MODUL MATA KULIAH

TEKNIK PEMODELAN SISTEM INFORMASI

IF062 – 3 SKS

FAKULTAS TEKNOLOGI INFORMASI
UNIVERSITAS BUDI LUHUR

JAKARTA
SEPTEMBER 2020

TIM PENYUSUN

Brury Trya Sartana, S.Kom., M.M., M.Kom
Atik Ariesta, S.Kom., M.Kom

KATA PENGANTAR

Puji syukur ke hadirat Tuhan Yang Maha Kuasa, yang telah memberikan rahmat-Nya sehingga Modul Perkuliahan Teknik Pemodelan Sistem Informasi untuk mahasiswa/i Program Studi Sistem Informasi Fakultas Teknologi Informasi Universitas Budi Luhur ini dapat diselesaikan dengan sebaik-baiknya.

Modul perkuliahan ini dibuat sebagai penunjang mata kuliah Teknik Pemodelan Sistem Informasi pada Program Studi Sistem Informasi Universitas Budi Luhur. Modul perkuliahan ini diharapkan dapat membantu mahasiswa/i dalam mempersiapkan dan melaksanakan perkuliahan dengan lebih baik, terarah, dan terencana. Pada setiap sub modul berisi teori untuk memperdalam pemahaman mahasiswa/i mengenai materi yang dibahas.

Penyusun menyakini bahwa dalam pembuatan Modul Perkuliahan ini masih jauh dari sempurna. Oleh karena itu penyusun mengharapkan kritik dan saran yang membangun guna penyempurnaan modul perkuliahan ini dimasa yang akan datang.

Akhir kata, penyusun mengucapkan banyak terima kasih kepada semua pihak yang telah membantu baik secara langsung maupun tidak langsung dalam pembuatan modul perkuliahan ini, dan semoga modul perkuliahan ini bermanfaat.

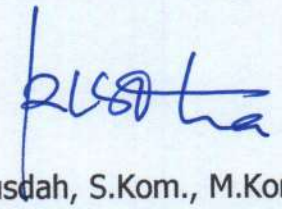
Jakarta, Juli 2020

Penyusun

PENGESAHAN
MODUL PERKULIAHAN

No. Tabel	:	IF062
Nama Matakuliah	:	Teknik Pemodelan Sistem Informasi
Nama Modul	:	Modul Perkuliahan Teknik Pemodelan SIstem Informasi
Kerangka Modul	:	• Konsep Pemodelan dan Pengenalan UML 2.0 • Modeling Business System: Use Case Diagram • Modeling Business System: Activity Diagram • Modeling Business System: Sequence Diagram • Modeling Business System: Package Diagram • Modeling Business System: Class Diagram • Modeling Business System: Activity DIagram (Internal View) • Modeling IT System: Structural ViewI • Modeling IT System: Structural View • Modeling IT System: Behavioral View • Modeling IT System: Interaction View • Component dan Deployment Diagram • Interaction Flow Modeling Language (IFML) • Fishbone Diagram

Disahkan,
Jakarta, Juli 2020
Ketua Program Studi Sistem Informasi
Fakultas Teknologi Informasi
Universitas Budi Luhur



Dr. Rusdah, S.Kom., M.Kom

DAFTAR ISI

Halaman

UNIVERSITAS BUDI LUHUR
FAKULTAS TEKNOLOGI INFORMASI

MODUL PERKULIAHAN #1
KONSEP PEMODELAN DAN PENGENALAN UML 2.0

Capaian Pembelajaran	:	1. Mahasiswa memahami pentingnya penggunaan model dalam pengembangan sistem informasi 2. Mahasiswa memahami tentang sejarah object oriented dan UML.
Sub Pokok Bahasan	:	1.1. Pengantar 1.2. Perkenalan Studi Kasus 1.3. Model, View, dan Diagram a. Apakah Model Itu? b. Kenapa Model Dibutuhkan? c. Purpose (Tujuan) dan Target Group dari Model d. Proses Analisis e. Diagram sebagai View 1.4. Information System dan IT System 1.5. Model dari Studi Kasus 1.6. Sejarah UML: Method dan Notation 1.7. Spesifikasi Kebutuhan a. Panduan untuk Pengambil Keputusan b. Verification 1.8. UML 2.0 a. Overview UML 2.0

		b. Efek pada Business System Model c. Efek pada IT System Model d. Efek pada System Integrasi 1.9. Kesimpulan 1.10. Soal Latihan
Daftar Pustaka	:	Grässle, P., Baumann, H. and Baumann, P., 2005. UML 2. 0 in Action: A Project-based Tutorial. Packt Publishing Ltd.

PENGANTAR

Unified Modeling Language (UML) memungkinkan untuk menggambarkan sistem dengan kata dan gambar. UML dapat diguankan untuk memodelkan berbagai jenis sistem: software system, business system, atau sistem lainnya. Yang paling penting adalah beberbagai jenis chart grafik – use case diagram dengan stick figure atau class diagram yang banyak digunakan. Sementara diagram ini bukanlah fundamental yang baru, penyatuan Bahasa modeling dunia adalah hal baru dengan UML, merupakan standar yang dikeluarkan oleh **Object Management Group (OMG),** asosiasi interneal yang memajukan standart terbuka dari aplikasi berbasis obyek.

Kebanyakan buku tentang UML menggambarkan hampir kesluruhan diagramnya, berdasarkan pengalaman telah dilihatkan bahwa pada dunia nyata sering kekurangan waktu, pengetahuan sebelumnya, atau motiviasi untuk berurusan dengan topic dengan intsitas yang dibutuhkan. Pada kasus ini, material tidak dapat dimengerti seluruhnya dan dilakukan aksi. Buku ini digunakan untuk kasus tersebut. Bagian UML dikumpulkan menjadi satu yang aplikasinya telah terbukti praktis. Dengan usaha sedikit, siapapun dapat memanfaatkan UML.

Beberapa alasan menggunakan UML sebagai bahasa modeling (modeling language):

1. Penyatuan dari terminology dan standarisasi dari notasi menuntun ke hal penting yaitu komunikasi mudah untuk semua pihak yang terlibat. UML memfasilitasi pertukaran dari model antara bagian atau perusahaan yang berbeda. Selain itu, juga mempermudah proses transfer dari proyek antara team proyek atau anggota team proyek.
2. UML berkembang sebagai kebutuhan dari perkembangan model. Karena UML adalah bahasa modeling yang sangat bagus, maka dapat dimulai dengan mengembangkan model sederhana atau sistem model kompleks dalam detail yang baik. Jika fungsionalitas dari UML tidak mencukupi, dapat diperluah dengan menggunakan stereotype
3. UML dibangun atas penggunaan yang banyak dan pendekatan yang sudah terbukti. UML tidak dirancang sebagai khayalan tetapi dikembangkan dari

permasalahan nyata dan bahasa modeling yang sudah ada. Hal ini menjamin kegunaan dan fungsionality dunia nyata.

4. UML didukung secara luas
5. Tawaran dengan basis UML untuk sistem software dapat dibandingkan lebih mudah.

Modul-Modul pada perkuliahan ini berdasarkan UML Versi 2.0, yang diadopsi oleh Object Management Group. Modul ini dibuat untuk ilmuwan komputer dan untuk orang-orang yang terlibat pada pengembangan proses dari sistem IT (IT System), seperti analis, pengambil keputusan, pengguna (user), dan ahli (expert). Ditunjukan bagaimana, dengan UML, model sederhana dari proses bisnis dan spesifikasi model dapat dibuat dan dibaca dengan usaha sedikit. Berdasarkan pengalaman dengan proyek, ditunjukan bahwa:

1. Seringnya hanaya komponen dari model yang dibuat
2. Sebagian besar waktu sistem keseluruhan *tidak* dimodelkan
3. Waktu yang sangat sedikit yang digunakan untuk pelatihan dalam bahasa modeling dan metodologi
4. Singkatnya, modeling tidak diberikan prioritas yang banyak.

Tentu saja sedikit proyek menggunakan model UML lengkap secara tepat. Tetapi, sebagian besar proyek menggunakan UML atau bahasa pemodelan lain, modeling tools, dan metode modeling hanya dalam porsi sedikit, jika ada. Sedangkan atusiasme dan motivasi sangat kuat diawal-awal sebuah proyek, modeling dan dokumentasi dari hasil sebuah model seringnya yang pertama menjadi korban untuk meningkatkan tekanan waktu sebagai pendekatan batas waktu.

Sayangnya, hal tersebut tidak dapat diubah. Mengingat keadaan ini, dicoba menggambarkan UML yang sangat sederhana, agar memungkinkan dapat menggunakan UML lebih efisien dan tepat hanya dengan investasi waktu sedikit.

Berdasarkan pengalaman menunjukan bahwa menguasai hanya beberapa elemen dari UML menuntun ke hasil yang lebih baik dibandingkan dengan pengetahuan dangkal

dari banyak elemen UML. Oleh karena itu pada modul ini hanya dipilih elemen yang perlu digunakan – secara subyektif. Tidak banyak elemen UML yang disebutkan, dan penjelasan dari elemen lainnya dengan bentuk sederhana. Walaupun hal ini bukan maksud awalnya, tetapi dapat mencerminkan pengalaman praktis.

Dengan perubahan dan penambahan, UML 2.0 mendukung pemodelan dari proses bisnis lebih baik. Peningkatan ukuran dari kosa kata UML 2.0 menunjukan bahwa tidak lagi sesuai untuk mendefinisikan hanya elemen tertentu dari bahasa, tetapi juga diperlukan untuk mendefinisikan penggunaan dari UML pada area khusus, seperti, integrase sistem atau data warehouse.

Pada modul akan ditekankan aspek tersebut di atas melalui penggunaan profil, selama mendefinisikan elemen bahasa (kosa kata/terminology) dari UML 2.0. Profil akan dirujuk pada tempat yang sesuai didalam modul.

Modul ini disusun sehingga dapat dibaca selama mengerjakan proyek, akan dimulai dengan pemodelan business system (sistem bisnis) dan proses bisnis. Kemudian lebih spesifik pada IT System (Sistem IT) yang dimasukan pada business system dan terakhir menggambarkan integrase dari Sistem IT kedalam lingkungannya. Tiga hal tersebut merupakan entitas independen. Dapat dibaca hanya pada bagian yang dibutuhkan pada proyek. Tetapi dalam kasus tertentu, harus membaca dari bagian prinsip dan background, dimana akan dikenalkan studi kasusnya. Modul ini juga akan menjelaskan beberapa istilah dasar dan konsep yang akan digunakan lebih lanjut.

Pada modul ini menggunakan studi kasus untuk menyampaikan pengetauan teoritis tentang UML. Studi kasus ini menyajikan satu tujuan untuk mengilustrasikan UML, semenjak banyak yang dapat dijelaskan dan dimengerti sepanjang penggunaan contoh daripada definisi abstrak. Pembaca seharusnya mendapat 'fee' untuk UML. Tentu tidak mungkin menggunakan setiap diagram dan setiap jenis dokumentasi dari seluruh studi kasus: hal ini akan menjadi diluar jangakauan modul dan tidak penting untuk pemahaman UML. Studi kasus – Passenger Service pada Airport fiksi yaitu Airport UML – tidak harus secara tepat mewakili passenger service: sudah dibuat

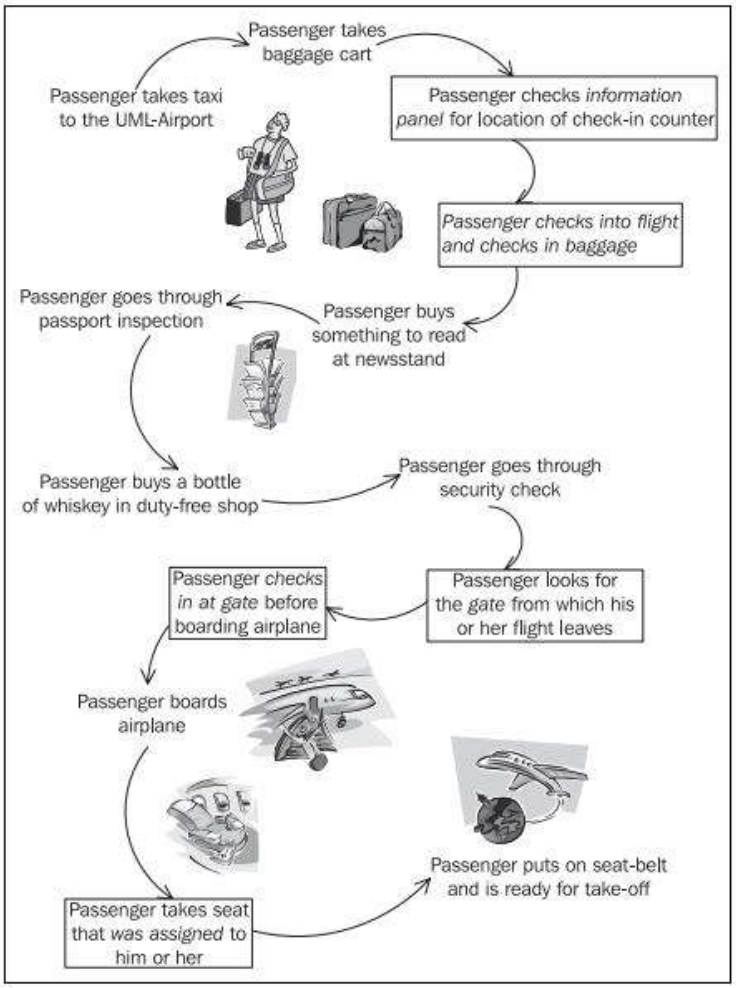
dengan bentuk sederhana. Tetapi, tidak menjadi efek negative dalam pemahaman UML.

Modul ini tidak menganggap pengetahuan UML yang sudah dimiliki atau pemrograman berorientasi obyek. Tetapi, diharapkan memiliki pengetahuan dasar dari pengembangan sistem IT.

PERKENALAN STUDI KASUS

Studi kasus yang dibahas adalah tentang bandara - UML Airport. Siapapun yang pernah dalam penerbangan tidak memiliki kendala dalam memahami contohnya.

Studi kasus dibatasi hanya pada area dari airport yang kontak lagsung dengan penumpang pesawat (passenger) selama keberangkatkan, artinya akan melihat lebih dekat mengenai **passenger check-in** dan **boarding**.



Gambar 1. Studi Kasus "Penumpang menggunakan pesawat untuk berlibur"

Gambar 1. menggambarkan bagaimana passenger service dapat dibedakan dengan area lain pada airport. Ditampilkan beberapa tahap yang dilalui oleh penumpang sampai penumpang duduk dalam pesawat, memakai sabuk pengaman, kemudian siap untuk terbang. Tidak semua tahap dilalui oleh penumpang berhubungan dengan passenger service. Tahap yang merupakan bagian dari passenger service dibingkai dan dicetak dengan huruf miring.

Urutan langkah seperti ini disebut dengan **skenario (scenario)**. Tetapi, penggambaran skenario merupakan satu dari banyak kemungkinan skenario lainnya. Pengecualian di bawah ini adalah kemungkinan yang terjadi untuk passenger check-in dan boarding:

1. Penumpang hanya memiliki satu bagasi carry-on
2. Penumpang tidak membeli apapun di kios koran (newsstand)
3. Penumpang terlambat sehingga harus segera check-in secepatnya
4. Penumpang kehilangan boarding pass
5. Penumpang tiba dengan pesawat dan harus berganti pesawat, artinya penumpang tidak meninggalkan area transit
6. Penumpang check-in, tetapi tertidur pada kursi yang tidak nyaman pada ruang tunggu, dan terlewat waktu keberangkatannya, walaupun sudah dipanggil berulang kali
7. Penumpang tidak lolos pemeriksaan passport karena passportnya sudah expired

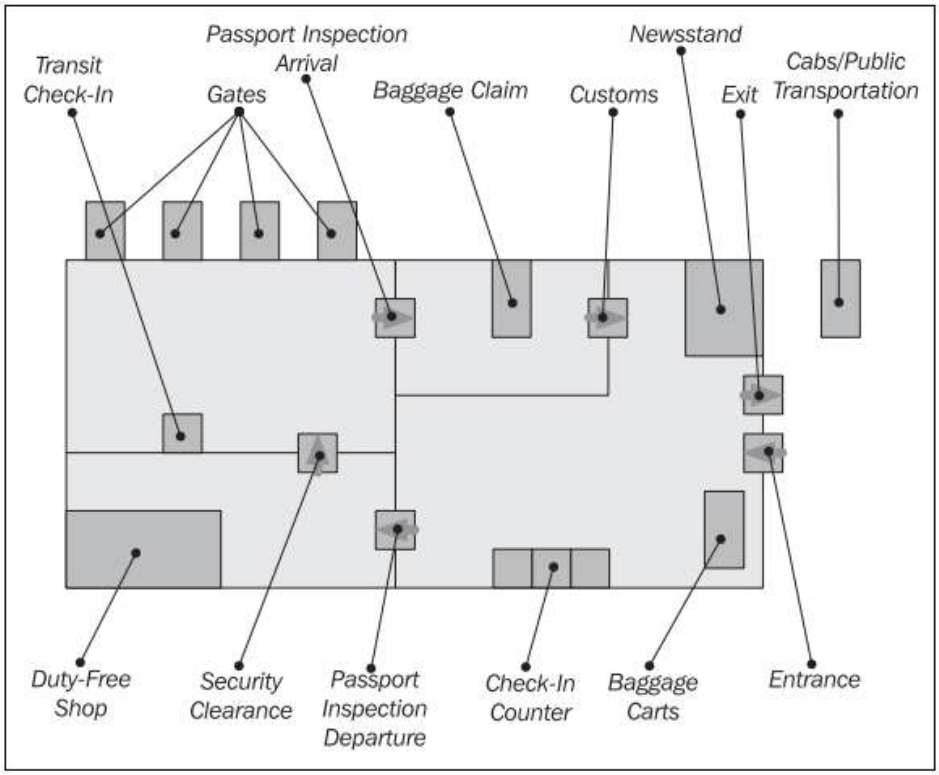
Silahkan dipertimbangkan manakah dari skenario diatas yang relevan pada keberangkatan penumpang dan juga apakah ada lagi yang lebih relevan dari yang disebutkan di atas.

Skema ilustrasi dari UML Airport pada Gambar 2 dapat membantu untuk memahami kejadian pada studi kasus lebih baik. Banyak area pada passenger service yang saling terkait satu dengan lainnya dalam passenger servcie. Seperti contohnya:

1. Penjualan Tiket
2. Toko Koran

3. Toko Duty-free
4. Pemeriksaan Passport/Imigrasi
5. Flight Control
6. Meja Informasi
7. Check-in Bagasi dan Transportasi

Passenger Service harus bertukar data dengan beberapa area tersebut. Selain itu juga Passenger Service harus berkomunikasi dengan area lain di Airport. Area tersebut akan dibicarakan pada diskusi mengenai Model Bisnis (Business Model) dan Model Integrasi Sistem (Model of System Integration). Oleh karena itu, kasus studi akan menjangkau lebih jauh pada modul lainnya.



Gambar 2. Bagan Ilustrasi dari UML Airport

UML Airport merupakan airport kecil dan studi kasus dibuat sederhana. Siapapun yang pernah dalam penerbangan dapat dengan mudah memahami contohnya.

Tujuan dari studi kasus adalah untuk menyediakan contoh yang jelas pada setiap modul. Beberapa detil dari studi kasus membutuhkan penjelasan lebih lanjut:

1. Tiket pesawat terdiri dari tiket aktual (actual ticket) dan dapat ditambahkan lebih dari 4 (empat) bagian. **Tiket (ticket)** adalah *booklet* kecil yang memiliki kupon terpisah untuk setiap bagian perjalanan. Contohnya, sebuah tiket dapat berisi kupon untuk penerbangan dari Zurich ke Frankfurt, satu untuk penerbangan dari Frankfurt ke London, dan satu untuk penerbangan balik (return flight) dari London ke Zurich. Setiap kali pada saat check-in kupon yang sesuai akan diganti dengan boarding pass. Tiket selalu dipegang oleh penumpang.
2. Penerbangan (flight) dengan Nomor Penerabangan (flight number) akan dibedakan. Seperti, **flight number** berisi LH433 atau LX016. Flight number merupakan penerbangan reguler yang terjadi pada waktu yang sudah ditentukan dari airport asal ke aiport tujuan. Sedangkan untuk **Flight,** contohnya LH435 pada 26 Agustus 2000 merupakan eksekusi dari flight number. Flight dapat dibatalkan karena cuaca buruk. Flight number digunakan selama maskapai penerbangan memberikan penerbangan tertentu secara regular.
3. Check-in dibagi menjadi 3 (tiga) pilihan:
    a. Normal Check-In dengan bagasi di konter normal check-in
    b. Express Check-In dengan bagasi di konter special check-in
    c. Automated Check-In tanpa bagasi di mesin check-in otomatis

MODEL, VIEW, DAN DIAGRAM

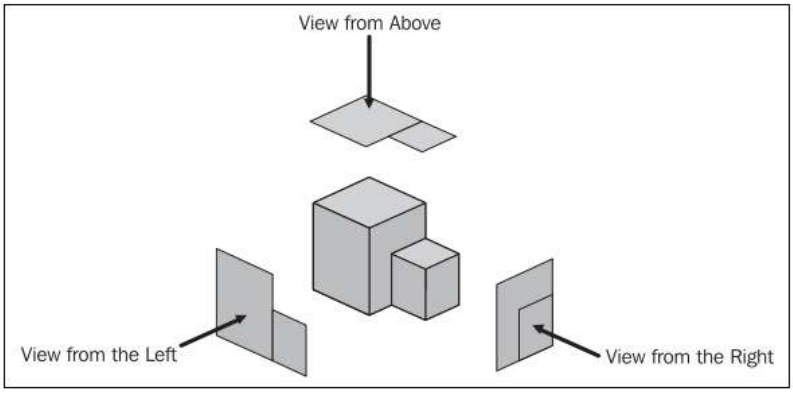
1.1. APAKAH MODEL ITU?

Model sering dibangun dalam konteks bisnis dan sistem IT (IT system) agar mudah memahami sistem yang ada atau sistem akan datang. Tetapi, sebuah model tidak pernah sesuai dengan kenyataan. Pemodelan selalu memiliki arti **menekankan (emphasizing)** dan **mengabaikan (omitting)**: menekankan detil penting, dan mengabaikan yang tidak relevan. Apakah yang dimaksud dengan penting dan apakah yang dimaksud tidak relevan? Tidak ada jawaban yang sama untuk pertanyaan ini. Melainkan, jawaban tergantung pada **apa (what)** tujuan dari model dan **siapa (who)** yang melihat dan membaca model.

Pertimbangkan tentang apa yang harus ditekankan (emphasized) atau diabaikan (omitted) pada model di bawah ini:

1. Model terowongan angin dari mobil
2. Model dari sebuah gedung dengan skala 1:50
3. Rencana rute dari sebuah kertea bawah tahan
4. Sebuah Peta
5. Sebuah struktur organisasi

Semakin banyak informasi yang seharusnya diberikan oleh model, akan menjadi lebih kompleks dan sulit. Sebagai contoh, Peta Eropa, yang didalamnya menyampaikan informasi mengenai politik, geologi, demografi, dan transportasi akan sulit dibaca. Solusi dari permasalah ini adalah dengan menyampaikan setiap jenis informasi pada satu peta. **Pandangan berbeda (different view)** akan terbentuk dengan pertimbangan. Pandangn ini saling berhubungan dalam berbagai cara. Secara umum, jika satu pandangan berubah, maka pandangan lain juga harus disesuaikan. Sebagai contoh, jika di Belanda mengklaim sebuah pulau dari Laut Utara, semua pandangan – yaitu semua peta – harus diperbaharui.



Gambar 3. Pandangan Berbeda (Different View) dari Sebuah Object

Sama juga seperti sebuah model bangunan. Jika sebuah sisi ditambahkan pada bangunan yang ada maka akan mempengaruhi berbagai pandangan, termasuk floor plan, pandangan luar (exterior view) yang berbeda, dan model 3D yang dibuat dari kayu. Gambar 3 merupakan ilustrasi bagan dengan pola skematik. Pada model studi kasus, secara spesifik mebahas tentang hubungan antara model pada modul.

Perbedaan pandangan antara setiap model akan dijelaskan menjadi 3 (tiga) model yaitu Modeling Business System, Modeling IT System, dan Modeling for System Integration.

1.2. KENAPA MODEL DIBUTUHKAN?

Sesuai dengan aturan umum, model dari sebuah sistem harus melakukan tugas berikut:

1. **Komunikasi (Communication)** antara semua pihak yang terlibat: agar dapat membangun sistem yang benar, sangat penting untuk semuah pihak yang terlibat memiliki pemikirian yang sama. Terutama sangat penting untuk setiap orang mengerti istilah yang digunkaan, customer setuju terhadap kebutuhan yang sama, pengembang mengerti akan kebutuhannya, dan keputusan yang dibuat masih dapat dimengerti beberapa bulan kedepan.
2. **Visualisasi (Visualization)** dari semua fakta untuk customer, ahli (experts), dan pengguna (user): semua fakta relevan yang dikumpulkan untuk kebutuhan sistem disampaikan sedemikian rupa sehingga semua yang berhubungan dapat mengerti. Tetapi, berdasarkan pengalaman di kehidupan nyata, terkadang terbentur dinding perlawanan ketika ingin berkomunikasi dengan diagram daripada teks. Sangat penting untuk mengatasi perlawanan ini. Dibaliknya sering khawatir hal-hal yang belum diketahui; dan awalnya diagram akan terlibat sedikit rumit. Oleh karena itu, modul berisi arahan bagaimana membaca setiap diagram.
3. **Verfikasi (Verification)** dari fakta sebagai istilah kelengkapan, konsistensi, dan kebenaran; Model (yang kurang lebih) dalam bentuk formal memungkinkan untuk memverifikasi fakta yang diperoleh untuk kelengkapan, konsistensi, dan kebenaran. Khususnya, penggambaran yang jelas dari keterkaitan memungkinkan untuk bertanya lebih spesifik, dan menjawabnya. Pertanyaan akan dibuat pada setiap diagram.

Jawablah pertanyaan di bawah ini, untuk diri sendiri:

1. *Kapankah terakhir kali anda merasa bahwa anda ada di posisi bertanya-tanya ketika mendiskusikan sebuah sistem?*
2. *Kapankah terakhir kali anda merasa bahwa anda berdiskusi dengan permasalahan yang sama berulang kali?*
3. *Kapankah terakhir kali anda berharap bahwa kesepakatan yang dihasilkan pada saat diskusi telah direkam?*

1.3. PURPOSE DAN TARGET GROUP DARI MODEL

Pada kehidupan nyata kita sering mengamati bahwa hasil dari sebuah model yang rumit, membosankan, dan mahal hanya hilang pada tumpukan kertas di meja seseorang. Kita akan bertanya-tanya kenapa hal itu bisa terjadi. Dua faktor yang sangat besar mempengaruhi hasil sebuah pemodelan: **untuk siapa (form whom)** model dibuat dan untuk **tujuan apa (what purpose)** model seharusnya digunakan. Jika kedua hal tersebut tidak didiskusikan dan ditetapkan secara cukup, maka beresiko membuat model yang tidak berisi mengenai hal terpenting untuk user. Dalam kata lain, jika detail tidak ditekankan (emphasized) dan dihilangkan (omitted) secara tepat, maka model yang dibuat tidak berguna.

Dalam menetapkan tujuan (purpose) dan sasaran kelompok (traget group) pertanyaan-pertanyaan di bawah ini harus terjawab:

1. Sebarapa besar **keahlian bisnis (business expertise)** yang bisa kita harapkan? Dapatkah kita mengasumsikan pengetahuan dasar (basic knowledge) dari subyek, atau apakah kita perlu menjelaskan kegiatan dan proses mendasar dari sebuah model?
2. Berapa jumlah **detail** yang dibutukan oleh target group? Tingkat kerumitan apa yang diizinkan oleh model? Jika proses dan sistem berubah terus-menerus, model dengan detail yang rinci sangat tidak realistis. Hal ini dikarenakan, sebagian besar waktu, sangat tidak mungkin untuk mempertahankan model tersebut dalam bentuk yang memuaskan. Model dengan detil yang kurang membutuhkan upaya lebih kecil untuk mengembangkan dan memperbaharui, tetapi juga kurang tepat.

3. Berapa lama **waktu (time)** yang dibutuhkan oleh target group untuk membaca dan menerjemahkan model? Cegah model anda menghilang dalam tumpukan kertas di meja kerja seseorang dengan memilih tingkatan yang tepat dari detail dan kerumitan; kalau tidak, tidak ada yang memiliki cukup waktu untuk membacanya.
4. **Bahasa (language)** apa yang dapat digunakan dalam model? Apakah target group mengerti istilah bisnis teknik (technical business)? Apakah mereka mengerti istilah IT? Mari kita perjelas dengan contoh yang mudah: jika sebuah botol diisi dengan air dan diberi label "AIR", pada hakekatnya setiap orang yang dapat membaca akan mengerti isi dari botol. Akan tetapi, jika botol diberikan label "H_2O" (walaupun ini benar) maka hanya akan menjangkau kelompok kecil, seperti, pekerja dari lab kimia. Tetapi, keuntungan tambahan adalah label tersebut menunjukan komposisi dari botol: hydrogen dan oksigen. Dari kedua kasus tersebut, harus diputuskan label apa yang lebih tepat untuk target group.
5. Apakah tingkat **keabstrakan (abstraction)** yang dapat dipilih? Keabstrakan yang kurang dari sebuah model, maka akan mudah dipahami dan jelas untuk user. Hal ini dikarenakan model dengan keabstrakan yang kurang lebih mendekati penggunaan aktual user dan bahasanya. Disamping itu, model dengan tingkat keabstrakan yang tinggi dapat digunakan kembali dan lebih mudah diubah ke sistem IT. Kita dapat membuktikan secara tepat bahwa model tersebut benar. Spesialis IT mungkin mengelola keabstrakan model tingkat tinggi lebih baik. Dilain pihak, pengguna akan terganggu jika ditanya untuk menangani model dengan keabstrakan tingkat tinggi.

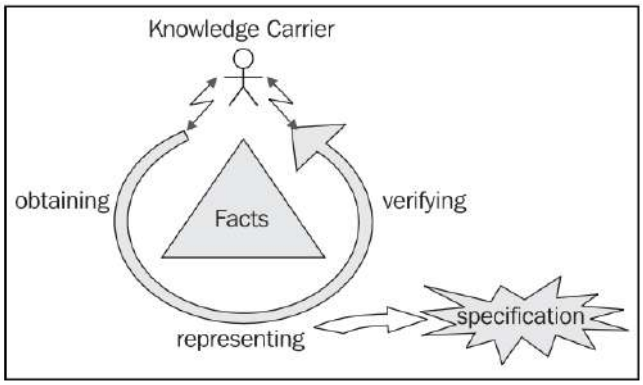
PRACTICAL TIPS

Kompromi (Compromises) harus terjadi antara tingkat keabstrakan, kejelasan, dan jumlah rincian (detail) yang digunkaan untuk sebuah model. Sangat dimungkikan untuk mengembangkan beberapa komponen model, dibedakan dalam bentuk formalitasnya dan detailnya, sehingga dapat memenuhi target group yang berbeda. Dengan cara ini komunikasi antara pembuat model, pelanggan (customer), user, dan pengembang (developer) dapat difasilitasi dengan lebih mudah. Penting untuk tidak 'berlebihan', tetapi menyesuaikan model ke target group dan kegunaannya.

Analisis (Analysis) atau **pola (pattern)** desain adalah contoh model yang menggambarkan desain umum dan metode pemodelan.

1.4. PROSES ANALISIS

Pada Gambar 4 menunjukan proses analisis, yang terdiri dari **mendapatkan (obtaining), menggambarkan (representing), memverifikasi fakta (verifying facts):**



Gambar 4. Proses Analisis

Ini merupakan pekerjaan dari seorang analis. Proses analisis menghasilkan spesifikasi yang berasal dari model dan gambaran lainnya. Analis bekerja dengan pembawa pengetahuan (knowledge carriers), seperti customer, users, dan domain experts:

1. Fakta **didapatkan (obtained)** dengan berkolaborasi antara analis dan domain expert dimana knowledge carriers menyumbang domain knowledge dan analis menyumbang methodological knowledge.
2. Fakta **digambarkan (represented)** dalam diagram dan dokumen, yang biasanya disiapkan oleh analis
3. Fakta **diverifikasi (verified)** hanya dengan knowledge carriers, karena hanya knowledge carrier yang dapat memutuskan jika fakta yang digambarkan adalah benar. Verfikasi sangatlah penting. Tanpa verifikasi dihasilkan diagram yang indah, tetapi kemungkinan besar fakta yang digambarkan adalah tidak benar. Dalam Bahasa singkat: pengembangan dari sebuah model tanpa verifikasi adalah sama sekali tidak berguna.

PRACTICAL TIPS

Tidak mungkin untuk mengembangkan dan memverifikasi model yang dapat digunakan tanpa menguasai teknik dasar dari sebuah topik. Dimana kita dapat menemukan **knowledge carriers** yang mengetahui sesuatu tentang sistem yang dibuatkan modelnya? Kita memiliki pengalaman yang bagus dengan kelompok orang di bawah ini:

1. Orang-orang yang terlibat dalam menjalankan, mengoperasikan, dan mengatur proses bisnis
2. User dari sistem IT yang terkait atau mirip.
3. Customer, sering menjadi knowledge carriers yang kritikal dan kreatif
4. Business Partners
5. Domain Experts
6. Management
7. External Observers

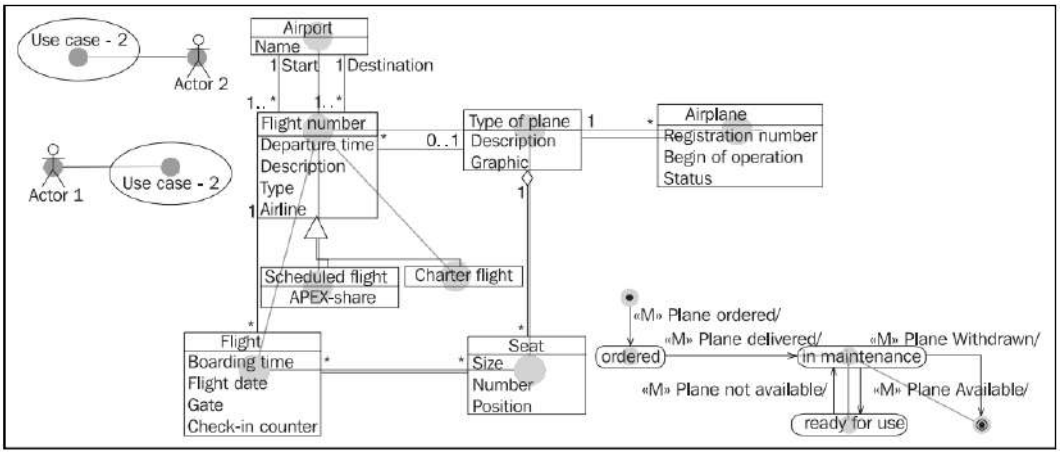
Beberapa teknik bermanfaat terbukti sangat berguna untuk menganalisa dan memahami proses bisnis:

1. Mengamati karyawan di tempat kerja
2. Berpartisipasi dalam meneliti proses bisnis
3. Menjadi peran orang luar (seperti customer)
4. Melakukan survey
5. Melakukan wawancara
6. Bertukar pikir dengan setiap orang yang terlibat
7. Berdiskusi dengan domain expert
8. Meninjau kembali form, dokumentasi, spesifikasi, buku panduan, dan perangkat kerja yang sudah ada
9. Menggambarkan struktur organisasi dan alur kerja manajemen (organization chart, dll)

1.5. DIAGRAM SEBAGAI VIEW

Setiap diagram UML tertentu berhubungan dengan satu **tampilan (view)** dari sebuah model sistem. Tergantung dari jenis diagram yang digunakan, aspek yang

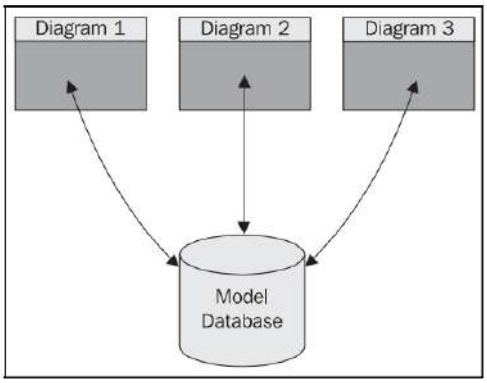
berbeda adalah antara menekankan (emphasized) atau menghilangkan (omitted). Semua tampilan yang berbeda digabungkan menghasilkan sebuah model sistem yang baik. Hampir sebagian besar UML diagram adalah **grafik (graphs)** seperti yang terlihat pada Gambar 5, mengartikan bahwa diagram terdiri dari elemen yang saling terkait melalui garis:



Gambar 5. Diagram Sebagai Grafik (Graphs)

Membaca diagram, harus mengetahui jenis garis elemen apa yang diizinkan dan apa artinya. Akan dijelaskan untuk diagram yang digunakan pada modul berikutnya.

Computer-aided software engineering (CASE) tools memperlakukan diagram UML sebagai tampilan (view). CASE tools menggunakan **database** dimana informasi mengenai model disimpan. Setiap diagram menampilkan – sebagai view – bagian dari informasi. Dengan cara ini, CASE tools membantu menjaga konsistensi setiap view. Jika, sebagai contoh, nama sebuah class diganti pada sebuah class diagram, maka statechart diagram dari class tersebut secara otomatis akan diubah:



Gambar 6. CASE tool sebagai Database

Pada Gambar 6 database model adalah apa yang secara fundamental membedakan sebuah CASE tools dari sebuah program grafis (graphical program). Diagram UML manapun dapat dihasilkan dengan mudah menggunakan kertas dan pensil atau graphical program. Tetapi dalam kasus ini, berbagai macam diagram tidak lebih hanyalah sebuah gambar. Hanya penggunaan sebuah CASE tool dengan database, berdasarkan spesifikasi UML, memperbolehkan pengumpulan, pengelolaan, dan perubahan konsisten dari informasi model. UML menyediakan database model sendiri: meta-model UML, komponen dari spesifikasi UML. Semua elemen yang ditemukan dalam diagram UML, maupun deskripsi dari elemen-elemen tersebut, terdapat pada meta-model UML. Dinyatakan, sebagai contoh, bahwa sebuah class dapat memiliki attribut dan method. "Data Model" dari UML ini sebagai Bahasa, merupakan dasar dari database model dari semua CASE tool UML. Sayangnya, banyak CASE tool yang masih memerlukan sumber daya, mahal, pengembangan yang buruk, rumit, dan memerlukan pelatihan yang mahal. Meskipun banyak kekurangan, kecuali untuk proyek kecil, penggunaan CASE tools sangatlah bermanfaat.

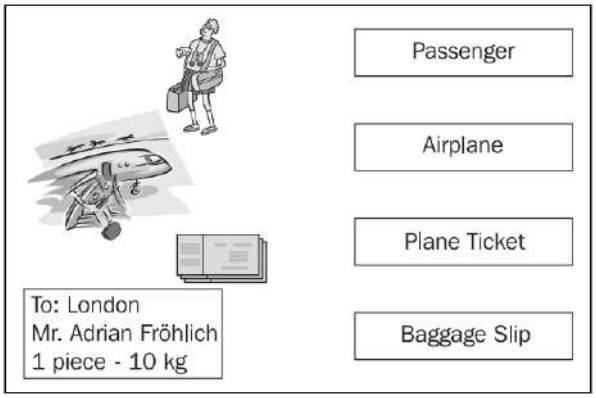
INFORMATION SYSTEM DAN IT SYSTEM

Hampir semua pekerjaan, bagian dari pekerjaannya adalah berurusan dengan informasi. Hal ini sudah terjadi selama ribuan tahun yang lalu dan salah satu alasan dibalik modul ini. Beberapa buku paling tua ditemukan di Eropa termasuk, contohnya, daftar stok dari istana Knossos di Crete. Jika kita mampu melihat manajer stok bekerja 3.500 tahun yang lalu, mungkin kita dapat memetakan proses bisnisnya yang digunakan pada saat itu. Kita dapat melihat bahwa orang-orang ini berurusan dengan pemasok dan pembeli, bahwa mereka bertukar barang, dan mereka menyimpan catatan dari kegiatan bisnisnya. Hal yang sama terjadi juga pada penjual minyak zaitun dari Roma 1.500 tahun kemudian, pada kantor datang Hanseatic di abad 15 Jerman Utara, atau pada Lloyd dari London pada awal abad terakhir.

Pada contoh di atas, sistem informasi yang lebih atau kurang kompleks telah digunakan untuk menangani tugas harian. Tujuan dari sistem informasi adalah untuk mengatur informasi yang dibutuhkan dalam menjalankan bisnis. Tentu saja, semua kegiatan ini terjadi tanpa menggunakan komputer. Sistem informasi didukung dengan

teknik lain seperti papan tulis kapur, sistem file yang besar, dan kartu indeks. Sekarang, dengan komputer memungkinkan untuk melaksanakan sistem informasi sebagai sistem IT (IT System). Hal ini menciptakan kemungkinan baru yang mungkin belum terpikirkan oleh penjualan minyak zaitun dari Roma. Tetapi pada dasarnya, intinya adalah masih menyediakan dan memproses data yang dibutuhkan untuk berurusan dengan proses bisnis setiap harinya. Secara umum akan dibahas tentang IT System pada modul, semenjak adanya anggapan bahwa sistem informasi yang dimodelkan dengan UML diimplementasikan oleh teknologi IT.

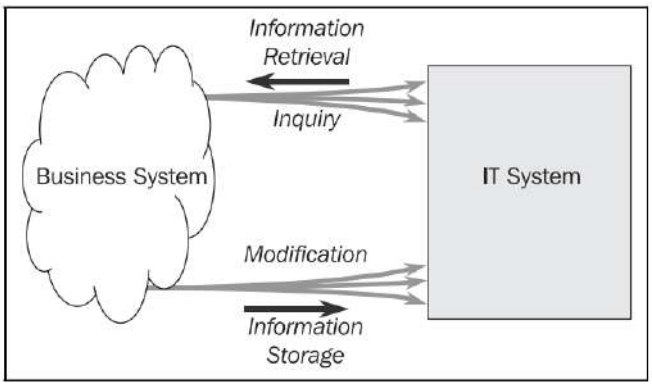
Pada studi kasus – **passenger services** pada UML Airport – karyawan pada bagian check-in berurusan dengan penumpang, tiket pesawat, dan penerbangan yang nyata. Disamping itu, terdapat penggambaran atau **gambar (image)** dari penumpang, tiket pesawat, dan penerbangan di sistem informasi. Image ini terdiri dari **informasi** tentang penumpang, tiket, dan penerbangan yang disimpan pada sistem informasi, dibutuhkan untuk menjalankan proses, seperti yang terlihat pada Gambar 7:



Gambar 7. Obyek dari dunia nyata dan penggambarannya (image)

Sistem IT (IT System) adalah sistem berbasis komputer – sebuah sistem yang menyediakan informasi yang dibutuhkan untuk melaksanakan proses bisnis tertentu, secara umum sebagai jawaban dari sebuah pertanyaan (query) oleh user. Tentu saja IT System harus 'diberikan' informasi, sehingga dapat menjawab pertanyaan.

Gambar 8 menunjukkan kerjasama antara sistem bisnis (business system) dan IT system secara skematik. Dalam framework dari proses bisnis sebuah business system, informasi diambil dari dan disimpan pada IT system:



Gambar 8. IT System

Teknik pemodelan diperkenalkan pada modul tidak hanya untuk pengembangan IT System dan, tetapi dapat juga digunakan kapanpun sebuah sistem informasi dibutuhkan untuk dianalisa. Penggambarannya diberikan dalam contoh kedua – selain studi kasus passenger service pada UML Airport – yang akan dibahas kembali pada tempat berbeda di modul.

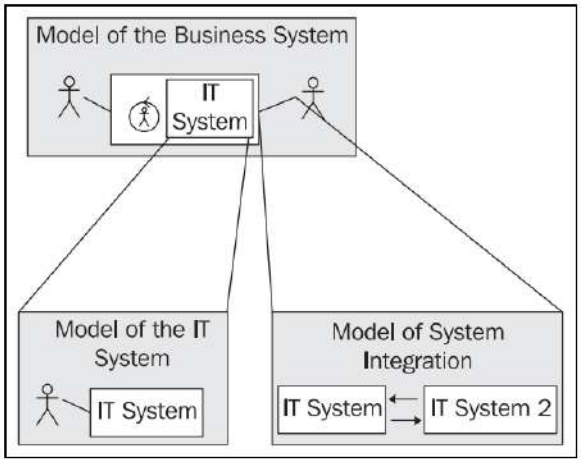
Contoh kedua adalah kantor perdangangan Hanseatic di abad pertengahan yang dimiliki oleh Mr. Hafenstein. (Liga Hanseatic merupakan aliansi serikat pedangan yang kuat di kota Jerman Utara dan Baltic yang mengontrol perdagangan pada daerah tersebut selama abad pertengahan). Supervisor pada kantor perdangann sangat setia dan rajin yaitu Sekretaris Hildebrandt. Kantor perdangan memiliki beberapa buku, seperti buku harian, buku besar penjualan, dan buku indeks customer. Setiap buku dipegang oleh penganggung jawab berbeda. Tidak ada orang lain, selain dari petugas yang bertanggung jawab diijinkan untuk membuat perubahan pada buku, dan hanya petugas yang mengetahui tepatnya dibagian mana informasi tertentu di simpan.

Pada istilah kita, kantor, termasuk sekretaris Hildebrandt, petugas, dan buku, menyusun sistem informasi. Dengan bantuan dari contoh ini ditampilkan perbedaan pada modul, walaupun sistem informasi *dapat* diimplementasikan sebagai IT system dengan bantuan teknologi komputer, secara konsep hal ini tidak ada kaitannya dengan komputer. Sebagai gantinya, sistem informasi dapat diwujudkan dengan berbagai cara.

MODEL DARI STUDI KASUS

Pada studi kasus, dibangun 3 (tiga) model dari sistem berbeda:

1. Model dari Sistem Bisnis (Business System) menggambarkan passenger service, artinya bisnis sekitar IT System. Berurusan dengan proses bisnis, penumpang, rekan bisnis, karyawan, dll.
2. Model dari Sistem IT (IT System) menjelaskan IT system yang dibangun untuk passenger service. Model dari business system passanger service disajikan sebagai dasar dari model IT system.
3. Model dari Integrasi Sistem (System Integration) menggambarkan penggabungan menjadi sebuah lingkungan, terutama sebagai gerbang ke dunia luar. Disini business system passenger service juga disajikan sebagai dasar dari System Integration.



Gambar 9. Model dari Studi Kasus

Ketiga model dibutuhkan untuk membangun dan menyatukan IT system; hanya model IT system saja tidaklah cukup. Hal ini berlaku tidak hanya untuk studi kasus kita saja, tetapi untuk semua kasus.

Dapat dilihat pada Gambar 9 bahwa model dari business system menyediakan dasar dari model lainnya. Dengan ini, dapat menjelaskan dasar untuk bekerja untuk semua yang terlibat dalam proyek. Karena hal ini, merupakan keuntungan yang sangat bagus untuk menggunakan **unified modeling language,** yang dapat dimengerti oleh orang-orang dari departemen berbeda dan juga dari teknologi informasi. Hal ini memungkinkan pertukaran secara halus dari model antara beberapa area. Dan juga

secara signifikan memudahkan verifikasi model. Kita yakin bahwa fungsi UML sebagai penghubung yang memiliki kemampuan untuk memperkecil perbedaan antara kebutuhan teknis dan karakteristik kinerja aktual dari IT system.

SEJARAH UML: METHOD DAN NOTATION

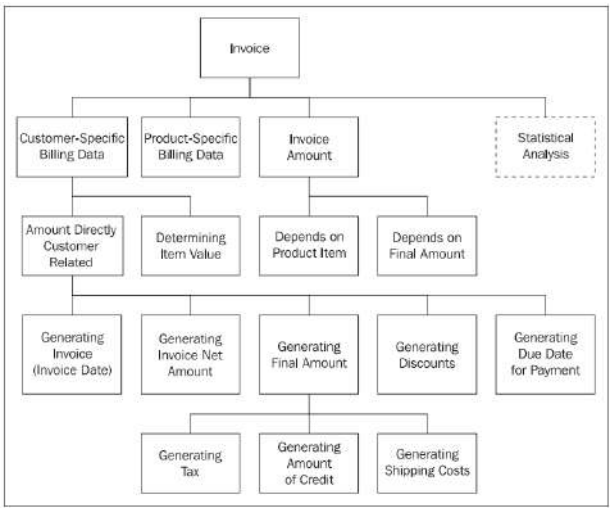
Dalam sejarah singkat, teknologi informasi telah menghasilkan banyak **metode** dan **notation**. Terdapat metode dan notation untuk desain, struktur, proses, dan penyimpanan dari informasi. Selain itu juga ada metode untuk perencanaan (planning), model (modeling), implementasi (implementation), assembly, pengujian (testing), dokumetasi (documentation), penyesuaian (adjustment), dll dari sistem. Beberapa konsep yang digunakankan adalah fundamental, oleh karena itu, konsep dapat ditemukan diluar area teknologi informasi. Salah satu contoh adalah inheritance, yang ada pada alam, tetapi juga merupakan landasan dari pemrograman berorientasi obyek.

Sampai tahun 1970-an, pengembang software melihat pengembangan dari software sebagai sebuah usaha artistic. Tetapi karena sistem makin lama makin kompleks, pengembangan software dan pemeliharannya tidak lagi dapat dikuasasi dengan pendekatan kreatif-individu (creative-individual). Akhirnya pendekatan ini menuntun ke **krisis software (software crisis)**

Krisis software menuntun ke **pendekatan teknik (engineering approach)** atau software engineering dan pemrograman terstruktur. Metode dikembangkan untuk menyusun sistem dan untuk desain proses, pengembangan, dan pemeliharaan. Pendekatan berorientasi proses, seperti contoh metode **Hierarchy Input Processing Output (HIPO)**, menekankan pada fungsi dari sistem. Dengan metode ini keseluruhan sistem dibagi menjadi komponen kecil melalui penguraian fungsi.

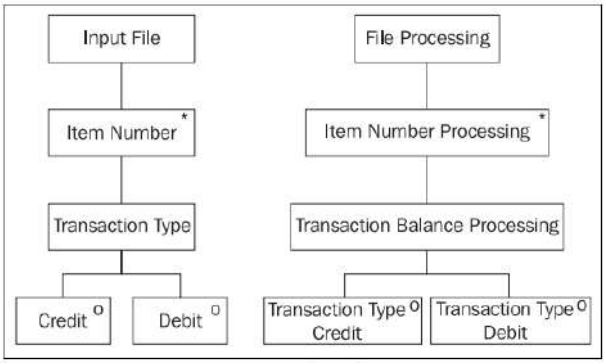
Pada Gambar 10 diberikan gambaran visual (diagram hirarki) dari sebuah sub-fungsi dalam contoh invoice. Sebuah sekma input-proses-output digambarkan pada setiap elemen fungsi.

Pada waktu yang sama, pendekatan berorientasi data-struktur dikembangkan, seperti metode Jackson, dimana struktur program berasal dari tampilan grafis struktur data.



Gambar 10. Diagram HIPO

Gambar 11 menampilikan, pada kolom bagian kiri, struktur dari data set inventory. Pada kolom bagian kanan menampilkan struktur program yang berasal dari struktur data:



Gambar 11. Diagram Jackson

Pada semua metode dan notation, sistem dibagi menjadi dua bagian – bagian data dan prosedur. Hal ini sangat dikenal pada Bahasa pemrograman yang lama seperti COBOL. Flow-chart data, structure chart, diagram HIPO, dan diagram Jackson digunakan untuk menggambarkan jangakauan dari fungsi. Secara alamiah, metode awal ini menekankan pada pengembangan dari sistem baru.

Pada tahun 1980-an, analisis terstruktur klasik dikembangkan lebih lanjut. Pengembang menghasilkan entity relationship diagram untuk pemodelan data dan Petri net untuk pemodelan proses.

Selama sistem menjadi semakin kompleks, sistem sudah tidak lagi dapat didesain "dari coreatan" ("from scratch"). Properti, seperti maintainability dan re-usability, menjadi lebih penting. Bahasa pemrograman berorientasi obyek dikembangkan, dan dengan Bahasa pemrograman tersebut, Bahasa pemrograman berorientasi obyek muncul pada tahun 1970-an, 1980-an. Pada tahun 1990-an, publikasi pertama tentang analisa berorientasi obyek dan desain berorientasi obyek mulai diketahui public. Pertengahan tahun 1990-an, sudah ada lebih dari 50 metode berbasis obyek, dan juga banyaknya format desain. **Unified** modeling language sepertinya sangat diperlukan.

Diawal tahun 1990-an, metode berbasis obyek oriented dari Grady Booch dan James Rumbaugh sudah banyak digunakan. Pada bulan oktober 1994, Rational Software Corporation (bagian dari IBM sejak Februari 2003) memulai pembuatan dari unified modeling language. Pertama, mereka menyetujui adanya standarisasi dari notation (Bahasa), karena ini sepertinya kurang lebih rumit dari standarisasi metode. Dalam mengurangi kerumitannya, mereka menyatukan **Metode Booch (Booch Method)** dari Grady Booch, **Object Modeling Technique (OMT)** oleh James Rumbaugh, dan **Object-Oriented Software Engineering (OOSE)** oleh Ivar Jacobsen, dengan elemen dari metode lain dan menerbitkan notasi baru ini dengan nama **UML** versi 0.9. Tujuannya adalah bukan untuk memformulasikan notasi baru, tetapi untuk beradaptasi, untuk memperluas, dan untuk menyederhanakan yang sudah ada dan menerima jenis diagram dari beberapa metode berbasis obyek seperti class diagram, Use Case Diagram oleh Jacobson, atau Statechart Diagram oleh Harel. Maksud dari penggambaran yang digunakan pada pemodelan terstruktur diterapkan pada UML. Seperti, UML activity diagram, sebagai contoh, dipengaruhi oleh susunan dari data flowchart dan Petri nets.

Yang menarik dan baru pada UML adalah bukan isinya, tetapi standarisasi menjadi satu unified language dengan makna yang didefinisikan secara formal.

Perusahaan terkenal, seperti IBM, Oracle, Microsoft, Digital, Hewlett-Packard, dan Unisys diikutsertakan dalam pengembangan lebih lanjut dari UML. Tahun 1997, UML Versi 1.1 telah diserahkan dan disetujui oleh OMG. UML Versi 1.2, dengan adaptasi editorial, direleas pada tahun 1998, diikuti oleh versi 1.3 satu tahun kemudian, dan UML 1.5 pada Maret 2003.

Pengembang telah bekerja pada versi 2.0 dari UML semenjak tahun 2000, dan telah disetujui sebagai Final Adopted Spesfication oleh OMG pada Juni, 2003. Ketika buku yang merupakan sumber pembuatan modul dicetak pada Juni, 2005 tahap final adopsi oleh OMG sebagai **Sepesifikasi yang tersedia (Availabel Spesification)** belum diselesaikan.

SPESIFIKASI KEBUTUHAN

Model dari system yang akan dikembangkan tersusun dari bagian utuh setiap spesifikasi kebutuhan. Modul menyediakan dasar yang untuk memperkuat pengembangan dari modelnya. Sangat disayangkan, tidak ada resep global untuk spesifikasi kebutuhan. Sebaliknya, pilihan dan tingkatan detail dari model bergantung pada berbagai macam factor. Berdasarkan pengalaman menunjukkan bahwa 3 (tiga) hal di bawah ini merupakan hal yang sangat penting:

1. Siapa yang menentukan?
2. Untuk siapa hal tersebut ditentukan?
3. Apa yang harus ditentukan?

1.1. PANDUAN UNTUK PENGAMBIL KEPUTUSAN

Model dan view yang disediakan pada modul pada dasarnya merupakan block bangunan yang dapat dipilih model yang dibutuhkan untuk spesifikasi kebutuhan. Table 1. di bawah ini dapat membantu dalam membuat pilihan tepat dari model dan view:

Table 1. Tabel Panduan untuk Decision Making

Model (What)	View	Originator (Who)	Target Audience (for Whom)	Purpose (for What)
Business System	External View	User Agent	User Agent	Business Documentation
			IT Agent	Basis for IT System Specification
	Internal View	User Agent	User Agent	Business Documentation, Description of Procedures
			IT Agent	Basis for IT System Specification
IT System	External View	User Agent	IT Agent User Agent	Requirements of an IT System
	Structural View	IT Agent	IT Agent	IT System Specification
	Performance View	User Agent IT Agent	IT Agent	IT System Specification
	Interaction View	User Agent IT Agent	IT Agent	IT System Specification
System Integration	Process View	User Agent	IT Agent	IT System Integration Specification
	Static View	IT Agent	IT Agent	IT System Integration Specification

1.2. VERIFICATION

Semua view pada modul menggambarkan sebuah model yang mendokumentasikan kebutuhan dari sudutpandang user. Artinya model dan view yang digunakan:

1. Hanya dapat dibuat bekerjasama dengan users agent
2. Hanya dapat diverifikasi oleh user agent dengan menghormati kebenaran dari isi.

Walaupun yang dikembangkan adalah pemodelan dari IT System untuk target audiensi yaitu IT agent, kita tidak dapat melakukannya tanpa user agent, yang harus menyediakan kebutuhan dan memverifikasi model. User agen mewakili sudut pandang user dan knowledge carrier dari user domain.

Semenjak berabagai macam kelompok terlibat dalam pengembangan dan verifikasi dari spesifikasi kebutuhan, sangatlah penting untuk menggunakan unified modeling language, untuk menghindari kesalahpahaman dan salah pengertian.

UML 2.0

1.1. OVERVIEW UML 2.0

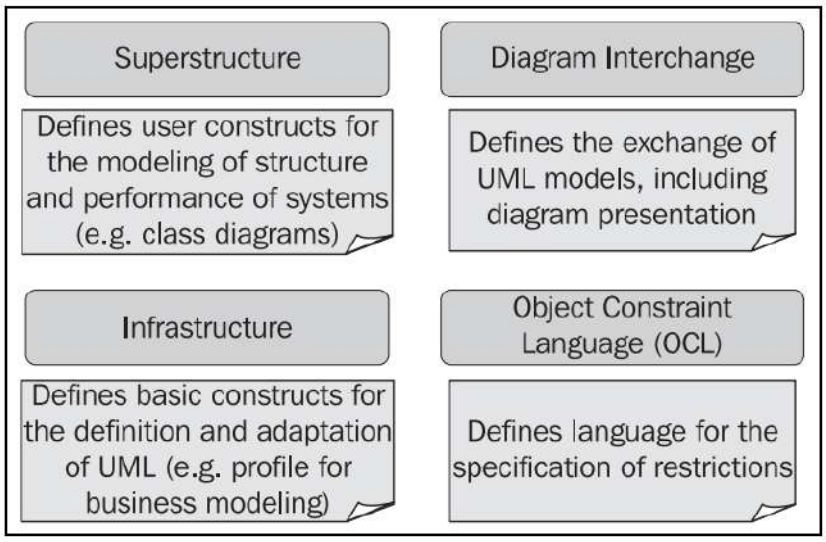
Modul ini berdasarkan versi terbari dari UML - UML 2.0. Pada versi 2.0, struktur dan dokumentasi dari UML diperbaiki secara menyeluruh. Sekarang ada dua dokumentasi tersedia untuk menggambarkan UML:

1. UML 2.0 *Infrastructure* menjelaskan konstruksi dasar dari Bahasa yang didasari oleh UML. UML ini tidak hanya relevan secara langsung bagi user UML (yang membaca), tetapi juga lebih diarahkan ke pengembangan dari perangkat model (modeling tools).
2. UML 2.0 *Superstructure* menjelaskan konstruksi user dari UML 2.0, artinya elemen dari UML yang digunakan oleh users pada tingkat langsung.

Diantara hal lain, revisi dari UML dilakukan untuk mengejar tujuan berikut:

1. Merestrukturisasi dan menghaluskan UML sehingga kegunaan (usability), penerapan (implementation), dan adaptasi (adaptasi) telah menjadi sederhana.
2. Infrastruktur UML diharapkan:
   a. Menyediakan inti meta-language yang dapat digunakan kembali (reusable), UML mana yang dapat menjelaskan dirinya sendiri.
   b. Menyediakan mekanisme untuk penyesuaian dari bahasa
3. Superstruktur UML diharapkan:
   a. Mengutamakan pendukung yang lebih baik untuk pengembangan berbasi component (component-based)
   b. Meningkatkan konstruksi dari spesifikasi arsitektur
   c. Menyediakan pilihan lebih baik untuk modeling tingkah laku.

Tambahan dari spesifikasi proposal UML Infrastructrue dan UML Superstructure, proposal terpisah diterbitkan untuk **Object Constraint Language (OCL)** baru dan juga untuk Diagram Interchange. Bersama, kedua hal tersebut menyusun package UML 2.0 menjadi lengkap seperti yang terlihat pada Gambar 12.



Gambar 12. Package UML 2.0 Lengkap

UML 2.0, sebagai kesatuan, lebih luas dan lebih kompleks dibandingkan versi sebelumnya. Tingkat dari dokumentasi UML juga lebih meningkat. Sementara dokumentasi dari UML 1.5, termasuk OCL, terdiri dari 730 halaman, dokumentasi dari UML 2.0 juga termasuk OCL, berisi sekitar 1050 halaman.

Walaupun bagian dari dokumentasi tidak memperhatikan user 'normal' UML, bagi anggota dari proyek pengembangan software, membaca pekerjaan lengkap sangatlah tidak realistis. Hal ini tidak hanya dikarenakan jumlah halaman, tetapi juga karena jumlah dan kompleksitas dari konstruksi UML. Karena hal ini, pengurangan dari konstruksi UML dibutuhkan untuk pekerjaan proyek sehari-hari lebih dibutuhkan dibandingkan dengan versi sebelumnya.

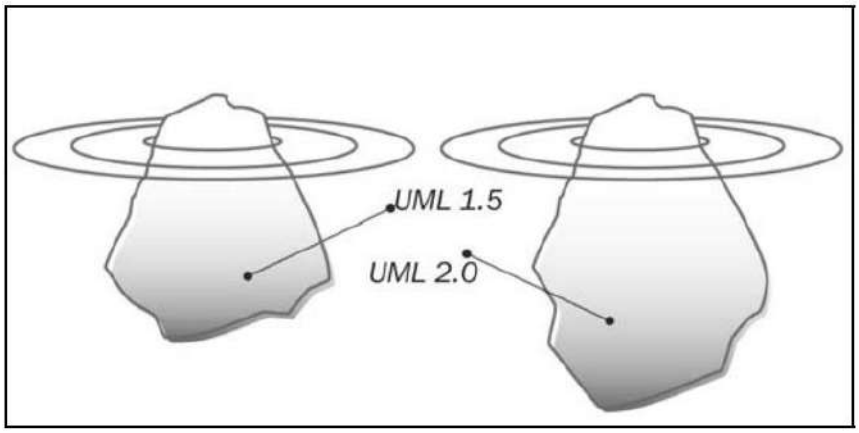
Dari sini dihasilkan dua kesimpulan dari buku yang menjadi sumber pembuatan modul.

Konsep dari buku tersebut adalah menampilkan gambar sederhana dari UML. Hal ini bahkan menjadi lebih penting dengan bertambahnya cakupan dari UML, semenjak aksesibilitas dari UML tidak menjadi lebih besar dengan versi 2.0.

Sayangnya, banyak fitur baru dari UML 2.0 hanya sedikit atau tidak ada pengaruh pada tingkatan detail di modul. Karena itu, hanya ada sedikit perubahan dibandingkan

edisi buku yang sama dari jerman. Batasan cakupan dari modul memastikan stabilitas menuju perubahan dari versi UML.

Secara sadar pada modul hanya ditampilkan puncak dari gunung es, sedangkan bagian yang tersembunyi di bawah air menjadi semakin besar. Lebih dari sebelumnya, opini yang ada bahwa puncak dari gunung es (lihat Gambar 13) cukup untuk pembaca modul – anggota dari tim proyek IT – yang mengerti UML cukup untuk menggunakannya dalam proyek:



Gambar 13. Gunung Es UML

Pada moudul juga akan ditunjukan kemungkinan baru yang dibuka oleh UML 2.0. Salah satu tujuan dari UML 2.0 adalah ketentuan formal dan ketentuan semantic yang sepenuhnya. Jika kemungkinan ini dimanfaatkan untuk pengembangan model, kesesuaian sistem dapat dihasilkan dari modelnya. Hal ini menghasilkan keuntungan sebagai berikut:

1. Model yang digambarkan dengan UML mencerminkan sistem sebenarnya
2. Dimungkinkan untuk memperbaiki kesalahan dalam model lebih awal dan berkelanjutan
3. Langkah tengah seperti mengubah kode diluar dari desain model dapat dihilangkan
4. Dimungkinkan untuk membuat model yang sama dijalankan pada platform berbeda (perangkat keras dan juga perangkat lunak)

Meskipun demikian, ada harga yang harus dikeluarkan dari keuntungan ini. Penting untuk mendapatkan pengertian yang mendalam dan akurat dari UML, dan harus memberikan usaha yang besar dalam pengembangan model.

1.2. EFEK PADA BUSINESS SYSTEM MODEL

Beberapa perubahan yang dilakukan dalam kinerja membuat model meningkatkan kemungkinan pembuatan business system model. Pertama, diberikan sebuah contoh dari beberapa perubahan dan perkembangan.

Activity diagram tidak lagi bagian special dari statechart diagram. Awalnya, fakta ini tidak relevan untuk pengguna UML normal. Tetapi, tambahan dari otonomi baru dalam meta-model, dilakukan beberapa perbaikan dan perubahan lain:

Hingga saat ini, langkah terpisah pada activity diagram ditunjuk sebagai aktifitas. Sekarang seluruh diagram disebut **aktifitas (activity)**, dimana langkah yang sebelumnya disebut aktifitas sekarang disebut sebagai **aksi (action).** Sebuah aksi dapat memanggil operasi utama dan juga aktifitas lainnya. Hal ini memungkinkan modulasi yang fleksibel pada top-down view dari model.

Sebuah divisi tidak perlu harus di sinkronisasikan.

Sebuah aktifitas dapat memiliki lebih dari satu initial state. Dengan ini, beberapa kegiatan dapat dimulai dengan waktu bersamaan.

Paramter input dan output dapat ditambahkan pada aktifitas.

Salah satu perbaikan yang dilakukan pada sequence diagram adalah penambahan hal yang disebut **operator**. Operator ini dapat memungkinkan untuk mengelompokan (package) beberapa aksi/aktifitas dalam sequence diagram. Misalnya, operator dapat digunakan untuk merujuk ke sequence diagram lain atau sequence diagram berdiri sendiri. Operator yang tepat dapat menggambarkan iterasi. Dengan operator yang baru dikenalkan, sequence diagram sekarang mendukung top-down view.

Sekarang OCL merupakan bagian yang diturunkan dari UML. OCL dapat digunakan untuk menggambarkan perjanjian, invariant, kondisi sebelum (precondition), dan kondisi sesudah (post condition) dalam model UML, sehingga memungkinkan pemodelan yang lebih tepat dari business system dan proses bisnis.

1.3. EFEK PADA IT SYSTEM MODEL

Diagram yang akan digunakan pada modul di view berbeda dari IT System tidak terjadi perubahan yang signifikan.

Perubahan terbesar terjadi pada notasi dari sequence diagram. Pada Sequence Diagram, diantara hal lainnya, referensi interaksi tersedia sebagai konstruk dari modularisasi. Tetapi, tidak ada perubahan tentang tujuan dan fungsi dari sequence diagram pada tingkatan detail yang digunakan pada modul. Hal yang sama pada Class Diagram dan Case Diagram.

Statechart diagram mengalami perubahan yang paling menarik untuk pemodelan IT System: titik koneksi memperbolehkan, sebagai contoh, modulasi terbaik dari statechart diagram. Tetapi, hal ini diputuskan untuk tidak menggunakan elemen bahasa ini pada pendekatan serhana pada UML.

1.4. EFEK PADA SYSTEM INTEGRASI MODEL

Tentu saja, perbaikan pada modeling tingkah laku (behavioral modeling) juga memiliki efek pada proses view di system integrasi model. Perbaikan signifikan adalah kemampuan untuk menambahkan parameter input dan output ke aktifitas.

Hampir tidak ada perubahan dilakukan pada static view, tujuan dari obyek bisnis pada class diagram.

Tambahan dari perubahan yang dilakukan pada framework UML 2.0, profil UML untuk **Enterprise Application Integration (EAI)** adalah meningkatkan kepentingan dalam ruang lingkup integrasi sistem. Selain dari operasi dasar yang dibutuhkan pada integrasi sistem, ditampilkan data-model dari beberapa jenis Bahasa pemrograman

yang bukan berorientasi obyek. Tetapi, hal ini tampil pada tingkatan mendetail yang tidak berpengaruh pada modul ini.

KESIMPULAN

1. Studi kasus yang digunakan sebagai contoh pada modul adalah UML Airport.
2. Pemodelan harus memiliki arti emphasizing dan omitting: menekankan detail penting, dan mengabaikan yang tidak relevan.
3. Model dari sebuah sistem harus melakukan tugas:
    a. Komunikasi
    b. Visualisasi
    c. Verifikasi
4. Tiga proses analisis:
    a. Obtaining
    b. Representing
    c. Verifying facts
5. Tiga pemodelan pada studi kasus:
    a. Pemodelan Sistem Bisnis (Modeling Business System)
    b. Pemodelan Sistem IT (Modeling IT System)
    c. Pemodelan Integrasi Sistem (Modeling System Integration)
6. Dua dokumentasi untuk menggambarkan UML
    a. UML 2.0 Infrastructure
    b. UML 2.0 Superstructure
7. Keuntungan menggunakan UML:
    a. Model yang digambarkan dengan UML mencerminkan sistem sebenarnya
    b. Dimungkinkan untuk memperbaiki kesalahan dalam model lebih awal dan berkelanjutan
    c. Langkah tengah seperti mengubah kode diluar dari desain model dapat dihilangkan
    d. Dimungkinkan untuk membuat model yang sama dijalankan pada platform berbeda (perangkat keras dan juga perangkat lunak)
8. Bagi pengguna normal, UML 2.0 tidak merubah versi UML sebelumnya secara besar, tetapi memperlihatkan perbaikan pada konsep yang sudah ada. Sangat

disarankan untuk menggunakan UML 2.0 untuk model kedepannya. Dilain pihak, sangatlah memungkinkan untuk tetap menggunakan konstruksi yang sudah ada dan model UML berdasarkan versi sebelumnya. Untuk proyek yang mendatang keuntungan (terutama modeling) harus dibandingkan dengan kerugian (pekerjaan tambahan).

SOAL LATIHAN

1. Di bawah ini yang BUKAN merupakan alasan penggunaan UML sebagai modeling language adalah _____
   a. Penyatuan dari terminologi dan standarisasi dari notasi untuk mempermudah komunikasi pada semua pihak yang terkait
   b. Dapat digunakan untuk mengembangkan model sederhana hingga kompleks
   c. Sebagai fasilitias pertukaran dari model antara bagian atua perusahaan yang berbeda
   d. Dapat membuat model tidak sesuai permasalahan nyata
2. Asosiasi yang menentukan standar UML adalah _____
   a. Object Management Group (OMG)
   b. Group Management Object (GMO)
   c. Object Contstraint Language (OCL)
   d. Workflow Management Coalition (WMC)
3. Dalam pemodelan memiliki arti *emphasizing* dan *omitting*, artinya _____
   a. Menekankan yang tidak relevan, mengabaikan detil penting
   b. Menekankan detil penting, mengabaikan yang tidak relevan
   c. Menekankan yang relevan, mengabaikan detil penting
   d. Menekankan detil tidak penting, mengabaikan hal relevan
4. Agar semua pihak yang terlibat memiliki pemikiran yang sama seperti pengertian istilah yang digunakan, kebutuhan yang diperlukan adalah tugas model pada sebuah sistem sebagai _____
   a. Komunikasi
   b. Visualisasi
   c. Verifikasi

d. Integritas

5. Semua fakta relevan untuk kebutuhan sistem harus dapat dimengerti oleh customer, experts, dan pengguna merupakan tugas model pada sebuah sistem sebagai _____
   a. Komunikasi
   b. Visualisasi
   c. Verifikasi
   d. Integritas
6. Semua fakta harus diperoleh kelengkapan, konsistensi, dan kebenarannya merupakan tugas model pada sebuah sistem sebagai _____
   a. Komunikasi
   b. Visualisasi
   c. Verifikasi
   d. Integritas
7. Dua faktor yang sangat mempengaruhi hasil sebuah pemodelan adalah _____
   a. Untuk apa tujuan model, siapa yang melihat dan membaca model
   b. Untuk apa melihat dan membaca model, siapa yang dituju untuk pemodelan
   c. Untuk siapa model dibuat, tujuan apa model seharusnya digunakan
   d. Untuk siapa model seharusnya digunakan, tujuan apa model dibuat
8. Dalam menetapkan tujuan (purpose) dan sasaran kelompok (target group) harus menjawab beberapa pertanyaan di bawah ini, KECUALI _____
   a. Seberapa besar keahlian bisnis (business expertise) yang diharapkan?
   b. Berapa jumlah detail yang dibutuhkan oleh target group?
   c. Berapa lama waktu (time) yang dibutuhkan oleh target group untuk membaca dan menerjemahkan model?
   d. Bahasa apa yang tidak digunakan dalam model?
9. Proses analisis terdiri dari obtaining (mendapatkan), representing (menggambarkan), verifying facts (verifikasi fakta). Obtaining artinya _____
   a. Melakukan verifikasi fakta kepada knowledge carrier
   b. Menggambar fakta dalam bentuk digaram dan dokumen
   c. Mendapatkan fakta dengan berkolaborasi antara analis dan domain expert.
   d. Menentukan tingkat keabstrakan proses

10. Proses analisis terdiri dari obtaining (mendapatkan), representing (menggambarkan), verifying facts (verifikasi fakta). Representing artinya _____
    a. Melakukan verifikasi fakta kepada knowledge carrier
    b. Menggambar fakta dalam bentuk digaram dan dokumen
    c. Mendapatkan fakta dengan berkolaborasi antara analis dan domain expert.
    d. Menentukan tingkat keabstrakan proses
11. Proses analisis terdiri dari obtaining (mendapatkan), representing (menggambarkan), verifying facts (verifikasi fakta). Verifying facts artinya _____
    a. Melakukan verifikasi fakta kepada knowledge carrier
    b. Menggambar fakta dalam bentuk digaram dan dokumen
    c. Mendapatkan fakta dengan berkolaborasi antara analis dan domain expert.
    d. Menentukan tingkat keabstrakan proses
12. Di bawah ini yang BUKAN termasuk kedalam Knowledge carrier adalah _____
    a. Analis
    b. User
    c. Partner bisnis
    d. Eksternal observer
13. IT System adalah _____
    a. Konstruksi dasar dari bahasa yang didasari oleh UML
    b. Konstruksi elemen dari UML yang digunakan oleh user
    c. Orang-orang yang terlibat dalam menjalankan, mengoperasikan, dan mengatur proses bisnis
    d. Sebuah sistem yang menyediakan informasi yang dibutuhkan untuk melaksanakan proses bisnis tertentu
14. UML Versi 0.9 yang pertama kali diformulasikan adalah gabungan dari tiga metode yaitu _____
    a. Booch Method, Object Modeling Techinque (OMT), Class Diagram
    b. Booch Method, Object Modeling Technique (OMT), Object-Oriented Software Engineering (OOSE)
    c. Object Modeling Technique (OMT), Object-Oriented Software Engineering (OOSE), Statechart Diagram
    d. Booch Method, Class Diagram, Statechart Diagram

15. Pengembangan UML versi 2.0 dimulai semenjak tahun _____
    a. 2003
    b. 2000
    c. 2005
    d. 1998
16. Dalam menentukan kebutuhan terdapat tiga hal yang sangat penting di bawah ini, KECUALI _____
    a. Bagaimana menentukan kebutuhan?
    b. Siapa yang menentukan kebutuhan?
    c. Untuk siapa kebutuhan ditentukan?
    d. Kebutuhan apa yang harus ditentukan?
17. Dua dokumentasi yang tersedia untuk menggambarkan UML 2.0 yaitu UML 2.0 Infrastruktur dan UML 2.0 Superstruktur, yang dimaksud dengan UML 2.0 Infrastruktur adalah _____
    a. Menjelaskan konstruksi user dari UML 2.0
    b. Menjelaskan sistem elemen dari UML 2.0
    c. Menjelaskan konstruksi software dan hardware oleh UML
    d. Menjelaskan konstruksi dasar dari bahasa yang didasari oleh UML
18. Dua dokumentasi yang tersedia untuk menggambarkan UML 2.0 yaitu UML 2.0 Infrastruktur dan UML 2.0 Superstruktur, yang dimaksud dengan UML 2.0 Superstructure adalah _____
    a. Menjelaskan konstruksi user dari UML 2.0
    b. Menjelaskan sistem elemen dari UML 2.0
    c. Menjelaskan konstruksi software dan hardware oleh UML
    d. Menjelaskan konstruksi dasar dari bahasa yang didasari oleh UML
19. Perubahan UML 2.0 Infrastruktur diharapkan:
    a. Meningkatkan konstruksi dari spesifikasi arsitektur
    b. Menyediakan mekanisme untuk penyesuaian bahasa
    c. Menyediakan pilihan lebih baik untuk behavior modeling
    d. Mengutamakan pendukung untuk pengembangan berbasis komponen (component-based)

20. Di bawah ini yang diharapkan pada perubahan UML 2.0 superstructure, KECUALI _____

a. Meningkatkan konstruksi dari spesifikasi arsitektur
b. Menyediakan mekanisme untuk penyesuaian bahasa
c. Menyediakan pilihan lebih baik untuk behavior modeling
d. Mengutamakan pendukung untuk pengembangan berbasis komponen (component-based)

UNIVERSITAS BUDI LUHUR
FAKULTAS TEKNOLOGI INFORMASI

MODUL PERKULIAHAN #2
MODELING BUSINESS SYSTEM: USE CASE DIAGRAM

Capaian Pembelajaran	:	1. Mahasiswa memahami tentang Modeling Business System 2. Mahasiswa memahami dan dapat membuat Use Case Diagram.
Sub Pokok Bahasan	:	1.11. Modeling Business System 1.12. Business Process dan Business System a. Apakah Business Process? b. Definisi Workflow Management Coalition c. Business System d. Menggunakan UML untuk Business Processes dan Business System e. Practical Tips untuk Business Process Modeling 1.13. Satu Model – Dua Sudut Pandang 1.14. Eksternal View a. Apakah Keuntungan yang Diberikan Business System? b. Elemen dari View 1.15. Use Case Diagram a. Membaca Use Case Diagram b. Membuat Use Case Diagram 1.16. Kesimpulan

		1.17. Soal Latihan
Daftar Pustaka	:	Grässle, P., Baumann, H. and Baumann, P., 2005. UML 2. 0 in Action: A Project-based Tutorial. Packt Publishing Ltd.

MODELING BUSINESS SYSTEM

Dalam memastikan transaksi bisnis berjalan lancar pada penggunaan IT system, sangat diperlukan untuk mengetahui dan mengerti **lingkungan bisnis (business environtment)** dari IT System. Oleh karena itu, analisis dan pemodelan dari proses bisnis merupakan komponen penting dalam pengembangan dan integrasi dari IT system.

Sekarang, kebanyakan IT system tidak hanya menyatu dalam business environtment, tetapi juga terhubung dengan IT System lain. Jadi, setiap IT system baru harus dapat masuk tidak hanya pada satu, tetapi dua lingkungan target yang berbeda:

- **Integration pada level proses bisnis:** setiap IT System harus ditugaskan ke aktifitas dari proses bisnis dengan cara yang memungkinkan untuk memperbaiki dan eksekusi yang tepat dari keseluruhan proses bisnis dan semua kompenen yang terlibat
- **Integration pada level IT-System:** komunikasi dengan IT System lain yang terlibat dalam proses bisnis yang harus berjalan lancar. Hal ini membutuhkan interface tepat yang semantic dan teknis.

Jawablah pertanyaan di bawah ini, untuk diri sendiri:

6. *Kapankah terakhir kali anda mengalami jarak fungsionalitas antara IT System baru dan lingkungannya yang terlihat selama proses pengembangan?*
7. *Berapa banyak IT System yang anda ketahui yang tidak mendukung secara maksimal, bahkan menghalangi, ketika user menjalankan proses?*
8. *Kapan terakhir kali anda mengalami IT System harus dihentikan pada saat diluncurkan, karena error fungsi dalam interface membuat operasi tidak mungkin terjadi?*

Model yang dibuat memerlukan bisnis proses dan bisnis struktur dikarenakan dalam pembuatan model melihat tidak hanya dari aspek dinamik dari model tetapi juga elemen statis.

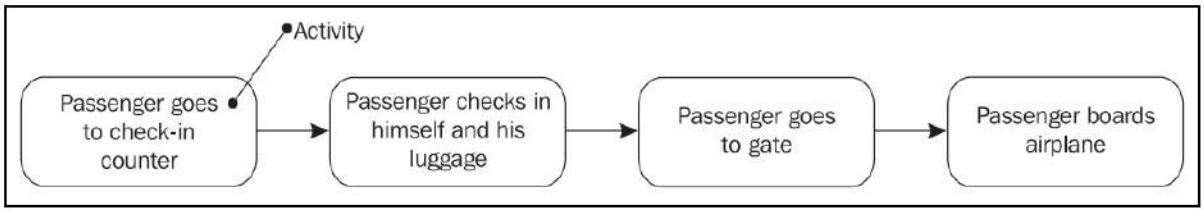
BUSINESS PROCESSES DAN BUSINESS SYSTEM

1.5. APAKAH BUSINESS PROCESSES?

Kebanyakan orang secara intuisi mengerti **proses bisnis (business process)** sebagai *prosedur (procedure)* atau *kegiatan (event)* dengan maksud untuk mendapatkan *tujuan (goal)*. Pada UML Airport dihasilkan banyak tujuan dan proses bisnis berbeda:

1. Tujuan dari penumpang (passenger) adalah untuk berlibur. Dalam mencapai tujuan ini, penumpang harus memesan penerbangan dan hotel, menyiapkan tas, pergi ke UML Airport, check-in dan boarding ke pesawat, keluar pesawat pada airport tujuan, pergi ke hotel, masuk ke kamar, dan membuka tas.
2. Pemilik toko koran pada UML Airport ingin menjual barang dagangan miliknya. Untuk ini, pemilik akan membeli barang dengan harga murah dan menjual ke pelanggannya dengan harga yang tinggi.
3. Agar penumpang check in pada UML Airport, karyawan dari passenger service menerima tiket dan bagasi penumpang, bertanya tentang keinginan posisi duduk penumpang, dan menggunakan IT System. Pada akhir prosedur, penumpang menerima boarding pass dengan nomor kursi yang sudah ditentukan dan gate penerbangan sesuai dengan tandanya.

Seperti yang terlihat, proses bisnis biasanya sering diselesaikan dalam beberapa tahap. Tahap ini biasa disebut dengan **aktifitas (activities)**, dan harus diselesaikan sesuai dengan urutan. Pemilik toko koran tidak dapat menjual barang jika pemilik toko koran belum membeli barang tersebut sebelumnya. Penumpang mengemasi tas sebelum penumpang pergi ke airport. Karyawan dari passenger service pada counter check-in hanya dapat menerbitkan boarding pass setelah proses check-in selesai. Seperti yang terlihat pada Gambar 14



Gambar 14. Aktifitas Dari Proses Bisnis "Passenger Service" (disederhanakan)

Aktifitas dapat berjalan secara berurutan (*sequenctially)* atau bersamaan *(parallel)*. Jadi, penumpang dapat membeli botol minuman pada toko duty-free, sementara bagasi penumpang dimasukan ke Airbus 320 menuju London.

Aktifitas individu dapat di susun secara menyebar. Prosedur Check-in terjadi pada counter check-in dan dilakukan oleh karyawan dari passenger service, sementara pada waktu bersamaan proses boarding terjadi pada lokasi berbeda dan dilakukan oleh karyawan dari passenger service lainnya.

Biasanya, aktifitas dari proses bisnis saling bergantung. Kebergantungan dibuat oleh interaksi dari semua aktifitas milik proses bisnis yang mempunyai satu tujuan sama.

