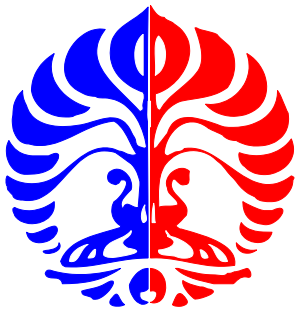
PENERAPAN *DESIGN PATTERNS* UNTUK IMPLEMENTASI ALAT BANTU *REQUIREMENTS MANAGEMENT* BERBASIS KOMPUTER GENGGAM

Disusun Sebagai Laporan Tugas Akhir

A Sasmito Adibowo
1299000029



FAKULTAS ILMU KOMPUTER
UNIVERSITAS INDONESIA
AGUSTUS 2003

LEMBAR PERSETUJUAN

Judul	:	Penerapan *Design Patterns* untuk Implementasi Alat Bantu *Requirements Management* berbasis Komputer Genggam
Nama	:	A Sasmito Adibowo
NPM	:	1299000029
Tanggal	:	Juli 2003

Tugas akhir ini telah diperiksa dan disetujui oleh:

Pembimbing Tugas Akhir

Eko K. Budiardjo, Msc.

ABSTRAK

Proyek tugas akhir ini membuat sebuah alat bantu untuk *requirements management* yang dapat digunakan di komputer genggam. Perangkat lunak yang dinamakan Rambutan – *Requirements Management Tool for Busy System Analysts* ini terdiri dari dua aplikasi, satu untuk digunakan di komputer *desktop* sedangkan yang lain untuk di komputer *handheld*. Program di *desktop* diimplementasikan dengan Java 2 Standard Edition sebagai aplikasi GUI Swing. Sedangkan program untuk komputer genggam diterapkan menggunakan SuperWaba sehingga dapat dijalankan di komputer-komputer *handheld* berbasis Palm OS, Windows CE, dan PocketPC. Perangkat lunak ini diterapkan dengan *design patterns* untuk meningkatkan kualitas arsitektur program sehingga dapat memfasilitasi pengembangan selanjutnya.

Landasan konsep *requirements management* yang dipergunakan bertolak dari *Requirements Markup Language* (RQML), sebuah *format* XML untuk menyimpan dokumen *requirements*, yang juga menjadi *format* penyimpanan data yang dikelola oleh program. Bentuk *requirements* yang menjadi cakupan RQML adalah yang ditulis secara tekstual dalam bahasa alami. Sedangkan rancangan RQML sendiri didasarkan pada sekelompok *best practices* di dalam *requirements management* sehingga RQML tidak tergantung pada metodologi pengembangan perangkat lunak tertentu.

Laporan tugas akhir ini terdiri dari dua bagian utama yaitu landasan teori yang diikuti dengan bab-bab yang terkait pada pembuatan program. Landasan teori memberikan perkenalan terhadap berbagai teori yang digunakan dalam pembuatan program; meliputi *requirements engineering*, XML, *Requirements Markup Language* (RQML), dan *design patterns*. Sedangkan pembahasan program meliputi analisis dan perancangan, implementasi, serta pengujian. Tulisan ini diakhiri dengan satu bab penutup yang memberikan rangkuman, kesimpulan, dan saran untuk penelitian-penelitian selanjutnya.

xi + 143 hlm; 37 gbr; 10 tbl.
Daftar Pustaka: 69 (1977 – 2003).

KATA PENGANTAR

Dulu, sewaktu belum terlalu lama tertarik pada komputer dan *programming*, saya itu orangnya sangat *technical* sekali. Bacaan sehari-hari tidak jauh dari *programming manuals* dan dokumentasi API. Selain itu saya juga senang mencoba-coba berbagai *development tools* serta belajar macam-macam bahasa pemrograman. Bahkan saya dulu suka melihat-lihat koran bagian lowongan pekerjaan untuk memutuskan *programming language* atau *tool* apa yang akan saya coba berikutnya – dengan cara melihat lowongan-lowongan IT (*Information Technology*) serta kemampuan apa saja yang diminta untuk posisi itu.

Di bangku kuliah saya sempat kerja *programming* sambilan dan cukup beruntung untuk terlibat dalam beberapa proyek 'swasta'. Selain itu dari berbagai kuliah yang saya ambil ada beberapa tugas *programming* yang cukup menantang. Dalam berbagai kesempatan, saya cenderung untuk memprogram dalam bahasa yang *object-oriented* dan berbagai masalah teknis saya pecahkan secara *ad-hoc*.

