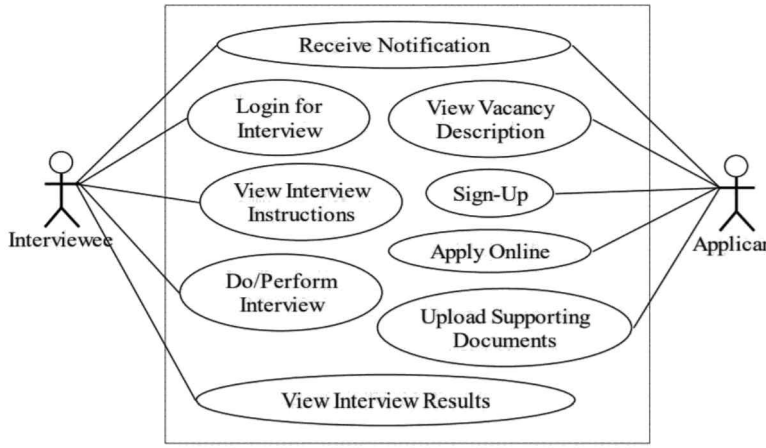
F5	**Fungsi umum/dasar**	
	F5.1 Sistem harus mengizinkan pendaftaran pengguna sistem (termasuk sistem admin dan pewawancara).	Nyata
	F5.2 Sistem harus mengizinkan pengguna sistem untuk masuk dan mendapatkan izin untuk menggunakan sistem.	Nyata
	F5.3 Sistem harus mengizinkan pencarian beberapa jenis informasi yang berbeda menggunakan berbagai kriteria pencarian.	Nyata
	F5.4 Sistem harus dapat melacak pengguna dengan aktivitas yang lengkap.	Sembunyi
	F5.5 Sistem harus dapat menghasilkan laporan statistik.	Nyata
	F5.6 Sistem harus melakukan penghematan semua informasi yang relevan ke dalam sistem.	Sembunyi

Aktor dan Use Case:

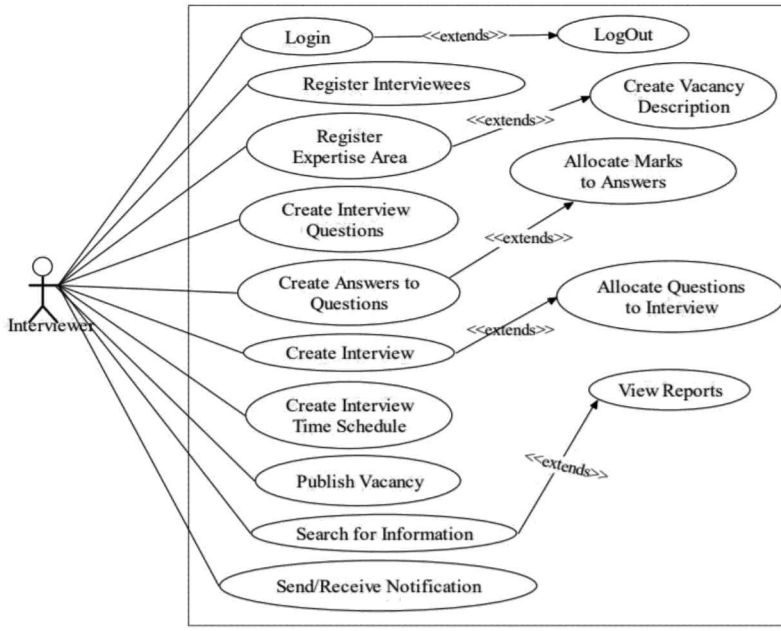
Aktor-aktor yang teridentifikasi dalam sistem wawancara online adalah sebagai berikut.