Pertimbangkan tentang yang manakah aktifitas di bawah ini tidak saling tergantung dengan studi kasus UML Airport, karena aktifitas tidak memiliki satu tujuan sama kepada penumpang untuk pergi berliburan dengan Airbus 320:

1. *Mengisi Airbus 320 dengan makanan dan minuman*
2. *Mengisi bahan bakar dari Boeing 737*
3. *Membersihkan dari tempat istirahat UML Airport*
4. *Promosi dari karyawan UML Airport sampai wakil presiden*

1.6. DEFINISI WORKFLOW MANAGEMENT COALITION

Definisi resmi dari istilah **proses (process)** dan **proses bisnis (business proses)** diadaptasi oleh **Workflow Management Coalition**. Berikut adalah definisi yang dapat ditemukan pada daftar istilah di **Workflow Reference Model** dari Workflow Manajgemen Coalition:

"A process is a coordinated (parallel and/or serial) set of process activity(s) that are connected in order to achieve a common goal. Such activities may consist of manual activity(s) and/or workflow activity(s)"

"Sebuah proses adalah serangkaian proses yang terkoordinasi (secara parallel dan/atau serial) yang saling terhubung untuk mencapai tujuan yang sama. Aktifitas tersebut dapat berisi aktifitas manual dan/atau alur kerja aktifitas"

Berdasarkan definisi di atas, sebuah proses adalah serangkaian aktifitas yang terjadi dengan pola terkoordinasi, baik secara parallel atau serial, yang mempunyai tujuan sama. Aktifitas ini dapat dilakukan secara manual atau dapat dilakukan dengan dukukngan IT System.

"A business process is a kind of process in the domain of business organizational structure and policy for the purpose of achieving business objective"

"Sebuah proses bisnis adalah jenis proses pada domain dari struktur organisasi bisnis beserta kebijakannya untuk tujuan mencapai tujuan bisnis"

1.7. BUSINESS SYSTEM

Proses bisnis adalah dinamis secara alami dan melibatkan aktivitas. Akan tetapi, jika ingin melihat keseluruhan sistem bisnis, juga harus mempertimbangkan aspek statis. Hal ini melibatkan, misalnya, struktur organisasi dalam proses bisnis yang dilakukan. Hal ini juga melibatkan beberapa obyek bisnis dan obyek informasi, seperti tiket atau pemesanan. Untuk aspek statis dan dinamik secara keseluruhan, akan digunakan istilah sistem bisnis (business system).

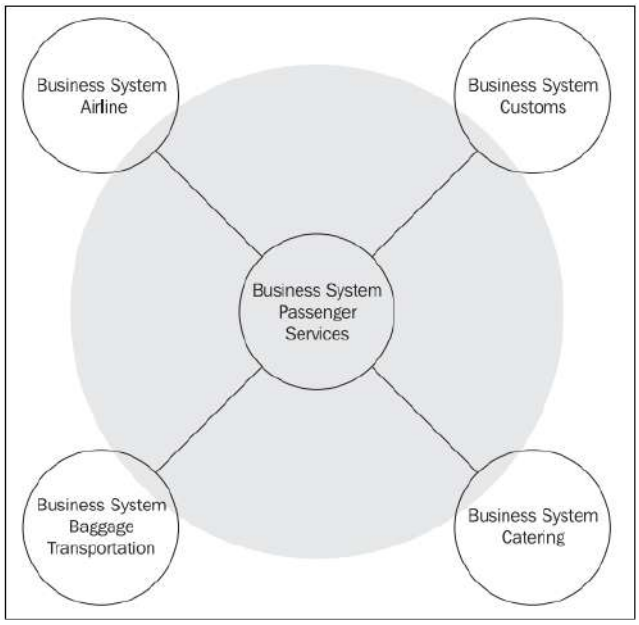
Dalam terminologi bisnis, **bisnis sistem (business system)** merujuk pada rantai nilai tambah (value-added chain), yang menggambarkan proses nilai tambah (value-added process), artinya persediaan dari barang dan jasa. Sebuah bisnis dapat menjangkau satu atau beberapa bisnis sistem.

Setiap sistem bisnis, di dalamnya, menghasilkan keuntungan ekonomis. Jadi, administrasi bisnis artinya binsis sistem tidak banyak berbeda dari istilah sistem binsis. Dimana 'hasil' (result) sebuah sistem bisnis diartikan sebagai 'fungsi' (functionality).

Pada analisis dan pemodelan dari sistem bisnis sangat penting untuk mendefinisikan batasan sistem. Sebuah sistem bisnis yang dibuatkan modelnya dapat diperluas untuk seluruh organisasi. Dalam kasus ini, disebut dengan **model organisasi (organization model)**.

Sangatlah perlu untuk mempertimbangkan dan memodelkan hanya bagian yang dipilih dalam organisasi. Pada Studi Kasus, IT System terintegrasi kedalam pelaksanaan Passenger Service. Oleh karena itu, cukup mengamati operasinya dan mempersempit sistem bisnis hanya tentang Passenger Service.

Passenger Service adalah bagian didalam UML Airport, dengan karyawan, struktur organisasi, dan IT system, seperti yang digambarkan pada Gambar 15. Divisi sekitarnya seperti baggage transportation atau catering, juga termasuk dalam UML Airport, tetapi tidak dalam sistem bisnis studi kasus. Jadi, mereka diperlakukan seperti sistem bisnis eksternal lainnya:



Gambar 15. Batasan Sistem Selama Analisis dari Sistem Bisnis

Eksternal bisnis sistem yang lain secara keseluruhan tidak dibahas, tetapi pada *interface* antara mereka dan bisnis sistem studi kasus. Misalnya, staff pada passenger service perlu mengetahui bahwa staff perlu memindahkan bagasi penumpang ke baggage transportation, sehingga bisa dimasukan ke dalam pesawat. Tentu saja,

untuk ini, Passenger Service perlu mengetahui bagaimana baggage transportation menerima bagasi, sehingga dapat digunakan seusai dengan prosesnya. Sangat mungkin IT System dari Passenger Service dan baggage transportation harus dihubungkan, artinya interface harus dibuat. Dilain pihak, Passanger Service sangat tidak memperdulikan tentang bagaimana baggage transportation diatur, dan apakah setiap koper secara individu dibawa sepanjang runway atau kereta digunakan untuk memindahkan bagasi ke pesawat.

1.8. MENGGUNAKAN UML UNTUK BUSINESS PROCESSES DAN BUSINESS SYSTEM

Sebelum berlanjut ke pemodelan dari proses bisnis dan sistem bisnis dengan UML, perlu dicari tahu apakah UML cocok untuk pemodelan proses bisnis dan sistem bisnis. Dalam mencari tahu maka dapat melihat definisi UML oleh OMG (Object Management Group Inc. – Asosiasi internasional yang mempromosikan standar terbuka untuk aplikasi berbasis obyek, yang menerbitkan setiap versi UML yang disubmit).

"The Unified Modeling Language is a visual language for specifying, constructing, and documenting the artifacts of system"

"Unified Modeling Language adalah Bahasa visual untuk menentukan, membangun, dan mendokumentasikan artifact dari sistem"

Definisi di atas menunjukan bahwa UML adalah Bahasa untuk modeling dan mewakili sistem secara umum, jadi, termasuk sistem bisnis.

Dikasus apapun, UML memenuhi setidaknya satu kebutuhan dari pemodelan bisnis sistem: mencerminkan beberapa view dari sistem bisnis, untuk menangkap aspek berbeda. Beberapa diagram UML yang sudah distandarkan memenuhi kebutuhan ini, karena setiap diagram memberikan view berbeda dari pemodelan bisnis sistem.

Dicapai batasan dari UML ketika pemodelan memperluas proyek proses bisnis, misalnya, rekayasa ulang proses bisnis, atau ketika membuat model untuk seluruh organisasi. Tetapi, untuk jenis proyek ini diperlukan metode yang kuat dan tools yang

tersedia, seperti **Architecture of Integrated IT System (ARIS)**. Hal ini bukan berarti menutup kemungkinan menggunakan UML untuk proyek tersebut, meskipun direkomendasikan dalam pembelajaran UML spesifikasi yang mendalam dan penggunaan CASE tool.

Studi kasus ini dibuat dengan tujuan pengembangan IT System. Bahkan, dibuat menuju proyek yang jaminan utamanya adalah kelancaran intergarsi IT System dari sistem bisnis. Di bawah ini adalah karakteristik dari proyek tersebut:

1. Bisnis proses yang terdampak oleh pembangunan dan integerasi dari IT System dapat dipertimbangkan
2. Pemodelan proses bisnis bukanlah focus pada proyeknya. Sebagai gantinya, model sebagai dasar untuk membangun dan integrasi dari IT System. Integrasi bisnis proses dapat menentukan keberhasilan atau kegagalan dari proyek; tetapi tugas utama masih pembangunan IT System.
3. Dikarenakan pendanaan biasanya sangat ketat, investasi waktu dalam metodologi dan Bahasa dibutuhkan untuk pemodelan proses bisnis, jumlahnya tidak lebih dari 5-10% dari total proyek.

1.9. PRACTICAL TIPS UNTUK BUSINESS PROCESSES MODELING

Seseorang sering diperingatkan tentang kompleksitas dari analisis proses bisnis dan pemodelan proses bisnis. Tetapi, berdasarkan pengalaman sebagian besar proses bisnis secara keseluruhan dapat dipahami dan dikendalikan. Sebaliknya, kurang jelasnya dan kurang transparan membuat proses bisnis terlihat lebih kompleks dari yang seharusnya.

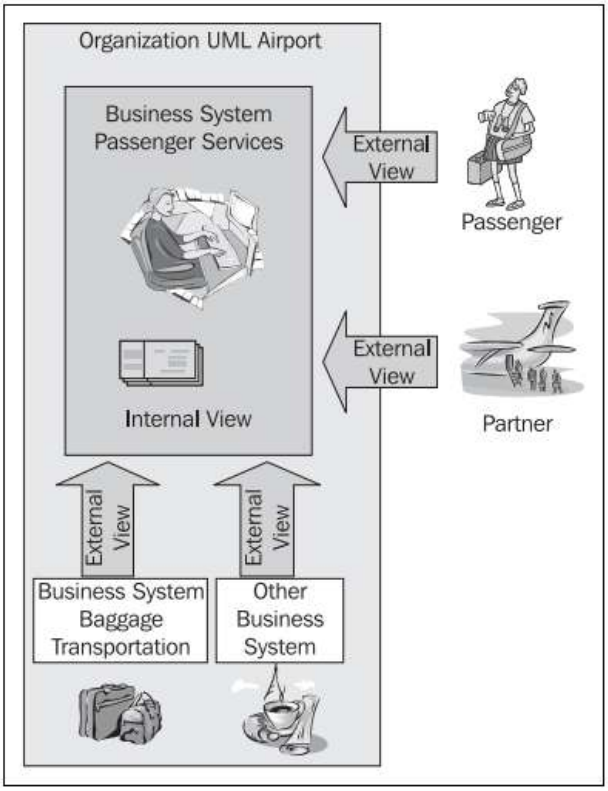
Dalam banyak kasus, proses bisnis yang ada didokumentasikan dengan tidak baik atau tidak ada dokumentasi sama sekali. Hal ini dapat ditelusuri dengan fakta bahwa selama bertahun-tahun sebagian besar fungsionalitas diperlakukan sebagai 'pulau' daripada sebagai bagian dari proses bisnis menyeluruh. Oleh karena itu, penghubung antara aktivitas – rantai proses – telah hilang. Jika overview hilang, proses bisnis akan terlihat sulit.

Akan ada banyak hambatan dimasa depan jika proses bisnis ditangani oleh IT System. Sebagian besar waktu, dokumentasi dari manual alur kerja (workflow) dilakukan antara sistem individu tidak tersedia. Pada kasus lain, fungsionalitas dari IT System adalah tidak diketahui karena proses berjalan secara otomatis, tersembunyi disuatu tempat dalam kotak hitam, dan hanya input dan output yang terlihat.

Arsitektur proses bisnis yang ada atau referensi model yang sudah ada dapat mempercepat dan mempermudah proses pemodelan. Membandingkan proses dengan proses yang mirip atau sama dalam organisasi lain dapat membantu dalam mendefinisikan perbedaan dan memperoleh kemungkinan untuk perbaikan.

SATU MODEL – DUA SUDUT PANDANG

Sebuah sistem bisnis dapat ditampilkan dari sudut pandang berbeda. Karena ini, model sistem bisnis terdiri dari dua tampilan (view). Setiap view menekankan aspek tertentu dari sistem bisnis, dan setiap view terhubung satu dengan lainnya, seperti yang terlihat pada Gambar 16



Gambar 16. Tampilan (view) Eksternal dan Internal dari Sistem Bisnis

Melihat sebuah sistem bisnis dari luar, dapat mengambil peran sebagai customer, partner bisnis, supplier, atau sistem bisnis lainnya. Pada **eksternal view (external view),** hanya proses bisnis yang melibatkan orang luar yang penting. Eksternal view menggambarkan lingkungan dari sistem bisnis. Bisnis sistemnya sendiri tetap berada pada kotak hitam.

Dalam sistem bisnis, ditemukan *karyawan (employees)* dan *alat (tools)* yang bertanggung jawab untuk memenuhi permintaan dari lingkungan, dan untuk menangani proses bisnis yang dibutuhkan. Dibalik proses bisnis adalah *alur kerja (workflow)* dan *IT System*. Setiap individu karyawan adalah bagian dari struktur organisasi. Normalnya **internal view** ini tetap tidak terlihat untuk orang luar.

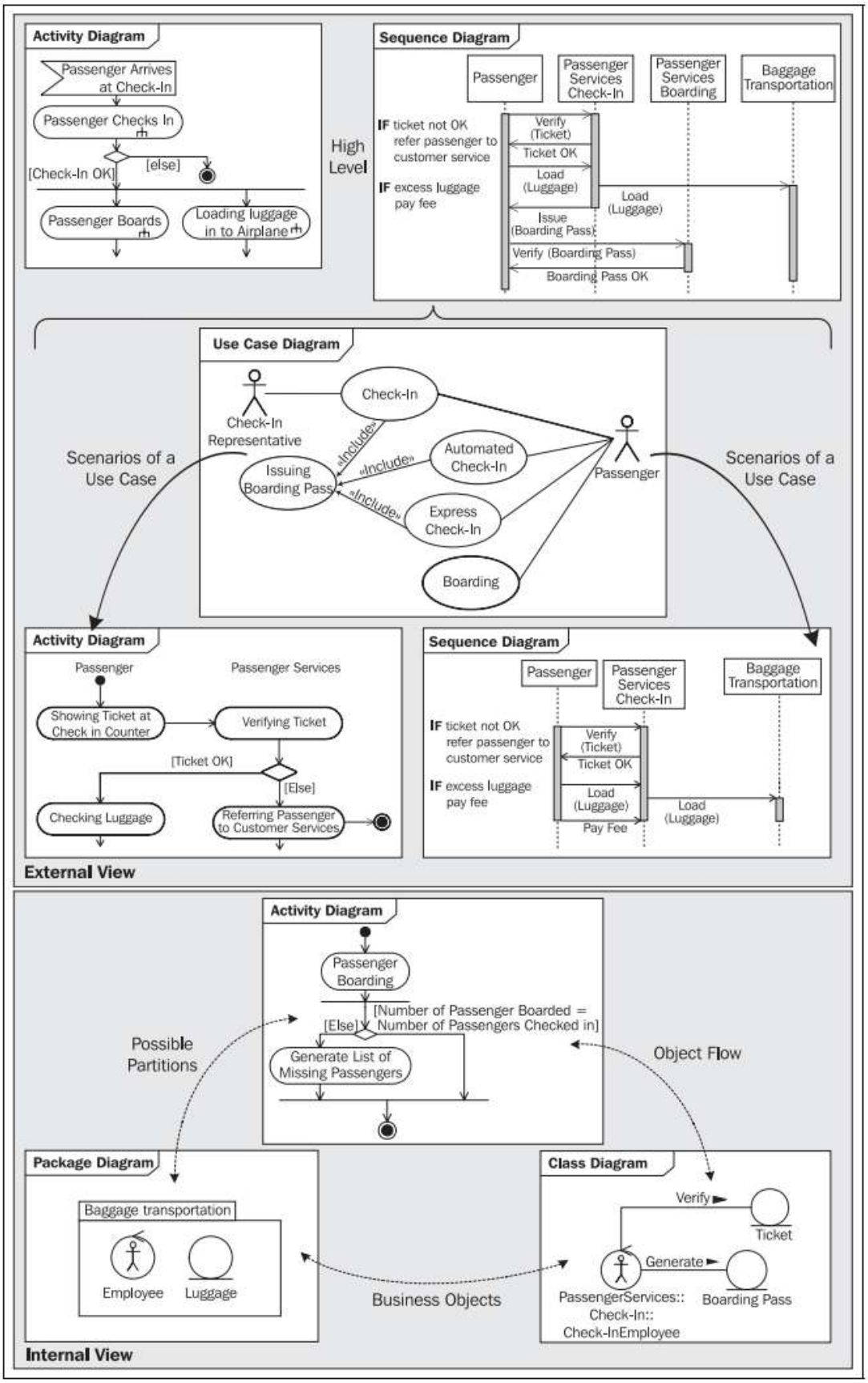
Perhatikan studi kasus UML Airport:

1. Pikirkan tentang layanan apa yang disediakan pada Passenger Service untuk anda, sebagai penumpang.
2. Karyawan yang mana dari Passenger Service yang akan berhubungan dengan penumpang?
3. Sebagai penumpang, apakah anda melihat prosedur yang dilakukan oleh karyawan bagian check-in, ketika anda mendapatkan boarding pass? Apakah anda ingin mengetahui bagaimana bagasi anda berakhir pada pesawat, atau anda tidak peduli, selama bagai tidak hilang?

Membuat model dari Bisnis sistem dimulai dari eksternal view. Dengan cara ini, dimulai pada deskripsi sistem bisnis dari sudut pandang customer, partner bisnis, dan supplier pada Gambar 17.

Berdasarkan Gambar 17, tampilan internal menggambarkan *bagaimana (how)* sistem bisnis menyediakan jasa. Use Case dari eksternal view melayani dengan baik sebagai basis untuk membuat test skenario, yang dibutuhkan untuk testing IT System yang sudah jadi.

Setiap view yang digunakan pada pemodelan sistem bisnis, dan diagram UML yang tergabung didalamnya, digambarkan pada Gambar 17



Gambar 17. Perbedaan Tampilan (view) dan Diagram

EXTERNAL VIEW

1.10. APAKAH KEUNTUNGAN YANG DIBERIKAN BUSINESS SYSTEM?

Sebagai customer atau partner bisnis dari sebuah organisasi, tidak perlu mengetahui jika transaksi dalam organisasi terjadi secara manual atau berbasis IT. Selain itu juga tidak tertarik dalam berapa banyak formulir yang harus diisi oleh karyawan dalam organisasi, baik itu 2 atau 20. Customer dan partner bisnis hanyalah tertarik pada jenis barang dan jasa apa yang ditawarkan, dan bagaimana mereka dapat menggunakannya. View dari sisi customer menggambarkan interaksi dengan pihak eksternal, seperti customer dan partner, dan menyajikan sistem bisnis sebagai kotak hitam.

Pertimbangkan sistem bisnis, seperti Passanger Service atau toko koran di aiport dari luar. Output apa yang diminati oleh customer dan partner bisnis? Apakah output merupakan jasa atau barang material?

Dari view pada administrasi bisnis, tujuan dari sistem bisnis adalah output yang menguntungkan. Output secara umum dibagi menjadi barang atau jasa. Produksi dari barang, untuk contoh, produksi kemasan kotak dari cokelat swiss.

Bagaimana membedakan jasa? Jasa adalah barang tidak berwujud, seperti memesan kursi atau memasukan bagasi ke pesawat. Tidak seperti barang material, jasa tidak dapat diberikan kecuali supplier dan customer melakukan kontak. Tetapi, jasa dapat melibatkan barang material. Jika sebuah kotak cokelat swiss dijual pada toko koran, transaksi ini adalah jasa. Berdasarkan contoh tersebut transaksi dari barang material diperlakukan sebaga jasa karena melibatkan customer.

Karena itu, persedian dari barang material dan jasa adalah relevan untuk eksternal view. Tampilan eksternal tidak menggambarkan bagaimana karyawan dan IT System menyediakan barang dan jasa, dan bagaimana proses bisnis terjadi dalam sistem bisnis. Pada eksternal view, hanya aktifitas yang melibatkan orang luar yang penting.

Dalam studi kasus, sangat penting untuk penumpang mengetahui bahwa mereka dapat check in dengan tiket valid pada counter check-in, dan mereka kemudian menerima sebuah boarding pass. Apa yang dilakukan oleh karyawan dan IT System agar mereka menerima boarding pass tetap tersembunyi dari penumpang – dan dalam banyak kasus penumpang tidak perlu mengetahui bagaimana.

Dalam prakteknya, eksternal view sangat sulit digambarkan, jika karyawan dari organisasi, yang ada *dalam* sistem bisnis, mengembangkan pemodelannya. Sangat sulit untuk seseorang dalam sistem bisnis, yang mengetahui semua transaksi internal, untuk membangun view dari customer, yang tidak mempertimbangkan transaksi internal sama sekali. Jika eksternal dan internal view dicampur, akan menghalangi tampilan yang jelas dari luar sistem bisnis dan proses bisnis (Jadi, dibuat sistem yang tidak user-friendly) Karena itu, berkonsultasi dengan karyawan yang tidak bias, yang dapat menempatkan dirinya sebagai bagian luar dengan mudah, misalnya, karyawan atau bagian lain atau konsultan eksternal.

BUSINESS USE CASE

Sebelum membahas ke **business use case,** perlu diketahui definisi umum dari use case pada UML. Use Case adalah spesifikasi dari rangkaian aksi yang dilakukan oleh sistem, yang menyebabkan hasil yang tampak yaitu sebuah nilai khusus untuk satu atau lebih aktor atau stakeholder lain dari system.

Apakah yang bisa diamati, hasil yang berharga dalam sistem bisnis? Pertanyaan ini – bagaimana menemukan use case – telah menyibukkan analis dan desainer semenjak hari pertama istilah digunakan. Use case dari bisnis sistem adalah jasa dari sistem bisnis yang diberikan ke customer, partner bisnis, atau sistem bisnis lainnya. Berlawanan dengan hal ini, fungsionalitas yang ada dalam sistem bisnis, yang tidak terlihat dan tidak dapat diakses oleh orang luar, mewakili sebuah aktifitas internal, artinya sebuah proses bisnis internal.

Pada level pemodelan bisnis digunakan istilah business use case daripada use case. Alasan dari pemisahan ini adalah pemisahan jelas dan eliminasi dari kekeliruan dalam

transisi dari pemodelan sistem bisnis ke pemodelan IT System. Business Use Case digunakan untuk pemodelan sistem bisnis. Dibalik semua ini, tidak ada perbedaan antara business use case dan use case.

Proses bisnis dapat dilakukan secara manual atau dengan bantuan IT. Sekarang ini, semua proses bisnis dapat dimulai dan dibuat seluruhnya tanpa bantuan manusia. Berdasarkan realita ini, business use case dapat meliputi tugas manual, dan juga kegiatan dengan bantuan IT.

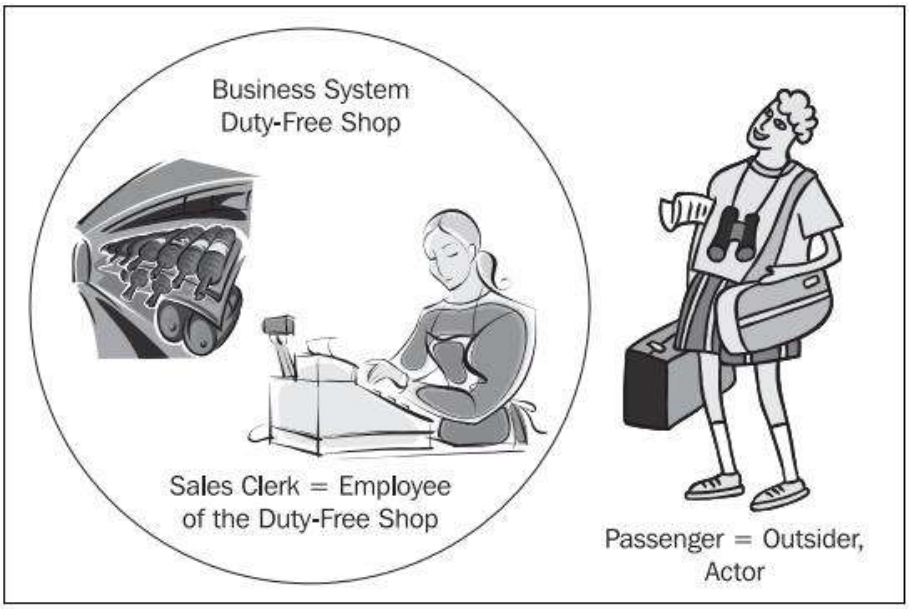
Jika melihat dari kantor perdagangan Hanseatic, secara khusus ditemukan business use case yang dilakukan secara manual. Jika customer dari kantor dagang memesan kulit Rusia, pegawai toko menggunakan pena dan tinta memasukan pesanan kedalam buku pesanan. Jadi, business use case sudah ada pada abad pertengahan.

Pada studi kasus, passanger service yang dilakukan secara manual serta kegiatan yang dibantu dengan IT. Seperti contoh, IT System dari Passanger Service melakukan pemesanan kursi untuk penumpang, sedangkan karyawan melakukan verifikasi dari tiket secara manual.

Penumpang yang melakukan check in pada mesin tidak berhubungan dengan manusia. Mesin Check-in melakukan semua business use case

ACTOR

Diluar dari sistem bisnis, misalnya, customer atau partner bisnis, yang menggunakan output dari sistem bisnis. Tidaklah penting **orang luar (outsider)** ini mengetahui secara detail bagaimana kasus bisnis dilakukan. Bagi penumpang, penting untuk mengetahui bahwa mereka dapat membeli minuman di toko duty-free. Botol minuman adalah *barang material* yang disediakan oleh toko duty-free; menjual botol ke customer, dilain pihak, adalah *jasa*. Penumpang tidak peduli bagaimana pegawai toko duty-free melakukan penjualan. Orang luar ini disebut **actor** (lihat Gambar 18)



Gambar 18. Orang Luar (Outsider) dan Staff

Business Use Case selalu dimulai oleh actor, artinya bahwa customer atau partner bisnis memanfaatkan jasa. Penumpang, yang berjalan-jalan pada toko duty-free, mempertimbangkan Botol minuman merupakan tawaran yang murah, sehingga memutuskan membeli botol minuman tersebut. Hal ini membuat penumpang sebagai inisiator dari penjualan. Selama transaksi dari business use case, actor dapat berinteraksi dengan manusia dan IT system dalam sistem bisnis yang bertanggung jawab terhadap transaksi. Contohnya, penumpang harus menyerahkan uang dengan jumlah tertentu, agar mendapatkan botol minuman.

Aktifitas yang diawali oleh karyawan atau IT System dalam sistem bisnis bukan merupakan business use case eksternal view, tetapi merupakan aktifitas internal view dan digambarkan pada activity diagram atau sequence diagram pada internal view.

Seperti yang terlihat, aktor dari sistem bisnis dapat berupa manusia, organisasi, atau IT System. Walupun organisasi digambarkan sebagai aktor, seperti dalam kasus baggage transportation, pada akhirnya akan menemukan orang atau IT System dibalik aktor yang mengawali dan melakukan kegiatan. Tetapi, yang relevan untuk model yang akan dibuat adalah **peranan (role)** yang dimainkan. Pada level pemodelan bisnis sistem, tidak penting jika peranan tersebut dimainkan oleh manusia, system IT, organisasi, atau divisi pada organisasi, mesin, atau system lainnya.

Perhatikan Studi Kasus UML Airport, tentukan semua orang, unit organisasi, dan IT system yang terlibat. Kemudian, coba untuk menyusun mereka berdasarkan kriteria ini:

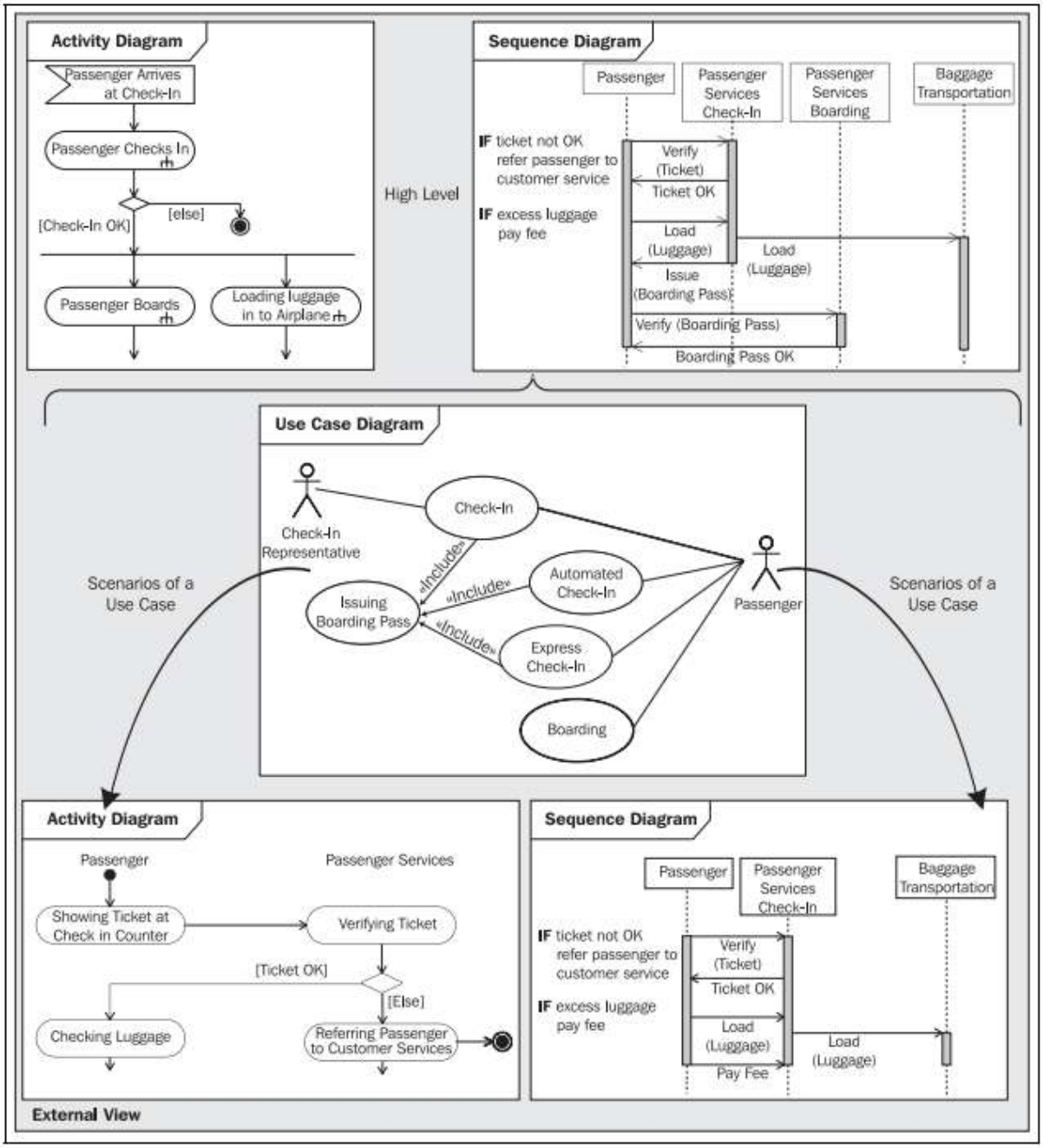
1. Yang manakah outsider (customer, partner bisnis, dll) dan output apa yang diperlukan outsider?
2. Orang apa yang ditemukan pada Passanger Service sebagai karyawan, dan tugas apa yang dikerjakan?
3. IT System apa yang terlibat?
4. Agar penumpang membeli botol minuman duty-free, pegawai dari toko duty-free harus memeriksa boarding pass dari penumpang, menerima uang, memasukan botol k etas, dan menyerahkan struk. Kegiatan mana yang merupakan milik dari internal view, dan mana yang merupakan milik ke eksternal view?

1.11. ELEMEN DARI VIEW

Jenis UML diagram di bawah ini merupakan eksternal view:

1. **Use Case Diagram** menampilkan aktor, business use case, dan hubungannya. Use Case Diagram tidak menggambarkan prosedur. Skenario alternative tetap tersembunyi. Diagram ini memberikan overview yang benar dari fungsionalitas pada sistem bisnis.
2. **Activity Diagram** menggambarkan prosedur, dalam studi kasus, proses bisnis dari sistem bisnis. Subyek dari deskripsi adalah interaksi antara aktor dan sistem bisnis, artinya barang dan jasa yang ditawarkan ke customer dan partner bisnis. Pada activity diagram, pihak luar dapat mengenali bagaimana berinteraksi dengan sistem bisnis. Activity diagram berguna untuk mengilustrasikan sequence, alternative, dan kegiatan parallel. Activity diagram dapat dibuat dalam tingkat detail berbeda-beda.
3. **Sequence Diagram** menampilkan urutan secara kronologis dari interaksi. Sequence diagram menggambarkan setiap kegiatan dengan semua cabang dan paralelnya, tetapi informasi yang bertukaran antara pihak yang terlibat. Diagram

ini adalah dasar dari pertukaran data dan informasi dengan partner dan customer (Gambar 19):



Gambar 19. Eksternal View (Tampilan Eksternal)

Diagram UML untuk deskripsi dari business use case dapat dinotasikan dengan penjelasan tertulis dan gambar ilustrasi. Setiap diagram tidak harus digunakan pada setiap kasus. Jenis diagram mana yang harus digunakan tergantung pada kebutuhan karakteristik sistem yang ditekankan. Pada kasus lain, direkomendasikan

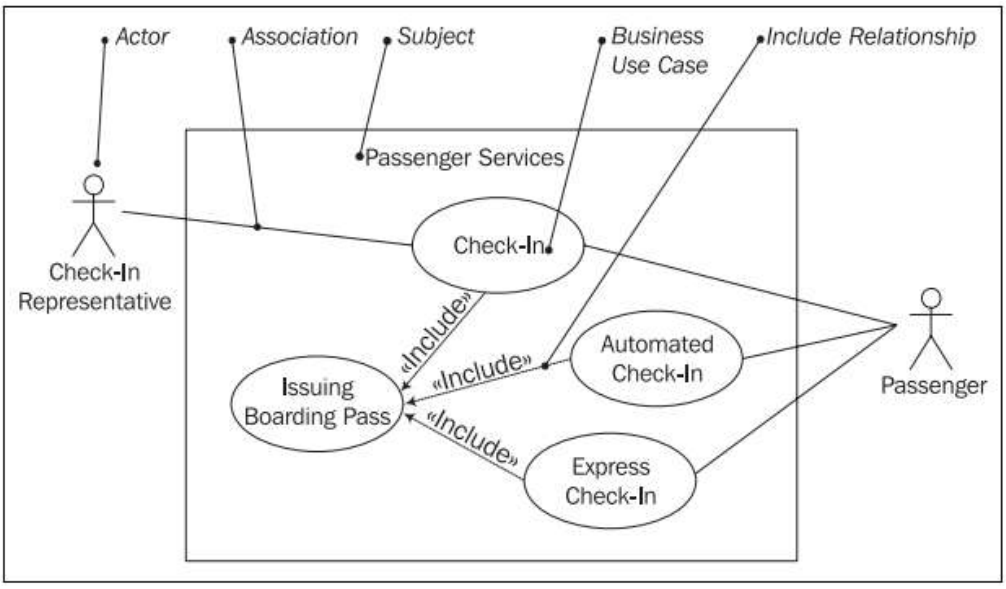
menggunakan use case diagram karena diagram jenis ini cocok untuk berkomunikasi dengan partner sistem dan domain expert tentang dasar fungsionalitas dan konteks dari sistem. Activity diagram tingkat tinggi dengan derajat detail rendah, yang didalamnya dapat dimasukan beberapa use case, juga cukup untuk tujuan ini.

Ketika memperbaiki business use case dan mengidentifikasikan beberapa skenario, sangat penting untuk menggambarkan beberapa kegiatan dengan activity diagram.

Sequence diagram menampilkan pertukaran informasi dengan partner dan customer. Berdasarkan pengalaman, sequence diagram mendapatkan penerimaan besar dalam pemodelan proses bisnis. Dikarenakan sequence diagram mudah dibaca dan hanya membutuhkan sedikit elemen grafis. Selama pengetahuan dasar ada tentang kegiatan teknis, sequence diagram sering cocok untuk overview dari interaksi sistem bisnis daripada activity diagram.

USE CASE DIAGRAM

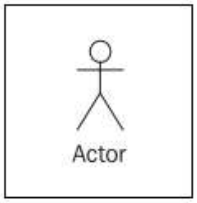
Use case diagram menampilkan business use case, actor, dan hubungan diantaranya. Hubungan antara actor dan business use case menyatakan bahwa actor dapat menggunakan fungsionalitas tertentu dari sistem bisnis. Tidak akan ditemukan informasi apapun tentang bagimana atau dengan urutan apa jasa ini diberikan (Gambar 20)



Gambar 20. Element dari Use Case Diagram

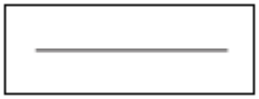
Di bawah ini elemen yang digunakan pada use case diagram:

Actor: actor menggambarkan sebuah peran yang diambil oleh pihak luar ketika berinteraksi dengan sistem bisnis. Misalnya, aktor dapat menjadi customer, partner bisnis, supplier, atau sistem bisnis lainnya. Setiap actor memiliki nama:



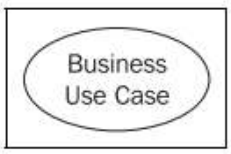
Daripada simbol stick figure, simbol lain dapat digunakan juga, jika cocok pada karakteristik dari aktor dan menuju pada diagram yang praktis dan mudah dibaca.

Association: asosiasi adalah hubungan antara actor dan business use case. Asosiasi menandakan bahwa aktor dapat menggunakan fungsionalitas tertentu dari sistem bisnis – business use case:



Sayangnya, Asosiasi tidak memberikan informasi apapun tentang cara bagaimana fungsionalitas digunakan. Jika business use case memasukan beberapa aktor, tidak jelas pada use case diagram jika setiap actor dapat mengadakan business use case sendiri, atau jika aktor mengadakan business use case bersamaan. Faktanya, asosiasi hanya berarti bahwa aktor *terlibat* dalam business use case.

Business Use Case: business use case menggambarkan interaksi antara actor dan sistem bisnis, artinya business use case menggambarkan fungsionalitas dari sistem bisnis yang memanfaatkan actor:

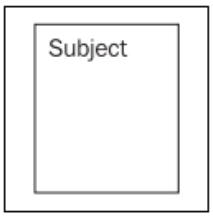


Busines Use Case menggambarkan dari sudut pandang actor. Terlepas dari kegunaan khusus dari business use case sebagai use case pada sistem bisnis, tidak ada perbedaan antara business use case dan 'normal' use case.

Include Relationship: hubungan include adalah hubungan antara dua business use case yang menandakan bahwa business use case pada sisi dimana mata panah menunjuk adalah termasuk ke use case pada sisi lain panah. Artinya bahwa untuk satu fungsi yang disediakan oleh sistem bisnis, fungsionality lainnya dari sistem bisnis telah di akses. Dengan cara ini, fungsionality yang diakses berulang-ulang dapat digambarkan sebagai individual business use case, yang dapat digunakan berbagai macam cara:

Dalam berjalannya waktu, arah dari panah dapat membingungkan; hubungan harus dibaca disamping arah dari panah (check-in includes issuing the boarding pass)

Subject: subyek menggambarkan sistem bisnis yang memiliki satu atau lebih dari business use case didalamnya. Subyek digambarkan dengan kotak persegi yang mengelilingi business use case, dan disertai dengan nama:

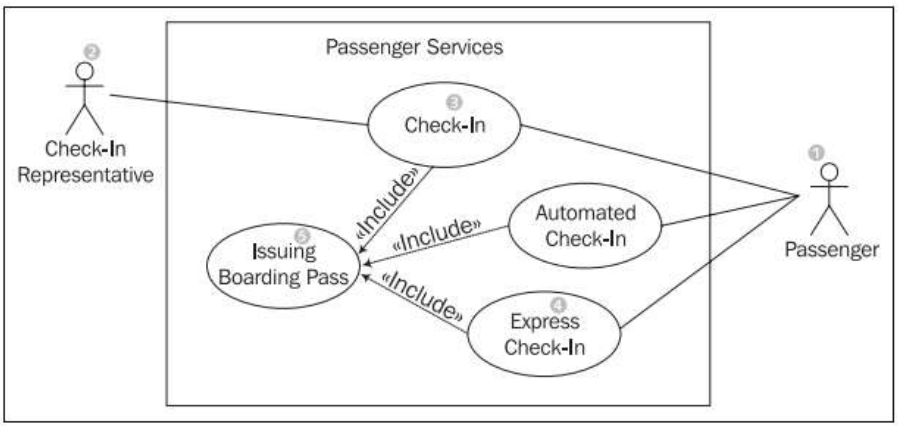


Menggambarkan subyek (dan dengan batasan sistem) adalah optional.

1.12. MEMBACA USE CASE DIAGRAM

Gambar 21 merupakan ilustrasi dari use case diagram dengan aktor: **passanger** (1) dan **check-in representative** (2), dan juga business use case **check-in** (3) dan **express check-in** (4).

Tergantung pada apa yang disukai, membaca use case diagram dapat dimulai dengan actor atau dengan business use case. Dimulai dengan actor, **passanger** (1), ditemukan asosiasi (garis) ke dua business use case, **check-in** (3) dan **express check-in** (4). Ini artinya bahwa orang, sebagai passenger, dapat pergi melalui check-in, atau express check-in, yang dapat dilakukan tanpa bagasi.



Gambar 21. Use Case Diagram

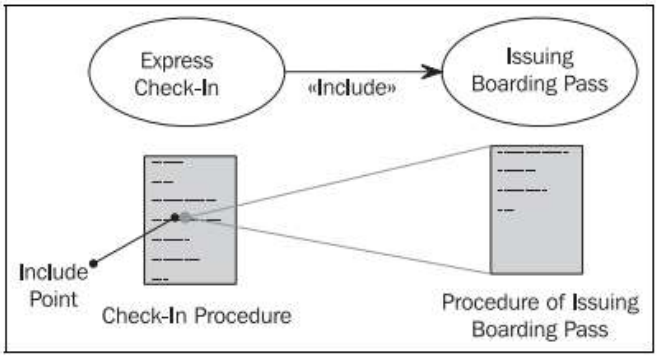
Satu dari dua business use case dibawah yang lain tidak berarti apapun. Sebuah use case diagram tidak mendokumentasikan urutan yang berarti yaitu bagaimana business use case dilakukan. Tentu saja, urutan penting untuk mendeskripsikan dan menghubungkan proses bisnis. Hal ini digambarkan pada activity diagram.

Actor **check-in representative** (2) juga memiliki asosiasi ke business use case **check-in** (3). Ini artinya tidak hanya passenger, tetapi juga seseorang yang diwakili dapat melakukan check-in. Actor, **passenger** (1), juga memiliki asosiasi ke uce case **check-in** (3) artinya bahwa passanger dan check-in representative dapat melakukan check-in. Tetapi, yang diagram tidak tampilkan secara jelas adalah tidak berarti bahawa kedua actor melaukan check-in secara bersamaan. Fakta ini hanya dapat digambarkan pada diagram lain atau bisa digambarkan dalam bentuk komentar yang berisi teks informasl.

Actor **check-in representative** (2) hanya memiliki asosiasi ke business use case **check-in** (3) artinya bahwa UML Airport sebagai perwakilan dari penumpang tidak besa melakukan **express check-in** (4).

Dapat dilihat diagram yang simple dapat berisi banyak informasi. Business use case **check-in** (3) dan business use case **express check-in** (4) masing-masing memiliki hubungan include dengan **issuing boarding pass** (5). Keduanya menggunakan business use case issuing boarding pass pada satu titik di interaksinya. (Use Case tidak dapat menentukan *kapan* use case lain dijalankan.) Terkadang ketika check-in maka

boarding pass akan keluar dan diserahkan ke penumpang atau check-in representative. Gambar 22 menjelaskan secara jelas prosedur include:



Gambar 22. Hubungan Include Antara Use Case

1.13. MEMBUAT USE CASE DIAGRAM

Check list di bawah ini menampilkan langkah-langkah yang diperlukan untuk membuat use case diagram, setiap langkah akan dijelaskan secara detail.

Check List Membuat Use Case Diagram dari Eksternal View:

1. Mengumpulkan sumber infomrasi – bagaimana saya dapat mengetahui hal itu?
2. Identifikasi potensial Actor – siapa partner dan customer yang menggunakan barang dan jasa dari sistem bisnis?
3. Identifikasi business use case – barang dan jasa apa yang dapat digunakan actor?
4. Menghubungkan business use case – siapa yang dapat menggunakan barang dan jasa dari sistem bisnis?
5. Menggambarkan actor – siapa dan apa yang actor wakilkan?
6. Mencari lebih lanjut business use case – hal lain apa yang harus diselesaikan?
7. Menguah business use case – apakah ayng seharusnya dimasukan kedalam business use case?
8. Dokumentasi business use case – apa yang terjadi dalam business use case?
9. Moel relationship antara business use case – kegiatan apa yang dilakukan berulang kali?
10. Verifikasi View – apakah sudah benar?

List di atas bukan merupakan kewajiban, dalam prakteknya, langkah individu sangat berat sering tumpeng tindih. Di satu sisi, pengetahuan umum dari sistem bisnis dan proses bisnis sangat penting untuk realisasi dari setiap langkah individu. Di sisi yang lain, dalam banyak langkah juga diperlukan untuk berkonsultasi dengan knowledge carrier.

MENGUMPULKAN SUMBER INFORMASI – BAGAIMANA SAYA DAPAT MENGETAHUI HAL ITU?

Sebagai langkah pertama, sangat penting untuk mencari pembawa pengetahuan (knowledge carrier), agar analis dan knowledge carrier dapat mengerjakan prinsip dasar bersama-sama. Knowledge carrier adalah, seperti:

1. Orang yang terlibat dalam melakukan, mengoperasikan, dan mengatur proses bisnis
2. Pengguna dari IT System yang sama atau terkait.
3. Customers, orang yang sering menjadi knowledge carrier yang kritis dan kreatif
4. Partner Bisnis
5. Domain Expert
6. Management
7. Pengamat eksternal (External Observer)

Beberapa teknik yang membantu telah terbukti praktis untuk analis memahami proses bisnis:

1. Mengamati karyawan yang sedang bekerja
2. Berpartisipasi dalam proses bisnis yang sedang diteliti
3. Mengambil peranan sebagai outsider (contohnya: sebagai customer)
4. Memberikan survey
5. Melakukan wawancara
6. Brainstorming dengan semua orang yang terlibat
7. Berdiskusi dengan domain expert
8. Mereview formulir, dokumentasi, spesifikasi, buku panduan, dan alat bantu yang sudah ada
9. Menggambarkan struktur organisasi dan alur kerja manajemen

10. Mereview chart organisasi dan deskripsi pekerjaan

1. Ingat kembali Modul 1. Pada modul 1 sudah dijelaskan dasar dari studi kasus, untuk memahami proses bisnis lebih baik. 2. Dalam pikiran anda, lakukan semua peranan yang diketahui dan proses bisnis dari peranan tersebut (penumpang, pegawai toko duty-free, dll) 3. Kegiatan yang mana yang anda ketahui dari sudut pandang penumpang? Bagaimana anda menyegarkan ingatan?

Hasil dari langkah ini adalah berupa kumpulan formulir, instruksi kerja, survey yang sudah selesai, deskripsi proses yang sudah ada, obyek bisnis seperti tiket atau boarding pass, dan lain lain. Overview ini sering tidak lengkap, dan harus diperluas selama proses pembuatan model.

IDENTIFIKASI POTENSIAL ACTOR – SIAPA PARTNER DAN CUSTOMER YANG MENGGUNAKAN BARANG DAN JASA DARI SISTEM BISNIS?

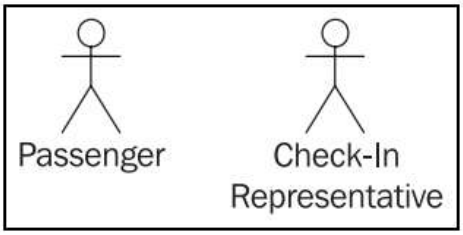
Langkah ini adalah mengenai penentuan semua yang berpotensi menjadi actor. Disini, aturan ini berlaku: lebih banyak semakin meriah (the more the merrier). Pada langkah ini dapat bekerjasama dengan actornya pada langkah selanjutnya; atau dapat dikurangi jumlahnya, atau mungkin digabung.

Actor yang paling potensial dapat ditemukan dengan menjawab pertanyaan berikut (didapat melalui konsultasi dengan knowledge carrier). Dalam melakukan ini, disarankan untuk membuat kelompok dan jenis organisasi dengan memisahkan secara langsung contoh konkrit dari orang dan organisasi tertentu:

1. Customer mana yang merupakan customer dari sistem bisnis, dan mana customer yang merupakan proses bisnis?
2. Siapakah partner eksternal dari sistem bisnis? Mana barang dan jasa yang digunakan oleh partner eksternal?
3. Posisi in-house dan unit organisasi mana yang merupakan partner dari bisnis sistem dan menggunakan barang dan jasanya?

4. Dengan sistem bisnis eksternal mana sistem bisnis berinteraksi?

Sebagai langkah pertama, penjelasan sebelumnya dari studi kasus menghasilkan actor berikut:



Gambar 23. Potensial Aktor

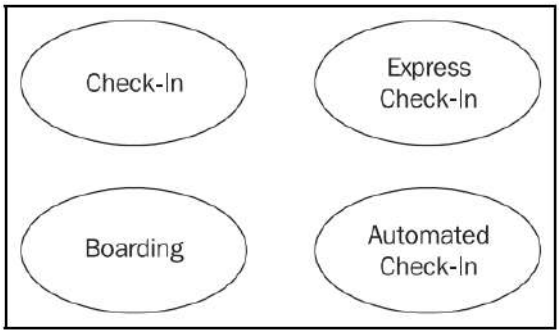
Tambahan dari passenger, yang mewakili sebagai wisatawan, ada check-in representative. **Check-in representative** adalah orang yang bukan penumpang sebenarnya, tetapi agen dari penumpang. Check-in representative memiliki tugas untuk melakukan check-in menggunakan tiket penumpang.

IDENTIFIKASI BISNIS USE CASE – BARANG DAN JASA APA YANG DAPAT DIGUNAKAN ACTOR?

Langkah ini tentang menemukan potensial business use case. Aturannya adalah – lebih banyak semakin meriah (the more the merrier) – juga berlaku disini (dengan wajar). Pontensial business use case dapat ditemukan dengan menjawab pertanyaan ini:

1. Barang dan jasa apa yang disediakan untuk digunakan oleh customer?
2. Barang dan jasa apa yang disediakan untuk digunakan oleh partner eksternal?
3. Barang dan jasa apa yang disediakan oleh sistem bisnis yang melibatkan supplier (supplier dari barang dan jasa)?
4. Apa yang dilakukan oleh individual actor?
5. Bagaimana dan kesempatan apa komunikasi terjadi dengan sistem bisnis lain atau partner bisnis lain?
6. Kegiatan mana yang memicu aktifitas tertentu?

Pertimbangan pertama dari studi kasus hasilnya adalah business use case di bawah ini (lihat Gambar 24):



Gambar 24. Potensial Business Use Case

Awalnya, business use case hanya dapat digambarkan dalam bentuk ringkas dan informal:

1. Prosedur **check-in** termasuk menyerahkan tiket, check-in bagasi, penentuan tempat duduk, dan mengeluarkan serta menyerahkan boarding pass.
2. Penumpang yang hanya memiliki tas tangan dapat menggunakan **express check-in**. Tidak melakukan check-in bagasi.
3. Saat **boarding**, boarding pass penumpang diperiksa pada boarding gate.
4. **Automated check-in** dilakukan tanpa bantuan dari petugas check-in, langsung pada mesin (layar). Bagasi tidak dapat dilakukan check-in.

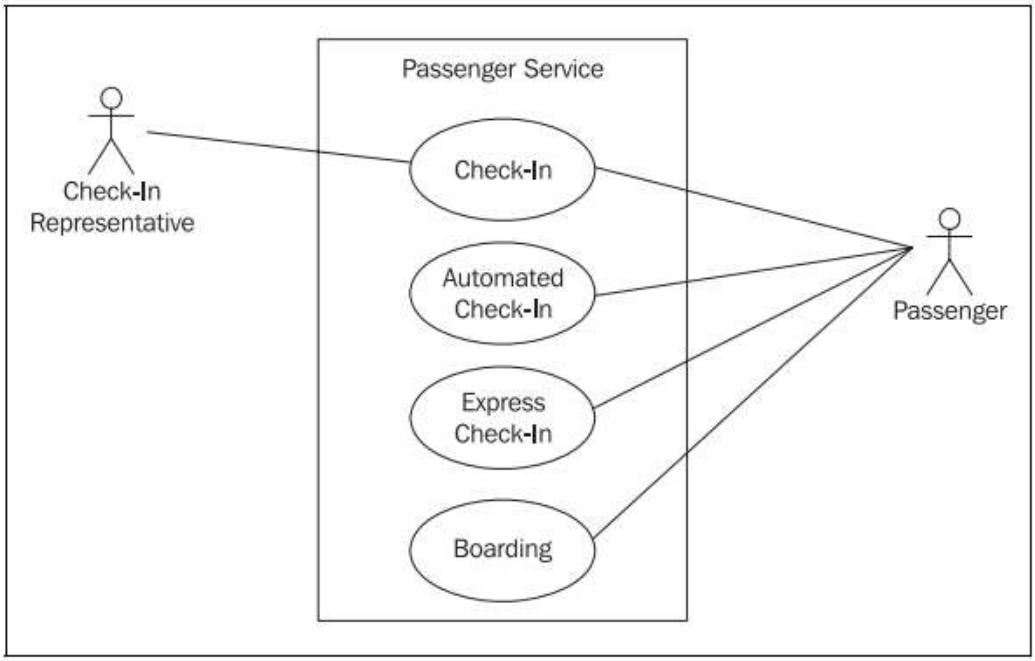
PRACTICAL TIPS

Dalam prakteknya, teknik pengamatan terbukti lebih efektif untuk mengidentifikasi business use case. Dengan mengamati orang yang terlibat pada proses bisnis, tercipta daftar aktifitas. Berdasarkan daftar aktifitas, aktifitas dapat dikelompokkan berdasarkan kegiatan yang menuntun ke business use case pertama.

MENGHUBUNGKAN BUSINESS USE CASE – SIAPA YANG DAPAT MENGGUNAKAN BARANG DAN JASA DARI SISTEM BISNIS?

Dengan menugaskan business use case ke actor, konsep pertama dari use case diagram berkembang (Gambar 25). Hal ini dicapai dengan menjawab pertanyaan di bawah ini:

- Customer atau partner bisnis mana yang memiliki ketersediaan fungsionaliti untuk mereka?



Gambar 25. Konsep Pertama dari Use Case Diagram

Dengan konsep pertama ini (Gambar 25) diperoleh bentuk dasar yang kedepannya use case diagram dapat diedit atau diperbaiki.

Penumpang dapat memilih antara normal check-in, automated check-in, dan express check-in. Penumpang jalan menuju ke gate dan menyerahkan boarding pass. Check-in representative dapat melakukan regular check-in, tetapi tidak dapat melakukan express check-in dan automated check-in.

MENGGAMBARKAN ACTOR – SIAPA DAN APA YANG ACTOR WAKILKAN?

Actor pada diagram harus diberikan nama sehingga dapat menjelaskan peranan yang digambarkannya. Disini, sangat penting bahwa terminology dari domain area, artinya istilah berorientasi bisnis, harus digunakan. Dengan tambahan nama, actor dapat di definisikan lebih lanjut dengan deskripsi. Pertanyaan untuk hal ini adalah:

- Bagaimana actor dapat di deskripsikan lebih lanjut? Misalnya, deskripsi ini termasuk area dari tanggungjawab, kebutuhan dari sistem, atau deskripsi formal dari peranannya. Tidak perlu takut untuk menambahkan deskripisi pekerjaan atau profil organisasi (contohnya perusahaan katering) – walaupun perusahaan katering tidak digambarkan dalam UML.

MENCARI LEBIH LANJUT BUSINESS USE CASE – HAL LAIN APA YANG HARUS DISELESAIKAN?

Setelah ditemukan beberapa business use case, mereka dapat digunakan sebagai titik awal untuk pertanyaan lebih lanjut. Dimulai dari business use case tertentu, pertanyaan di bawah ini dapat ditanyakan:

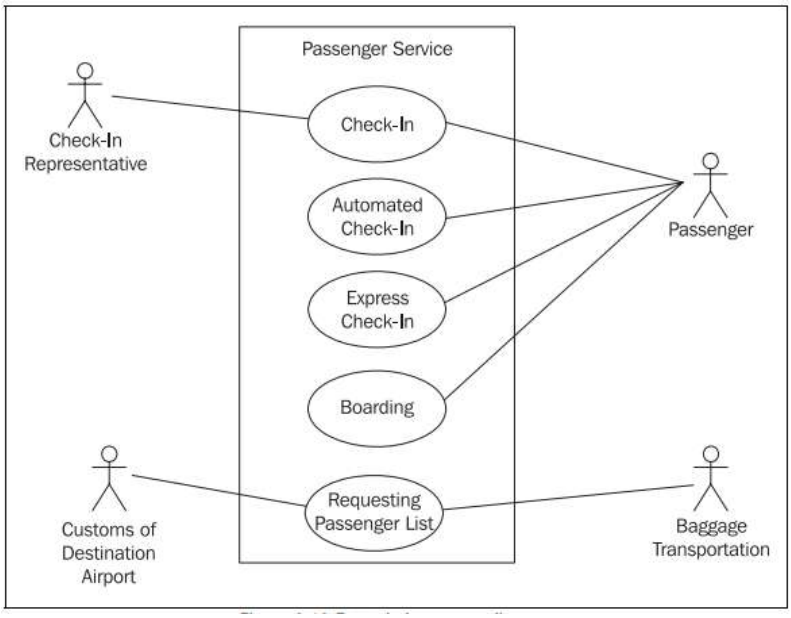
1. Apakah ada hal apapun yang harus diselesaikan pada suatu titik sebelumnya, sebelum menilai fungsionality tertentu?
2. Apakah ada hal apapun yang harus diselesaikan pada satu titik setelahnya, setelah melakukan business use case tertentu?
3. Apakah ada hal apapun yang harus diselesaikan jika tidak ada orang yang melakukan business use case tertentu?

Dalam melakukan hal ini, sangat penting untuk mempertimbangkan sistem bisnis yang tepat. Banyak kegiatan terjadi sebelum atau sesudah business use case yang terjadi diluar dari sistem bisnis dalam pertimbangan. Pada studi kasus, misalnya, memesan penerbangan atau menuju ke airport tidak masuk dalam sistem yang dipertimbangkan.

Dilihat dari dekat, terlihat penumpang sering pergi jalan-jalan dengan bagasi, yang dapat di check-in. Baggage transportation bertanggung jawab untuk memasukan bagasi kedalam pesawat. Baggage transportation dilakukan oleh organisasi independen, diketahui dengan nama handling agent. Karena itu, dipertimbangkan sebagai actor, lebih spesifiknya, penyedia jasa pihak ketiga. Pada diagram yang dibuat tidak peduli terhadap tugas yang dilakukan oleh karyawan dari pihak ketiga ini.

Sepuluh menit sebelum penerbangan berangkat, baggage transportation meminta daftar penumpang dari Passanger Service, termasuk setiap penumpang yang check-in, tetapi tidak naik ke pesawat. Berdasarkan list ini, semua bagasi yang terikat dikeluarkan lagi dari pesawat. Jika penerbangan adalah penerbangan internasional, petugas imigrasi dari negara yang dituju oleh penerbangan juga meminta daftar

penumpang. Hasilnya ditambahkan dua actor baru: baggange transportation dan custom authorities pada airport tujuan (Gambar 26)



Gambar 26. Penambahan Use Case Diagram

MENGUBAH BUSINESS USE CASE – APAKAH YANG SEHARUSNYA DIMASUKAN KEDALAM BUSINESS USE CASE?

Tidak diragukan, sangat sulit untuk mencari jumlah yang benar dari detail pada pembuatan model sistem bisnis. Jika hampir semua aktifitas dari actor pada business use case digabungkan, use case diagram kehilangan hampir semua kepentingannya. Jika aktifitas dirinci sepenuhnya, use case diagram menjadi kompleks dan berisi terlalu banyak aktifitas yang saling terkait sehingga sulit dikenali.

Untungnya, beberapa kriteria dapat membantu dalam menentukan jangkauan optimal dari business use case. Untuk tujuan ini, dapat ditanyakan pertanyaan di bawah ini:

1. Apakah business use case terdiri dari urutan terkait tingkah laku dari interaksi yang saling memiliki (sebuah urutan interaksi)? Hal yang termasuk dalam business use case harus dihubungkan langsung. Mengeluarkan boarding pass dan mencari bagasi yang hilang tidak berhubungan sama sekali. Business use case yang melanggar kriteria ini harus dibagi. Hal ini menghindari terjadinya business use case yang terlalu besar

2. Berapa banyak actor yang terlibat pada business use case? Business use case yang terlalu banyak memiliki actor harus dibagi. Hal ini menghindari terjadinya business use case yang terlalu besar
3. Apakah business use case menyampaikan barang dan jasa yang nyata dan relevan? Sebuah business use case seharusnya tidak menggambarkan langkah yang tidak lengkap, seperti contoh, menghitung jumlah bagasi. Lebih baik, setidaknya dalam kasus umumnya, harusnya menghasilkan keuntungan yang memiliki arti dari sudut pandang customer. Business use case yang melanggar kriteria ini harus digabungkan dengan business use case lain. Cara ini, business use case yang terlalu kecil dapat dihindari.
4. Apakah business use case tidak pernah dilakukan sendiri, tetapi selalu dalam urutan kombinasi dengan business use case lain? Sebuah business use case seharusnya tidak menggambarkan barang atau jasa yang hanya digunakan dalam kombinasi dengan barang atau jasa lainnya. Business use case yang melanggar kriteria ini, harus digabungkan dengan business use case lain. Hal ini menghindari business use case yang terlalu kecil.
5. Apakah business use case dimulai oleh actor? Business use case yang tidak dimulai oleh actor bukanlah sebuah use case tetapi aktifitas internal yang digambarkan dalam tampilan internal dari sistem bisnis.

Review dari business use case yang sudah ada sebagai dasar dari pertanyaan di atas dapat menuntun ke penggabungan atau pemecahan dari business use case.

DOKUMENTASI BUSINESS USE CASE – APA YANG TERJADI DALAM BUSINESS USE CASE?

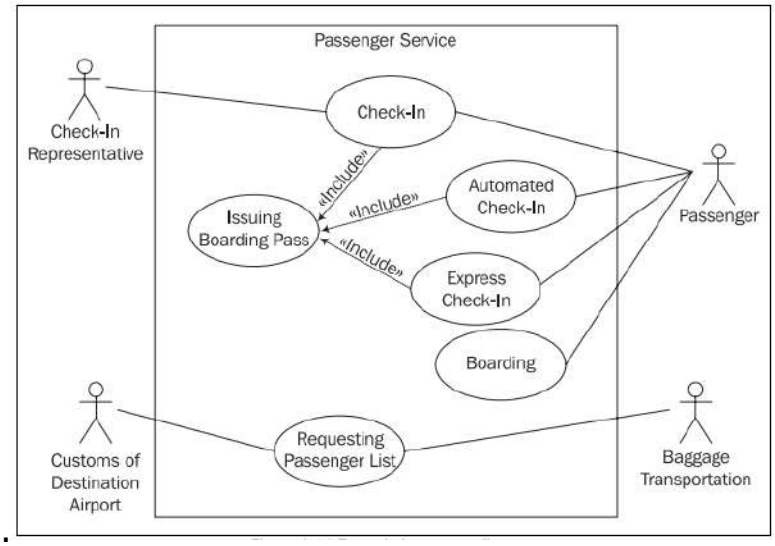
Untuk mengerti sebuah business use case, informasi dari use case diagram tidaklah cukup. Rantai setiap interaksi dari dan beberapa skenario dibelakang setiap business use case harus di gambarkan. Artinya barang dan jasa yang disediakan oleh sistem bisnis harus digambarkan, yaitu urutan kegiatan dari sudut pandang customer atau partner bisnis.

Dalam tambahan penjelasan verbal, dokumentasi dalam activity diagram dan sequence diagram terbukti sangat berharga.

MODEL RELATIONSHIP ANTARA BUSINESS USE CASE – KEGIATAN APA YANG DILAKUKAN BERULANG KALI?

Jika diperhatikan beberapa bagian dari interaksi adalah sama dibeberapa business use case, interaksi sama ini dapat diambil dan dijadikan satu menjadi business use case sendiri. Business use case baru ini dapat digunakan pada business use case lain dengan **hubungan include (include relationship)**

Pada studi kasus, business use case **issuing boarding pass** belum ditugaskan. Boarding pass terjadi dan dikeluarkan pada saat check-in. Pada satu titik saat business use case check-in, express check-in, dan automated check-in, boarding pass dikeluarkan. (Lihat Gambar 27)



Gambar 27. Penambahan Use Case Diagram

VERIFIKASI VIEW – APAKAH SUDAH BENAR?

Semua diagram dan data sudah harus diverifikasi oleh knowledge carriers. Yang harus ditanyakan kepada knowledge carrier pada setiap diagram adalah:

- Apakah semua isi yang ada dalam diagram sudah benar dan lengkap?

Walaupun knowledge carriers dapat membaca dan mengerti diagram dengan sendirinya, diagram tetap harus dibacakan kepada knowledge carriers. Hanya dengan langkah terakhir ini lingkaran lengkap. Hasilnya adalah tampilan yang sudah diverifikasi, yang mencerminkan pemahaman bersama dari sistem bisnis dan proses bisnis.

Use Case Diagram yang sudah selesai dapat diverifikasi dengan daftar check di bawah ini:

1. **Completeness:** Use Case Diagram sudah lengkap jika tidak ada lagi business use case lebih lanjut dalam sistem. Semua barang dan jasa yang tersedia untuk customer dan partner dari sistem bisnis sudah digambarkan dalam bentuk business use case (jika diperlukan, business use case dapat disebar menjadi beberapa diagram)
2. **Extent:** semua business use case yang masuk kedalam use case diagram adalah business use case nyata, artinya business use case sudah sesuai dengan definisi dari business use case.
3. **Degree of Detail:** tingkatan dari detail pada business use case harus sesuai dengan persyaratan di bawah ini:
   Sebuah business use case mewakili tingkah laku urutan interaksi yang jelas.

   Sebuah business use case diawali oleh actor, dan hanya memiliki sedikit actor

   Sebuah business use case mewakili sebuah fungsionalitas yang nyata dan hasil yang relevan.
4. **Relationship between business use case:** hubungan include harus digunakan dengan wajar
5. **Naming dan describing**: nama dari business use case menggambarkan fungsionalitas yang disediakan oleh sistem bisnis. Pemberian nama harus sesuai dengan terminology normal mewakili peran yang digunakannya.
6. **Actor**: actor pada use case diagram mewakili peran yang dilakukan oleh orang luar, organisasi, atau bisnis sistem lain selama interaksi.

PRACTICAL TIPS

Pada saat menggunakan use case diagram untuk memodelkan sistem bisnis dan proses bisnis, disarankan untuk menurunkan tingkat abstarksinya. Untuk kelengkapan dari diagram dan komunikasi antara pihak yang terlibat, sangat baik menambahkan redudansi daripada abstrak yang banyak.

Merupakan dasar penting bahwa terminology dari proses bisnis atau organisasi digunakan, dan dekripsi dari business use case dipilih agar dapat dipahami secara intuitive.

Terminologi pada bidang **Information Technology (IT)** bukan merupakan bagian dari use case didagram pada tingkatan proses bisnis. Penggabungan istilah proses bisnis dan komunitas IT mendapatkan hasil yang buruk. Kenyataannya, sering ditemukan use case yang sudah terlalu dekat dengan IT pada tingkatan proses bisnis, seperti updating a customer index. Hal ini menuntun kepada kebingungan pada dua aspek:

1. User – artinya orang yang terlibat dalam proses bisnis, dan orang yang tidak paham dengan terminology IT – tidak mengerti business use case. Semenjak business use case menggambarkan kebutuhan kinerja untuk sistem bisnis, sistem bisnis dan proses bisnis tidak dapat dimengerti, oleh karena itu tidak dapat diverifikasi. Dalam proyek dengan formula business use case yang buruk, departemen IT menampilkan business use case kepada user untuk verifikasi dan menerima hanya satu jawaban:"Men throwing arrows?!"
2. Detail teknis pada level dari business use case menggangu spesifikasi kebutuhan proses bisnis dari sistem.

KESIMPULAN

1. Dua lingkungan target dari IT System:
   a. Integrasi pada level proses bisnis
   b. Integrasi pada leve IT-System
2. Proses bisnis adalah prosedur datau kegiatan dengan maksud untuk mendapatkan tujuan.

3. Berdasarkan Workflow Reference Model, proses atau proses bisnis adalah serangkaian proses yang terkoordinasi (secara paralel dan/atau serial) yang saling terhubung untuk mencapai tujuan yang sama. Aktifitas tersebut dapat berisi aktifitas manual dan/atau alur kerja aktifitas.
4. Dalam terminilogi bisnis, bisnis sistem adalah rantai nilai tambah (value-added chain), yang menggambarkan proses nilai tambah (value-added process).
5. Contoh Studi kasus yang digunakan adalah Passenger Service
6. Unified Modeling Language adalah bahasa visual untuk menentukan, membangun, dan mendokumentasikan artifact dari sistem"
7. Dua sudut pandang pada Pemodelan Bisnis Sistem (Business System Modeling):
    a. Eksternal View
    b. Internal View
8. Use Case adalah spesifikasi dari rangkaian aksi yang dilakukan oleh sistem, yang menyebabkan hasil yang tampak yaitu sebuah nilai khusus untuk satu atau lebih aktor atau stakeholder lain dari sistem.
9. Actor adalah customer atau partner bisnis, yang menggunakan output dari sistem bisnis.
10. Elemen Internal View: Use Case Diagram, Activity Diagram, Seqeunce Diagram.
11. Elemen dari Use Case Diagram: Actor, Association, Business Use Case, Include Relationship, Subject.
12. Cara membaca use case dapat dimulai dari Actor atau Use Case dengan melihat dari association yang terjadi antara actor dengan use case. Association yang terjadi menandakan actor terlibat pada sebuah use case atau use case melibatkan seorang actor.
13. Membuat Use Case Diagram Eksternal View perlu memperhatikan langkah-langkah berikut:
    a. Mengumpulkan sumber infomrasi – bagaimana saya dapat mengetahui hal itu?
    b. Identifikasi potensial Actor – siapa partner dan customer yang menggunakan barang dan jasa dari sistem bisnis?
    c. Identifikasi business use case – barang dan jasa apa yang dapat digunakan actor?

d. Menghubungkan business use case – siapa yang dapat menggunakan barang dan jasa dari sistem bisnis?
e. Mencari lebih lanjut business use case – hal lain apa yang harus diselesaikan?
f. Menguah business use case – apakah ayng seharusnya dimasukan kedalam business use case?
g. Dokumentasi business use case – apa yang terjadi dalam business use case?
h. Verifikasi View – apakah sudah benar?

SOAL LATIHAN

1. Dua lingkungan target dari IT sistem baru yaitu _____
   a. Integrasi pada level proses bisnis, dan pada level IT System
   b. Integrasi pada level IT System, dan pada level Internal
   c. Integrasi pada level internal dan eksternal
   d. Integrasi pada level proses bisnis, dan pada level Eksternal
2. Proses bisnis adalah _____
   a. Prosedur yang dapat berjalan secara berurutan atau bersamaan yang dapat diselesaikan dalam beberapa tahap
   b. Value-added chain, yang menggambarkan value-added process untuk menghasilkan keuntungan ekonomis.
   c. Prosedur atau kegiatan dengan maksud untuk mencapai tujuan
   d. Prosedur yang digambarkan modelnya untuk seluruh organisasi
3. Aktifitas adalah _____
   a. Prosedur yang dapat berjalan secara berurutan atau bersamaan yang dapat diselesaikan dalam beberapa tahap
   b. Value-added chain, yang menggambarkan value-added process untuk menghasilkan keuntungan ekonomis.
   c. Prosedur atau kegiatan dengan maksud untuk mencapai tujuan
   d. Bahasa visual untuk menentukan, membangun, dan mendokumentasikan artifak dari sistem
4. Sistem bisnis adalah _____

a. Prosedur yang dapat berjalan secara berurutan atau bersamaan yang dapat diselesaikan dalam beberapa tahap
b. Value-added chain, yang menggambarkan value-added process untuk menghasilkan keuntungan ekonomis.
c. Prosedur yang digambarkan modelnya untuk seluruh organisasi
d. Bahasa visual untuk menentukan, membangun, dan mendokumentasikan artifak dari sistem

5. Unified Modeling Language adalah _____
   a. Prosedur yang dapat berjalan secara berurutan atau bersamaan yang dapat diselesaikan dalam beberapa tahap
   b. Value-added chain, yang menggambarkan value-added process untuk menghasilkan keuntungan ekonomis.
   c. Prosedur yang digambarkan modelnya untuk seluruh organisasi
   d. Bahasa visual untuk menentukan, membangun, dan mendokumentasikan artifak dari sistem
6. Pada Business System Modeling terdapat dua sudut pandang, yaitu _____
   a. Internal View dan Eksternal View
   b. Internal View dan Structural View
   c. Structural View dan Interaction View
   d. Eksternal View dan Structural View
7. View yang melibatkan orang luar adalah _____
   a. Structural View
   b. Internal View
   c. Static View
   d. Eksternal View
8. View yang menggambarkan bagaimana sistem bisnis menyediakan jasa adalah _____
   a. Structural View
   b. Internal View
   c. Static View
   d. Eksternal View

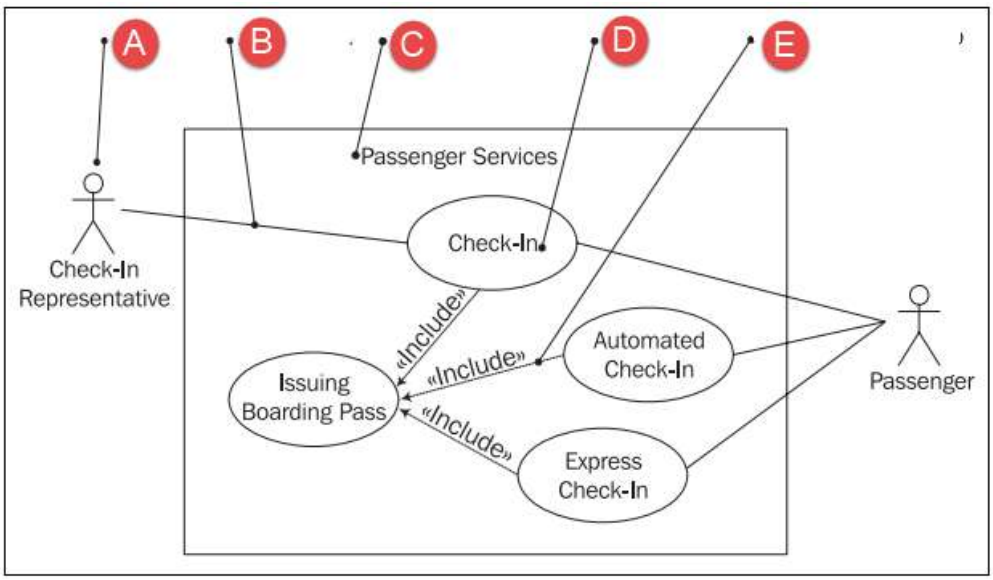
9. Di bawah ini merupakan elemen/diagram pada Internal View, KECUALI _____
    a. Package Diagram
    b. Use Case Diagram
    c. Sequence Diagram
    d. Activity Diagram
10. Elemen/diagram yang terdapat di Internal View dan juga Eksternal View adalah _____
    a. Package Diagram
    b. Use Case Diagram
    c. Sequence Diagram
    d. Activity Diagram
11. Spesifikasi dari rangkaian aksi yang dilakukan oleh sistem adalah _____
    a. Actor
    b. Activity
    c. Use Case
    d. Association
12. Business Use Case dimulai dari _____
    a. Actor
    b. Activity
    c. Use Case
    d. Association
13. Elemen/diagram dari eksternal view yang menampilkan actor, business use case dan hubungannya adalah _____
    a. Sequence Diagram
    b. Class Diagram
    c. Activity Diagram
    d. Use Case Diagram
14. Elemen/diagram dari eksternal view yang menggambarkan prosedur, use case, proses bisnis dari sistem bisnis adalah _____
    a. Sequence Diagram
    b. Class Diagram
    c. Activity Diagram

d. Use Case Diagram

15. Elemen/diagram dari eksternal view yang menampilkan urutan secara kronologis dari interaksi adalah _____

a. Sequence Diagram
b. Class Diagram
c. Activity Diagram
d. Use Case Diagram

Gambar Soal No. 16 dan No. 17



16. Elemen Use Case Diagram yang menggambarkan sebuah peran dari orang luar (outsider) ketika berinteraksi dengan sistem bisnis ditunjukan dengan huruf _____

a. A
b. B
c. C
d. D

17. Elemen Use Case Diagram yang menggambarkan sistem bisnis yang memiliki satu atau lebih dari business use case di dalamnya ditunjukan dengan huruf dan bernama _____

a. A – Actor
b. B – Subject
c. C – Subject

d. D - Actor

18. Membaca use case diagram dapat dimulai dari _____
    a. Actor atau association
    b. Use case atau include
    c. Use case atau subject
    d. Actor atau use case
19. Salah satu langkah dalam membuat use case diagram adalah mengumpulkan informasi, di bawah ini adalah teknik yang dapat dilakukan untuk mendapatkan informasi sehingga dapat memahami proses bisnis, KECUALI _____
    a. Mengamati karyawan yang sedang bekerja
    b. Mengisi formulir yang ada
    c. Melakukan wawancara
    d. Memberikan survey
20. Beirkut ini BUKAN termasuk daftar cek untuk melakukan verifikasi pada Use Case Diagram
    a. Completeness
    b. Extent
    c. Ommitness
    d. Degree of Detail

UNIVERSITAS BUDI LUHUR
FAKULTAS TEKNOLOGI INFORMASI

MODUL PERKULIAHAN #3
MODELING BUSINESS SYSTEM: ACTIVITY DIAGRAM

Capaian Pembelajaran	:	Mahasiswa memahami dan dapat menggunakan activity diagram
Sub Pokok Bahasan	:	1.18. Activity Diagram a. Membaca Activity Diagram b. Membuat Activity Diagram 1.19. Kesimpulan 1.20. Soal Latihan
Daftar Pustaka	:	Grässle, P., Baumann, H. and Baumann, P., 2005. UML 2. 0 in Action: A Project-based Tutorial. Packt Publishing Ltd.

ACTIVITY DIAGRAM

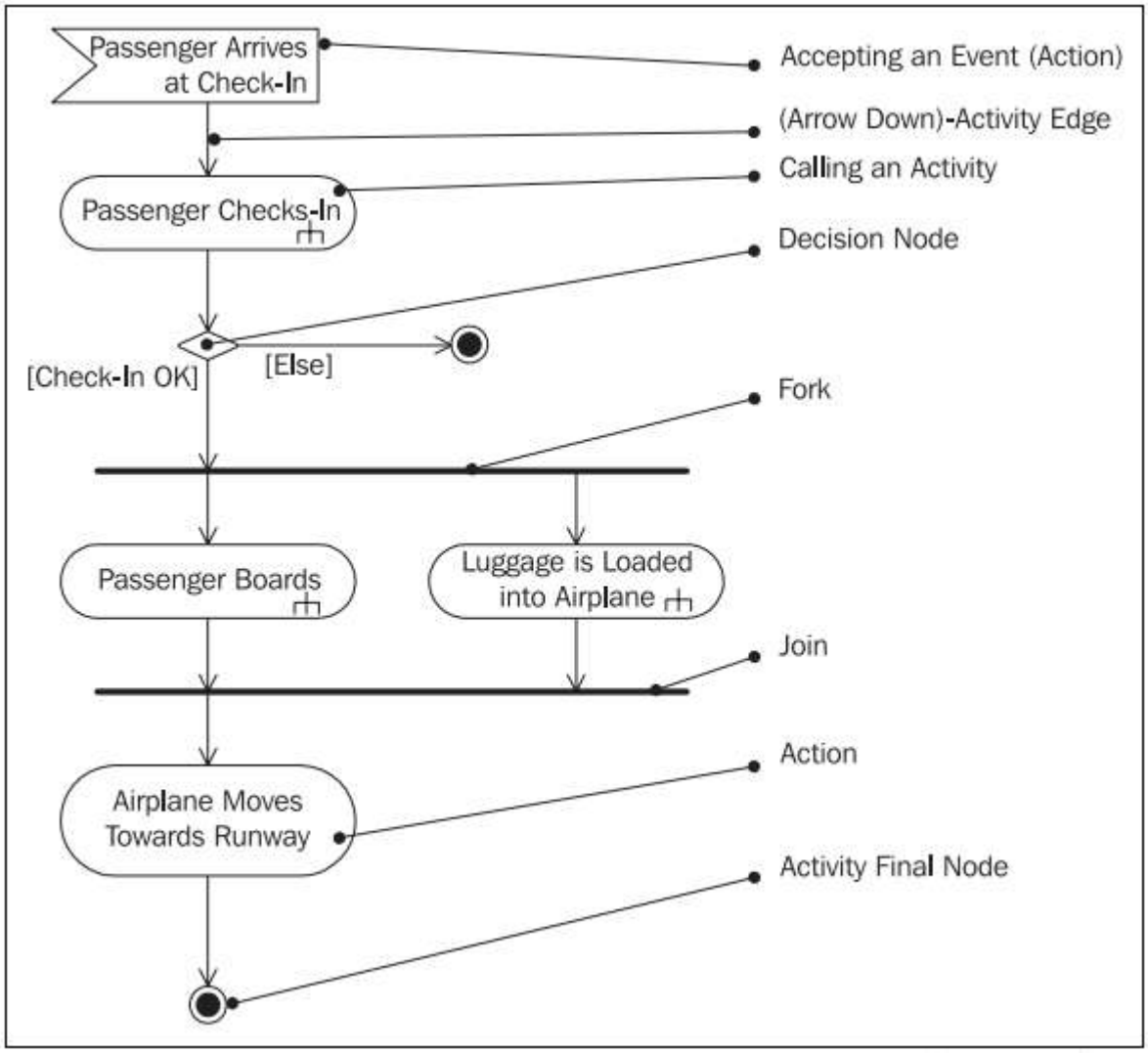
Activity diagram, yang berhubungan dengan rencana alur program (flowchart), digunakan untuk mengilustrasikan aktifitas. Pada eksternal view, activity diagram digunakan untuk mendeskripsikan proses bisnis yang menggambarkan fungsionalitas dari sistem bisnis.

Berbeda dengan use case diagram, pada activity diagram sangat jelas apakah actor dapat melakukan business use case bersamaan atau independen satu dengan lainnya.

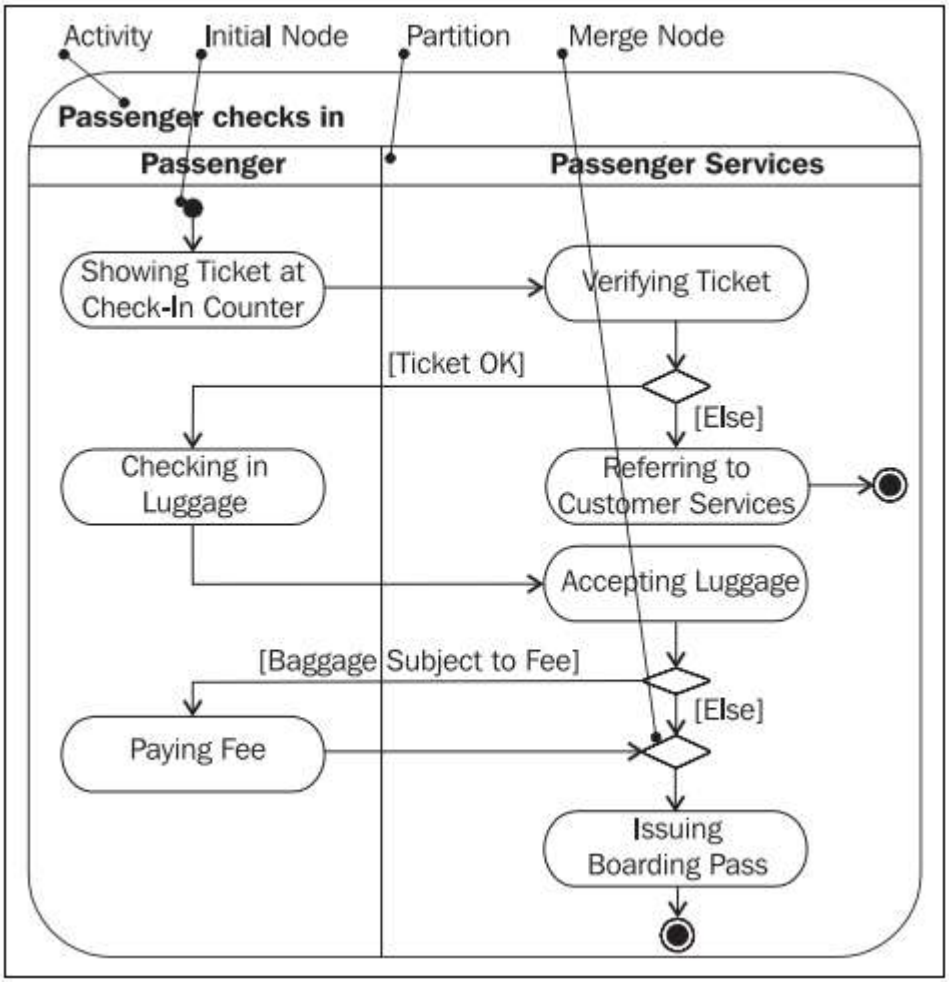
Activity diagram memungkinkan berfikir secara fungsional. Purist dari pendekatan berbasis obyek tidak menyukai fakta ini. Faktanya hal ini menjadi keuntungan, semenjak user dari metodologi berorientasi obyek, dan juga user dari pola pikir fungsional, menemukan format umum dan mudah dipahami, yang menjadi alat bantu signifikan dalam pemodelan proses bisnis.