Sewaktu semester lima saya diperkenalkan dengan *design patterns*, waktu itu saya merasa senang sekali. Karena dari dulu saya suka dengan yang namanya *programming libraries* – komponen-komponen siap pakai untuk digunakan dalam suatu program – maka saya memandang *design patterns* sebagai sebuah *higher-order library* (analogis dengan *higher-order functions*). Cuma waktu itu agak sedih juga: kenapa baru diperkenalkan pada semester yang cukup 'tua' – andaikata dikenalkan lebih awal maka banyak masalah *programming* yang saya alami dulu akan menjadi lebih mudah dipecahkan.

Pada waktu akan mengerjakan tugas akhir ini, ketertarikan akan *design patterns* saya bawa ke Bapak Eko K Budiardjo, yang dulu memperkenalkan saya kepada *design patterns* dan kemudian menjadi pembimbing tugas akhir ini. Beliau pada waktu itu memperluas wawasan saya lebih jauh ke *requirements engineering* serta menawarkan topik ini untuk dibahas dalam tugas akhir selain *design patterns*.

Perkenalan saya dengan bidang *requirements engineering* benar-benar mengubah perspektif saya tentang bidang IT. Saya yang dulunya *suka bikin-bikin* berbagai *utility* canggih namun tidak ada yang memakainya; sewaktu kerja *part-time* hanya mengikuti instruksi tambahan *requirement* dari *employer* yang datang setiap Sabtu; juga yang

cenderung mempromosikan teknologi-teknologi tertentu dengan menjelek-jelekkan yang lain. Banyak pandangan saya yang keliru jadi terkoreksi.

Ternyata *requirements engineering* itulah yang menghubungkan segi bisnis dengan IT – di mana bidang yang kedua ini umumnya didominasi oleh para *techies*. Ternyata masalahnya bukanlah bahasa C versus Java, bukanlah JSP versus PHP atau bahkan Windows versus Linux. Masalah sebenarnya ada di *requirements*. Bukankah IT ditujukan untuk memecahkan masalah di dunia nyata, masalah-masalah yang mayoritas dimiliki oleh orang-orang di luar dunia IT?

Menerawang lebih jauh, bukankah bisnis di bidang manapun yang paling penting adalah *customer satisfaction*? Kepuasan pelanggan berarti pemenuhan akan kebutuhan yang dimilikinya... Sebuah penyedia jasa IT tentulah perlu mengerti masalah yang sedang dihadapi oleh pelanggannya sehingga bisa ditemukan solusi yang tepat guna, bukanlah hanya setumpuk *hardware* dan bergiga-giga *program code* yang tidak terpakai....

Pada kesempatan ini, ijinkan saya untuk mengucapkan beberapa patah kata berikut:

- Kepada Bapak Eko K Budiardjo: terima kasih telah meluangkan waktu, pikiran, serta *bandwidth* untuk membimbing saya dalam pengerjaan tugas akhir ini.
- Kepada Bapak Dadan Hardianto: terima kasih telah memegang peran sebagai pembimbing akademis selama masa kuliah saya di Fasilkom UI.
- Terima kasih kepada orang tua saya yang telah memberi dukungan moral dan material selama masa hidup saya sampai kini.
- Kepada Guilherme C Hazan: terima kasih sudah membuat SuperWaba dan membantu menunjukkan *workaround* pada awal-awal pembuatan program.
- Kepada Minnarto Djojo: terima kasih telah memperkenalkan saya kepada dunia komputer genggam.
- Kepada Jeane Anne: terima kasih sudah mau diganggu malam-malam dengan pertanyaan-pertanyaan yang trivial – walaupun sedang stress mengurusi gusi-gusi orang ;-)
- Kepada Maya, Hansen, Lerry, dkk: terima kasih udah mau 'berbagi' Pak Eko dan 'Hari Selasa', sorry banget waktu itu kalo gua nyinggung dan rada ngeselin.