1. Pelamar (*Applicant*): pengguna yang membaca iklan pekerjaan dan menunjukkan minat untuk mengajukan hal yang sama.
2. Orang yang diwawancarai (*Interviewee*): pengguna yang memenuhi syarat untuk melakukan wawancara setelah disaring. Orang-orang yang diwawancarai akan terdaftar ke dalam sistem dan mereka akan diminta untuk login agar diizinkan untuk melakukan wawancara.
3. Pewawancara (*Interviewer*): Seorang aktor yang bertanggung jawab untuk mempersiapkan seluruh proses wawancara online.
4. Pakar Subjek (*Subject Expert*): Seorang aktor yang bertanggung jawab untuk mempersiapkan pertanyaan wawancara sehubungan dengan bidang keahlian masing-masing.
5. Admin Sistem (*System Admin*): Aktor yang akan mengelola seluruh sistem, termasuk manajemen pengguna.

Sementara itu, berdasar kebutuhan fungsional (Tabel 7.2), use case yang teridentifikasi ada pada Gambar 7.6 dan Gambar 7.7.



Gambar 7.6 Diagram use case peran interviewee dan pelamar (*applicant*)



Gambar 7.7 Diagram use case peran interviewer

Beberapa use case perluasan (*expanded*) dideskripsikan lebih detail pada Tabel 7.3, Tabel 7.4, dan Tabel 7.5.

Tabel 7.3 Use Case Membuat Pertanyaan Wawancara

Nama	Deskripsi
Use case	Membuat pertanyaan wawancara **(Create Interview Questions)**.
Tujuan	Sistem mengizinkan pewawancara untuk membuat bank pertanyaan wawancara.
Aktor	Pewawancara (*Interviewer*).
Prakondisi	Pewawancara harus dikenal sistem.
Pascakondisi	Sejumlah pertanyaan wawancara dibuat dan disimpan ke dalam sistem.
Proses Utama	1. Pewawancara meminta ketentuan untuk membuat pertanyaan. 2. Sistem menampilkan sejumlah opsi, alat yang mungkin untuk membuat pertanyaan (pilihan ganda, isi yang kosong, esai, dll.). 3. Pewawancara memilih alat pertanyaan yang diperlukan. 4. Sistem menampilkan tempat alat yang dipilih terkait ke pewawancara. 5. Pewawancara membuat pertanyaan dan mengirimkan. 6. Sistem memvalidasi pertanyaan yang diajukan dan menyimpan pertanyaan ke dalam sistem. 7. Proses dari 2 hingga 6 berulang sampai pewawancara mengeklik tombol "akhir". 8. Sistem mengakui berhasil membuat pertanyaan kepada pewawancara.
Proses Alternatif	-

