Pengenalan "Unified Modeling Language/UML"
(Bagian I)

Dalam suatu proses pengembangan software, analisa dan rancangan telah merupakan terminologi yang sangat tua. Pada saat masalah ditelusuri dan spesifikasi dinegoisasikan, dapat dikatakan kita berada pada tahap rancangan. Merancang adalah menemukan suatu cara untuk menyelesaikan masalah, salah satu tool / model untuk merancang pengembangan software yang berbasis object oriented adalah UML.

Konsep Objek

Obyek dalam *'software analysis & design'* adalah sesuatu berupa konsep (*concept*), benda (*thing*), dan sesuatu yang membedakannya dengan lingkungannya. Secara sederhana obyek adalah mobil, manusia, *alarm* dan lain-lainnya. Tapi obyek dapat pula merupakan sesuatu yang abstrak yang hidup didalam sistem seperti tabel, *database, event, system messages.*

Obyek dikenali dari keadaannya dan juga operasinya. Sebagai contoh sebuah mobil dikenali dari warnanya, bentuknya, sedangkan manusia dari suaranya. Ciri-ciri ini yang akan membedakan obyek tersebut dari obyek lainnya.

Alasan mengapa saat ini pendekatan dalam pengembangan software dengan *object-oriented*, pertama adalah *scalability* dimana obyek lebih mudah dipakai untuk menggambarkan sistem yang besar dan komplek. Kedua *dynamic modeling*, adalah dapat dipakai untuk permodelan sistem dinamis dan *real time*.

Teknik Dasar OOA/D *(Object-Oriented Analysis/Design)*

Dalam dunia pemodelan, metodologi implementasi obyek walaupun terikat kaidah-kaidah standar, namun teknik pemilihan obyek tidak terlepas pada subyektifitas software analyst & designer. Beberapa obyek akan diabaikan dan beberapa obyek menjadi perhatian untuk diimplementasikan di dalam sistem. Hal ini sah-sah saja karena kenyataan bahwa suatu permasalahan sudah tentu memiliki lebih dari satu solusi. Ada 3 (tiga) teknik/konsep dasar dalam OOA/D, yaitu pemodulan (*encapsulation*), penurunan (*inheritance)* dan *polymorphism.*

a. Pemodulan (*Encapsulation*)

Pada dunia nyata, seorang ibu rumah tangga menanak nasi dengan *menggunakan rice cooker*, ibu tersebut menggunakannya hanya dengan menekan tombol. Tanpa harus tahu bagaimana proses itu sebenarnya terjadi. Disini terdapat penyembunyian informasi milik *rice cooker*, sehingga tidak perlu diketahui seorang ibu. Dengan demikian menanak nasi oleh si ibu menjadi sesuatu yang menjadi dasar bagi konsep *information hiding*.

b. Penurunan (*Inheritance*)

Obyek-obyek memiliki banyak persamaan, namun ada sedikit perbedan. Contoh dengan beberapa buah mobil yang mempunyai kegunaan yang berbeda-beda. Ada mobil bak terbuka seperti truk, bak tertutup seperti sedan dan minibus. Walaupun demikian obyek-obyek ini memiliki kesamaan yaitu teridentifikasi sebagai obyek mobil, obyek ini dapat dikatakan sebagai obyek induk (*parent*). Sedangkan minibus dikatakan sebagai obyek anak (*child*), hal ini juga berarti semua operasi yang berlaku pada mobil berlaku juga pada minibus.

c. *Polymorphism*

Pada obyek mobil, walaupun minibus dan truk merupakan jenis obyek mobil yang sama, namun memiliki juga perbedaan. Misalnya suara truk lebih keras dari pada minibus, hal ini juga berlaku pada obyek anak (*child*) melakukan metoda yang sama dengan algoritma berbeda dari obyek induknya. Hal ini yang disebut *polymorphism*, teknik atau konsep dasar lainnya adalah ruang lingkup / pembatasan. Artinya setiap obyek mempunyai ruang lingkup kelas, atribut, dan metoda yang dibatasi.