- Kepada anak-anak BNCC: sorry banget gua ga muncul-muncul lagi ke sekret dan dateng expo cuma pas workshop J2ME doang – harus ngejar deadline TA nih ;-)
- Kepada mbak-mbak sekretariat, ibu-ibu perpustakaan, bapak-bapak satpam, dan bapak-bapak janitor: terima kasih atas berbagai macam dukungannya selama kehidupan saya di Fasilkom UI.
- Kepada teman-teman lain dan tidak dapat disebutkan satu per satu: terima kasih atas bantuan dan dorongannya selama ini.

Jakarta, Agustus 2003

A Sasmito Adibowo

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

Bab 1

PENDAHULUAN

1.1 Latar Belakang

Proses *requirements engineering* bukanlah merupakan hal yang mudah. Seorang *system analyst*, *project manager*, atau siapapun yang memegang peran *project champion* harus mengumpulkan berbagai *requirement* dari para *stakeholder*, menganalisa *requirement* tersebut, mengkomunikasikasikannya dengan para programmer, serta menyelesaikan konflik yang terjadi antar berbagai *requirement* yang ada.

Seringkali *project champion* ini harus bekerja di luar kantor untuk bertemu dengan para *stakeholder*. Hal ini terutama terjadi pada kasus proyek *software development* di mana organisasi pengembang berbeda dengan organisasi yang pada akhirnya akan menggunakan perangkat lunak tersebut.

Sebagai seorang pekerja IT yang *mobile* dengan jadwal yang padat, besar kemungkinannya ia akan menggunakan sebuah PDA (*Personal Digital Assistant*) dalam mengatur jadwal pekerjaannya sehari-hari. Aplikasi-aplikasi PIM (*Personal Information Management*) yang menjadi ciri khas suatu PDA belum tentu dapat secara optimal membantu tugasnya dalam pengumpulan serta perumusan *requirement*. Alangkah baiknya apabila ada suatu perangkat lunak di dalam PDA-nya yang ditujukan khusus pengaturan berbagai requirement – yang merupakan salah satu tugas utama *project champion* ini.

1.2 Tujuan Penelitian

Dari beberapa pencarian singkat di Google <http://www.google.com> (terakhir pada tanggal 2 Juli 2003), ternyata belum ada alat bantu *requirements management* yang ditujukan untuk komputer genggam. Dua jenis *software* yang paling mendekati kegunaan ini adalah untuk *Project Management* dan *Mind-Mapping*. Fakta ini

menandakan adanya suatu *market opportunity* untuk sebuah *requirements management tool* berbasis komputer genggam[1].

Problem statement (atau lebih tepatnya, *opportunity statement*) tersebut menjadi titik tolak penelitian ini yang bertujuan untuk membuat sebuah prototipe alat bantu *requirements management* berbasis komputer genggam. Sebuah nilai tambah diberikan melalui penggunaan *design patterns* dalam implementasinya.

Alat bantu yang dibuat dinamakan Rambutan – *Requirements Management Tool for Busy System Analysts*. Rambutan terdiri dari dua *software* terpisah di mana keduaduanya berfungsi saling melengkapi. Salah satunya untuk digunakan di komputer *desktop*, sedang yang lain untuk digunakan di komputer genggam. Kedua aplikasi ini ditujukan hanya untuk digunakan oleh satu orang (*single-user system*).

1.3 Ruang Lingkup Penelitian

Ruang lingkup penelitian meliputi penerapan sebuah alat bantu *requirements management* yang berbasis RQML serta dapat digunakan di sebuah komputer genggam. Fungsi utama yang diharapkan adalah *data entry* pada saat sang *requirements engineer* berinteraksi dengan klien. Beberapa fungsi esensial namun kompleks terpaksa dikesampingkan karena keterbatasan waktu. Fungsi-fungsi ini termasuk *traceability*, *focus group*, dan *reporting*. Demikian pula dengan fungsi kolaborasi yang tidak termasuk dalam ruang lingkup sebuah *single-user system*.

Alat bantu *requirements management* ini dikembangkan atas dasar konsep-konsep RQML, sebuah *format* XML untuk menyimpan dokumen *requirements* [Gudg00]. Sebagian besar *theoretical work* yang diperlukan pada *problem domain* – yaitu *requirements engineering* – telah dilakukan oleh Gudgeirsson dalam penelitiannya pada saat membuat RQML. Penelitian ini melanjutkan pekerjaannya dengan mengimplementasikan sebuah *software* yang dibuat atas dasar RQML dan juga menyimpan datanya dalam bentuk XML.