Tabel 7.4 Use Case Membuat Jawaban Pertanyaan

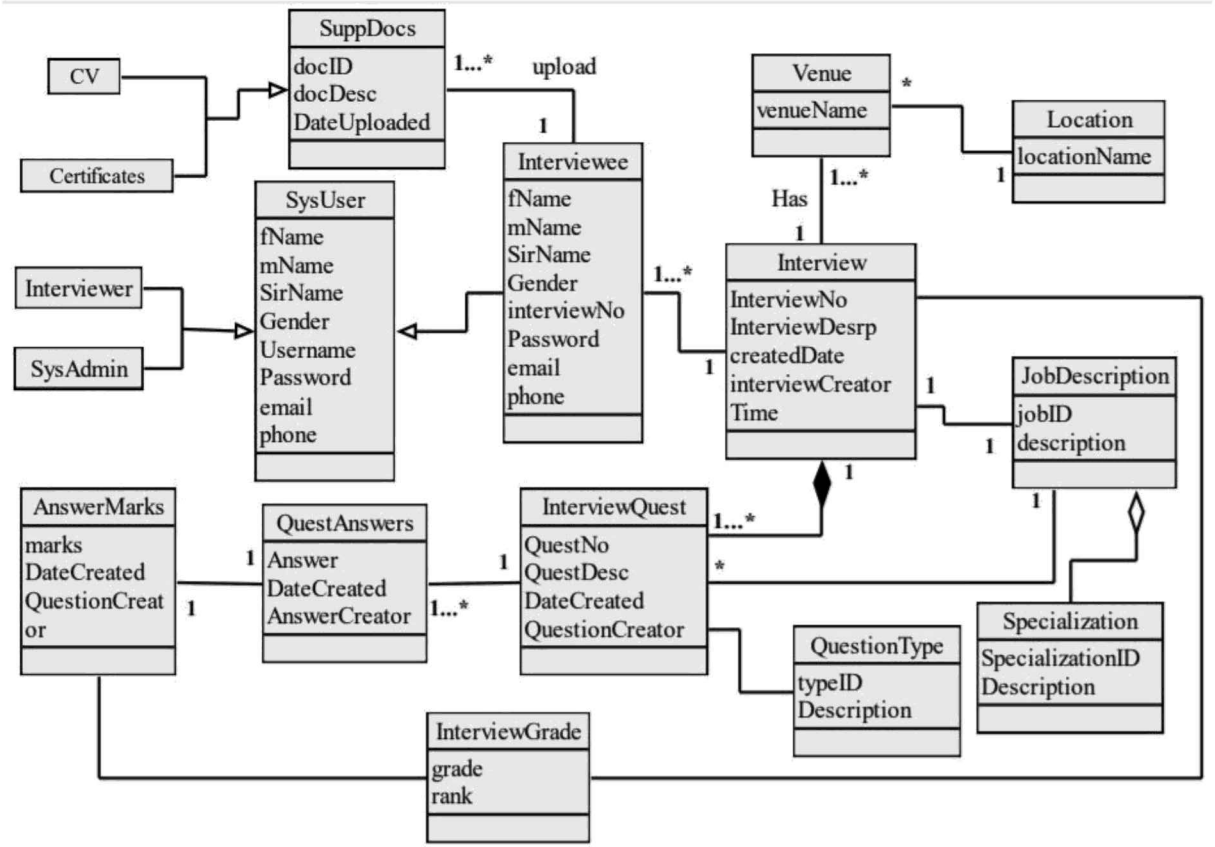
Nama	Deskripsi
Use case	Membuat jawaban pertanyaan **(Create Answers to Questions)**.
Tujuan	Sistem mengizinkan pewawancara untuk membuat jawaban atas pertanyaan wawancara yang sudah dibuat.
Aktor	Pewawancara (*Interviewer*).
Prakondisi	Bank pertanyaan wawancara yang disimpan ke dalam sistem.
Pascakondisi	Pertanyaan wawancara akan memiliki jawaban.
Proses Utama	1. Pewawancara meminta ketentuan untuk membuat jawaban atas pertanyaan. 2. Sistem menampilkan daftar pertanyaan dan memungkinkan pewawancara memilih pertanyaan untuk menjawab. 3. Pewawancara memilih pertanyaan. 4. Sistem menampilkan deskripsi pertanyaan dan formulir/sarana untuk memberikan jawaban. 5. Pewawancara membuat jawaban atas pertanyaan dan menyimpan/mengirimkan. 6. Sistem memvalidasi jawaban yang dikirimkan dan menyimpan jawabannya ke dalam sistem. 7. Sistem mengakui keberhasilan pembuatan jawaban. 8. Proses dari 3 hingga 6 berulang sampai pewawancara mengeklik tombol "akhir".
Proses Alternatif	-

Berikut ini adalah Model Konseptual atau Diagram Kelas Sistem sebagaimana tersaji pada Gambar 7.8. Sementara konsep-konsep yang teridentifikasi mencakup hal-hal sebagai berikut.

1. *Interview*: serangkaian pertanyaan yang dialokasikan untuk wawancara tertentu dan terhubung dengan informasi relevan lainnya seperti tempat.
2. *InterviewQuest*: satu set pertanyaan yang dikategorikan dalam hal deskripsi pekerjaan dan jenis pertanyaan.
3. *QuestAnswers*: jawaban yang disiapkan untuk pertanyaan.
4. *AnswerMarks*: tanda/nilai yang dialokasikan untuk jawaban pertanyaan.
5. *Jobdescription*: deskripsi pekerjaan wawancara sesuai iklan.
6. *Specialization*: deskripsi pekerjaan yang didasarkan pada spesialisasi yang berbeda.
7. *Interviewee*: seseorang yang memenuhi syarat dan dipilih untuk melakukan wawancara.
8. *SysUser*: penyimpanan informasi dari semua pengguna lain yang berinteraksi dengan sistem, selain dari orang yang diwawancarai.