Sejarah Singkat UML

UML (*Unified Modeling Language*) adalah sebuah bahasa yang berdasarkan grafik/gambar untuk memvisualisasi, menspesifikasikan, membangun, dan pendokumentasian dari sebuah sistem pengembangan software berbasis OO (*Object-Oriented*). UML sendiri juga memberikan standar penulisan sebuah sistem blue print, yang meliputi konsep bisnis proses, penulisan kelas-kelas dalam bahasa program yang spesifik, skema database, dan komponen-komponen yang diperlukan dalam sistem software (http://www.omg.org).

Pendekatan analisa & rancangan dengan menggunakan model OO mulai diperkenalkan sekitar pertengahan 1970 hingga akhir 1980 dikarenakan pada saat itu aplikasi software sudah meningkat dan mulai komplek. Jumlah yang menggunakaan metoda OO mulai diuji cobakandan diaplikasikan antara 1989 hingga 1994, seperti halnya oleh Grady Booch dari *Rational Software Co.,* dikenal dengan OOSE (*Object-Oriented Software Engineering*), serta James Rumbaugh dari *General Electric*, dikenal dengan OMT (*Object Modelling Technique*).

Kelemahan saat itu disadari oleh Booch maupun Rumbaugh adalah tidak adanya standar penggunaan model yang berbasis OO, ketika mereka bertemu ditemani rekan lainnya Ivar Jacobson dari Objectory mulai mendiskusikan untuk mengadopsi masing-masing pendekatan metoda OO untuk membuat suatu model bahasa yang uniform / seragam yang disebut UML (*Unified Modeling Language*) dan dapat digunakan oleh seluruh dunia.

Secara resmi bahasa UML dimulai pada bulan oktober 1994, ketika Rumbaugh bergabung Booch untuk membuat sebuah project pendekatan metoda yang uniform/seragam dari masing-masing metoda mereka. Saat itu baru dikembangkan draft metoda UML version 0.8 dan diselesaikan serta di release pada bulan oktober 1995. Bersamaan dengan saat itu, Jacobson bergabung dan UML tersebut diperkaya ruang lingkupnya dengan metoda OOSE sehingga muncul release version 0.9 pada bulan Juni 1996. Hingga saat ini sejak Juni 1998 UML version 1.3 telah diperkaya dan direspons oleh OMG (Object Management Group), Anderson Consulting, Ericsson, Platinum Technology, ObjectTime Limited, dll serta di pelihara oleh OMG yang dipimpin oleh Cris Kobryn.
UML adalah standar dunia yang dibuat oleh *Object Management Group* (OMG), sebuah badan yang bertugas mengeluarkan standar-standar teknologi *object-oriented* dan *software component*.

3. Pengenalan UML

UML sebagai sebuah bahasa yang memberikan *vocabulary* dan tatanan penulisan kata-kata dalam *'MS Word'* untuk kegunaan komunikasi. Sebuah bahasa model adalah sebuah bahasa yang mempunyai vocabulary dan konsep tatanan / aturan penulisan serta secara fisik mempresentasikan dari sebuah sistem. Seperti halnya UML adalah sebuah bahasa standard untuk pengembangan sebuah software yang dapat menyampaikan bagaimana membuat dan membentuk model-model, tetapi tidak menyampaikan apa dan kapan model yang seharusnya dibuat yang merupakan salah satu proses implementasi pengembangan software.

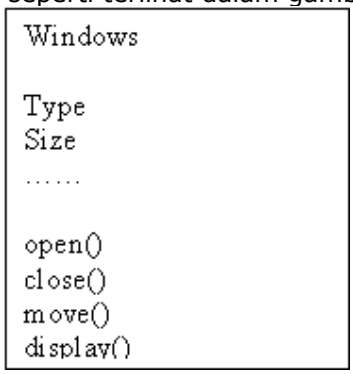
UML tidak hanya merupakan sebuah bahasa pemograman visual saja, namun juga dapat secara langsung dihubungkan ke berbagai bahasa pemograman, seperti JAVA, C++, Visual Basic, atau bahkan dihubungkan secara langsung ke dalam sebuah object-oriented database. Begitu juga mengenai pendokumentasian dapat dilakukan seperti; *requirements*, arsitektur, *design, source code, project plan, tests*, dan *prototypes*.
Untuk dapat memahami UML membutuhkan bentuk konsep dari sebuah bahasa model, dan mempelajari 3 (tiga) elemen utama dari UML seperti *building block*, aturan-aturan yang menyatakan bagaimana *building block* diletakkan secara bersamaan, dan beberapa mekanisme umum (common).

a. Building blocks

3 (tiga) macam yang terdapat dalam building block adalah katagori benda/Things, hubungan, dan diagram. Benda/things adalah abstraksi yang pertama dalam sebuah model, hubungan sebagai alat komunikasi dari benda-benda, dan diagram sebagai kumpulan / group dari benda-benda/things.