Karena dengan eksplisit dapat menggambarkan kegiatan parallel, activity diagram cocok untuk mengilustrasikan proses bisnis, semenjak proses bisnis jarang terjadi pada pola linear dan sering memperlihatkan paralelisme.

Activity diagram dapat dikembangkan menjadi beberapa level detail, mereka dapat diperbaiki langkah demi langkah. Pada eksternal view, activity diagram, seperti use case diagram, secara khusus menggambarkan proses bisnis dan aktifitas dari sudut pandang luar. Memperbaiki diagram bukan berarti menggambarkan detail proses yang dilakukan dalam bisnis sistem, yang sering menuntun ke perubahan yang tidak diperhatikan pada internal view (Gambar 28)

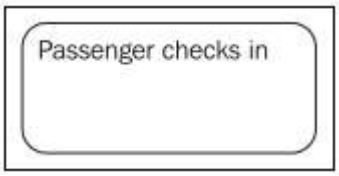


Gambar 28. Activity Diagram "Passenger Service" Dengan Detail Tingkat Rendah ("High Level")



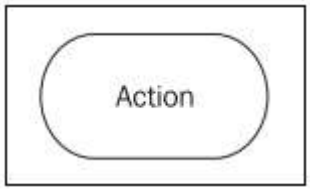
Gambar 29. Activity Diagram Dari Kegiatan "Passenger Checks In"

Activity: sebuah activity diagram diilustrasikan dengan satu individual aktifitas. Dalam konteks, sebuah aktifitas menggambarkan sebuah proses bisnis (Gambar 29). Elemen dasar dari activity adalah action dan elemen kontrol (decision, division, merge, initiation, end, dll):



Elemen saling terhubung dengan yang disebut "activity edges" dan dari "control flow", yang biasanya disebut dengan 'flow'. Eksekusi sebuah aktifitas dapat berisi aliran parallel. Batasan dapat mengelilingi activity, artinya semua activity diagram.

Action: sebuah action adalah langkah individual dalam aktifitas, contohnya, langkah perhitungan tidak di dekonstruksi lebih lanjut. Hal ini tidak perlu berarti bahwa aksi tidak dapat dibagi lagi di dunia nyata, tetapi pada diagram ini tidak akan diperbaiki lagi lebih lanjut:



Action dapat memproses informasi input dan output. Output dari satu action dapat menjadi input dari action selanjutnya dalam sebuah aktifitas. Action tertentu dapat memanggil action lain, menerima sebuah event, dan mengirim sinyal.

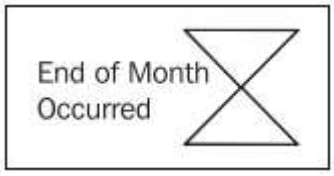
Calling an Activity (Action): dengan simbol ini sebuah aktifitas dapat dipanggil dari acitivty lain. Memanggil, artinya diri sendiri, merupakan sebuah action; keluaran dari pemanggilan adalah activity lain:

Dengan cara ini, aktifitas dapat bersarang didalamnya satu dengan yang lainnya dan dapat digambarkan dengan tingkatan detail yang berbeda.

Accepting an Event (Action): action ini menunggu even terjadi. Setelah even diterima, aliran yang datang dari action ini (dan didefinisikan pada activity diagram) kemudian dijalankan. Menerima even merupakan elemen penting dari proses bisnis pada activity diagram:

Banyak proses bisnis diawali dengan even, contohnya, memproses pesanan berdsarkan bukti pesanan, atau pengantaran berdasarkan bukti pembayaran.

Accepting a Time Even (Action): pada suatu titik waktu, action ini memulai aliran pada activity diagram. Simbol jam pasir dapat digunakan untuk menggambarkan penerimaan dari sebuah even waktu:

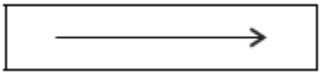


Contoh khas dari time even adalah memicu pengingat setleah batas waktu pembayaran lewat.

Sending Signal (Action): mengirimkan sinyal artinya bahwa sinyal telah dikirim ke activity penerima:

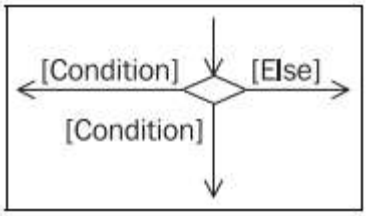
Activity penerima menerima sinya dengan action "accepting an event" dan bereaksi sesuai dengan activity, artinya berdasarkan aliran yang berasal dari node pada activity diagram.

Edge (Control Flow): Edges, digambarkan dengan panah, menghubungkan komponen individual dari activity diagram dan mengilustrasikan kontrol aliran pada activity:



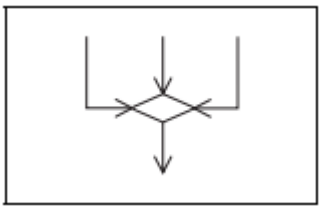
Dalam kontrol aliran sebuah panah masuk memulai langkah tunggal dari activity; setelah langkah selesai aliran akan berlanjut sepanjang panah keluar. Nama dapat diberikan pada edge (dekan dengan panah).

Decision Node: bentuk diamod di bawah ini menggambarkan sebuah titik cabang kondisi atau node decision. Sebuah decision memiliki satu input dan dua atau lebih output



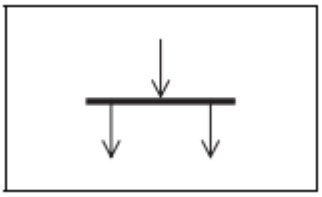
Setiap output memiliki kondisi yang ditempel padanya, yang ditulis diantara bracket. Jika sebuah kondisi terpenuhi, aliran akan lanjut sepanjang sesuai output. Sebuah 'Else' output dapat diberikan sepanjang aliran dapat diproses jika tidak ada kondisi yang terpenuhi.

Merge Node: bentuk diamond di bawah ini memiliki beberapa input dan hanya satu output:



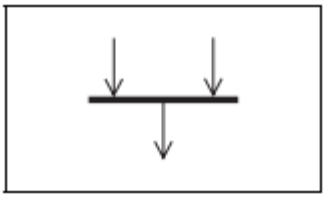
Tujuan dari merge ada menyatukan aliran. Input tidak harus synchronized; jika sebuah aliran sampai merge akan berlanjut ke output tanpa harus menunggu aliran lainnya.

Fork: pada percabngan dari aliran pada satu atua model parallel dapat menggunakan garis synchronous, yang digambarkan dengan garis horizontal atau vertika yang tebal.



Percabangan memungkinkan aliran parallel dalam aktifitas. Sebuah fork memiliki satu input dan dua atau lebih output.

Join: Untuk penggabungan dari dua atau lebih aliran parallel juga dapat menggunakan sinkronisasi bar, yang digambarkan dengan haris tebal secara horizontal maupun vertical:



Dalam penggabungan terjadi sinkronisasi, artinya aliran akan berlajut hanya jika setelah *semua* aliran masuk sampai ke titik penggabungan. Join memiliki dua atau lebih input dan satu output

Intial Node: initial node adalah titik awal dari aktifitas. Sebuah aktifitas dapat memiliki lebih dari satu initial node; dalam kasus ini beberapa aliran dimulai pada permulaan sebuah aktifitas:

Sangat mungkin sebuah aktifitas tidak memiliki initial node, tetapi diawali oleh sebuah even (action: menerima even)

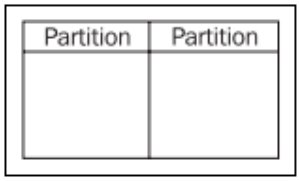
Activity Final Node: activity final node menunjukan bahwa semua aktifitas sudah selesai. Sebuah activity diagram dapat memiliki lebih dari satu exit dalam bentuk activity final node:

Jika beberapa aliran parallel terdapat dalam sebuah aktifitas, semua aliran berhenti pada saat aktifitas mencapai final node.

Flow Final Node: sebuah flow final node mengakhiri satu buah flow. Tidak seperti activity final node, yang mengakhiri seluruh aktifitas, sampai pada flow final node tidak memiliki efek pada aliran parallel lainnya yang diproses dalam aktifitas pada satu titik waktu yang sama:

Dengan cara ini, aliran parallel dapat diakhiri secara individu dan secara selektif.

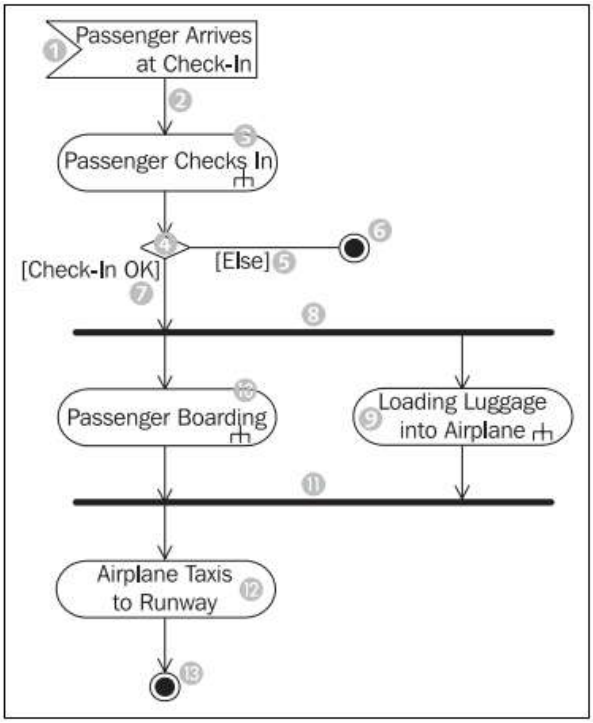
Activity Partition: elemen individual dari activity diagram dapat dibagi menjadi individual area atau 'partisi'. Beberapa kriteria yang menuntun ke pembuatan dari partisi: entitas organisasi, cost center, lokasi, dll:



Langkah individual dari sebuah aktifitas akan ditugaskan pada partisi. Setiap partisi adalah rangkaian terpisah dari partisi yang ada disebelahnya dengan jalur horizontal atau vertical yang berkelanjutan; berdasarkan akar kata terdapat istilah **swimlanes**. Setiap partisi menerima nama. Partisi dapat diatur dalam bentuk dua dimensi; pada kasus ini activity diagram dibagi menjadi kolom individual seperti grid.

1.14. MEMBACA ACTIVITY DIAGRAM

Mulai membaca dimulai dari **initial node**, atau pada Gambar 30 diawali dengan menerima even **passenger arrives at check-in** (1), dan berlanjut sepanjang panah dari control flow (2). Action selanjutnya **passenger checks in** (3) artinya bahwa pada titik ini aktifitas **'passenger checks in'** telah diproses. Hal ini digambarkan lebih detail pada activity diagram lain yang ditandai oleh 'fork' pada simbol action:



Gambar 30. Activity Diagram

Jika mengikuti control flow, selanjutnya akan sampai pada cabang kondisi atau decision node (4); jika check-in OK maka langkah selanjutnya adalah mengikuti sepanjang control flow. Selain itu (5), penumpang tidak dapat terbang dan tugas dari passenger service selesai. Hal ini dapat dilihat pada titik hitam dengan border – activity final node.

Setelah sukses melakukan check-in (7) selanjutnya bertemu dengan bar kotak hitam. Semua panah yang berasal dari bar ini (7) melambangkan aliran yang diproses bersamaan. Sementara bagasi dimasukan ke pesawat (9) penumpang masuk ke pesawat (10). Antara titik (8) dan titik (11) aliran adalah independen antara satu dengan lainnya. Pada bar kotak hitam kedua (11) proses aliran yang dilakukan bersamaan (9 dan 10) digabungkan, artinya bahwa hanya ketika penumpang di dalam pesawat (10) *dan* bagasi sudah dimasukan ke dalam pesawat (9), control flow akan berlanjut di bawah bar kotak hitam (11). Pada contoh, satu atau beberapa action (12) dan menyusul mengikuti final state (13), artinya bahwa setelah penumpang di dalam pesawat (10) dan bagasi sudah masuk ke pesawat (9), pesawat dapat menunggu dengan menuju ke runway (12). Seperti yang terlihat action terakhir adalah airplane taxis toward runway (12) yang hanya di definisikan sebagai action tunggal, walaupun proses ini sangat kompleks dan dapat digambarkan dengan bentuk activity diagram lainnya. Tetapi, dalam konteks, tidak perlu menggambarkan langkah ini secara detail.

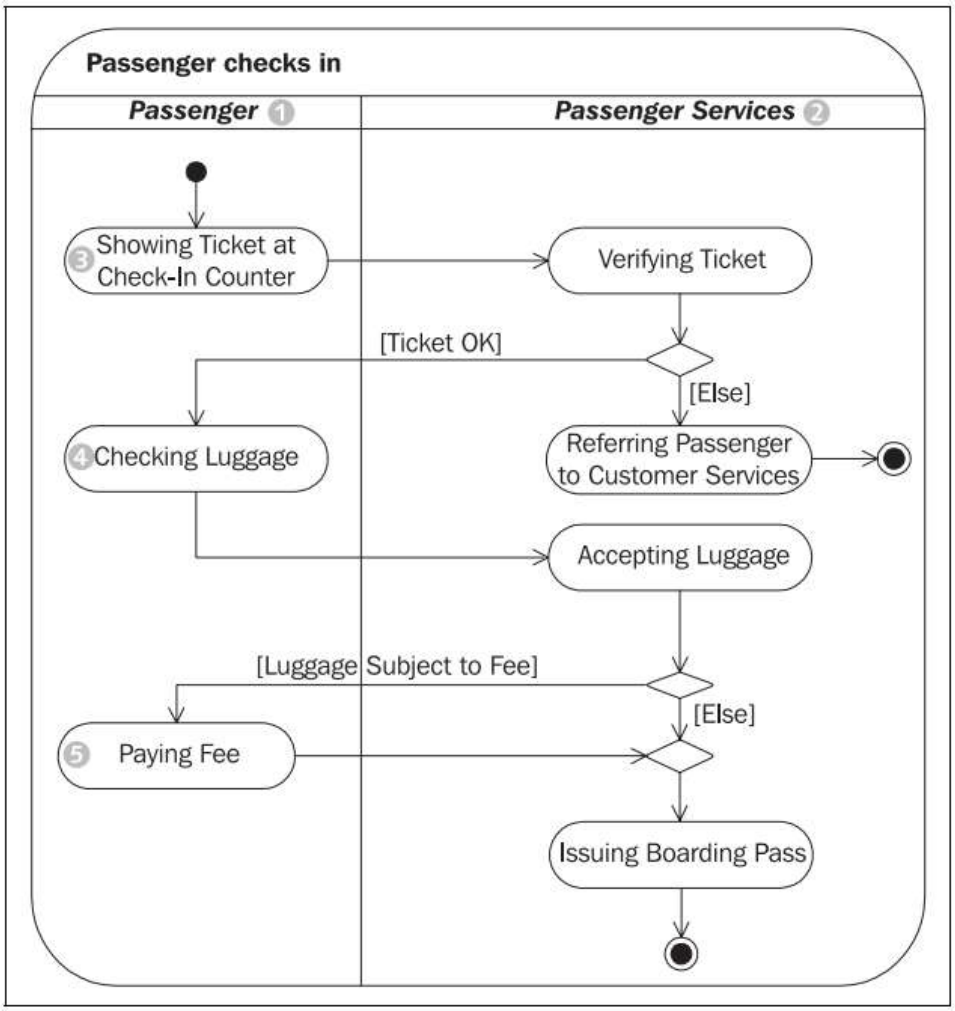
Acitivity Diagram pada Gambar 31 dibagi menjadi dua partisi: **passenger** (1) dan **passenger service** (2). Passenger, misalnya, melakukan **showing ticket at check-in caounter** (3), **checking luggage** (4), dan **paying fee** (5). Semua action yang lain diletakan pada partisi (swimlane) dari passenger service (2) dan dilakukan oleh passenger service.

1.15. MEMBUAT ACTIVITY DIAGRAM

Seperti biasa, akan ditunjukan penggunaan terminology yang sesuai dan penamaan yang cocok untuk action dan aktifitas sangatlah penting untuk kejelasan dan kelengkapan dari activity diagram. Jangan putus asa jika membutuhkan waktu

berjam-jam untuk mencari nama yang sesuai – pada kebanyakan kasus upaya yang dilakukan sangat sepadan.

Disarankan memulai activity diagram yang berisi detail tingkat rendah ('high level'), yang dapat memperluas beberapa business use case. Hal ini memberikan overview baik dari rangkaian interaksi antara customer dan partner dalam sistem bisnis.



Gambar 31. Activity Diagram Dengan Partisi

Nanti, pada langkah lebih detail skenario dari business use case dapat digambarkan dengan activity diagram. Jika business use case dibuat dari beberapa skenario berbeda, setiap skenario digambarkan dalam activity diagram.

Berikut adalah checklist langkah-langkah yang diperlukan untuk membuat activity diagram:

Checklist Membuat Activity Diagram pada Eksternal View:

5. Mengumpulkan sumber informasi – bagaimana saya harus melakukannya?
6. Mencari aktifitas dan action – apa yang harus dilakukan ketika actor menggunakan barang dan jasa yang ditawarkan?
7. Menggunakan actor dari business use case – siapa yang bertanggung jawab pada setiap action?
8. Menghubungkan action – dalam urutan bagaimana action diproses?
9. Memperbaiki Activity – apakah activity diagram lain dapat ditambahkan?
10. Memverfikasi tampilan – apakah semuanya sudah benar?

Urutan yang diberikan pada langkah di atas dipilih dengah hati-hati. Akan tetapi, urutan tesebut tidak wajib, dikarenakan dalam prakteknya langkah kerja individual sering tumpang tindih.

MENGUMPULKAN SUMBER INFORMASI – BAGAIMANA SAYA HARUS MELAKUKANYA?

Untuk membuat activity diagram dapat menggunakan infomrasi yang sudah dikumpulkan dalam pembuatan use case diagram. Selain itu, saran yang sama juga diberikan untuk membuat activity diagram.

MENCARI AKTIFITAS DAN ACTION – APA YANG HARUS DILAKUKAN KETIKA ACTOR MENGGUNAKAN BARANG DAN JASA YANG DITAWARKAN?

Pada langkah ini, dapat dimulai dari use case diagram. Pada langkah pertama dapat memperolah aktifitas dari business use case. Menjawab pertanyaan di bawah ini dapat membantu mendapatkan akatifitas dan action:

1. Langkah kerja mana yang dibutuhkan untuk menjalankan business use case, artinya tahap mana yang dibutuhkan untuk memperolah dan memproses barang dan jasa?
2. Apa yang dilakukan actor individual?

3. Jika beberapa actor terlibat apda business use case, langkah kerja mana yang dilakukan oleh setiap individual actor?
4. Even mana yang mengawali langkah kerja tertentu?
5. Action mana yang sangat luas sehingga harus diperbaiki pada acitivty diagram lain?

Pada studi kasus, diketahui langkah kerja untuk passenger service sebagai berikut:

1. Penumpang check in (diperoleh dari use case diagram); hal ini menyebabkan dikeluarkannya boarding pass melalui passenger service
2. Penumpang boarding ke pesawat (diperolah dari use case diagram)

Tambahannya, terdapat langkah dan even lain:

1. Penumpang tiba pada counter check-in dan menunjukkan tiketnya; even ini mengawali aktifitas check-in
2. Bagasi dimasukan kedalam pesawat oleh baggage transportation.

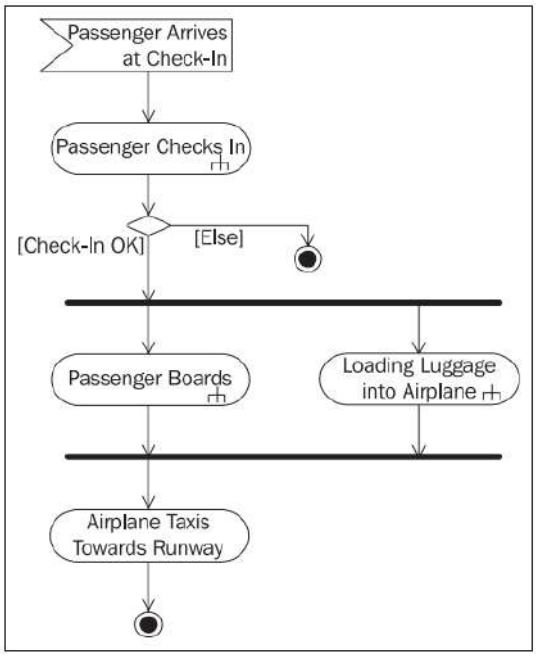
Awalnya, seperti aktifitas di atas, aktifitas dapat digambarkan dalam bentuk informal. Terkadang ditemukan dokumentasi proses dari sistem yang sudah ada, baik dalam bentuk informasi atau terstruktur, yang dapat digunakan sebagai dasar untuk mendapatkan aktifitas dan action.

MENGHUBUNGKAN ACTION – DALAM URUTAN BAGAIMANA ACTION DIPROSES?

Menghubungkan action dan aktifitas yang sebelumnya sudah disebutkan kedalam aliran menghasilkan activity diagram tahap awal. Aliran ini disebut **control flow**. Pertanyaan di bawah ini akan membantu untuk mengembangkan sebuah control flow:

1. Dalam urutan apa sebuah action di proses?
2. Kondisi mana yang harus dipenuhi agar sebuah action dapat dijalankan?
3. Apakah percabangan (branches) diperlukan?
4. Action mana yang terjadi bersamaan?
5. Apakah penyelesaian sebuah action diperlukan sebelum aliran dapat berlanjut ke action yang lain?

Passenger checks in, passenger board, dan **loading luggage into airplane** merupakan aktifitas yang komples, setiap aktfitas dapat di detailkan dalam activity diagram lain. Sebuah 'fork' dengan simbol action menandakan hal ini (Gambar 32):



Gambar 32. High-level Activity Diagram Lintas Beberapa Business Use Case

MEMPERBAIKI ACTIVITY – APAKAH ACTIVITY DIAGRAM LAIN DAPAT DITAMBAHKAN?

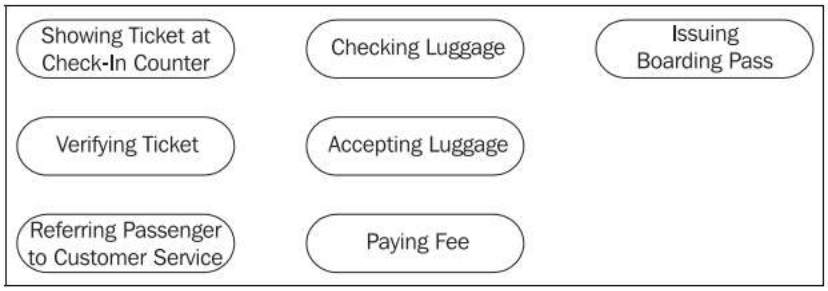
Seperti yang terlihat pada Gambar 32, sangat perlu untuk mengubah beberapa langkah proses. Pada tahap ini, ditampilkan aktfitas dari **passenger checks in** lebih detail.

Ketika passenger check in, pertama penumpang menunjukan tiketnya di counter check-in. tiket akan diperiksa kebenarannya. Jika tiket tidak OK penumpang diarahkan ke customer service. Jika tiket OK penumpang check in bagasinya. Jika berat bagasi melebih berat yang ditentukan, penumpang membayar biaya kelebihannya. Bagasi akan diteruskan ke baggage transportation. Penumpang menerima boarding pass.

Dalam menentukan tingkat detail dari activity diagram dalam posisi harus sadar. Uji level detail user mana dari diagram yang dapat bertahan dan mana yang membutuhkan jumlah detail lebih dikit. Tidak ada aturan umum yang valid,

dikarenakan tingkatan detail dasarnya bergantung pada target group dan tujuan dari model.

Sekarang, sudah dihasilkan action tambahan sebagai berikut (Gambar 33):



Gambar 33. Action Dari Aktifitas Dengan Tingkatan Detail Tinggi

MENGGUNAKAN ACTOR DARI BUSINESS USE CASE – SIAPA YANG BERTANGGUNG JAWAB PADA SETIAP ACTION?

Pada proses bisnis sangat penting untuk mengetahui siapa yang bertanggung jawab untuk setiap individual action dan siapa yang menjalankannya (Gambar 34).

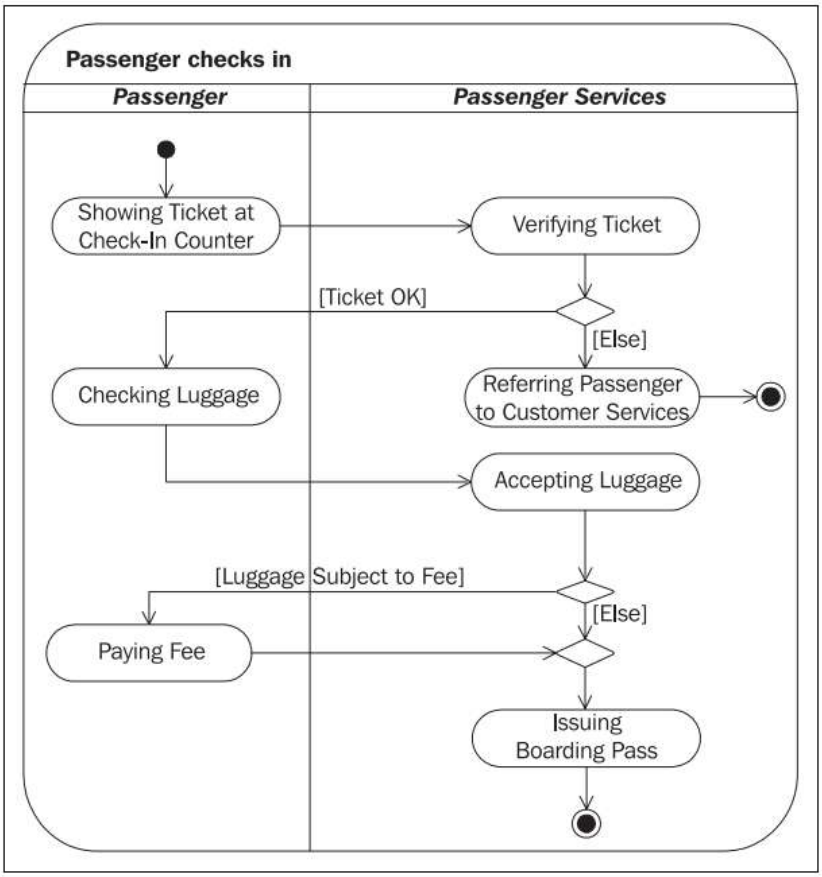
Untuk eksternal view actor yang digunakan didapat dari use case diagram. Setiap actor bertanggung jawab untuk action tertentu dan dituliskan pada partition (swimlane) sebagai pihak yang bertanggung jawab.

Aktifitas individual ditugaskan sebagai pihak yang bertanggung jawab. Pembagian dari activity diagram menjadi partisi memungkinkan kejelasan dari overview tanggung jawab. Akan tetapi partisi juga dapat dibentuk berdasarkan kriteria yang lain.

Activity Diagram seharusnya, misalnya, dapat dibagi dengan cara tertentu sehingga action manual, otomatis dan semi-otomatis dapat disusun pada satu partisi. Hal ini merupakan dasar bagus untuk konversi dari aliran menjadi Sistem IT.

MEMVERFIKASI TAMPILAN – APAKAH SEMUANYA SUDAH BENAR?

Seperti Use Case Diagram, Activity Diagram juga harus diverifikasi dalam istilah kebenaran dari isi, berkerjasama dengan knowledge carrier.



Gambar 34. Activity Diagram Dari Skenario Business Use Case "Check-in"

Checklist Memverifikasi Activity Diagram pada External View

4. Ketika membuat activity diagram dari eksternal view, selalu ingat bahwa internal prosedur dan proses bisnis tidak relevan. Batasi pertimbangan untuk deskripsi dari fungsionalitasnya pada sistem bisnis yang dimanfaatkan oleh outsider
5. Kondisi dari output yang berbeda tidak boleh tumpang tindih. Jika tidak, control flow menjadi ambigu, artinya tidak jelas dimana aliran proses setelah decision node.
6. Kondisi harus termasuk semua kemungkinan; jika tidak, control menjadi berhenti atau terjebak. Jika ragu, masukan output dengan kondisi 'else'
7. Fork dan Join harus seimbang. Jumlah aliran yang meninggalkan sebuah cabang harus sama dengan jumlah aliran yang berakhir pada Join yang sesuai.

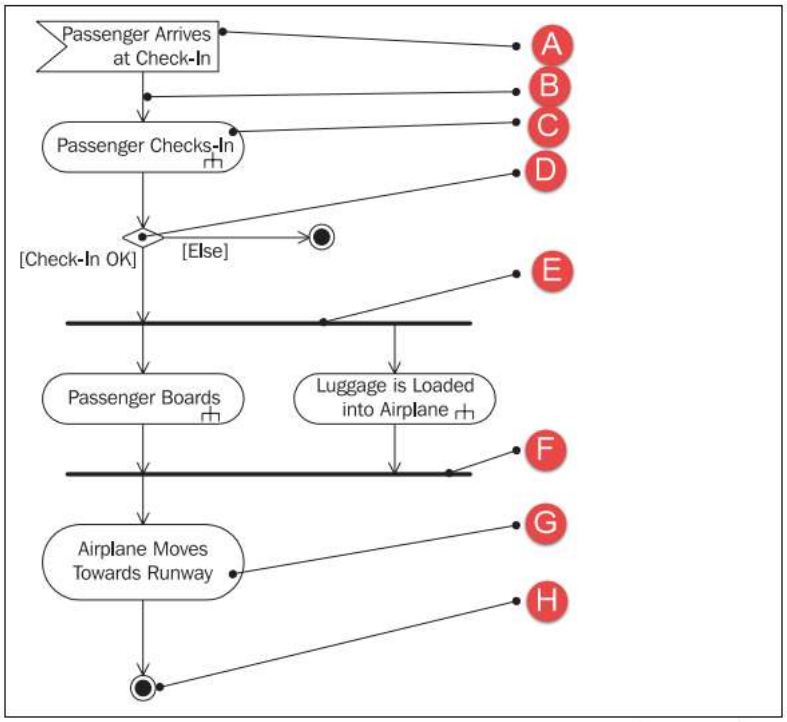
KESIMPULAN

1. Elemen yang dapat digunakan dalam membuat activity diagram:
   a. Activity
   b. Action
   c. Calling an Activity (Action)
   d. Accepting an Event (Action)
   e. Accepting a Time Even (Action)
   f. Send Signal (Action)
   g. Edge (Control Flow)
   h. Decision Node
   i. Merge Node
   j. Fork
   k. Join
   l. Initial Node
   m. Activity Final Node
   n. Flow Final Node
   o. Activity Partition
2. Membaca activity diagram dimulai dari initial node dan diakhiri pada final nodel. Membacanya mengikuti arah dari control flow baik pada satu partition atau lintas partition.
3. Dalam membuat activity diagram eksternal view dapat mengikuti langkah berikut:
   a. Mengumpulkan sumber informasi – bagaimana saya harus melakukannya?
   b. Mencari aktifitas dan action – apa yang harus dilakukan ketika actor menggunakan barang dan jasa yang ditawarkan?
   c. Menggunakan actor dari business use case – siapa yang bertanggung jawab pada setiap action?
   d. Menghubungkan action – dalam urutan bagaimana action diproses?
   e. Memperbaiki Activity – apakah activity diagram lain dapat ditambahkan?
   f. Memverfikasi tampilan – apakah semuanya sudah benar?

SOAL LATIHAN

1. Pada eksternal view, activity diagram digunakan untuk _____
   a. Menggamabrkan urutan pesan secara kronologis
   b. Menggambarkan fungsionalitas dari sistem
   c. Menggambarkan komunikasi antara obyek
   d. Menggambarkan strktur sebuah organisasi

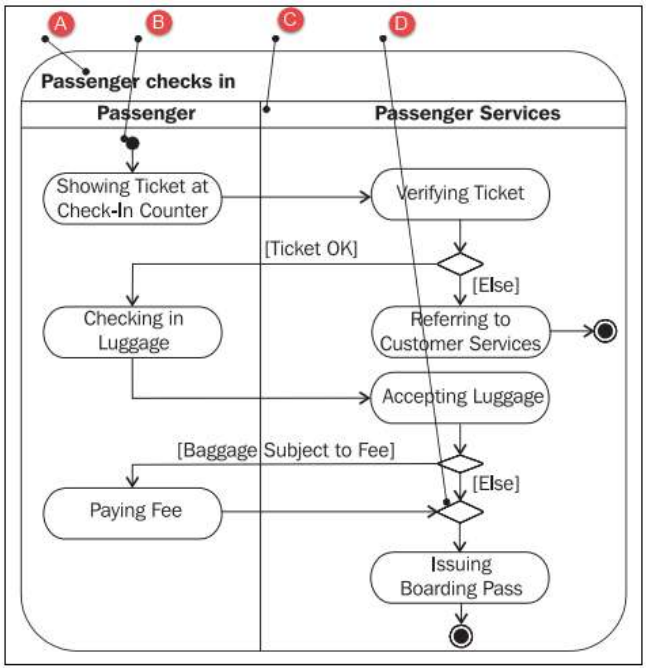
Gambar Soal No. 2 – No. 7



2. Simbol yang digunakan untuk menghubungkan komponen individual dari activity diagram adalah huruf _____
   a. B
   b. D
   c. F
   d. H
3. Decision Node ditunjukan dengan huruf _____
   a. B
   b. D
   c. F
   d. H
4. Simbol huruf E berfungsi untuk _____
   a. Menggambarkan penggabungan dari dua atau lebih kegiatan paralel
   b. Menggambarkan penyatuan aliran
   c. Menggambarkan pengiriman sinyal
   d. Menggambarkan percabangan dari dua atau lebih kegiatan paralel

5. Simbol huruf F berfungsi untuk _____
   a. Menggambarkan penggabungan dari dua atau lebih kegiatan paralel
   b. Menggambarkan penyatuan aliran
   c. Menggambarkan pengiriman sinyal
   d. Menggambarkan percabangan dari dua atau lebih kegiatan paralel
6. Simbol yang digunakan untuk mengakhiri semua aktifitas adalah huruf _____
   a. A
   b. D
   c. F
   d. H
7. Sebuah aktifitas yang dapat dipanggil dari aktifitas lain, artinya merupakan sebuah action dimana keluaran dari pemanggilan adalah activity lain, merupakan fungsi dari simbol yang ditunjuk dengan huruf _____
   a. A
   b. C
   c. E
   d. G

Gambar Soal No. 8 – No. 9



8. Simbol yang digunakan untuk menyatukan aliran adalah huruf _____
   a. A
   b. B
   c. C
   d. D
9. Nama simbol huruf C adalah Activity Partition, memiliki nama lain yaitu _____

a. Merge
b. Swimlane
c. Decision
d. Activity

10. Sebuah final node yang mengakhir satu buah flow (tidak mengakhir seluruh aktifitas) adalah simbol _____
    a. Flow Final Node
    b. Final Node
    c. Intitial Node
    d. Merge Node
11. Membaca activity dimulai dari _____
    a. Flow Final Node
    b. Final Node
    c. Intitial Node
    d. Merge Node
12. Simbol yang digunakan untuk memulai aliran pada activity diagram sesuai dengan suatu titik waktu adalah _____
    a. Accepting and Event (Action)
    b. Accepting a Time Even (Action)
    c. Calling an Activity (Action)
    d. Sending Signal (Action)
13. Salah satu langkah dari pembuatan activity diagram adalah mencari aktifitas dan action, di bawah ini yang BUKAN pertanyaan untuk membantu mencari aktifitas dan action adalah _____
    a. Apa yang dilakukan aktor individual?
    b. Even mana yang mengawali langkah kerja tertentu
    c. Dalam urutan apa sebuah action di proses?
    d. Langkah kerja apa yang dilakukan actor pada business use case?
14. Simbol panah yang digunakan untuk menghubungkan komponen individual dari activity diagram adalah Control Flow, memiliki nama lain _____
    a. Flow
    b. Start

c. End
d. Edge

15. Salah satu langkah dari pembuatan activity diagram menghubungkan action, di bawah ini yang merupakan pertanyaan untuk membantu menghubungkan action adalah _____
    a. Apa yang dilakukan aktor individual?
    b. Even mana yang mengawali langkah kerja tertentu
    c. Dalam urutan apa sebuah action di proses?
    d. Langkah kerja apa yang dilakukan actor pada business use case?
16. Pada activity diagram eksternal view menentukan actor yang bertanggung jawab untuk action tertentu dan dituliskan pada swimlane dapat menggunakan actor dari _____
    a. Use Case Diagram
    b. Class Diagram
    c. Sequence Diagra
    d. Package Diagram
17. Pada saat proses verifikasi Activity Diagram eksternal view, maksud dari internal prosedur tidak relevan adalah _____
    a. Aliran akan menjadi berhenti atau terjebak jika kondisi ada yang tidak terpenuhi
    b. Deskripsi fungsionalitas yang dipertimbangkan adalah yang dimanfaatkan oleh outsider (pihak luar)
    c. Jumlah aliran yang meninggalkan sebuah cabang harus sama dengan jumlah aliran yang berakhir pada penggabungan
    d. Aliran menjadi ambigu karena aliran tidak jelas kemana arahnya

18. Pada saat verifikasi Activity Diagram eksternal view, maksud dari kondisi dari output yang berbeda tidak boleh tumpang tindih adalah _____
    a. Aliran akan menjadi berhenti atau terjebak jika kondisi ada yang tidak terpenuhi
    b. Deskripsi fungsionalitas yang dipertimbangkan adalah yang dimanfaatkan oleh outsider (pihak luar)

c. Jumlah aliran yang meninggalkan sebuah cabang harus sama dengan jumlah aliran yang berakhir pada penggabungan
d. Aliran menjadi ambigu karena aliran tidak jelas kemana arahnya

19. Pada saat verifikasi Activity Diagram eksternal view, maksud dari kondisi harus termasuk semua kemungkinan adalah _____
    a. Aliran akan menjadi berhenti atau terjebak jika kondisi ada yang tidak terpenuhi
    b. Deskripsi fungsionalitas yang dipertimbangkan adalah yang dimanfaatkan oleh outsider (pihak luar)
    c. Jumlah aliran yang meninggalkan sebuah cabang harus sama dengan jumlah aliran yang berakhir pada penggabungan
    d. Aliran menjadi ambigu karena aliran tidak jelas kemana arahnya
20. Pada saat veriifkasi Activity Diagram eksternal view, maksud dari fork dan join harus seimbang adalah _____
    a. Aliran akan menjadi berhenti atau terjebak jika kondisi ada yang tidak terpenuhi
    b. Deskripsi fungsionalitas yang dipertimbangkan adalah yang dimanfaatkan oleh outsider (pihak luar)
    c. Jumlah aliran yang meninggalkan sebuah cabang harus sama dengan jumlah aliran yang berakhir pada penggabungan
    d. Aliran menjadi ambigu karena aliran tidak jelas kemana arahnya

UNIVERSITAS BUDI LUHUR
FAKULTAS TEKNOLOGI INFORMASI

MODUL PERKULIAHAN #4
MODELING BUSINESS SYSTEM: SEQUENCE DIAGRAM

Capaian Pembelajaran	:	Mahasiswa memahami dan dapat menggunakan sequence diagram
Sub Pokok Bahasan	:	1.21. Sequence Diagram a. Membaca Sequence Diagram b. Membuat Sequence Diagram 1.22. Sequence Diagram Tingkat Tinggi 1.23. Sequence Diagram untuk Skenario dari Business Use Case 1.24. Sequence Diagram dari Skenario Use Case Description 1.25. Kesimpulan 1.26. Soal Latihan
Daftar Pustaka	:	1. Grässle, P., Baumann, H. and Baumann, P., 2005. UML 2. 0 in Action: A Project-based Tutorial. Packt Publishing Ltd. 2. Dennis, A., Wixom, B.H. and Tegarden, D., 2009. Systems Analysis and Design UML Version 2.0. Wiley.

SEQUENCE DIAGRAM

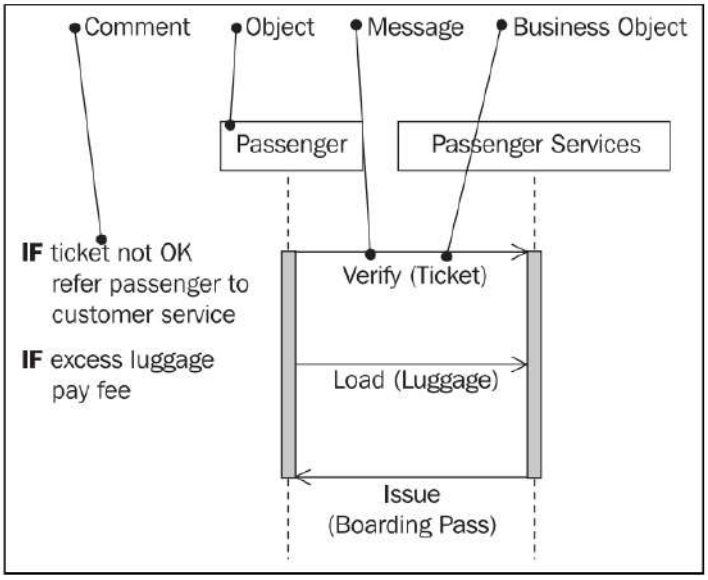
UML menyediakan dua jenis diagram untuk menggambarkan interaksi: sequence diagram dan communication diagram. Kedua diagram tersebut memvisualisasikan pertukaran informasi. Akan tetapi, penekanannya berbeda: **communication diagram** menekankan pada hubungan dari individual obyek dan topologinya; **sequence diagram** menekankan pada jalannya kronologi dari pertukaran informasi. Pada eksternal view, akan menggunakan sequence diagram dan tidak menggunakan communication diagram dikarenakan dua alasan:

1. Sequence diagram lebih mudah untuk dimengerti untuk pengembang dan pembacanya. Pada prakteknya dalam sebuah proyek sequence diagram banyak diterima karena kemudahannya.
2. Menghindari penggunaan jenis diagram yang tidak dibutuhkan dengan fakta yang sama. Lebih sedikit menjadi lebih banyak.

Jika customer atau partner bisnis menggunakan jasa yang ditawarkan, partner berkomunikasi antara satu dengan lainnya. Proses dapat digambarkan sebagai rangkaian interaksi. Interaksi ini dengan jelas diatata pada sequence diagram, sedangkan aktifitas setiap partner dan kondisi dimana interaksi terjadi dihilangkan dari diagram. Akan tetapi, dapat digambarkan dengan tambahan komentar.

Seperti activity diagram, sequence diagram dapat dibuat modelnya menjangkau beberapa use case, dan juga digunakan untuk memperbaiki business use case. Sequence diagram mengilustrasikan beberapa skenario dari business use case.

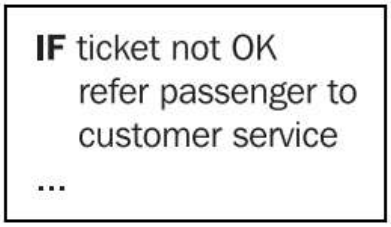
Sequence diagram dapat digunakan sebagai dasar pertukaran pesan antara sistem bisnis dan pihak luar (Gambar 35).



Gambar 35. Elemen Dari Sequence Diagram

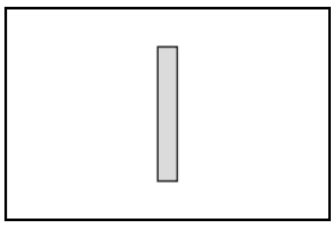
Pada Sequence Diagram, bekerja dengan elemen di bawah ini:

Comment: sequence diagram dapat dinotasikan dengan komentar (umumnya UML mengizinkan komentar pada semua diagram



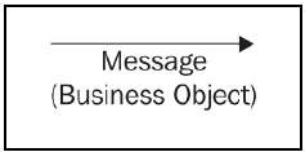
Misalnya, aktifitas dari partner atau kondisi dapat di spesifikasikan sebagai komentar.

Object: obyek yang terlibat pada interaksi diletakan pada *x*-axis. Obyek adalah pengirim dan penerima dari pesan pada sequence diagram:



Pada model sistem bisnis (eksternal view) obyek ini menggambarkan actor dari sistem bisnsi dan sistem bisnis itu sendiri.

Message dan Business Object: pesan yang dikirim dan diterima obyek ditampilkan pada *y*-axis. Pesan dimasukan pada urutan kronologi meningkat dari top ke buttom. Arah dari panah menunjukan arah pengiriman pesan:



Obyek bisnis dituliskan dalam tanda kurung. Obyek bisnis disampaikan bersamaan dengan pesan. Beberapa contoh dari obyek bisnis adalah tiket, boarding pass, dan bagasi.

1.16. MEMBACA SEQUENCE DIAGRAM

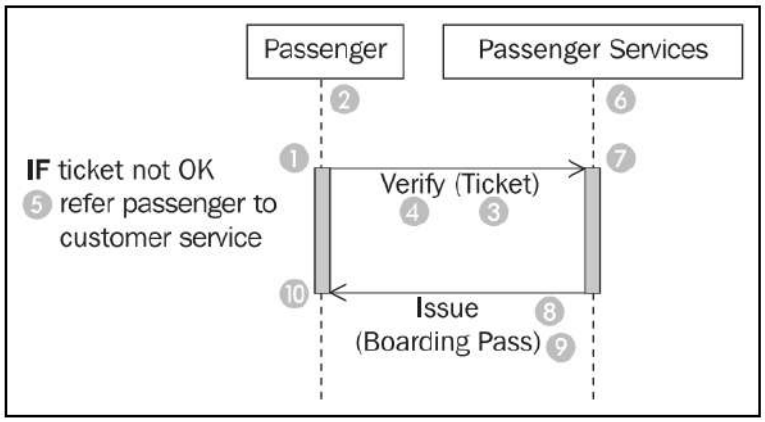
Gambar 36 menampilkan sequence diagram dengan obyek **passenger** dan **passenger services.** Keseluruhan diagram mendokumentasikan proses dari business use case **passenger check-in.**

Mulai membaca sequence diagram dari atas (1). Titik awal pada bagian kiri atas (1) dilokasikan pada garis vertical yang menggambarkan **passenger** (2) sebagai pengirim dan penerima dari pesan. Alur dimulai ketika **passenger** menyerahkan **tiket**nya (3) ke **passenger service** untuk **verifikasi** (4). Pada pesan **verify** (4); **tiket** (3) yang diserahkan ke obyek bisnis. Petunjuk arrah dari panah menunjukan bahwa **passenger** merupkakan pengirim dari pesan dan **passenger service** merupakan **receiver** (6). Tanda terima dari pesan pada passenger service mengawali aktifitas, yang ditunjukan oleh bar abu-abu vertical (7). Diagram tidak menunjukan bagaimana passenger service menangani proses, artinya tidak ditunjukan aktifitas mana yang dilaksanakan:

Hanya komentar (5) yang dapat memasukan petunjuk. Komentar dapat dimasukan pada batas kiri dari sequence diagram. Deksripsi yang tepat dari proses dapat ditemukan pada activity diagram 'passenger check in'.

Pada langkah terakhir, passenger service **issue** (8) a **boarding pass** (9) ke penumpang. Dengan hal itu, interaksi yang diilustrasikan pada sequence diagram ini

sudah selesai untuk kedua pihak. Hal ini ditunjukan oleh bagian akhir dari bar lebah abu-abu vertical (10).



Gambar 36. Sequence Diagram "Passenger Check-in"

Pada model bisnis, semua option pada sequence diagram tidak digunakan. UML menyediakan banyak kemungkinkan untuk jenis diagram seperti ini, berdasarkan pengalaman ditampilkan bahwa hal ini sudah cukup untuk berkomunikasi dengan aspek-aspek penting.

1.17. MEMBUAT SEQUENCE DIAGRAM

Ceklist di bawah ini menampilkan langkah yang diperlukan untuk membuat sequence diagram. Selanjutnya, setiap langkah akan dijelaskan lebih lanjut.

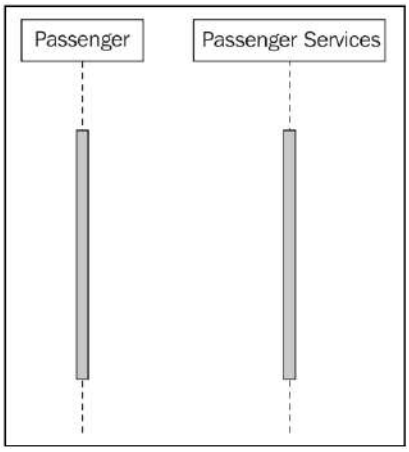
Checklist Membuat Sequence Diagram pada Eksternal View:

11. Menentukan actor dan sistem bisnis – siapa yang ambil bagian?
12. Menentukan inisiator – siapa yang memulai interaksi?
13. Menggambarkan pertukaran pesan antara actor dan sistem bisnis – pesan apa yang dipertukarkan?
14. Mengindentifikasi jalur interaksi – apa urutannya?
15. Menambahkan informasi tambahan – apa lagi yang penting?
16. Memverfikasi tampilan – apakah semuanya sudah benar?

MENENTUKAN ACTOR DAN SISTEM BISNIS – SIAPA YANG AMBIL BAGIAN

Sequence diagram mengilustrasikan interaksi antara aktor dan sistem bisnis. Secara fundamental akan ada sekumpulan pola interaksi dari use case diagram. Tergantung pada aliran yang telah digambarkan dalam sequence diagram, actor yang cocok dan sistem bisnis dapat dipilih dari kumpulan tersebut.

Pada studi kasus (Gambar 37), ditemukan pihak yang berinteraksi yaitu **passenger** dan **passenger service** dari sequence diagram Gambar 36.



Gambar 37. Membuat Sequence Diagram

MENUNJUK INISIATOR – SIAPA YANG MEMULAI INTERAKSI?

Setiap urutan dari interaksi actor yang memulai interaksi harus di identifikasi. Actor ini disebut **initiator.** Semenjak pada eksternal view dari model bisnis setiap business use case diawali oleh actor, dari hal ini dapat dipilih actor dari kumpulan beberapa actor pada use case diagram.

Pada sequence diagram passanger check-in, penumpang memulai interaksi dengan memanfaatkan layanan check-in dari passenger service.

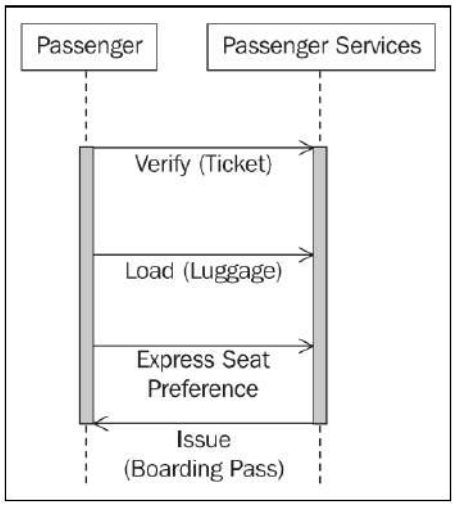
MENGGAMBARKAN PERTUKARAN PESAN ANTARA ACTOR DAN SISTEM BISNIS – PESAN APA YANG DITUKARKAN?

Setelah inisiator sudah ditentukan, progress interaksi selanjutnya harus di identifikasi. Setiap langkah komunikasi harus ditentukan informasi apa yang saling bertukar. Dengan cara ini pesan akan dapat ditentukan. Pesan adalah permintaan untuk

melakukan sesuatu secara langsung kepada pihak tertentu. Obyek bisnis yang saling bertukar dengan pesan juga harus ditentukan.

IDENTIFIKASI JALANNYA INTERAKSI – APAKAH URUTANNYA?

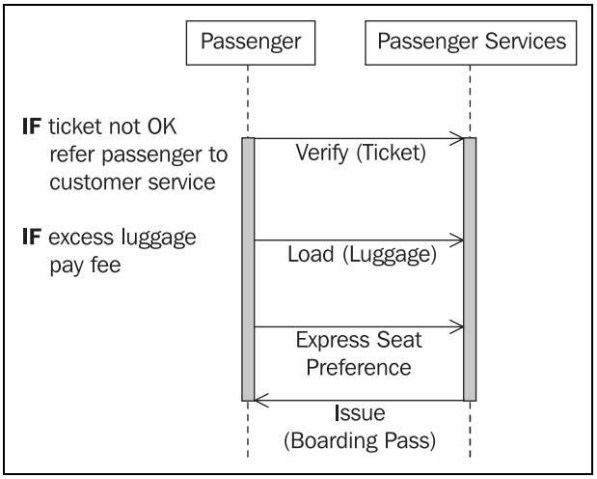
Semua pesan yang saling bertukar dalam urutan secara kronologi harus diidentifikasi. Pesan yang dimasukan sepanjang *y*-axis meningkatkan urutan kronologisnya, dari atas ke bawah. (Gambar 38)



Gambar 38. Membuat Sequence Diagram

MENEMPATKAN INFORMASI TAMBAHAN – APA LAGI YANG PENTING?

Aktifitas penting yang melibatkan actor dan sistem bisnis sersta kondisi penting dapat dimasukan kedalam diagram sebagai komentar. Komentar dimasukan pada tingkatan pesan yang cocok. Batasi hal ini pada komentar penting yang memberikan makna sehingga diagram tidak terlalu penuh dengan teks.



Gambar 39. Membuat Sequence Diagram

MEMVERFIKASI TAMPILAN – APAKAH SEMUANYA SUDAH BENAR?

Sequence diagram yang sudah selesai dapat diverifikasi menggunakan ceklist berikut:

Checklist Memverifikasi Sequence Diagram Pada Eksternal View:

1. Apakah semua sequence diagram sudah selesai dan tersedia? Harus ada sequence diagram untuk setiap use case.
2. Apakah sequence diagram sudah benar? Setiap sequence diagram berisi hanya satu obyek yang mewakili sistem bisnis, dan sebanyak-banyaknya obyek sebagai aktor yang ditugaskan pada business use case.
3. Apakah setiap actor yang ada pada list di use case diagram sudah disebutkan setidaknya pada satu sequence diagram?
4. Apakah setiap actor yang memulai business use case sudah disebutkan sebagai titik awal pada satu sequence diagram?
5. Apakah semuah komentar penting sudah dimasukan kedalam diagram? Apakah mungkin ada komentar yang terlalu banyak yang ditambahkan kedalam diagram sehingga mengurangi kejelasan diagram?

SEQUENCE DIAGRAM TINGKAT TINGGI

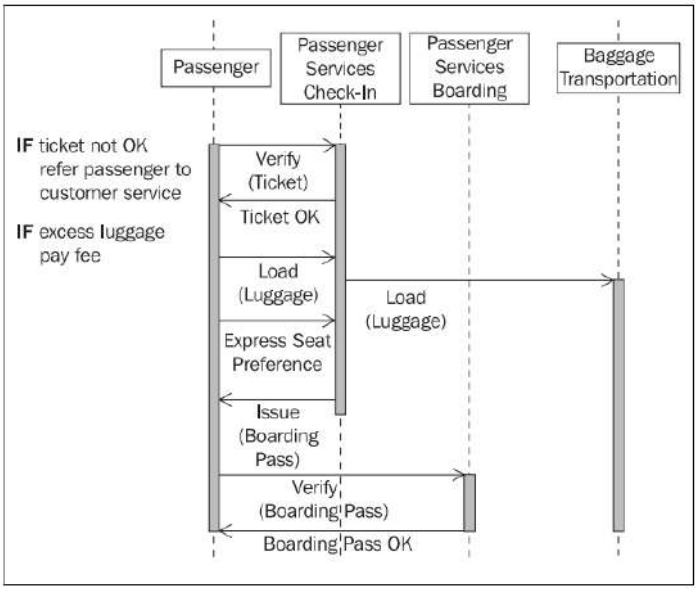
High-level sequence diagram yang menjangkau beberapa business use case dapat digunakan untuk mengilustrasikan proses bisnis secara kasar. High-leve sequence diagram memberikan overview bagus dari interaksi antara customer, partner, dan sistem bisnis. High-level sequence diagram digunakan sebagai dasar untuk mentransfer data elektronik antara sistem bisnis dan customer, partner bisnis, dan supplier.

Gambar 40 mengilustrasikan passenger service. Seluruh proses menjangkau business use case **check-in** dan **boarding.**

SEQUENCE DIAGRAM UNTUK SKENARIO DARI BUSINESS USE CASE

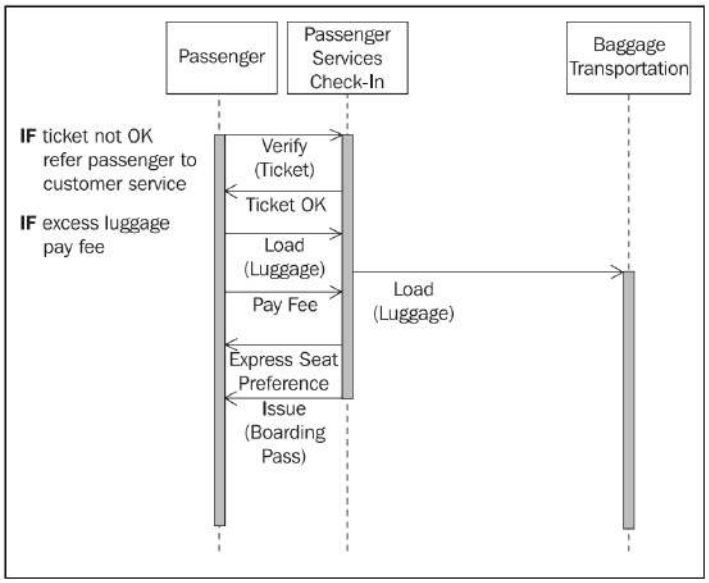
Sequence diagram membantu mendetailkan dan memberikan spesifikasi dari business use case dengan menekankan pada pesan yang saling bertukar. Beberapa skenario dari business use case dapat digambarkan pada sequence diagram. Penggambaran

dibatasi hanya pada pesan yang saling bertukar dalam setiap business use case. Secara umum, tingkatan detail dari sequence daigram ini lebih tinggi daripada sequence daigram yang menjangkau use case.



Gambar 40. Sequence Diagram "Passenger Check-In"

Gambar 41 memperlihatkan sequence diagram dari business use case **check-in**. Seqeunce diagram ini memperlihatkan skenario **passenger check-in**, yang dapat terlihat dari pihak yang berkomunikasi check-in representative tidak terlihat. Sequence diagram, seperti activity diagram, menampilkan actor jika actor melakukan business use case bersamasama atau sendiri antara satu dengan lainnya (yang tidak dapat dilihat pada use case diagram):



Gambar 41. Sequence Diagram dari Business Use Case "Passenger Check-In"

SEQUENCE DIAGRAM DARI SEKNARIO USE CASE DESCRIPTION

Use case deskripisi berisi semua informasi yang dibutuhkan untuk membangun diagram structural dan behavioral, tetapi use case deskripsi menyatakan informasi dalam bentuk kurang-formal yang dapat dimengerti oleh user. Gambar 42.

<table>
<tr><td colspan="2">Use Case Name: Passenger Check-In</td><td>ID: 1</td><td>Importance Level: Low</td></tr>
<tr><td colspan="2">Primary Actor: Passenger</td><td></td><td>Use Case Type: Detail</td></tr>
<tr><td colspan="4">Stakeholder and interest:
Passenger – ingin melakukan check-in dengan membawa tiket
Check-in Representative – menerima tiket untuk membantu proses check-in</td></tr>
<tr><td colspan="4">Brief Description:
Use Case ini menggambarkan bagaimana penumpang melakukan check-in baik itu pemesanan tempat duduk dan bagasi (jika penumpang memiliki bagasi)</td></tr>
<tr><td colspan="4">Trigger: Penumpang datang ke counter check-in
Type: Eksternal</td></tr>
<tr><td colspan="4">Relationship:
Association: Passenger, Check-in Representative
Include: Issuing Boarding Pass
Extend: -
Generalization: -</td></tr>
<tr><td colspan="4">Normal Flow of Events:</td></tr>
<tr><td>No.</td><td colspan="2">User Action</td><td>System Response</td></tr>
<tr><td>1.</td><td colspan="2">Verifikasi Tiket</td><td></td></tr>
<tr><td>2.</td><td colspan="2">Masukan Bagasi</td><td></td></tr>
<tr><td>3.</td><td colspan="2">Menentukan Kursi</td><td>Keluar Boarding Pass</td></tr>
<tr><td colspan="4">Subflow: -</td></tr>
<tr><td colspan="4">Alternate/Exceptional Flow:</td></tr>
<tr><td colspan="4">1a. Jika tiket tidak ok, pelanggan diarahkan ke customer service
2a. jika berat bagasi berlebih, membayar biaya kelebihan</td></tr>
</table>

Gambar 42. Deskripsi Use Case Passenger Check-In

Use case deskripsi memiliki tiga bagian yaitu overview information, relationship, dan flow of events.

Overview information mengindentifikasi use case dan menyediakan dasar latar belakang informasi tentang use case. *Use Case Name* harus dalam bentuk frase kata kerja-kata benda (contoh, Automated Check-in, boarding). *Use Case ID Number* menyediakan cara unik sebagai cara untuk mencari setiap use case dan dapat memungkinkan team untuk mencari keputusan desain kembali pada kebutuhan khusus. *Use Case Type* berisi antara overview, detail, essential, atau real. *Primary Actor* biasanya yang mengawali use case – orang atau sesuatu yang memulai kegiatan use case. Tujuan utama dari use case adalah untuk memenuhi sasaran dari primary actor. *Brief description* biasanya kalimat tunggal yang menggambarkan inti dari use case.

Importance level dapat digunakan untuk memprioritaskan use case. Importance level memungkinkan user secara eksplisit memprioritaskan fungsi bisnis mana yang sangat penting dan perlu menjadi bagian dari versi pertama sistem dan mana yang kurang penting sehingga dapat menunggu pada versi selanjutnya. Importance level dapat menggunakan skala fuzzy seperti high, medium, dan low.

Sebuah use case mungkin dapat memiliki *stakeholder* lebih dari satu yang memiliki kepentingan pada use case. Setiap use case akan membuat list stakeholder dengan kepentingannya pada use case (contoh check-in representative, passenger). List stakeholder termasuk primary actor (contoh check-in representative)

Setiap use case biasanya memiliki *trigger* – kejadian yang menyebabkan dimulainya use case (contoh penumpang boarding ke pesawat). Trigger dapat berupa *eksternal trigger*, seperti customer memberikan pesanan atau bunyi alarm kebakaran, atau dapat berupa *temporal trigger*, seperti pengembalian buku yang terlambant pada perpustakaan atau keperluan membayar sewa.

Relationship use case relationship menjelaskan bagaimana use case saling terhubung antara satu use case dengan user. Ada empat jenis *relationship*: association, extend, include, dan generalization. *Association relationship* mendokumentasikan komunikasi yang terjadi antara use case dengan actor yang menggunakan use case. Actor adalah penggambaran untuk peran yang diambil oleh user pada use case.

Include relationship menggambarkan inklusi wajib pada use case lain. Include relationship memungkinkan *functional decomposition* – pemecahan dari use case kompleks menajdi beberapa use case yang simple. Include relationship memungkinkan bagian dari use case untuk digunakan kembali dengan membuatnya menjadi use case terpisah.

Extend relationship menggambarkan perluasan dari fungsionalitas sebuah use case untuk menggabungkan tingkah laku optional. *Generalization relationship* memungkinkan use case mendukung *inheritance*.

Flow of event akhirnya, langkah individual dalam proses bisnis dijelaskan. Tidak jenis kategori dari *flow of event* yang dapat didokumentasikan: normal flow of event, subflow, dan alternative atau exceptional flow.

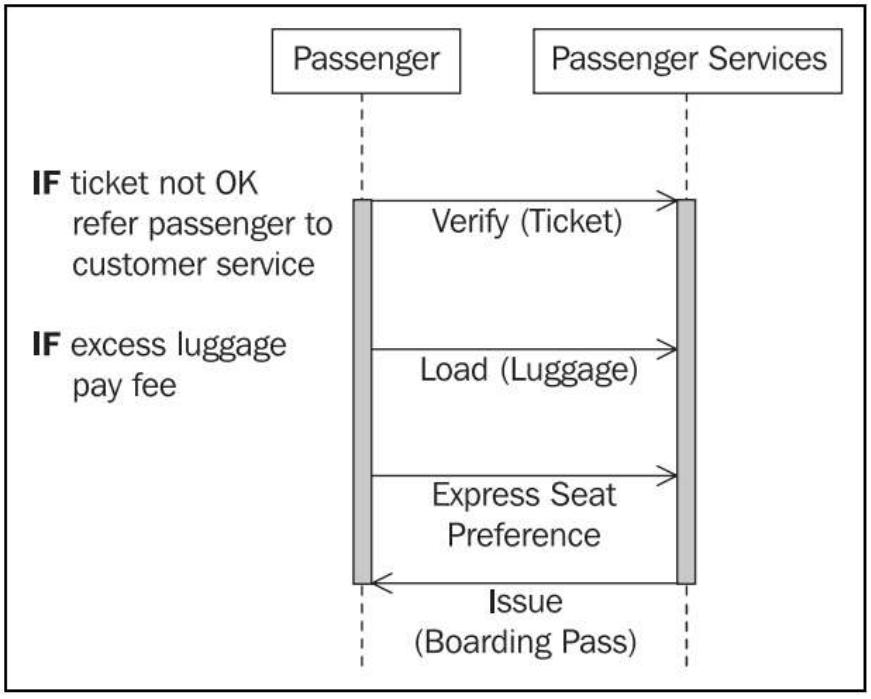
Normal flow of event termasuk langkah yang biasanya dijalankan pada use case. Langkah-langkah dibuat list dengan urutan sesuai kejadian. Pada beberapa kasus, normal flow of event sebaiknya diuraikan menjadi rangkaian *subflow* agar normal flow of event menjadi simple. Sebagai alternative, subflow dapat diganti menjadi use case terpisah yang dapat digabungkan dengan mengunakan include relationship. Akan tetapi, hal ini dapat dilakukan jika use case baru yang dibuat masuk akal.

Alternative atau exceptional flows adalah salah satu yang mungkin terjadi tetapi tidak dimasukan sebagai normal. Ini harus didokumentasikan. Seperti subflow, tujuan utama dari memisahkan alternate atau exceptional flow adalah untuk menjaga normal flow of envent sesimple mungkin. Kembali lagi, dengan subflow, mengganti alternate

atau exceptional flow, ganti alternate atau exceptional flow dengan use case berbeda yang nanti dapat dijadikan satu melalui extended relationship.

Optional Characteristic karakteristik lain dari use case yang dapat didokumentasikan oleh deskripsi use case. Termasuk level of complexity dari use case, perkiraan waktu yang diperlukan untuk melaksanakan use case, sistem yang berasosiasi dengan use case, data flow tertentu antara primary actor dan use case, dan spesifik atribut, constraint, atau operation yang berhubungan dengan use case, semua prekondisi yang harus dipenuhi untuk use case untuk dijalankan, atau jaminan apapun yang dapat dibuat berdasarkan use case.

Pada sutdi kasus use case dieskirpsi pada Gambar 42 akan mengahsilkan Sequence Diagram



Gambar 43. Sequence Diagram berdasarkan Use Case Deskripsi

Pertukaran pesan antara obyek pada sequence diagram berdasrkan Normal Flow Event pada use case deskripsi, sedangkan komentar didapat dari alternate atau exceptional flow.

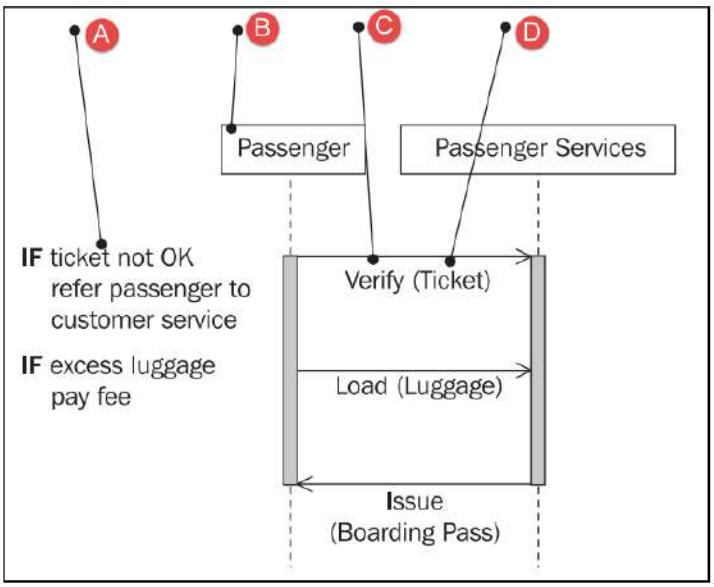
KESIMPULAN

1. UML menyediakan dua jenis diagram untuk menggambarkan interaksi: Sequence Diagram dan Communication Diagram.
2. Sequence Diagam menekankan pada jalannya kronologi dari pertukaran informasi, Communication Diagram menekankan pada hubungan dari individual obyek dan topologinya.
3. Elemen yang dapat digunakan pada pembuatan Sequence Diagram:
    a. Object
    b. Message dan Business Object
4. Sequence Diagram dibaca mulai dari atas (bagian kiri) kemudian mengikuti urutan anak panah diantaran garis vertikal secara terurut sampai dengan anak panah terakhir.
5. Dalam membuat Sequence Diagram Eksternal View perlu diperhatikan langkah-langkah berikut:
    a. Menentukan actor dan sistem bisnis – siapa yang ambil bagian?
    b. Menentukan inisiator – siapa yang memulai interaksi?
    c. Menggambarkan pertukaran pesan antara actor dan sistem bisnis – pesan apa yang dipertukarkan?
    d. Mengindentifikasi jalur interaksi – apa urutannya?
    e. Menambahkan informasi tambahan – apa lagi yang penting?
    f. Memverfikasi tampilan – apakah semuanya sudah benar?
6. High-level Sequence Diagram menjangkau beberapa business use case digunakan untuk mengilustrasikan proses bisnis secara kasar dan juga sebagai dasar untuk mentransfer data elektornik antara sistem bisnis dengan customer, partner bisnis, serta supplier
7. Sequence diagram dapat digunakan untuk membuat skenario dari business use case yang salah satu caranya dapat diperoleh dari deskripsi use case bagian normal flow of events nya.

SOAL LATIHAN

1. Dua diagram pada UML yang digunakan untuk menggambarkan interaksi adalah _____
   a. Use Case Diagram dan Sequence Diagram
   b. Communication Diagram dan Activity Diagram
   c. Sequence Diagram dan Communication Diagram
   d. Activity Diagram dan Use Case Diagram
2. Communication diagram merupakan diagram untuk menggambarkan interaksi yang menekankan pada _____
   a. Jalannya kronologi dari pertukaran informasi
   b. Hubungan dari individual obyek dan topologinya
   c. Alur proses bisnis yang terjadi
   d. Struktur obyek bisnis yang ada pada sistem bisnis
3. Sequence diagram merupakan diagram untuk menggambarkan interaksi yang menekankan pada _____
   a. Jalannya kronologi dari pertukaran informasi
   b. Hubungan dari individual obyek dan topologinya
   c. Alur proses bisnis yang terjadi
   d. Struktur obyek bisnis yang ada pada sistem bisnis

Gambar Soal No. 4 – No. 8



4. Pada sequence diagram dapat ditambahkan sebuah komentar, contohnya spesifikasi sebuah kondisi yang harus dipenuhi, simbol tersebut adalah huruf _____

   a. A
   b. B
   c. C
   d. D

5. Simbol yang digunakan sebagai pengirim dan penerima dari pesan pada sequence diagram yang biasanya untuk menggambarkan aktor dari sistem bisnis adalah huruf _____

   a. A
   b. B
   c. C
   d. D

6. Pesan yang dikirim dan diterima obyek untuk menggambarkan urutan kronologi dari atas ke bawah adalah huruf _____

   a. A
   b. B
   c. C
   d. D

7. Pesan yang dikrim dan diterima obyek untuk menggambarkan urutan kronologi dari atas ke bawah dengan diberikan tanda kurung adalah huruf _____

   a. A
   b. B
   c. C
   d. D

8. Sebutkan nama simbol dari huruf A hingga huruf D _____

   a. A: Business Object, B: Message, C: Object, D: Comment
   b. A: Comment, B: Object, C: Message, D: Business Object
   c. A: Comment, B: Message, C: Object, D: Business Object
   d. A: Business Object, B: Object, C: Message, D: Comment

9. Di bawah ini adalah langkah-langkah dalam membuat sequence diagram, KECUALI _____
    a. Menentukan aktor dari sistem bisnis
    b. Menentukan inisiator
    c. Menggambarkan pertukaran antara aktor dan sistem bisnis
    d. Verifikasi informasi tambahan
10. Actor yang memulai kegiatan pada urutan di sequence diagram, adalah _____
    a. Initiator
    b. Class worker
    c. Business object
    d. Pihak ketiga
11. Di bawah ini yang BUKAN langkah melakukan verifikasi sequence diagram yang sudah dibuat adalah _____
    a. Apakah sequence diagram sudah selesai dan tersedia?
    b. Apakah sequence daigram sudah benar
    c. Apakah business use case sudah dimasukan?
    d. Apakah semua komentar penting sudah dimasukan?
12. Maksud dari apakah semua sequence diagram sudah selesai dan tersedia? Pada langkah verifikasi sequence diagram adalah _____
    a. Semua use case diagram harus ada sequence diagramnya
    b. Setiap aktor pada list use case sudah disebutkan minimal pada satu sequence diagram
    c. Aktor yang memulai business use case sudah disebutkan sebagai titik awal
    d. Komentar yang terlalu banyak ditambahkan mengurangi kejelasan diagram.
13. Skenario dari business use case dapat digambarkan pada _____
    a. Package Diagram dan Sequence Diagram
    b. Sequence Diagram dan Activity Diagram
    c. Class diagram dan Activity Diagram
    d. Activity Diagram dan Package Diagram

14. Use Case Deskripsi merupakan _____
    a. Deskripsi aliran pada business use case
    b. Deskripsi aliran pada activity diagram
    c. Informasi yang dibutuhkan untuk membangun diagram struktural dan behavioral dalam bentuk informal
    d. Informasi yang dibutuhkan untuk membangun diagram struktural dan behavioral dalam bentuk formal
15. Pada deskripsi use case, normal flow of event adalah _____
    a. Langkah-langkah urutan kejadian pada business use case
    b. Langkah-langkah urutan kejadian pada business use case yang bukan langkah normal
    c. Uraian dari kejadian normal, agar mudah di pahami
    d. Langkah-langkah urutan kejadian pada business use case yang dapat digabungkan dengan menggunakan include relationship.

UNIVERSITAS BUDI LUHUR
FAKULTAS TEKNOLOGI INFORMASI

MODUL PERKULIAHAN #5
MODELING BUSINESS SYSTEM: PACKAGE DIAGRAM

Capaian Pembelajaran	:	Mahasiswa memahami dan dapat menggunakan package diagram
Sub Pokok Bahasan	:	1.27. Internal View 1.28. Package Diagram a. Membaca Package Diagram b. Membuat Package Diagram 1.29. Class Package Diagram dan Use Case Package Diagram a. Class Package Diagram b. Use Case Package Diagram 1.30. Kesimpulan 1.31. Soal Latihan
Daftar Pustaka	:	1. Grässle, P., Baumann, H. and Baumann, P., 2005. UML 2. 0 in Action: A Project-based Tutorial. Packt Publishing Ltd. 2. Ambler, S.W., 2004. The object primer: Agile model-driven development with UML 2.0. Cambridge University Press.

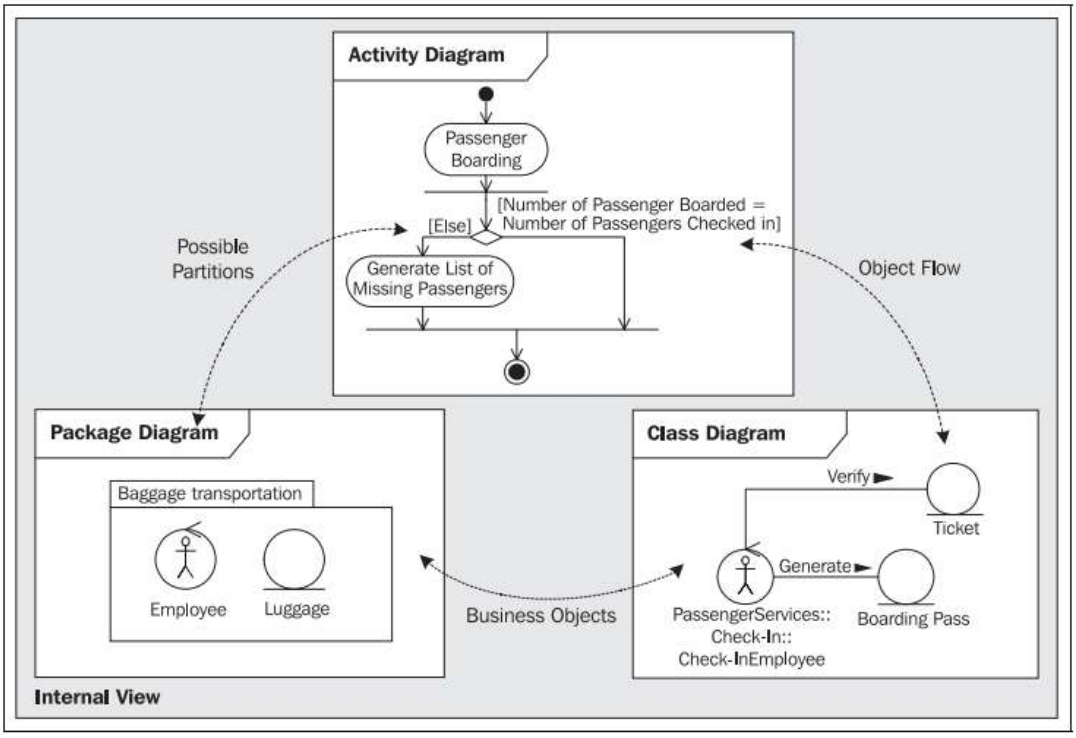
INTERNAL VIEW

Internal view menggambarkan internal **proses (processes)** dan **aktifitas (activity)**, **hubungan (relationship)**, dan **struktur (structure)** dari sistem bisnis. Sistem IT dan orang dalam sistem bisnis bertanggung jawab untuk menawarkan barang dan jasa dari sistem bisnis. Dengan ini, lingkungan sistem bisnis akan ditinggalkan kemudian masuk ke kotak hitam (black box). Mulai saat ini sangat penting untuk memperhatikan apakah proses pada sistem bisnis terjadi secara manual atau didukung oleh IT, apakah karyawan pada organisasi harus mengisi dua atau dua puluh formulir, dan apakah membutuhkan supplier.

1.18. ELEMEN DARI INTERNAL VIEW

Diagram di bawah ini mengilustrasikan internal view:

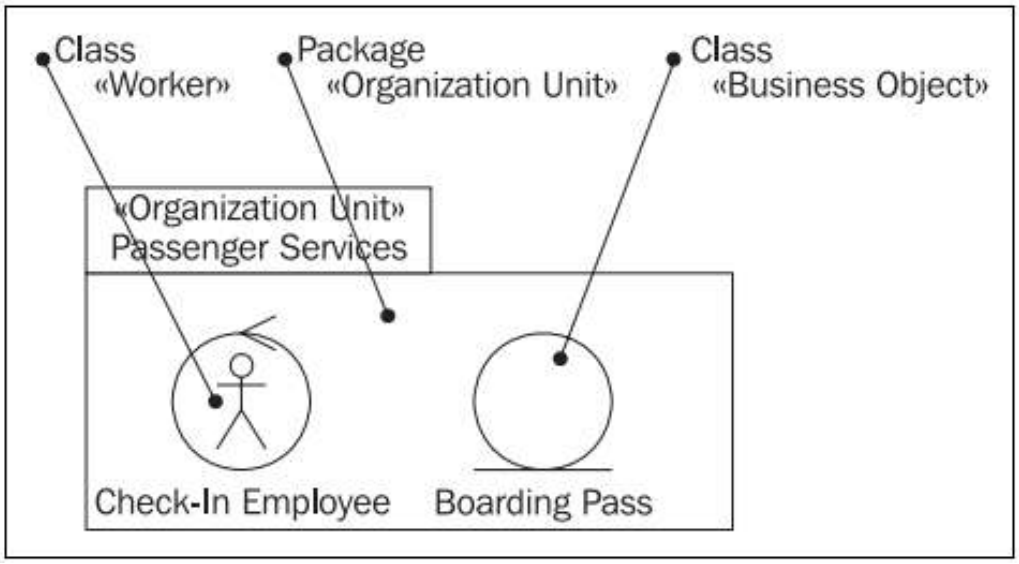
1. **Package diagram** menggambarkan unit organisasi dalam bentuk package
2. **Class diagram** menggambarkan hubungan dan koneksi antara pekerja dan obyek bisnis.
3. **Activity diagram** menggambarkan proses bisnis pada sistem bisnsi. Subyek dari deskripsinya adalah barang dan jasa yang disediakan oleh sumber daya internal sistem bisnis.



Gambar 44. Jenis Diagram dari Internal View Pada Sistem Bisnis

PACKAGE DIAGRAM

Struktur dari unit organisasi sangat penting untuk internal view pada sistem bisnis. Pada UML, unit organisasi digambarkan sebagai package, yang dapat berisi karyawan, obyek bisnis, dan unit organisasi lainnya. Pada studi kasus unit organisasinya adalah passenger service (Lihat Gambar 45)



Gambar 45. Package Diagram

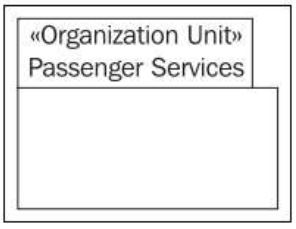
Unit organisasi dapat bertanggung jawab untuk mengeksekusi dari aktifitas proses bisnis. Unit organisasi adalah pemisahan dari pekerjaan individual dalam organisasi.

Pada UML sebuah unit organisasi mencakup pekerja, obyek bisnis, unit organisasi lainnya, termasuk hubungannya. Sebagai prinsip dasar, unit organisasi terletak dalam sistem bisnis. Unit organisasi yang terletak diluar sistem bisnis adalah actor.

Pada package diagram, berikut elemen yang dapat digunakan:

Package <<Organization Unit>>

Unit organisasi digambarkan sebagai package. Dalam kotak kecil pada bagian kiri atas dimasukan nama unit organisasi dibawah stereotype **<<Organization Unit>>:**



Isi dari unit organisasi dimasukan pada kotak utama. Sebagian besar waktu, cukup membuat daftar dari elemen paling penting (karyawan, obyek bisnis).

Class <<Worker>>

Stereotype **<<Worker>>** digunakan untuk menggambarkan peranan dari orang-orang yang mengeksekusi proses bisnis atau orang yang terlibat pada saat mengeksekusi proses bisnis:

Tidak perlu khawatir dengan 'status' dari pekerja, seperti karyawan yang digaji, karyawan lepas, atau sukarelawan, tetapi dengan peranannya, artinya pekerjaan mereka. Pekerja bertanggung jawab untuk menyediakan barang dan jasa. Pekerja berada dalam sistem bisnis. Berikut adalah karakter yang penting:

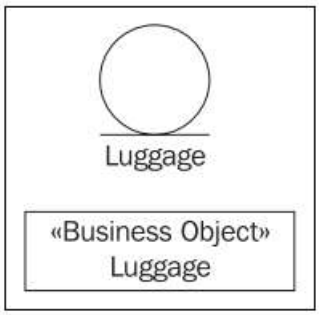
1. Pekerja adalah orang
2. Pekerja ada pada sistem bisnis
3. Pekerja dapat berkomunikasi dengan pekerja lain dan dengan actor dari luar sistem bisnis.

Pekerja dapat memeiliki simbol sendiri; dibawah simbol pekerja dimasukan peran dari pekerja. Pada simbol diperlihatkan simbol actor yang dikelilingi oleh lingkaran – ini seharusnya menandakan bahwa pekerja berada *didalam* sesuatu. Simbol pekerja dapat dihilangkan. Dengan kasus ini, simbol class digunakan dan istilah **<<worker>>** dituliskan sebagai stereotype didalam angle bracket.