Tabel 7.5 Use case Melaksanakan Wawancara

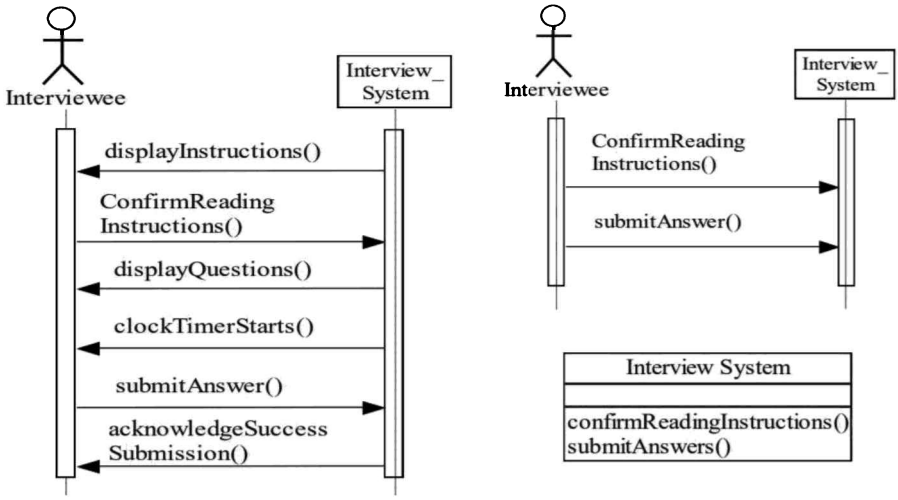
Nama	Deskripsi
Use case	Melaksanakan wawancara **(Do/Perform Interview)**.
Tujuan	Di sinilah orang yang diwawancarai menyelesaikan wawancara dengan menjawab pertanyaan.
Aktor	Yang diwawancarai (*Interviewee*).
Prakondisi	1. Interview terdaftar ke dalam sistem. 2. Wawancara sudah diatur dan di tempat.
Pascakondisi	Orang yang diwawancarai akan dicatat karena sudah melewati proses wawancara.
Proses Utama	1. Sistem menampilkan instruksi untuk wawancara dan minta orang yang diwawancarai untuk mengonfirmasi bahwa dia telah membaca dan siap untuk memulai wawancara. 2. Orang yang diwawancarai mengeklik tombol OK untuk mengonfirmasi instruksi membaca. 3. Sistem menampilkan semua pertanyaan wawancara dan waktu jam mulai dihitung. 4. Orang yang diwawancarai memberikan jawaban atas pertanyaan, menyimpan, dan mengirimkan jawaban setelah selesai. 5. Sistem mengakui keberhasilan pengiriman jawaban kepada orang yang diwawancarai.
Proses Alternatif	1. Sistem menampilkan satu demi satu pertanyaan dan waktu jam mulai dihitung. 2. Orang yang diwawancarai memberikan jawaban atas pertanyaan yang diberikan, kemudian akan diberi kesempatan untuk melakukan pertanyaan berikut. Menyimpan dan mengirimkan jawaban setelah selesai (*interviewee* dapat maju dan mundur untuk menavigasi ke pertanyaan).



Gambar 7.8 Diagram Kelas Sistem OIMS

Selanjutnya Diagram Sekuens Sistem (DSS) untuk use case melaksanakan wawancara tersaji pada Gambar 7.9. DSS ini terdiri atas 2 operasi sistem (lihat 2 anak panah menuju sistem), yaitu confirmReadingInstructions() dan submitAnswers().

Untuk setiap operasi sistem tersebut, kemudian dapat ditentukan diagram sekuensnya.



Gambar 7.9 DSS use case melaksanakan wawancara

DAFTAR PUSTAKA

Chong Fen Yu. 2009. *BIT 201: Object Modelling Using UML*. Lecture Material of Dual Degree Program. Malaysia: Stikom Bali-HELP University College.

Dennis, A., Wixom, B.H., & Tegarden D. 2015. *System Analysis & Design: An Object-Oriented Approach with UML.* Fifth Edition. USA: Wiley & Sons.