- Benda/Things

Adalah hal yang sangat mendasar dalam model UML, juga merupakan bagian paling statik dari sebuah model, serta menjelaskan elemen-elemen lainnya dari sebuah konsep dan atau fisik. Bentuk dari beberapa benda/thing adalah sebagai berikut:
Pertama, adalah sebuah kelas yang diuraikan sebagai sekelompok dari object yang mempunyai atribute, operasi, hubungan yang semantik. Sebuah kelas mengimplementasikan 1 atau lebih interfaces. Sebuah kelas dapat digambarkan sebagai sebuah persegi panjang, yang mempunyai sebuah nama, atribute, dan metoda pengoperasiannya, seperti terlihat dalam gambar 1.



Gambar 1. Sebuah Kelas dari model UML

Kedua, yang menggambarkan *'interface'* merupakan sebuah antar-muka yang menghubungkan dan melayani antar kelas dan atau elemen. *'Interface'* / antar-muka mendefinisikan sebuah set / kelompok dari spesifikasi pengoperasian, umumnya digambarkan dengan sebuah lingkaran yang disertai dengan namanya. Sebuah antar-muka berdiri sendiri dan umumnya merupakan pelengkap dari kelas atau komponen, seperti dalam gambar 2.

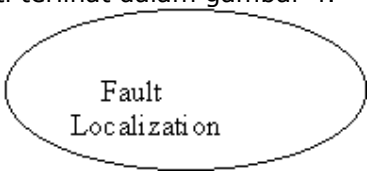
ISpelling

Gambar 2. Sebuah interface/antar-muka

Ketiga, adalah *collaboration* yang didefinisikan dengan interaksi dan sebuah kumpulan / kelompok dari kelas-kelas/elemen-elemen yang bekerja secara bersama-sama. C*ollaborations* mempunyai struktura dan dimensi. Pemberian sebuah kelas memungkinkan berpartisipasi didalam beberapa *collaborations* dan digambarkan dengan sebuah *'elips'* dengan garis terpotong-potong.

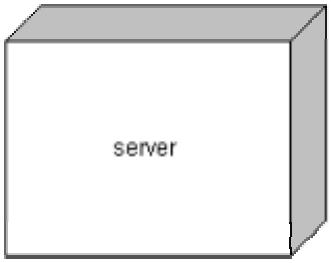
Gambar 3. Collaborations

Keempat, sebuah '*use case*' adalah rangkaian/uraian sekelompok yang saling terkait dan membentuk sistem secara teratur yang dilakukan atau diawasi oleh sebuah aktor. '*use case*' digunakan untuk membentuk tingkah-laku benda/ *things* dalam sebuah model serta di realisasikan oleh sebuah collaboration. Umumnya *'use case'* digambarkan dengan sebuah *'elips'* dengan garis yang solid, biasanya mengandung nama, seperti terlihat dalam gambar 4.



Gambar 4. Use Case

Kelima, sebuah node merupakan fisik dari elemen-elemen yang ada pada saat dijalankannya sebuah sistem, contohnya adalaha sebuah komputer, umumnya mempunyai sedikitnya *memory* dan *processor*. Sekelompok komponen mungkin terletak pada sebuah *node* dan juga mungkin akan berpindah dari node satu ke node lainnya. Umumnya node ini digambarkan seperti kubus serta hanya mengandung namanya, seperti terlihat dalam gambar 5.



Gambar 5. Nodes

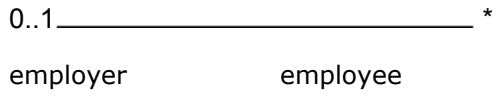
- **Hubungan / Relationship**

Ada 4 macam hubungan didalam penggunaan UML, yaitu; *dependency, association, generalization*, dan *realization.*

Pertama, sebuah *dependency* adalah hubungan semantik antara dua benda/things yang mana sebuah benda berubah mengakibatkan benda satunya akan berubah pula. Umumnya sebuah *dependency* digambarkan sebuah panah dengan garis terputus-putus seperti terlihat dalam gambar 6.