<<Business Object>>

Obyek bisnis adalah **passive (pasif)**, artinya obyek bisnis tidak mengawali interaksi. Obyek bisnis dapat terlibat pada beberapa business use case berbeda dan bertahan lebih lama dibandingkan interaksi individual. Hal ini membuat obyek bisnis membentuk

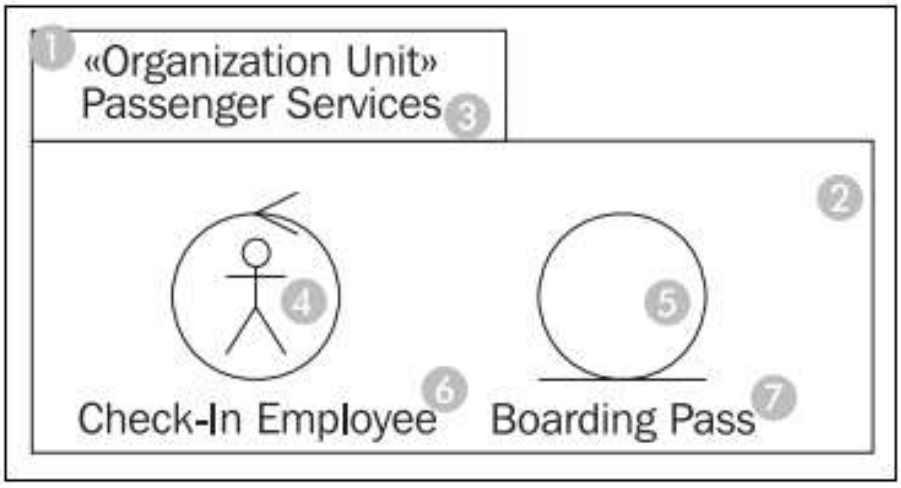
link koneksi antara business use case atau pekerja yang terliat pada beberapa use case:



Pekerja menangani (utilize, control, manipulate, produce, dll) obyek bisnis. Pada studi kasus obyek bisnis adalah, sebagai contoh, tiket, bagasi, atau boarding pass. Obyek bisnis diilustrasikan dengan simbolnya sendiri; deksripsi dari obyek bisnis ditulis dibawah simbol obyek bisnis.

Simbol obyek bisnsi dapat dihilangkan. Dalam kasus ini, simbol class diguanakn dan istilah **<<Business Object>>** ditulis sebagai stereotype didala angel bracket.

1.19. MEMBACA PACKAGE DIAGRAM



Gambar 46. Package Diagram

Melalui stereotype **<<Organizational Unit>>** (1) dapat dilihat package (2) mewakili sebuah unit organisasi. Nama dari unit organisasi adalah **passenger services** (3). Dalam unit organisasi ini ditemukan **check-in employee** (4) dan obyek bisnis: **boarding pass** (5). Simbol grafik (4) dibagian kiri menggambarkan **pekerja**

(worker); labelnya (6) di bawah simbol grafik menunjukan **peran pekerja (worker's role)** dalam organisasi. Simbol grafik (5) pada bagian kanan menggambarkan **obyek bisnis (business obyek)**; labelnya (7) di bawah simbol grafik menunjukan **jenis (type)** dari bisnis obyek yang sedang terlibat.

Hanya ada *satu* simbol untuk check-in employee. Bukan berarti hanya ada satu check-in employee, melainkan simbolnya menggambarkan sebuah peran yang dipenuhi oleh berapapun jumlahnya dari check-in employee. Tentunya, ada peran pekerja lain pada passanger service (manager, asisten, dll). Akan tetapi, hal ini tidak relevan untuk mengilustrasikan prosesnya, sehingga tidak perlu dimasukan pada package diagram (lihat Gambar 46)

1.20. MEMBUAT PACKAGE DIAGRAM

Checklist di bawah ini menampilkan langkah yang diperlukan untuk membuat package diagram. Setiap langkah akan dijelaskan lebih rinci pada bagian selanjutnya:

Checklist Membuat Package Diagram pada Internal View:

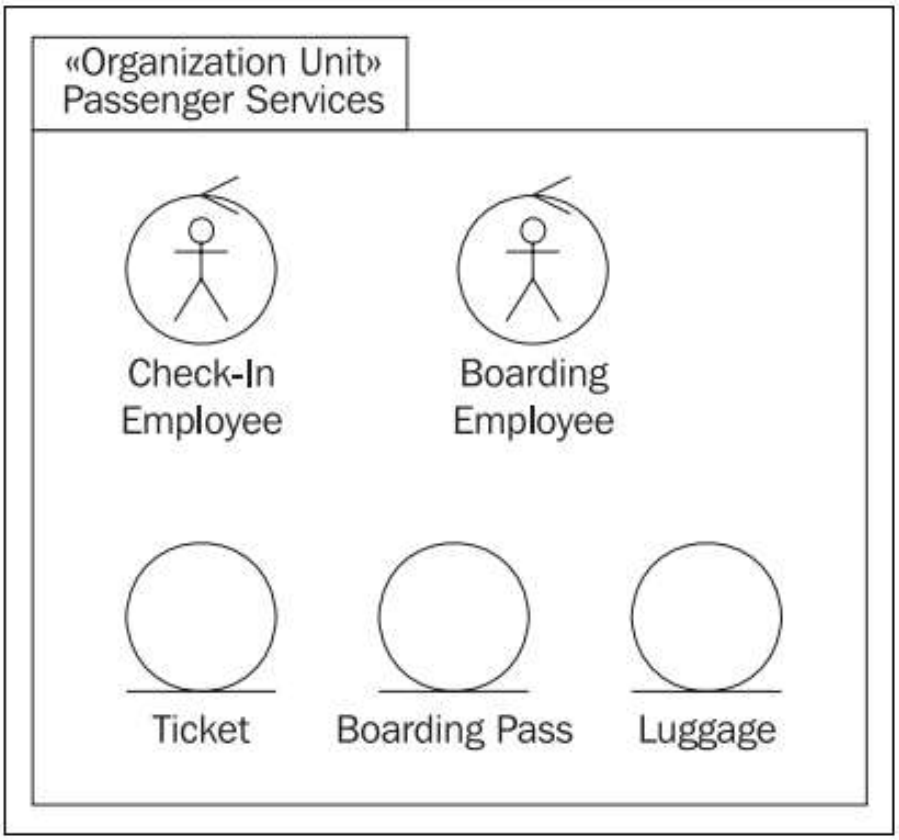
6. Mengembangkan package diagram awal dari sistem bisnis – pekerja dan obyek bisnis apa yang melengkapi sistem bisnis?
7. Menemukan unit organisasi tambahan – siapa lagi yang ada disana?
8. Menugaskan pekerja dan obyek bisnis ke unit organisasi – siapa yang harus ada dimana?
9. Menemukan unit organisasi, pekerja, atau obyek bisnis tambahan – apalagi yang ada disana?
10. Memverifikasi tampilan – apakah semuanya sudah benar?

MENGEMBANGKAN PACKAGE DIAGRAM AWAL DARI SISTEM BISNIS – PEKERJA DAN OBYEK BISNIS APA YANG MELENGKAPI SISTEM BISNIS?

Pertama, sistem bisnis melengkapi unit organisasi yang harus digambarkan. Pada studi kasus, Pessenger Service (lihat Gambar 47). Permulaan, mencari peran pekerja yang relevan (jobs) dan obyek bisnis untuk unit organisasi. Deskripsi pekerjaan yang sudah ada dan chart organisasi dapat membantu hal ini.

MENEMUKAN UNIT ORGANISASI TAMBAHAN – SIAPA LAGI YANG ADA DISANA?

Sangat mungkin sekali, unit organisasi dapat dibagi menjadi unit organissai lebih jauh lagi (divisi, team, group). Chart organisasi dan deskripsi pekerjaan dapat digunakan sebagai dasar pemilihan unit organisasi yang relevan untuk model. Pekeraan dan unit organisasi yang relevan adalah yang secara langsung terintegrasi kedalam pemrosesan barang dan jasa.



Gambar 47. Membuat Package Diagram

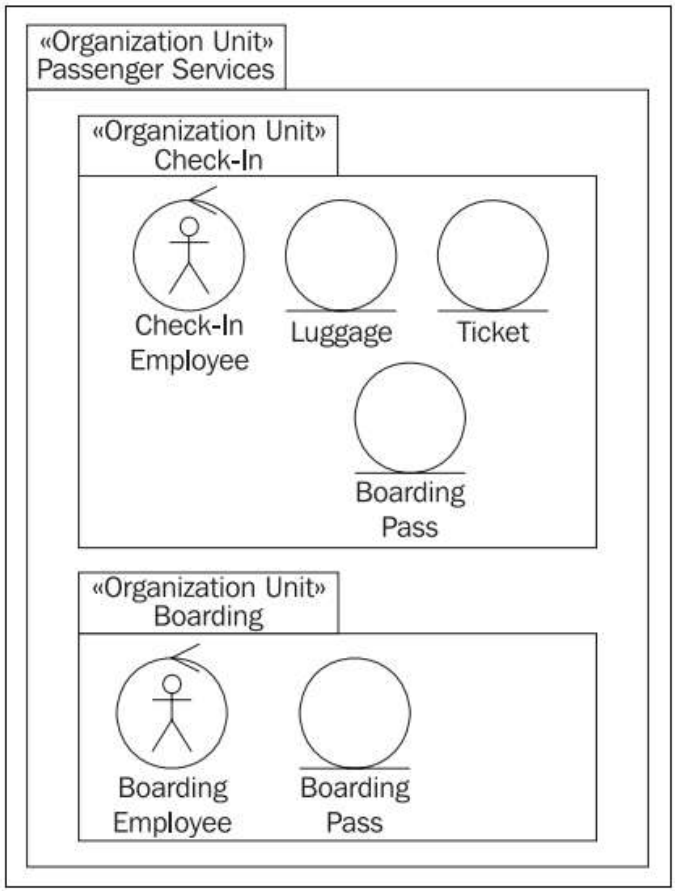
Pada studi kasus, Passenger Service dibagi menjadi unit organisasi yang lebih jauh: check-in and boarding. Divisi yang lebih lanjut sangat bijak jika divisi tersebut penting untuk ilustrasi pada proses bisnis. Contohnya, dengan pertimbangan kelompok sekretaris tidak penting dalam proses bisnis.

MENUGASKAN PEKERJA DAN OBYEK BISNIS KE UNIT ORGANISASI – SIAPA YANG HARUS ADA DIMANA?

Employee dan obyek bisnis harus ditugaskan pada unit organisasi tambahan. Seperti pada Gambar 48 bahwa obyek bisnis sudah terbagi. Dikarenkan hal ini membuat struktur dan penugasan menjadi mudah terlihat dan jelas.

MENEMUKAN UNIT ORGANISASI, PEKERJA, ATAU OBYEK BISNIS TAMBAHAN – APA LAGI YANG ADA DISANA?

Package Diagram yang menggambarkan unit organisasi tidak harus tertukar dengan chart organisasi. Faktanya, chart organisasi terhubung dengan package diagram seperti yang terlihat pada Gambar 48. Akan tetapi, package diagram berisi obyek bisnis sebagai tambahan dari karyawan. Dari chart organisasi diperoleh sturuktur hirarki dan peran dari bermacam-macam pekerja, dan dapat digunakan sebagai dasr dari membuat package diagram.



Gambar 48. Unit Organisasi "Passenger Service"

MEMVERFIKASI TAMPILAN – APAKAH SEMUANYA SUDAH BENAR?

Dalam menyelesaikan package diagram dapat dilihat checklist di bawah ini:

Checklist Verifikasi Package Diagram Pada Internal View:

1. Apakah semua pekerja dan unit organisasi terlibat pada pemrosesan barang dan jasa yang disediakan oleh sistem bisnis sudah dimasukan kedalam package diagram?
2. Apakah sudah tidak ada lagi pekerja dan unit organisasi yang tidak terkait pada pemrosesan barang dan jasa dari sistem bisnis sudah dimasukan dalam package diagram?
3. Apakah semua obyek bisnis yang dibutuhkan untuk menyediakan dan memproses barang dan jasa sudah dimasukan dalam package diagram?
4. Apakah sudah tidak ada lagi obyek bisnis yang tidak terhubung pada pemrosesan barang dan jasa dari sistem bisnis dimasukan dalam package daigram?

CLASS PACKAGE DIAGRAM DAN USE CASE PACKAGE DIAGRAM

Package diagram adalah diagram UML yang terdiri dari package dan dependecies diantaranya. Package adalah konstruk UML yang dapat digunakan untuk mengatur model elemen, seperti use case atau class, menjadi satu kelompok. Package digambarkan sebagai folder dan dapat diaplikasikan pada UML diagram manapun. Package diagram dibuat untuk:

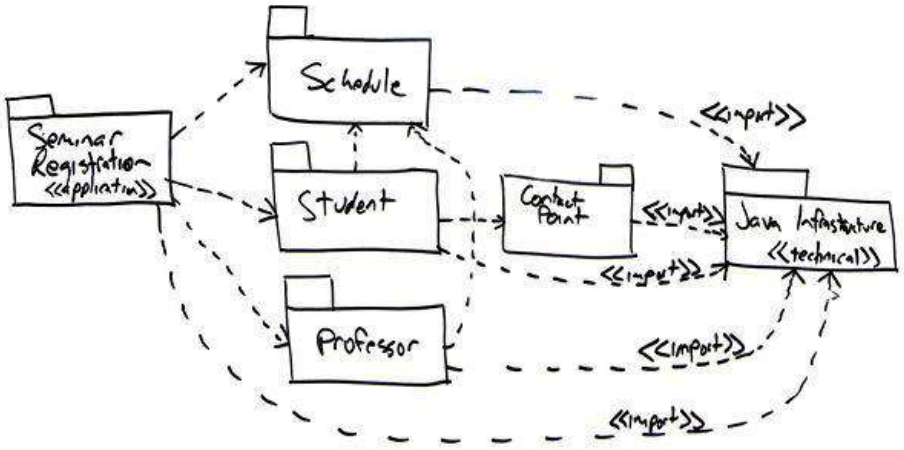
1. Menggambarkan high-level overview dari kebutuhan (penggambaran dari kumpulan Use Case Diagram UML)
2. Menggambarkan high-level overview dari arsitektur/desain (penggambaran dari kumpulan Class Diagram UML)
3. Modularisasi secara logika dari diagram kompleks
4. Mengorganisasikan source code java.

1.21. CLASS PACKAGE DIAGRAM

Dalam membuat class package diagram terdapat aturan yang dapat diikut:

1. Tempatkan class dari framework pada package yang sama

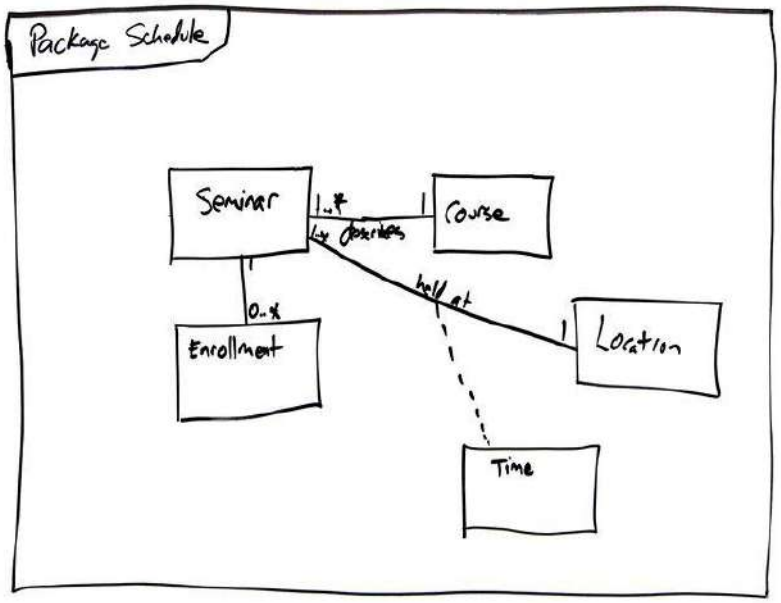
2. Class yang berada dalam satu hirarki inheritance biasanya ditempatkan pada package yang sama
3. Class yang berhubungan satu dengan lainnya melalui agregation atau composition sering ditempatkan pada package yang sama
4. Class yang sering berkolaborasi, informasi yang diberikan dari Sequence Diagram UML dan Collaboration Diagram UML, sering ditempatkan pada package yang sama.



Gambar 49. Package Diagram

Gambar 49 merupakan package diagram seminar registrasi terdapat *application* stereotype, menunjukan bahwa package berisi class user interface (UI) dan aplikasi spesifik yaitu business class untuk pendaftaran mahasiswa pada seminar. Stereotype *technical* yang ada pada package Java Infrastructure menunjukan bahwa package berisi class technical, mungkin framework user interface. Stereotype *import* menunjukan bahwa package Java Infrastructure diimport kedalam package lain.

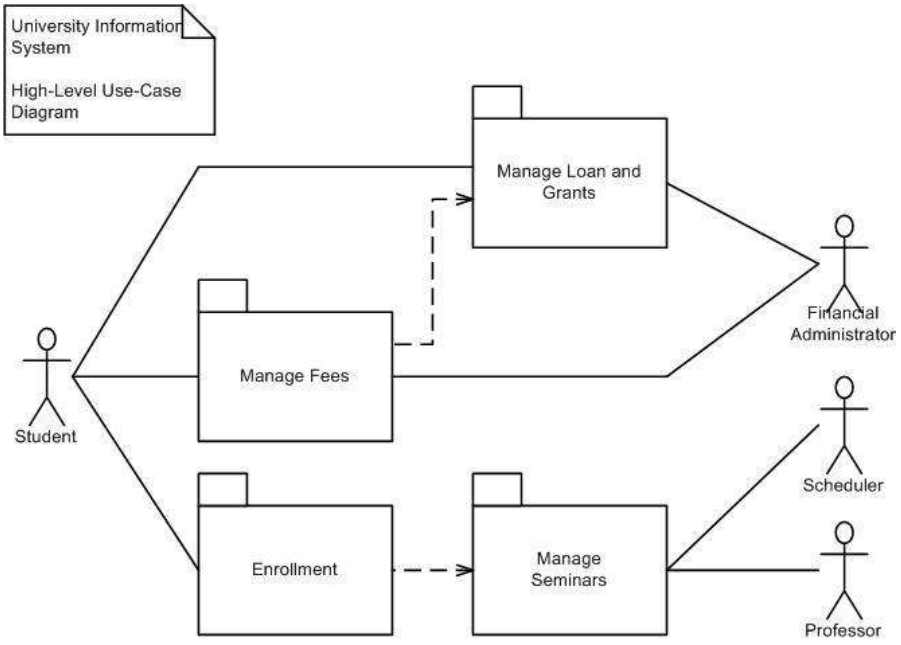
Setiap packge akan menuju ke diagram yang lebih detail, bisa saja package diagram lain untuk sistem yang kompleks atau bisa juga menuju ke Class Diagram UML. Gambar 50 menggambarkan sebuah frame yang digunakan untuk menggambarkan isi dari *Package Schedule*, pada kasus ini high-level class diagram konseptual. Frame dapat digunakan untuk menampilkan detail isi dari jenis model UML apapun, seperti package, komponen, class, atau operasi.



Gambar 50. Isi Dari Package Schedule

1.22. USE CASE PACKAGE DIAGRAM

Gambar 51 menggambarkan UML Use case diagram yang telah diorganisasikan kedalam package (aktor masih ditunjukan pada diagram, walaupun bisa dimasukan kedalam package juga). Apakah ini merupakan use case diagram atau UML Package diagram? Hal yang penting adalah diagramnya bagaimanapun menambahkan nilai – bagaimana mengkategorikan diagram – yang menjadi konsekuensi kecil.



Gambar 51. Use Case Package Diagram

Dua aturan yang dapat diikuti ketika membuat Use Case Package Diagram:

1. Include dan extend pada use case dimasukan kedalam satu package sebagai base/parent use case. Hal ini bekerja dengan baik karena biasanya use case diperkenalkan dengan mengamil logika dari base/parent use case untuk memulai.
2. Menganalisa mana use case yang melibatkan actor utama. Setiap aktor akan berinteraksi dengan sistem untuk menghasilkan tujuan utama, contohnya seperti pada Gambar 51, student berinteraksi dengan universitas untuk masuk ke universitas, untuk mengatur jadwal, bayar tagihan, dan mengatur pinjaman serta pendanaan.

KESIMPULAN

1. Internal view menggambarkan internal proses dan aktifitas, hubungan, dan struktur dari sistem bisnis.
2. Elemen dari internal view:
   a. Package Diagram
   b. Class Diagram
   c. Activity Diagram
3. Elemen pada package diagram:
   a. Package <<Organization Unit>>
   b. Class <<Worker>>
   c. <<Business Object>>
4. Membaca package diagram dimulai dari unit organisasinya kemudian menyebutkan bagian dari unit organisasi tersebut baik itu class <<worker>> atau <<Business Object>>
5. Dalam membuat Package Diagram perlu diperhatikan langkah-langkah berikut:
   a. Mengembangkan package diagram awal dari sistem bisnis – pekerja dan obyek bisnis apa yang melengkapi sistem bisnis?
   b. Menemukan unit organisasi tambahan – siapa lagi yang ada disana?
   c. Menugaskan pekerja dan obyek bisnis ke unit organisasi – siapa yang harus ada dimana?

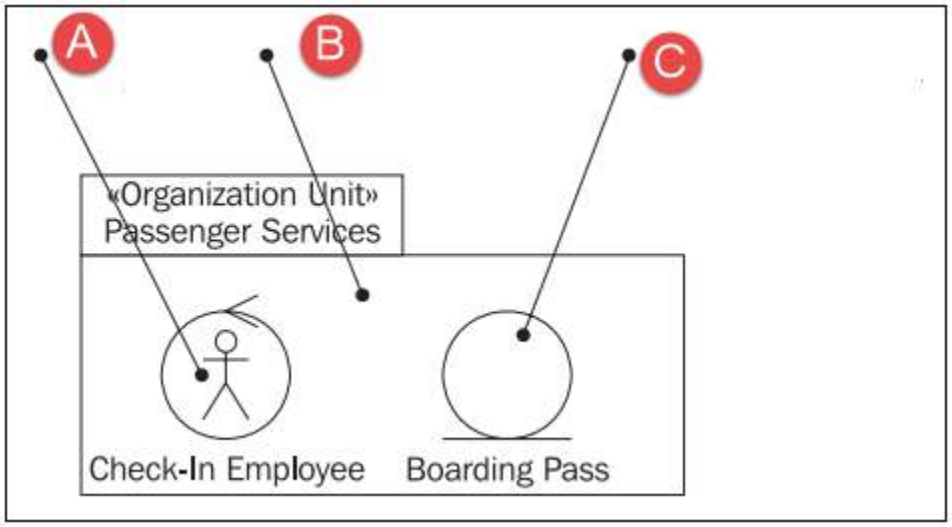
d. Menemukan unit organisasi, pekerja, atau obyek bisnis tambahan – apalagi yang ada disana?
e. Memverifikasi tampilan – apakah semuanya sudah benar?

6. Package diagram juga dapat digunakan untuk membuat Class Package Diagram yang digunakan untuk megelompokan Class dan Use Case Package Diagram yang digunakan untuk pengelompokan Use Case

SOAL LATIHAN

1. Di bawah ini merupakan elemen dari internal view, KECUALI _____
   a. Package Diagram
   b. Acitivty Diagram
   c. Class Diagram
   d. Sequence Diagram
2. Elemen internal view yang menggambarkan unit organisasi dalam bentuk folder adalah _____
   a. Package Diagram
   b. Acitivty Diagram
   c. Class Diagram
   d. Sequence Diagram
3. Elemen internal view yang menggambarkan hubungan dan koneksi antara pekerja dan obyek bisnis adalah _____
   a. Package Diagram
   b. Acitivty Diagram
   c. Class Diagram
   d. Sequence Diagram
4. Elemen internal view yang menggambarkan proses bisnis pada sistem binsis adalah _____
   a. Package Diagram
   b. Acitivty Diagram
   c. Class Diagram
   d. Sequence Diagram

Gambar Soal No. 5 – No. 7



5. Nama simbol yang ditunjuk huruf A sampai C adalah _____
   a. A: Class <<workder>>, B: Class <<Business Object>>, C: Package <<organization unit>>
   b. A: Class <<worker>>, B: Package <<organization unit>>, C: Class <<business object>>
   c. A: Package <<organization unit>>, B: class <<worker>>, C; class <<business object>>
   d. A: Package <<organization unit>>, B: class <<business object>>, C: class <<worker>>
6. Simbol yang digunakan untuk menggambarkan peranan dari orang-orang yang mengeksekusi proses bisnis adalah huruf _____
   a. A
   b. B
   c. C
   d. Tidak ada pada gambar
7. Simbol yang digunakan untuk menggambarkan dokumen, barang yang terlibat pada sistem bisnis adalah _____
   a. A
   b. B
   c. C
   d. Tidak ada pada gambar

8. Pada sistem bisnis, class <<worker>> atau pekerja pada sistem memiliki karakter di bawah ini, KECUALI _____
   a. Pekerja adalah orang
   b. Pekerja ada pada sistem bisnis
   c. Pekerja adalah barang tidak berwujud
   d. Pekerja dapat berkomunikasi dengan pekerja lain
9. Membaca package diagram dimulai dari _____
   a. Class <<worker>>
   b. Class <<business obyek>>
   c. Package <<actor>>
   d. Package <<organization unit>>
10. Di bawah ini adalah langkah-langkah dalam membuat package diagram, KECUALI _____
   a. Menemukan siapa yang ada pada sistem bisnis
   b. Menugaskan class <<worker>> dan <<business object>> ke unit organisasi
   c. Mengembangkan package diagram awal dari sistem bisnis
   d. Menemukan unit organisasi, pekerja, atau obyek bisnis tambahan
11. Di bawah ini merupakan ketentuan pada class package diagram, KECUALI _____
   a. Class masuk ke package yang sama jika berasal dari satu framework
   b. Class masuk ke package yang sama jika berada dalam satu hirarki inheritance
   c. Class masuk ke package yang sama jika memiliki asosiasi include dan extend
   d. Class masuk ke package yang sama jika memiliki hubungan agregation atau composition
12. Di bawah ini yang merupakan ketentuan pada Use Case Package Diagram adalah _____
   a. Class masuk ke package yang sama jika berasal dari satu framework
   b. Class masuk ke package yang sama jika berada dalam satu hirarki inheritance

c. Class masuk ke package yang sama jika memiliki asosiasi include dan extend

d. Class masuk ke package yang sama jika memiliki hubungan agregation atau composition

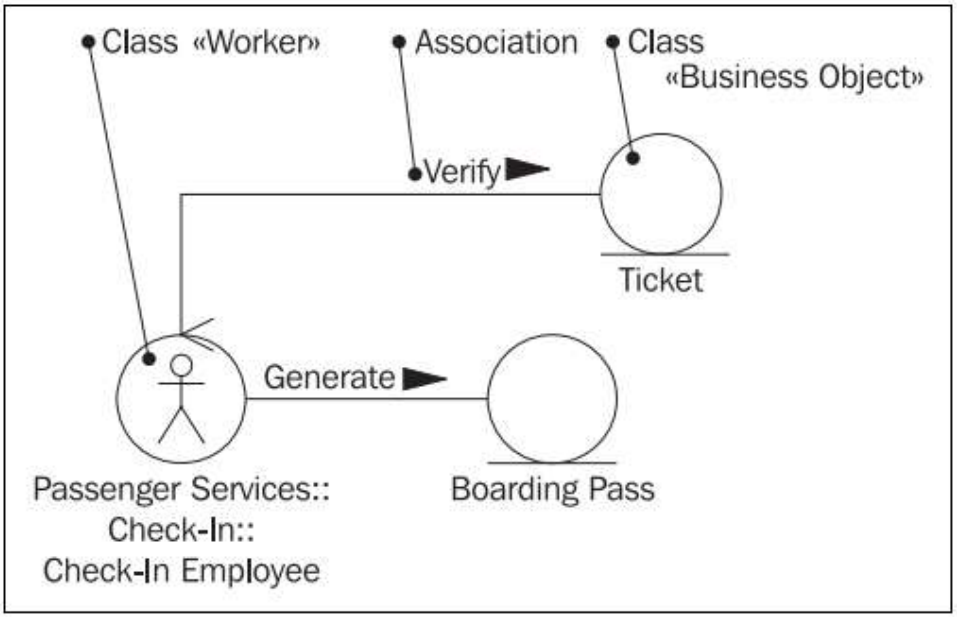
UNIVERSITAS BUDI LUHUR
FAKULTAS TEKNOLOGI INFORMASI

MODUL PERKULIAHAN #6
MODELING BUSINESS SYSTEM: CLASS DIAGRAM

Capaian Pembelajaran	:	Mahasiswa dapat memahami dan menggambarkan class diagram
Sub Pokok Bahasan	:	1.32. Class Diagram a. Membaca Class Diagram b. Membuat Class Diagram 1.33. Kesimpulan 1.34. Soal Latihan
Daftar Pustaka	:	Grässle, P., Baumann, H. and Baumann, P., 2005. UML 2. 0 in Action: A Project-based Tutorial. Packt Publishing Ltd.

CLASS DIAGRAM

Class diagram dapat digunakan untuk mengilustrasikan bagian structural dari sistem bisnis, artinya hubungan antara individu karyawan, obyek bisnis, dan pihak luar. Secara signifikan pada tingkatan model bisnis class diagram dibuat dengan sederhana dan hanya menggunakan sedikit elemen. Hal ini masih berlaku: kurang sering berarti lebih! Ketika bermacam-macam pilihan pada class diagram digunakan, diagramnya sudah tidak lagi mudah dibaca. Pada tingkatan sistem bisnis harus mempunyai asumsi bahwa pihak yang terlibat memiliki sedikit atau bahkan tidak sama sekali keahlian IT dan tidak mengetahui apapun tentang terminology class dan class diagram. Keuntungan yang diharapkan dari UML, yaitu mempermudah komunikasi antara beberapa pihak terlibat, seharusnya tidak tergangu secara signifikan.

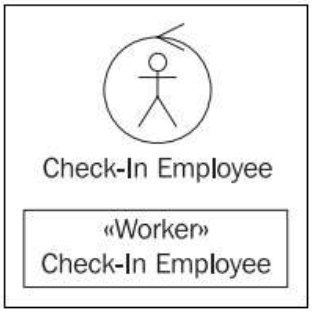


Gambar 52. Class Diagram

Pada class diagram berikut elemen yang digunakan:

Class <<worker>>

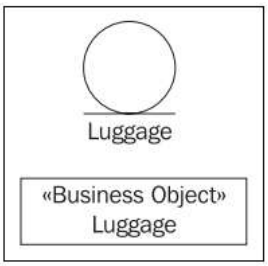
Sebelumnya sudah dijelaskan pada Package Diagram. **Class <<worker>>** pada class diagram sama persis dengan class <<worker>> pada package diagram; dan penggambarannya juga sama:



Seperti yang terlihat pada Gambar 52, dapat dinyatakan seluruh bagian nama dari class untuk mengilustrasikan keanggotaan dari package. Pada contoh, seluruh bagian menandakan bahwa class **check-in employee** merupakan bagian dari package **check-in**, dan package **check-in** merupakan bagian dari package **passenger service**, yang dibagi menggunakan double colon. Class **worker** digunakan pada class diagram untuk mengilustrasikan hubungan dengan karyawan lain, actor, dan obyek bisnis.

Class <<Business Object>>

Pada pembahasan mengenai *Package Diagram* sudah digambarkan class **business object**. Class business object pada package diagram merupakan hal sama yang akan digunakan pada class diagram.



Seperti pada package diagram, class business object dapat digambarkan degnan simbol obyek bisnis atau simbol class.

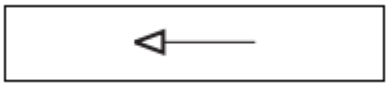
Association

Asosiasi menggambarkan sebuah hubungan yang memiliki definisi arti tepat. Asosiasi dapat diberikan label dengan nama asosiasi. Jika ingin memberikan arah pada nama asosiasi, dapat menambahkan segitiga mengarah ke nama yang seharusnya dibaca:

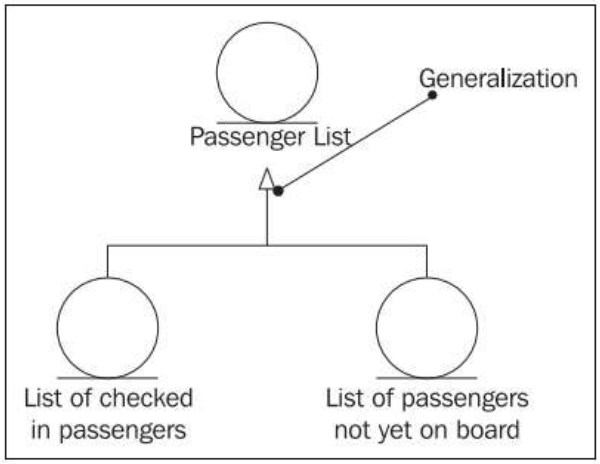
Tambahan dari elemen di atas, maka dapat disebutkan sebuah generalization. Akan tetapi, penggunaan generalization tidak wajib.

Generalization

Generalization adalah hubungan spesifik antara elemen umum dan khusus. Generalization dan specialization membantu dengan hirarki struktur. Jika beberapa obyek bisnis yang seharusnya digabungkan menjadi item yang komprehensif, generalization adalah alat yang tepat. (Lihat Gambar 53)



Akan tetapi, untuk pekerja direkomendasikan menyusun package diagram:



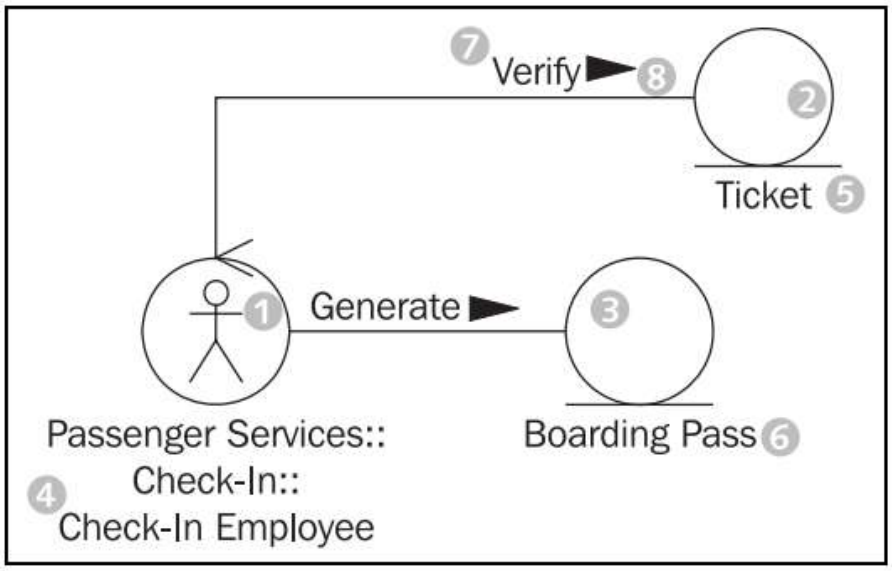
Gambar 53. Class Diagram Dengan Generalization

1.23. MEMBACA CLASS DIAGRAM

Gambar 54 menampilkan bagian kecil dari class diagram berdasarkan studi kasus. Pada class diagram berisi beberapa class **check-in employee** (1), **ticket** (2), dan **boarding pass** (3), beserta dengan asosiasinya:

Dapat dilihat berdasarkan label (4) dari simbol pekerja (1), bahwa check-in employee merupakan milik dari unit organisasi check-in, yang merupakan bagian dari passenger service.

Label yang ditulis pada bagian depan label untuk pekerja, dipisahkan dengan double colon, menunjukan unit organisasi yang menaungi pekerja. Dapat dilihat bahwa passenger service dan check-in merupakan unit organisasi dari package diagram.



Gambar 54. Class Diagram

Label dari simbol obyek bisnis (5 dan 6) memperlihatkan bawa terdapat dua obyek bisnis: **ticket** (5) dan **boarding pass** (6).

Asosiasi antara class harus dibaca dengan format berikut:

- Check-in Empoloyee (4) memverifikasi (7) sebuah ticket (5)

Segitga kecil (8) disebelah dari nama asosiasi (7) menunjukan **arah (direction)** yang merupakan nama dari asosiasi yang harus dibaca. Semua asosiasi dalam class diagram dana dibaca dengan cara tersebut.

Tidak akan menggunakan multiplicity pada class diagram untuk model sistem bisnis, artinya, untuk manfaat kejelasan, tidak akan membuat pernyataan tentang jumlah obyek pada class yang terlibat pada asosiasi.

Saat ini belum penting jika check-in employee mengeluarkan satu atau beberapa boarding pass. Kuantitas yang penting dapat dimasukan ke komentar.

1.24. MEMBUAT CLASS DIAGRAM

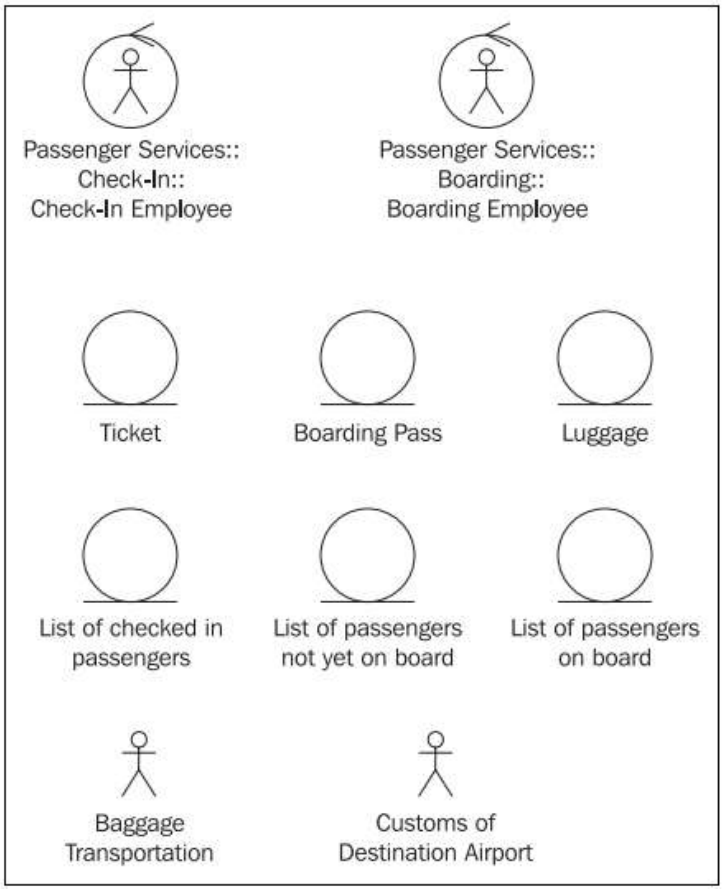
Langkah di bawah ini harus dijalankan untuk membuat class diagram:

Checklist Membuat Class Diagram pada Internal View:

11. Menemukan Class – class mana yang ada pada class diagram?
12. Membuat asosiasi antara class – class mana yang berurusan satu sama lain?
13. Memperkuat asosiasi – apa arti dari hubungannya?
14. Memasukan generalization – apakah obyek bisnis dapat dikelompokan?
15. Memverifikasi tampilan – apakah semuanya sudah benar?

MENEMUKAN CLASS – CLASS MANA YANG ADA PADA CLASS DIAGRAM?

Class pada package diagram dapat digunakan untuk class diagram dari sistem bisnis internal view, artinya pekerja dan obyek bisnis. Actor dari use case diagram juga merupakan class yang dapat digunakan pada class diagram. Pada studi kasus akan ditampilkan class sebagai berikut: (Gambar 55)



Gambar 55. Class Dari Internal View Pada Sistem Bisnis

MEBUAT ASOSIASI ANTARA CLASS – CLASS MANA YANG BERURUSAN SATU SAMA LAIN?

Pada class diagram, hubungan antara class yang ditemukan dan juga aturan bisnis digambarkan dengan asosiasi.

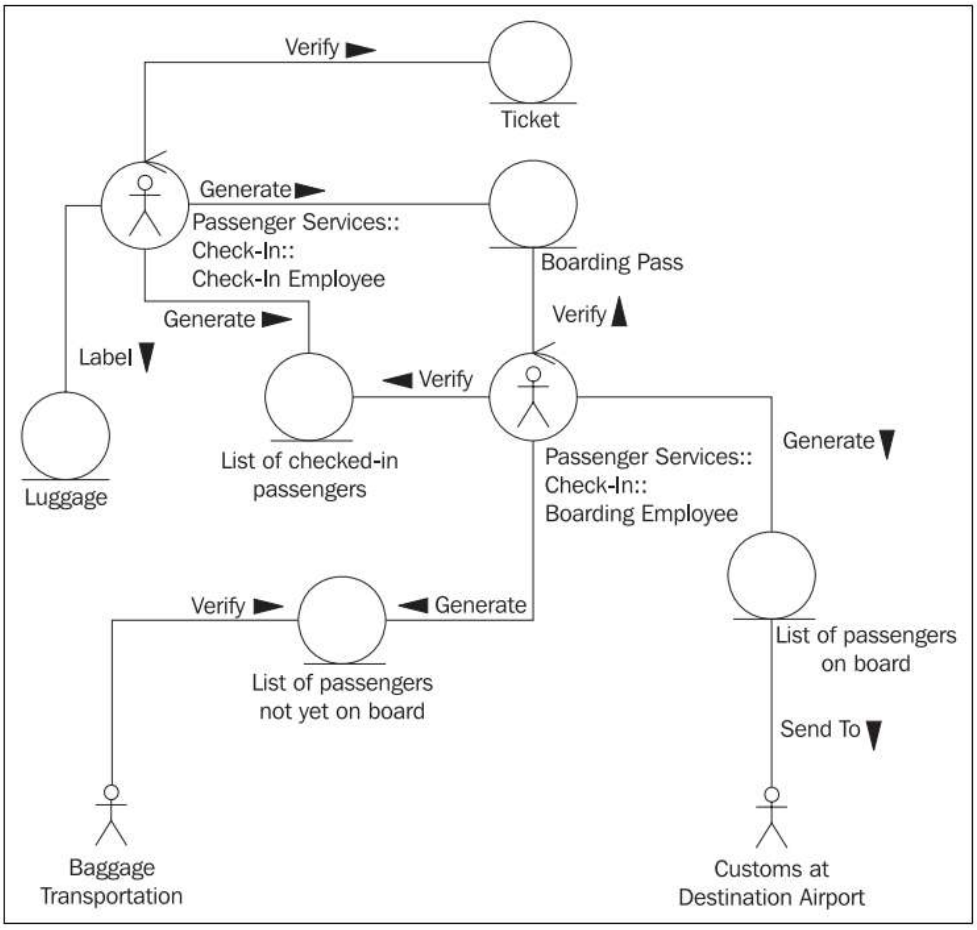
Pertanyaanya adalah:

- Hubungan apa yang ada antara pekerja, obyek bisnis, dan obyek lainnya?

Walaupun pembuatan class diagram dimulai dengan class yang sudah ditemukan, biasanya akan ditemukan class lagi pada langkah ini selama diskusi domain.

MEMPERKUAT ASOSIASI – APA ARTI DARI HUBUNGANNYA?

Asosiasi antara invidual class harus diberikan label dengan nama yang berarti, sehingga class diagram dapat dimengerti dengan mudah dan intuitif. Umumnya, arah ditambahkan pada asoasiasi, sesuai dengan arah bacanya. (lihat Gambar 56)

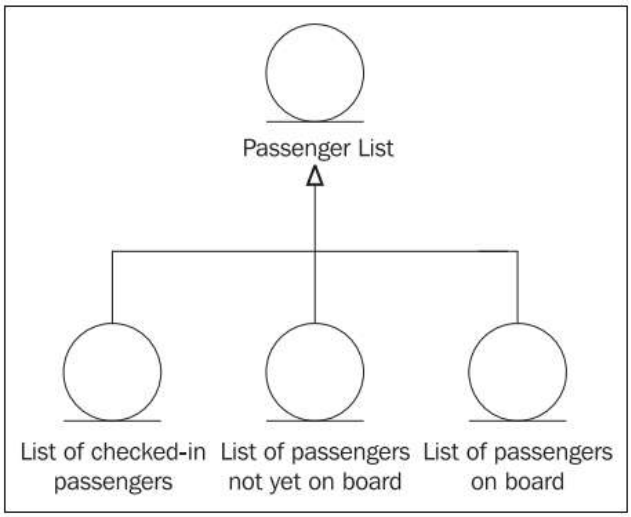


Gambar 56. Class Diagram dari Passenger Service

MEMASUKAN GENERALIZATION – APAKAH OBYEK BISNIS DAPAT DIKELOMPOKAN?

Mungkin masuk akal bagi kelompok obyek bisnis menjadi higher-rangking class yang lain. Pada studi kasus, sangat membantu untuk mengilustrasikan bahwa list of checked-in passenger, list of passenger on board, dan list of passenger not yet on board, merupakan jenis dari **<<Passenger List>>** (Lihat Gambar 57)

Gambar 57 memperlihatkan bahwa setiap list memiliki struktur yang sama.



Gambar 57. Generalization Pada Class Diagram

MEMVERFIKASI TAMPILAN – APAKAH SEMUANYA SUDAH BENAR?

Dalam menyelesaikan class diagram dapat diverifikasi dengan ceklist berikut ini:

Checklist Verifikasi Class Diagram Pada Internal View:

1. Apakah class diagram sudah selesai? Apakah semua class dari package diagram sudah ada pada class diagram?
2. Apakah semua asosiasi diberikan label yang memiliki arti? Apakah arah dari panah benar?
3. Apakah class diagram sudah benar? Membaca dengan intensif pada kolaborasi dengan knowledge carriers dan melintasi setiap jasa akan memperlihatkan banyak kesalahan.
4. Apakah tingkatan dari detail sudah dioptimalkan? Apakah diagram cukup detail sehingga meliputi semuanya, atau apakah diagram terlalu detail dan terlalu kabur karena kurang kejelasan pada aspek tertentu?

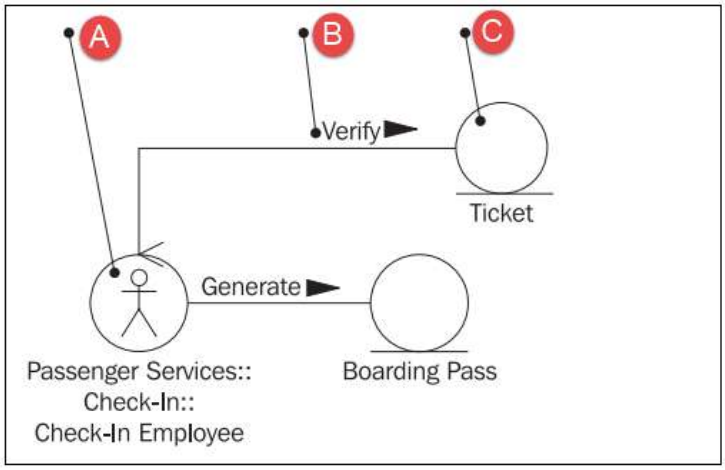
KESIMPULAN

1. Class diagram digunakan untuk mengilustrasikan bagian struktural dari sistem bisnis, artinya hubungan antara individu karyawan, obyek bisnis, dan pihak luar.
2. Elemen yang dapat digunakan dalam membuat class diagram:
   a. Class <<worker>>
   b. Class <<Business Object>>
   c. Association
   d. Generalization
3. Cara membaca class adalah membaca dari Class worker dan dilanjutkan dengan asosiasinya terhadap business object, caranya dengan menyebutkan nama asosiasi sesuai dengan arah segitiga.
4. Dalam membuat class diagram dapat mengikuti langkah-langkah berikut
   a. Menemukan Class – class mana yang ada pada class diagram?
   b. Membuat asosiasi antara class – class mana yang berurusan satu sama lain?
   c. Memperkuat asosiasi – apa arti dari hubungannya?
   d. Memasukan generalization – apakah obyek bisnis dapat dikelompokan?
   e. Memverifikasi tampilan – apakah semuanya sudah benar?

SOAL LATIHAN

1. Class diagram digunakan untuk _____
   a. Mengilustrasikan kegiatan proses bisnis secara bertahap
   b. Mengilustrasikan bagian struktur dari sistem bisnis
   c. Menggambarkan komunikasi secara kronologis
   d. Menggambarkan struktur organisasi perusahaan

Gambar Soal No.2 – No. 9

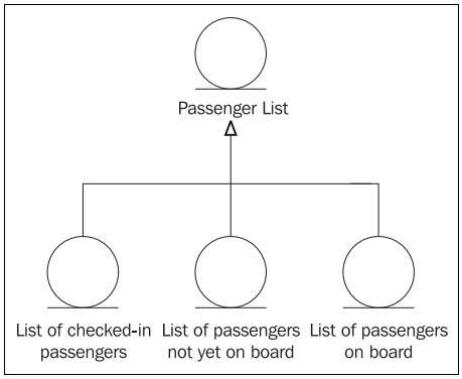


2. Nama simbol yang ditunjukan oleh huruf A adalah _____
   a. Class <<worker>>
   b. Class <<business object>>
   c. Class <<actor>>
   d. Class <<association>>
3. Fungsi dari simbol yang ditunjukan oleh huruf A adalah _____
   a. Menggambarkan dokumen atau barang yang digunakan pada sistem bisnis
   b. Menggambarkan peranan dari orang-orang yang mengeksekusi proses bisnis
   c. Menggambarkan sebuah hubungan
   d. Menggambarkan sebuah hubungan spesifik antara elemen umum dan khusus
4. Fungsi dari simbol yang ditunjukan oleh huruf B adalah _____
   a. Menggambarkan dokumen atau barang yang digunakan pada sistem bisnis
   b. Menggambarkan peranan dari orang-orang yang mengeksekusi proses bisnis
   c. Menggambarkan sebuah hubungan

d. Menggambarkan sebuah hubungan spesifik antara elemen umum dan khusus

5. Fungsi dari simbol yang ditunjukan pada huruf C adalah _____
   a. Menggambarkan dokumen atau barang yang digunakan pada sistem bisnis
   b. Menggambarkan peranan dari orang-orang yang mengeksekusi proses bisnis
   c. Menggambarkan sebuah hubungan
   d. Menggambarkan sebuah hubungan spesifik antara elemen umum dan khusus
6. Nama simbol yang ditunjukan oleh huruf B adalah _____
   a. Class <<worker>>
   b. Class <<business object>>
   c. Actor
   d. Association
7. Nama simbol yang ditunjukan oleh huruf C adalah _____
   a. Class <<worker>>
   b. Class <<business object>>
   c. Actor
   d. Association
8. Hubungan Generalization adalah _____
   a. Menggambarkan dokumen atau barang yang digunakan pada sistem bisnis
   b. Menggambarkan peranan dari orang-orang yang mengeksekusi proses bisnis
   c. Menggambarkan sebuah hubungan
   d. Menggambarkan sebuah hubungan spesifik antara elemen umum dan khusus
9. Cara membaca simbol yang ditunjukan oleh huruf B adalah _____
   a. Check-in employee memverifikasi Tiket
   b. Tiket memverifikasi Check-in Employee
   c. Passenger Service memverifikasi Tiket
   d. Check-in men-generate boarding pass

10. Perhatikan gambar di bawah ini, cara membaca relasi antara calss di bawah adalah _____



a. List of check-in passenger, List of passenger not yet on board, dan List of passenger on board merupakan jenis dari Passenger list
b. List of check-in passenger, List of passenger not yet on board, dan List of passenger on board berasosiasi dengan Passenger list
c. List of check-in passenger, List of passenger not yet on board, dan List of passenger on board turunan dari Passenger list
d. List of check-in passenger, List of passenger not yet on board, dan List of passenger on board merupakan jenis dari Passenger list

11. Di bawah ini yang BUKAN merupakan langkah membuat class diagram adalah _____

a. Menemukan class
b. Menghilangkan generalization
c. Membuat asosiasi antar class
d. Memperkuat asosiasi

12. Pada langkah pembuatan class diagram, cara untuk menemukan class actor dapat diperoleh dari _____

a. Activity Diagram
b. Communication Diagram
c. Sequence Diagram
d. Use Case Diagram

13. Diagram yang dapat digunakan untuk menemukan class adalah _____

a. Package Diagram dan Use Case Diagram
b. Package Diagram dan Activity Diagram

c. Use Case Diagram dan Activity Diagram

d. Sequence Diagram dan Package Diagram

14. Pada langkah pembuatan class diagram, cara membuat asosiasi antar class dapat menggunakan pertanyaan di bawah ini_____

a. Apa arti dari hubungannya?

b. Hubungan apa yang ada antara pekerja, obyek bisnis, dan obyek lainnya?

c. Apakah obyek bisnis dapat dikelompokan?

d. Apakah semua asosiasi yang diberikan memiliki arti?

15. Di bawah ini adalah pertanyaan yang dapat dijawab ketika melakukan langkah verifikasi class diagram, KECUALI _____

a. Apakah semua class dari package diagram sudah ada pada class diagram?

b. Aapakah semua asosiasi diberikan label yang memiliki arti?

c. Apakah tingkatan dari detail sudah dioptimalkan?

d. Apakah semua obyek bisnis yang dibutuhkan untuk menyediakan dan memproses barang dan jasa sudah dimasukan ke class diagram?

UNIVERSITAS BUDI LUHUR
FAKULTAS TEKNOLOGI INFORMASI

MODUL PERKULIAHAN #7
MODELING BUSINESS SYSTEM: ACTIVITY DIAGRAM (INTERNAL VIEW)

Capaian Pembelajaran	:	Mahasiswa memahami dan dapat menggunakan activity diagram internal view
Sub Pokok Bahasan	:	1.35. Activity Diagram a. Membaca Activity Diagram b. Membuat Activity Diagram 1.36. Kesimpulan 1.37. Soal Latihan
Daftar Pustaka	:	Grässle, P., Baumann, H. and Baumann, P., 2005. UML 2. 0 in Action: A Project-based Tutorial. Packt Publishing Ltd.

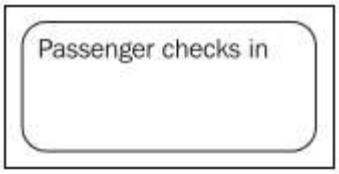
ACTIVITY DIAGRAM

Activity diagram sesuai untuk menampilkan proses internal dari sistem bisnis. Berlawanan dengan activity diagram untuk eksternal view, pada activity diagram untuk internal view hubungan terhadap actor sudah tidak lagi menjadi titik focus.

Activity diagram pada internal view juga sesuai sebagai dasar untuk instruksi.

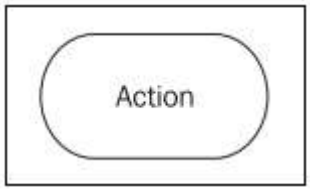
Elemen activity diagram internal view sama seperti elemen pada eksternal view.

Activity: sebuah activity diagram diilustrasikan dengan satu individual aktifitas. Dalam konteks, sebuah aktifitas menggambarkan sebuah proses bisnis. Elemen dasar dari activity adalah action dan elemen kontrol (decision, division, merge, initiation, end, dll):



Elemen saling terhubung dengan yang disebut "activity edges" dan dari "control flow", yang biasanya disebut dengan 'flow'. Eksekusi sebuah aktifitas dapat berisi aliran parallel. Batasan dapat mengelilingi activity, artinya semua activity diagram.

Action: sebuah action adalah langkah individual dalam aktifitas, contohnya, langkah perhitungan tidak di dekonstruksi lebih lanjut. Hal ini tidak perlu berarti bahwa aksi tidak dapat dibagi lagi di dunia nyata, tetapi pada diagram ini tidak akan diperbaiki lagi lebih lanjut:



Action dapat memproses informasi input dan output. Output dari satu action dapat menjadi input dari action selanjutnya dalam sebuah aktifitas. Action tertentu dapat memanggil action lain, menerima sebuah event, dan mengirim sinyal.

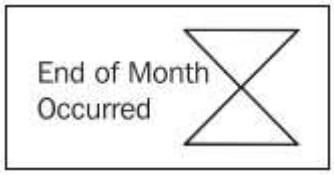
Calling an Activity (Action): dengan simbol ini sebuah aktifitas dapat dipanggil dari acitivty lain. Memanggil, artinya diri sendiri, merupakan sebuah action; keluaran dari pemanggilan adalah activity lain:

Dengan cara ini, aktifitas dapat bersarang didalamnya satu dengan yang lainnya dan dapat digambarkan dengan tingkatan detail yang berbeda.

Accepting an Event (Action): action ini menunggu even terjadi. Setelah even diterima, aliran yang datang dari action ini (dan didefinisikan pada activity diagram) kemudian dijalankan. Menerima even merupakan elemen penting dari proses bisnis pada activity diagram:

Banyak proses bisnis diawali dengan even, contohnya, memproses pesanan berdsarkan bukti pesanan, atau pengantaran berdasarkan bukti pembayaran.

Accepting a Time Even (Action): pada suatu titik waktu, action ini memulai aliran pada activity diagram. Simbol jam pasir dapat digunakan untuk menggambarkan penerimaan dari sebuah even waktu:

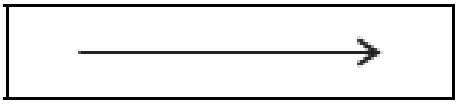


Contoh khas dari time even adalah memicu pengingat setleah batas waktu pembayaran lewat.

Sending Signal (Action): mengirimkan sinyal artinya bahwa sinyal telah dikirim ke activity penerima:

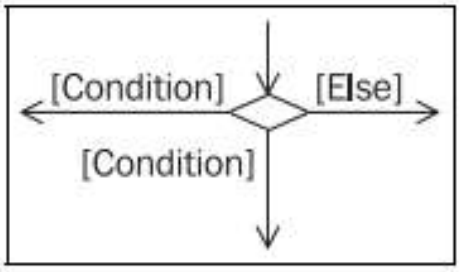
Activity penerima menerima sinya dengan action "accepting an event" dan bereaksi sesuai dengan activity, artinya berdasarkan aliran yang berasal dari node pada activity diagram.

Edge (Control Flow): Edges, digambarkan dengan panah, menghubungkan komponen individual dari activity diagram dan mengilustrasikan kontrol aliran pada activity:



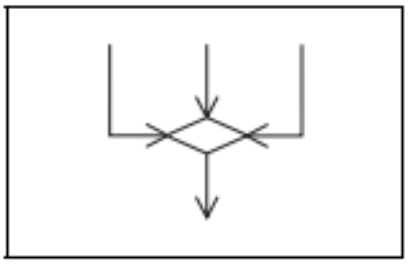
Dalam kontrol aliran sebuah panah masuk memulai langkah tunggal dari activity; setelah langkah selesai aliran akan berlanjut sepanjang panah keluar. Nama dapat diberikan pada edge (dekan dengan panah).

Decision Node: bentuk diamod di bawah ini menggambarkan sebuah titik cabang kondisi atau node decision. Sebuah decision memiliki satu input dan dua atau lebih output



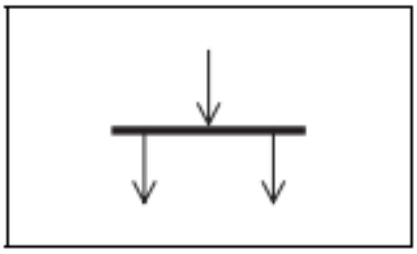
Setiap output memiliki kondisi yang ditempel padanya, yang ditulis diantara bracket. Jika sebuah kondisi terpenuhi, aliran akan lanjut sepanjang sesuai output. Sebuah 'Else' output dapat diberikan sepanjang aliran dapat diproses jika tidak ada kondisi yang terpenuhi.

Merge Node: bentuk diamond di bawah ini memiliki beberapa input dan hanya satu output:



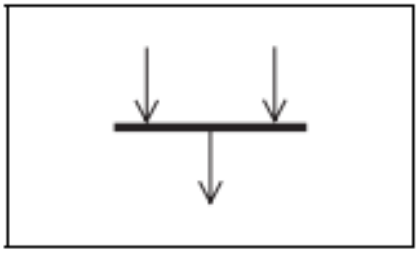
Tujuan dari merge ada menyatukan aliran. Input tidak harus synchronized; jika sebuah aliran sampai merge akan berlanjut ke output tanpa harus menunggu aliran lainnya.

Fork: pada percabngan dari aliran pada satu atua model parallel dapat menggunakan garis synchronous, yang digambarkan dengan garis horizontal atau vertika yang tebal.



Percabangan memungkinkan aliran parallel dalam aktifitas. Sebuah fork memiliki satu input dan dua atau lebih output.

Join: Untuk penggabungan dari dua atau lebih aliran parallel juga dapat menggunakan sinkronisasi bar, yang digambarkan dengan haris tebal secara horizontal maupun vertical:



Dalam penggabungan terjadi sinkronisasi, artinya aliran akan berlajut hanya jika setelah *semua* aliran masuk sampai ke titik penggabungan. Join memiliki dua atau lebih input dan satu output

Intial Node: initial node adalah titik awal dari aktifitas. Sebuah aktifitas dapat memiliki lebih dari satu initial node; dalam kasus ini beberapa aliran dimulai pada permulaan sebuah aktifitas:

Sangat mungkin sebuah aktifitas tidak memiliki initial node, tetapi diawali oleh sebuah even (action: menerima even)

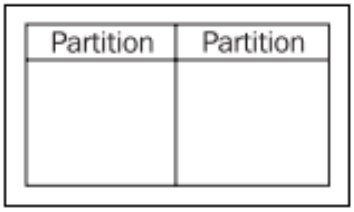
Activity Final Node: activity final node menunjukan bahwa semua aktifitas sudah selesai. Sebuah activity diagram dapat memiliki lebih dari satu exit dalam bentuk activity final node:

Jika beberapa aliran parallel terdapat dalam sebuah aktifitas, semua aliran berhenti pada saat aktifitas mencapai final node.

Flow Final Node: sebuah flow final node mengakhiri satu buah flow. Tidak seperti activity final node, yang mengakhiri seluruh aktifitas, sampai pada flow final node tidak memiliki efek pada aliran parallel lainnya yang diproses dalam aktifitas pada satu titik waktu yang sama:

Dengan cara ini, aliran parallel dapat diakhiri secara individu dan secara selektif.

Activity Partition: elemen individual dari activity diagram dapat dibagi menjadi individual area atau 'partisi'. Beberapa kriteria yang menuntun ke pembuatan dari partisi: entitas organisasi, cost center, lokasi, dll:

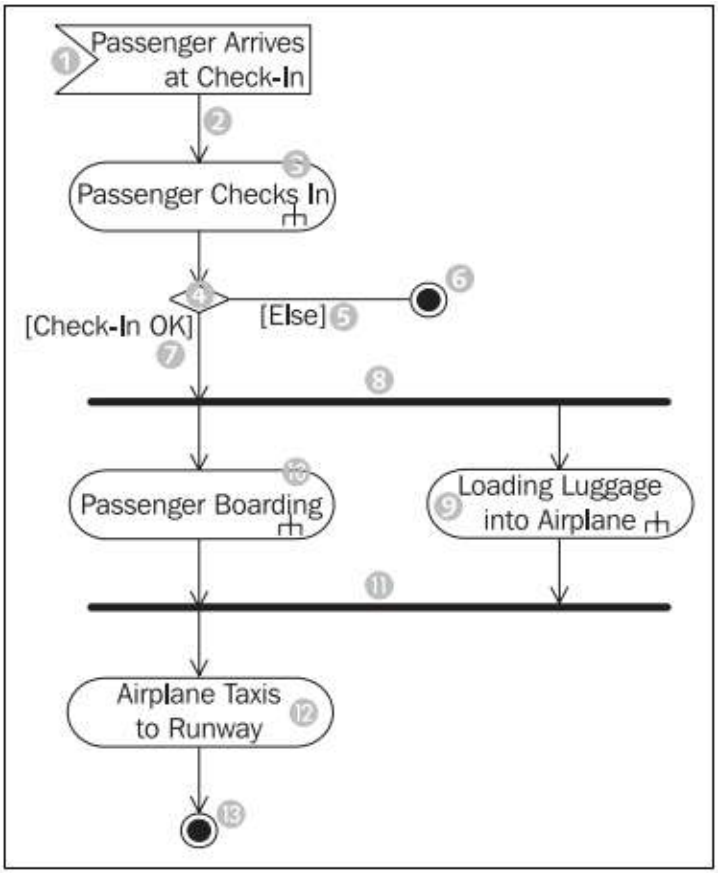


Langkah individual dari sebuah aktifitas akan ditugaskan pada partisi. Setiap partisi adalah rangkaian terpisah dari partisi yang ada disebelahnya dengan jalur horizontal atau vertical yang berkelanjutan; berdasarkan akar kata terdapat istilah **swimlanes**. Setiap partisi menerima nama. Partisi dapat diatur dalam bentuk dua dimensi; pada kasus ini activity diagram dibagi menjadi kolom individual seperti grid.

1.25. MEMBACA ACTIVITY DIAGRAM

Cara membaca activity diagram internal sama seperti membaca activity diagram eksternal.

Mulai membaca dimulai dari **initial node**, atau pada Gambar 30 diawali dengan menerima even **passenger arrives at check-in** (1), dan berlanjut sepanjang panah dari control flow (2). Action selanjutnya **passenger checks in** (3) artinya bahwa pada titik ini aktifitas **'passenger checks in'** telah diproses. Hal ini digambarkan lebih detail pada activity diagram lain yang ditandai oleh 'fork' pada simbol action:



Gambar 58. Activity Diagram

Jika mengikuti control flow, selanjutnya akan sampai pada cabang kondisi atau decision node (4); jika check-in OK maka langkah selanjutnya adalah mengikuti sepanjang control flow. Selain itu (5), penumpang tidak dapat terbang dan tugas dari passenger service selesai. Hal ini dapat dilihat pada titik hitam dengan border – activity final node.

Setelah sukses melakukan check-in (7) selanjutnya bertemu dengan bar kotak hitam. Semua panah yang berasal dari bar ini (7) melambangkan aliran yang diproses bersamaan. Sementara bagasi dimasukan ke pesawat (9) penumpang masuk ke pesawat (10). Antara titik (8) dan titik (11) aliran adalah independen antara satu dengan lainnya. Pada bar kotak hitam kedua (11) proses aliran yang dilakukan bersamaan (9 dan 10) digabungkan, artinya bahwa hanya ketika penumpang di dalam pesawat (10) *dan* bagasi sudah dimasukan ke dalam pesawat (9), control flow akan berlanjut di bawah bar kotak hitam (11). Pada contoh, satu atau beberapa action (12) dan menyusul mengikuti final state (13), artinya bahwa setelah penumpang di dalam pesawat (10) dan bagasi sudah masuk ke pesawat (9), pesawat dapat menunggu dengan menuju ke runway (12). Seperti yang terlihat action terakhir adalah airplane taxis toward runway (12) yang hanya di definisikan sebagai action tunggal, walaupun proses ini sangat kompleks dan dapat digambarkan dengan bentuk activity diagram lainnya. Tetapi, dalam konteks, tidak perlu menggambarkan langkah ini secara detail.

Acitivity Diagram pada Gambar 30Gambar 31 dibagi menjadi dua partisi: **passenger** (1) dan **passenger service** (2). Passenger, misalnya, melakukan **showing ticket at check-in caounter** (3), **checking luggage** (4), dan **paying fee** (5). Semua action yang lain diletakan pada partisi (swimlane) dari passenger service (2) dan dilakukan oleh passenger service.

1.26. MEMBUAT ACTIVITY DIAGRAM

Pada dasarnya, pembuatan activity diagram pada internal view terjadi sama persis seperti membuat activity diagram pada eksternal view.

Di bawah ini merupakan checklit dan penjelasan dari setiap langkah individual yang sudah disesuaikan dengan acitivty diagram internal view:

Checklist Membuat Activity Diagram pada Internal View:

5. Mengumpulkan sumber informasi – bagaimana cara mengetahuinya?
6. Menemukan aktifitas dan action – aktifitas apa yang harus dilakukan sehingga barang dan jasa yang digunakan actor dapat disediakan dan diberikan?
7. Mengadopsi actor dari business use case – siapa yang bertanggung jawab untuk setiap action?
8. Menghubungkan action – dalam urutan apa action di proses?
9. Memperbaiki aktifitas – apakah ada acitity diagram lain yang harus ditambahkan?
10. Memverifikasi tampilan – apakah semua sudah benar?

MENGUMPULKAN SUMBER INFORMASI – BAGAIMANA CARA MENGETAHUINYA?

Ketika membuat activity diagram pada internal view, petunjuk yang sama seperti membuat use case diagram, benar untuk mendapatkan informasi yang dibutuhkan.

Sebagai langkah pertama, sangat penting untuk mencari pembawa pengetahuan (knowledge carrier), agar analis dan knowledge carrier dapat mengerjakan prinsip dasar bersama-sama. Knowledge carrier adalah, seperti:

8. Orang yang terlibat dalam melakukan, mengoperasikan, dan mengatur proses bisnis
9. Pengguna dari IT System yang sama atau terkait.
10. Customers, orang yang sering menjadi knowledge carrier yang kritis dan kreatif
11. Partner Bisnis
12. Domain Expert

13. Management
14. Pengamat eksternal (External Observer)

Beberapa teknik yang membantu telah terbukti praktis untuk analis memahami proses bisnis:

11. Mengamati karyawan yang sedang bekerja
12. Berpartisipasi dalam proses bisnis yang sedang diteliti
13. Mengambil peranan sebagai outsider (contohnya: sebagai customer)
14. Memberikan survey
15. Melakukan wawancara
16. Brainstorming dengan semua orang yang terlibat
17. Berdiskusi dengan domain expert
18. Mereview formulir, dokumentasi, spesifikasi, buku panduan, dan alat bantu yang sudah ada
19. Menggambarkan struktur organisasi dan alur kerja manajemen
20. Mereview chart organisasi dan deskripsi pekerjaan

Hasil dari langkah ini adalah berupa kumpulan formulir, instruksi kerja, survey yang sudah selesai, deskripsi proses yang sudah ada, obyek bisnis seperti tiket atau boarding pass, dan lain lain. Overview ini sering tidak lengkap, dan harus diperluas selama proses pembuatan model.

MENEMUKAN AKTIFITAS DAN ACTION – AKTIFITAS APA YANG HARUS DILAKUKAN SEHINGGA BARANG DAN JASA YANG DIGUNAKAN ACTOR DAPAT DISEDIAKAN DAN DIBERIKAN?

Pada langkah ini, dapat dipinjam dari use case dan action pada activity diagram di eksternal view. Berikut pertanyaan yang harus diberikan kepada proses bisnis individual yang akan digambarkan pada eksternal view: bagaimana proses internal terjadi dan seperti apa proses internal bisnisnya? Dengan menjawab pertanyaan ini dapat membantu menemukan aktifitas dan action.

1. Langkah kerja mana yang harus dilakukan oleh karyawan dari sistem bisnis untuk menyediakan dan memberikan jasa?

2. Apa yang dilakukan setiap karyawan?
3. Kegiatan diluar mana yang memulai aktifitas dan action?

Seringnya, ketika menemukan pada dokumentasi aliran yang sudah ada, baik secara informal atau terstruktur, dapat digunakan untuk menemukan aktifitas.

MENGADOPSI ACTOR DARI BUSINESS USE CASE – SIAPA YANG BERTANGGUNG JAWAB UNTUK SETIAP ACTION?

Pada pokoknya, pekerja dan unit organisasi dari package diagram bertanggun jawab pada action. Actor dari use case diagram juga digunakan, selama actor terlibat dalam penggambaran proses bisnis.

Setiap pekerja, setiap unit organisasi, dan setiap actor bertanggung jawab untuk aktifitas tertentu dan dimasukan kedalam partisi aktifitas (simlane) sebagai pihak yang bertanggungjawab. Individual action ditugaskan untuk tanggung jawab ini.

Jika activity diagram diperbaiki, hal ini memungkinkan bahwa area tanggung jawab lain ditambahkan, sebagai contoh, posisi individu atau team.

MENGHUBUNGKAN ACTION – DALAM URUTAN APA ACTION DI PROSES?

Menghubungkan individual action pada aliran menghasilkan activity diagram awal, yang menggambarkan proses bisnis internal. Pertanyaan di bawah ini membantu pembuatan control flow:

1. Dalam order apa action diproses?
2. Kondisi apa yang harus dipenuhi agar action bisa dijalankan?
3. Dimana cabang diperlukan?
4. Action apa yang terjadi bersamaan?
5. Apakah action yang sudah selesai diperlukan sebelum aliran dapat berlanjut ke action lain?

MEMPERBAIKI AKTIFITAS – APAKAH ADA ACITITY DIAGRAM LAIN YANG HARUS DITAMBAHKAN?

Sangat mungkin bahwa setiap individual action harus dibagi lebih lanjut atau diperbaiki dengan activity diagram lainnya. Skenario berbeda akan digambarkan pada activity diagram lainnya.

MEMVERIFIKASI TAMPILAN – APAKAH SEMUA SUDAH BENAR?

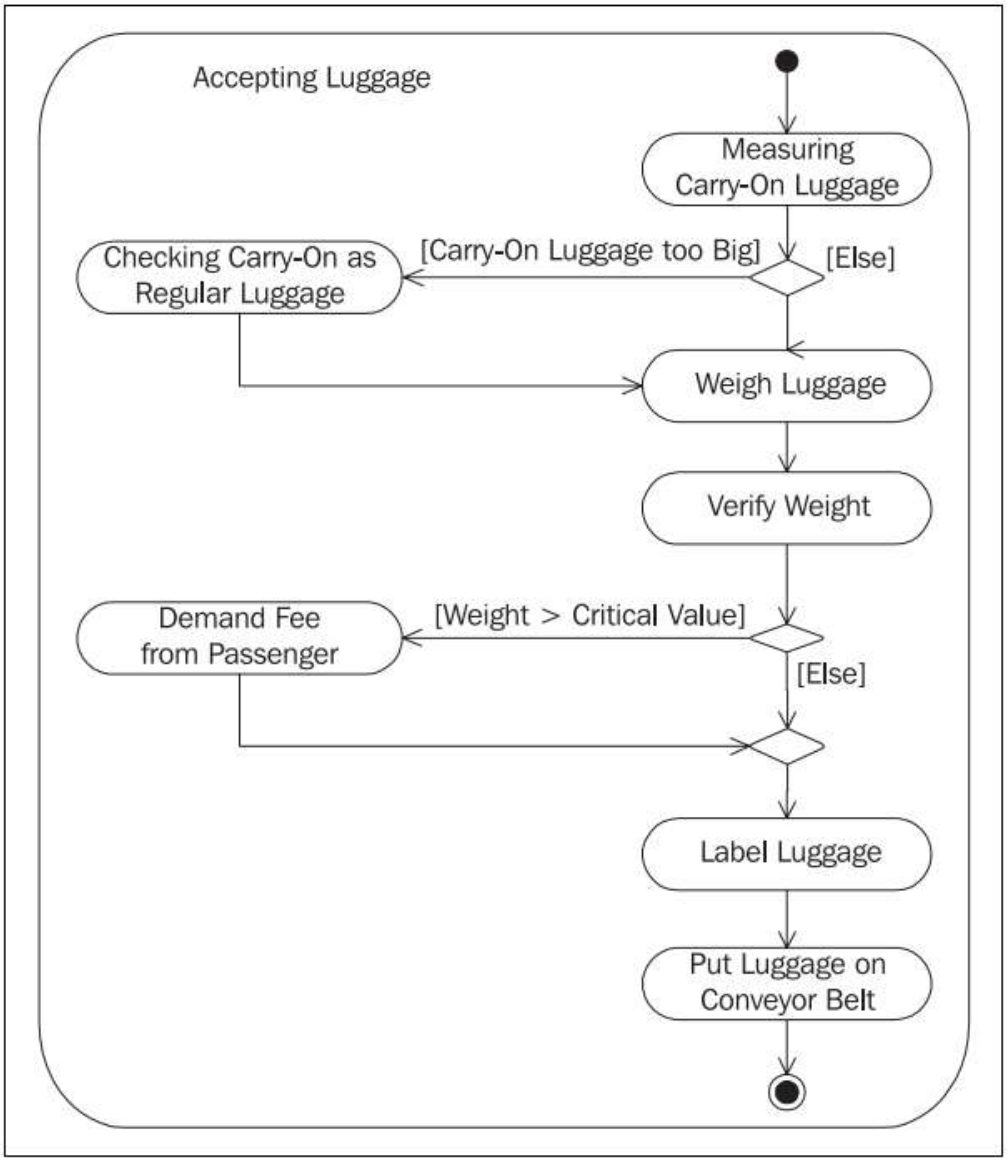
Activity diagaram pada internal view juga harus diverifikasi dalam hal ketepatan isinya. Hal ini harus dilakukan berkolaborasi dengan knowledge carriers.

Checklist Verifikasi Activity Diagram Pada Internal View:

1. Ketika membuat activity diagram dari eksternal view, harus selalu diingat bahwa hanya prosedur internal dan proses bisnis internal yang relevan.
2. Kondisi dari output berbda pada sebuah decision node tidak boleh tumpang tindih. Jika tidak, control flow akan ambigu – tidak jelas kemana aliran berlanjut setelah decision node
3. Kondisi harus memasukan semua kemungkinan. Jika tidak, control flow akan terjebak atau berhenti. Jika ragu, masukan sebuah output dengan kondisi 'else'
4. Fork dan join harus seimbang. Jumlah dari aliran yang meninggalkan fork harus sama dengan jumlah aliran yang berakhir pada join yang sesuai.

Pada Gambar 59 menggambarkan activity diagram yang mewakliki proses internal dari aktifitas accepting luggage pada saat check-in di passenger service:

Aktifitas **accepting luggage**, pada gambar Gambar 59, dilakukan oleh passenger service. Tidak penting untuk penumpang atau untuk baggage transportation action apa yang menjalankan dengan cara bagaimana. Penumpang hanya tertarik apakah tas yang mereka bawa terlalu besar dan apakah mereka harus membayar kelebihan beratnya; baggage transportation membutuhkan label pada setiap bagasi. Semua detail lain yang ditampilkan pada diagram merupakan detail proses internal dari passenger service dan kemudian diberikan labal sebagai 'internal view'



Gambar 59. Activity Diagram dari Internal Activity "Accepting Luggage"

KESIMPULAN

1. Activity Diagram Internal View sudah tidak lagi focus pada hubungan terhadap actor. Activity diagram internl view dapat digunakan sebagai dasar untuk instruksi atau petunjuk.
2. Cara membaca activity diagram internal view caranya sama dengan cara membaca activity diagram eksternal view
3. Dalam membuat activity diagram internal view dapat mengikuti langkah-langkah berikut:
   a. Mengumpulkan sumber informasi – bagaimana cara mengetahuinya?

b. Menemukan aktifitas dan action – aktifitas apa yang harus dilakukan sehingga barang dan jasa yang digunakan actor dapat disediakan dan diberikan?
c. Mengadopsi actor dari business use case – siapa yang bertanggung jawab untuk setiap action?
d. Menghubungkan action – dalam urutan apa action di proses?
e. Memperbaiki aktifitas – apakah ada acitity diagram lain yang harus ditambahkan?
f. Memverifikasi tampilan – apakah semua sudah benar?

SOAL LATIHAN

1. Elemen pada activity diagram yang digunakan untuk menggambarkan satu aktivitas individual adalah _____
   a. Activity
   b. Action
   c. Calling and Activity
   d. Accepting an Event (Action)
2. Elemen yang digunakan untuk menggambarkan langkah individual dalam aktivitas adalah _____
   a. Activity
   b. Action
   c. Calling and Activity
   d. Accepting an Event (Action)
3. Elemen yang digunakan untuk mememanggil aktivitas lain adalah _____
   a. Activity
   b. Action
   c. Calling and Activity
   d. Accepting an Event (Action)
4. Elemen yang digunkaan untuk menunggu even terjadi baru dapat melanjutkan action lainnya adalah _____
   a. Activity
   b. Action

c. Calling and Activity
d. Accepting an Event (Action)

5. Elemen yang digunakan untuk memulai aliran pada activity diagram pada suatu titik waktu adalah
   a. Accepting a Time Event (Action)
   b. Sending Signal (Action)
   c. Edge (Control Flow)
   d. Decision Node
6. Elemen yang digunakan untuk mengirimkan sinya adalah _____
   a. Accepting a Time Event (Action)
   b. Sending Signal (Action)
   c. Edge (Control Flow)
   d. Decision Node
7. Elemen yang digunakan untuk menghubungkan komponen individual dari activity diagram adalah _____
   a. Accepting a Time Event (Action)
   b. Sending Signal (Action)
   c. Edge (Control Flow)
   d. Decision Node
8. Elemen yang memiliki satu input dan dua atau lebih keluaran sesuai dengan kondisi yang ditentukan adalah _____
   a. Accepting a Time Event (Action)
   b. Sending Signal (Action)
   c. Edge (Control Flow)
   d. Decision Node
9. Elemen yang memiliki beberapa input dan hanya satu output adalah _____
   a. Merge Node
   b. Fork
   c. Join
   d. Initial Node
10. Elemen yang terdapat percabangan parallel adalah _____
    a. Merge Node

b. Fork
c. Join
d. Initial Node

11. Elemen yang digunakan untuk menggabungkan dua atua lebih aliran parallel adalah _____
    a. Merge Node
    b. Fork
    c. Join
    d. Initial Node
12. Elemen yang digunakan untuk menggambarkan titik awal aktivitas adalah _____
    a. Merge Node
    b. Fork
    c. Join
    d. Initial Node
13. Elemen yang digunakan untuk menandakan seluruh aktivitas sudah selesai adalah _____
    a. Activity Final Node
    b. Flow Final Node
    c. Activity Partition
    d. Actor
14. Elemen yang digunakan untuk mengakhiri satu buah flow tetapi tidak mengakhiri seluruh aktivitas adalah _____
    a. Activity Final Node
    b. Flow Final Node
    c. Activity Partition
    d. Actor
15. Elemen yang digunakan untuk membagi area aktivitas adalah _____
    a. Activity Final Node
    b. Flow Final Node
    c. Activity Partition
    d. Actor

16. Di bawah ini adalah teknik praktis untuk memahami proses bisnis, KECUALI _____
    a. Mengamati karyawan yang sedang bekerja
    b. Berpartisipasi dalam proses bisnis
    c. Memberikan survey
    d. Memperbaiki struktur organisasi
17. Maksud dari langkah menemukan aktifitas dan action pada pembuatan activity diagram internal view adalah _____
    a. Menentukan siapa yang bertanggung jawab untuk setiap action
    b. Menentukan aktifitas apa yang harus dilakukan sehingga barang dan jasa yang digunakan aktor dapat diberikan atua disediakan
    c. Mengatahui urutan apa action diproses
    d. Mencari apakah ada activity diagram lain yang dapat ditambahkan.
18. Pada saat menemukan aktifitas dan action ketika membuat activity diagram internal view, dapat menajwab pertanyaan di bawah ini _____
    a. Dalam order apa action diproses?
    b. Kegiatan diluar mana yang memulai aktifitas dan action?
    c. Kondisi apa yang harus dipenuhi agar action dapat dijalankan?
    d. Dimana cabang diperlukan?
19. Pada saat menghubungkan action pada activity diagram internal view, dapat menjawab pertanyaan di bawah ini, KECUALI _____
    a. Dalam order apa action diproses?
    b. Kegiatan diluar mana yang memulai aktifitas dan action?
    c. Kondisi apa yang harus dipenuhi agar action dapat dijalankan?
    d. Dimana cabang diperlukan?
20. Dalam melakukan verifikasi dari activity diagram internal harus berkolaborasi dengan _____
    a. Domain expert
    b. Pimpinan
    c. Knowledge carrier
    d. Analis

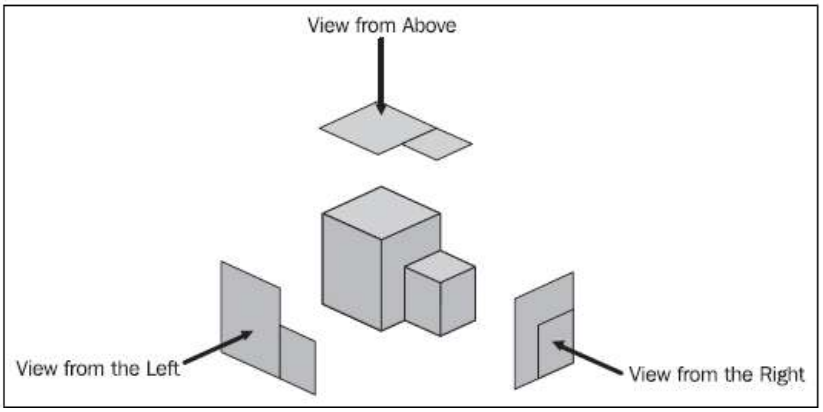
UNIVERSITAS BUDI LUHUR
FAKULTAS TEKNOLOGI INFORMASI

MODUL PERKULIAHAN #9
MODELING IT SYSTEM: EXTERNAL VIEW

Capaian Pembelajaran	:	Mahasiswa memahami dan dapat menggunakan Modeling IT system eksternal view
Sub Pokok Bahasan	:	1.38. Modeling IT System 1.39. External View a. Pandangan/View User b. Elemen dari External View 1.40. Use Case Diagram a. Membaca Use Case Diagram b. Query Event (Permintaan) dan Mutation Event (Perpindahan) 1.41. Use Case Diagram / System Sequence Diagram a. Membaca Sistem Sequence Diagram/Use Case Sequence Diagram 1.42. Membangun External View 1.43. Kesimpulan 1.44. Latihan Soal
Daftar Pustaka	:	Grässle, P., Baumann, H. and Baumann, P., 2005. UML 2. 0 in Action: A Project-based Tutorial. Packt Publishing Ltd.