Jackson, D. 2011. *Software Abstractions: Logic, Language, and Analysis*. Revised Edition. USA: The MIT Press.

Kalinga, E.A. 2021. "Learning Software Development through Modeling Using Object Oriented Approach with Unified Modeling Language: A Case of an Online Interview System". *Jurnal of Learning for Development*, Vol. 8, No. 1.

Larman, C. 2004. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Third Edition. USA: Prentice Hall.

Seidl, M., Scholz, M., Huemer, C., & Kappel, G. 2015. *UML @ Classroom: An Introduction to Object-Oriented Modeling*. Switzerland: Springer International Publishing.

Wazlawick, R.S. 2014. *Object-Oriented Analysis and Design for Information Systems*. USA: Morgan Kaufmann.

GLOSARIUM

Abstraksi Sistem: Mengenali karakteristik sistem secara umum.

Agregasi: Relasi yang kuat antarobjek-objek dari sebuah kelas dengan objek-objek lain dari kelas yang berbeda, yang secara nyata merupakan bagian-bagiannya (relasi bagian dari).

Agregasi Berbagi: Agregasi dengan ikatan yang lemah atas bagian-bagian (atau elemen-elemen) terhadap keseluruhan, yang berarti bahwa elemen-elemen tersebut ada secara independen dari keseluruhan.

Agregasi Komposisi: Menyatakan bahwa bagian/elemen tertentu hanya dapat terkandung/termuat dalam (paling banyak) satu objek komposit pada satu titik waktu tertentu.

Aktor: Merupakan pengguna yang berinteraksi dengan sistem, dalam konteks use case yang terlibat dengannya.

Asosiasi: Representasi relasi antarkelas (yang berbeda).

Atribut Kelas: Merupakan tempat menyimpan informasi bagi semua objek, tetapi bernilai spesifik dan berbeda bagi setiap instans/objek.

Diagram Kelas: Merepresentasikan aspek/struktur statis dari sistem. Elemen diagram kelas berupa kelas-kelas, relasi di antaranya, dan multiplisitas.

Diagram Aktivitas: Memfokuskan pada pemodelan proses bisnis dari sistem atau prosedur pemrosesan sebuah sistem. Artefak diagram aktivitas berupa gambar-gambar visualisasi aliran kontrol dan data di antara berbagai langkah/aksi yang diperlukan dalam mengimplementasikan aktivitas-aktivitas sebuah sistem.

Diagram Sekuens (DS): Merupakan salah satu bentuk diagram interaksi yang memodelkan interaksi antara objek dalam sebuah use case. DS menggambarkan bagaimana bagian-bagian yang berbeda dari sebuah sistem berinteraksi satu sama lain untuk melaksanakan fungsi dan urutan ketika use case tertentu dieksekusi. DS merupakan aspek detail dari perilaku sistem.

Diagram Sekuens Sistem (DSS): Merupakan diagram yang menggambarkan interaksi antara pengguna dengan sistem. DSS dibangun untuk setiap use case.

Diagram Use Case: Menjelaskan semua skenario penggunaan (use case) yang akan dikembangkan sistem, yaitu tentang fungsi-fungsi apa yang sistem bisa/harus dilakukan, tetapi tidak membahas aspek detail implementasi. Diagram use case menggambarkan kebutuhan sistem yang akan dibangun.

Enkapsulasi: Pemaketan beberapa item bersama ke dalam sebuah unit atau merupakan gabungan dari proses dan data ke dalam satu entitas.

Extend: Tipe relasi antar-use case, dengan use case penyerta bersifat opsional dalam eksekusinya (ada delay). Use case penyerta dapat dieksekusi secara independen.

Generalisasi: Digunakan untuk menyoroti kemiripan/keserupaan antarkelas sehingga karakteristik umum antarkelas tidak perlu disebutkan secara berulang. Generalisasi dapat dipakai untuk mendapatkan kelas yang lebih spesifik dari kelas yang ada.

Include: Tipe relasi antar-use case, yang use case penyerta bersifat wajib dieksekusi (seketika). Use case penyerta dapat dieksekusi secara independen.