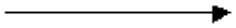
Gambar 6. *Dependency*

Kedua, sebuah *association* adalah hubungan antar benda struktural yang terhubung diantara obyek. Kesatuan obyek yang terhubung merupakan hubungan khusus, yang menggambarkan sebuah hubungan struktural diantara seluruh atau sebagian. Umumnya assosiation digambarkan dengan sebuah garis yang dilengkapi dengan sebuah label, nama, dan status hubungannya seperti terliahat dalam gambar 7.



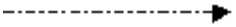
Gambar 7. Association

Ketiga, sebuah *generalization* adalah menggambarkan hubungan khusus dalam obyek anak/*child* yang menggantikan obyek *parent* / induk . Dalam hal ini, obyek anak memberikan pengaruhnya dalam hal struktur dan tingkah lakunya kepada obyek induk. Digambarkan dengan garis panah seperti terlihat dalam gambar 8.



Gambar 8. Generalizations

Keempat, sebuah *realization* merupakan hubungan semantik antara pengelompokkan yang menjamin adanya ikatan diantaranya. Hubungan ini dapat diwujudkan diantara *interface* dan kelas atau *elements*, serta antara *use cases* dan *collaborations*. Model dari sebuah hubungan realization seperti terlihat dalam gambar 9.



Gambar 9. Realizations

- **Diagram**

UML sendiri terdiri atas pengelompokkan **diagram-diagram** sistem menurut aspek atau sudut pandang tertentu. Diagram adalah yang menggambarkan permasalahan maupun solusi dari permasalahan suatu model. UML mempunyai 9 diagram, yaitu; *use-case, class, object, state, sequence, collaboration, activity, component,* dan *deployment diagram.*

Diagram pertama adalah *use case* menggambarkan sekelompok *use cases* dan aktor yang disertai dengan hubungan diantaranya. Diagram *use cases* ini menjelaskan dan menerangkan kebutuhan / requirement yang diinginkan/ dikehendaki *user*/pengguna, serta sangat berguna dalam menentukan struktur organisasi dan model dari pada sebuah sistem.

Pengenalan "Unified Modelling Language/UML"
(Bagian II)

Dalam suatu proses pengembangan software, analisa dan rancangan telah merupakan terminologu yang sangat tua. Pada saat masalah ditelusuri dan spesifikasi dinegosiasikan, dapat dikatakan bahwa kita berada pada tahap rancangan. merancang adalah menemukan suatu cara untuk menyelesaikan masalah, salah satu tool/model untuk merancang pengembangan software yang berbasis object-oriented adalah UML. Alasan mengapa UML digunakan adalah, pertama, scalability dimana objek lebih mudah dipakai untuk menggambarkan sistem yang besar dan komplek. Kedua, dynamic modeling, dapat dipakai untuk pemodelan sistem dinamis dan real time.
Sebagaimana dalam tulisan pertama, penulis menjelaskan konsep mengenai obyek, OOA&D (Obyek Oriented Analyst/ Design) dan pengenalan UML, maka dalam tulisan kedua ini lebih ditekankan pada cara bagaimana UML digunakan dalam merancang sebuah pengembangan software yang disertai gambar atau contoh dari sebuah aplikasi.

1. Use Case

Sebuah *use case* menggambarkan suatu urutan interaksi antara satu atau lebih aktor dan sistem. Dalam fase *requirements,* model *use case* mengambarkan sistem sebagai sebuah kotak hitam dan interaksi antara aktor dan sistem dalam suatu bentuk naratif, yang terdiri dari input user dan respon-respon sistem. Setiap use case menggambarkan perilaku sejumlah aspek sistem, tanpa mengurangi struktur internalnya. Selama pembuatan model use case secara pararel juga harus ditetapkan obyek-obyek yang terlibat dalam setiap use case.