---

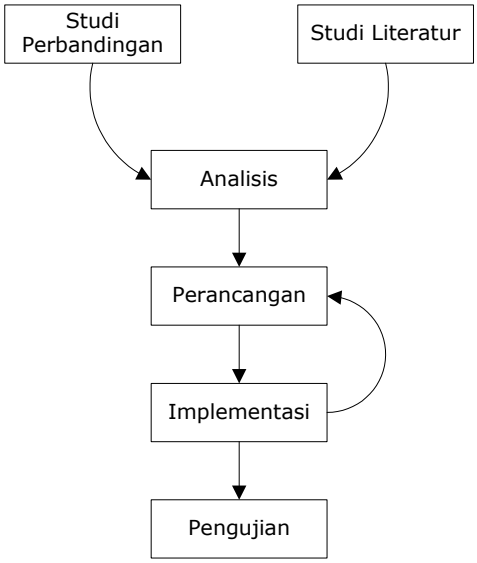
[1] Ada orang bijak yang pernah mengatakan, "*Problem* dan *opportunity* itu adalah dua sisi pada koin yang sama." – Sebuah permasalahan membuka peluang bagi yang dapat memecahkan masalah tersebut.

Design patterns yang digunakan adalah pola-pola yang dimuat di [Gamm95]. Walaupun masih banyak *pattern* lainnya, berbagai *pattern* dalam buku ini telah terbukti cukup *general purpose* dalam penerapannya di berbagai *object-oriented systems*.

Komputer genggam yang menjadi sasaran adalah yang dijual sebagai *Personal Digital Assistant* (PDA) dan berbasis PalmOS maupun PocketPC. Namun karena keterbatasan sumber daya maka pengujian aplikasi komputer genggam hanya dilakukan di PalmOS.

1.4 Metodologi Penelitian

Penelitian melalui tahap-tahap studi literatur, studi perbandingan, analisis, perancangan, implementasi dan terakhir pengujian. Dalam studi literatur, tiga referensi utama yang digunakan adalah [Gudg00], [Gamm95], dan [Leff00] di samping berbagai referensi pendamping. Studi perbandingan menggunakan program Rational RequisitePro 2002 sebagai contoh alat bantu *requirements management* yang telah matang. Analisis yang dilakukan terutama terhadap spesifikasi RQML serta bagaimana sebuah komputer genggam dapat digunakan untuk pengumpulan *requirement*. Pada tahap perancangan dibuat berbagai tampilan *mock-up* serta sebuah himpunan *requirement* untuk program yang akan dibuat. Pada tahap implementasi, program dibuat dengan semaksimal mungkin digunakan *design patterns* serta memperlakukan [Gamm95] sebagai 'buku solusi' untuk masalah-masalah pemrograman. Setelah implementasi selesai, program hasilnya diujikan dari berbagai perspektif dengan himpunan *requirement*-nya masing-masing. Tahapan studi literatur dilakukan secara parallel dengan studi perbandingan. Sedangkan tahapan-tahapan perancangan dan implementasi dilakukan secara iteratif.



Gambar 1.1 Metodologi Penelitian

Analisis dilakukan atas dasar sebuah *user story* yang menggambarkan situasi penggunaan Rambutan nantinya. Cara ini merupakan salah satu varian dari metode skenario seperti CREWS [Maid96]. Bersamaan dengan spesifikasi RQML, *user story* ini menjadi dasar untuk merumuskan *requirement* yang dibebankan kepada Rambutan.