MODELING IT SYSTEM

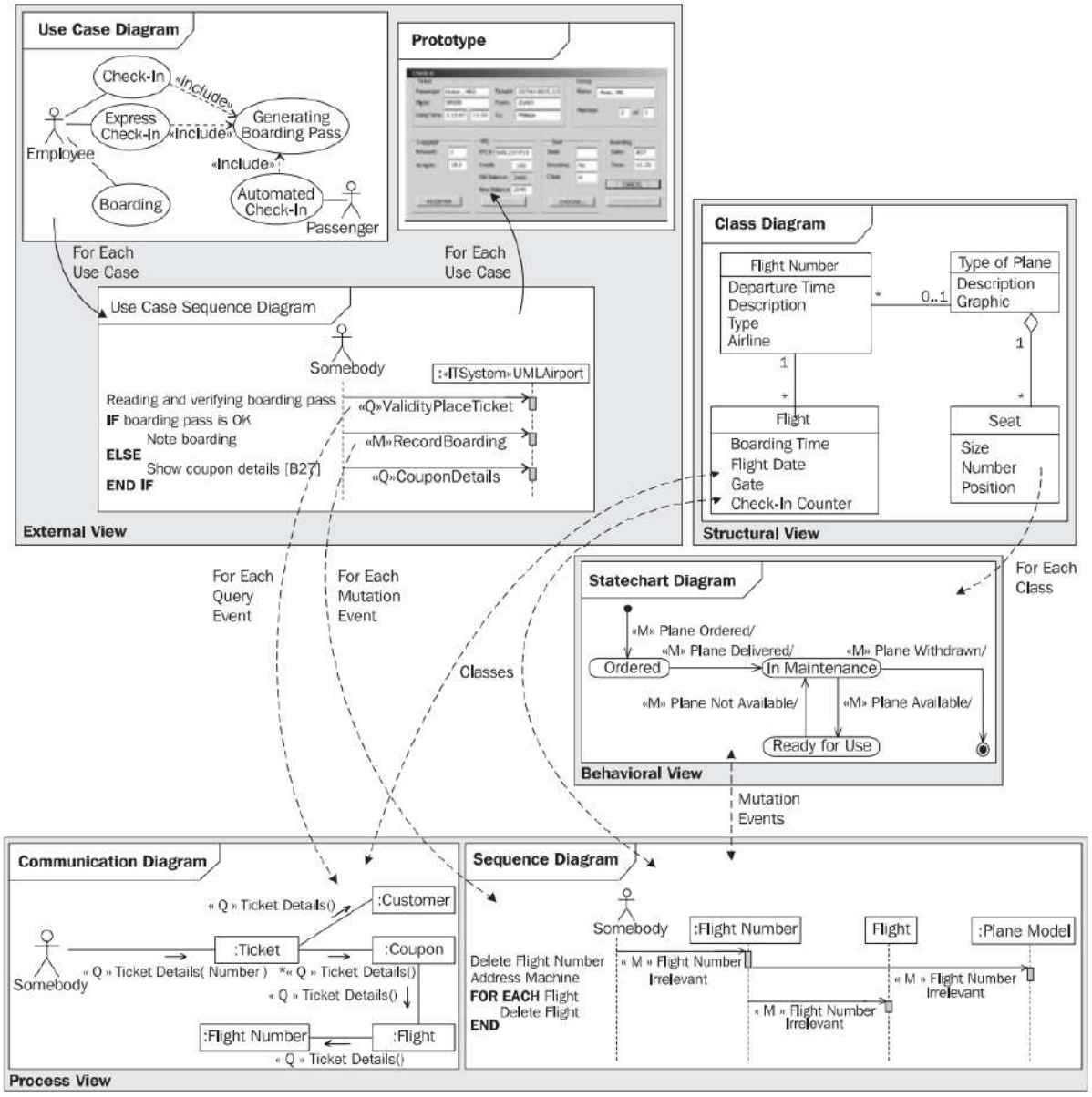
Pemodelan adalah dasar keberhasilan pengembangan dan implementasi sistem TI baru. Model yang benar dan lengkap memastikan bahwa pada akhirnya user mendapatkan sistem TI yang mereka butuhkan.



Gambar 60. Cara Pandang dari Berbagai Sudut Sistem IT

Dalam bab ini, kami menunjukkan bagaimana model konseptual sistem TI dapat dikembangkan dengan bantuan UML. Meninmbang aturan 80:20, kita tidak perlu menggunakan semua diagram UML. Dalam kenyataan tidak realistis untuk memodelkan segala sesuatu secara mendalam dengan UML, karena dalam tahap implementasi awal, kita tidak dapat memprediksi perubahan selama tahap pemodelan. Model harus dikembangkan dengan upaya seminimal mungkin.

Model sistem TI terdiri dari empat view (pandangan) yang berbeda, masing-masing menekankan aspek-aspek tertentu dan yang terkait erat satu sama lain. Pendekatan model yang terdiri dari berbagai pandangan ini, diilustrasikan Gambar 60, dan diagram UML termasuk di dalamnya dalam Gambar 61:



Gambar 61. Perbedaan pandangan dalam sebuah IT System

- External View—Use case diagram dan use case sequence diagram
- Structural View—Class diagram
- Interaction View(Interaksi)—Sequence diagram and communication diagram
- Behavioral View (Perilaku)—Statechart diagram

Masing-masing pandangan ini menekankan aspek-aspek tertentu dengan mengabaikan pandangan lain. Semua gabungan pandangan membentuk model fungsionalitas sistem TI yang lengkap:

- **External View** menunjukkan skenario sistem TI dalam bentuk diagram use case UML dan prototipe interface (*interface*). Fungsi mana yang disediakan sistem TI untuk user dijelaskan disini.
- **Structural View** menunjukkan kelas yang relevan dari sistem TI dalam bentuk class diagram UML. **Struktur informasi** yang diajukan dalam sistem TI dijelaskan disini.
- **Behavioral View** menunjukkan **perilaku tiap objek** dalam bentuk *statechart diagram*. Semua kemungkinan yang dapat terjadi dengan objek yang diajukan dalam sistem TI.
- **Interaction View** menunjukkan **aliran yang terjadi selama perpindahan atau permintaan** dalam sistem TI, dalam bentuk *sequence diagram* dan *communication diagram*. Ketika user menggunakan sistem IT.

Event/Kejadian/peristiwa adalah suatu hubungan/link yang menyatukan view yang berbeda yang berisi tiga dari empat view:

- *External view*, tiap use case digambarkan sebagai urutan event yang dikirim ke sistem TI.
- *Behavioral view* menunjukkan di masing-masing kelas bagaimana objek merespons tiap event .
- *Interaction view* menunjukkan bagaimana masing-masing event dalam sistem TI diteruskan ke objek yang terpengaruh.

Hanya pada class diagram (*structural view*), event tidak terlihat.

Class diagram menunjukkan kelas dan hubungan, tetapi bukan aspek dinamis antar kelas.

Kita tidak menggunakan semua jenis diagram yang disediakan UML untuk memodelkan sistem TI. Umumnya, kombinasi diagram yang dijelaskan dalam bab ini bermanfaat untuk memodelkan sistem TI yang konsisten dan lengkap. Pada model lain, kombinasi diagram bisa berbeda. Misal, activity diagram dapat bermanfaat untuk modelkan sistem bisnis.

EXTERNAL VIEW

1.27. PANDANGAN/VIEW USER ATAU "SAYA TIDAK PEDULI BAGAIMANA CARA KERJANYA, SELAMA SISTEM BEKERJA DENGAN BAIK"

Kebanyakan user yang menggunakan peralatan modern, tidak peduli bagaimana cara kerja mesin atau software didalam peralatan tersebut. Mereka hanya ingin tau untuk apa mesin itu digunakan dan memenuhi kebutuhan fungsional user.

View ini disebut juga kotak hitam (blackbox). Biasanya dalam sistem TI, user tidak perlu melihat kedalam, tidak perlu tahu cara kerjanya, yang penting memenuhi persyaratan atau kebutuhan contoh seperti menggunakan mesin minuman kopi otomatis (vending machine) dengan menggunakan payment gateway tertentu dan keluar kopi.

Fungsi yang didefinisikan dalam view ini pada harus digunakan untuk memverifikasi jika sistem TI memenuhi persyaratan.

Eksternal view terdiri dari elemen dengan menggunakan use case, sequence diagram dari use case, dan prototipe antarmuka(*interface*). Tidak cukup dengan mencatat dalam bentuk kalimat, dibutuhkan diagram UML untuk membantu menghindari salah tafsir dari kebutuhan dan sesuai dengan persyaratan TI.

Inrface ATM

Interface sistem kopi

Interface sistem Self Check-in

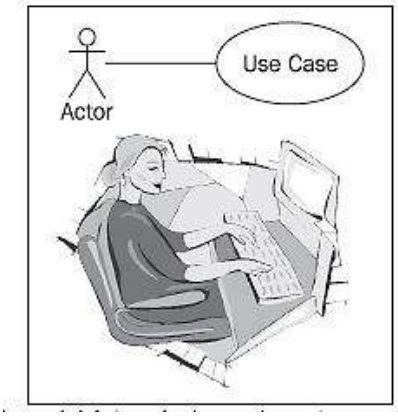
Interface sistem Go Food

Gambar 62. Contoh Interface-Interface Sistem

Elemen penting dari suatu sistem adalah interface user. Interface user ATM, misalnya, terdiri dari monitor kecil, kunci, dan tempat untuk kartu, tagihan, dan tanda terima, serta bunyi bip. Interface user adalah satu-satunya jalur akses yang dimiliki user ke suatu sistem. Interface user mewakili tampilan fungsi perangkat(*device*). Bila tidak ada dalam interface pasti tidak dapat diakses. Interface user bersifat statis. Pada view ini tidak dijelaskan bagaimana sistem seharusnya digunakan, dan elemen operasi mana yang harus digunakan untuk menyelesaikan tugas tertentu.

Karena itu, interface user memerlukan instruksi manual. Ini berarti bahwa kita memerlukan deskripsi yang mengidentifikasi tindakan yang mungkin dan urutan apa yang harus diikuti untuk sistem yang akan digunakan dan bermakna. Pada UML, tindakan (*actions*) ini disebut **use case**. Use case adalah instruksi manual untuk interface user. Seperti dalam buku manual, hanya berisi instruksi penting (bermakna).

Contoh action flow yang penting untuk case telepon: mencari nomor telepon, menunggu seseorang untuk menjawab telepon, berbicara, dan menutup telepon. Aliran yang tidak didefinisikan dalam instruksi manual dianggap tidak bermakna dan tidak didukung oleh sistem.



Gambar 63. Actor Sedang Menjalankan Use Case

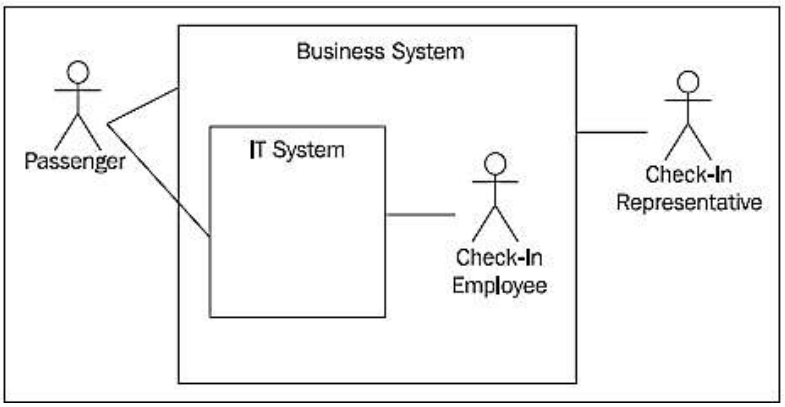
Pendekatan terbaik untuk memodelkan use case sistem TI adalah dengan membayangkan seorang user yang duduk di depan keyboard dan bekerja dengan sistem TI. User menjadi aktor, dan use case tidak lebih dari deskripsi abstrak dari aktivitas user.

Aktor:

- Berinteraksi langsung dengan suatu sistem
- Selalu berada di luar sistem yang berinteraksi dengannya

Untuk sistem TI, ini berarti bahwa aktor selalu orang yang secara langsung mengoperasikan sistem TI.

Pekerja yang mengoperasikan sistem TI adalah bagian dari sistem bisnis sehingga tidak dapat menjadi aktor dari sistem bisnis (Lihat Modul Pemodelan Sistem Bisnis)

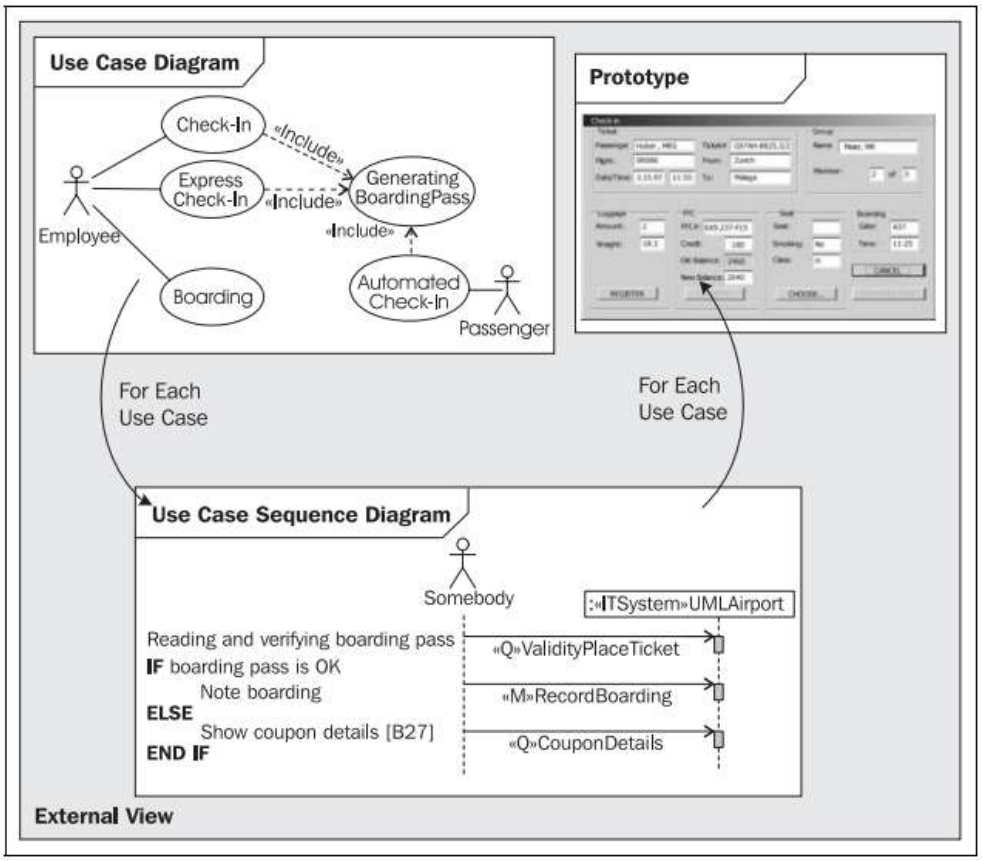


Gambar 64. Aktor Dalam Sistem Bisnis dan Sistem TI

Lihat situasi yang ditunjukkan pada Gambar 64:

- Seorang penumpang (*passenger*) adalah aktor dari sistem bisnis, dan umumnya berurusan dengan karyawan check-in (*check-in employee*). Namun, ia juga bisa menjadi aktor langsung dari sistem TI, misalnya saat menggunakan mesin check-in otomatis.
- Seorang karyawan check-in (*check-in employee*) adalah bagian dari sistem bisnis. Karena itu dia bukan aktor sistem bisnis. Di sisi lain, sebagai user ia adalah aktor dari sistem TI.
- Perwakilan check-in(*check-in representative*) , dimana melakukan check-in untuk menggantikan orang lain, hanya aktor dari sistem bisnis, karena dia tidak dapat melakukan check-in otomatis dan oleh karena itu tidak pernah memiliki kontak langsung dengan sistem TI.

Intinya, untuk menentukan aktor dan use case baiknya berdiskusi dengan user atau pakar tentang fungsionalitas yang ada di sistem TI yang nantinya akan terlihat dari luar.



Gambar 65. Eksternal View

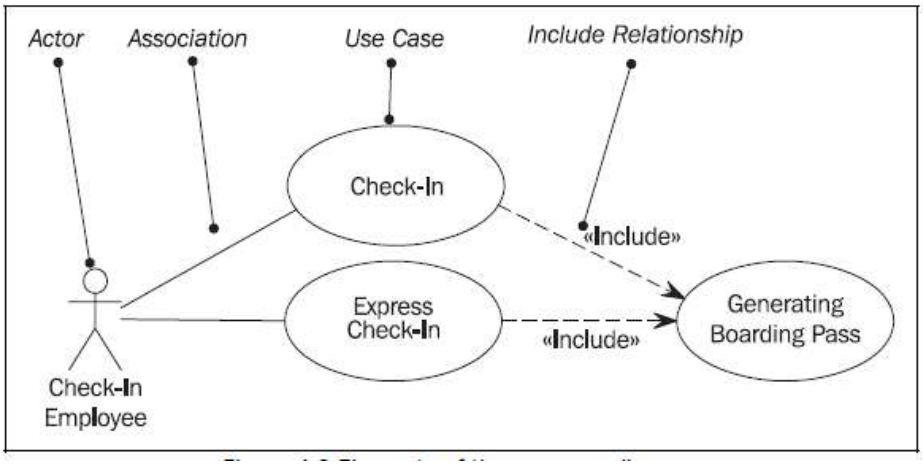
1.28. ELEMEN DARI EXTERNAL VIEW

View eksternal sistem TI terdiri dari tiga elemen berikut:

- **Use case diagram** menunjukkan semua user sistem IT (aktor) dan semua skenario user dengan sistem IT (use case). Pada umumnya dibahas tiap use case, tetapi pada prakteknya, seringkali terlalu rumit untuk menggambarkan semua use case sistem TI dalam satu diagram karena akan terlalu penuh.
- **Use case sequence diagram** menunjukkan proses selama interaksi antara user dan sistem IT untuk tiap use case .
- **Prototipe interface** menunjukkan bagaimana tampilan interface dari use case itu.

Ketiga elemen yang dikombinasikan memberikan gambaran yang baik atas sistem TI dari perspektif user.

USE CASE DIAGRAM

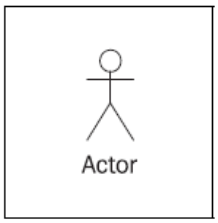


Gambar 66. Elemen Use Case Diagram

Gunakan diagram use case, seperti yang ditunjukkan pada Gambar 66, dengan elemen-elemen berikut:

Aktor

Anda dapat menggambarkan aktor sebagai user sistem TI, misalnya Pak Budi atau Bu Wati dari proses check-in. Karena masing-masing individu tidak relevan dengan model, mereka diabstraksikan. Jadi para aktor disebut "karyawan check-in" (*check-in employee*) atau "penumpang" (*passenger*):

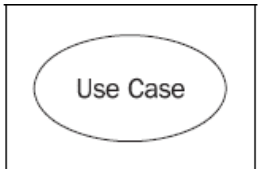


Aktor mewakili peran yang diambil user saat mereka menggunakan sistem TI, mis., Peran karyawan check-in. Satu orang dapat bertindak dalam lebih dari satu peran terhadap sistem TI. Penting untuk sistem TI di mana tindakan orang berdasarkan peran nya. Karena itu, perlu dilakukan analisa mendalam dalam sistem TI untuk peran tertentu, misalnya, sebagai user normal atau sebagai administrator. Dalam setiap skenario/case, akses ke fungsi sistem diberikan hak akses yang sesuai dengan perannya.

Aktor bukan bagian dari sistem TI. Namun, sebagai karyawan mereka dapat menjadi bagian dari sistem bisnis (lihat Gambar 11.5).

Use Case

Use cases menggambarkan interaksi yang terjadi antara pelaku dan sistem TI selama proses bisnis dilaksanakan:



Case use merupakan bagian dari fungsionalitas sistem TI dan memungkinkan user (aktor) mengakses fungsionalitas ini.

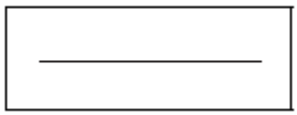
Apa pun yang ingin dilakukan user dengan sistem TI harus tersedia sebagai usecase (atau bagian dari use case). Fungsi-fungsi yang ada dalam sistem TI, tetapi tidak diakses melalui use case, tidak tersedia untuk user.

Meskipun use case adalah untuk menggambarkan interaksi, aliran pemrosesan batch (ditumpuk/ditunda), pada umumnya tidak masuk interaksi, juga dapat digambarkan

sebagai use case. Aktor dari use case batch tersebut kemudian adalah orang yang memulai pemrosesan batch.
Misalnya, membuat **statistik check-in** misalnya dapat menjadi use case batch.

Asosiasi

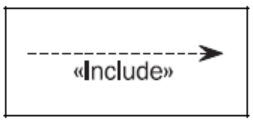
Asosiasi adalah koneksi antara aktor dan use case. Sebuah asosiasi menunjukkan bahwa seorang aktor dapat melakukan use case. Beberapa aktor pada satu use case berarti bahwa masing-masing aktor dapat melakukan use case:



Menurut UML, asosiasi hanya berarti bahwa seorang aktor terlibat dalam use case. Beberapa aktor dapat menjalankan use case yang sama dan bukan berarti use case tersebut harus dijalankan bersama-sama user lain.

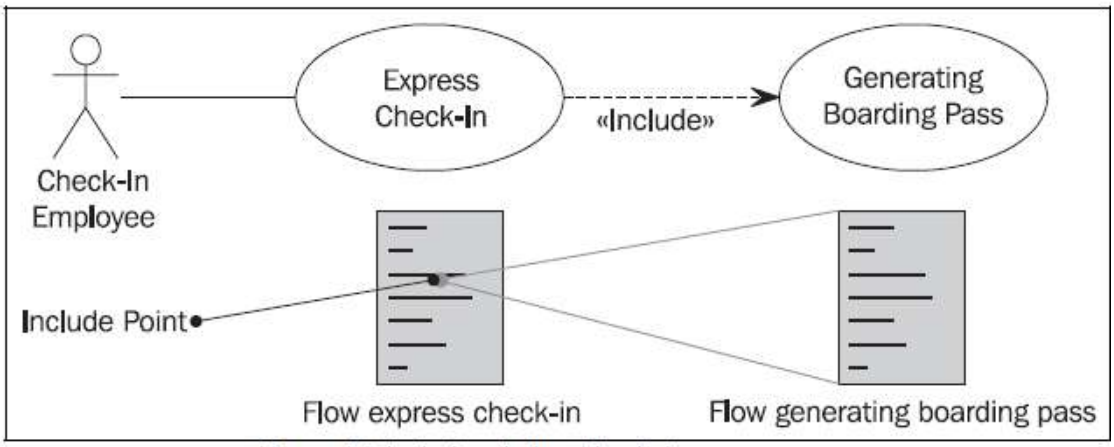
Include Relationship

Include relationship adalah hubungan antara dua use case:



Ini menunjukkan bahwa use case ke arah panah termasuk dalam use case di sisi lain panah. Ini memungkinkan untuk menggunakan kembali (reuse) use case untuk use case lain.

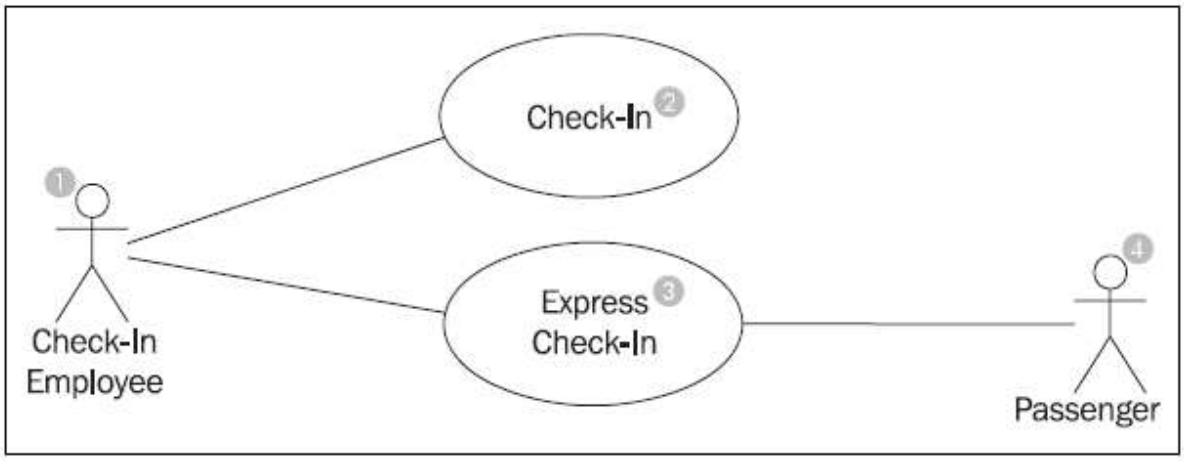
Gambar 67vmenunjukkan contoh hubungan ini. Dalam aliran use case, check-in adalah titik di mana case "generating boarding pass" di inculude kan. Ini berarti bahwa pada titik ini seluruh proses generatiing boarding pass dilakukan. Termasuk hubungan dilihat sebagai jenis panggilan ke subprogram:



Gambar 67. Include Relationship Antar Use Case

1.29. MEMBACA USE CASE DIAGRAM

Gambar 68 menunjukkan diagram use case dengan aktor (karyawan dan penumpang) serta use case check-in dan "express check-in":



Gambar 68. Diagram Use Case Sederhana

Tergantung anda, ingin membaca mulai dari diagram use case dengan aktor atau dengan use case.

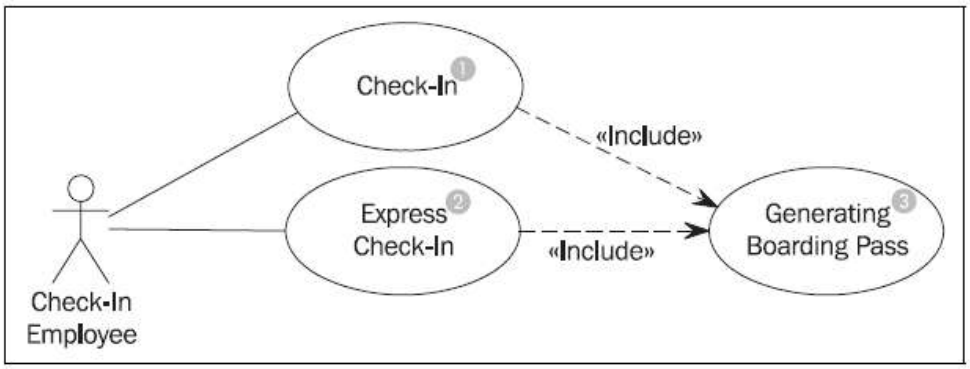
Dimulai dengan aktor **karyawan check-in (check-in employee)** (1) Anda dapat menemukan hubungan antara dua use **case check-in** (2) dan **express check-in** (3). Ini berarti bahwa orang yang berinteraksi dengan sistem TI sebagai check-in employee dapat melakukan use case "check-in" dan "express check-in".

Agar diagram mudah dibaca, sebaiknya case terletak satu di bawah yang lain. Namun, ini tidak berpengaruh yang penting dalam meletakkan user case adalah mudah dibaca dan dapat didokumentasikan dengan baik.

Use case check-in (2) dan express check-in (3) adalah sesuatu yang dapat dilakukan karyawan check-in dengan sistem TI.

Aktor penumpang (5) memiliki hubungan dengan use case express check-in (3), yang berarti bahwa orang yang berinteraksi dengan sistem IT sebagai penumpang dapat melakukan use case express check-in (3) langsung dengan IT sistem. Pegawai check-in aktor (1) juga memiliki hubungan dengan use case express check-in (3), berarti penumpang dan karyawan check-in dapat melakukan use case ini. Tidak berarti bahwa keduanya bekerja bersama selama express check-in.

Saat skenario use case check-in (2), penumpang melakukan check-in sendiri dan bukan karyawan yang melakukan, tetapi aktor dari sistem IT selalu menjadi orang yang secara langsung berinteraksi dengan sistem IT. Untuk use case express check-in (3) ini bisa saja penumpang dengan tiket pesawat yang sudah ada dapat memperoleh boarding pass melalu mesin, atau pegawai check-in yang membantu untuk menggantikan penumpang. Tetapi, untuk sistem bisnis penumpang selalu menjadi aktor, karena dia berada di luar bisnis. Karyawan dalam hal ini, bukan aktor dari perspektif sistem bisnis, karena dia bekerja di dalam sistem bisnis.



Gambar 69. Diagram Use Case Dengan Menyertakan Hubungan

Gambar 69 menunjukkan diagram use case dengan hubungan include yang dimiliki oleh use case check-in (1) dan express check-in (2) dengan use case generating boarding pass (3). Ini berarti bahwa selama check-in dan express check-in, boarding pass dicetak. Ini adalah cara termudah untuk menggunakan kembali (*reuse*) bagian (part) dari use case.

1.30. QUERY EVENT (PERMINTAAN) DAN MUTATION EVENTS (PERPINDAHAN)

Suatu event/kejadian di UML adalah spesifikasi dari **skenario**: deskripsi tentang sesuatu yang terjadi. Dalam konteks use case, event adalah sesuatu yang dilakukan user dengan sistem TI. Event di-trigger/dipicu oleh user melalui interface user, misalnya, dengan mengklik tombol Cari atau tekan tombol Enter. Ini memiliki akibat dalam sistem TI yaitu **ada sesuatu yang diproses**.

Terdapat 2 jenis event yaitu :

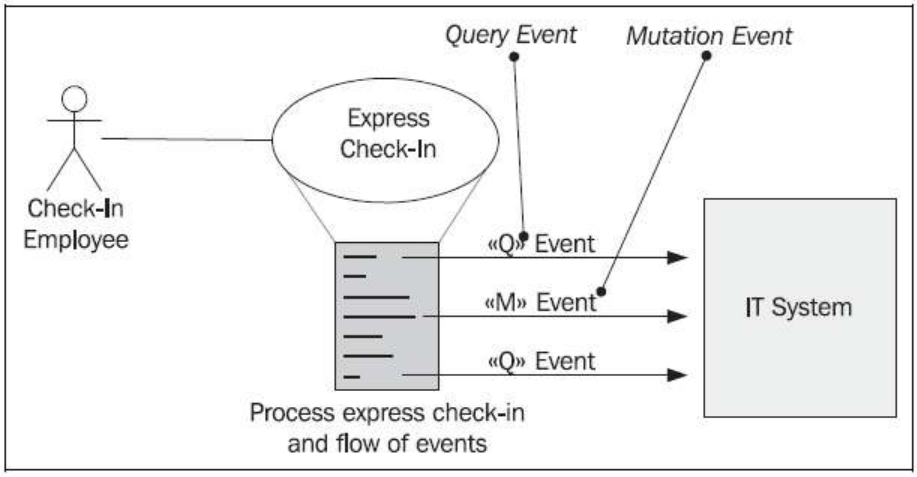
- **Query events** adalah event yang memiliki tujuan **menampilkan informasi dan biasanya tidak mengubah** apa pun dalam sistem TI. Query event biasanya berupa tampilan informasi.
- **Mutation Event** adalah event yang bertujuan **menyimpan, mengubah, atau menghapus informasi** dalam sistem TI. Hasil dari mutation events tergantung pada suksesnya mutasi: informasi telah berhasil disimpan, dimodifikasi, atau dihapus, yang harus disampaikan kepada user; misal kegagalan perubahan.

UML tidak membedakan antara query events dan mutation events, tetapi kita bisa memberikan stereotype UML dengan elemen khusus. Contoh :

- Stereotype **«Q»** di depan nama event menunjukkan kasus khusus *query event*.
- Stereotype **«M»** di depan nama event menandakan case khusus dari *event mutation*.

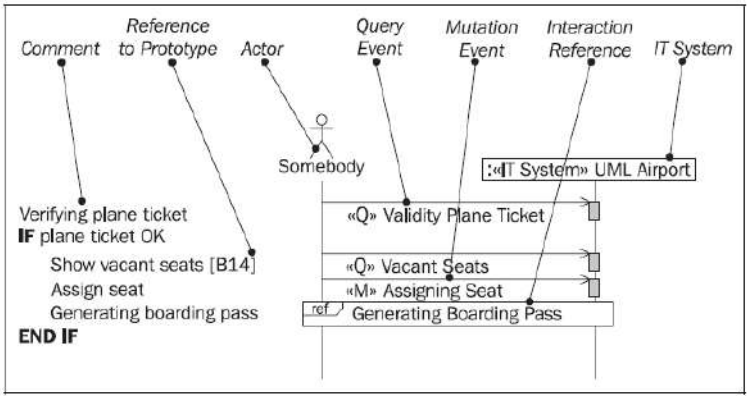
Dengan cara ini, kita dapat mendeskripsikan event dalam tipe diagram yang berbeda, dan membuat jelas jenis event yang kita kerjakan.

Gambar 11.12 menunjukkan query event dan mutation events dalam use case. Respons sistem TI terhadap event tersebut tidak digambarkan sebagai event terpisah. Setiap query dan mutation events memiliki **feedback** (hasil) secara **implisit**: informasi yang diperoleh, atau pesan berhasil atau gagal:



Gambar 70. Query Event «Q» dan Mutation Event «M»

USE CASE DIAGRAM / SYSTEM SEQUENCE DIAGRAM

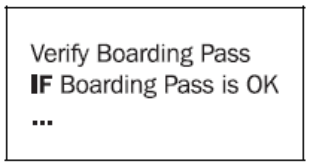


Gambar 71. Elemen Sistem Sequence Diagram (Use Case Sequence Diagram)

Gambar 71 Sistem Sequence diagram UML (lihat Gambar 71). Akan dibahas sequence diagram secara rinci di Bagian *Interaction View*. Dalam sistem sequence diagram, berikut adalah elemen:

-Komentar (*Comment*)

Alur use case dijelaskan dalam kombinasi deskripsi tekstual:



Komentar menggambarkan aliran use case dengan cara yang sederhana; UML pada umumnya memungkinkan menempatkan komentar di seluruh diagram.

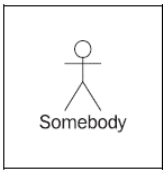
-Referensi to Prototype

Referensi berasal dari form di layar, list dan elemen lain pada user interface dijelaskan dengan comment :

Ini yang membentuk hubungan (link) antara use case sequence diagram dan prototype.

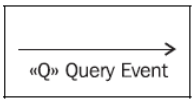
-Actor "Somebody"

Actor "somebody" merepresentasikan aktor mana saja di use case diagram. "Somebody" adalah sumber awal terjadinya event melalui use case menuju sistem TI :



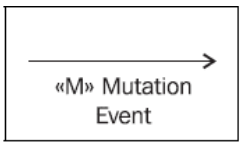
Karena use case dapat terdiri dari bermacam aktor, maka pada contoh diatas dituliskan "somebody".

-Query Event



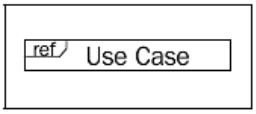
Query event adalah event yang dikirim ke sistem TI dengan tujuan untuk membaca informasi (lihat sub bab, *Query Events* dan *Perpindaan(Mutation) Events*).

-Mutation Event



Mutation events adalah event yang dikirim ke sistem TI dengan tujuan mengubah informasi (lihat sub bab, Query events dan Mutation events).

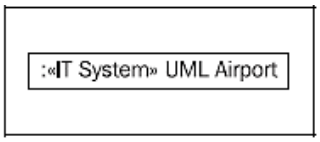
-Referensi Interaksi *(Interaction Reference*)



Referensi interaksi menyebutkan referensi use case sequence diagram lain (lihat penjelasan hubungan include di Bagian Use Case Diagram).

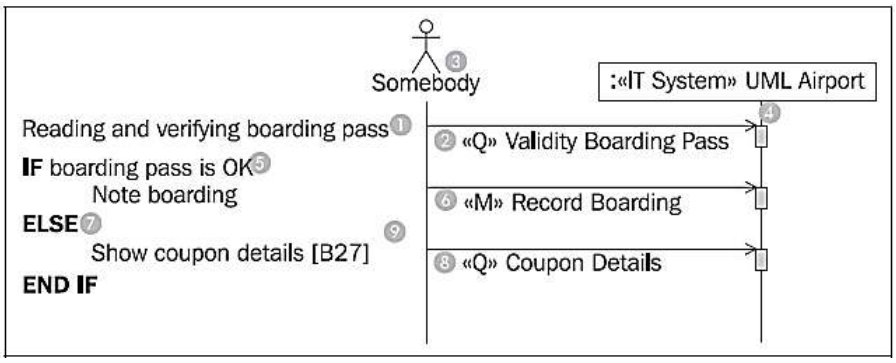
-Sistem IT

Sistem TI mewakili kotak hitam dengan semua objeknya dan seluruh fungsinya:



Semua event dalam use case menuju ke sistem IT. Dalam external view, kita tidak mempedulikan objek individu mana dalam sistem TI yang dipengaruhi oleh event tersebut.

1.31. MEMBACA SISTEM SEQUENCE DIAGRAM/USE CASE SEQUENCE DIAGRAM



Gambar 72. Diagram Use Case Sequence Diagram

Gambar 72 menunjukkan sequence diagram dari use case **boarding**. Diagram sequence berasal dari use case, menggambarkan aliran interaksi use case. Aliran dimulai dari komentar di kiri. Pertama, boarding pass dibaca dan diverifikasi (1) dengan mengirimkan query event **«Q»validity boarding pass** (2) dari aktor use case (3) ke sistem IT (4). Bagaimana event diperlakukan secara internal tidak dapat dilihat dalam diagram ini, karena sistem TI dipandang sebagai kotak hitam. Validity boarding pass diverifikasi, artinya informasi mengenai boarding pass yang dibaca ke dalam sistem TI dibandingkan dengan informasi yang tersimpan. Berdasarkan informasi yang dikembalikan oleh sistem TI, user dapat melihat apakah verifikasi berhasil atau tidak. Hasilnya tidak ada panah, melainkan mengalir kembali melalui panah query (2).

Jika boarding pass OK (5), maka dicatat dalam sistem IT penumpang telah naik ke pesawat. Ini terjadi melalui event mutasi **«M» record boarding** (6), yang dikirim ke sistem IT (4). Sekali lagi, kita tidak dapat melihat apa yang terjadi dalam sistem TI. Jika boarding pass tidak OK (7), layar akan menampilkan informasi tentang tiket milik boarding pass ini. Untuk ini, sekali lagi, event kupon **«Q» coupon detil** (8) /perincian kupon, dikirim ke sistem TI. Query ini menghasilkan informasi yang diinginkan, selama data ada di dalam sistem TI. Tampilan informasi dapat diilustrasikan dalam prototipe interface. Referensi [B27] (9) sesuai dengan bentuk layar di prototipe interface. Sehingga, seluruh case boarding dijelaskan sebagai urutan query dan event mutasi.

MEMBANGUN EXTERNAL VIEW

Berikut adalah langkah-langkah yang diperlukan untuk membangun external view :

Checklist 4.1. Menyusun Diagram pada External View:

- Kumpulkan sumber informasi — Bagaimana saya bisa mengetahuinya?
- Identifikasi aktor potensial — Siapa yang bekerja dengan sistem TI?
- Identifikasi business use case potensial — Apa yang dapat dilakukan dengan sistem TI?
- Hubungkan business use case— Siapa yang dapat melakukan apa dengan sistem TI?
- Jelaskan aktor — Siapa atau apa yang diwakili aktor?
- Cari lebih banyak business use case— Fungsi lain apa yang harus disiapkan oleh sistem TI?
- Ubah business use case— Apa yang seharusnya dimasukkan ke dalam business use case?
- Dokumentasikan business use case — Apa yang terjadi dalam business use case?
- Modelkan hubungan antara business use case — Use case apa yang dapat digunakan kembali (reusable)?
- Verifikasi view — Apakah sudah benar semua?

Langkah diatas adalah langkah ideal, tetapi mungkin pada prakteknya, langkah-langkah tsb sering tumpang tindih.

Langkah-langkah ini biasanya dilakukan oleh seorang analis, dimana dia perlu memiliki pemahaman umum tentang sistem TI serta sistem bisnis seperti yang dimodelkan dalam Bab Pemodelan Sistem Bisnis. Namun, konsultasikan dengan orang yang memiliki pengetahuan tentang sistem tersebut, seperti user sistem. Hasil dari langkah kerja ini adalah external view yang harus dibaca dan dipahami oleh orang ahli di bidang tersebut.

KUMPULKAN SUMBER INFORMASI—BAGAIMANA SAYA MENGETAHUI?

Secara umum, cukup banyak sumber informasi yang dapat digunakan untuk merumuskan external view:

- Tentu saja, model sistem bisnis adalah sumber pertama yang paling penting. Aktor, karyawan, dan use case sistem bisnis itu merupakan titik awal untuk mendapatkan aktor dan use case dari sebuah sistem TI (lihat Bab Memodelkan Sistem Bisnis).
- Spesifikasi teknis, spesifikasi proyek, dan dokumen terkait.
- User yang nanti akan menggunakan sistem adalah sumber sangat penting untuk external dengan melihat cara pandangnya.
- Ahli teknis di bidang aplikasi sistem TI.
- Struktur dan bagan organisasi serta uraian tugas.

Melihat sudut pandang user sangat membantu. Bicara dengan user atau amati mereka melakukan pekerjaanya.

IDENTIFIKASI AKTOR POTENSIAL— SIAPA YANG BEKERJA DENGAN SISTEM TI?

Langkah bagaimana kita mengidentifikasi pemilihan aktor di awal. Pilihan ini tidak harus lengkap atau benar. Aturannya: semakin banyak, semakin baik. Teruslah bekerja dengan aktor-aktor ini untuk langkah-langkah selanjutnya.

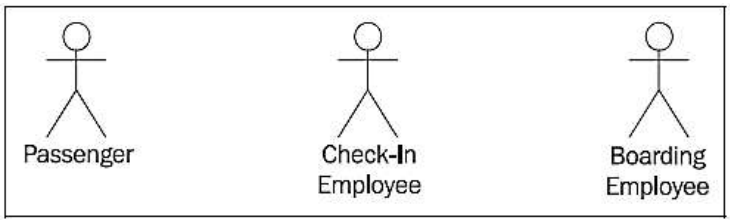
Menjawab pertanyaan-pertanyaan dibawh (misalnya, dengan user sistem) akan membantu mengidentifikasi pelaku potensial. Saat melakukanya , sebaiknya tidak menggunakan nama orang, usahakan untuk membentuk kelompok orang atau aktor:

- Aktor dan karyawan mana dari sistem bisnis yang berhubungan langsung dengan sistem TI?
- Kelompok orang mana yang memerlukan dukungan dari sistem TI untuk menyelesaikan pekerjaan sehari-hari mereka?
- Kelompok orang mana yang melakukan fungsi utama dan paling penting dari sistem TI?
- Kelompok orang mana yang diperlukan untuk menjalankan fungsi sistem pendukung, seperti pemeliharaan dan administrasi?
- Kelompok orang mana dari model organisasi (lihat bagan organisasi) perusahaan atau divisi yang bekerja dengan sistem TI?

Penting untuk menggambarkan interaksi yang nyata dan berhubungan langsung dengan sistem.

Di counter check-in, check-in employee yang berinteraksi langsung dengan sistem. Pada mesin check-in, penumpang tanpa bagasi dapat check-in dengan tiket yang dapat dibaca mesin, adalah penumpang yang secara langsung berinteraksi dengan sistem TI, memasukkan tiket dan memilih kursi yang kosong.

Seorang penumpang adalah aktor dalam sistem bisnis; tetapi check-in employee dan "boarding" employee tidak termasuk. Mereka adalah karyawan yang termasuk bagian dari sistem bisnis, seperti yang ditunjukkan pada Gambar 73:



Gambar 73. Aktor Potensial

IDENTIFIKASI USE CASE POTENSIAL— APA YANG DAPAT DILAKUKAN DENGAN SISTEM TI?

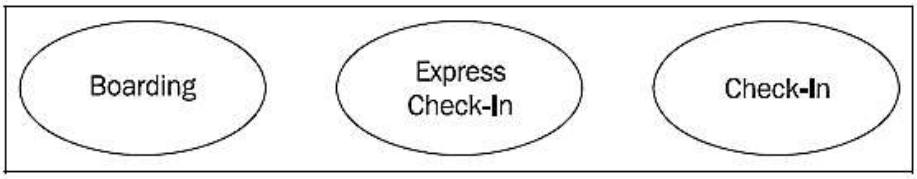
Langkah ini adalah bagaimana kita menemukan dan memilih use case di awal. Ingat aturan: semakin temukan banyak, semakin baik. Pertanyaan-pertanyaan berikut membantu mengidentifikasi use case potensial:

- Use case bisnis dari sistem bisnis mana yang didukung oleh sistem TI?
- Kegiatan bisnis mana dari sistem bisnis yang harus didukung oleh sistem TI?

Bergantung pada tingkat rincian kegiatan bisnis, pada langkah ini use case dapat dibangun untuk setiap kegiatan bisnis.

- Apa tujuan dan tujuan sistem TI?
- Apa fungsi utama sistem TI?
- Untuk apa pelaku membutuhkan sistem TI?
- Fungsi sistem sekunder mana, seperti pemeliharaan dan administrasi, yang dibutuhkan sistem TI?

- Apa fungsi yang dimiliki prototipe interface?

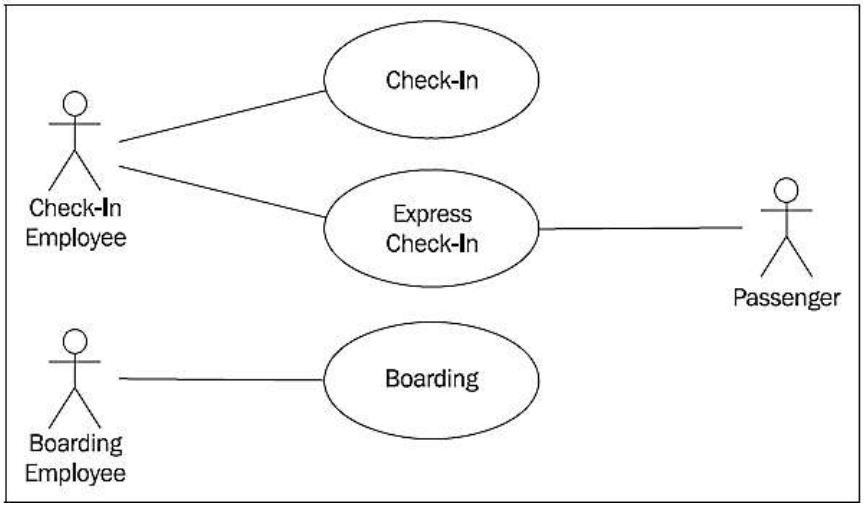


Gambar 74. Use Case Potensial

MENGHUBUNGKAN AKTOR DAN USE CASE — SIAPA ? DAPAT MELAKUKAN APA? PADA SISTEM TI

Menugaskan use case bisnis ke aktor menghasilkan rancangan awal dari use case diagram, seperti yang ditunjukkan pada Gambar 74. Di sini, pertanyaan berikut harus dijawab:

- Aktor mana yang dapat melakukan use case yang mana?



Gambar 75. Draf Awal Use Cse Diagram

Draf pertama ini merupakan dasar dari mana diagram use case dapat diedit dan disempurnakan, seperti yang ditunjukkan pada Gambar 75.

DESKRIPSIKAN AKTOR — SIAPA ATAU APA YANG DIWAKILI OLEH AKTOR?

Seorang aktor dalam diagram harus dinamai (bisa diganti namanya) dengan cara menjelaskan peran yang ada dan dapat direpresentasikan.

Pertanyaan :

- Bagaimana seorang aktor digambarkan secara akurat? Hal ini penting untuk kita identifkasi batasan (*domain terminologi*) . User sistem TI harus mengenali diri mereka sendiri sebagai aktor; jika tidak mereka tidak akan mengerti membaca use case diagram!. Jika perlu, aktor didefinisikan dengan komentar di samping nama aktor secara detil. Komentar itu dapat mencakup bidang tanggung jawab aktor, persyaratan sistem TI dari perspektif aktor, atau definisi formal tentang peran yang dimainkan aktor.

CARI USE CASE LEBIH BANYAK—FUNGSI TI APA YANG HARUS DISEDIAKAN OLEH SISTEM?

Setelah mengidentifikasi pilihan use case awal sebagai titik awal penyelesaian diagram use case. Untuk mencari use case lain yang mungkin tidak terpikirkan sebelumnya dapat diidentifikasi dengan mengajukan pertanyaan berikut, pada sebuah use case:

- Apakah ada sesuatu yang harus dilakukan sistem TI sebelum use case tersebut dapat dieksekusi?
- Apakah ada sesuatu yang harus dilakukan dengan sistem TI di setelah use case tersebut dijalankan?
- Apakah ada sesuatu yang harus dilakukan dengan sistem TI jika tidak menjalankan use case tersebut?

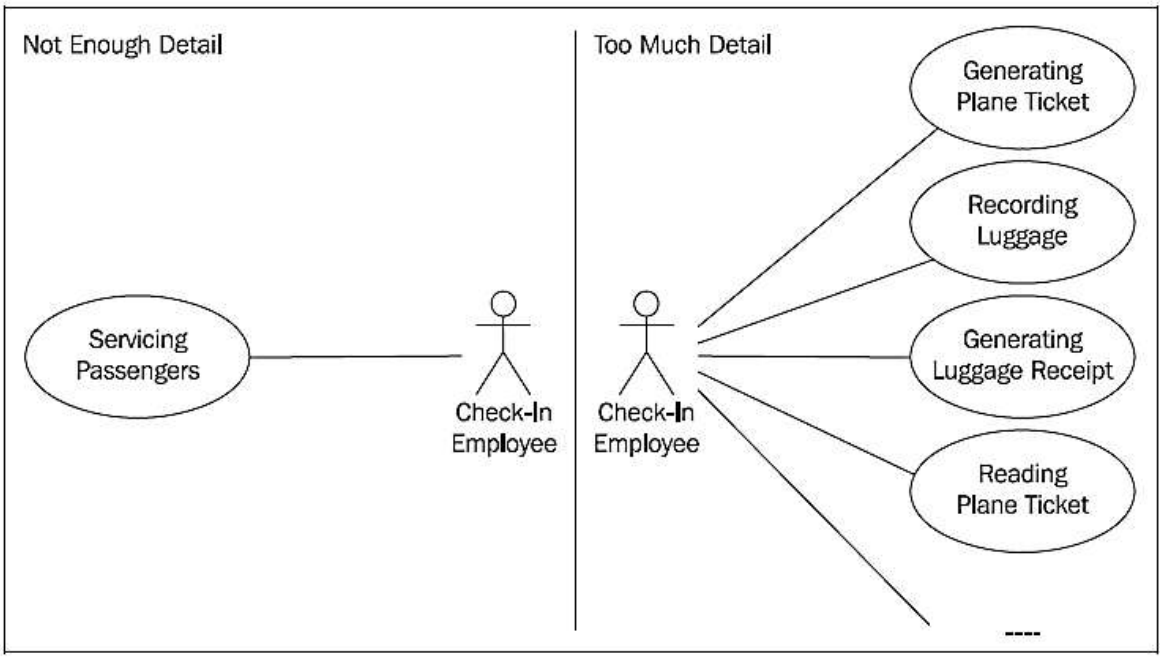
Sangat penting untuk tetap fokus pada sistem TI kita. Bisa saja ada risiko pemodelan use case yang ada di luar sistem TI yang sedang dibangun. Misalnya, saat membeli tiket pesawat, (kejadian sebelum check-in), kadang bukan milik sistem TI kita (misal membeli tiket peswat via traveloka dll).

Dalam studi kasus ini, ditentukan untuk use case check-in adalah:

- Informasi tentang tiket dan penerbangan harus diperoleh sebelum check-in.
- Boarding (naik pesawat) harus dilakukan setelah check-in.
- Tiket pesawat harus dibatalkan jika penumpang tidak muncul saat check-in.

PERBAIKI USE CASE — APA SEBENARNYA YANG HARUS DIMASUKKAN KE DALAM USE CASE?

Bagian tersulit dari pemodelan use case adalah menemukan tingkat detail yang sesuai untuk use case. Berikut adalah contoh ekstrem 2 use case diagram yang ditunjukkan pada Gambar 76:



Gambar 76. Contoh Perbandingan Ekstrem dalam Sitem TI "Passenger Services"

Memang tidak ada ketentuan yang pasti. Bayangkan jika seluruh sistem TI digambarkan ke dalam satu use case saja membuat diagram tidak bermakna dan tidak bermakna untuk dipelajari. Jika use case dirinci terlalu banyak, diagram menjadi terlalu rumit dan sehingga hubungan antar use case sulit untuk dipelajari.

Terdapat beberapa kriteria yang membantu Anda menemukan ruang lingkup use case yang optimal.

Untuk mencegah use case menjadi terlalu besar, anda dapat mengajukan pertanyaan berikut untuk tiap use case:

- Apakah tiap use case telah berperilaku interaksi secara berurutan dan terkait (urutan interaksi)? Misal mencetak boarding pass dan mencari barang hilang tidak ada hubungannya sehingga dipisah.
- Hanya satu aktor melakukan sebuah use case? Meskipun UML memungkinkan bisa lebih dari satu aktor terlibat dalam menjalankan use case, lebih baik tidak.

kita lakukan. Jika use case menggambarkan interaksi seseorang dengan komputer, itu berarti bahwa tidak lebih dari satu orang terlibat dalam interaksi. Use case yang tidak sesuai kriteria ini harus dibagi.

Untuk menghindari membuat use case yang terlalu kecil, kita dapat mengajukan pertanyaan berikut untuk tiap use case:

- Apakah use case memberikan hasil dan relevan? use case tsb misalnya tidak bisa menjelaskan sub-step yang lengkap dengan sendirinya, seperti memilih pelanggan "choose customer". Sebuah use case harus ada hasil. Use case/skenario yang tidak sesuai kriteria ini harus digabungkan dengan use case lain
- Apakah ada use case yang tidak pernah berjalan sendiri, tetapi selalu berurutan dikombinasikan dengan use case bisnis lain? Sebuah use case seharusnya tidak menjelaskan sub-step yang hanya dijalankan dalam kombinasi dengan subteps lainnya. Use case yang ini tidak sesuai dan harus digabungkan dengan use case lain.

DOKUMENTASI USE CASE — APA YANG TERJADI PADA SEBUAH USE CASE?

Informasi hanya dari diagram use case tidak cukup untuk memahami use case. Aliran interaksi di belakang use case harus dijelaskan. Sebuah deskripsi verbal, deskripsi dalam sistem sequence diagram telah terbukti memilki makna.

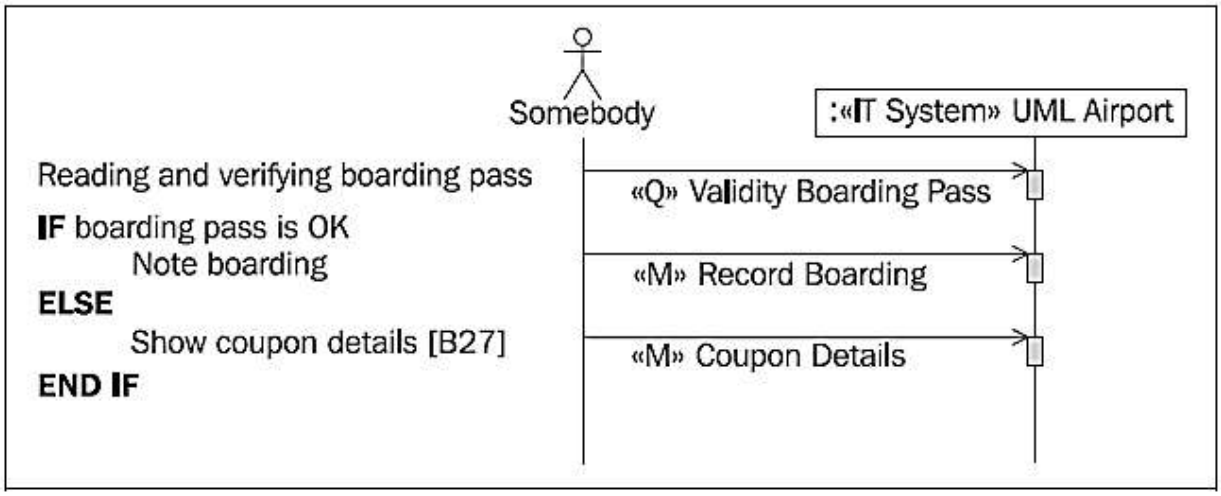
Berikut pertanyaan saat anda mengembangkan sistem sequence diagram untuk sebuah use case:

- Langkah-langkah apa yang bekerja di sistem TI? Untuk menjawab ini kita harus mengamati kerja aktor dengan sistem TI. Apa saja yang dilakukan aktor dengan sistem IT? Apa yang dimasukkan? Apa yang ditampilkan oleh sistem TI? Interaksimya seperti apa? Penting menemukan tingkat detail yang sesuai. Tidak setiap tombol yang ditekan akan membuat langkah kerja.

- Dua pertanyaan berikut akan membantu Anda menemukan tingkat detail yang sesuai.
    - Informasi apa pada sebuah use case yang diberikan ke aktor? Jika informasi harus ditampilkan, query permintaan dikirim ke sistem TI.
    - Informasi mana yang disimpan, diubah, atau dihapus dalam sistem TI? Jika informasi harus diubah, mutation event dikirim ke sistem TI.

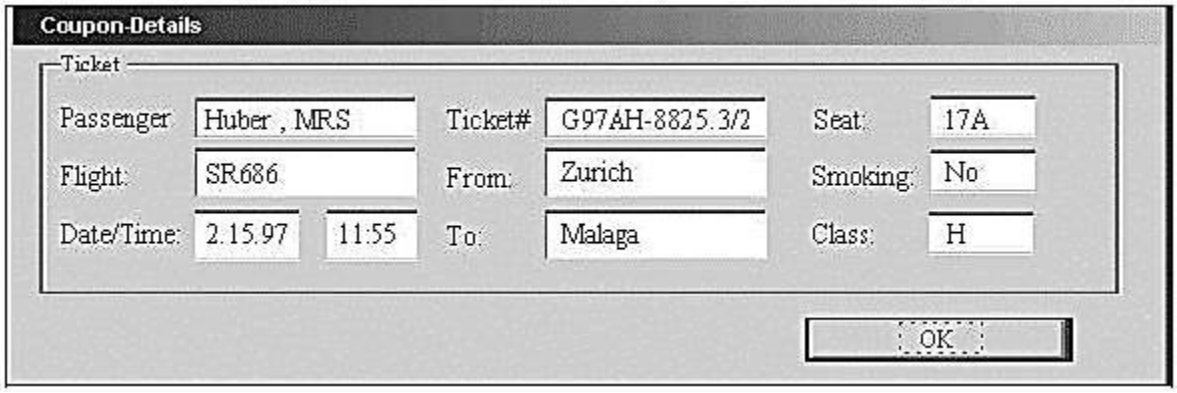
Oleh karena itu, deskripsi dari sebuah flow berisi langkah-langkah di mana informasi dimasukkan atau ditanyakan dan sistem TI dipandang sebagai kotak hitam.

Pada kenyataan, sebuah skenario tidak selalu berjalan sukses, gunakan struktur kontrol sederhana dalam deskripsi untuk menunjukkan alternatif dan cabang, seperti contoh dalam Gambar 77:



Gambar 77. Use Case Sequence Diagram untuk Use Case "Boarding"

Dokumentasi use case juga harus mencakup deskripsi interface user yang akan digunakan. Contoh berikut adalah jendela dialog label [B27], ditunjukkan pada Gambar 78:



Gambar 78. Jendela dialog [B 27] dari Use Case "Boarding"

MODEL HUBUNGAN ANTAR USE CASE — APA YANG DAPAT DIGUNAKAN KEMBALI (REUSABLE)?

Perhatikan terdapat beberapa bagian interaksi yang sama dalam beberapa use case, kesamaan ini dapat dimasukkan ke dalam use case itu. Include relationship pada use case baru dapat digunakan yang terdapat pada use case lain.

VERIFIKASI VIEW — APAKAH SEMUANYA BENAR?

Diagram use case, atau use case sequence diagram, harus diverifikasi dengan user yang memiliki pengetahuan. Idealnya, user dapat membaca dan memahami diagram (tidak terlalu sulit, karena memiliki instruksi untuk mudah dibaca). Kemudian, user diposisikan untuk menjawab pertanyaan tentang kelengkapan dan kebenarannya. Jika ini memungkinkan, diagram harus dibacakan ke user, diverifikasi kebenaran dan kelengkapan. Langkah terakhir ini adalah melengkapi tahap keseluruhan dan tampilan yang terverifikasi mencerminkan pemahaman bersama tentang sistem TI yang akan dibangun.

Diagram use case case yang lengkap dapat diverifikasi dengan daftar periksa berikut:

Checklist Verifikasi Use Case Diagram di External View:

- Apakah diagram use case sudah lengkap? Diagram use case lengkap jika tidak ada use case lebih lanjut. Apa pun yang harus dilakukan user dengan sistem TI digambarkan dalam bentuk use case (bila perlu, use case dibagi dalam beberapa bagian).
- Apakah semua use case benar? Use case dikatakan benar jika menggambarkan use case dari sistem IT dan memenuhi definisi dari use case.
- Apakah tingkat kerinciannya sesuai? Tingkat rincian use case bisnis harus memenuhi persyaratan berikut:
- Sebuah use case mewakili urutan interaksi yang koheren secara perilaku.
- Use case dilakukan oleh aktor tunggal.
- Sebuah use case mewakili suatu fungsionalitas yang nyata dan yang menghasilkan sebuah hasil yang relevan.
- Umumnya, sebua use case berjalan lengkap.
- Apakah penamaan use case sesuai? Nama tiap use case harus menggambarkan aktivitas yang dieksekusi dalam sistem TI.
- Apakah aktor benar? Aktor dalam diagram use case mewakili peran yang dilakukan seseorang (mis., Seseorang) atau sesuatu (mis., Sistem lain) dalam interaksi dengan sistem TI.

Sebuah Use Case Sequence Diagram lengkap dapat diverifikasi dengan pertanyaan berikut :

Checklist Verifikasi Use Sequence Diagram di External View:

- Apakah use case sequence diagram sudah digambarkan lengkap? Setiap use case harus ada sequence diagram dan kemungkinan aliran use case.
- Apakah sequence diagram sudah benar? Sequence diagram harus hanya berisi satu objek mewakili sistem TI dan dibuat khusus dari event query(permintaan) dan event mutasi (perubahan).

KESIMPULAN

SOAL LATIHAN

UNIVERSITAS BUDI LUHUR
FAKULTAS TEKNOLOGI INFORMASI

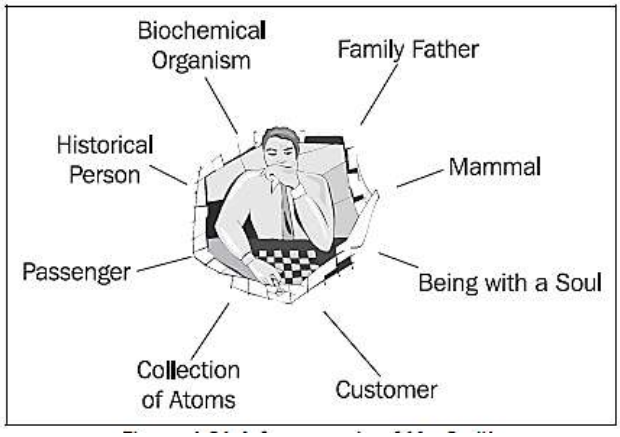
MODUL PERKULIAHAN #10
MODELING IT SYSTEM: STRUCTURAL VIEW

Capaian Pembelajaran	:	Mahasiswa memahami dan dapat menggunakan Modeling IT system structural view
Sub Pokok Bahasan	:	1.45. Class dan Objek 1.46. Generalisasi, Spesialisasi, dan Pewarisan 1.47. Elemen dari External View 1.48. Aturan Bisnis Statis dan Dinamis 1.49. Elemen dari View a. Membaca Class Diagram b. Membangun class diagram 1.50. Kesimpulan 1.51. Latihan Soal
Daftar Pustaka	:	Grässle, P., Baumann, H. and Baumann, P., 2005. UML 2. 0 in Action: A Project-based Tutorial. Packt Publishing Ltd.

CLASS DAN OBJEK

Dasar dari pendekatan berorientasi objek adalah sebaik mungkin sesuatu yang ada di dunia nyata direpresentasikan dalam suatu model dan nantinya diterapkan dalam sistem TI.

Namun, representasi ini tidak pernah sepenuhnya sesuai dengan kenyataan. Segala sesuatu di dunia nyata, apakah itu makhluk hidup, objek, atau ide, sangat kompleks dan memiliki banyak aspek, sehingga kompleksitas ini tidak pernah dapat sepenuhnya dapat diwakili



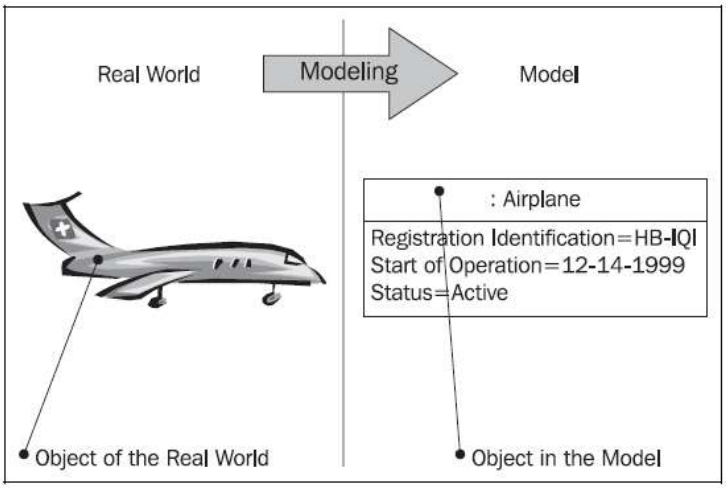
Gambar 79. Beberapa Aspek dari Pak Budi

Untuk memungkinkan representasi sebagai model, kita perlu fokus pada beberapa aspek tertentu dan mengabaikan semua aspek lainnya. Aspek esensial yang penting diambil dan ditekankan dan aspek lain dihilangkan. Inilah seni dalam pemodelan objek.al

Untuk memodelkan objek dengan baik, kita harus tahu untuk tujuan apa mereka dibutuhkan dalam sistem TI. Objek "Pak Budi" akan terlihat berbeda dalam sistem manajemen pelanggan dibandingkan dengan sistem informasi rumah sakit atau dalam daftar pajak (lihat Gambar 79). Ketika kita mengetahuinya atau memperkirakan, tujuan dari sistem IT, maka kita dapat membangun objek fungsional.

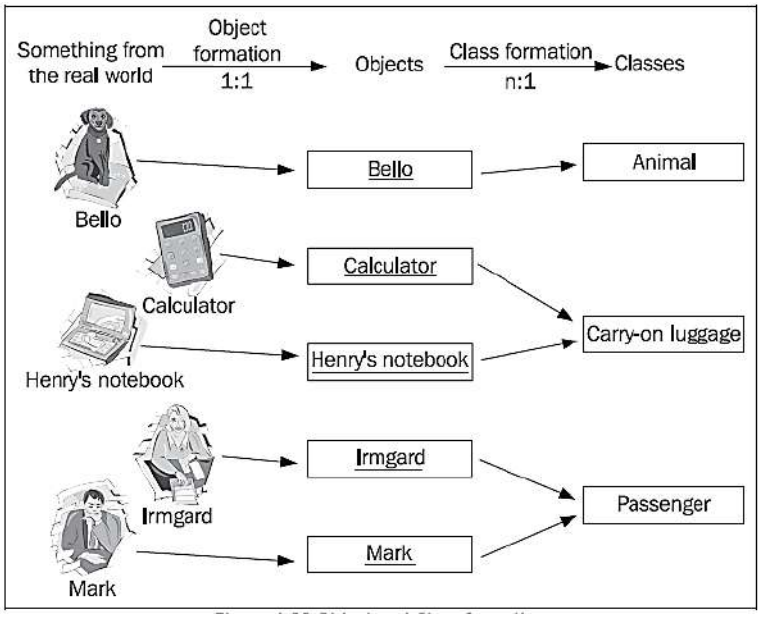
Dalam model selalu dibuat abstrak dengan berorientasi pada target. Dibatasi pada pertimbangan aspek-aspek penting untuk tujuan tertentu dan menghilangkan yang lain.

Gambar 80 menunjukkan langkah abstraksi ini dengan contoh pesawat terbang:



Gambar 80. Pemodelan

Saat menggambarkan dunia nyata dalam model abstrak, dibedakan dua langkah. Langkah pertama, kita mengabstraksi dari individu atau sesuatu menjadi benda. Pada langkah kedua, menggabungkan objek serupa ke dalam kelas. Gambar 81 menunjukkan, dengan beberapa contoh **"sesuatu"** / "things" dari dunia nyata digambarkan pertama sebagai objek dan kemudian sebagai kelas:



Gambar 81. Formasi Object dan Class

Ilustrasi model **sesuatu** yang ada di dunia nyata mengarah ke suatu objek. Hubungan 1: 1 terjadi antara sesuatu dari dunia nyata dan objek.

Objek mewakili satu contoh tertentu dari dunia nyata. Dalam database, objek memiliki hubungan, misalnya, dengan entri spreadsheet. Definisi sebuah objek merupakan langkah pertama abstraksi, karena hanya fitur yang relevan dimodelkan dalam objek. Misalnya, dalam objek Mark, Mark sebagai orang dikurangi menjadi **aspek-aspek yang penting sebagai penumpang**, misalnya: nama depan, nama belakang, dan tanggal lahir.

Tuliskan 12 object dari lingkungan pribadi anda!

Pada langkah kedua abstraksi, kita menggabungkan objek serupa ke dalam kelas. Serupa artinya:

- Tujuan dari abstraksi adalah serupa
- Karakteristik serupa
- Objek memiliki perilaku yang sama

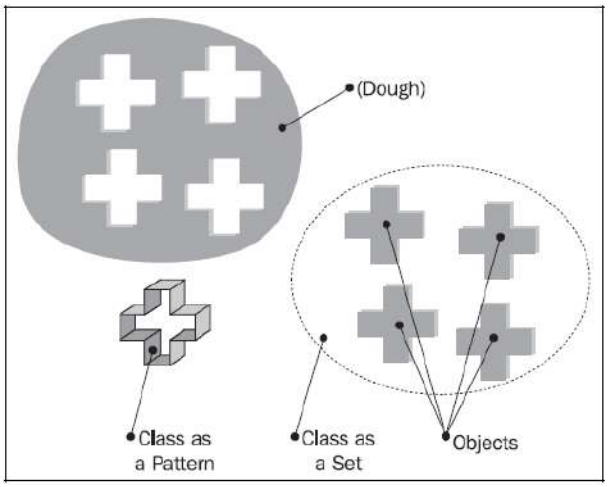
Pemodelan yang kadang sulit adalah bila memodelakan sesuatu yang tidak fisik yaitu berupa konsep atau ide.

Kelompokkan objek yang anda tulis tadi ke dalam class.

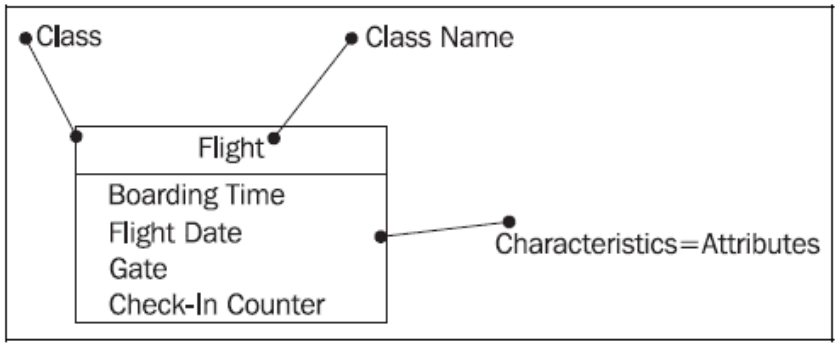
Membuat kelas menjadi lebih mudah ketika Anda kelas memiliki dua makna yang berbeda:

- Kelas adalah pola sama dari objek yang dibentuk.
- Di sisi lain, kelas adalah himpunan objek sesuai dengan kelas yang dibentuk.

Kelas sebagai pola menentukan karakteristik dan perilaku objek yang dibuat dari kelas. Pada Gambar 82, kelas diibararatkan cetakan kue (*cookie cutter*), yang digunakan untuk memotong kue (objek) dari sebuah adonan (*dough*):



Gambar 82. Cookies, Class dan Object



Gambar 83. Class dari sebuah Pola

Kelas sebagai satu set berisi dan mengetahui semua objek didalamnya. Dapat digambarkan sebagai tabel dalam database.

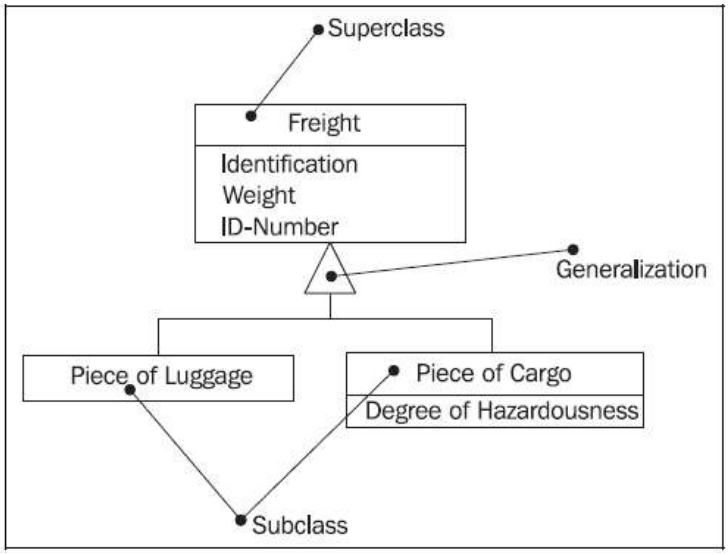
Biasanya, kelas, selain atribut, berisi metode (*method*), yang menentukan perilaku objek. Namun, dalam pembhasan pemodelan sistem TI ini, kita tidak detil membahasnya. Perilaku sangat tergantung pada keadaan (*state*) masing-masing objek. Misalnya, method "cancel flight" dari kelas "flight" akan melakukan sesuatu yang berbeda antara objek flight kondisi "in execution" dan objek flight dalam kondisi "planned".

Aturan seperti itu umumnya akan lebih mudah dimodelkan dalam diagram statechart dalam behaviour view dibandingkan model operasi.

Pada tahap desain dan implementasi, **perilaku** kelas dikonversi menjadi **metode** sesuai dengan bahasa pemrograman yang digunakan.

GENERALISASI, SPESIALISASI, DAN PEWARISAN

Istilah-istilah superclass, subclass, atau inheritance muncul dalam pikiran ketika memikirkan pendekatan berorientasi objek. Konsep-konsep ini sangat penting ketika berhadapan dengan bahasa pemrograman berorientasi objek seperti Java, Smalltalk, atau C ++. Untuk memodelkan class yang menggambarkan konsep teknis, biasanya bentuk ini digunakan sebagai dukungan. Istilah-istilah ini dapat digambarkan pada Gambar 84:



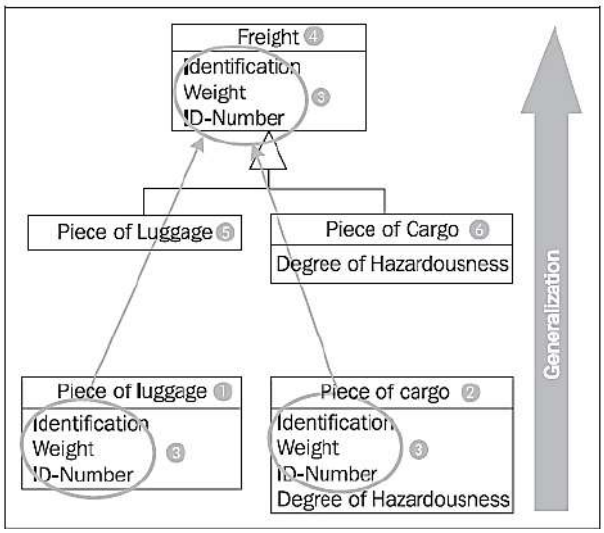
Gambar 84. Notasi Generalisasi

Generalisasi adalah proses mengekstraksi karakteristik yang sama dari dua kelas atau lebih, dan menggabungkannya menjadi superclass. Karakteristik bersama dapat berupa atribut, asosiasi, atau metode.

Pada Gambar 84, kelas **Piece of Luggage** dan **Piece of Cargo** secara parsial berbagi atribut yang sama. Dari perspektif domain, kedua kelas ini juga sangat mirip.

Secara umum (generalisasi), karakteristik bersama digabungkan dan digunakan untuk membuat superclass baru **Freight**. **Piece of Luggage** dan **Piece of Cargo** menjadi subclass dari kelas Freight.

Atribut bersama (3) hanya ada di superclass, berlaku hanya untuk dua subclass, meskipun attribut tidak ada di subclass.

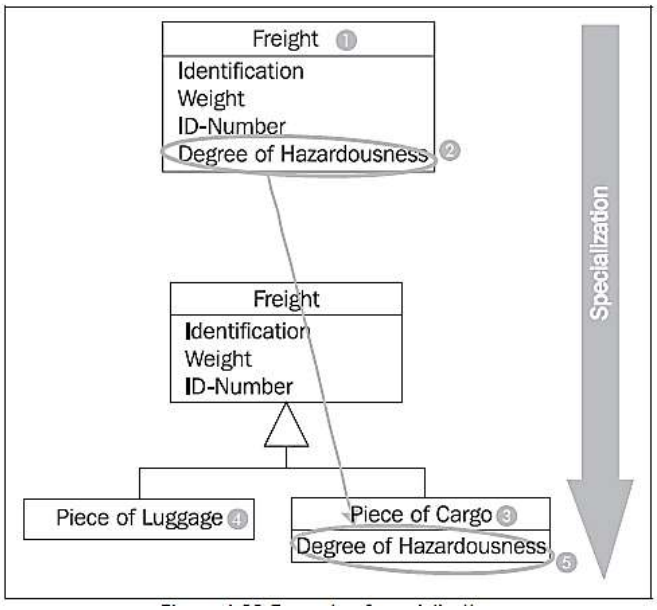


Gambar 85. Contoh Sebuah Generalisasi

Coba lihat, apakah kelas yang anda buat sebelumnya dapat digeneralisasikan?

Berbeda dengan generalisasi**, spesialisasi** berarti membuat subclass baru dari kelas yang ada. Jika ternyata atribut, asosiasi, atau metode tertentu hanya berlaku pada beberapa objek kelas, dapat dibuat subclass. Kelas paling inklusif dalam sebuah generalisasi / spesialisasi disebut superclass dan umumnya terletak di bagian atas diagram. Kelas yang lebih spesifik disebut subclass dan umumnya ditempatkan di bawah superclass.

Pada Gambar 86, kelas **Freight** (1) memiliki atribut **Degree of Hazardousness** (2), yang dibutuhkan hanya untuk cargo, tetapi tidak untuk bagasi (*luggage*) penumpang. Selain itu (tidak terlihat pada Gambar 12.8), hanya bagasi penumpang yang memiliki koneksi ke kupon. Dua dua konsep domain serupa tetapi berbeda bila digabungkan menjadi satu kelas. Melalui spesialisasi, dua masalah ini digambarkan: **Piece of Cargo** (3) dan **Piece of Luggage** (4). Degree of hazardousness-Tingkat Bahaya (5) ditempatkan pada tempatnya (Piece of Cargo). Atribut pada class Freight (1) berlaku untuk dua subclass (3) dan (4):



Gambar 86. Contoh Spesialisasi

Coba cek mungkin class yang anda temukan sebelumnya dapat di spesialisasi

Aturan-aturan yang berlaku untuk hubungan antara superclass dan subclass. :

- Semua pernyataan yang dibuat di superclass berlaku juga untuk semua subclass.
- Subclass "**mewarisi/inherit**" atribut, asosiasi, dan operasi dari superclass. Sebagai contoh: Jika superclass Freight memiliki atribut Weight, maka subclass luggage juga memiliki atribut Weight, meskipun atribut ini tidak ada dalam subclass **Piece of Luggage**.
- Dalam terminologi sistem yang sedang dimodelkan, subkelas harus merupakan bentuk khusus dari superclass. Sebagai contoh: **Piece of Luggage** adalah kasus khusus pengiriman (freight).

ELEMEN DARI EXTERNAL VIEW

View eksternal sistem TI terdiri dari tiga elemen berikut:

- **Use case diagram** menunjukkan semua user sistem IT (aktor) dan semua skenario user dengan sistem IT (use case). Pada umumnya dibahas tiap use case, tetapi pada prakteknya, seringkali terlalu rumit untuk menggambarkan semua use case sistem TI dalam satu diagram karena akan terlalu penuh.

- **Use case sequence diagram** menunjukkan proses selama interaksi antara user dan sistem IT untuk tiap use case .
- **Prototipe interface** menunjukkan bagaimana tampilan interface dari use case itu.

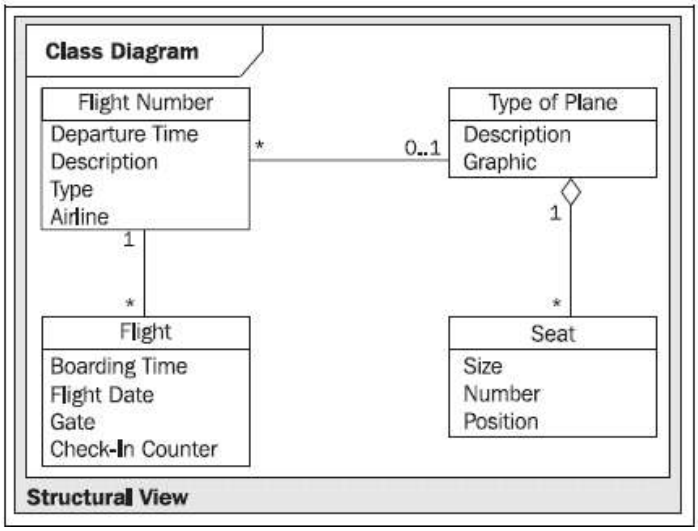
ATURAN BISNIS STATIS DAN DINAMIS

Aturan bisnis adalah aturan batasan (domain) yang digambarkan dalam model sistem TI. Aturan domain dapat berasal dari strategi bisnis, persyaratan, pedoman teknis, dan pembatasan. Aturan bisnis tidak terkait dengan teknologi informasi. Contoh aturan bisnis adalah:

- Selama check-in, setiap penumpang harus diberi kursi.
- Untuk setiap penerbangan, setiap kursi hanya untuk satu penumpang.
- Penerbangan tidak dapat dibatalkan setelah dimulai.

Aturan bisnis dapat dibagi menjadi dua kategori:

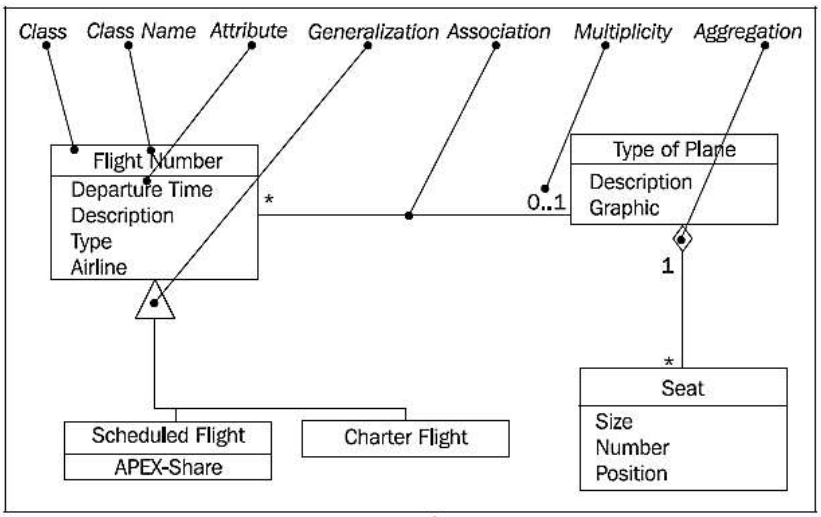
- Aturan bisnis **statis**: Aturan bisnis yang dapat diverifikasi kapan saja. Aturan bisnis ini berhubungan dengan struktur statis sebuah kelas. Aturan bisnis ini dapat didokumentasikan dalam kelas diagram dari structural view.
- Aturan bisnis **dinamis**: Aturan bisnis yang hanya dapat diverifikasi pada titik waktu tertentu, ketika sesuatu terjadi. Aturan bisnis ini berhubungan dengan perilaku dinamis objek dari kelas. Aturan bisnis ini dapat didokumentasikan dalam statechart diagram dari behaviour view (behavioral view).