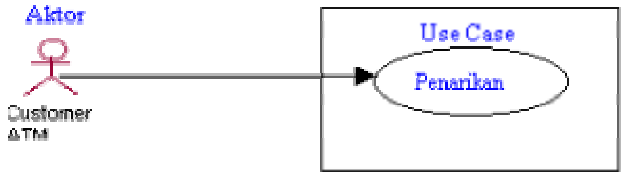
Perhatikan satu contoh sederhana dari proses perbankan, yaitu mesin teller otomatis (Automated Teller Machine-ATM) yang memberikan kemudahan pada customernya untuk mengambil uang dari rekening bank secara langsung. Pada proses ini terdapat satu aktor, yaitu *ATM Customer* dan satu use case, yaitu *Penarikan Dana.* Proses ini dapat dilihat pada Gambar 1. Use case Penarikan Dana menggambarkan urutan interaksi antara customer dengan sistem, diawali ketika customer memasukan kartu ATM ke dalam mesin pembaca kartu dan akhirnya menerima pengeluaran uang yang dilakukan oleh mesin ATM.

2. Aktor

Sebuah aktor mencirikan suatu bagian *outside user* atau susunan yang berkaitan dengan user yang berinteraksi dengan sistem [Rumbaugh, Booch, dan Jacobson 1999]. Dalam model use case, aktor merupakan satu-satunya kesatuan eksternal yang berinteraksi dengan sistem.

Terdapat beberapa variasi bagaimana aktor dibentuk [Fowler dan Scott 1999]. Sebuah aktor sering kali merupakan manusia (*human user*). Pada sejumlah sistem informasi, manusia adalah satu-satunya aktor. Dan mungkin saja dalam sistem informasi, seorang aktor bisa saja menjadi suatu sistem eksternal. Pada aplikasi real-time dan distribusi, sebuah aktor bisa saja menjadi satu perangkat eksternal I/O atau sebuah alat pengatur waktu. Perangkat eksternal I/O dan pengatur waktu aktor secara khusus lazimnya berada dalam real-time yang tersimpan dalam sistem (*real-time embedded systems*), sistem berinteraksi dengan lingkungan eksternal melalui sensor dan aktuator.

Primary actor (aktor utama) memprakarsai sebuah use case. Jadi, suatu primary aktor memegang peran sebagai proaktif dan yang memulai aksi dalam sistem. Aktor lainnya yang berperan sebagai *secondary* aktor bisa saja terlibat dalam use case dengan menerima *output* dan memberikan *input*. Setidaknya satu aktor harus mendapatkan nilai dari use case. Biasanya adalah *primary* aktor (aktor utama). Bagaimanapun, dalam *real-time embedded systems*, *primary* aktor dapat berperan sebagai perangkat eksternal I/O atau pengatur waktu, penerima utama dari *use case* bisa menjadi *secondary human* aktor yang menerima sejumlah informasi dari sistem.



Gambar 1. Contoh aktifitas Aktor dan Use Case

Aktor manusia bisa saja menggunakan berbagai perangkat I/O untuk berinteraksi fisik dengan sistem. Aktor manusia dapat berinteraksi dengan sistem melalui perangkat standar I/O, seperti keyboard, display, atau mouse. Aktor manusia bisa juga berinteraksi dengan sistem melalui perangkat non-standar I/O seperti bermacam-macam sensor. Dalam keseluruhan hal tersebut, manusia merupakan aktor dan perangkat I/O adalah *bukan* aktor.

Perhatikan beberapa contoh human aktor (aktor manusia). Pada sistem perbankan, satu contoh aktor adalah manusia yang berperan sebagai teller yang berinteraksi dengan sistem melalui perangkat standar I/O, seperti keyboard, display, atau mouse. Contoh lainnya adalah manusia yang berperan sebagai *customer* yang berinteraksi dengan sistem melalui mesin teller otomatis (ATM). Dalam hal ini, customer berinteraksi dengan sistem dengan menggunakan beberapa perangkat I/O, termasuk perangkat pembaca kartu (*card reader*), pengeluar uang (*cash dispenser*), dan pencetak tanda terima (*receipt printer*), ditambah lagi keyboard dan display.