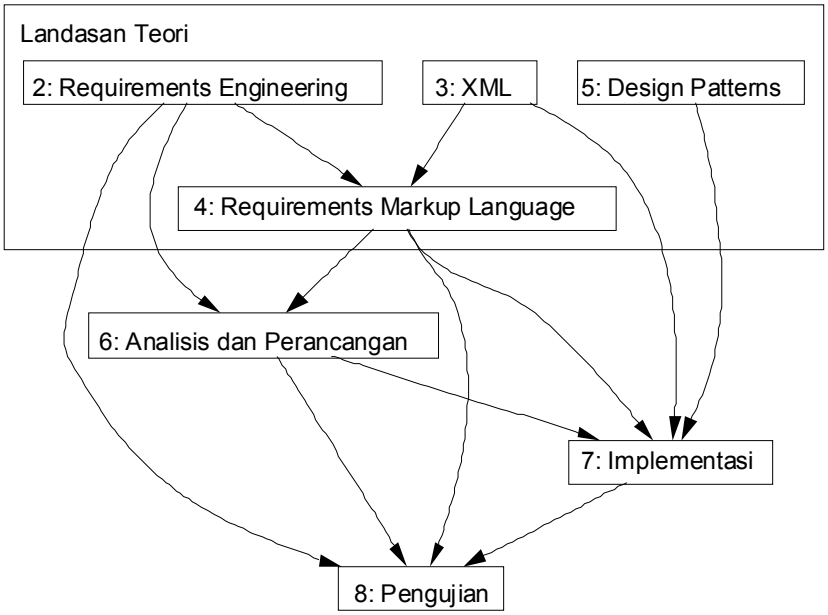
Berangkat dari *data model* RQML, perancangan Rambutan diawali dengan gambaran garis besar sistem secara deskriptif (dituliskan berbentuk prosa) serta pembuatan beberapa rancangan tampilan layar (*screenshots*) dari aplikasi *handheld*. Rancangan kasar ini kemudian diperinci seiring dengan proses implementasi secara iteratif, dengan *design patterns* [Gamm95] sebagai panduan untuk mengambil keputusan-keputusan rancangan program.

1.5 Sistematika Penulisan

Laporan tugas akhir ini terdiri dari dua bagian utama yaitu landasan teori yang diikuti dengan bab-bab yang terkait pada pembuatan program. Landasan teori memberikan perkenalan terhadap berbagai teori yang digunakan dalam pembuatan program; meliputi *requirements engineering*, XML, *Requirements Markup Language* (RQML), dan *design patterns*. Sedangkan pembahasan program meliputi analisis dan perancangan,

implementasi, serta pengujian. Tulisan ini diakhiri dengan satu bab penutup yang memberikan rangkuman, kesimpulan, dan saran untuk penelitian-penelitian selanjutnya.

Gambar 1.2 menunjukkan hubungan ketergantungan antar bab yang ada dalam laporan ini.



Gambar 1.2 Sistematika Penulisan

Bab 2

REQUIREMENTS ENGINEERING

The King's Companion

The King called his wizard, Harry, into his chambers. "Harry, I require a companion. I want you to find one for me. She must be strong and capable, yet warm and affectionate. She must be intelligent, yet loyal and accepting of my supreme authority. She must be attractive, yet find me loveable. She must be able to entertain herself, yet be ready to cheerfully accompany me anywhere at a moments notice. She must be quick to joy, slow to anger, and must never criticize me. Oh - and Harry - I need her by tonight."

Harry stared at the King. After a long pause, he started to speak. The King abruptly stood up and waved him away. "I have given you my requirements. You are a powerful wizard, I know you can find the perfect companion for me - now go."

Harry slowly nodded and retired to his tower.

Several hours later, while eating his supper, the King allowed himself to dream about the wonderful woman the wizard would bring to him. She would be sweet and warm, yet strong and intelligent. Surely, she would be the queen to match his magnificent kingliness. He chortled as he thought about the other kings with their chatty, stupid queens. They will most certainly envy him at next month's Monarch Golf Tournament and Banquet.

Imagine the King's surprise when Harry arrived at the door with a beautiful golden retriever. Imagine Harry's frustration as he was led to the dungeon. Harry's yells can still be heard echoing in the vast corridors of the castle.... "But..... you said you wanted...."

Diangkat dari [Saut03], kisah “The King’s Companion” ini menceritakan salah satu masalah yang melatarbelakangi munculnya *requirements engineering* sebagai bidang ilmu. Klien (“The King”) memberikan berbagai *requirement* dengan *deadline* yang ketat kepada *vendor* (“Harry”) tanpa memberikan kesempatan untuk memperjelas apa yang

diinginkannya . Untung saja *vendor* dapat memberikan suatu produk ("golden retriever") yang sesuai dengan semua *requirement* yang diberikan – namun sayangnya 'solusi' ini sama sekali bukan yang diinginkan oleh klien. Cerita ini berakhir dengan tragis di mana kesalahan ditimpakan kepada *vendor*.

2.1 Definisi *Requirements*

Sebuah *requirement* dalam konteks rekayasa perangkat lunak adalah sebuah kemampuan yang harus dimiliki dari suatu *software* [Dorf90]. Kemampuan ini dapat ditujukan untuk memecahkan suatu permasalahan ataupun diperlukan untuk memenuhi ketentuan-ketentuan tertentu (seperti standar tertentu, keputusan manajemen, ataupun alasan-alasan politis).