Gambar 87. Structural View

ELEMEN DARI VIEW

Structural view dari sistem TI pada Gambar 87, terdiri dari satu atau beberapa kelas diagram. Kelas diagram menunjukkan semua kelas yang relevan dari sistem TI, hubungan satu sama lain (**asosiasi**), karakteristik (**atribut**), dan perilaku (**metode**). Class Diagram menunjukkan struktur internal secara statis dari sebuah sistem TI.

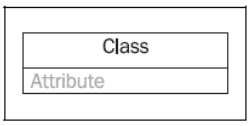


Gambar 88. ELemen Class Diagram

Elemen-elemen class diagram pada Gambar 88:

Class

Mewakili konsep relevan dari domain sistem, bisa terdiri dari set orang, objek atau ide yang menggambarkan sebuah sistem TI.



Contoh class : penumpang(passenger), planes, atau tickets.

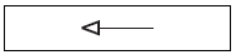
Attribute

Attribut dari class mewakili karakteristik dari sebuah class yang dibutuhkan user pada sebuah sistem TI.

Karakteristik yang dibutuhkan dari class passenger misalnya adalah : nama, tanggal lahir

Generalization/Generalisasi

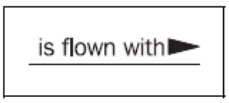
Generalisasi adalah relasi antar dua class : class umum dan class spesial.



(Lihat sub bab Generalisasi, spesialisasi dan inheritance).

Associaton/Asosiasi

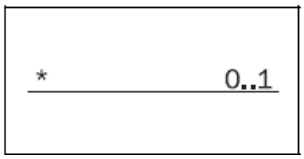
Asosiasi menggambarkan hubungan dua class



Asosiasi menujukkan objek dari sebua class ber relasi dengan kelas lain, koneksi didefinisikan dengan arti tertentu (misalnya is flown with, bekerja untuk dll).

Multiplicity

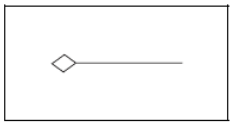
Multiplicity menunjukkan berapa objek yang terlibat dalam asosiasi.



Lihat juga contoh Gambar 90

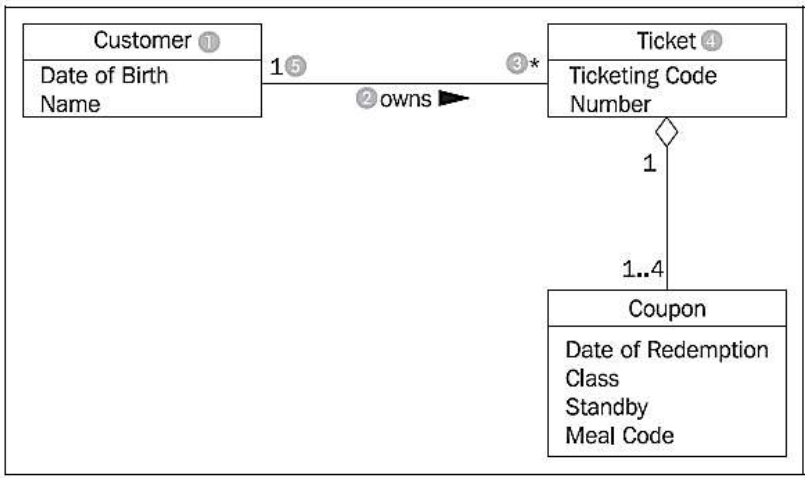
Aggregation/Agregasi

Agregasi adalah kasus khusus(spesial) dari asosasi dengan arti "consist of" / "terdiri dari"



1.32. MEMBACA CLASS DIAGRAM

Gambar 89 menunjukkan class diagram dari kasus yang kita pelajari dengan class customer, ticket dan coupun serta attribut dan asosiasi.



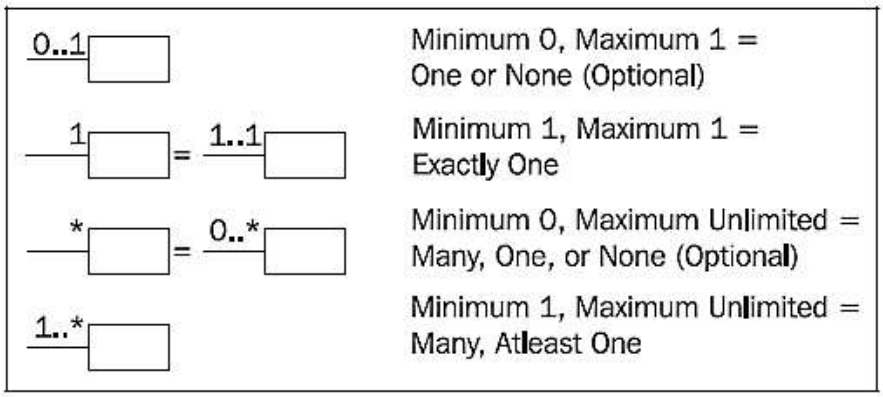
Gambar 89. Class Diagram dengan Asosiasi

Lihat contoh pada Gambar 89, anda dapat membaca asosiasi (hubungan) antar class customer dan tiket sbb :

Satu (kalimat **selalu dimulai dengan "satu"**) objek pada satu class memiliki asosiasi berapa dengan sejumlah objek di kelas berikutnya.

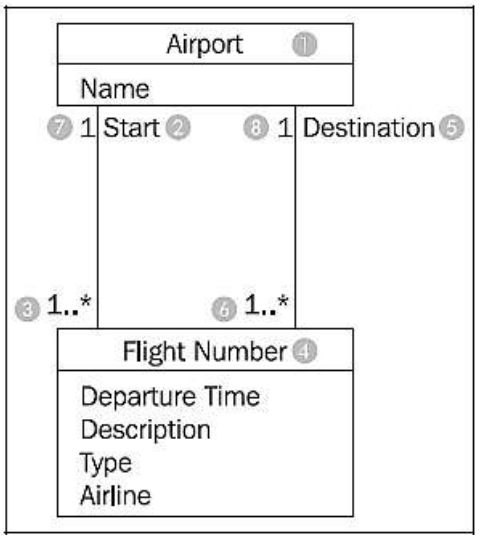
- satu objek customer dapat memiliki 0,1 atau * (lebih dari satu) tickets dan (baca ke arah sebaliknya) - satu objek tickets hanya dimiliki oleh 1 customer
- Ticket terdiri dari 1..4 (min 1 maks 4) coupon dan 1 kupon hanya dimiliki oleh 1 tiket

Spesifikasi jumlah asosiasi antar class itu disebut multiplicity. Min..Maks adalah cara penulisan multiplicity.



Gambar 90. Multiplicity

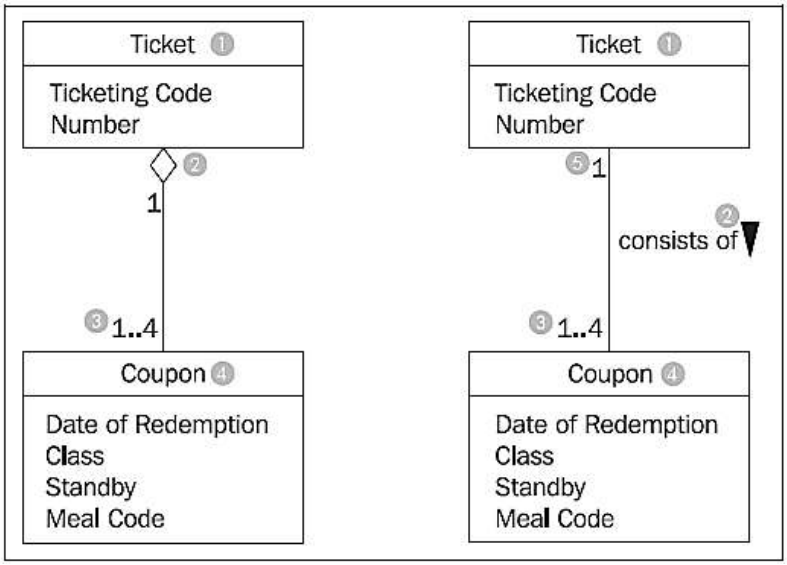
Peran adalah cara lain dalam UML untuk memberikan relasi/hubungan antar class.



Gambar 91. Class Diagram dengan *roles* (peran)

Melihat class diagram pada Gambar 91, dapat dibaca dari asosiasi sebelah kiri dahulu dengan peran dari class flight number dan airport sbb :

- 1 airport dapat start(lokasi) untuk 1 atau beberapa flightnumbers (1->1..*)
- Satu flightnumbers hanya menuju ke satu airport (1->1)

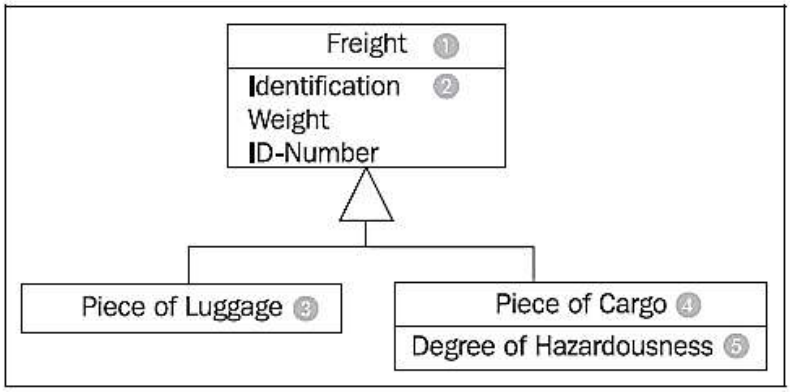


Gambar 92. Class Diagram dengan Agregasi

Asosiasi yang memilki arti bagian dari **"whole-part"** atau **agregasi** adalah tipe relasi bila sebuah objek dari sebuah class merupakan bagian dari kelas lain.

Perbedaan :

- Pada gambar sebelah **kiri** : terdiri dari/**consist of** : Object Coupun adalah bagian dari object ticket.
- Pada gambar **kanan** : Object Coupun berdiri sendiri (bukan bagian dari objek ticket)



Gambar 93. Class Diagram dengan Generalisasi/Spesialisasi

Telah dijelaskan diatas.

1.33. MEMBANGUN CLASS DIAGRAM

Masalah utama untuk membangun kelas diagram adalah menemukan kelas yang "benar". Untuk mengatasi masalah ini kita lihat dari dua perspektif dan membangun kelas diagram dalam dua langkah.

Analisis **top-down**, kelas ditemukan di awal berdasarkan pemahaman umum tentang permasalahan. Analisis ini untuk menemukan struktur dasar dari kelas yang nantinya dapat dibangun berdasarkan analisis rinci dengan bottom-up.

Pertanyaan:

- Informasi apa atau sesuatu konsep domain yang dapat digunakan untuk sistem IT ? Batasan/domain pengetahuan, deskripsi lisan tentang area aplikasi, dan user adalah sumber informasi penting. Dengan cara ini, struktur dasar kelas dapat ditemukan pada sebagian besar sistem TI.

Dalam analisis **bottom-up**, kelas ditemukan berdasarkan input dan output dari sistem TI.

Pertanyaan:

- Informasi apa yang diperlukan untuk masing-masing input dan output dari sistem TI?

Kelas-kelas yang ditemukan selama analisis top-down berfungsi sebagai dasar untuk menemukan kelas-kelas ini. Input dan output yang sudah ada, misalnya **bentuk layar** dan **dokumen** adalah sumber informasi penting.

Checlist berikut menunjukkan langkah-langkah yang diperlukan untuk membuat kelas diagram:

Checklist Menyusun kelas Diagram Kelas pada Structural View:

Analisis Top-down

- Identifikasi dan modelkan kelas — kelas apa yang kita butuhkan?
- Identifikasi dan modelkan asosiasi — Bagaimana kelas terhubung?
- Tentukan atribut — Apa yang ingin kita ketahui tentang objek?

Analisis Bottom-up

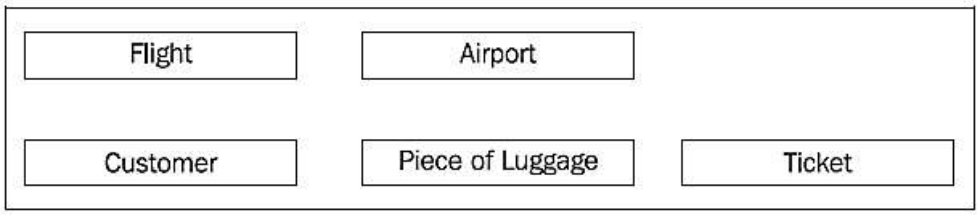
- Buat list query(permintaan) dan input yang dibutuhkan— Apa yang perlu diberikan dan diterima sistem TI?
- Rumuskan query dan input — Bagaimana display yang tepat?
- Lakukan analisis informasi — Kelas, asosiasi, dan atribut apa yang dibutuhkan?
- Review kembali kelas diagram — Apakah sesuai semua/pas?
- Verifikasi kelas diagram — Apakah sudah benar semua?

IDENTIFIKASI DAN MEMODELKAN KELAS— KELAS MANA YANG DIBUTUHKAN?

Analisis hubungan, kebutuhan informasi, dan aktor serta prototipe berdasarkan pengetahuan, diskusikan dengan ahli, dan lihat dokumen. Pertanyaan yang harus ditanyakan adalah:

- Apa hal terpenting yang akan dikerjakan dalam Sistem IT?
- Kelas apa yang bisa dibangun dari hal diatas?

Jawaban atas pertanyaan-pertanyaan ini akan menghasilkan sejumlah kelas potensial, yang dimodelkan dalam draf pertama kelas diagram. Dalam prakteknya, hasil dari langkah kerja pertama ini sangat bervariasi. Jika belum berpengalaman dalam mengidentifikasi kelas, jalankan analisis top-down berulang kali. Seiring waktu waktu, akan dapat mengembangkan apa masuk dalam kelas dan apa yang tidak (Gambar 94):

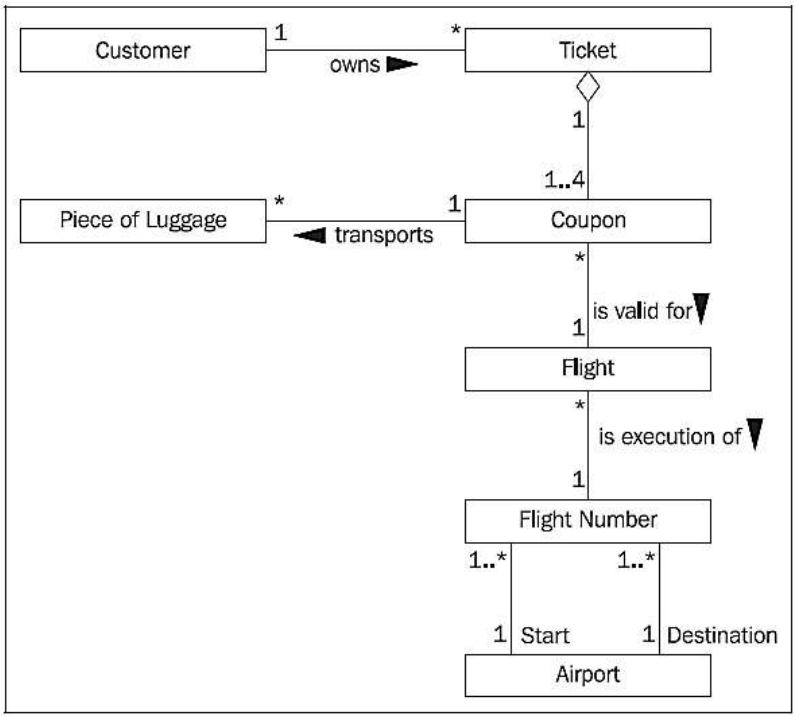


Gambar 94. Potensial Class

IDENTIFIKASI DAN MEMODELKAN ASOSIASI-BAGAIMANA KELAS DIHUBUNGKAN ?

Pertanyaan :

- Relationship apa yang akan muncul antar objek ?
- Berapa objek dari tiap kelas yang terlibat dalam sebuah relasi ?



Gambar 95. Class dan Asosiasi

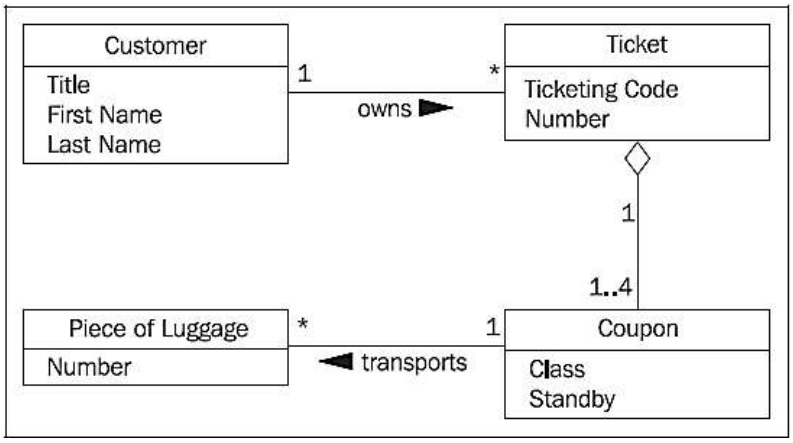
Kelas awal yang ditemukan adalah seperti pada contoh Gambar 95 melalui diskusi awal, tetapi kelas baru akan ditemukan seiring dengan tahap pekerjaan berikutnya.

MENEMUKAN ATTRIBUT-INGIN MENGETAHUI APA TENTANG OBJEK ?

Informasi yang dibutuhkan dari sebuah class harud diidentifikasikan dan dimodelkan dalam bentuk atribut.

Pertanyaan :

- Informasi apa dalam kelas yang dibutuhkan ? Cari attribut dengan menganalisa input dan permintaan (biasanya ditemukan dalam tahap bottom-up analysis).



Gambar 96. Class dan Attribut

LIST KEBUTUHAN PERMINTAAN DAN INPUT-APA YANG HARUS DIBERIKAN DAN DITERIMA SISTEM TI ?

Pada tahap awal analisis bottom-up, sebuah permintaan dan input dari sebuah sistem TI harus teridentifikasi. Permintan sangat penting karena menjawab permintaan adalah tujuan awal dari sebuah sistem TI.

Pertanyaan:

- Informasi apa yang dapat diberikan oleh sistem TI ?
- Informasi apa yang dapat diterima oleh sistem TI ?

Saat menjawab pertanyaan diatas, anda dapat membuatnya berdasarkan use case yang telah dibuat sebelumnya. Di awal, query dan mutation ditemukan pada sistem sequence diagram/use case sequence diagram. Informasi lain anda dapatkan dari bisnis proses dari sistem bisnis (lihat Modul Modeling Business System). Hasil dari langkah ini terdiri dari daftar kebutuhan informasi pada Gambar 97.

Requirements	Type	Use Case
Boarding Pass	Output	Generating Boarding Pass
Passenger List	Output	Complete Boarding
Ticket Details	Output	Check-In, Express Check-In
Customer Details	Output	Check-In, Express Check-In
Coupon	Input	Check-In, Express Check-In
...		

Gambar 97. Daftar Informasi yang Dibutuhkan

RUMUSKAN PERMINTAAN/QUERY DAN INPUT-BAGAIMANA TAMPILAN YANG SEHARUSNYA?

Dalam membuat sebuah class diagram untuk sebuah permintaan dan input, awalnya kita harus menentukan bagaimana tampilanya nanti. Hasil permintaan yang kompleks atau input dikumpulkan atau di draft. Gambar 12.20 menunjukkan passenger list, conto lain lihat juga gambar 12.21 (display) dan Gambar 12.22 (boarding pass).

Pertanyaan :

- Bagaimana tepatnya tampilan sebuah permintaan atau input ?

Sumber informasi ada di form yang ada saat in (contoh:passenger list dari Gambar 98) dan tampilan/display dari prototipe.


Passenger List for Flight LX317
of 4.9.2003, Stand 08:10 hr

Mr	Shirodkar	Abhishek
Mr	Adams	Douglas
Ms	Sakpal	Shilpa
Mr	Baumann	Philippe
Mr	Mohite	Nilesh
Mr	Cleese	John
Ms	Padmanabhan	Nanda
Mr	Jahagirdar	Niranjan
Ms	Chakrabarti	Paramita
Mr	Jarchow	Thomas
Mr	Karnal	Jone
Mr	Gubser	Rolf
Mr	Smith	Harry
Mr	Pande	Ashutosh
Mr	Milligan	Spike
Mr	Schacher	Mark
Mr	Kandalgaonkar	Dinesh

Side 1

Gambar 98. Contoh Tampilan Passenger List

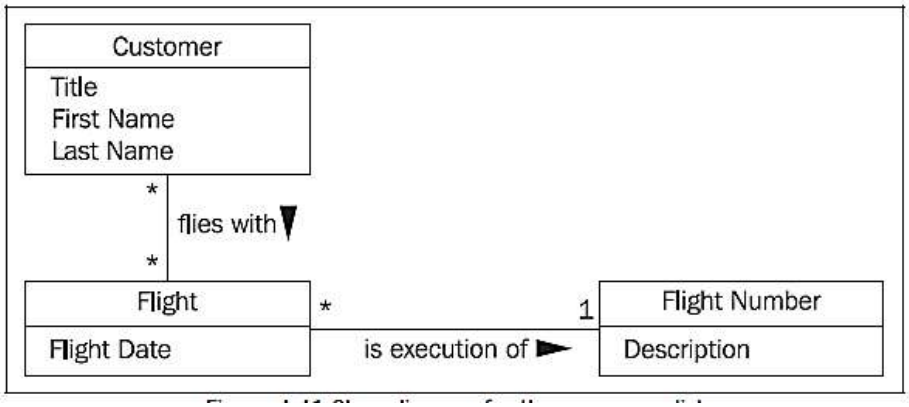
MELAKUKAN ANALISA INFORMASI-CLASS MANA, ASOSIASI DAN ATRIBUT YANG DIBUTUHKAN ?

Pada langka ini, analisa bottom-up dijalankan. Untuk tiap permintaan atau input dibuatlah kelas diagram dari kelas-kelas yang ada. Memodelkan beberapa class pada lingkup yang kecil dahulu.

Pertanyaan :

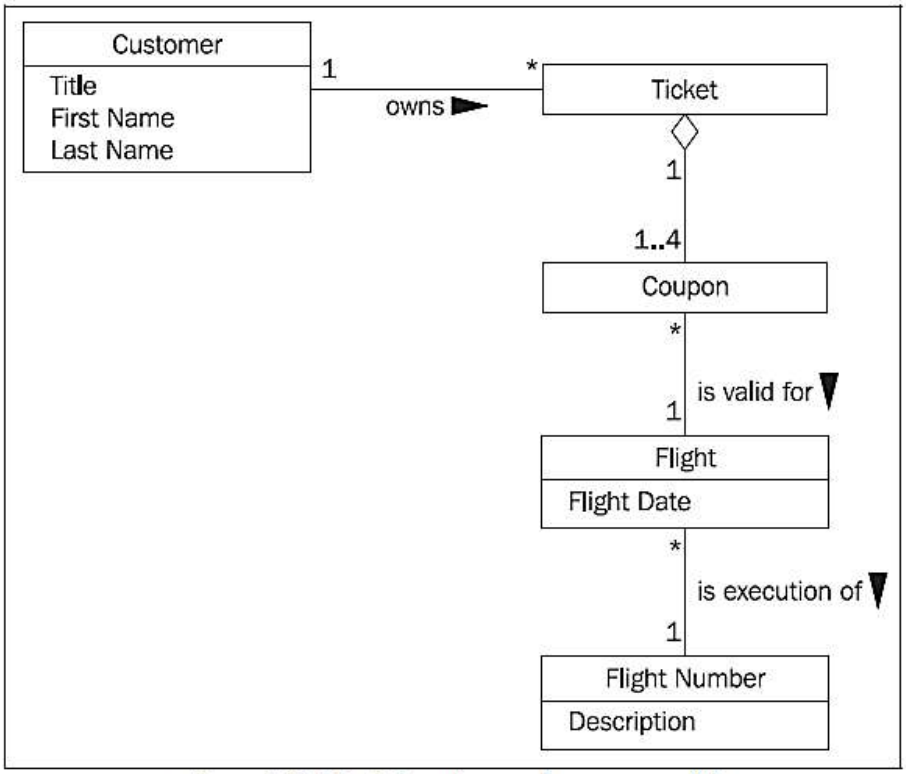
- Elemen data apa yang ada dalam input dan output?
- Objek apa yang disembunyikan di balik elemen data ini?
- Hubungan apa yang ada antara objek yang ditemukan?
- Objek mana yang sudah dimodelkan yang dapat digunakan?

Untuk passenger list/daftar penumpang pada Gambar 98, diagram kelas pada Gambar 99 dibangun sbb:



Gambar 99. Class Diagram Passenger List

Dalam analisa top-down sebelumnya diatas, diubah/disesuaikan dengan class diagram yang baru ditemukan sehingga menjadi (Gambar 100):



Gambar 100. Perubahan Class Diagram

MENGGABUNGKAN DIAGRAM KELAS —SEMUA COCOK BILA DIGABUNG BERSAMA?

Langkah terakhir ini, tiap kelas diagram kelas harus dikonsolidasikan ke dalam satu penggabungan kelas diagram. Bisa saja terjadi ketidakkonsistenan dan diperbaiki. Pertanyaan:

- Apakah ada kelas dalam kelas diagram memiliki nama berbeda, tetapi mewakili maksud sama?
- Apakah ada beberapa relasi yang memiliki arti sama?
- Apakah ada atribut di kelas yang diberi nama berbeda, memiliki makna yang sama?

VERIFIKASI DIAGRAM KELAS — APAKAH SEMUANYA BENAR?

Kelas diagram lengkap dalam stuktural view dapat diverifikasi dengan checklist sbb:

Checklist Verifikasi kelas Diagram Kelas di Structural View:

- Apakah keals diagram sudah lengkap? Pertanyaan ini akan dijawab di Bagian *Interaction View*. Dalam *interaction view*, ditunjukkan bagaimana kelas diagram digunakan untuk menjawab semua permintaan yang diperlukan dari sistem TI. Jika ini sudah sesuai, diagram kelas selesai.
- Apakah kelas diagram sudah benar? pertanyaan ini sedikit lebih sulit untuk dijawab. Sebaiknya berkolaborasi dengan user yang pembawa pengetahuan (yang mengerti).

KESIMPULAN

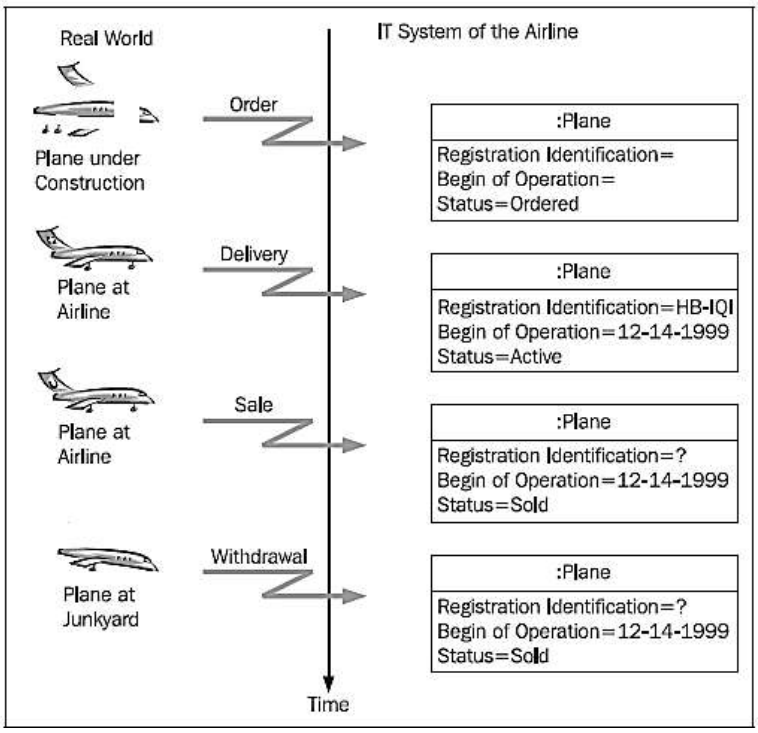
SOAL LATIHAN

UNIVERSITAS BUDI LUHUR
FAKULTAS TEKNOLOGI INFORMASI

MODUL PERKULIAHAN #11
MODELING IT SYSTEM: BEHAVIORAL VIEW

Capaian Pembelajaran	:	Mahasiswa memahami dan dapat menggunakan Modeling IT system behavioral view
Sub Pokok Bahasan	:	1.52. Kehidupan Sebuah Obyek 1.53. Behavioral View (Pandangan Perilaku) 1.54. Statechart Diagram a. Membaca Diagram Statechart b. Membuat Statechart Diagram 1.55. Kesimpulan 1.56. Latihan Soal
Daftar Pustaka	:	Grässle, P., Baumann, H. and Baumann, P., 2005. UML 2. 0 in Action: A Project-based Tutorial. Packt Publishing Ltd.

KEHIDUPAN SEBUAH OBYEK



Gambar 101. Kehidupan Object

Orang, objek, atau konsep di dunia nyata, yang kita modelkan sebagai objek dalam sistem TI memiliki "**kehidupan**". Biasanya, kehidupan dimulai dengan kelahiran, penciptaan dan berakhir dengan kematian, penghapusan, atau kehancuran. Seperti yang diilustrasikan dalam Gambar 101:

Ilustrasi pesawat Airbus A330-223 dari Swiss International Airlines dengan nomor registrasi HB-IQI.

- Kelahiran Airbus A330-223 (dari aslinya) terjadi, tergantung pada perspektif, pada awal pembuatan atau pada penerbangan pertama.
- Kelahiran objek Airbus A330-223 dalam sistem IT terjadi ketika informasi tentang pesawat direkam pertama kalinya. Bisa pada titik pembelian, karena pesawat dapat direkam dalam sistem TI untuk tujuan perencanaan, atau ketika pesawat dikirim. Inisiator kelahiran objek dalam sistem TI selalu merupakan event mutasi.
- Karena pesawat komersial sering dijual jauh sebelum pembuatan, ada kemungkinan kelahiran objek terjadi sebelum kelahiran (fisik) pesawat.
- Kematian yang nyata terkait kehancuran fisik. Dalam kasus Airbus A330-233 ini, kematian terjadi pada saat penarikan atau mungkin kecelakaan pesawat.

- Kematian objek terjadi, ketika objek dihapus dari sistem IT Swiss Airline. Inisiator untuk kematian suatu objek dalam sistem TI selalu merupakan event mutasi.

Karena pesawat komersial sering dijual setelah periode waktu tertentu, ada kemungkinan logis kematian suatu objek dalam sistem TI terjadi sebelum (fisik) kematian aslinya.

Antara kelahiran dan kematian objek hidup dalam sistem TI, yaitu, itu akan dibaca dan diubah. Itu akan dibaca sebagai hasil dari query event; akan diubah sebagai hasil dari event mutasi (lihat Bagian Query event dan Mutation event).

Aturan untuk modifikasi harus diperhatikan, menjadi penting untuk mendokumentasikan aturan ini diletakkan di suatu tempat.

Sebuah aturan bisnis yang dinamis (lihat Bagian Aturan Bisnis Statis dan Dinamis). **Aturan bisnis dinamis** adalah aturan yang hanya **berlaku pada titik waktu tertentu**, yaitu saat **query event atau mutation event terjadi**. Perilaku objek sangat ditentukan oleh aturan bisnis yang dinamis.

Contoh aturan bisnis yang dinamis adalah:

- Sebuah pesawat tidak dapat diizinkan terbang selama dalam pemeliharaan.
- Sebuah pesawat tidak dapat ditarik selama masih dijadwalkan untuk penerbangan.

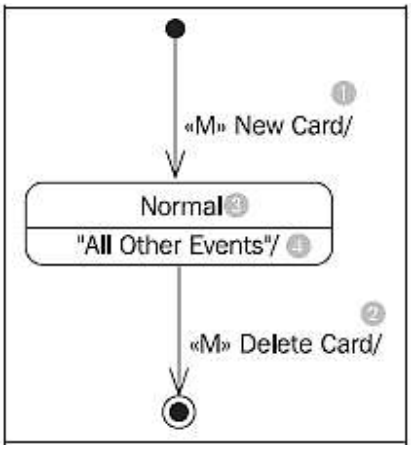
Jika melihat lebih dekat pada aturan bisnis, ini itu merujuk pada event tertentu di satu sisi, dan ke state objek di sisi lain:

- Mutation event yang **menetapkan penerbangan(*flight*)** pesawat tidak diizinkan bila objek pesawat kondisinya sedang dalam pemeliharaan.
- Mutation event **menarik pesawat** pada kondisi/keadaan(state) object pesawat sudah terjadwal pada penerbangan.
- Mutation event **starting plane** (jalankan pesawat) tidak boleh dilakukan bila object pesawat sedang dalam kondisi/keadan (state) transit

Dengan kata lain: pada event tertentu, dimungkinkan untuk menentukan apakah event bisa dilakukan/tidak dalam keadaan/state objek, dan bagaimana objek tsb bereaksi terhadap event.

Diskusikan aturan bisnis dinamis yang dapat berlaku untuk **objek tiket pesawat** dalam sistem layanan penumpang.

Dalam behavioral view, satu diagram statechart untuk tiap kelas digunakan untuk mendokumentasikan aturan bisnis dinamis apa yang harus diikuti, dan event apa yang diizinkan dalam state sebuah objek. Contoh kasus sederhana, semua event diizinkan. Gambar 102 menunjukkan diagram statechart sederhana untuk kelas kartu frequent flyer (*member card*) dengan kondisi normal:



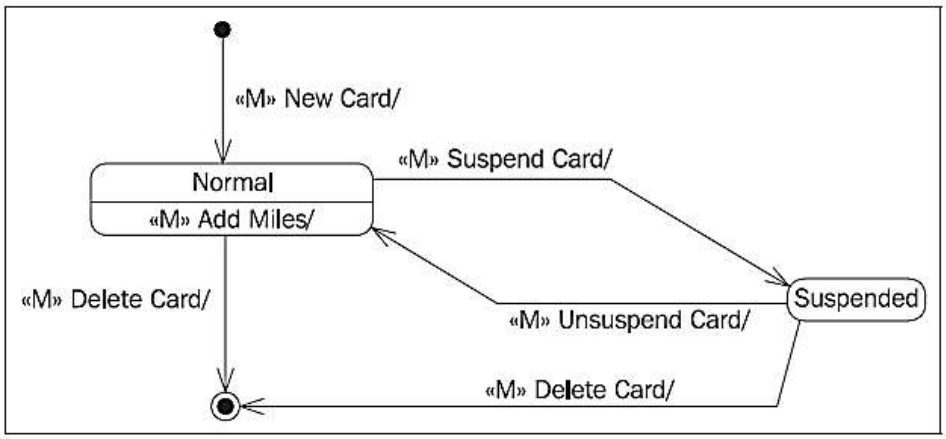
Gambar 102. Diagram Statechart Sederhana Kartu Frequent Flyer

Objek baru dibuat oleh event «M» New Card (1). Objek dihapus oleh event«M» Delete Card(2). Di antara objek berada dalam state Normal (3), di mana "all other events" diizinkan (4).

Jika ditambahkan aturan bisnis, diagram statechart menjadi lebih kompleks. Perubahan diagram statechart dengan ditambahkan aturan bisnis sbb:

- Harus dimungkinkan untuk supsend(ditangguhkan) dan pengembalian kartu
- Tidak bisa menambahkan jumlah miles (jarak terbang) untuk kartu yang di suspend

Diagram statechart berubah menjadi (Gambar 103) :



Gambar 103. Diagram Statechart Lebih Kompleks

Contoh diagram statechart kartu frequent flyer pada Gambar 102, menunjukkan jalur mana atau di mana batas kehidupan objek kartu frequent flyer dapat dilanjutkan. Dalam diagram, rantai event yang mungkin dan tidak mungkin dapat dikenali.

Urutan yang dibolehkan **:** misalnya,

«M new card » (kartu baru)**, «M»add miles**(tambahkan mile), **«M»add miles,«M» add miles, «M»suspend card**(kartu ditangguhkan)**, «M» delete card** (hapus kartu)**.**

Contoh urutan yang tidak diizinkan adalah:

«M» new card, **«M»add miles**, **«M» add miles**, **«M» supsend card**(kartu ditangguhkan), **«M» add miles**, «**M»delete card**.

Event kedua hingga terakhir, «M» add miles, **tidak diterima** jika kartu unsuspend(tidak ditangguhkan), mile dapat ditambahkan kembali.

Pertimbangkan apakah rantai event berikut diizinkan atau tidak sesuai dengan diagram statechart kelas kartu frequent flyer pada Gambar 102:

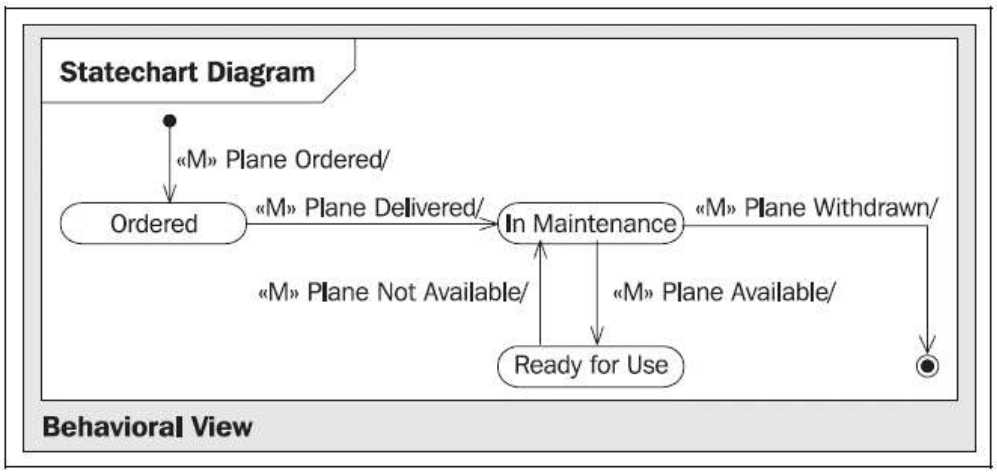
- **«M»add miles, «M» add miles, «M» add miles, «M»delete card, «M»delete card.**

- **«M» new card, «M» add miles, «M»suspend card, «M» unsuspend card, «M» add miles, «M»delete card.**

- **«M» new card, «M» add miles, «M»suspend card, «M» kartu penangguhan, «M»delete card.**

Secara umum, kehidupan suatu objek mengikuti arah yang telah ditentukan sebelumnya, berarti objek tersebut harus mengikuti aturan tertentu. Dengan demikian, behavioral view (pandangan perilaku) sangat penting, adalah tugas sistem TI untuk memastikan bahwa aturan ini dipatuhi. Aturan perlu didokumentasikan dengan cara yang benar dan lengkap, untuk menghindari kesalahpahaman di sisi pengguna dan sisi pengembang. Dalam sistem TI yang lengkap, seharusnya tidak mungkin bagi pengguna untuk menghapus atau memodifikasi objek ketika tidak diizinkan oleh aturan bisnis.

BEHAVIORAL VIEW (PANDANGAN PERILAKU)



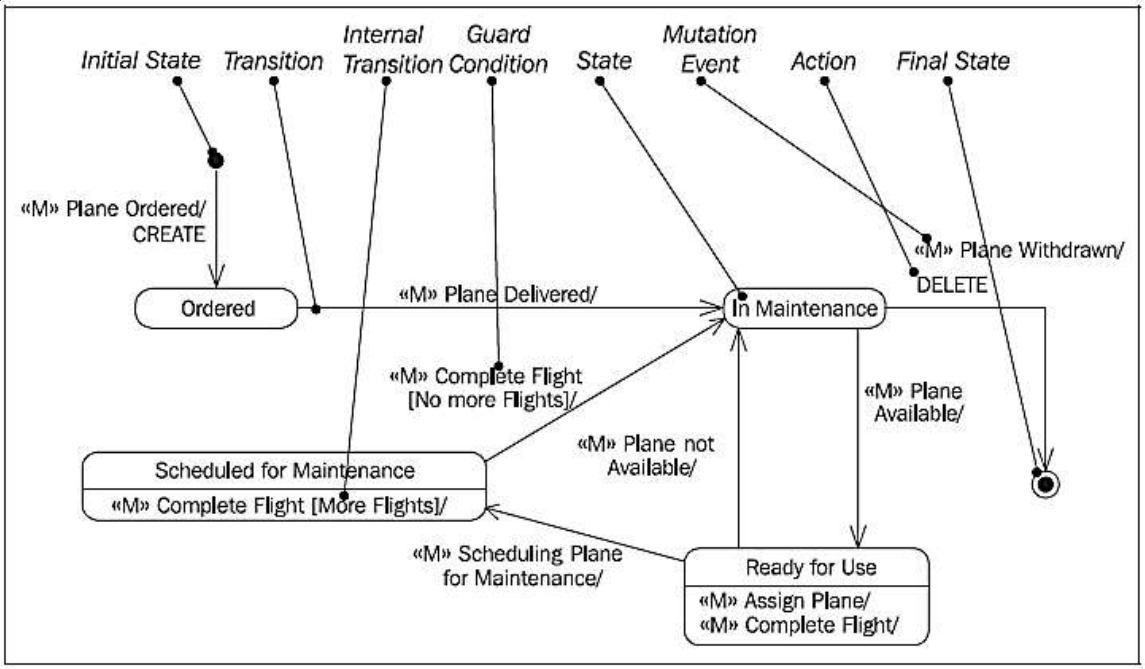
Gambar 104. Behavioral View

Behavioral View dalam Gambar 104, terdiri dari banyak diagram statechart, yang masing-masing menunjukkan perilaku tiap objek. Oleh karena itu, gabungan semua

diagram statechart akan menunjukkan perilaku semua objek dari sistem TI. Namun, dalam prakteknya tidak semua diagram statechart dibangun, hanya diagram yang:

- Mengandung banyak atau aturan bisnis penting atau
- Menggambarkan objek-objek penting

STATECHART DIAGRAM

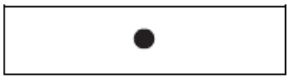


Gambar 105. Elemen Statechart Diagram

Elemen-elemen dalam diagram statechart yang ditunjukkan pada Gambar 105:

State awal (Initial State)

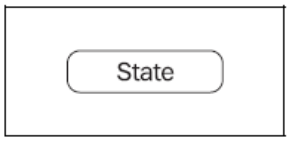
State awal mewakili sumber dari semua objek:



Ini bukan state(keadaan) normal, karena objek dalam state ini belum ada.

State (Keadaan)

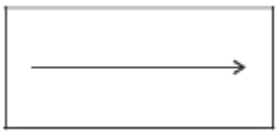
State objek selalu ditentukan oleh atribut dan asosiasinya. Status dalam diagram statechart mewakili sekumpulan kombinasi nilai tersebut, dimana objek berperilaku dalam merespons event:



Karena itu, tidak setiap modifikasi atribut mengarah ke state baru.

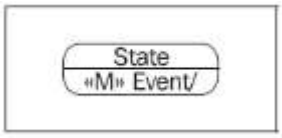
Transisi / Transition

Transisi mewakili perubahan dari satu state ke state lain:



Transisi Internal / Internal Transitition

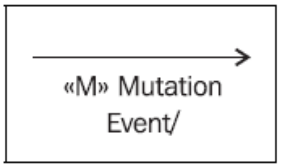
Transisi internal adalah transisi dari satu state ke state itu sendiri. Objek menangani event tanpa mengubah kondisinya:



Event yang memulai transisi internal tercantum di bagian bawah simbol state. Sebagai contoh, objek kartu frequent flyer dalam status normal tetap dalam status normal ketika event «M» add miles terjadi.

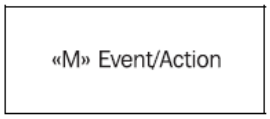
Query Event

Event mutasi adalah pencetus transisi dari satu state ke state lain, atau untuk transisi internal di mana state nya sama:



Action(Tindakan)

Suatu tindakan adalah aktivitas dari suatu objek yang diprakarsai oleh suatu event:



Suatu tindakan menggambarkan apa yang dilakukan objek sebagai respons terhadap event tersebut. Deskripsi ini dapat bersifat tekstual atau formal.

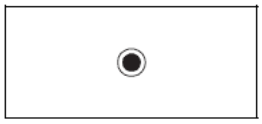
Guard Condition (Penjaga Kondisi)

Kondisi penjaga adalah kondisi yang harus dipenuhi untuk memungkinkan transisi:

Kondisi penjaga dapat digunakan untuk mendokumentasikan bahwa event tertentu, tergantung pada kondisinya, dapat menyebabkan transisi yang berbeda.

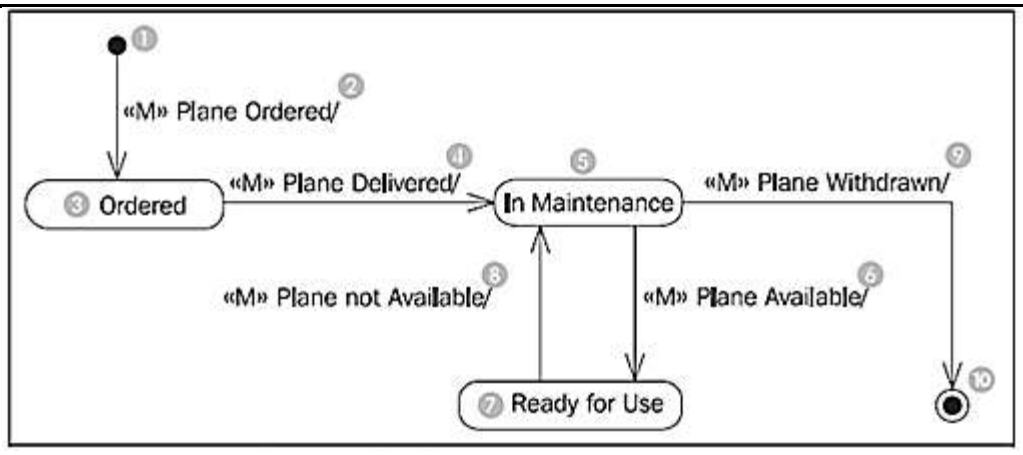
Final State(State Akhir)

State akhir mewakili akhir dari keberadaan objek:



State akhir bukanlah state nyata, karena objek dalam state ini tidak ada lagi.

1.34. MEMBACA DIAGRAM STATECHART



Gambar 106. Diagram Statechart dengan Berbagai Events

Diagram pada Gambar 106 menunjukkan semua state(keadaan) objek-objek berada selama masa hidupnya. Ini menunjukkan kemungkinan transisi antara state-state(keadaan) dan event(event/kejadian) yang mengawali transisi ini.

Setiap objek dari kelas pesawat (1) (initial state) dan menghilang, menjadi tidak ada (10) (final state). Ini biasanya berlaku untuk semua kelas, artinya sebagian besar kelas akan menemukan initial state(1) dan final state(10).

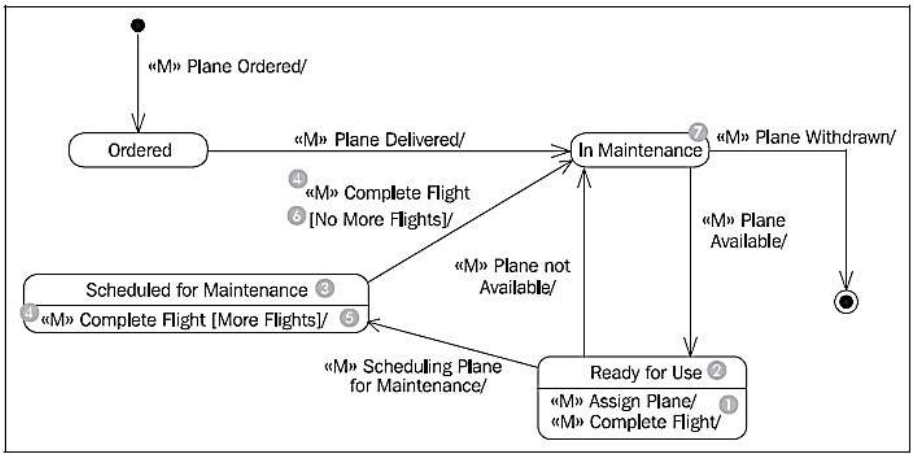
Selama masa hidupnya, sebuah objek pesawat (bukan objek pesawat nyata ya ☺) ada tiga state: **ordered**(dipesan) (3), **in Maintenance**(pemeliharaan) (5), dan **ready for use** (siap digunakan) (7)

Event **«M» plane ordered**, mengarah pada kejadian (1) objek pesawat baru terbentuk dalam sistem TI (kelahiran). Dan berubah state menjadi state **ordered** (3).

Jika event **«M» plane delivered** (4) terjadi, dan pesawat dalam state ordered (3), state berubah menjadi **in maintenance**(5). Jika pesawat berada dalam state selain **ordered**, tidak ada yang terjadi.

Melalui event **«M» plane available**(pesawat tersedia) (6) dan **«M» plane not available** (pesawat tidak tersedia) (8), state pesawat berubah beberapa kali antara state **in maintenance** (5) dan **ready to use** (7).

Pada akhir hidupnya, objek pesawat menghilang melalui event **«M» plane withdrawn**(pesawat ditarik) (9) menjadi tidak ada (10), berarti dihapus (mati).
Gambar 107 menampilkan lebih banyak elemen yang dapat terjadi dalam diagram statechart:

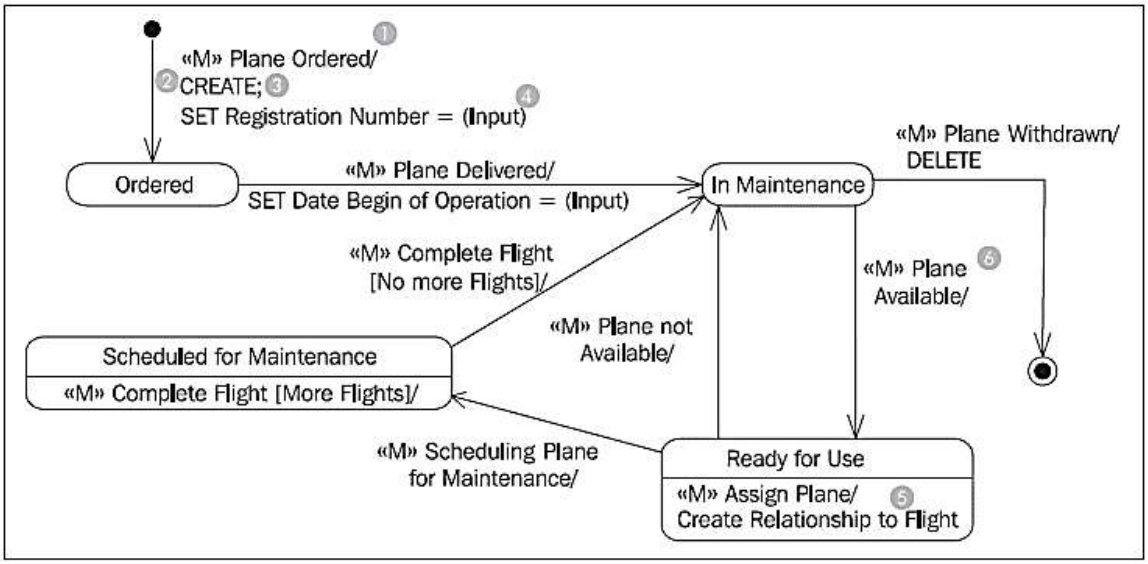


Gambar 107. Statechart Diagram dengan Transisi Internal dan Guard Condition

Selain transisi yang telah djelaskan, terdapat transisi internal.

Event **«M» Assign Plane** (1) terjadi ketika pesawat ditugaskan untuk penerbangan, tidak memulai transisi ke state lain. Sebaliknya, pesawat tetap dalam kondisi **ready for use** (siap untuk digunakan) (2). Ini merupakan transisi internal; objek pesawat dalam kondisi **ready for use** (2) sebelum dan sesudah event.

Guard conditions memungkinkan penerimaan atau penolakan terhadap suatu event tergantung pada suatu kondisi. Jika dalam state Scheduled for Maintenance (dijadwalkan untuk pemeliharaan) (3) Event **«M» Complete Flight** (4) terjadi, respons objek tergantung pada guard conditions. Jika kondisi [**More Flights**] (5) benar (artinya ada lebih banyak penerbangan yang ditugaskan ke pesawat) transisi internal terjadi. Pesawat tetap dalam kondisi terjadwal untuk pemeliharaan (3). Namun, jika kondisi [**No More Flight**] benar (artinya tidak ada penerbangan lain yang ditugaskan ke pesawat) transisi ke state **in Maintenance**(7) terjadi.



Gambar 108. Statechart Diagram dengan Action

Action menunjukkan bagaimana suatu objek merespons mutation event. Gambar 108 menunjukkan beberapa jenis action. Suatu action selalu didahului dengan garis miring (1) setelah nama event.

Action **CREATE** (2) dan **SET Registration Number = (input)** (4) mengikuti event **mutation «M» Plane Ordered**. **CREATE** menunjukkan objek baru dibentuk**; SET Registration Number = (input)** menunjukkan nilai yang dimasukkan pengguna dalam use case ditetapkan ke nomor registrasi atribut. Tiap action dipisahkan dengan titik koma (;) (3). Action juga dapat dijelaskan dalam teks (lihat Bagian 4.3.4, Membangun Diagram Statechart). Setelah mutation event **«M» assign plane** adalah action **create relationship to flight** (5), menunjukkan dibentuknya hubungan dengan objek flight.

Jika tidak ada action yang di state untuk suatu event(6), bisa berarti action belum ditentukan, atau objek hanya bertransisi ke state lain.

Mendapatkan pemahaman yang lebih dalam tentang studi kasus ini, Anda melihat bahwa diagram statechart pada Gambar 14.8 harus diubah dengan state dan event lebih lanjut.

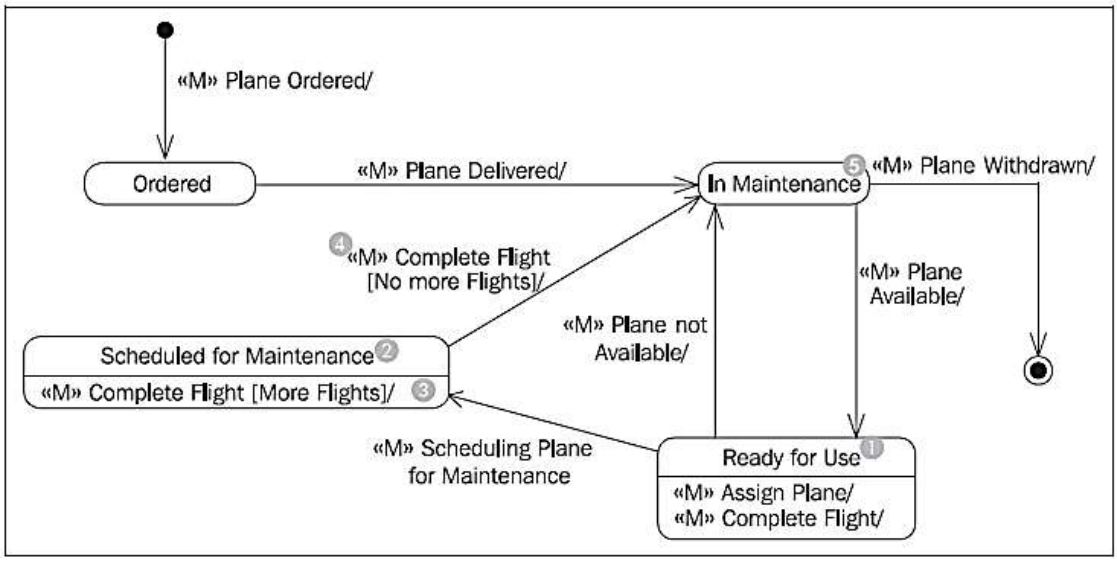
Diagram statechart yang mendokumentasikan semua jalur (path) yang mungkin dari suatu objek tidak bisa begitu saja dibaca secara berurutan. Namun, ini membantu pembaca untuk menjawab beberapa pertanyaan umum :

Apa yang terjadi pada objek jika suatu event tertentu terjadi? Jawaban untuk pertanyaan ini adalah dalam setiap kasus tergantung pada state objek saat ini, pertanyaannya sebaiknya:

Bagaimana objek dalam state tertentu merespon event tertentu ?

- Event apa yang berhubungan dengan objek ?
- Bagaimana ? artinya melalui event, sebuah state ditinggalkan ?
- Bagaimana ? artinya melalu event, sebuah state dapat tercapai ?

Untuk menjawab pertanyaan diatas, kita lihat Gambar 109



Gambar 109. Pembacaan Selektif Statechart Diagram

- Bagaimana sebuah object **plane** pada state **ready for use** (1) bereaksi terhadap event «**M**» **assign plane**? Untuk menjawab ini, kita lihat event «**M**» **assign plane** yang ada di state **ready for use** (1). Event membolehkan transisi (panah) ke state yang ada atau transisi internal. Pada contoh diatas, transisi ke state lain tidak ada, tetapi transisi internal ada. Artinya : sebuah objek plane pada state ready for use (1) menerima event «**M**» **assign plane** dan tetap berada pada state **ready for use** (1)
- Bagaimana sebuah objek plane pada state **scheduled for maintenance** (2) bereaksi terhadap event «**M**» **complete flight**? Untuk menjawab ini, kita lihat dulu event «**M**» **complete flight** yang ada di state **scheduled for maintenance** (2). Pada contoh, terdapat transisi ke state lain dan juga transisi internal. Karena hanya 1 transisi yang diperbolehkan (objek hanya boleh memiliki 1 state pada satu saat), harus diuat kriteria untuk menentukan transisi. Contoh digunakan guard condition **[more flighs]** (3) dan **[no more flights]**. Periksa apakah masih ada penerbangan(flight) yang di assign ke pesawat. Misalnya, sudah tidak ada lagi penerbangan yang di assigned ke pesawat, berarti : Objek plane (pesawat) dalam kondisi **scheduled for maintenance** (2) menerima event «**M**» **complete flight** dan ber transisi ke state **in Maintenance** karena sudah tidak ada penerbangan (flights) yang di assigned ke plane (pesawat).

- Bagaimana objek plane pada state scheduled for maintenance (2) bereaksi terhadap event «M» assign flight? Untuk menjawab ini, kita cek dulu event «M» assign flight pada state scheduled for maintenance (2). Pada contoh, tidak ada transisi ke state lain atau transisi internal. Artinya: objek **plane** berada pada state **scheduled for maintenance** (2) tidak menerima event **«M» assign plane**. (Sistem TI harus mengeluarkan menginformasikan kepada user mengapa tidak berjalan).
- Event apa yang relevan untuk objek plane? Jawab:seluruh event yang ada di statechart diagram pada kelas **plane** yaitu **«M»plane ordered, «M»plane delivered, «M» plane available, «M»plane not available, «M» assign plane, «M» complete flight, «M» scheduling plane for maintenance, and «M» plane withdrawn.**
- Melalui event yang mana objek plane meninggalkan **state in maintenance**(5) ? Lihat transisi (panah) yang berasal dari state in maintenance(5) ke state lain. Pada contoh ada 2 transisi. Artinya: Objek **plane** pada state **in maintenance**(5) dapat keluar dari state melalu event **«M» plane available atau «M» plane withdrawn.**
- Melalui event mana objek plane mencapai state ready for use(1)? Lihat seluruh transisi(panah) menuju state ready for use(1). Pada contoh hanya satu. Artinya: Objek Plane dapat mencapai state ready for use(1) melalu event **«M» plane available** (berasal dari state in maintenance (2))

Dari pembahasan diatas menunjukkan bahwa pada statechart diagram, apa yang tidak tertulis sama pentingnya dengan yang tertulis. Event-event yang tidak muncul pada sebuah state tidak dapat diterima saat sebuah objek berada pada state tertentu. Artinya event tersebut tidak dapat dijalankan pada sistem TI bila tidak diterima dan dibuat pesan error. Event yang tiak ada di sebuah state pasti akan diabaikan. Dapat dikatakan :

- Jika sebuah pesawat dihasilkan tidak pernah langsung ke state ready for use, selalu awalnya ke state in maintenance.
- Sebuah pesawat yang ready for use tidak dapat ditarik (withdrawn). Jika dijalankan, maka mutation event akan gagal dan muncul pesan error.

1.35. MEMBUAT STATECHART DIAGRAM

Checklist dibawah ini adalah langkah yang diperlukan untuk membuat statechart diagram dari sebuah class. Urutannya adalah sbb :

Checklitst Membuat statechart diagram dalam Behavioral View

- Identifikasi mutation events yang relevan pada objek—apa dampak terhadap objek?
- Kelompokkan event berdasarkan urutan kronologis— Bagaimana terlihatnya hidup yang normal ?
- Modelkan state dan transisi—Ada state apa saja?
- Tambahkan action pada statechart diagram—Objek melakukan apa?
- Verifikasi statechart diagram-apaka sudah benar?

IDENTIFIKASI MUTATION EVENT YANG RELEVAN UNTUK OBJEK — APA DAMPAK TERHADAP OBJEK?

Pertama, kita harus menentukan mutation events yang relevan terhadap objek, artinya mutation event mana yang meng-inisiasi actions, transisi state atau keduanya pada sebuah objek.

Pertanyaan :

- Mutation event apa yang mengarah pada pembentukan objek atau penghapusan?
- Mutation event apa yang mengarah pada penetapan atau perubahan atribut?
- Mutation event apa yang nantinya mengerah pembentukan hubungan ke objek lain atau pemutusan hubungan antar objek?
- Mutation event apa yang menghasilkan transisi state dari sebuah objek?
- Mutation event apa dari use case sequence diagram/sistem sequence diagram berasal dari external view yang berdampak terhadap objek ?

Jawaban dari pertanyaan diatas mengarah pada daftar mutation event yang relevan pada sebuah objek. Karena mutation event bersumber dari Use Case, sebuah use case baru dibentuk bila event mutation yang baru tidak terdapat di use case sequence

diagram. Sebuah event yang tidak dikirimkan ke sistem TI pada use case tidak penah dikirim ke sistem TI. Ini menunjukkan bahwa use case baru harus dimodelkan. Pada kasus ini misalnya, ditemukan mutation event yang relevan untuk object flight:

«M» flight defined, «M» flight started, «M» flight landed, «M» flight canceled, «M»new flight date, «M» flight irrelevant, and «M» flight number irrelevant.

KELOMPOKKAN EVENT YANG RELEVAN SECARA KRONOLOGIS-BAGAIMANA TERLIHATNYA HIDUP YANG NORMAL ?

Mutation Event dibagi tiga kelompok: event yang mengarah pada **penciptaan objek baru** (kelahiran), **event penting** selama keberadaan suatu objek (kehidupan), dan event yang mengarah pada **penghapusan suatu objek** (kematian) . Pertanyaannya adalah:

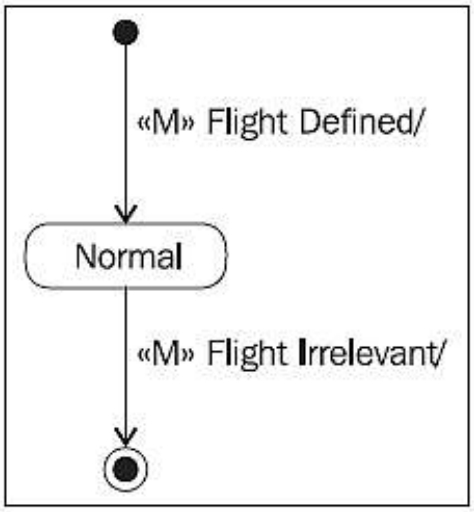
- Pada tahap mana kehidupan suatu objek dimiliki oleh setiap mutation event?

Event mutasi dari kasus ini untuk objek flight(penerbangan) dikelompokkan sebagai berikut:

- Lahir : **«M» flight defined**
- Hidup: **«M» flight started, «M» flight landed, «M» flight canceled, and «M»new flight date**
- Kematian: **«M» flight irrelevant and «M» flight number irrelevant**

MODEL STATES DAN TRANSISI-ADA STATE APA SAJA?

Pada draft awal, dimulai dengan statechart diagram sederhana, yang terdiri dari initial state, normal state dan final state. Contoh Gambar 110 menunjukkan diagram dari objek flight.

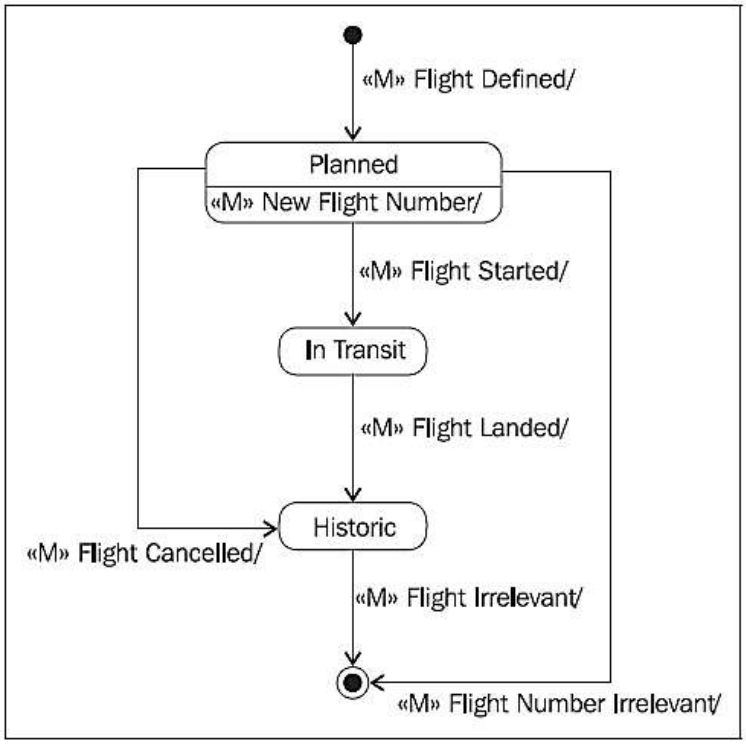


Gambar 110. Statechart Sederhana

Mulai dengan diagram sederhana dulu, kemudian mutation event **dapat ditambahkan** mengacu beberapa pertanyaan tiap event:

- Apakah mutation event ini berlaku disetiap case artinya semua state atau state tertentu tidak dibolehkan? Tergantung pada case yang terjadi, bila **mutation event diizinkan** maka digambarkan sebagai **state**. Dibelakang case ini ada aturan (rule) bisnis yang dibahas diatas (sub bab Aturan Bisnis Statis dan Dinamis).
- Pada **state mana terjadinya mutation**? State baru tergantung pada state objek sebelum terjadi mutation event.
- Apakah **transisi dari state baru tergantung pada kondisi tertentu**? Kita dapat menggunakan **guard condition** untuk dokumentasi mutation event-tergantung kondisi-dapat saja mengarah pada terbentuknya state baru. Pada contoh Gambar 111, event **«M» flight started** diizinkan apabila flight sebelum dalam state in transit.

Bila seluruh pertanyaan diatas terjawab untuk setiap mutation event, dibuatlah diagram statechart pada Gambar 111.



Gambar 111. Statechart Diagram Pada Class Flight

TAMBAHKAN ACTION PADA STATECHART DIAGRAM-APA SAJA YANG DILAKUKAN OBJEK?

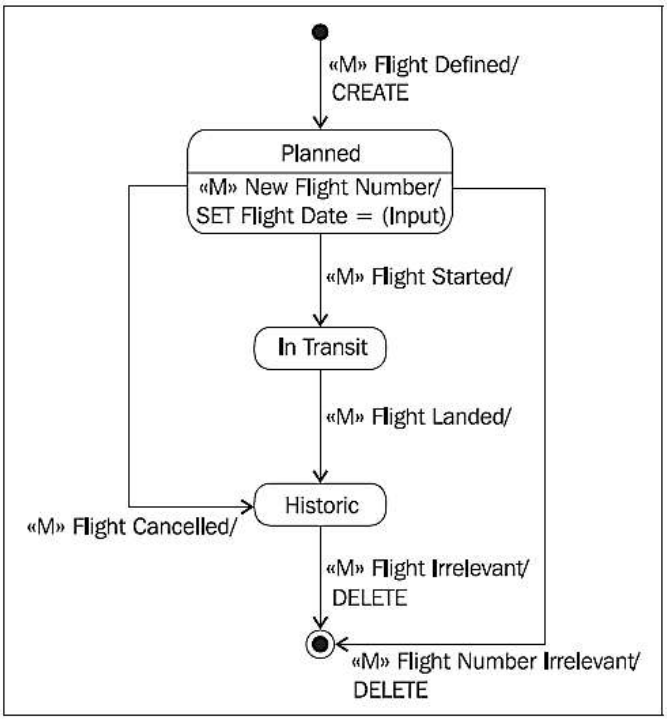
Setelah mutation event pada sebuah objek ditemukan dan dimodelkan, selanjutnya adalah menentukan action. Berikut adalah pertanyaan yang harus dijawab:

- Dimana action dibutuhkan saat berurusan dengan perubahan nilai atribut?
- Dimana action dibutuhkan saat menangani hubungan?
- Dimana lagi action dibutuhkan(mengaktifkan query, kalkulasi)?

Action yang dibutuhkan dimasukkan ke dalam statechart diagram. Tidak masalah menggambarkan action secara informal menggunakan bahasa Inggris biasa, tapi baiknya menggunakan bahasa formal dengan beberapa kata yang sering digunakan :

- **CREATE / DELETE**: Membuat atau menghapus kelas objek
- **SET** <attribute>: = ...: Mengatur nilai atribut.
- **TIE TO** <object> / **CUT FROM**<object>: Membangun hubungan dengan objek lain atau memutuskan hubungan dengan objek lain.

Gambar 112 menunjukkan diagram statechart untuk kelas penerbangan (flight) dari kasus menggunakan action:



Gambar 112. Statechart Diagram dari Class "Flight" dengan Action

PERIKSA STATECHART DIAGRAM-APAKAH SEMUA SUDAH BENAR?

Sebuah statechart diagram yang lengkap dapat verifikasi dengan checklist sbb:

Checklist Verifikasi Statechart Diagram pada Behavioral View

- Apakah ada final state yang dirumuskan, atau ada objek yang tidak pernah mati?
- Apaka ada transisi(tidak langsung) dari tiap state yang ada ke final state?
- Apakah ada perbedaan (jika relevan), antara kematian secara logis (objek dibekukan) dan kematian secara fisik (penghapusan objek)?
- Apakah ada setidaknya satu event tertentu untuk masing-masing state, di mana timbul respon tertentu dari state ini? Jika tidak ada, maka state harus diperbaiki.
- Jika dua atau lebih transisi yang di inisiasi oleh event yang sama meninggalkan state yang sama, guard conditions harus dipisahkan (tidak mungkin waktu bersamaan).

KESIMPULAN

SOAL LATIHAN

UNIVERSITAS BUDI LUHUR
FAKULTAS TEKNOLOGI INFORMASI

MODUL PERKULIAHAN #12
MODELING IT SYSTEM: INTERACTION VIEW

Capaian Pembelajaran	:	Mahasiswa memahami dan dapat menggunakan Modeling IT system interaction view
Sub Pokok Bahasan	:	1.57. Melihat Apa Yang Terjadi Di dalam Sistem TI 1.58. Elemen dari View 1.59. Communication Diagram a. Membaca Communication Diagram 1.60. Sequence Diagram a. Stereotype Class b. Membaca Sequence Diagram 1.61. Membangun Communication Diagram 1.62. Membangun Sequence Diagram 1.63. Kesimpulan 1.64. Latihan Soal
Daftar Pustaka	:	Grässle, P., Baumann, H. and Baumann, P., 2005. UML 2. 0 in Action: A Project-based Tutorial. Packt Publishing Ltd.

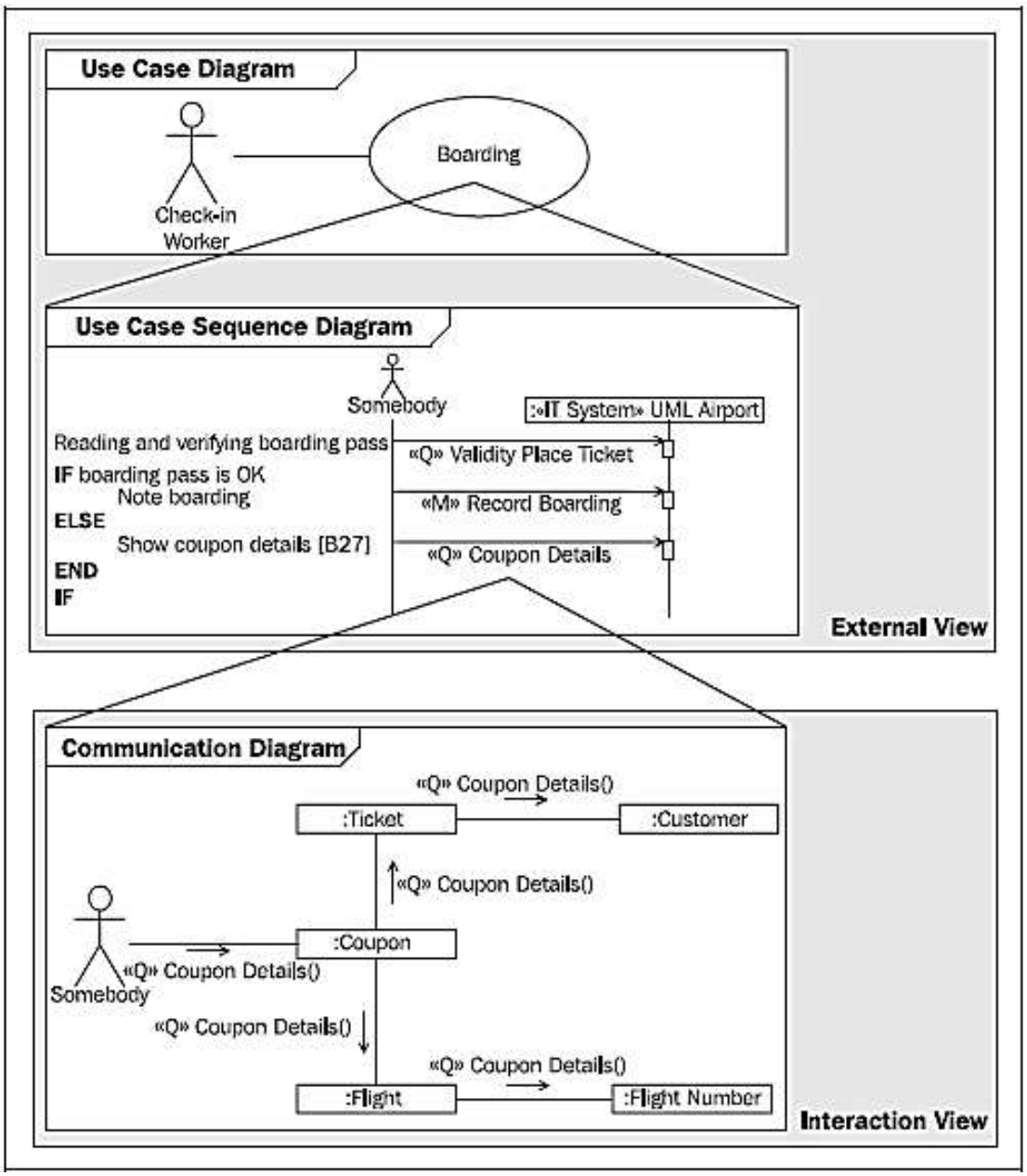
MELIHAT APA YANG TERJADI DI DALAM SISTEM TI

Dalam kebanyakan kasus, fokus hanya pada struktur tidak cukup untuk memahami suatu sistem. **Mengidentifikasi aliran(flow)** adalah tool yang ampuh untuk menjelaskan sesuatu. Static View dari kelas tidak cukup untuk memahami sistem TI. Misalnya, apa yang terjadi dalam sistem TI ketika penumpang check-in? Aliran mana yang sedang diproses? Objek mana yang terpengaruh? Pertanyaan-pertanyaan ini dijawab oleh **Interaction View**.

Sama pentingnya, bahwa memodelkan interaction view membantu untuk verifikasi dan penyelesaian Static View. Dengan mengkaji kelas diagram dari memodelkan query sampai ke tiap atribut, dipastikan bahwa kelas diagram memenuhi semua kebutuhan. Aspek mana yang penting tergantung pada seberapa lengkap kelas diagram yang dimodelkan pada interaction view.

- Class Diagram yang cukup lengkap, **diverifikasi** dengan memodelkan interaction view.
- Class Diagram yang **tidak lengkap, ditingkatkan** dan dilengkapi dengan memodelkan interaction view.

Terdapat hubungan yang dekat antara interaction view dan Use Case. Use Case menunjukkan External View. Sistem TI dipandang sebagai kotak hitam. Dalam interaction view, kotak hitam ini dibuka dan apa yang terjadi di dalam sistem TI dapat dilihat. Interaction view menggambarkan objek mana yang diperlukan untuk memproses tugas tertentu dan bagaimana objek berkomunikasi satu sama lain. UML menggunakan dua jenis diagram untuk memodelkan interaction view yaitu : communication diagram (communication diagram) dan sequence daigram (sequence diagram) . Dalam bagian ini, akan dijelaskan lebih lanjut:



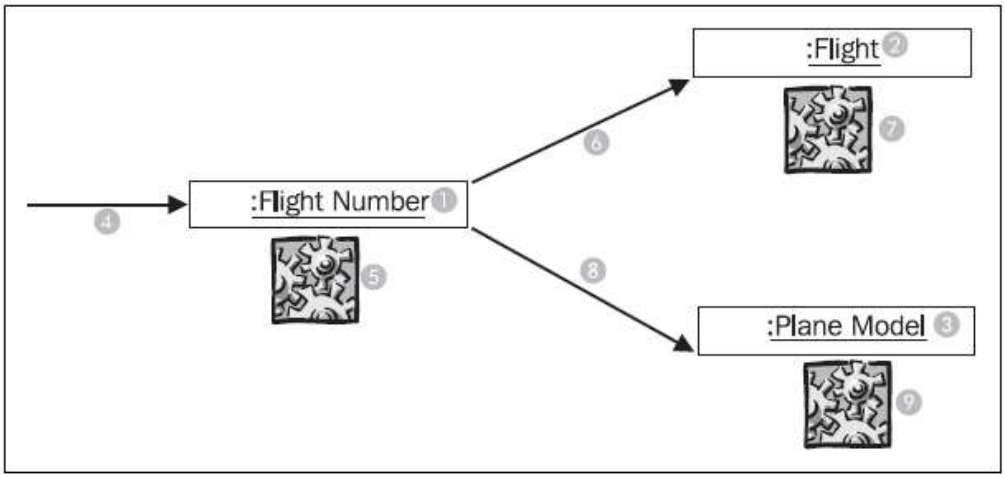
Gambar 113. Hirarki Diagram

Gambar 113 menunjukkan hubungan antara use case dan communication diagram dari interaction view:

- Di atas Anda dapat melihat use case diagram dengan use case **boarding**.
- Use case **boarding** dijelaskan dalam use case sequence diagram. Deskripsi ini terdiri dari serangkaian query event dan mutation event.
- Alur setiap **query event** dijelaskan dalam communication diagram.

Dengan mutation event analogi prosesnya: aliran event telah dijelaskan dalam sequence diagram.

Untuk memahami diagram interaction view dengan benar, dijelaskan bagaimana sistem berorientasi objek bekerja di dalam sistem TI. Suatu sistem berfungsi melalui objek, yang melakukan pekerjaan sendiri atau mendelegasikan pekerjaan ke objek lain. Ini sama dengan bagaimana sistem TI dimodelkan dengan UML. Tidak penting apakah sistem TI diimplementasikan dengan teknologi berorientasi objek atau tidak. Sistem TI dimodelkan pada abstraksi tingkat tinggi, **terputus** dari desain dan pemrograman:



Gambar 114. Kerjasama antar Objek

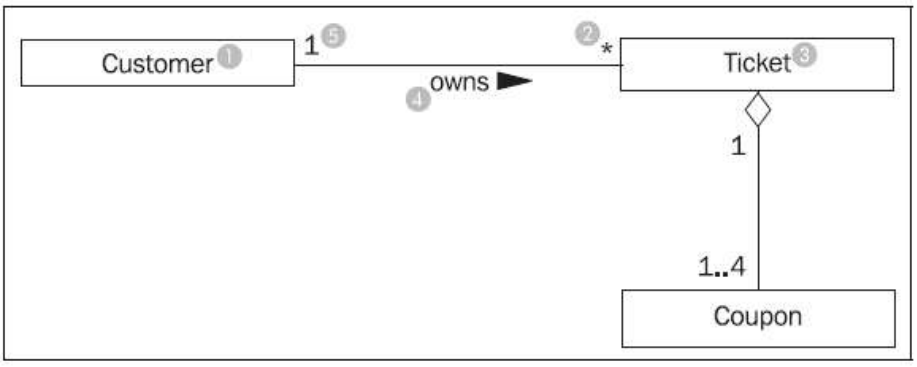
Gambar 114 menampilkan kerjasama antar tiga objek didalam sistem TI: objek **flight number** (1) ,objek flight(2) dan objek plane model(3). Tugasnya adalah delete flight number (misal:SR9011) didalam sistem TI.

Pada model yang sudah dibuat, dari use case, **mutation event-canceling flight number** (4) dikirim ke objek flight number(1). Kemudian objek flight number menjadi aktif(5), sehingga penghapusan dapat dilakukan. Juga dapat dikirim ke (6) (8) melalu event ke objek lain (2) (3) yang butuh untuk diaktifkan(7) (9).

Kerja sama yang tepat antara objek-objek ini didokumentasikan dalam communication diagram dan sequence diagram. Dalam interaction view, fokusnya adalah pada objek yang terlibat dan mengirim event. Apa yang terjadi dalam objek, artinya perilaku tiap

objek (modifikasi nilai atribut, perhitungan, penghapusan objek, dll.), tidak dapat dilihat dalam diagram ini. Perilaku objek didefinisikan dalam behavioral view. Behavioral view menunjukkan untuk setiap objek apa yang sebenarnya terjadi ketika event tertentu mencapai objek itu.

Kerja sama antar objek yang ditunjukkan pada Gambar 11.56 didasarkan pada mekanisme pengiriman event, artinya antar objek dapat saling mengirim event dan sehingga memulai event tertentu. Tetapi, untuk mengirim suatu event ke objek tertentu, objek tersebut harus diketahui. Jika, misalnya, suatu event seharusnya dikirim dari objek ticket ke objek customer yang sesuai, hanya mungkin jika objek ticket tahu apa objek customer. Informasi ini didokumentasikan dalam kelas diagram dari static view:



Gambar 115. Asosiasi (hubungan) dalam Class Diagram

Contoh class diagram pada Gambar 115 menyatakan seorang customer (1) memiliki (4) 0,1 atau beberapa (2) tickets (3) dan sebuah tiket dimiliki hanya tepat 1 (5) customer. Objek yang dihasilkan dari kelas ini berarti:

- Tiap objek **customer** "mengetahui" objek ticket yang diassigned(tugaskan) kepadanya
- Tiap objek **ticket** "mengetahui" objek **coupon** yang diassigned(tugaskan) kepadanya

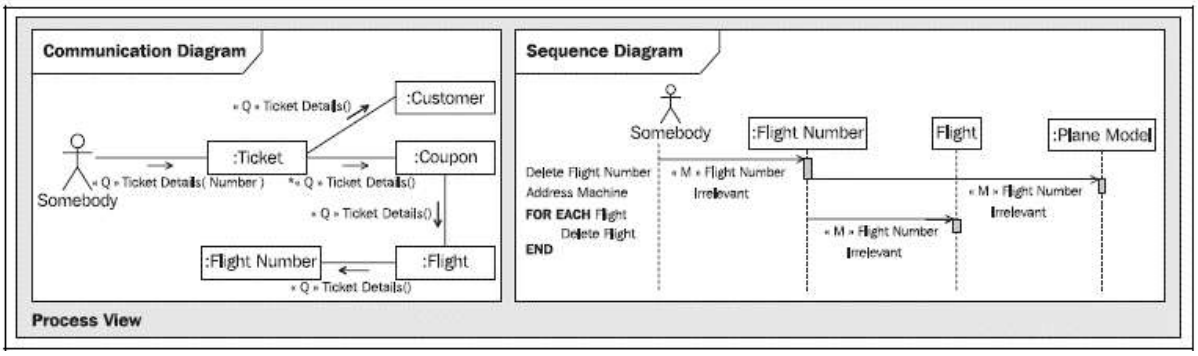
Objek yang terhubung satu sama lain pasti saling mengenal. Ini adalah syarat untuk memungkinkan event dapat dikirim. Biasanya dalam interaction view event dikirim bersama asosiasi dalam kelas diagram, atau dengan melampirkan identifikasi objek ke mana suatu event seharusnya dikirim sebagai parameter event.