Pada beberapa kasus, bagaimana pun juga sebuah aktor bisa saja berupa perangkat I/O. Hal ini bisa terjadi ketika sebuah use case tidak melibatkan manusia, seperti yang sering terjadi pada aplikasi-aplikasi *real-time*. Dalam hal ini, I/O aktor berinteraksi dengan sistem melalui sebuah sensor. Contoh aktor yang merupakan perangkat input adalah *Arrival Sensor* pada Sistem Kontrol Elevator. Sensor ini mengidentifikasi elevator tersebut pada saat hendak mencapai lantai dan perlu dihentikan. Kemudian sensor tersebut menginisiasikan *Stop Elevator at Floor use case*. Aktor lain dalam *Elevator Control System* adalah orang yang berada dalam elevator (*human passenger*) yang berinteraksi dengan sistem melalui tombol-tombol nomor pada tingkat lantai dan tombol-tombol elevator. Input dari aktor secara aktual dideteksi melalui sensor-sensor tombol lantai dan sensor-sensor tombol elevator berturut-turut.

Aktor dapat pula menjadi sebuah alat pengukur waktu yang secara periodik mengirimkan pengukuran waktu kejadian (*timer events*) pada sistem. Use case-

use case secara periodik diperlukan ketika beberapa informasi perlu di-*output* oleh sistem pada suatu basis reguler. Hal ini sangat penting dalam sistem-sistem real-time, dan juga sangat berguna dalam sistem informasi. Walaupun sejumlah metodologi menganggap pengukur waktu merupakan hal internal bagi sistem, dan akan lebih berguna dalam desain aplikasi real-time untuk memperhatikan pengukur-pengukur waktu sebagai eksternal logis bagi sistem dan menganggapnya sebagai primary aktor yang memulai aksi dalam sistem. Contohnya, pada sistem monitoring mobil, beberapa use case di-inisialisasi dengan suatu aktor pengukur waktu. Sebagai contoh dapat dilihat pada Gambar 2. Timer aktor mengawali *Calculate Trip Speed* use case, yang secara periodik menghitung rata-rata kecepatan melalui suatu jalan/ jejak dan menampilkan nilai ini ke *driver*. Dalam hal ini, pengukur waktu merupakan primary aktor (aktor utama) dan *driver* merupakan secondary aktor (aktor kedua).

Gambar 2. Contoh aktor pengukur waktu

Suatu aktor bisa juga menjadi sistem eksternal yang melakukan inisiatif (sebagai primary aktor) atau partisipan (sebagai secondary aktor) dalam use case. Satu contoh aktor sistem eksternal adalah pabrik robot dalam *Automation System*. Robot mengawali proses dengan use case *Generate Alarm* dan *Notify*, robot menggerakkan *alarm conditions* yang dikirim ke operator pabrik yang berkepentingan, yang telah terdaftar untuk menerima alarms. Dalam use case ini, robot merupakan primary aktor yang mengawali inisiatif use case, dan operator merupakan secondary aktor yang menerima *alarms*.

3. Identifikasi Use Case

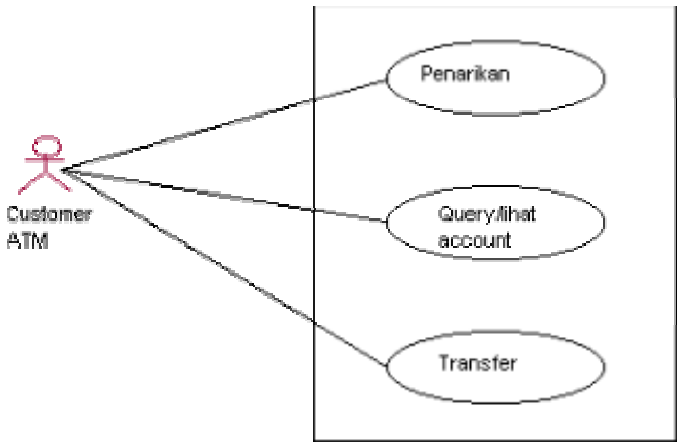
Sebuah use case dimulai dengan masukan/input dari seorang aktor. Use case merupakan suatu urutan lengkap kejadian-kejadian yang diajukan oleh seorang aktor, dan spesifikasi interaksi antara aktor dengan sistem. Use case yang sederhana hanya melibatkan satu interaksi/hubungan dengan sebuah aktor, dan use case yang lebih kompleks melibatkan beberapa interaksi dengan aktor. Use cases yang lebih kompleks juga melibatkan lebih dari satu aktor.

Untuk menjabarkan use case dalam sistem, sangat baik bila dimulai dengan memperhatikan aktor dan actions/aksi yang mereka lakukan dalam sistem. Setiap use case menggambarkan suatu urutan interaksi antara aktor dengan sistem. Sebuah use case harus memberikan sejumlah nilai pada satu aktor.