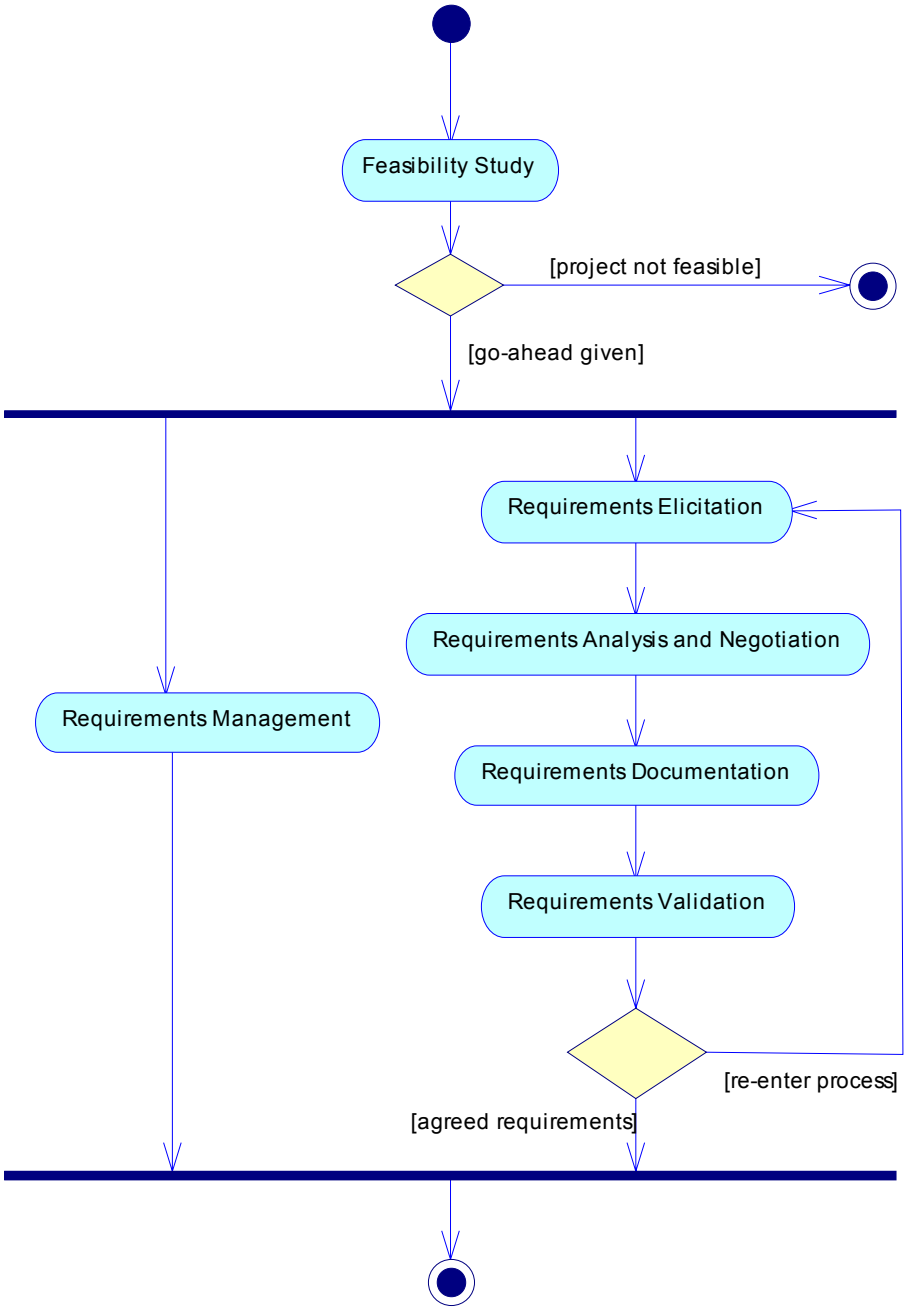
Kumpulan dari berbagai *requirement* digunakan dalam berbagai aspek dalam pengembangan sebuah sistem. Dalam tahap perancangan, *requirement* digunakan untuk menentukan berbagai fitur yang akan ada di dalam sistem. Pada penghujung sebuah *development effort*, himpunan *requirement* ini digunakan untuk melakukan *validation & verification* untuk memastikan perangkat lunak yang telah dibuat memang sesuai dengan yang diinginkan. Bahkan selagi pengembangan berjalan, himpunan *requirement* ini terus dimodifikasi untuk menyesuaikannya dengan berbagai kebutuhan para *stakeholder* serta tenggat waktu dan dana yang tersedia.