Perbedaan communication diagram dan sequence diagram :

- Dalam sequence diagram, memperlihatkan urutan kronologis. Waktu secara vertikal menunjukkan urutan yang jelas dari atas ke bawah. Communication diagram tidak memiliki sumbu waktu ini. Urutan biasanya didokumentasikan dengan penomoran event.
- Communication diagram mirip dengan kelas diagram statis. Nama dan atribut kelas dapat ditampilkan. Ini memungkinkan flow tertentu, seperti membaca informasi, dapat didokumentasikan dengan cara yang sederhana. Dalam sequence diagram, tidak bisa menggambarkan atribut mana yang harus dibaca dari suatu objek.
- Communication diagram berfungsi untuk menunjukkan jalur interaksi paralel. Meskipun sulit untuk melihatnya.

Perbedaan-perbedaan ini adalah alasan di balik pemilihan jenis diagram. **Communication diagram** cocok untuk mendokumentasi **query event**, di mana atribut dibaca dan di mana objek tidak melakukan pekerjaan lebih lanjut. **Sequence diagram** lebih cocok untuk dokumentasi **event mutation**, di mana objek melakukan pekerjaan besar, dan di mana **urutan** lebih penting.

ELEMEN DARI VIEW



Gambar 116. Interaction View

Interaction view pada sebuah sistem TI pada Gambar 116, terdiri dari dua elemen:

Communication diagram mendokumentasikan flow dari **query event** didalam sistem TI. Setiap query mewakili sebuah use case.

Sequence diagram mendokumentasikan flow dari **mutation event** dalam sistem TI yang juga menjadi bagian dari sebuah use case.

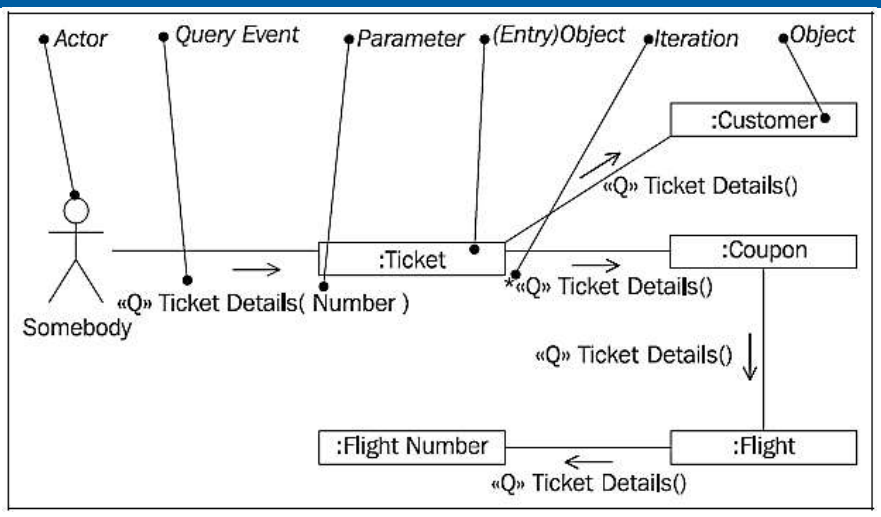
Kedua diagram menunjukkan bagaimana objek didalam sistem TI bekerjasama dalam memproeses event-event.

Setiap query event dari use case menjadi communication diagram dan setiap mutation event menjadi sequence diagram.

Dalam kenyataan, tidak harus mendokumentasikan setiap flow dari query dan mutasi. Memerlukan upaya yang besar. Dokumentasikan flow yang penting atau kompleks saja. Berikut adalah beberapa pertimbangan untuk membantu:

- Untuk setiap query event, verifikasi jika semua kelas, atribut, dan asosiasi yang diperlukan ada dalam kelas diagram. Sederhananya, periksa elemen data yang diperlukan pada hasil cetak (lihat Sub bab, Membangun Communication Diagram). Untuk pertanyaan seperti itu communication digram tidak diperlukan. Langkah kerja ini selanjutnya membantu memverifikasi atau melengkapi kelas diagram.
- Query yang sangat penting bagi user , atau yang sangat kompleks, harus didokumentasikan dalam communication diagram.

COMMUNICATION DIAGRAM

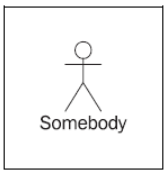


Gambar 117. Elemen dari Communication Diagram

Dalam communication diagram Gambar 117, berikut adalah beberapa elemen didalamnya:

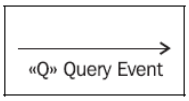
Actor "Somebody"/"Seseorang"

Aktor "Somebody" merepresentasikan aktor apa saja didalam use case diagram. Query event yang ada didalam communication diagram dapat berasal dari beberapa use case, dan use case memiliki aktor yang berbeda, sehingga dituliskan actor "somebody", sehingga kita tidak terikat pada aktor tertentu :



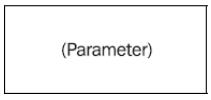
Query Event

Query Event merepresentasikan query dari informasi:normalnya query event dari use case dikirim ke sistem TI, contoh query untuk mendapatkan informasi detil penumpang dari sebuah tiket :



Parameter

Parameter adalah "lampiran" event, contoh:jumlah tiket :

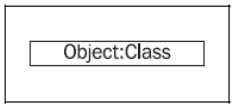


Iteration/Iterasi

Sebuah iterasi mengindikasikan bahwa semua objek didalam asosiasi terjadi event tersebut, misalnya coupon pada sebuah tiket :

Object dan Entry Object

Objek adalah representasi dari sebuah class dalam static view, misal "Brury Sartana", adalah objek dari **class passenger** :



Entry Object adalah objek pertama yang menerima query event dari actor. Pada entry object lah dimulainya alur interaksi.

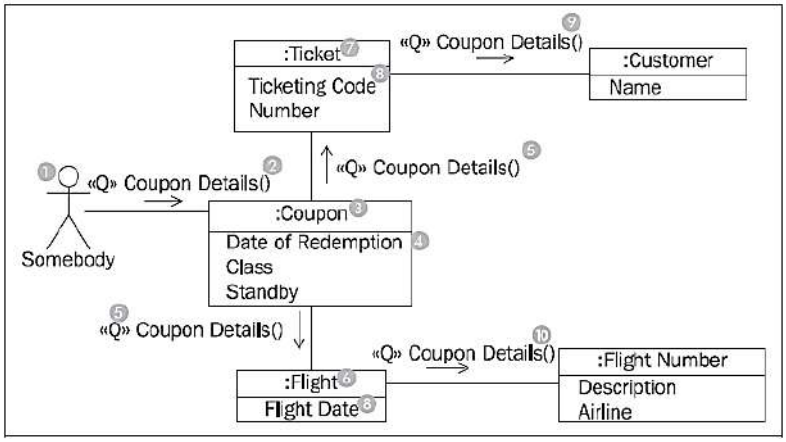
1.36. MEMBACA COMMUNICATION DIAGRAM

Gambar 118 menggambarkan communication diagram dengan actor "somebody" dan objek ticket, customer, coupon flight dan flight number. Diagram mendokumentasikan flow dari query **«Q» coupon details**.

Mulai dari kiri membacanya :

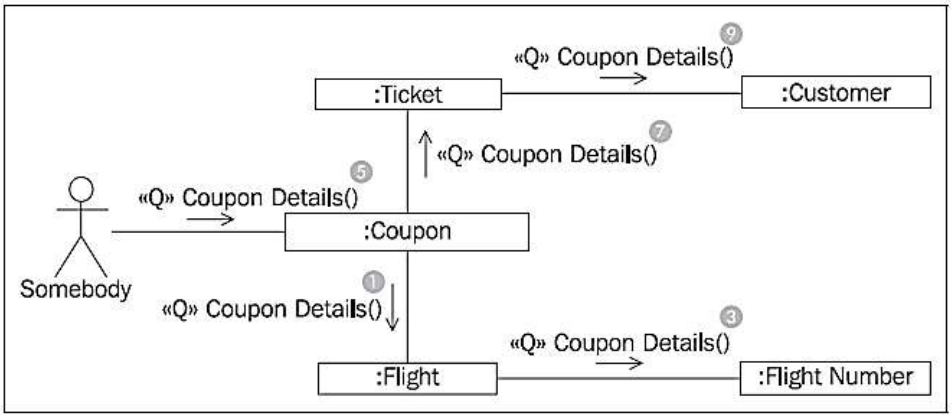
Actor somebody(1) kirim query event **«Q» coupon details**(2) kepada object dalam kelas **coupon**(3).

Dalam model sistem TI, use case adalah sumber event. Apa yang muncul di communication diagram berasal dari use case. Terbukti lebih baik menggunakan actor somebody daripada menggunakan nama actor dari use case, karena event di diagram dapat terjadi pada berbagai use case dengan aktor berbeda. Aktor somebody(1) menggantikan aktor dari usecase saat terjadi query event **«Q» coupon details :**



Gambar 118. Communication Diagram

Objek coupon(3) memilki atribut – date of redemption, class, standby (4)-mengirim query event **«Q» coupon details**(5) kepada dua objek: objek flight(6) dan objek ticket(7). Kedua objek tersebut sebagai balasanya menyediakan beberapa atribut (8) dan mengirim query event **«Q» coupon details** (9,10). Sehingga dapat dikatakan communication diagram dapat digunakan untuk mendokumentasikan "koleksi" dari attribut sebagai reaksi dari query event



Gambar 119. Urutan Dalam Communication Diagram

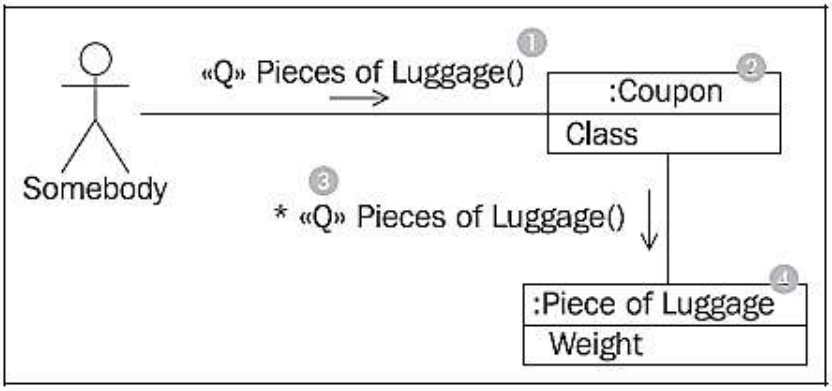
Urutan dalam diagram pada Gambar 119 dapat dinyatakan:

(angka dalam deskripsi sengaja dituliskan untuk menghindari kesalahan urutan):

- Pertama, event dikirim dari aktor somebody ke coupon (5).
- Setelah itu tidak ditentukan urutan:
    - Di satu sisi, event menuju ke flight (1) dan selanjutnya ke (3).
    - Di sisi lain, event tersebut berlaku untuk **ticket**(7) dan selanjutnya ke customer(9).

Event bercabang di kupon, tanpa memperhatikan urutan. Dalam kebanyakan kasus, urutan tetap tidak penting. Namun, jika urutan penting, UML memungkinkan penomoran urutan event dalam communication diagram.

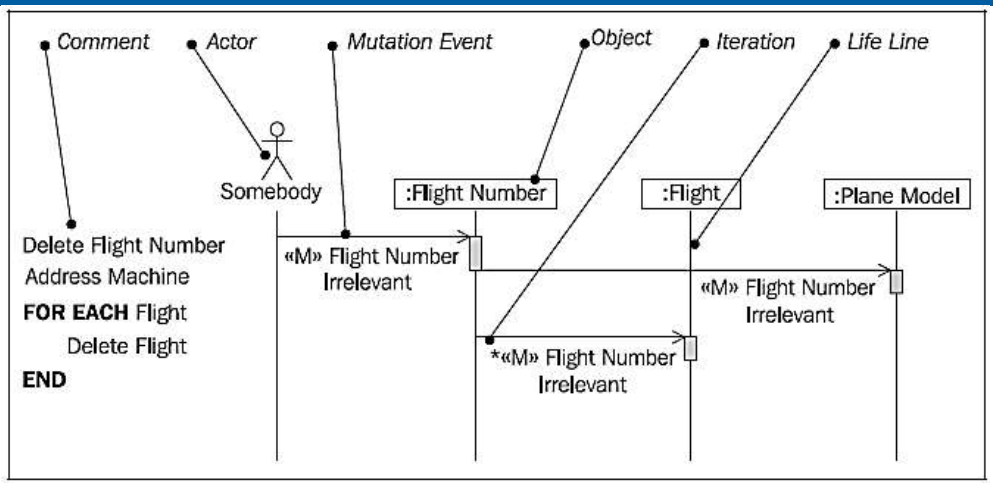
Iterasi menunjukkan bahwa semua objek yang dapat dijangkau dan **bukan hanya satu objek** tertentu yang ditangani:



Gambar 120. Iterasi dalam Communication Diagram

Pada Gambar 120 dapat dinyatakan bahwa query event **«Q» pieces of luggage**(1) pertama dikirim ke objek coupon(2) dan dari sini dikirim ke **semua** (3) (iterasi *) pieces of luggage(4). Tanda asterisk (*) di didepan event menunjukkan iterasi.

SEQUENCE DIAGRAM

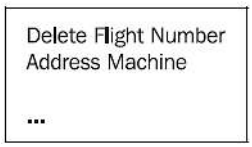


Gambar 121. Elemen dari Sequence Diagram

Sequence Diagram pada Gambar 121 terdapat elemen :

Comment

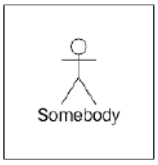
Comment didokumentasikan dengan kombinasi deskripsi teks:



Dalam comment, flow logika ditampilkan di level paling atas.

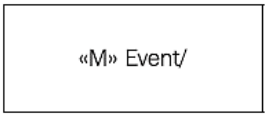
Actor "somebody"

Aktor "somebody" merepresentasikan aktor yang ada di use case diagram. Karena mutation event yang didokumentasikan dalam sequence diagram dapat berisi dari beberapa use case, dan use case juga dapat terdiri dari beberapa aktor, maka dituliskan aktor "somebody" sehingga tidak dituliskan spesifik.



Mutation Event

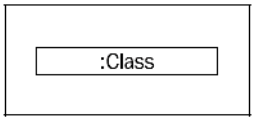
Mutation event adalah event yang dikirim dari use case,normalnya berasal dari user interface kepada sistem TI :



Tujuan dari event ini adalah melakukan mutasi(mutate) seperti create, change atau delete sesuatu didalam sistem.

Object

Objek merepresntasikan objek apa saja, bermakna dan sudah didefinisikan dalam kelas dari sebuah sistem TI.

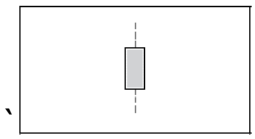


Iteration

Iterasi menginidikasikan bahwa semua objek yang berhubungan menerima event, misalnya seluruh flight dari flight number.

Lifeline

Lifeline mewakili kehidupan(perjalanan waktu). Kotak menunjukkan objek tersebut aktif. Aktivasi tidak terlalu dipentingakan dalam sequence diagram.

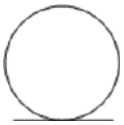


1.37. STEREOTYPE CLASS

Rational Objectory Process menyarankan menemukan class-class dalam sistem yang sedang dibangun dengan mencari class : **boundary, control dan entity**. Ketiga stereotypes ini menggambarkan sebuah sudut pandang **model-view-controller** sehingga membuat analis dapat **membagi sistem** dengan memisahkan sudut pandang dari domain dari control yang dibutuhkan oleh sistem.

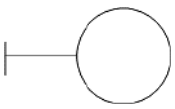
Berikut penjelasan dari masing-masing stereotype :

Entity Class



Entity class : Menggambarkan informasi yang **harus disimpan oleh sistem** (struktur data dari sebuah sistem)

Boundary Class

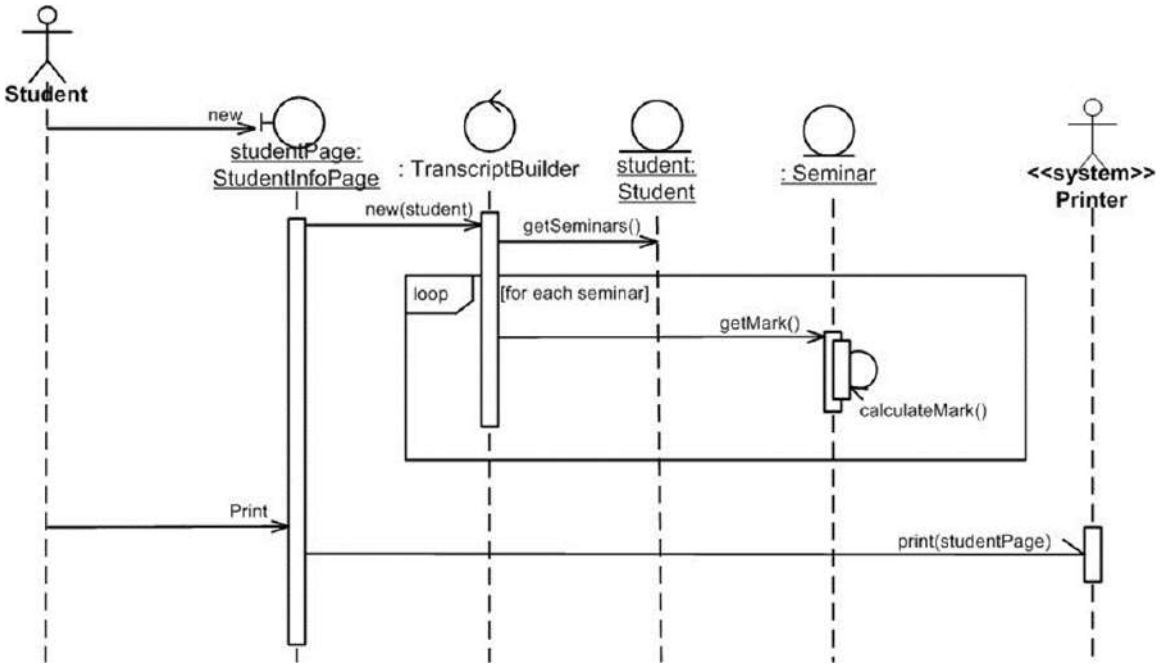


Boundary class Mengambarkan **interaksi antara satu atau lebih actor dengan sistem**, memodelkan bagian dari sistem yang bergantung pada pihak lain disekitarnya dan merupakan pembatas sistem dengan dunia luar.

Control Class

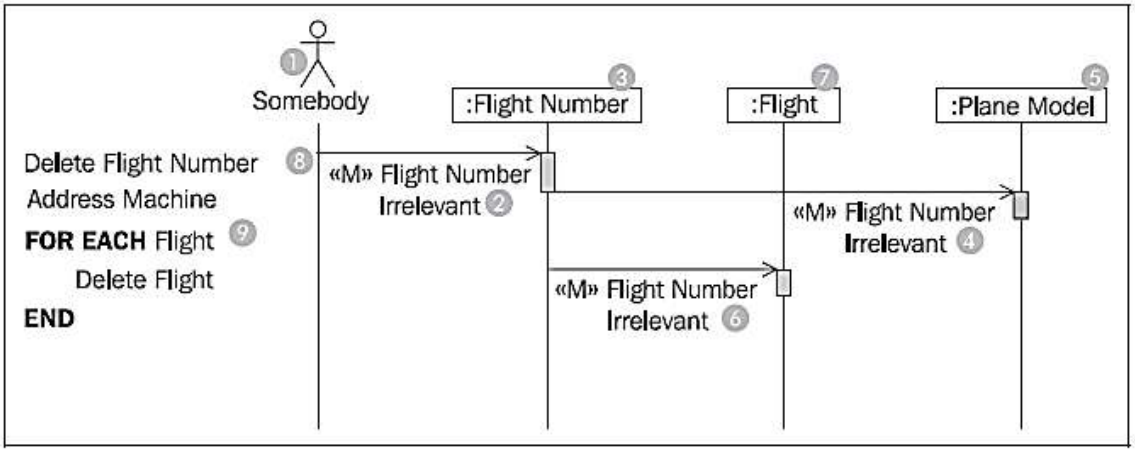
Control class memodelkan urutan perilaku (behavior) khusus untuk satu atau lebih use case. Pada awal phase, sebuah control class ditambahkan untuk setiap pasangan actor atau use case. Control class bertanggung jawab untuk event-event dalam use case.

Contoh :



Gambar 122. Sequence Diagram dengan Stereotype Class

1.38. MEMBACA SEQUENCE DIAGRAM



Gambar 123. Sequence Diagram

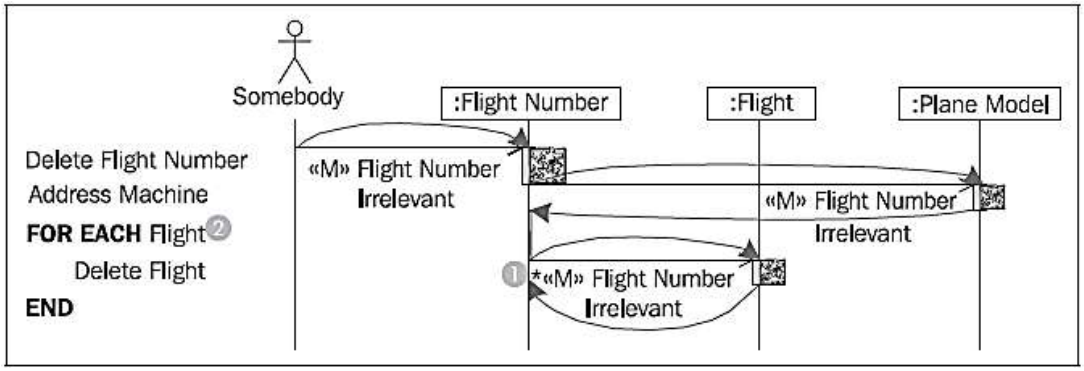
Diagram dibaca dari atas kebawah. Flow start dari **actor** (1) mengirim mutation event **«M» flight number irrelevant**(2) ke objek **flight number** (3) . (Ingat :Mutation event berasal dari use case). Actor(1) diambil dari use case.

Bagaimana event di proses di objek flight number tidak terlihat di sequence diagram. Tapi bisa diberikan catatan di comment(8). Deskripsi bagaimana diproses ada di statechart diagram (lihat sub bag *The Behavioral View*) dari class flight number.

Selanjutnya, objek dari kelas **flight number**(3) mengirim mutation event **«M» flight number irrelevant** (4) kepada objek dari class **plane model**(5). Pemrosesan event selesai pada objek plane model(5), dan kontrol kembali ke event pengirim ke objek **flight number** (3). Setelah pemrosesan selesai , tidak ada panah untuk event "reply/balasan".

Akhirnya, mutation event **«M» flight number irrelevant** (6) dikirim ke objek **flight** (7). Karena mungkin saja objek **flight number** mengetahui banyak **objek flight** (informasi diambil dari kelas diagram dari static view), event mutation dikirim ke semua objek flight dari objek flight number. Iterasi tanda bintang * pada event(1) dalam sequence diagram (Gambar 123) menandai proses ini.

Namun, disarankan untuk membuat komentar tambahan di kiri (2), agar diagram lebih mudah dibaca:



Gambar 124. Aliran Kontrol dari Sequence Diagram

Gambar 124 menunjukkan aliran kontrol dalam sequence diagram. Biasanya sequece diagram hanya digunakan untuk mendokumentasikan mutation event. Agar kita dapat

mengkomunikasikan aspek-aspek penting dari interaction view dengan penggunaan sequence diagram ini.

MEMBANGUN COMMUNICATION DIAGRAM

Checklist berikut menunjukkan langkah-langkah yang diperlukan untuk membuat communication diagram tiap query event dari use case.

Daftar Periksa Menyusun Communication Diagram dalam Interaction View:

- Draft hasil query — Apa yang kita inginkan?
- Identifikasi kelas yang terlibat — Kelas mana yang dibutuhkan?
- Tetapkan objek awal — Di mana kita mulai?
- Rancang jalur event(path)— Ke mana kita pergi?
- Ubah event path— Tepatnya objek apa yang kita butuhkan?
- Identifikasi atribut yang diperlukan — Apa sebenarnya yang ingin kita ketahui?
- Verifikasi communication diagram— Apakah semuanya benar?

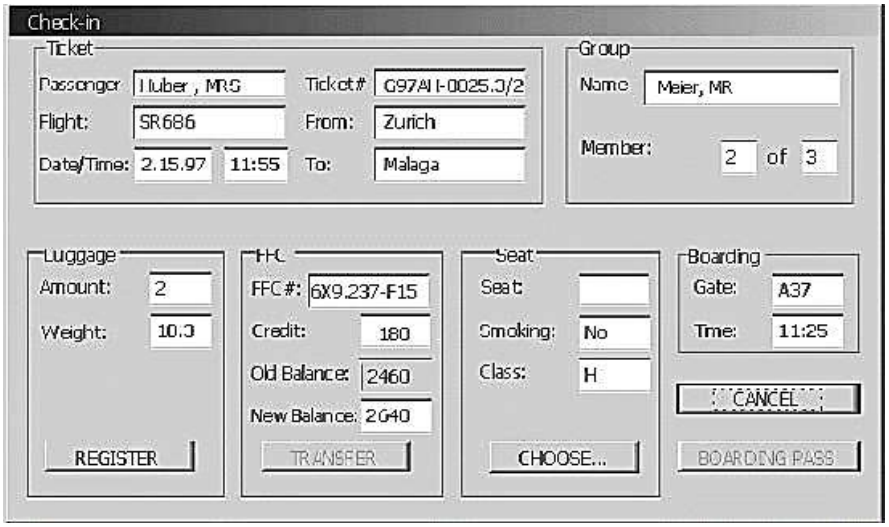
HASIL DRAF QUERY — APA YANG KITA INGINKAN?

Titik awal untuk memodelkan query event adalah hasil yang diharapkan.

Secara umum, hasil yang diinginkan adalah tampilan pada monitor atau dokumen cetak, misalnya, laporan atau tanda terima. Gambar 125 menunjukkan layar dari prototipe sistem TI, dan Gambar 126 bagian dari boarding pass.

Untuk setiap query event yang akan didokumentasikan dalam communication diagram, kita perlu mengetahui seperti apa hasilnya. Pertanyaan-pertanyaan berikut harus ditanyakan:

- Bagaimana (tepatnya) hasil dari suatu query seharusnya terlihat?
- Elemen mana yang harus berisi hasil; yang mana dari mereka adalah teks-teks tetap dan mana dari mereka adalah elemen data?



Gambar 125. Prototipe Interface dari Sistem Check-in Passenger

Sebaiknya gunakan sketsa hasil/keluaran. Untuk setiap elemen sketsa, harus menunjukkan apakah menyangkut informasi dari sistem TI atau ada elemen yang tetap, misalnya, gambar atau field label. Dalam bentuk layar, field label biasanya tidak berasal dari informasi dalam sistem TI. Namun, jika sistem TI mendukung beberapa bahasa, ada kemungkinan field label diambil dari informasi dalam sistem TI. Contoh lain adalah boarding pass, yang menggunakan formulir pre printing (sudah dicetak sebelumnya), berarti field label sudah ada. Untuk melengkapi harus membaca informasi dari sistem TI. PadaGambar 126, field dari boarding pass dilingkari yang menerima informasi dari sistem TI.

Boarding pass dari studi kasus berisi elemen data berikut:

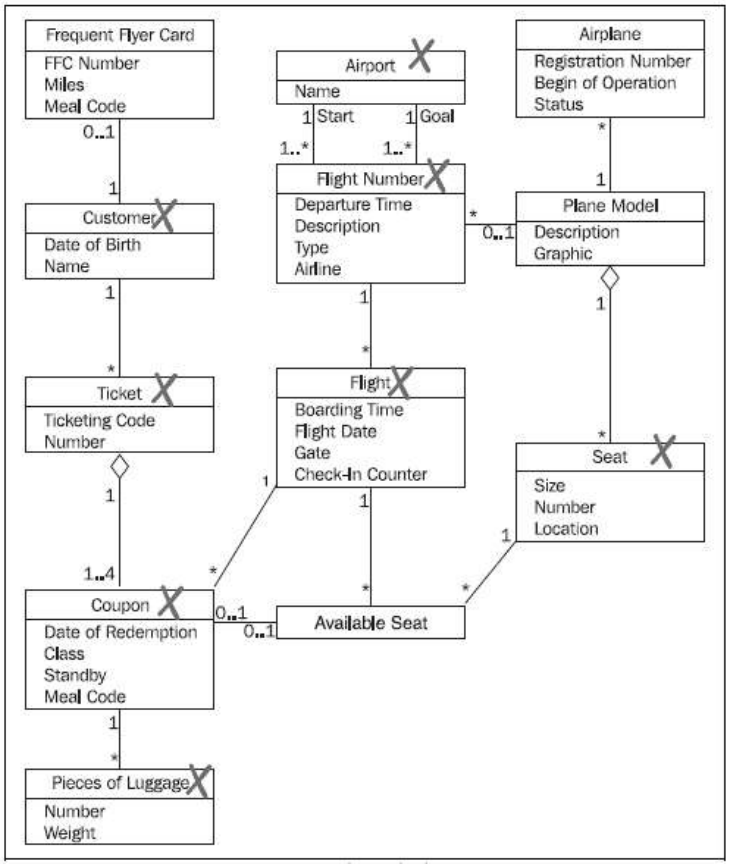
EKONOMI, GRAESSLE, PATRIC, MR, ZURICH, HERAKLION, EDW761, Y, 29MAY0745, B38, 0715, NO, 2, dan 34.

Tujuan dari pemodelan communication diagram adalah untuk menunjukkan bagaimana nilai-nilai yang ada dapat diambil dari kelas diagram.

Gambar 126. Bagian dari Boarding Pass

IDENTIFKASI CLASS YANG TERLIBAT- CLASS APA YANG KITA BUTUHKAN ?

Dari hasil sketsa, kita dapat menentukan class yang dibutuhkan yang ada pada class diagram.



Gambar 127. Class Diagram yang Ditandai (X)

Pertanyaannya adalah:

- Dari kelas mana informasi seharusnya didapat? Kita harus tentukan dari atribut dan kelas mana setiap elemen informasi dalam sketsa.
- Kelas mana yang dibutuhkan untuk jalur akses? Kelas diperlukan berdasarkan atribut mereka, yang diperlukan untuk menghasilkan hasil query, atau berdasarkan hubungan nya dengan kelas lain.
- Kelas mana yang tidak ada dalam kelas diagram? Tergantung pada tingkat penyelesaian kelas diagram, bisa saja kelas tambahan baru harus dimodelkan.

Dalam draf pertama, kelas-kelas yang dibutuhkan dapat ditandai dengan tulisan tangan pada cetakan kelas diagram. Pada Gambar 127, kelas yang diperlukan untuk menghasilkan boarding pass dari Gambar 126 yang ditandai.

TETAPKAN AWAL OBJEK (INITIAL OBJECT)— DI MANA KITA MULAI?

Query dimulai dengan initial objek. Initial objek dapat berupa kelas (dalam fungsinya sebagai set) atau objek tertentu dari suatu kelas. Jika objek tertentu ditangani secara langsung, misalnya, objek pelanggan tertentu, atau objek tiket tertentu, objek ini harus diketahui sebelum query event dikirim. Jika kelas telah ditentukan, parameter dapat dilampirkan ke query sebagai parameter pemilihan, sehingga objek tertentu dapat dipilih. Semua kelas yang dibutuhkan lebih lanjut harus dapat diakses dari objek awal melalui koneksi.

Pertanyaan adalah:

- Obyek mana yang pertama kali dikunjungi?
- Objek mana yang sudah kita ketahui?
- Di mana kita mulai mengumpulkan informasi?

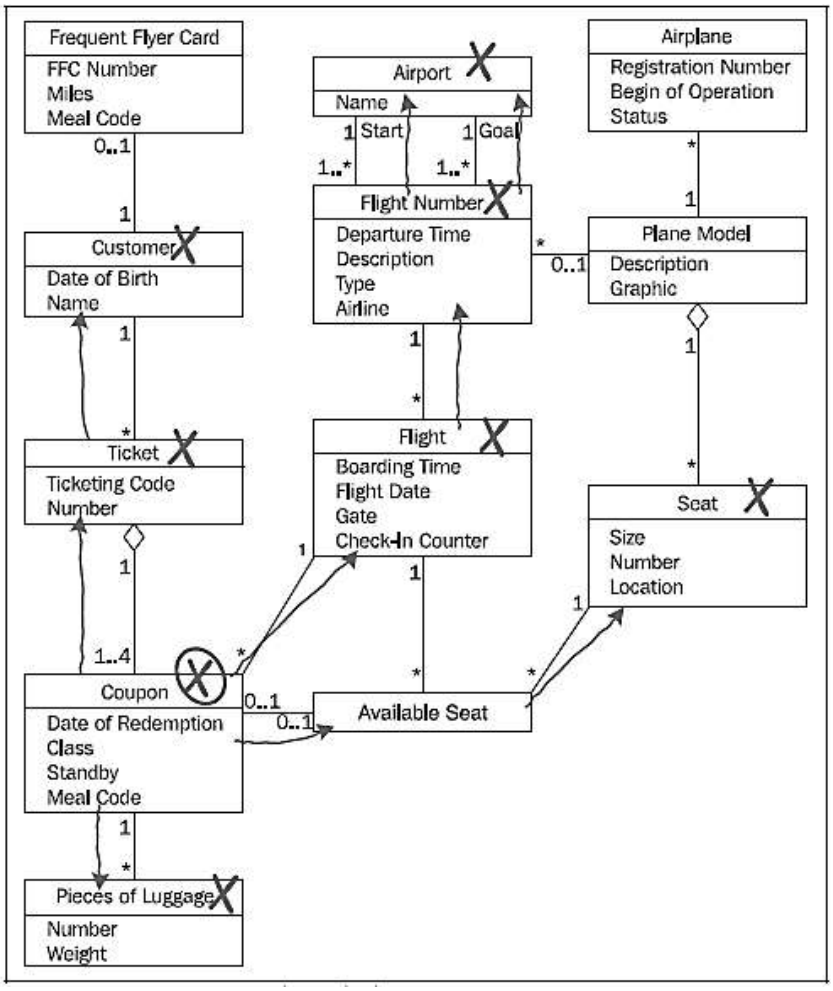
Untuk menghasilkan boarding pass dalam studi kasus ini, kita mulai dengan kupon tiket pesawat yang diinginkan untuk menghasilkan boarding pass.

RANCANG EVENT PATH — KE MANA KITA PERGI?

Mulai dari initial objek, kita tentukan jalur di mana objek yang dibutuhkan dapat dicapai. Pertanyaannya:

- Melalui jalur mana saya dapat menjangkau semua objek yang dibutuhkan dalam kelas diagram?

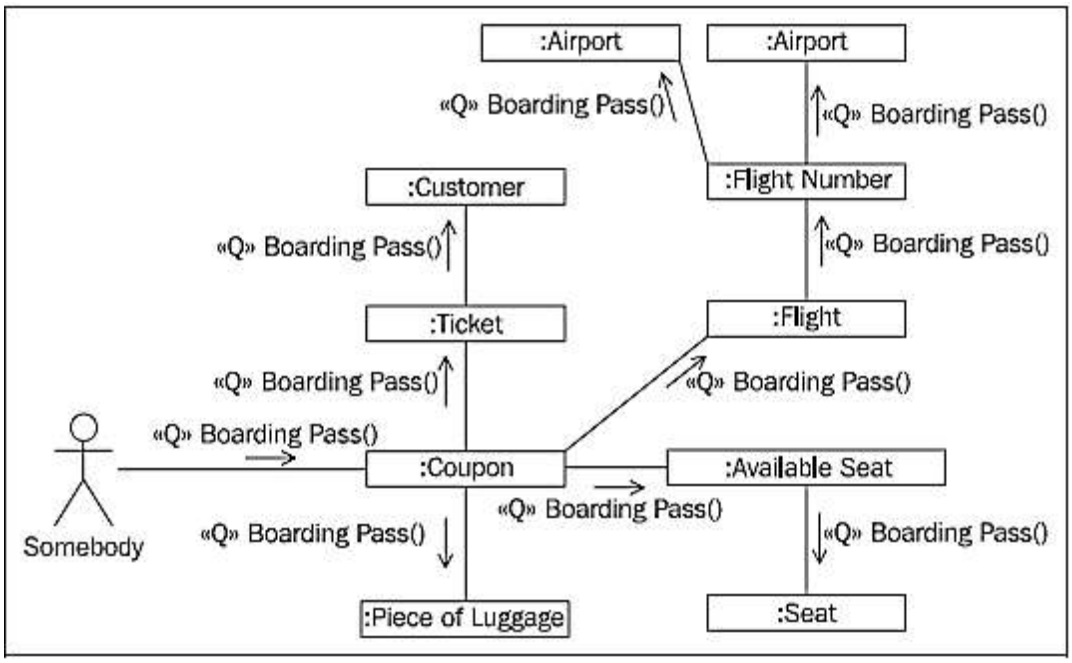
Untuk draft awal bisa saja jalur ini ditandai dengan tangan pada kelas diagram. Gambar 128 menunjukkan diagram kelas yang telah ditandai untuk query event yang menghasilkan boarding pass:



Gambar 128. Class Diagram Sederhana dengan Event Path

Dalam beberapa kasus, cukup untuk menandai event path dengan coretan pada kelas diagram. Untuk ini kita dapat dilakukan tanpa memodelkan communication diagram. Namun, untuk query yang kompleks atau penting disarankan menggunakan communication diagram.

Anda menghasilkan communication diagram untuk query, berdasarkan kelas diagram yang ditandai ini. Semua kelas yang ditandai ditransfer ke ini, dan event kueri dimasukkan di sepanjang event path. Gambar 129 menunjukkan hasil langkah ini:



Gambar 129. (Draft awal) Communication Diagram

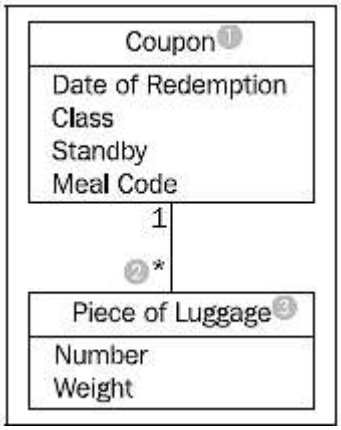
UBAH EVENT PATH — APA OBJEK YANG KITA BUTUHKAN?

Jalur event harus diisi dengan pilihan dan iterasi.

Pilihan dan iterasi digunakan jika sepanjang jalur beberapa objek dari satu kelas dapat dicapai dari kelas lain, yang berarti, ketika hubungan multiplicity dalam kelas diagram memiliki batas atas **lebih besar dari 1 (biasanya *)**. Dalam kasus seperti itu, kita perlu menyatakan apakah semua objek harus diulang, atau setiap objek harus dipilih berdasarkan kriteria tertentu.

Pertanyaannya adalah:

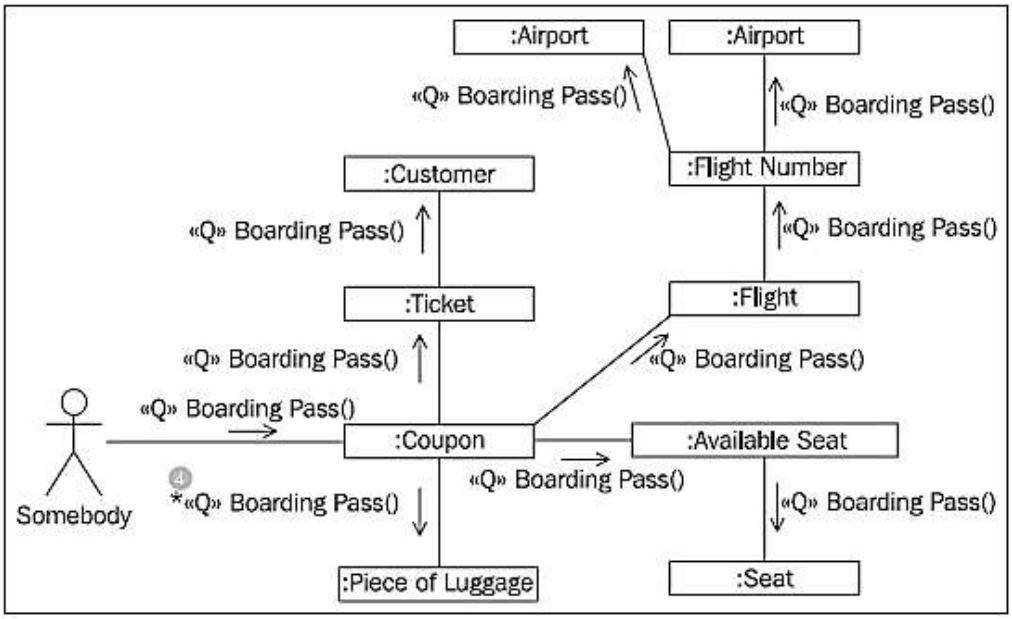
- Jika saya menemukan lebih dari satu objek di sepanjang jalan saya, apakah saya kemudian membutuhkan semuanya (iterasi), atau apakah tertentu saja (diseleksi)?



Gambar 130. Seleksi atau Iterasi?

Dalam studi kasus, pada Gambar 130 kita mungkin menemukan beberapa (2) piece of luggage-jumlah bagasi (3) di sepanjang event path yang berasal dari coupon (1). Kita membutuhkan semua barang bawaan untuk boarding pass, karena kita ingin mencetak jumlah dan berat totalnya. Karena itu, kita harus melakukan iterasi.

Hal ini didokumentasikan pada communication diagram Gambar 131 ditandai tanda bintang (tidak mencolok) dari nama event(4):



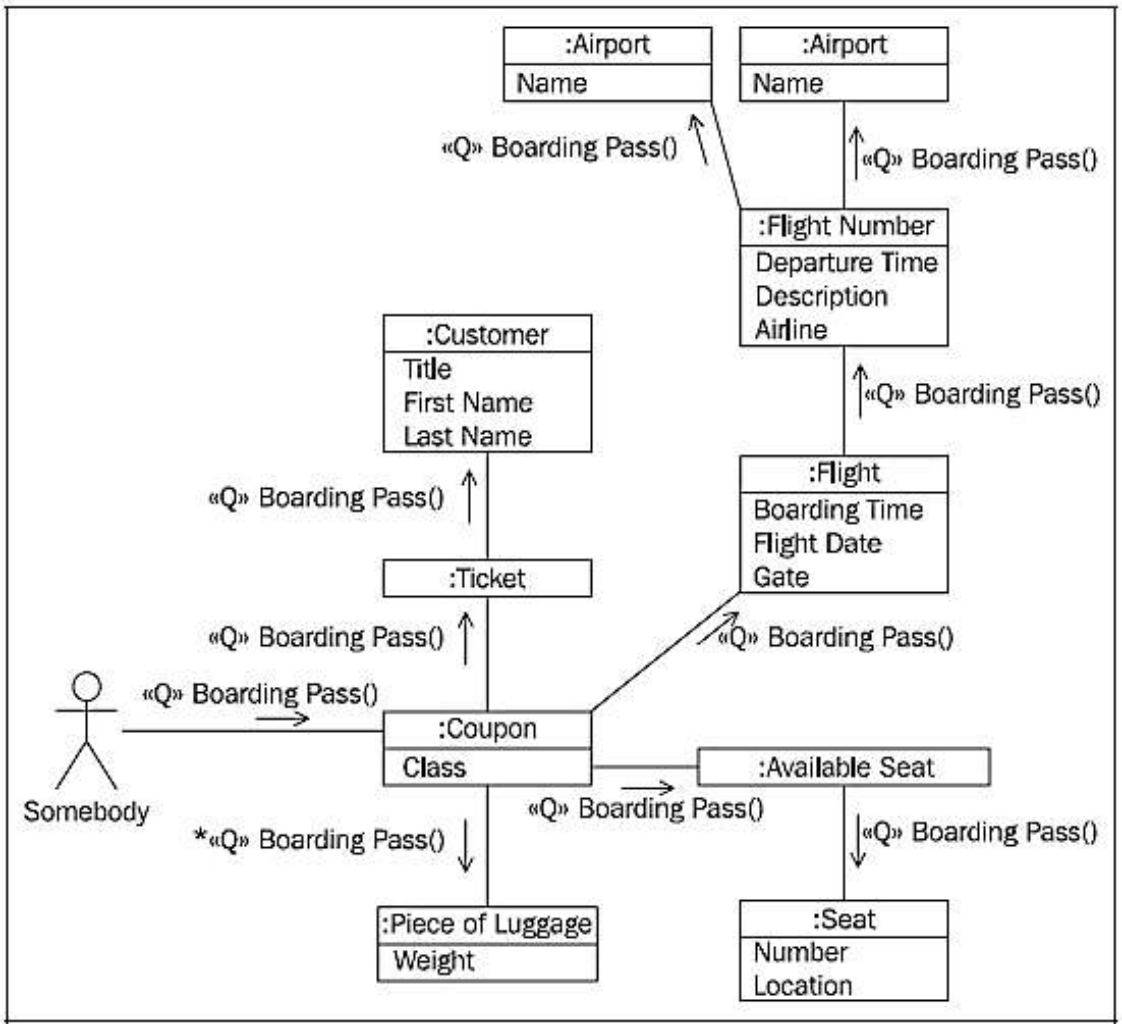
Gambar 131. Communication Diagram Dengan Iterasi

IDENTIFIKASI ATRIBUT YANG DIPERLUKAN — APA YANG SEBENARNYA INGIN KITA KETAHUI?

Langkah terakhir, kita mendokumentasikan atribut yang diperlukan untuk menjawab pertanyaan:

- Atribut mana yang diperlukan untuk menghasilkan query?
- Atribut mana yang tidak ada didalam kelas diagram?

Akhirnya, semua elemen data dari hasil query yang disusun harus dapat dilacak kembali ke atribut dalam kelas diagram dari static view. Dalam kasus sederhana, ini dapat dilakukan, sekali lagi, dengan menandai atribut pada cetakan diagram kelas. Ketika communication diagram telah dimodelkan, atribut dapat dimasukkan (Gambar 132):



Gambar 132. Communicaiton Diagram dengan Attribut

VERIFIKASI COMMUNICATION DIAGRAM— APAKAH SUDAH BENAR?

Communication diagram yang lengkap dapat diverifikasi dengan checklist berikut:

Checklist Memverifikasi Communication Diagram dalam Interaction View:

- Dapatkah hasil query yang disusun dapat dimasukkan ke communication diagram?
- Apakah event path dalam communication diagram mengalir di sepanjang asosiasi dalam kelas diagram?
- Apakah selalu dibutuhkan kesesuaian dengan kelas diagram, jika berurusan dengan iterasi atau seleksi?
- Jika jawaban untuk semua pertanyaan ini adalah ya, sebagian besar kemungkinan sumber kesalahan telah dihilangkan.

MEMBANGUN SEQUENCE DIAGRAM

Checklist berikut menunjukkan langkah-langkah yang diperlukan untuk membangun sequence diagram setiap mutation event dari use case. Selanjutnya, akan dijelaskan lebih lanjut tiap langkah-langkah:

Checklist Membangun sequence diagram di Interaction View:

- Identifikasi kelas yang terlibat — Apa yang dipengaruhi oleh mutation event?
- Menentukan objek awal — Ke mana perginya event mutation?
- Ajukan event— Bagaimana event mutation diteruskan?
- Tentukan parameter event— Apa yang harus diketahui objek?
- Periksa sequence diagram — Apakah semuanya benar?

IDENTIFIKASI KELAS YANG TERLIBAT — APA YANG DIPENGARUHI OLEH EVENT MUTATION?

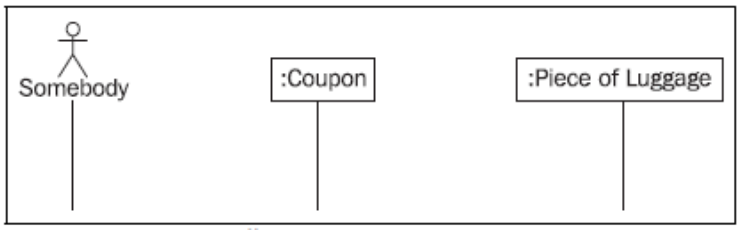
Kelas-kelas yang mempengaruhi event mutation harus diidentifikasi. Berdasarkan statechart diagram (lihat Bagian *The Behavioral View*). Pertanyaannya adalah:

- Kelas mana yang sudah dipengaruhi oleh event mutation tertentu? Untuk menjawab pertanyaan ini, harus melihat statechart diagram mana yang berisi event mutation. Jika event mutation ada dalam statechart diagram suatu kelas,

kelas ini dipengaruhi oleh event mutation, berarti event mutation harus dikirim ke kelas ini.

- Kelas lain mana yang dipengaruhi oleh mutasi? Bisa jadi ada kelas yang dipengaruhi oleh event mutation, tetapi belum ada dalam statechart diagram.

Pertanyaan pertama bisa dijawab dengan mudah. Sebagian besar alat CASE (Computer Aided Software Engineer) dapat membuatnya, misalnya, daftar kelas yang sudah terpengaruh. Untuk menemukan kelas yang terpengaruh lebih lanjut, lihat diagram kelas dan pikirkan apakah sesuatu harus terjadi dengan objek dari setiap kelas kapan event mutation terjadi. Paling tidak, harus melihat lagi kelas-kelas yang terletak dekat dengan beberapa kelas yang sudah terpengaruh pada kelas diagram. Tentu saja, jika kelas tambahan ditemukan, diagram statechart harus diperbarui dengan memasukkan mutation event. Pada kasus ini, **mutation «M» record piece of luggage** event terpengaruh terhadap class **coupon** dan **piece of luggage** seperti Gambar 133



Gambar 133. Class yang terpengaruh dalam Sequence Diagram

TENTUKAN INITIAL OBJEK— KEMANA KEJADIAN MUTASI LEBIH DULU?

Mutasi dimulai dengan initial objek. Objek awal dapat berupa kelas (dalam fungsinya sebagai set) atau objek tertentu dari suatu kelas. Jika objek tertentu ditentukan, misalnya, objek flight tertentu atau objek tiket tertentu, objek ini harus diketahui sebelum mutation event dikirim. Jika kelas yang ditentukan, parameter dapat dilampirkan ke mutation event sebagai parameter pemilihan, sehingga objek tertentu dapat dipilih. Semua kelas yang dibutuhkan lebih lanjut harus dapat diakses dari objek awal melalui hubungan.

Pertanyaan:

- Ke mana event dari objek pertama harus pergi?

- Objek mana yang sudah saya ketahui?
- Di mana saya mulai mengumpulkan informasi?

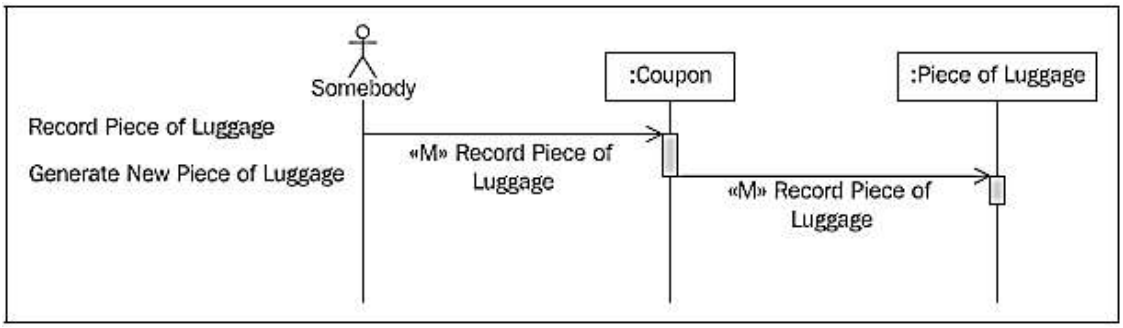
Dalam studi kasus ini, selama mutation event **«M» record piece of luggage**, objek coupon sudah diketahui, karena seluruh proses check-in terjadi pada coupon tertentu. Sebagian objek **new baggage**(bagasi baru) yang akan direkam belum ada. Oleh karena itu, objek awal adalah objek **coupon**.

PROPAGASI EVENT— BAGAIMANA MUTATION EVENT DITERUSKAN?

Mulai dari initial objek, urutan ditentukan, di mana objek yang terpengaruh menerima mutation event. Pertanyaannya:

- Dalam urutan manakah saya menjangkau semua objek yang terpengaruh dalam kelas diagram?

Event mutation diteruskan sepanjang hubungan dalam kelas diagram ke semua kelas yang dipengaruhi oleh event mutation. Dalam diagram kelas dari studi kasus ada asosiasi dari coupon ke piece of luggage, di mana mutation event dapat dikirim, seperti yang diilustrasikan dalam Gambar 134:



Gambar 134. Event Dalam Sequence Diagram

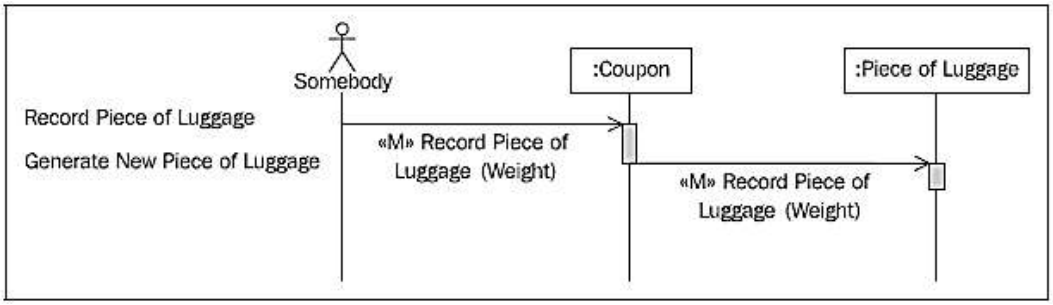
TENTUKAN PARAMETER EVENT— APA YANG HARUS DIKETAHUI OBJEK?

Informasi yang diperlukan untuk memproses peristiwa mutasi diteruskan sebagai parameter peristiwa. Pertanyaannya adalah:

- Informasi apa yang dibutuhkan benda untuk memproses mutasi?

Untuk menghasilkan objek bagasi baru dalam studi kasus ini, dibutuhkan beratnya (weight).

Oleh karena itu, weight akan diteruskan sebagai parameter peristiwa, seperti yang ditunjukkan pada Gambar 135:



Gambar 135. Event Parameter Dalam Sequence Diagram

VERIFIKASI SEQUENCE DIAGRAM— SEMUA SUDAH BENAR?

Sequence diagram yang lengkap dapat diverifikasi dengan cheklist berikut:

Daftar Periksa Memverifikasi Sequence Diagram dalam Interaction View:

- Apakah semua kelas dipengaruhi oleh event mutation yang ada di sequence diagram?
- Apakah event mutation diteruskan bersama asosiasi dalam kelas diagram?

Jika jawaban untuk kedua pertanyaan ini adalah ya, sumber kesalahan terbesar telah dihilangkan.

KESIMPULAN

SOAL LATIHAN

UNIVERSITAS BUDI LUHUR
FAKULTAS TEKNOLOGI INFORMASI

MODUL PERKULIAHAN #13
COMPONENT DAN DEPLOYMENT DIAGRAM

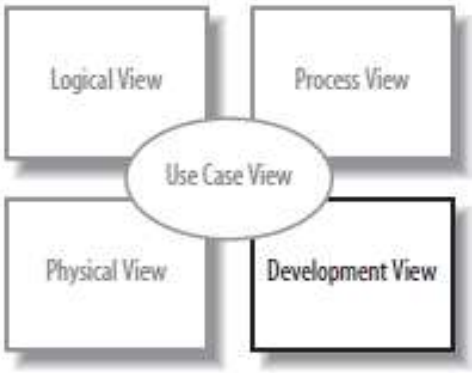
Capaian Pembelajaran	:	Mahasiswa memahami dan menggunakan component dan deployment diagram
Sub Pokok Bahasan	:	1.65. Mengatur dan Reuse Bagian Sistem: Component Diagram a. Apa Itu Component b. Component Dasar UML c. Interface Component Yang Tersedia (Provided) dan Diperlukan (Required) d. Menggambarkan Kerjasama Component e. Kelas Yang Membangun (Realize) Component f. Port dan Struktur Internal g. Blackbox dan White-Box Component View 1.66. Deployment Diagram a. Men-Deploy Sistem Sederhana b. Deployed Software: Artifact c. Apa Itu Node? d. Spesifikasi Deployment e. Kapan Menggunakan Deployment Diagram

		f. Contoh Deployment Diagram 1.67. Kesimpulan 1.68. Latihan Soal
Daftar Pustaka	:	Miles, R. and Hamilton, K., 2006. Learning UML 2.0. " O'Reilly Media, Inc.".

MENGATUR DAN REUSE BAGIAN SISTEM: COMPONENT DIAGRAM

Saat merancang sistem sofware, jarang dijumpai dari persyaratan lompat langsung ke mendefinisikan kelas di sistem. Akan sangat membantu untuk merencanakan sistem tingkat tinggi untuk membangun arsitektur dan mengelola kompleksitas dan ketergantungan di antara bagian-bagian. Component digunakan untuk mengatur sistem menjadi bagian-bagian sofware yang dapat dikelola, digunakan kembali, dan dapat ditukar.

Component Diagram UML memodelkan component di sistem dan sebagai bagian dari development view, seperti yang ditunjukkan pada Gambar 5-1. Development view menjelaskan bagaimana bagian-bagian sistem disusun dalam modul dan component dan sangat membantu mengelola lapisan dalam arsitektur sistem.



Gambar 136. Development View dari Model Bagaimana Bagian Sistem Diorganisasikan Menjadi Modul dan Komponen

1.39. APA ITU COMPONENT

Ingat pembahasan tentang enkapsulasi (peng-kapsul-an) yang merupakan karakteristik terpenting dari kelas dan berorientasi objek. Pada pendekatan berorientasi objek untuk pengembangan sistem, sesuatu dikatakan menjadi objek perlu memuat data(atribut) dan instruksi yang mempengaruhi data (operasi).

Component adalah bagian yang dienkapsulasi, dapat digunakan kembali, dan dapat diganti dari sofware kita. Component sebagai building block: menggabungkannya agar dapat bekerjasama (membangun component secara bertahap mulai dari kecil kebesar)

untuk membangun sofware. Karena itu, component itu bisa dari ukuran relatif kecil, (ukuran kelas), hingga sebuah subsistem besar.

Kandidat untuk component adalah bagian yang melakukan fungsi utama dan sering digunakan di seluruh sistem. Sofware, seperti log file, parser , XML, atau online shopping chart, adalah component yang mungkin sering kita jumpai. Ini adalah contoh component umum yang berasal dari pihak ketiga, tetapi prinsip yang sama berlaku untuk component yang kita buat sendiri.

Di sistem, kita dapat membuat component yang menyediakan layanan atau akses ke data. Misalnya, dalam CMS dapat memiliki component manajemen konversi yang mengubah blog ke format yang berbeda, seperti RSS Feed. RSS Feed biasanya digunakan untuk menyediakan pembaruan berformat XML untuk konten online (seperti blog).

Dalam UML, sebuah component dapat melakukan hal yang sama seperti yang dilakukan kelas: menggeneralisasi dan berasosiasi dengan kelas dan component lain, mengimplementasikan interface, menjalankan operasi, dan sebagainya. Lebih jauh, seperti halnya composite structure, component dapat memiliki port dan memperlihatkan struktur internal. Perbedaan utama antara kelas dan component adalah bahwa component umumnya memiliki tanggung jawab yang lebih besar daripada kelas. Misalnya, Kita dapat membuat kelas informasi pengguna yang berisi informasi kontak pengguna (nama dan alamat emailnya) dan component manajemen pengguna yang memungkinkan akun pengguna dibuat dan diperiksa kebenarannya. Selain itu, sangat umum sebuah component berisi dan menggunakan kelas atau component lain untuk melakukan tugasnya.

Karena component adalah pemain utama dalam desain sofware, penting agar sebuah component dapat berdiri sendiri (tingkat ketergantungan/coupling) antar komponen renda, sehingga bila terjadi perubahan pada component tidak mempengaruhi sistem secara keseluruhan. Agar dapat mudah digunakan, kopling dan enkapsulasi dibuat loose(direndahkan) dan component diakses melalui interface. Dengan mengizinkan

component untuk saling mengakses melalui interface, mengurangi kemungkinan perubahan dalam satu component akan menyebabkan terjadi kerusakan di seluruh sistem.

1.40. COMPONENT DASAR UML

Component berbentuk persegi panjang dengan stereotype <<component>> dan ikon persegi panjang tab opsional di sudut kanan atas. Gambar dibawah menunjukkan component *ConversionManagement* yang digunakan dalam CMS yang mengubah blog ke berbagai format dan menyediakan feed seperti RSS feed.

1.41. INTERFACE COMPONENT YANG TERSEDIA (PROVIDED) DAN DIPERLUKAN (REQUIRED)

Karena component perlu digabungkan secara longgar sehingga dapat diubah tanpa memaksakan perubahan pada bagian lain dari sistem, di sinilah terjadi interface. Component berinteraksi satu sama lain melalui interface yang disediakan dan diperlukan untuk mengontrol ketergantungan antar component dan membuat component dapat ditukar.

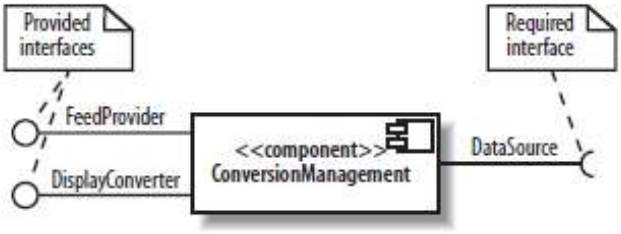
Provided Interface Component adalah realisasi interface component. Component dan kelas lain berinteraksi dengan component melalui interface yang disediakan. Interface yang disediakan dalam component menggambarkan layanan yang disiapkan oleh component.

Required Interface Component adalah interface dimana agar component berfungsi, component membutuhkan kelas lain. Component ini tetap mengakses kelas atau component melalui interface yang diperlukan. Interface yang diperlukan menyatakan service yang dibutuhkan component.

Ada tiga cara untuk menampilkan interface yang disediakan dan diperlukan di UML: notasi *simbol ball and socket*, *stereotype*, dan daftar teks.

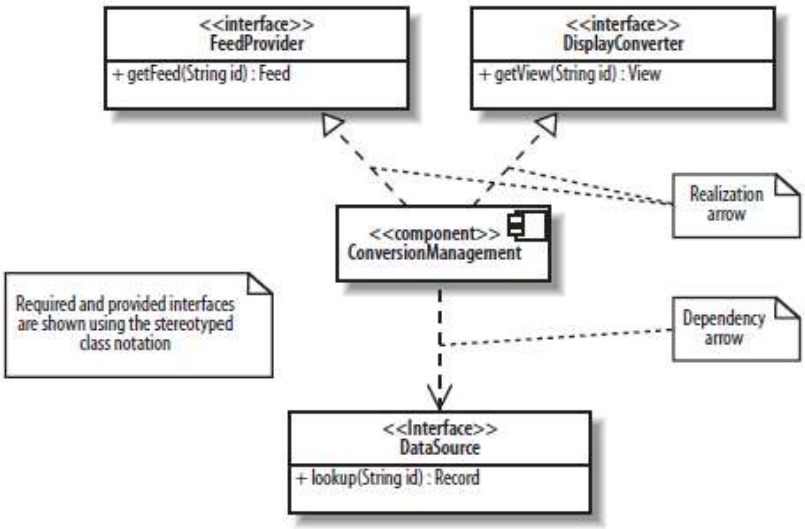
NOTASI BALL DAN SOCKET UNTUK INTERFACE

Interface yang diperlukan ditampilkan menggunakan setengah lingkaran bola - simbol socket seperti pada gambar dibawah, misal component *ConversionManagement* menyediakan interface *FeedProvider* dan *DisplayConverter* dan membutuhkan interface *DataSource*. Notasi socket and ball adalah cara umum untuk menunjukkan interface component:



NOTASI STEREOTYPE UNTUK INTERFACE

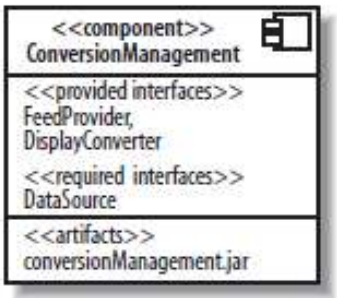
Kita juga dapat menunjukkan interface yang diperlukan dan disediakan component dengan menggambar interface dengan notasi stereotype kelas. Jika component realize(terdapat) interface, gambarkan panah realisasi dari component ke interface. Jika component membutuhkan interface, gambarkan panah ketergantungan dari component ke interface, seperti yang ditunjukkan pada Gambar 5.2.



Notasi ini digunakan jika ingin menunjukkan interface operation.

DAFTAR INTERFACE SEBUAH COMPONENT

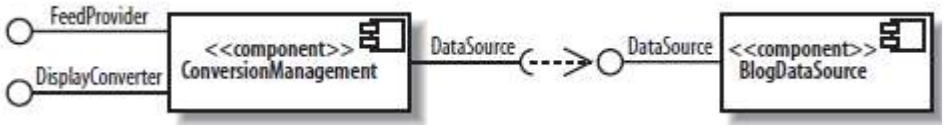
Cara paling ringkas untuk menunjukkan interface yang diperlukan dan disediakan adalah dengan menuliskannya di dalam component. Interface yang disediakan dan diperlukan tercantum secara terpisah, seperti yang ditunjukkan pada Gambar 12-6.



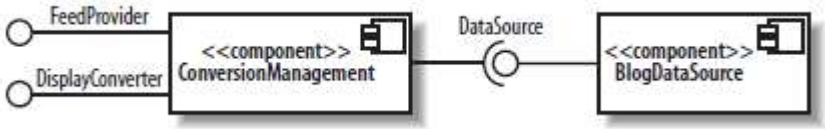
Notasi ini juga memiliki bagian <<artifacts>> yang mencantumkan artifact, atau file fisik, yang menggambarkan component ini.

1.42. MENGGAMBARKAN KERJASAMA COMPONENT

Jika component memiliki kebutuhan interface, maka memerlukan kelas atau component lain dalam sistem untuk menyediakannya. Untuk menunjukkan bahwa sebuah component bergantung pada component lain yang menyediakannya, digambarkan sbb :

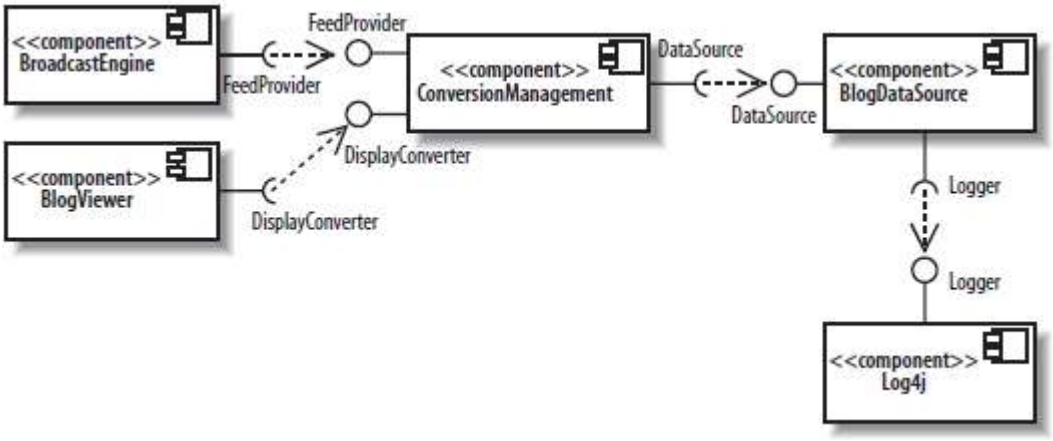


atau

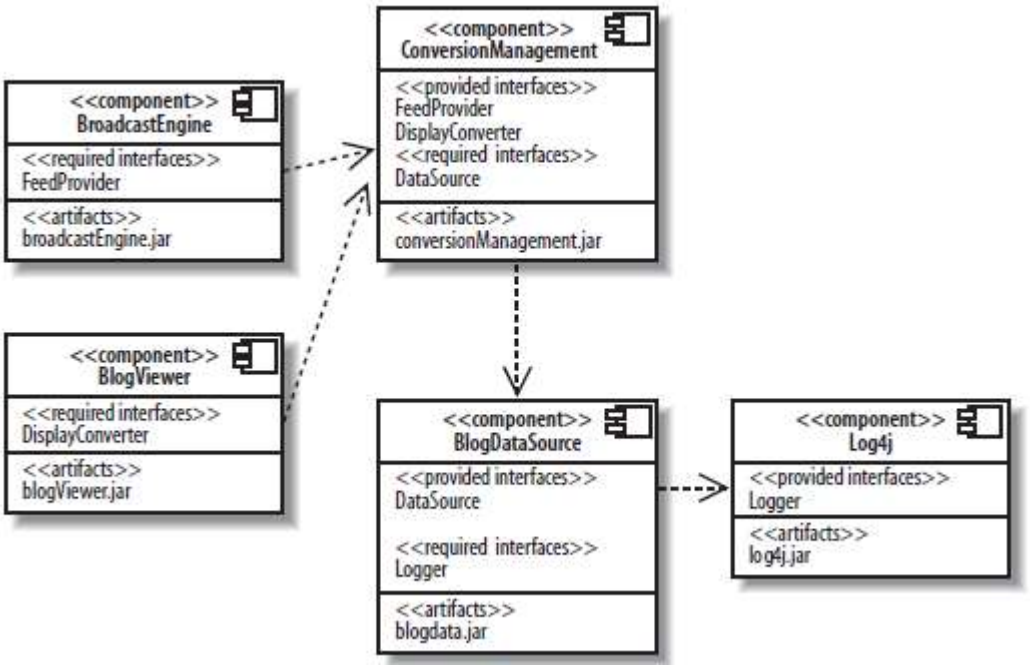


atau

Menampilkan component-component utama dalam sistem dan interkoneksi mereka melalui interface adalah cara yang bagus untuk menggambarkan arsitektur sistem dan seperti pada gambar dibawah ini :



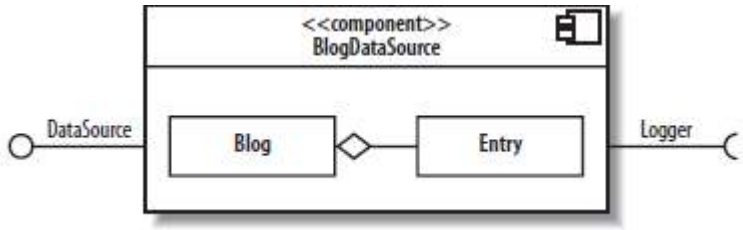
Notasi berikut adalah menunjukkan view tingkat tinggi dari dependensi component yang disederhanakan, berguna untuk memahami manajemen konfigurasi sistem atau masalah deployment karena menekankan ketergantungan component dan membuat daftar artifact sehingga terlihat jelas component dan file terkait yang diperlukan selama deployment, seperti gambar dibawah ini :



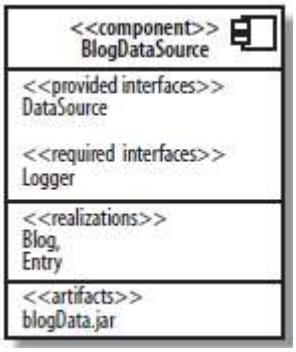
1.43. KELAS YANG MEMBANGUN (REALIZE) COMPONENT

Component sering berisi dan menggunakan kelas-kelas lain untuk mengimplementasikan fungsinya. Kelas-kelas semacam itu dikatakan merealisasikan component — mereka membantu component melakukan tugasnya.

Realisasi class dan hubungannya dapat digambarkan didalam component. Gambar dibawah menunjukkan component **BlogDataSource** berisi kelas **Blog** dan **Entri**, juga menunjukkan hubungan agregasi antara dua kelas.

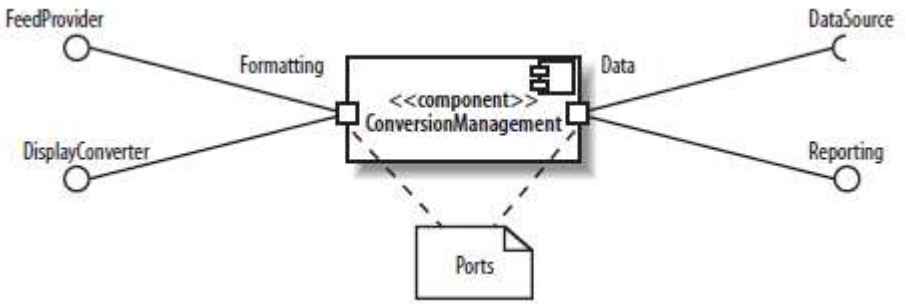


Cara lain untuk menunjukkan realisasi kelas adalah dengan membuat daftar di << realization>> di dalam component, seperti yang ditunjukkan pada gambar dibawah.

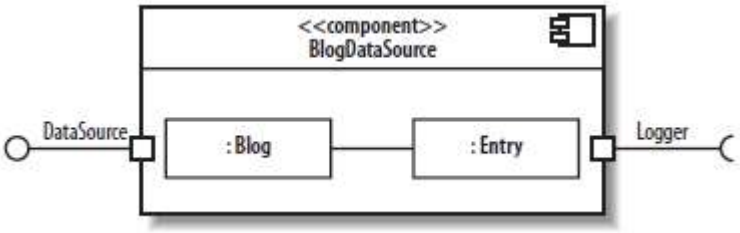


1.44. PORT DAN STRUKTUR INTERNAL

Component juga dapat memiliki port dan struktur internal. Kita dapat menggunakan port untuk memodelkan cara yang berbeda sehingga component dapat digunakan dengan interface terkait yang terpasang ke port. Pada gambar dibawah, component **ConversionManagement** memiliki port **Formatting** dan **Data**, masing-masing dengan interface terkait.



Kita dapat menunjukkan struktur internal component untuk memodelkan bagian, properti, dan konektornya. Gambar dibawah menunjukkan struktur internal component **BlogDataSource**.

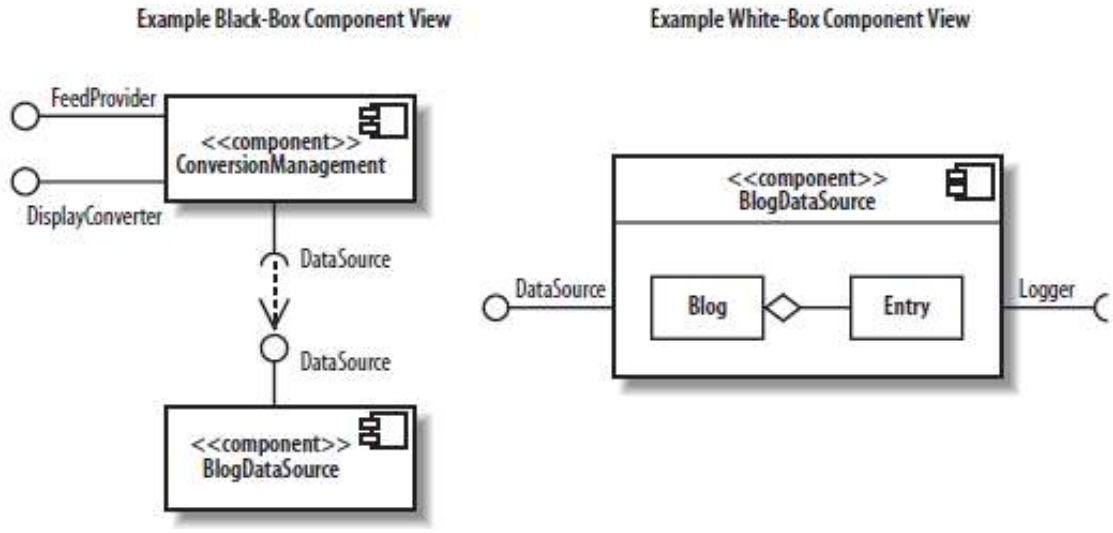


1.45. BLACKBOX DAN WHITE-BOX COMPONENT VIEW

Ada dua view dalam component UML: tampilan blackbox dan tampilan white-box.

Tampilan **blackbox** menunjukkan bagaimana component terlihat dari luar, termasuk interface yang diperlukan, interface yang disediakan, dan bagaimana hubungannya dengan component lain. Tampilan **blackbox** tidak menentukan apa pun tentang implementasi internal suatu component.

Tampilan white-box, menunjukkan kelas, interface, dan component lainnya yang membantu sebuah component untuk mencapai fungsinya.
Tampilan white-box memperlihatkan bagian di dalam component, sedangkan tampilan blackbox tidak, seperti yang ditunjukkan pada gambar dibawah:



Saat memodelkan sistem, sebaiknya gunakan tampilan blackbox untuk fokus pada masalah arsitektur berskala besar. Tampilan blackbox baik untuk menunjukkan component-component utama dalam sistem kita dan bagaimana mereka terhubung. Tampilan white-box, di sisi lain, berguna untuk menunjukkan bagaimana component mencapai fungsinya melalui kelas yang digunakannya.

Tampilan blackbox biasanya berisi lebih dari satu component, sedangkan dalam white-box lebih fokus pada isi dari component.

DEPLOYMENT DIAGRAM

Jika kita telah menerapkan teknik UML yang ditunjukkan pada bab-bab sebelumnya, maka kita telah melihat semua kecuali tampilan fisik sistem (*physical view*). Tampilan fisik berkaitan dengan eleomen fisik sistem, seperti file executable dan perangkat keras yang digunakannya.

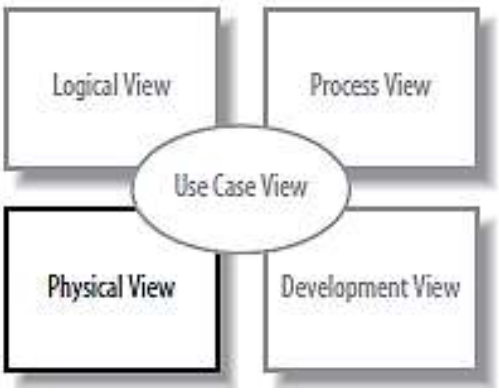
Dalam pengembangan sistem TI, **deployment** adalah kegiatan mengembangkan perangkat lunak (misal aplikasi/website) yang tidak bisa lepas dari kegiatan para developer(pengembang). Terdapat beberapa proses yang harus dikerjakan hingga akhirnya bisa membangun perangkat lunak. Contoh **kegiatan deployment** adalah menyusun kode (coding) untuk dimasukkan ke server dan setting server tersebut agar dapat menghasilkan suatu aplikasi.

Deployment adalah kegiatan yang bertujuan untuk **menyebarkan (*deploy*) aplikasi** yang telah dikerjakan oleh para orang-orang yang ahli di bidang developer.

Cara penyebarannya (deploy) sangat beragam, tergantung dari jenis aplikasinya. Misal bila kita pilih aplikasi Web, maka kita akan hosting pada server, jika aplikasi mobile, akan terdapat dua deployment: deployment untuk aplikasi ke Playstore atau Appstore, kedua adalah deployment API (*backend*) ke server.

Untuk melakukan deployment kita harus sabar karena akan banyak sesuatu yang tidak diinginkan bisa terjadi. Contoh kendala yang sering dialami adalah sistem yang tiba-tiba down, karena itulah butuh waktu tidak sebentar untuk men-deploy suatu perangkat lunak.

Deployment Diagram UML menunjukkan tampilan fisik sistem, memperlihatkan perangkat lunak ke dunia nyata dengan menunjukkan bagaimana perangkat lunak ditugaskan pada perangkat keras dan bagaimana bagian-bagian tersebut berkomunikasi (lihat Gambar 10.1).



Gambar 137. Physical View

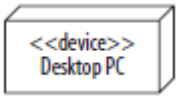
Ingat: Kata "sistem" dapat memiliki arti berbeda bagi orang yang berbeda; dalam konteks deployment diagram, sistem berarti perangkat lunak yang kita bangun dan perangkat keras dan perangkat lunak apa yang dibutuhkan oleh perangkat lunak yang kita bangun, agar dapat berjalan.

1.46. MEN-DEPLOY SISTEM SEDERHANA

Berikut adalah sistem Deployment Diagram yang sangat sederhana. Dalam kasus sederhana ini, perangkat lunak kita akan dikirimkan sebagai satu file yang dapat dieksekusi berada di satu komputer.

Untuk menunjukkan perangkat keras komputer, gunakan node, seperti yang ditunjukkan pada Gambar dibawah.

Gunakan node untuk mewakili perangkat keras system. Sistem berisi sebuah hardware-misal Desktop PC, diberi label **<<device>>.** Gambar berikut menunjukkan node untuk merepresentasikan hardware di sistem TI.



Kembali lagi seberapa detil dan penting anda ingin memberikan gambaran tentang **node** tersebut, misal lebih didetilkan dengan kebutuhan tertentu "Desktop PC-64 bit Intel Processor" atau cukup dengan tulisan PC saja.

Sekarang, misal kita ingin memodelkan sofware yang akan berjalan di hardware dengan nama *3dpacman.jar* yang berisi aplikasi 3D-pacman, digambarkan dibawah ini:

Kemudian, kita ingin meletakkan dua bagian ini kedalam deployment diagram pada sebuah sistem dengan cara gambarkan artifact kedalam node untuk menunjukkan software artifact di letakkan/deploy kedalam node hardware. Contoh 3dpacman.jar berjalan di hardware Desktop PC digambarkan sbb:

Apakah diagram tsb sudah lengkap? Apakah kita tidak memodelkan Java Virtual Machine (JVM) misalnya, karena tanpa JVM code program kita tidak dapat berjalan. Bagaimana dengan sistem operasi ? Apakah itu tidak perlu digambarkan ?.

Ingat : Deployment Diagram menggambarkan **hal penting** tergantung siapa yang membutuhkan. Bila memang diperlukan, silahkan cantumkan sistem operasi, firmware, runtime environment atau bahkan driver device anda.

1.47. DEPLOYED SOFTWARE: ARTIFACT

Diatas telah dicontohkan sedikit tentang notasi yang bisa digunakan untuk menampilkan sofware dan hardware pada sistem yang akan di deploy. Sofware *3dpacman.jar* diletakkan/deploy pada sebuah hardware node. File JAR tersebut disebut : artifact.

Artifact adalah file secara fisik yang akan dieksekusi atau dibutuhkan oleh sofware yang kita bangun. Artifact yang biasa digunakan adalah :

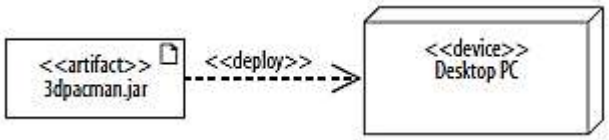
- File Executable:file .exe atau .jar
- File Library : file .dlls atau .dll
- Source file : file .java atau .cpp
- File Configuration (dibutuhkan untuk menjalankan software kita saat runtime, biasanya berformat .xml, .properties atau .txt

Artifact digambarkan dengan kotak bertuliskan <<artifact>>, atau ikon dokumen di pojok kanan atas, contoh artifact 3dpacman.jar dapat digambarkan pada salah satu gambar seperti dibawah ini :

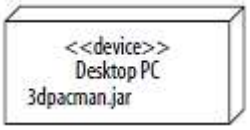
DEPLOY SEBUAH ARTIFACT KE NODE

Artifact akan diletakkan ke node dalam arti di install ke node. Contoh dibawah ini menunjukkan sebuah artifact 3dpacman.jar dari contoh sebelumnya di deploy ke Desktop PC hardware node dengan menuliskan simbol artifact didalam node.

Atau anda juga bisa memodelkan dengan cara berbeda yaitu menunjukkan dependencies(ketergantungan) menggunakan stereotype <<deploy>> spt dibawah ini :

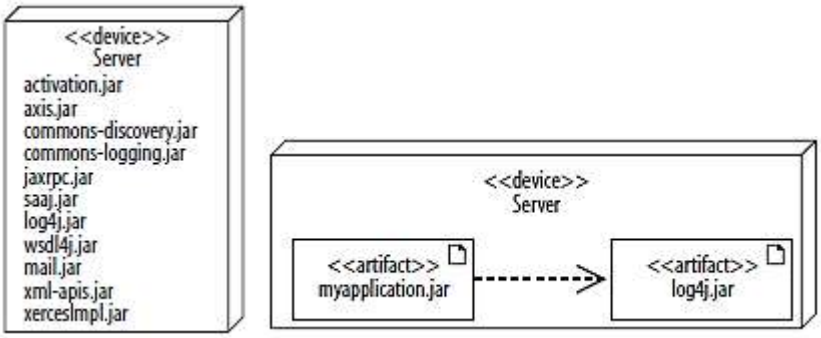


Atau dengan menggambarkan sederhana spt dibawah ini :



Berikut adalah petunjuk dalam memilih model diagram yang akan kita gunakan :

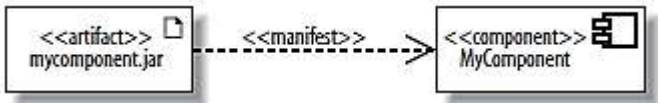
- List semua artifact (tanpa simbol artifact), bila anda menggunakan banyak sekali artifact. Ingat : bila hanya list artifact, kita tidak dapat menunjukkan dependencies antar artifact.
- Bila terdapat ketergantungan, harus digambarkan seperti dibawah :



MENGIKAT SOFTWARE KE ARTIFACT

Saat merancang perangkat lunak, kita memecahnya menjadi kelompok-kelompok fungsionalitas yang kohesif, seperti component atau package, yang akhirnya dikompilasi menjadi satu atau beberapa file — atau artifact. Dalam UML, jika artifact adalah aktualisasi fisik dari suatu komponen, maka artifak memanifestasikan komponen itu. Artifack dapat memanifestasikan tidak hanya komponen tetapi elemen yang dapat dipaketkan, seperti package dan kelas.