Kemudian, kebutuhan fungsional sistem dijelaskan dalam use case yang merupakan suatu spesifikasi eksternal dari sebuah sistem. Bagaimanapun juga, ketika membuat use case, sangatlah penting menghindari suatu dekomposisi fungsional yang dalam beberapa use case kecil lebih menjelaskan fungsi-fungsi

individual sistem daripada menjelaskan urutan kejadian yang memberikan hasil yang berguna bagi aktor.

Perhatikan lagi contoh pada perbankan. Disamping penarikan melalui ATM, *ATM Customer*, aktor juga bisa menanyakan jumlah rekening atau mentransfer dana antar dua rekening. Karena terdapat fungsi-fungsi yang berbeda yang diajukan oleh customer dengan hasil-hasil guna yang berbeda, fungsi-fungsi pertanyaan dan pentransferan harus dibuat sebagai use case yang terpisah, daripada menjadi bagian dari original use case. Oleh karena itu, *customer* dapat mengajukan tiga use case seperti yang dapat dilihat di Gambar. 3; *Withdraw Funds* (Penarikan dana), *Query Account*, dan *Transfer Funds* (Pentransferan Dana).



Gambar 3: Aktor dan use case dalam sistem Bank

Urutan utama use case menjelaskan urutan interaksi yang paling umum antara aktor dan sistem. Dan mungkin saja terdapat cabang-cabang urutan use case utama, yang mengarah pada berkurangnya frekuensi interaksi antara aktor dengan sistem. Deviasi-deviasi dari urutan utama hanya dilaksanakan pada beberapa situasi, contohnya jika aktor melakukan kesalahan input pada sistem. Ketergantungan pada aplikasi kebutuhan, alternatif ini memecahkan use case dan kadang-kadang bersatu kembali dengan urutan utama. Cabang-cabang alternatif digambarkan juga dalam use case.

Dalam use case *Withdraw Funds*, urutan utama adalah urutan tahap-tahap dalam keberhasilan pelaksanaan penarikan (*withdrawal*). Cabang-cabang alternatif digunakan untuk mengarahkan berbagai *error cases*, seperti ketika kartu ATM tidak dikenali atau dilaporkan telah hilang dan lain sebagainya.

4. Pendokumentasian Model Use Case

Use case didokumentasi dalam *use case model* sebagai berikut:

- **Use Case Name**. Setiap use case diberi nama.
- **Summary.** Deskripsi singkat use case, biasanya satu atau dua kalimat.
- **Dependency**. Bagian ini menggambarkan apakah use case yang satu tergantung pada use case yang lain, dalam arti apakah use case tersebut termasuk pada use case yang lain atau malah memperluas use case lain.

- **Actors.** Bagian ini memberikan nama pada actor dalam use case. Selalu terdapat use case utama (primary use case) yang memulai use case. Disamping itu terdapat juga secondary use case yang terlibat dalam use case. Contohnya, dalam use case Withdraw Funds, ATM Customer adalah actor-nya.

- **Preconditions.** Satu atau lebih kondisi harus berjalan dengan baik pada permulaan use case; contohnya mesin ATM yang tidak jalan, menampilkan pesan Selamat Datang.

- **Deskripsi.** Bagian terbesar dari use case merupakan deskripsi naratif dari urutan utama use case yang merupakan urutan yang paling umum dari interaksi antara aktor dan sistem. Deskripsi tersebut dalam bentuk input dari aktor, diikuti oleh respon pada sistem. Sistem ditandai dengan sebuah kotak hitam (black box) yang berkaitan dengan apa yang sistem lakukan dalam merespon input aktor, bukan bagaimana internal melakukannya.

- **Alternatif-alternatif.** Deskripsi naratif dari alternatif merupakan cabang dari urutan utama. Terdapat beberapa cabang alternatif dari urutan utama. Contohnya, jika rekening customer terdapat dana yang tidak sesuai, akan tampil permohonan maaf dan menolak kartu.

- **Postcondition.** Kondisi yang selalu terjadi di akhir use case, jika urutan utama telah dilakukan; contohnya dana customer telah ditarik.

- **Outstanding questions.** Pertanyaan-pertanyaan tentang use case didokumentasikan untuk didiskusikan dengan para user.