Secara luas, *software systems requirements engineering* (RE) adalah proses untuk menemukan suatu himpunan *requirement* yang tepat sehingga suatu perangkat lunak dapat memenuhi kegunaannya. Proses ini dilakukan dengan cara mengenali para *stakeholder* serta kebutuhan mereka serta mendokumentasikannya di dalam bentuk yang dapat digunakan untuk analisa, komunikasi, dan implementasi yang mengikutinya [Nuse00].

Zave [Zave97] memberikan salah satu definisi yang paling jelas dari RE:

> *Requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families.*

2.2 Proses RE



Gambar 2.1 Proses RE.

Terdapat lima aktivitas utama di dalam proses *requirements engineering* [Koto98]:

- *Requirements elicitation*
- *Requirements analysis and negotiation*
- *Requirements documentation*
- *Requirements validation*

- *Requirements management and evolution*

Sebuah *activity diagram* UML yang menunjukkan hubungan yang umum terjadi antara aktivitas-aktivitas tersebut tertera pada **Gambar 2.1**. Keterangan singkat mengenai masing-masing aktivitas akan diberikan selanjutnya. Rincian dari masing-masing tahap ini dapat ditemukan di [Koto98], [Wieg99] dan berbagai buku teks lainnya mengenai *requirements engineering*.

2.2.1 *Requirements Elicitation*

Pada tahap ini dikumpulkan berbagai *requirement* dari para *stakeholder* [Pres01]. Seorang pelanggan mempunyai masalah yang dapat ditangani oleh solusi berbasis komputer. Tantangan ini ditanggapi oleh seorang pengembang. Di sinilah komunikasi dimulai antara pelanggan, pengembang, dan calon pengguna dari sistem yang akan dibuat.

Namun istilah *elicitation* agak diperdebatkan. Ada yang menganalogikannya dengan seperti yang dilakukan oleh para arkeolog ketika mengumpulkan runtuhan-runtuhan di situs purbakala [Leff00]. Ada yang memberikan istilah *requirements capture* karena dilakukan terutama dengan mengumpulkan fakta-fakta [Benn00]. Bahkan [Gudg00] menyatakan bahwa *requirement* sebenarnya dibuat ketimbang didapatkan (*elicitated*). Walau yang terakhir ini nampaknya "lain sendiri", argumen ini dapat diterima untuk pengembangan *software* yang sama sekali baru maupun untuk *software-software* permainan (*games*) yang terkadang permasalahan yang akan dipecahkan oleh *game* tersebut cenderung tidak berhubungan dengan solusinya ataupun sebenarnya masalah yang ada berasal dari bagian *marketing*[2].

Sejalan dengan proses RE secara keseluruhan, tujuan dari *requirements elicitation* adalah [Gudg00]:

- Untuk mengetahui masalah apa saja yang perlu dipecahkan dan mengenali perbatasan-perbatasan sistem (*system boundaries*).
- Untuk mengenali siapa saja para *stakeholder*.

[2] Seperti misalnya masalah "untuk merebut pasar pria usia 14-30 tahun." Bahkan menurut [Leff00], masalah yang hendak dipecahkan untuk permainan seperti SimCity atau Myst adalah "terlalu banyak waktu luang yang dimiliki seseorang."

- Untuk mengenali tujuan dari sistem; yaitu sasaran-sararan yang harus dicapainya.