Inheritance/Pewarisan: Relasi antarkelas yang bersifat kelas yang satu (subkelas) tipe/macam dari kelas yang lain (superkelas). Subkelas mewarisi atribut dan operasi/metode superkelas.

Kelas: Sebuah wadah/template untuk mana objek-objek serupa dibangun. Notasi kelas berupa kotak persegi panjang dengan 3 subkotak. Subkotak teratas digunakan sebagai nama kelas, subkotak di tengah memuat nama-nama atribut, dan subkotak terakhir memuat nama-nama operasi/metode.

Metode/Operasi Kelas: Aksi objek-objek dalam sebuah kelas.

Model: Adalah gambaran atas sesuatu atau sebuah sistem yang nyata (atau abstrak) ke dalam bentuk yang lebih sederhana, dengan tujuan agar mudah dipelajari dan dimengerti.

Multiplisitas: Mengindikasikan berapa banyak dari suatu objek berasosiasi dengan objek lain.

Objek: Merupakan instans dari kelas. Objek yang memiliki perilaku, status, dan identitas.

Operasi Kelas: Menentukan bagaimana perilaku spesifik dapat dipicu oleh individu objek.

Orientasi Objek: Suatu cara menggambarkan/menyajikan sistem yang berhubungan dengan konsep-konsep tertentu, utamanya kelas, objek, dan pesan.

Perilaku Objek: Cara objek beraksi atau bereaksi terkait dengan perubahan status dan pelewatan pesan. Perilaku objek sesuai perilaku kelas.

Perilaku sistem: Merupakan muara dari pembahasan materi use case (dengan diagram use case) dan kelas (dengan diagram kelas). Apabila kelas menjelaskan aspek statis (struktur statis) dari sistem maka perilaku sistem menjelaskan tentang aspek dinamisnya (struktur dinamis).

Polimorpisme: Pengiriman pesan yang sama, tetapi dengan aksi yang berbeda bagi objek penerima.

Sistem: Terdiri atas komponen-komponen yang saling terintegrasi dan terkait satu sama lain serta saling memengaruhi sedemikian hingga dapat dianggap sebagai unit tunggal, berbasis tugas atau berbasis tujuan.

Sistem Berorientasi Objek: Himpunan atau kumpulan objek-objek yang bekerja sama atau berinteraksi guna mencapai/merealisasi tujuan sistem.

Status Objek: Nilai-nilai atribut objek.

Unified Modeling Language (UML): Bahasa yang digunakan untuk menspesifikasikan, memvisualisasi, membangun, dan mendokumentasikan artefak dari sistem perangkat lunak, serta untuk memodelkan sistem bisnis dan nonperangkat lunak.

Use Case: Merupakan serangkaian langkah demi langkah (*step-by-step*) interaksi aktor (pengguna) dalam berinteraksi atau berkomunikasi dengan sistem dalam rangka menyelesaikan sebuah problem/masalah/tujuan (fungsi).

Visibilitas: Digunakan untuk mewujudkan penyembunyian informasi, sebuah konsep yang penting dalam komputasi. Visibilitas operasi menentukan siapa yang diizinkan untuk menggunakan fungsionalitas operasi.

TENTANG PENULIS

Muhammad Rusli, lahir di Surabaya pada tanggal 3 September 1955. Sarjana Matematika Statistik FIPIA ITS, tahun 1981. Magister (S-2) Teknik Informatika-ITS, tahun 1999. Doktor (S-3) Teknologi Pembelajaran dari Universitas Negeri Malang, tahun 2013. Karier sebelumnya, sebagai Staf Peneliti pada Pusat Penelitian Perkebunan Gula Indonesia (P3GI) - Pasuruan, tahun 1982–2001. Pada saat ini berprofesi sebagai dosen tetap ITB STIKOM Bali (2004–sekarang), dan sebagai Staf Ahli Bidang Pengembangan Akademik dan Pembelajaran. Anggota APTIKOM dan Dewan Pakar IGI Bali. Beberapa buku telah diterbitkan, antara lain: *Belajar Pemrograman Bahasa Java dengan NetBeans*; *Logika & Matematika*; *Multimedia Pembelajaran yang Inovatif*; *Memahami E-learning*; dan *Pembelajaran Daring yang Efektif*.