Hubungan manifestasi ditunjukkan dengan panah ketergantungan dari artefak ke komponen dengan stereotip <<manifest>>, seperti yang ditunjukkan pada Gambar dibawah ini :



1.48. APA ITU NODE?

Diatas sudah dicontohkan bagaimana sebuah node digunakan untuk menunjukkan hardware, tetapi node tidak harus hardware. Bisa juga berupa software-software tertentu yang menyediakan lingkungan dengan komponen software lain yang dapat dieksekusi dapat juga digambarkan sebagai node.

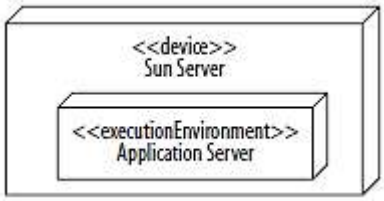
Node adalah sumber daya hardware atau software yang dapat menjadi host perangkat lunak atau file terkait. Kita dapat menganggap node software dalam konteks aplikasi; umumnya bukan bagian dari software yang kita kembangkan, tetapi lingkungan pihak ketiga yang menyediakan layanan untuk software kita.
Berikut adalah contoh hardware nodes yang bisa digambarkan : Server, Deksktop PC, Printer, Scanner dll. Conto Sofware nodes:Operating System, J2EE container, Web Server, Application Server dll.

PERANGKAT KERAS DAN NODE LINGKUNGAN EKSEKUSI (EXECUTION ENVIRONTMENT)

Node digambarkan dengan kubus dengan tipe di dalamnya, seperti yang ditunjukkan pada gambar dibawah ini. Stereotype <<device>> menunjukkan node hardware.

Gambar berikut menunjukkan node Application Server. Pada pengembangan perangkat lunak perusahaan akan mengenali ini sebagai jenis lingkungan eksekusi, karena ini adalah lingkungan perangkat lunak yang menyediakan layanan untuk aplikasi kita. Stereotype <executionEnvironment>> menunjukkan bahwa node ini adalah lingkungan eksekusi.

Lingkungan eksekusi tidak berdiri sendiri — mereka berjalan pada perangkat keras. Misalnya, sistem operasi membutuhkan perangkat keras komputer untuk dapat berjalan. Misal kita ingin menunjukkan bahwa lingkungan eksekusi berada pada perangkat tertentu dengan menempatkan node di dalam, nested seperti yang ditunjukkan pada gambar dibawah:



Dalam UML, tidak harus menggambarkan lingkungan environment, tetapi akan menjadi kebiasaan yang baik bila digambarkan dengan lengkap agar model menjadi jelas.

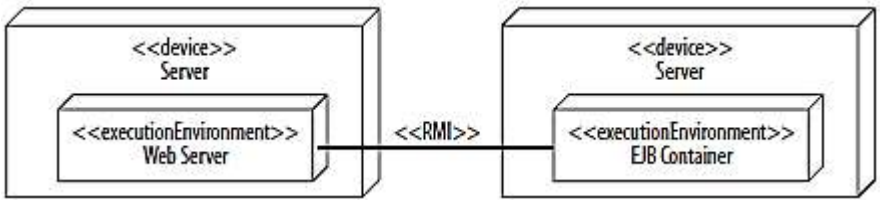
KOMUNIKASI ANTAR NODE

Untuk menyelesaikan tugas, sebuah node mungkin perlu berkomunikasi dengan node lain. Misalnya, aplikasi klien yang berjalan pada Desktop PC dapat mengambil data dari server menggunakan TCP / IP.

Jalur komunikasi digunakan untuk menunjukkan bahwa node berkomunikasi satu sama lain saat runtime. Jalur komunikasi digambarkan sebagai garis yang menghubungkan dua node. Jenis komunikasi ditunjukkan dengan menambahkan stereotype ke jalur. Gambar dibawah menunjukkan dua node — Desktop PC dan server — yang berkomunikasi menggunakan TCP / IP.

Kita juga dapat menunjukkan jalur komunikasi antara node pada lingkungan eksekusi. Misalnya, Anda bisa membuat model web server yang berkomunikasi dengan EJB container melalui RMI, seperti yang ditunjukkan pada Gambar dibawah. Ini lebih tepat daripada menunjukkan jalur komunikasi RMI di tingkat device node. Namun, beberapa pemodel menggambar jalur komunikasi pada level node diluar karena agar diagram menjadi lebih rapi.

Menetapkan stereotip ke jalur komunikasi kadang-kadang bisa rumit. RMI berlapis menggunakan lapisan transport TCP / IP. Jadi, haruskah Anda menetapkan stereotype <<RMI>> atau << TCP / IP >>? Sebagai patokan, stereotip komunikasi harus setinggi mungkin karena lebih banyak berkomunikasi tentang sistem Anda. Pada kasus ini, <<RMI>> adalah pilihan yang tepat; ini tingkat yang lebih tinggi, dan memberi tahu audiens bahwa implementasi menggunakan Java. Namun, seperti pada kebanyakan pemodelan UML,kita harus menyesuaikan diagram dengan audiens.

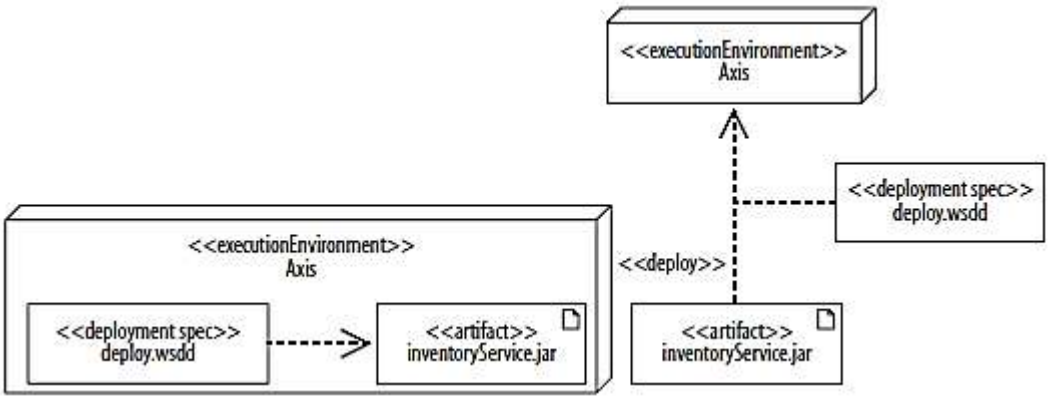


1.49. SPESIFIKASI DEPLOYMENT

Menginstal sofware sangat mudah seperti menjatuhkan file pada sebuah mesin; sering kita harus menentukan parameter konfigurasi sebelum software kita dapat berjalan. Spesifikasi deployment adalah artifact khusus yang menentukan bagaimana artifact lain digunakan untuk sebuah node. Ini memberikan informasi yang memungkinkan artefak lain berjalan dengan sukses di dalam lingkungannya.

Spesifikasi deployment digambarkan dengan persegi panjang dengan stereotype <<deployment spec>>. Ada dua cara untuk mengikat spesifikasi penempatan dengan deployment yaitu :

- Gambar panah ketergantungan dari spesifikasi penempatan ke artifact, yang keduanya nested di node target.
- Lampirkan deployment spec ke panah deployment, seperti yang ditunjukkan pada gambar dibawah ini :



File *deploy.wsdd*, ditunjukkan pada gambar diatas, adalah *file standard deployment descriptor* yang menentukan bagaimana web service digunakan untuk Axis web service engine. File ini menyatakan kelas mana yang mengeksekusi layanan web dan metode apa pada kelas yang dapat dipanggil. Kita bisa mencantumkan properti ini dalam spesifikasi penerapan menggunakan nama: notasi tipe.

Gambar dibawah menunjukkan spesifikasi penyebaran *deploy.wsdd* dengan properti *className* dan *allowedMethods*.

Simbol di sebelah kanan menunjukkan turunan dari deployment spec yang diisi dengan sebuah nilai. Gunakan notasi ini jika Anda ingin menunjukkan nilai properti.

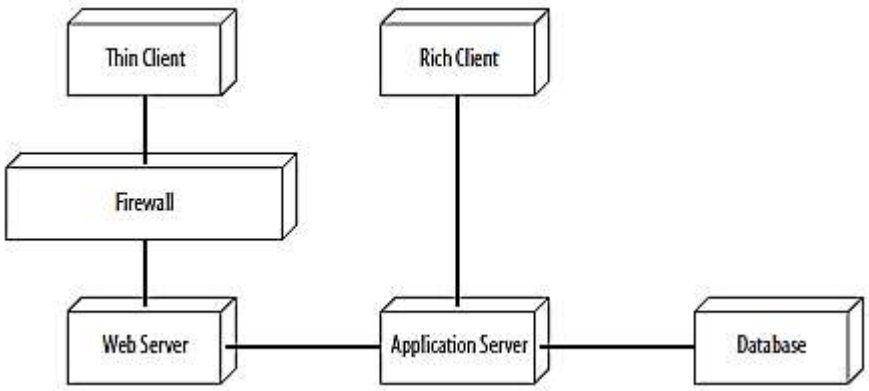
1.50. KAPAN MENGGUNAKAN DEPLOYMENT DIAGRAM

Deployment diagram berguna di semua tahap proses desain. Ketika kita mulai merancang suatu sistem, kita mungkin hanya tahu informasi dasar tentang tata letak fisik. Misalnya, jika kita membuat aplikasi web, kita mungkin tidak memutuskan perangkat keras mana yang akan digunakan dan mungkin tidak tahu apa nama perangkat lunak nya. Tetapi Anda ingin mengkomunikasikan karakteristik penting dari sistem Anda, seperti berikut ini:

- Arsitektur Anda mencakup server web, server aplikasi, dan basis data.
- Klien dapat mengakses aplikasi Anda melalui browser atau melalui antarmuka GUI.
- Server web dilindungi dengan firewall.

Bahkan pada tahap awal ini Anda dapat menggunakan deployment diagram untuk memodelkan karakteristik ini.

Gambar 138 menunjukkan sketsa kasar sistem kita. Nama node tidak harus tepat, dan kita tidak harus menentukan protokol komunikasi.



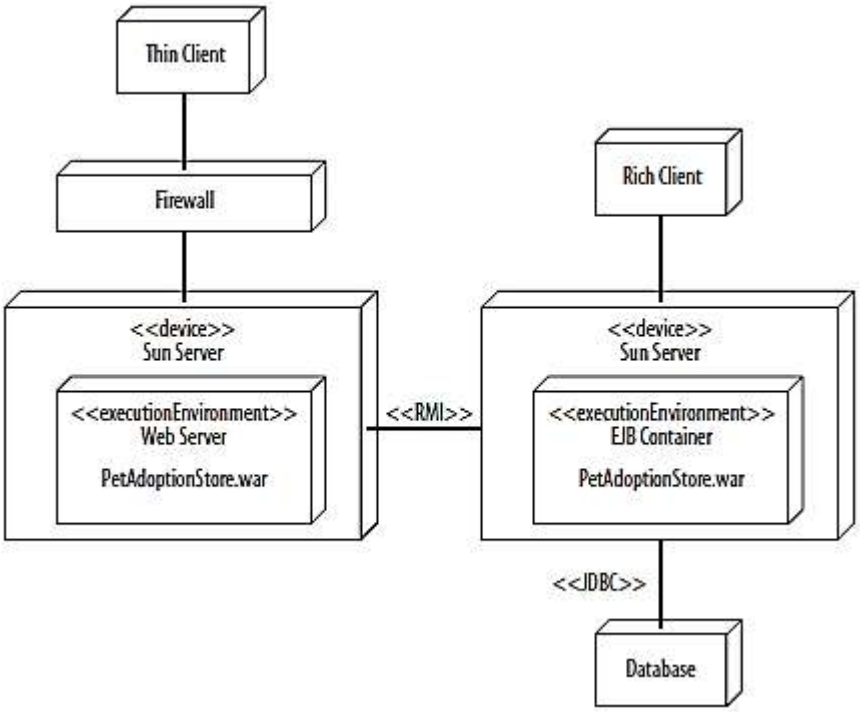
Gambar 138. Sketsa Kasar Deployment Diagram Aplikasi Web

Deployment Diagram juga berguna dalam tahap akhir pengembangan perangkat lunak.

Gambar 138 adalah contoh deployment diagram perinci yang menetapkan implementasi sistem J2EE.

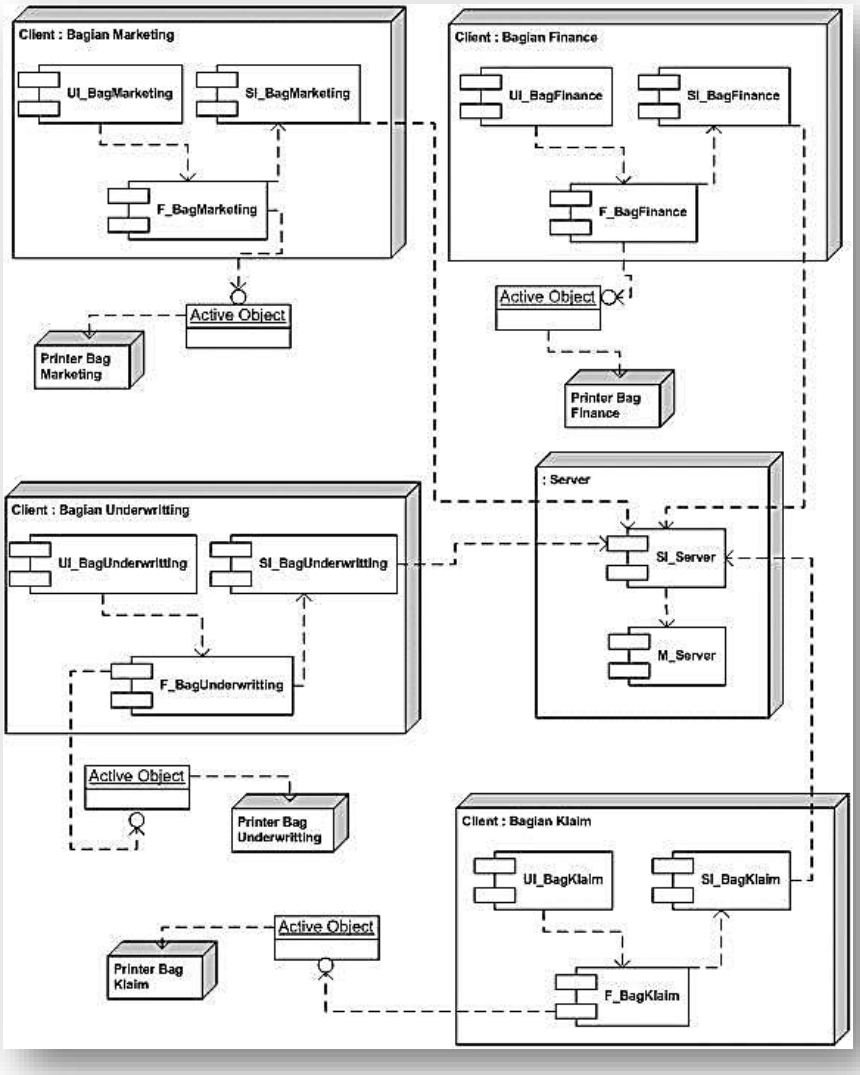
Gambar 139 lebih spesifik tentang jenis perangkat keras, protokol komunikasi, dan alokasi artifact software ke node. Deployment diagram Gambar 139, dapat digunakan digunakan sebagai blue print cara menginstal sistem kita.

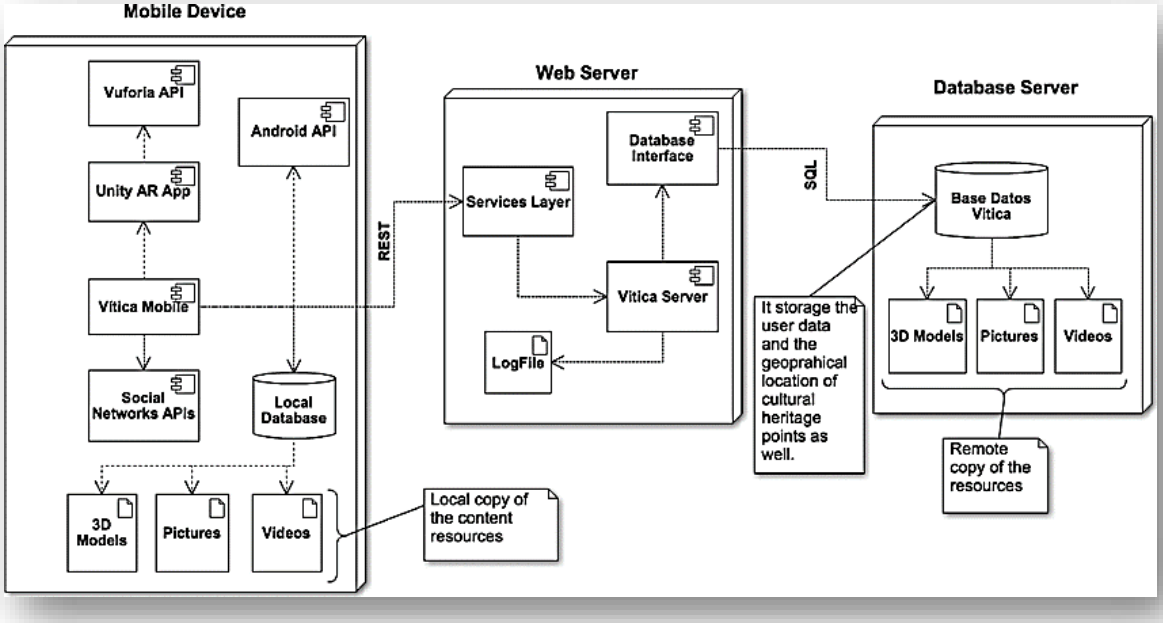
Kita dapat melihat kembali deployment diagram di seluruh desain sistem kita, untuk memperbaiki sketsa awal yang masih kasar, tambahkan detail saat kita memutuskan menggunakan teknologi, protokol komunikasi, dan artifact software yang akan digunakan. Deployment diagram yang disempurnakan ini memungkinkan kita untuk memperlihatkan tampilan tata letak sistem fisik saat ini dengan para stakeholders/ orang-orang yang berkepentingan terhadap sistem.



Gambar 139. Deployment Diagram Yang Sudah Dilengkapi (Rinci)

CONTOH DEPLOYMENT DIAGRAM


Client : Bagian Marketing
UI_BagMarketing
SI_BagMarketing
F_BagMarketing
Active Object
Printer Bag Marketing
Client : Bagian Finance
UI_BagFinance
SI_BagFinance
F_BagFinance
Active Object
Printer Bag Finance
: Server
SI_Server
M_Server
Client : Bagian Underwritting
UI_BagUnderwritting
SI_BagUnderwritting
F_BagUnderwritting
Active Object
Printer Bag Underwritting
Client : Bagian Klaim
UI_BagKlaim
SI_BagKlaim
F_BagKlaim
Active Object
Printer Bag Klaim



KESIMPULAN

SOAL LATIHAN

UNIVERSITAS BUDI LUHUR
FAKULTAS TEKNOLOGI INFORMASI

MODUL PERKULIAHAN #14
INTERACTION FLOW MODELING LANGUAGE (IFML)

Capaian Pembelajaran	:	Mahasiswa memahami dan menggunakan interaction flow modeling language
Sub Pokok Bahasan	:	1.69. Pengantar IFML 1.70. Sub-Element Dan View Component Part 1.71. Gambaran Umum Konsep Utama IFML 1.72. Contoh IFML
Daftar Pustaka	:	Brambilla, M. and Fraternali, P., 2014. Interaction flow modeling language: Model-driven UI engineering of web and mobile apps with IFML. Morgan Kaufmann.

PENGANTAR IFML

Interaction flow modelling language (IFML) merupakan bahasa pemodelan yang mendeskripsikan sebuah aplikasi berdasarkan **konten**, **interaksi** (user interaction), serta **perilaku** dan **kontrol** yang bekerja pada front-end dari aplikasi tersebut. Bahasa pemodelan ini kemudian ditetapkan menjadi sebuah standar pada tahun 2013.

Untuk contoh konsep-konsep utama yang terdapat pada IFML dapat dilihat pada Tabel 7.1

Table 7.1 - Essential IFML Concepts

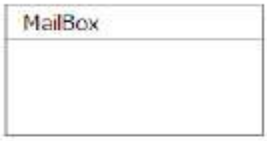
Concept	Meaning	IFML Notation	Example at Implementation level
View Container	An element of the interface that comprises elements displaying content and supporting interaction and/or other ViewContainers.		Web page Window Pane
XOR View Container	A ViewContainer comprising child ViewContainers that are displayed alternatively.		Tabbed panes in Java Frames in HTML
Landmark View Container	A ViewContainer that is reachable from any other element of the user interface without having explicit incoming InteractionFlows.		A logout link in HTML sites that is visible in every page.
Default View Container	A ViewContainer that will be presented by default to the user, when its enclosing container is accessed.		A welcome page

Table 7.1 - Essential IFML Concepts

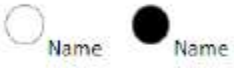
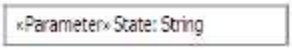
Concept	Meaning	IFML Notation	Example at Implementation level
View Component	An element of the interface that displays content or accepts input.		An HTML list. A JavaScript image gallery. An input form.
Event	An occurrence that affects the state of the application.		
Action	A piece of business logic triggered by an event; it can be server side (the default) or client-side, denoted as [Client].		A database update. The sending of an email. The spell checking of a text.
Navigation Flow	Update of the interface elements in view or triggering of an action caused by the occurrence of an event. Data may be associated with the flow through parameter bindings.		Sending and receiving of parameters in the HTTP request.
Data Flow	Data passing between ViewComponents or Action as consequence of a previous user interaction.		
Parameter	A typed and named value	Optionally shown. If necessary can be denoted as follows: 	HTTP query string parameters HTTP post parameters JavaScript variables and function parameters
Parameter Binding	Specification that an input parameter of a source is associated with an output parameter of a target.		

Table 7.1 - Essential IFML Concepts

Concept	Meaning	IFML Notation	Example at Implementation level
Parameter Binding Group	Set of ParameterBindings associated to an InteractionFlow (being it navigation or data flow).	«ParameterBindingGroup» Title → AlbumTitle Year → AlbumYear	
Activation Expression	Boolean expression associated with a ViewElement, ViewComponentPart or Event: if true the element is enabled	«ActivationExpression» not MessageSet.isEmpty()	
Interaction Flow Expression	Determine which of the InteractionFlows are going to be followed as consequence of the occurrence of an Event.	«InteractionFlowExpression» if AlbumDetails selected then AlbumDetails else Album details	Event triggered after selecting a given value in a ComboBox.

Beberapa konsep dalam IFML dapat diperluas layaknya UML Perluasan konsep pada bahasa pemodelan dilakukan agar dapat memodelkan aplikasi atau sistem dalam platform tertentu secara detail. Konsep-konsep IFML yang dapat diperluas ini yaitu :

- ViewContainer
- ViewComponent
- ViewComponentPart
- Event

SUB-ELEMENT DAN VIEW COMPONENT PART

Hirarki yang mengatur hubungan antar elemen dan sub-elemen.

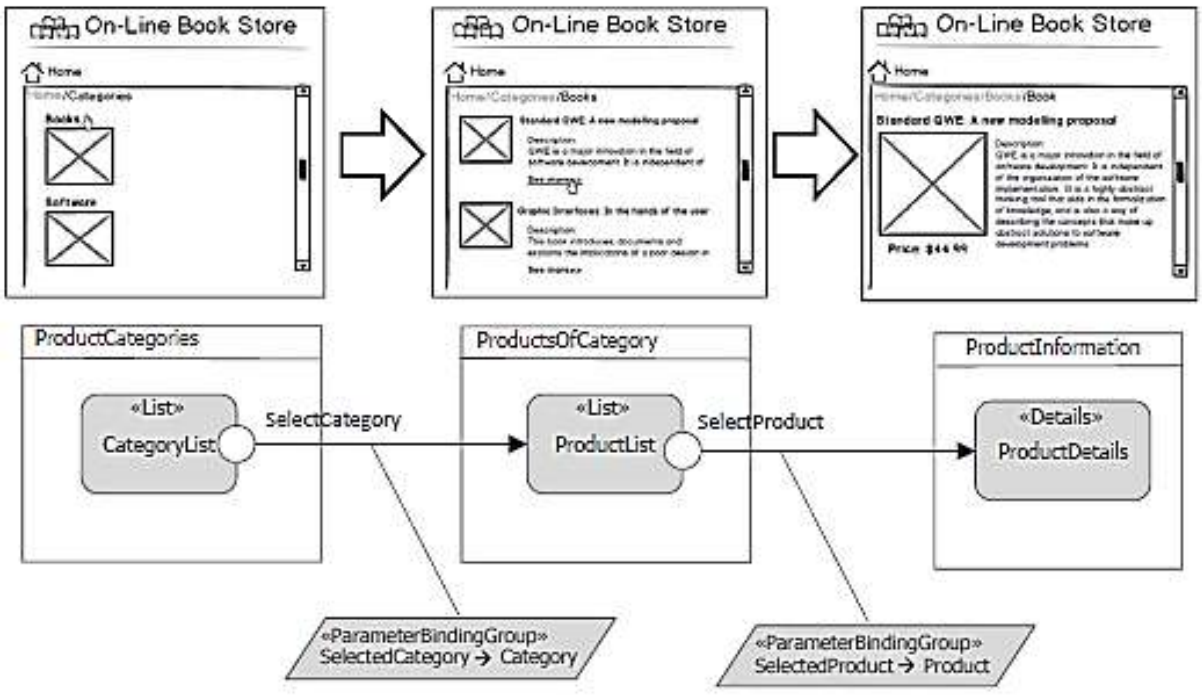
Hirarki	Nama konsep/elemen
I	ViewContainer
II	ViewComponent
III	Flow
IV	Property

Tiap elemen/konsep pada IFML dapat mempunyai *sub-element*.

Sebuah elemen dapat mempunyai sub-elemen dengan hirarki yang sama atau dibawahnya. Contohnya adalah sebuah Area dapat mempunyai sub-Area atau Page yang berada di dalam Area tersebut. Khusus untuk elemen yang diperluas dari konsep ViewComponent serta beberapa elemen dari ViewContainer tidak dapat mempunyai sub-element berupa ViewComponent.

Gambar 140 menunjukkan contoh kasus model IFML yang mendeskripsikan sebuah web. Terdapat tiga view container pada gambar tersebut yang mewakili sebuah halaman dalam aplikasi web yaitu ProductCategories, ProductList, dan Product. Selain itu juga terdapat interface berupa view component yang menampilkan konten serta memungkinkan user untuk melakukan interaksi tertentu.

Dalam kasus ini, view component yang terdapat pada masing-masing halaman tersebut merupakan List yang menampilkan kategori (CategoryList), produk (ProductList), serta detail dari sebuah produk (Product Details).



Gambar 140. Contoh Interface dan Spesifikasi IFML

Dapat dilihat pada gambar tersebut bahwa panah navigation flow menghubungkan antar view component dengan sebuah parameter binding. Parameter binding berarti output hasil interaksi yang dilakukan oleh user pada view component asal menjadi

parameter input yang akan mempengaruhi konten yang akan ditampilkan ViewComponent tujuan.

Contoh kasus diatas adalah ketika user memilih kategori "Books" pada ViewComponent CategoryList (lihat Gambar 140), maka kategori tersebut akan dijadikan parameter untuk menampilkan produk-produk yang ada pada view component ProductList (lihat Gambar 140), yaitu produk dengan kategori "Books".

GAMBARAN UMUM KONSEP UTAMA IFML

Diagram IFML terdiri dari satu atau lebih *ViewContainer* (mis., Elemen interface yang terdiri dari komponen untuk menampilkan konten dan interaksi pendukung).

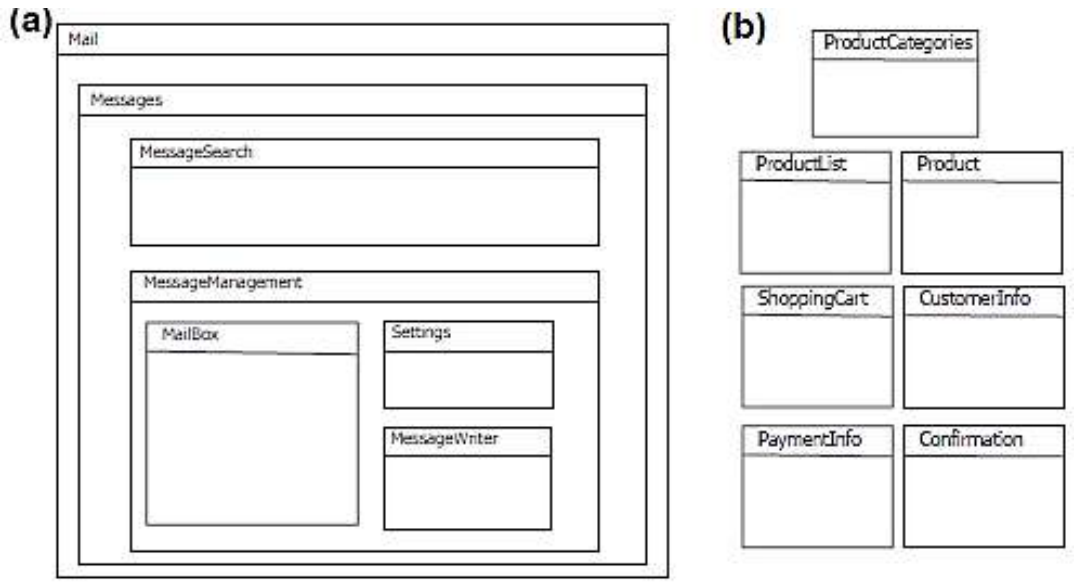
Gambar 141 disajikan dua bentuk pengaturan GUI:

(a) aplikasi e-mail yang terdiri dari container level atas dengan sub-container didalamnya dengan level berbeda,

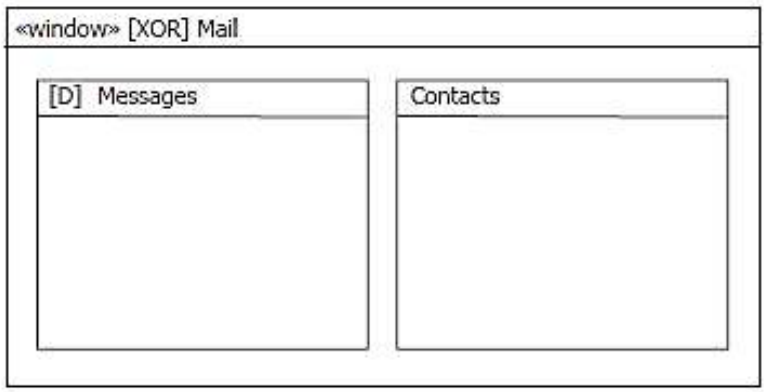
(b) situs web e-commerce yang mengatur interface user ke berbagai container display independen yang sesuai dengan template halaman.

Setiap container display dapat disusun dalam **hierarki subkontainer**. Misalnya, di desktop atau aplikasi yang ada, window utama berisi beberapa tab frame, yang berisi beberapa panel. Container anak dalam display container induk dapat ditampilkan secara bersamaan (mis., Panel objek dan panel properti) atau dalam pengecualian bersama (mis., Dua tab alternatif). Dalam hal kontainer yang saling eksklusif/mutual exclusion (XOR), satu container default, yang ditampilkan secara default ketika container induk diakses. Agar lebih jelas, arti sebuah container dapat dispesifikasikan dengan menambahkan stereotype sesuai dengan tujuan.

Contoh Gambar 142, ViewContainer ditandai sebagai **«window»**, seperti dalam kasus ViewContainer **"Mail"** pada untuk memberi petunjuk implementasi yang diharapkan.



Gambar 141. Contoh Perbedaan Struktur Top-Level Interface



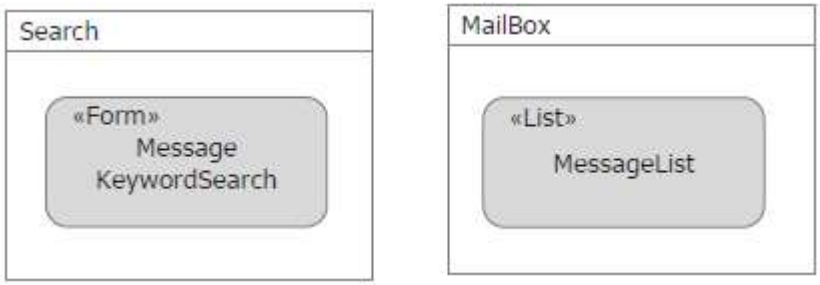
Gambar 142. Contoh Subcontainer *mutually exclusive*

Pada Gambar 142, container top-level **"Mail"** terdiri dari dua subkontainer, ditampilkan dengan alternatif: satu untuk **Message** dan satu untuk **Contacts**. Ketika container top-level diakses, interface menampilkan ViewContainer **"Message"** secara **default [D]**.

ViewContainer berisi ViewComponents, yang menunjukkan konten yang ditampilkan (mis., List objek) atau input data (mis., Form entri).

Gambar 143 menggambarkan notasi ViewComponents yang berada dalam ViewContainers. ViewContainer "Search" terdiri dari ViewComponent

"MessageKeywordSearch" yang mewakili form untuk pencarian; ViewContainer "MailBox" terdiri dari ViewComponen "MessageList" yang menunjukkan list objek. ViewComponent dapat memiliki parameter input dan output. Misalnya, komponen display yang memperlihatkan detail suatu objek memiliki parameter input yang sesuai dengan identifikasi objek yang akan ditampilkan. Form entri data sebagai parameter output nilai yang dikirimkan oleh user. dan list item yang dikirim sebagai parameter output item yang dipilih oleh user.



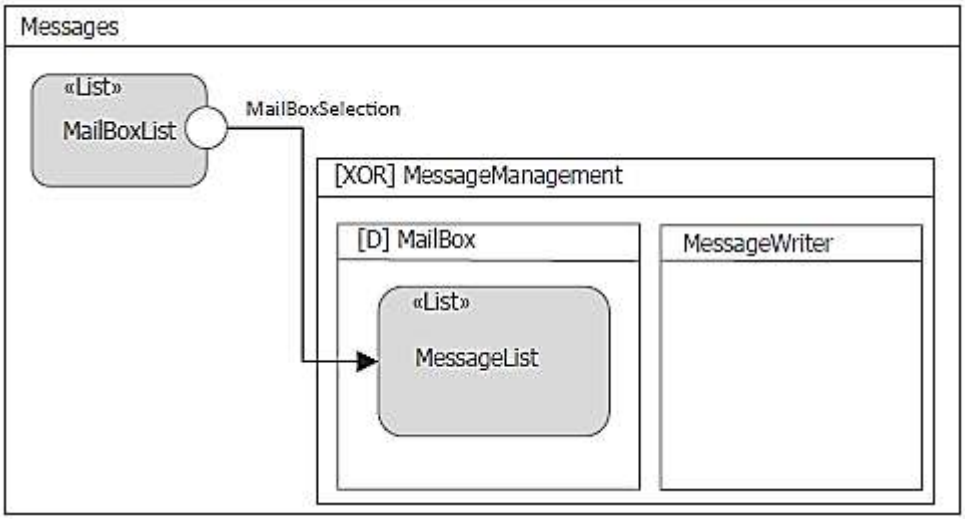
Gambar 143. Contoh ViewComponent Dalam View Container

ViewContainer dan ViewComponent dapat dikaitkan dengan event untuk menyatakan bahwa terjadi interaksi user. Misalnya, ViewComponent dapat mewakili list terkait dengan event untuk memilih satu atau lebih item, form yang terkait dengan event untuk pengiriman masukan, atau galeri gambar yang terkait dengan event untuk scrolling galeri. Event IFML dipetakan ke *interactor* dalam aplikasi yang diterapkan. Cara di mana interaksi tersebut diberikan tergantung pada platform spesifik yang digunakan aplikasi dan tidak ditangkap oleh IFML. Sebaliknya, itu didelegasikan ke aturan transformasi dari model platform-independent (PIM) ke model platform-spesifik (PSM). Misalnya, scrolling galeri foto dapat diimplementasikan sebagai tautan dalam aplikasi HTML dan sebagai penggeser isyarat gesek dalam aplikasi ponsel.

Efek dari suatu event diwakili oleh aliran interaksi, yang menghubungkan event ke ViewContainer atau ViewComponent yang dipengaruhi oleh event tersebut. Misalnya, dalam aplikasi web HTML event yang dihasilkan oleh pemilihan satu item dari list dapat menyebabkan display halaman baru dengan detail objek yang dipilih. Ini efek diwakili oleh aliran interaksi yang menghubungkan event yang terkait dengan komponen list di ViewContainer level atas (halaman web) dengan ViewComponent yang mewakili detail objek, yang diposisikan di ViewContainer yang berbeda (halaman web target).

Aliran interaksi menyatakan perubahan status interface user. Terjadinya event menyebabkan transisi dari sumber ke halaman web target.

Contoh, pada Gambar 143 ViewComponent "MailBoxList" menunjukkan list MessageList yang ada dan dikaitkan dengan event "MailBoxSelection", di mana user dapat membuka ViewContainer "MailBox" dan mengakses message-message dari mailBox List yang dipilih dalam ViewComponent Komponen "MessageList".



Gambar 144. Contoh Flow Antar ViewComponent

Suatu event juga dapat menyebabkan pemicu suatu action, yang dikerjakan sebelum memperbarui keadaan interface user. Efek dari suatu event yang memicu action diwakili oleh aliran interaksi yang menghubungkan event tersebut dengan simbol action (diwakili oleh segi enam). Misalnya, dalam aplikasi mail management, user memilih beberapa message dari list dan memilih untuk men delete. Event memicu action delete, setelah itu ViewContainer ditampilkan lagi dengan list yang diperbarui. Hasil eksekusi action diwakili oleh aliran interaksi yang menghubungkan action ke ViewContainer atau ViewComponent yang terpengaruh.

Pada Gambar 145, ViewContainer " Message Toolbar " berhubungan dengan event delete, archive, dan report message email. Event seperti ini dihubungkan oleh aliran ke simbol action (ikon heksagonal), yang mewakili operasi bisnis. Aliran keluar dari action menunjuk ke ViewContainer yang ditampilkan setelah action dijalankan; jika aliran keluar dari suatu action dihilangkan, ini berarti bahwa ViewContainer yang sama

dari mana action diaktifkan tetap terlihat (seperti yang diilustrasikan oleh action "Archive" dan "Report" pada Gambar 145).

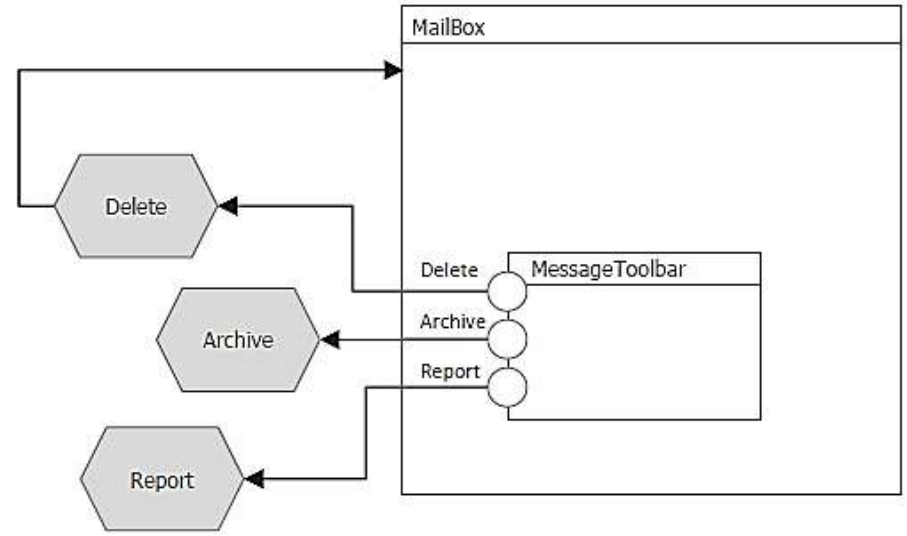
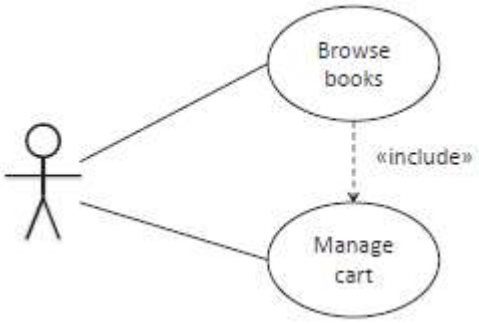


FIGURE 2.7
Example of events triggering business actions.

Gambar 145. Event Yang Memicu Business Action

CONTOH IFML

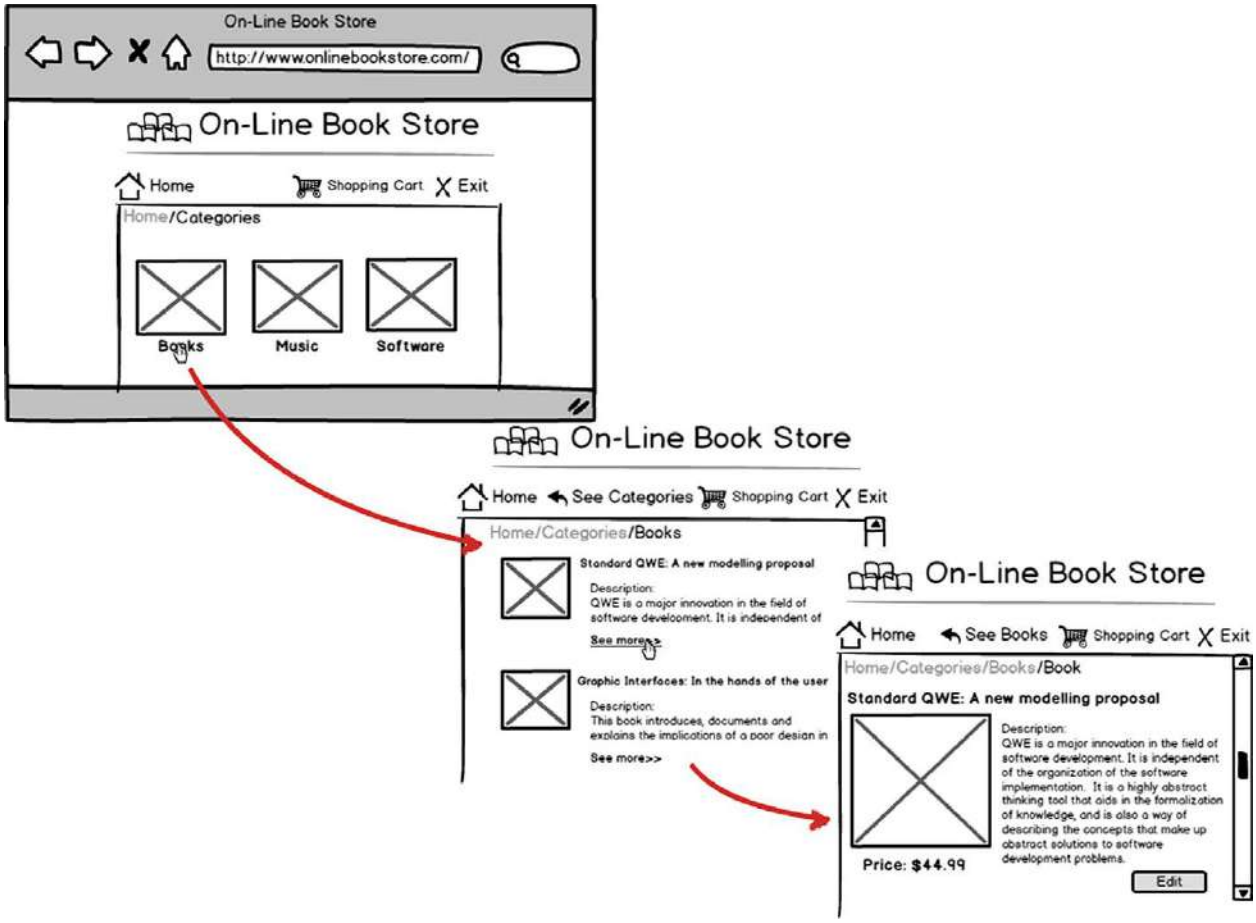
Berikut contoh sederhana IFML. Aplikasi ini adalah toko online tempat user dapat menelusuri produk, seperti buku, musik, dan perangkat lunak, dan menambahkan produk ke keranjang belanja, seperti yang ditunjukkan oleh diagram use case UML pada Gambar 146.



Gambar 146. Use Case Aplikasi Bookstore

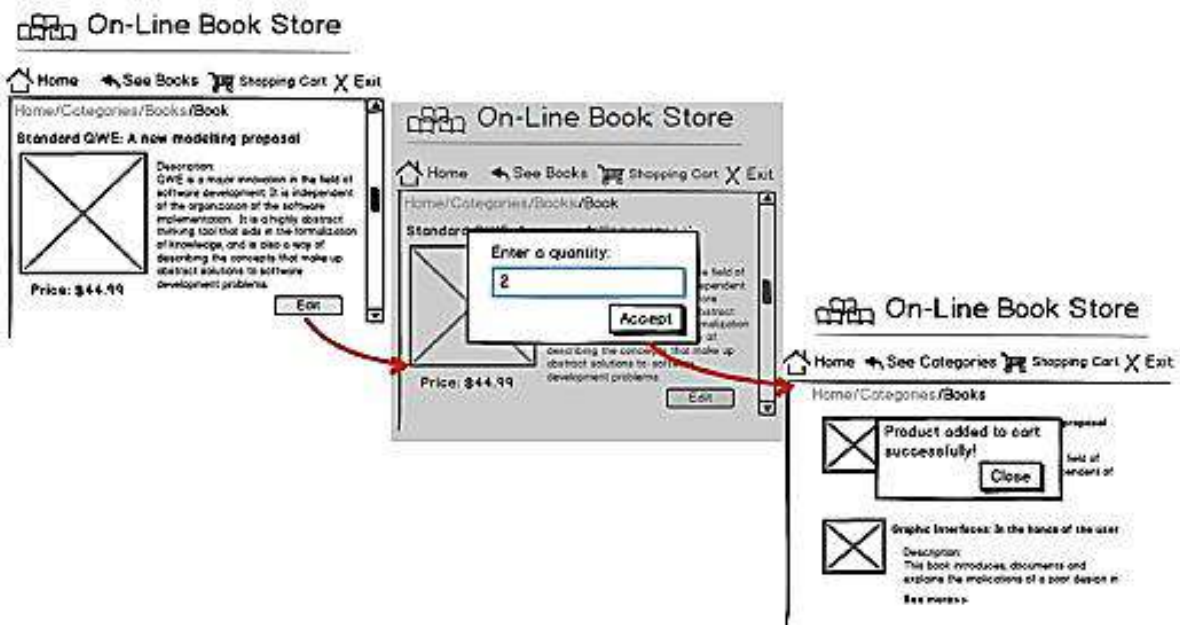
Aplikasi ini memiliki web front end. Dalam use case "*Browse book*", user mengakses halaman Home yang berisi list kategori produk. Mengklik pada kategori produk seperti "*Book*" mengarah ke halaman yang menampilkan data ringkasan tentang semua item dari kategori itu. Mengklik "*See more*" yang terkait dengan ringkasan satu item

membuka halaman di mana rincian lengkap dari objek yang dipilih disajikan. Gambar 147 menunjukkan mock-up dari front end aplikasi yang mendukung case use "Browse books".

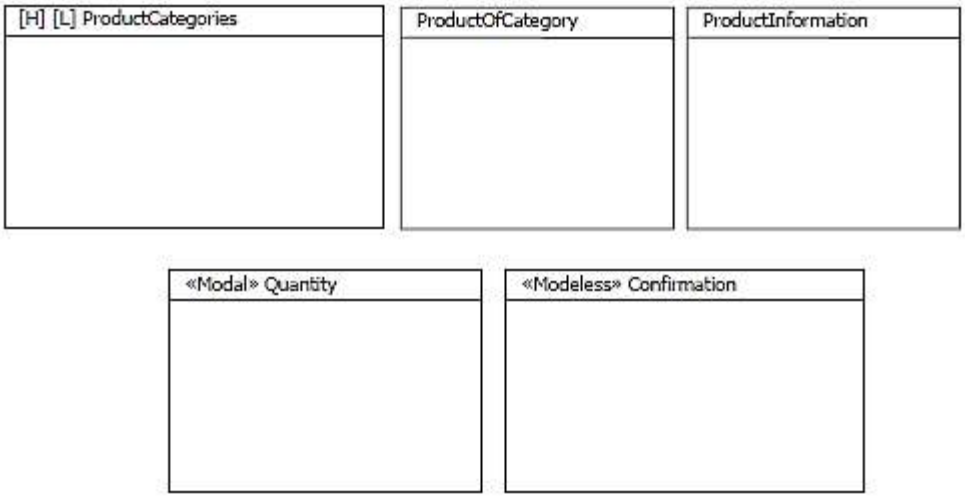


Gambar 147. Mock-ups User Interface Use Case "Browse Book"

Saat melihat detail item, user dapat menekan tombol "Add to chart" untuk menambahkan item ke troli belanja/shopping chart. Window muncul di mana user dapat menentukan jumlah barang yang ingin dia beli. Setelah mengirimkan jumlah yang diinginkan, window pop-up konfirmasi tampil untuk mengetahui penambahan produk ke troli. Gambar 148 menunjukkan mock-up interface use case "Manage cart".



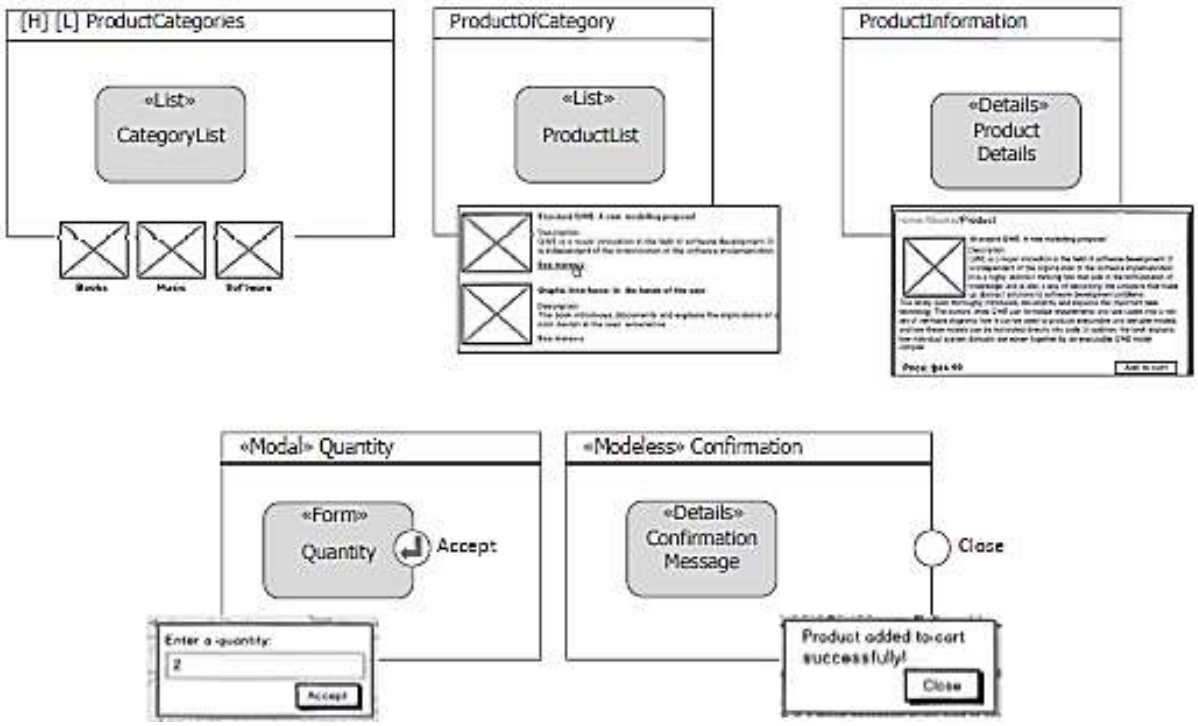
Gambar 148. Mock-Ups User Interface Use Case "Manage Cart"



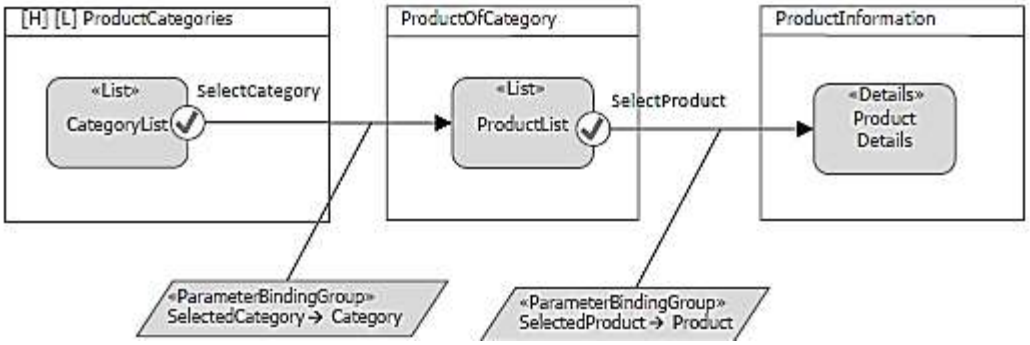
Gambar 149. IFML ViewContainer dari Aplikasi Bookstore

Model IFML dari aplikasi Bookstore berisi lima ViewContainers ditunjukkan pada Gambar 149. ViewContainers dengan stereotype (misal H, untuk "Home," L untuk "Landmark," dan "Modal" dan "Modeless") yang selanjutnya menentukan properti mereka.

Penjelasan ViewContainers disempurnakan dengan menentukan ViewComponents, seperti diilustrasikan dalam Gambar 150.



Gambar 150. View Component Didalam ViweContainer dengan Mock-Up yang Diperbaharui

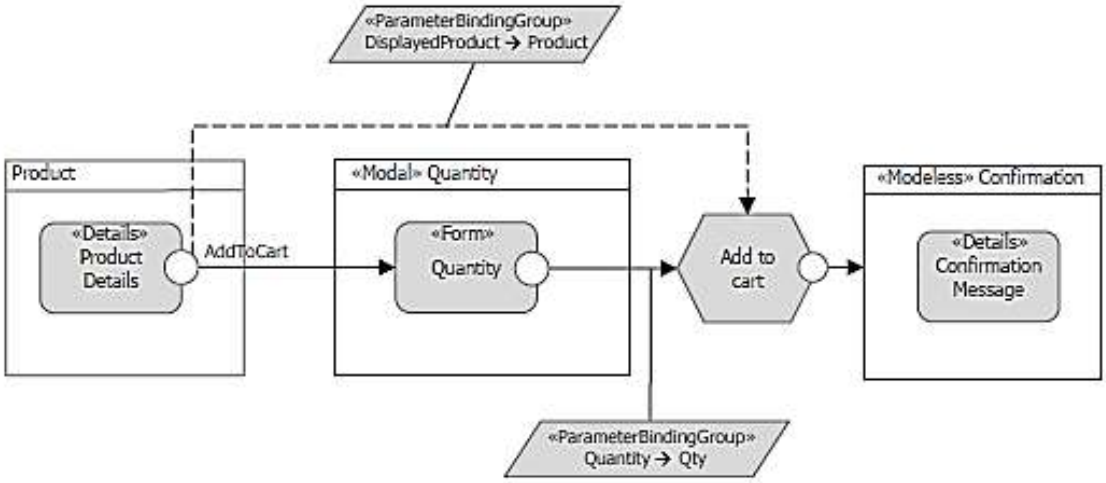


Gambar 151. Event IFML Aliran Interaksi dalam Use Case "Browse Books"

Interaktivitas diwakili dengan menambahkan event yang relevan dan menentukan aliran interaksi yang dipicu, bersama dengan parameter yang mengikat antara sumber dan komponen target dari aliran interaksi. Model Gambar 151 menunjukkan bahwa ViewComponent "CategoryList" terdapat event interaktif "SelectCategory," di mana user dapat memilih kategori dari indeks. Hasilnya, menampilkan page "ProductOfCategory", dan komponen ViewComponent "ProductList" menunjukkan item yang sesuai dengan kategori yang dipilih. Ketergantungan input-output antara "CategoryList" dan "ProductList" diwakili oleh kelompok pengikat parameter, yang mengaitkan parameter output "SelectedCategory" komponen sumber dengan

parameter input "Category" komponen target. Pola pemodelan yang sama digunakan untuk mengekspresikan interaksi untuk memilih produk dari komponen "Product List" dan kemudian mengakses data dalam komponen "ProductDetails".

Beberapa event dapat memicu eksekusi sebagian business logic. Contoh, Gambar 151 dan Gambar 152 menunjukkan aktivasi suatu action untuk memasukkan barang dalam keranjang belanja. Setelah user menekan tombol "Add to chart" terkait dengan komponen "ProductDetails", window muncul meminta jumlah item yang diinginkan. Event memasukkan kuantitas memicu pelaksanaan action "Add to chart". Nilai "quantity" dari Form ViewComponent dan "DisplayedProduct" parameter dari "ProductDetails" ViewComponent diserahkan sebagai parameter input untuk action "Add to chart". Setelah action selesai, menampilkan window konfirmasi.



Gambar 152. Event dan Aliran Interaksi dari Use Case "Browse Product"

Perhatikan bahwa paramater binding output "quantity" dikaitkan dengan aliran interaksi, yang menunjukkan akibat dari event pengiriman yang memerlukan interaksi user. Sebaliknya, paramater binding "DisplayedProduct" berhubungan dengan aliran data, yang menunjukkan ketergantungan input-output secara otomatis dilakukan oleh sistem dan tidak dipicu oleh interaksi user.

KESIMPULAN

SOAL LATIHAN

UNIVERSITAS BUDI LUHUR
FAKULTAS TEKNOLOGI INFORMASI

MODUL PERKULIAHAN #15
FISBONE DIAGRAM

Capaian Pembelajaran	:	Mahasiswa memahami dan menggunakan fishbone diagram
Sub Pokok Bahasan	:	1.73. Model Identifikasi Penyebab Masalah: Fishbone Diagram 1.74. Langkah-Langkah Pembuatan Fishbone Diagram 1.75. Contoh Kasus 1.76. Kesimpulan 1.77. Soal Latihan
Daftar Pustaka	:	

MODEL IDENTIFIKASI PENYEBAB MASALAH: FISHBONE DIAGRAM

Fishbone diagram (diagram tulang ikan — karena *bentuknya* seperti tulang ikan) sering juga disebut **Cause-and-Effect Diagram** atau **Ishikawa Diagram** diperkenalkan oleh Dr. Kaoru Ishikawa, seorang ahli pengendalian kualitas dari Jepang, sebagai satu dari tujuh alat kualitas dasar (*7 basic quality tools*). *Fishbone diagram* digunakan ketika kita ingin mengidentifikasi **kemungkinan penyebab masalah**.

Suatu tindakan dan langkah *improvement* akan lebih mudah dilakukan jika masalah dan akar penyebab masalah sudah ditemukan. Manfaat *fishbone diagram* dapat menolong kita untuk menemukan akar penyebab masalah secara *user friendly*, *tools* yang *user friendly* disukai orang-orang di industri manufaktur di mana proses di sana terkenal memiliki banyak ragam variabel yang berpotensi menyebabkan munculnya permasalahan.

Fishbone diagram akan mengidentifikasi berbagai sebab potensial dari satu efek atau masalah, dan menganalisis masalah tersebut melalui sesi *brainstorming*. Masalah akan dipecah menjadi sejumlah kategori yang berkaitan, mencakup manusia, material, mesin, prosedur, kebijakan, dan sebagainya. Setiap kategori mempunyai sebab-sebab yang perlu diuraikan melalui sesi *brainstorming*.
Untuk lebih jelasnya, diuraikan prosedur atau langkah-langkah pembuatan *fishbone diagram* di bawah ini.

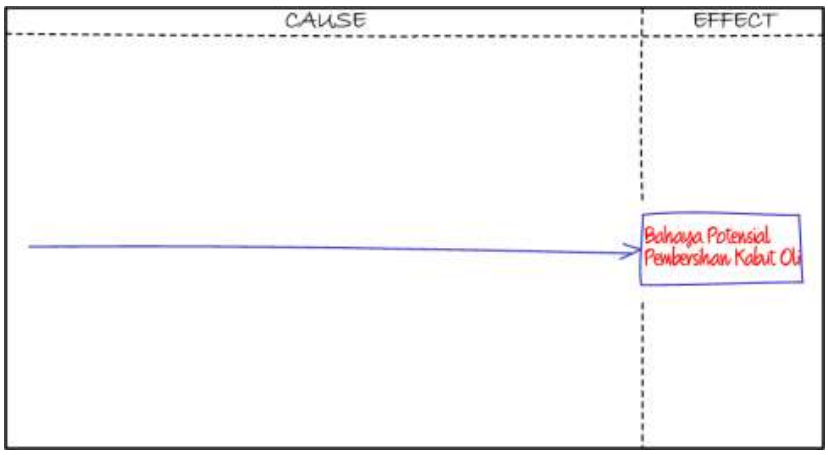
LANGKAH-LANGKAH PEMBUATAN FISHBONE DIAGRAM

Pembuatan *fishbone diagram* kemungkinan akan menghabiskan waktu sekitar 30-60 menit dengan peserta terdiri dari orang-orang yang kira-kira **mengerti/paham tentang masalah** yang terjadi, dan tunjuk satu orang pencatat untuk mengisi *fishbone diagram*. Alat-alat yang perlu disiapkan adalah: *flipchart* atau *whiteboard* dan *marking pens* atau *spidol*.

1.51. LANGKAH 1: MENYEPAKATI PERNYATAAN MASALAH

Sepakati sebuah pernyataan masalah (*problem statement*). Pernyataan masalah ini diinterpretasikan sebagai "effect", atau secara visual dalam *fishbone* seperti "kepala ikan".

Tuliskan masalah tersebut di tengah *whiteboard* di sebelah paling kanan, misal: "Bahaya Potensial Pembersihan Kabut Oli".



Gambar 153. Pembuatan Fishbone Diagram – Meyepakati Pernyataan Masalah

Gambarkan sebuah kotak mengelilingi tulisan pernyataan masalah tersebut dan buat panah horizontal panjang menuju ke arah kotak (lihat Gambar 153).

1.52. LANGKAH 2: MENGIDENTIFIASI KATEGORI-KATEGORI

Dari garis horisontal utama, buat garis diagonal yang menjadi "cabang". Setiap cabang mewakili "sebab utama" dari masalah yang ditulis. Sebab ini diinterpretasikan sebagai "cause", atau secara visual dalam *fishbone* seperti "tulang ikan".

Kategori sebab utama mengorganisasikan sebab sedemikian rupa sehingga masuk akal dengan situasi. Kategori-kategori ini antara lain:

Kategori 6M yang biasa digunakan dalam industri manufaktur:

- ***Machine*** (mesin atau teknologi),
- ***Method*** (metode atau proses),
- ***Material*** (termasuk *raw material*, *consumption*, dan informasi),
- ***Man Power*** (tenaga kerja atau pekerjaan fisik) / ***Mind Power*** (pekerjaan pikiran: *kaizen*, saran, dan sebagainya),

- ***Measurement*** (pengukuran atau inspeksi), dan
- ***Milieu*** / ***Mother Nature*** (lingkungan).

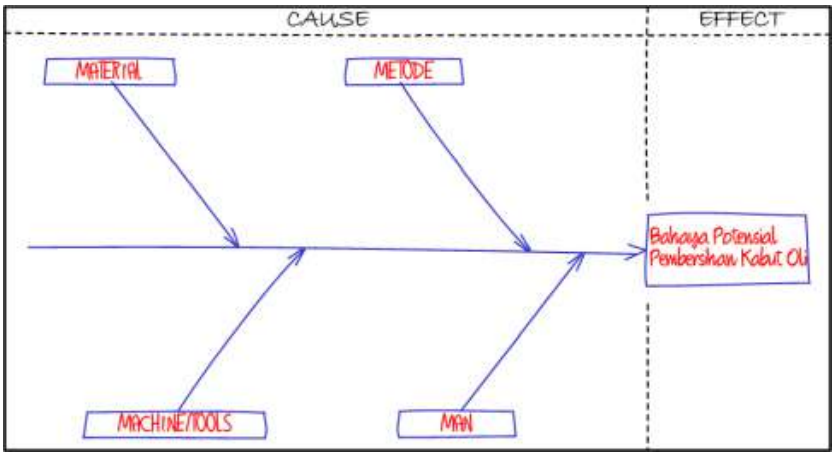
Kategori 8P yang biasa digunakan dalam industri jasa:

- ***Product*** (produk/jasa),
- ***Price*** (harga),
- ***Place*** (tempat),
- ***Promotion*** (promosi atau hiburan),
- ***People*** (orang),
- ***Process*** (proses),
- ***Physical Evidence*** (bukti fisik), dan
- ***Productivity & Quality*** (produktivitas dan kualitas).

Kategori 5S yang biasa digunakan dalam industri jasa:

- ***Surroundings*** (lingkungan),
- ***Suppliers*** (pemasok),
- ***Systems*** (sistem),
- ***Skills*** (keterampilan), dan
- ***Safety*** (keselamatan).

Kategori di atas hanya sebagai saran, kita bisa menggunakan kategori lain yang dapat membantu mengatur gagasan-gagasan. Jumlah kategori biasanya sekitar 4 sampai dengan 6 kategori. Kategori pada contoh ini lihat Gambar 2.



Gambar 154. Pembuatan Fishbone Diagram – Mengidentifikasi Kategori-Kategori

1.53. MENEMUKAN SEBAB-SEBAB POTENSIAL DENGAN CARA BRAINSTORMING

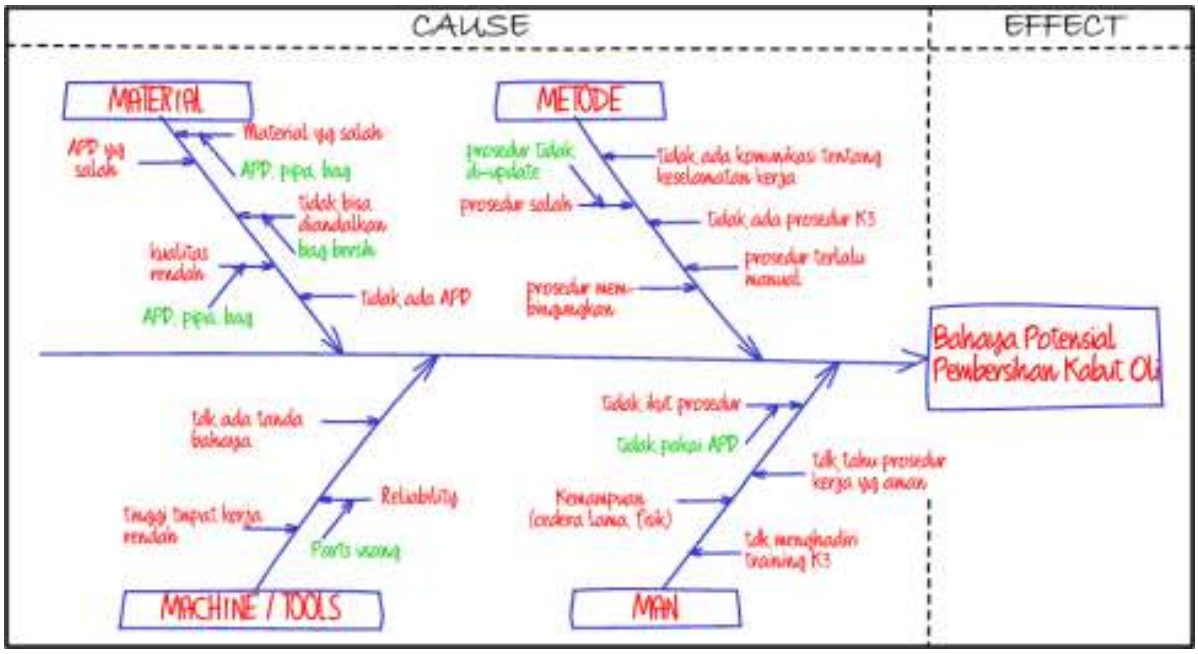
Setiap kategori mempunyai sebab-sebab yang perlu **diuraikan** melalui sesi *brainstorming*.

Saat sebab-sebab dikemukakan, tentukan bersama-sama di mana sebab tersebut harus ditempatkan dalam *fishbone diagram*, yaitu tentukan di bawah kategori yang mana gagasan tersebut harus ditempatkan, misal: "Mengapa bahaya potensial? Penyebab: Karyawan tidak mengikuti prosedur!" Karena penyebabnya karyawan (manusia), maka diletakkan di bawah "Man".

Sebab-sebab ditulis dengan garis horisontal sehingga banyak "tulang" kecil keluar dari garis diagonal.

Pertanyakan kembali "Mengapa sebab itu muncul?" sehingga "tulang" lebih kecil (sub-sebab) keluar dari garis horisontal tadi, misal: "Mengapa karyawan disebut tidak mengikuti prosedur? Jawab: karena tidak memakai APD" (lihat Gambar 3).

Satu sebab bisa ditulis di beberapa tempat jika sebab tersebut berhubungan dengan beberapa kategori.



Gambar 155. Pembuatan Fishbone Diagram – Menemukan Sebab-Sebab Potensial

1.54. LANGKAH 4: MENGKAJI DAN MENYEPAKATI SEBAB-SEBAB YANG PALING MUNGKIN

Setelah setiap kategori diisi carilah sebab yang paling mungkin di antara semua sebab-sebab dan sub-subnya.

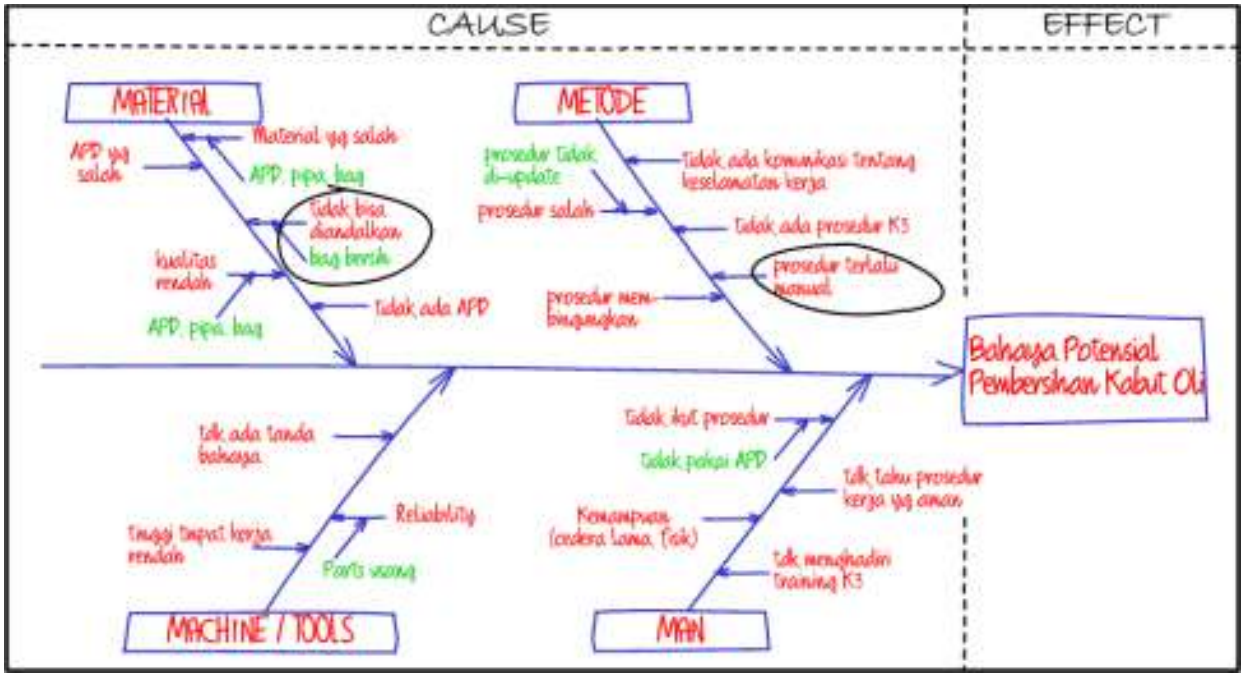
Jika ada sebab-sebab yang muncul pada lebih dari satu kategori, kemungkinan merupakan petunjuk sebab yang paling mungkin.

Kaji kembali sebab-sebab yang telah didaftarkan (sebab yang tampaknya paling memungkinkan) dan tanyakan , "Mengapa ini sebabnya?"

Pertanyaan "Mengapa?" akan membantu kita sampai pada sebab pokok dari permasalahan teridentifikasi.

Tanyakan "Mengapa ?" sampai saat pertanyaan itu tidak bisa dijawab lagi. Kalau sudah sampai ke situ sebab pokok telah terindentifikasi.

Lingkarilah sebab yang tampaknya paling memungkin pada *fishbone diagram* (lihat Gambar 4).



Gambar 156. Pembuatan Fishbone Diagram – Melingkari Sebab Yang Paling Mungkin

Diskusi selama sesi *brainstorming* hendaknya dirangkum, seperti terlihat pada Tabel 1 di bawah ini.

Contoh Tabel Rangkuman diskusi pada sesi brainstorming fishbone diagram

Table 2. Hasil Diskusi Selama Brainstorming

Possible Root Cause	Discussion	Root Cause?
MAN		
Kemampuan karyawan melakukan tugas (cedera lama, fisik)	Cedera personil teridentifikasi saat briefing K3*. Pelaksanaan tugas tidak tergantung pada fisik.	N
Tidak tahu prosedur K3	*Awareness* training di OJT sudah disediakan	N
Tidak mengikuti prosedur K3	Karyawan baru di-*briefing* K3 dan sistem penalti	N
Tidak menghadiri training K3	Pelatihan K3 diberikan dalam orientasi dan OJT	N
MACHINE / TOOLS		
Tinggi tempat kerja rendah	Bukan akar masalah jika metode dapat diubah	N
Part sudah usang	Tidak ada *part* usang menyebabkan insiden	N
Tidak ada tanda bahaya	Tanda bahaya sudah ada	N
METHOD		
Prosedur tidak diperbaharui	Review prosedur rutin setahun sekali	N
Tidak ada prosedur K3	Prosedur meliputi prosedur K3 untuk semua kegiatan	N
Prosedur K3 salah	Prosedur sudah ditinjau oleh supervisor, manajer, dept. head	N

Possible Root Cause	Discussion	Root Cause?
Prosedur K3 membingungkan	Prosedur sudah ditinjau oleh supervisor, manajer, dept. head	N
Prosedur terlalu manual	*Bag* dipegang operator, perlu memastikan tidak ada kebocoran oli, dll.	Y
Tidak ada komunikasi K3	Disertakan dalam OJT	N
MATERIAL		
APD** yang salah	Verifikasi dengan vendor sebelum membeli	N
Material yang tidak bisa diandalkan bahan *(bag* kimia)	*Bag* plastik rentan robek bila menyentuh objek tajam	Y
Kualitas rendah (pipa, APD, *bag*kimia)	Verifikasi dengan vendor sebelum membeli	N
Material yang digunakan salah (pipa, APD, *bag* kimia)	Verifikasi dengan vendor sebelum membeli	N
Tidak ada APD yang disediakan	APD sudah disediakan untuk semua aktivitas berbahaya	N

*) K3 = Kesehatan dan Keselamatan Kerja

**) APD = Alat Pelindung Diri

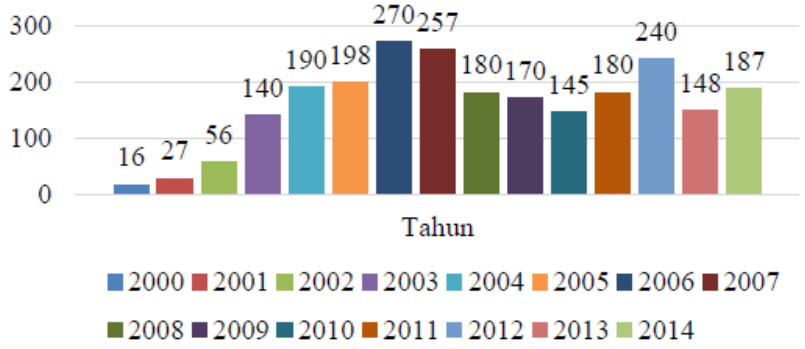
Dari contoh di atas, *fishbone diagram* dapat menemukan akar permasalahan, yaitu kabut oli selama ini dibersihkan dengan ditampung di *bag* plastik yang rentan robek dan selama tidak ada *bag* plastik ada kemungkinan oli menetes jika kran rusak, solusi bisa dengan menambahkan *containment tray* atau *safety cabinet* yang permanen menempel pada pipa.

Jika masalah rumit dan waktunya memungkinkan, kita bisa meninggalkan *fishbone diagram* di dinding selama beberapa hari untuk membiarkan ide menetas dan

membiarkan orang yang lalu lalang turut berkontribusi. Jika *fishbone diagram* terlihat timpang atau sempit, kita bisa mengatur ulang *fishbone diagram* dengan kategori sebab utama yang berbeda. Kunci sukses *fishbone diagram* adalah terus bertanya "Mengapa?", lihatlah diagram dan carilah pola tanpa banyak bicara, dan libatkan orang-orang di "grass root"(orang lapangan) yang terkait dengan masalah karena biasanya mereka lebih mengerti permasalahan.

CONTOH KASUS

PT. XYZ merupakan perusahaan jasa pelayanan ibadah haji yang banyak diikuti oleh masyarakat kota Palembang. Hal tersebut dapat dilihat pada grafik yang menunjukkan bahwa jumlah jamaah haji yang telah berangkat menunaikan ibadah haji bersama PT.XYZ konsisten dalam angka rata-rata 180 jamaah haji sejak 5 tahun terakhir.



Gambar 157. Grafik Jumlah Jamaah Haji PT. XYZ

Oleh karena itu banyaknya masyarakat memilih PT. XYZ untuk membimbing pelaksanaan ibadah haji, diperlukan sebuah Aplikasi untuk mempermudah bagian administrasi dalam melakukan pengelolaan data jamaah haji.

Saat ini, PT. XYZ belum memiliki aplikasi khusus untuk pengelolaan data jamaah haji. Adapun prosedur yang sedang berjalan dengan pemanfaatan teknologi yang sudah dilakukan pada PT. Arraudhah Wisata Imani Palembang adalah sebagai berikut.

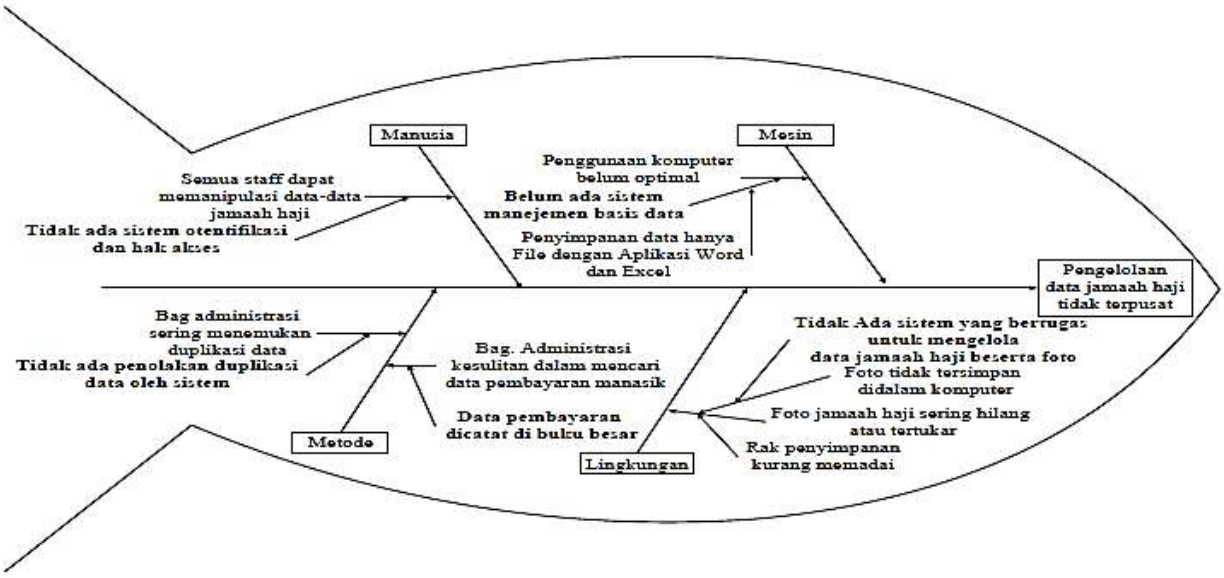
1. Penggunaan Ms. Word untuk pencatatan pendaftaran haji.
2. Penggunaan Ms. Excel untuk pembuatan laporan dan rekapitulasi.
3. Penggunaan buku besar untuk pencatatan akhir data jamaah haji.
4. Penggunaan proyektor dan Ms. PowerPoint dalam memberikan pengarahan manasik haji.

Dari uraian diatas, didapatkan kelebihan dan kekurangan pada prosedur yang sedang berjalan tersebut. Kelebihannya adalah sudah menggunakan teknologi komputer pada beberapa kegiatan perusahaan seperti, pencatatan, pembuatan laporan, dan penggunaan proyektor dalam melakukan manasik haji. Berdasarkan hasil wawancara di kantor PT. XYZ inti permasalahan dari pengelolaan data jamaah haji di PT. XYZ adalah pengelolaan data tidak terpusat.

Dapat diambil kesimpulan bahwa permasalahan yang ada di PT. XYZ adalah data yang sering hilang dan berubah tanpa ada koordinasi dari pihak yang berkepentingan, sering terjadinya kekeliruan penginputan dan duplikasi data yang diinput, dan pencarian data pembayaran serta data jamaah sering tertukar seperti foto jamaah haji dengan jamaah haji lainnnya.

Maka dibuatlah aplikasi pengelolaan data jamaah haji agar bisa mempermudah bagian administrasi dalam melakukan pencarian data dengan cepat dan membantu bagian administrasi dalam melakukan penyimpanan data karena datanya tersimpan dengan terpusat pada database secara otomatis.

Fishbone Diagram:



Cause	Keterangan
Manusia	
Semua staff dapat memanipulasi data-data jamaah haji	*Potential Cause*
Tidak ada sistem otentifikasi dan hak akses	*Most Possible Cause*
Mesin	
Penggunaan komputer belum optimal	*Potential Cause*
Belum ada sistem manajemen basis data	*Most Possible Cause*
Penyimpanan data hanya dengan Ms.Word dan Excel	*Potential Cause*
Metode	
Bag. Administrasi kesulitan dalam mencari data pembayaran manasik	*Potential Cause*
Data pembayaran dicatat di buku besar	*Most Possible Cause*
Bag. Administrasi sering menemukan duplikasi data	*Potential Cause*
Tidak ada penolakan duplikasi data oleh system	*Most Possible Cause*
Lingkungan	
Foto jamaah haji sering hilang atau tertukar	*Potential Cause*
Rak penyimpanan kurang memadai	*Potential Cause*
Foto tidak tersimpan didalam computer	*Potential Cause*
Tidak ada sistem yang bertugas untuk mengelola data jamaah haji beserta foto	*Most Possible Cause*

Dari analisis Fishbone diatas didapatkan bahwa **sebab yang paling mungkin** yang menyebabkan pengelolaan data jamaah haji tidak terpusat adalah sebagai berikut.

1. Tidak ada sistem otentifikasi dan hak akses.
2. Belum ada sistem manajemen basis data.
3. Data pembayaran dicatat di buku besar.
4. Tidak ada penolakan duplikasi data oleh sistem.
5. Tidak ada sistem yang bertugas untuk mengelola data jamaah haji beserta foto.

Dari permasalahan diatas, langkah berikutnya melakukan analisa kebutuhan menggunakan Use Case Diagram. Skenario(case) yang ada di use case diagram akan menjawab permasalahan pada fishbone diagram.

KESIMPULAN

SOAL LATIHAN

FAKULTAS TEKNOLOGI INFORMASI

UNIVERSITAS BUDI LUHUR

Jl. Raya Ciledug, Petukangan Utara, Pesanggrahan

Jakarta Selatan, 12260

Telp: 021-5853753 Fax : 021-5853752

http://fti.budiluhur.ac.id