Email : rusli@stikom-bali.ac.id; ruslim21@gmail.com

Evi Triandini, lahir di Jember pada tanggal 22 April 1970. Sarjana Pertanian Universitas Brawijaya, tahun 1993. Magister (S-2) Manajemen Informatika – AIT Bangkok, tahun 1997. Doktor (S-3) Teknik Informatika dari ITS, tahun 2018. Karier sebelumnya, sebagai dosen STIKOM Surabaya tahun 1993–2003. Pada saat ini berprofesi sebagai dosen tetap ITB STIKOM Bali (2003–sekarang). Anggota IEEE, anggota APTIKOM, Pengurus INDOCEISS dan Pengurus CORIS. Ada buku Penulis yang telah disusun dan diterbitkan melalui Penerbit ANDI Yogyakarta.

Email : evi@stikom-bali.ac.id; evi.triandini11@gmail.com

INDEKS

U

V

W

Memodelkan Sistem Informasi Berorientasi Objek

Konsep Dasar, Prosedur dan Implementasi

Dalam membangun/mengembangkan suatu sistem perangkat lunak yang dipesan pelanggan (klien), salah satu tantangan utama yang akan dihadapi adalah mengklarifikasi dan mengonfirmasi apa yang sebenarnya diinginkan pelanggan, dan apakah kita telah memahami dengan tepat kebutuhan sistem yang dikehendaki pelanggan sesuai dengan prospektif sebuah sistem. Langkah tersebut sangat penting, karena dapat menentukan keberhasilan atau kegagalan proyek yang akan kita kerjakan. Pertanyaannya adalah bagaimana kita sebaiknya berkomunikasi dengan pelanggan/manajer bisnis? Bahasa alami belum tentu merupakan pilihan yang baik karena tidak tepat dan ambigu. Kesalahpahaman dapat dengan mudah terjadi, dan memiliki risiko yang cukup serius ketika orang dengan latar belakang yang berbeda (misalnya, seorang ilmuwan/praktisi komputer dan pelanggan/manajer bisnis) berkomunikasi di lintas tujuan.

Untuk itu, diperlukan sebuah model dari perangkat lunak yang akan dibangun. Model ini hanya fokus pada aspek-aspek penting dari perangkat lunak dalam bentuk notasi/simbol yang jelas dan sesederhana mungkin, dengan mengabaikan abstraksi detail yang tidak relevan. Model ini, sebagaimana dalam arsitektur, disebut rencana konstruksi. Rencana konstruksi untuk sebuah bangunan berisi informasi/gambaran, antara lain tentang denah ruangan dan lantai. Bahan konstruksi yang akan digunakan tidak ditentukan pada saat ini; mereka tidak relevan dan akan membuat rencana lebih rumit daripada yang diperlukan. Rencana konstruksi juga tidak berisi informasi tentang bagaimana kabel listrik akan diletakkan. Rencana terpisah dibuat untuk aspek ini guna menghindari terlalu banyak informasi tersaji sekaligus. Untuk itu, penting dalam bidang teknologi informasi bahwa orang-orang dengan latar belakang yang berbeda (pemrogram, analis dan perancang sistem informasi, pengembang, pelanggan/manajer bisnis) dapat membaca, memahami, menafsirkan, dan mengimplementasikan model. Buku ini menjelaskan konsep dasar, prosedur, dan implementasi dalam memodelkan sistem informasi beorientasi objek dengan UML (*Unified Modelling Language*).

Penerbit ANDI
Jl. Beo 38-40 Yogyakarta
Telp.(0274) 561881 Fax.(0274) 588282
: andipenerbitan@gmail.com
: www.andipublisher.com

COMPUTING & INTERNET
ISBN 978-623-01-2596-6
ISBN 978-623-01-2597-3 (PDF)

Dapatkan Info Buku Baru, Kirim e-mail: info@andipublisher.com | andipublishercom@yahoo.com