Dalam [Nuse00] disebutkan beberapa jenis teknik pengumpulan *requirement*:

- **Traditional techniques** merupakan berbagai cara pengumpulan data. Cara-cara ini termasuk kuesioner, survey, wawancara, serta analisis dari berbagai dokumentasi yang ada seperti struktur organisasi, petunjuk pelaksanaan (*juklak*) serta manual-manual dari sistem yang sudah ada.
- **Group elicitation techniques** bertujuan untuk mengembangkan dan mendapatkan persetujuan *stakeholder*, sementara memanfaatkan dinamika kelompok untuk memperoleh pengertian yang lebih mendalam. Cara-cara ini termasuk *brainstorming* dan *focus group*, juga berbagai *workshop* RAD/JAD (*workshop* untuk membangun sebuah konsensus dengan menggunakan seorang fasilitator yang netral).
- **Prototyping techniques** membuat suatu implementasi parsial dari software yang akan dibangun untuk membantu para pengembang, pengguna, serta pelanggan untuk lebih mengerti berbagai *requirement* sistem [Leff00]. Digunakan untuk mendapatkan umpan-balik yang cepat dari para *stakeholder* [Davi92], teknik ini juga dapat digabungkan dengan berbagai teknik yang lain, seperti misalnya digunakan di dalam sebuah acara *group elicitation* ataupun sebagai basis dari sebuah kuesioner.
- **Model-driven techniques** menempatkan suatu model khusus dari jenis informasi yang akan dikumpulkan untuk digunakan sebagai pedoman proses *elicitation*. Termasuk di antaranya adalah *goal based methods* seperti KAOS [Lams98] dan I* [Chun00] dan juga cara-cara berbasis skenario seperti CREWS [Maid96].
- **Cognitive techniques** termasuk serangkaian cara yang semulanya dikembangkan untuk *knowledge acquisistion* untuk digunakan di *knowledge-based systems* [Shaw96]. Teknik-teknik ini termasuk *protocol analysis* (di mana seorang ahli melakukan sebuah tugas sembari mengutarakan pikiran-pikirannya), *laddering* (menggunakan berbagai pemeriksaan untuk mendapatkan struktur dan isi dari pengetahuan *stakeholder*), *card sorting* (meminta para *stakeholder* untuk menysun kartu-kartu secara berkelompok, di mana setiap kartu tertera nama

sebuah *domain entity*), dan *repertory grids* (membuat sebuah *attribute matrix* for entities di mana para *stakeholder* diminta untuk mengisi matriks tersebut).

- **Contextual techniques** muncul pada tahun 1990-an sebagai sebuah pilihan di luar *traditional* maupun *cognitive techniques* [Gogu94]. Termasuk di antaranya penggunaan teknik etnografis seperti pengamatan terhadap para peserta. Juga termasuk *ethnomethodogy* dan analisis percakapan, yang keduanya menggunakan analisis terinci untuk mengenali pola-pola dalam percakapan dan interaksi [Vill99].

Dalam aktivitas *requirements elicitation*, ada baiknya untuk mengkategorikan berbagai *requirement* yang ditemukan. Suatu *requirement* dapat diklasifikasi sebagai *functional requirement*, *non-functional requirement*, maupun *constraints* [Grad92]. Sedangkan[Koto98] mengatakan bahwa suatu *requirement* dapat diklasifikasikan menjadi *very general requirements*, *functional requirements*, *implementation requirements*, *performance requirements*, dan *usability requirements*.

Namun nyatanya klasifikasi (atau cara-cara pengkategorian lainnya) *requirement* ini tidak mutlak diperlukan; klasifikasi *requirement* ditujukan terutama untuk menuntun proses *elicitation*. Hal ini perlu diwaspadai karena gara-gara para anggota tim tidak dapat setuju akan klasifikasi dari sekumpulan *requirement*, maka *development effort* dari sebuah perusahaan Fortune 500 mengalami stagnasi [Leff00]. Terjebaknya mereka di dalam masalah semantik ini merupakan salah satu contoh dari *analysis paralysis* [Whit99].

2.2.2 *Requirements modeling and analysis*

Sebuah model adalah perwakilan dari benda lain yang mempunyai rincian yang cukup untuk membantu penyelesaian tugas-tugas tertentu [Benn00]. *Data modeling* bertujuan untuk mendapatkan pengertian dari pemrosesan serta pengaturan informasi. *Behavioral modeling* memodelkan berbagai perilaku dari para *stakeholder* serta berbagai sistem lain yang berhubungan dengannya. *Domain modeling* menyediakan suatu bentuk abstrak dari dunia tempat beroperasinya sistem yang akan dibuat.

Model-model yang dihasilkan dalam tahap ini ditujukan untuk analisa terhadap berbagai *requirement* yang ada. Para *stakeholder* berunding untuk mendapatkan

suatu himpunan *requirement* akhir yang akan digunakan untuk tahap pengembangan selanjutnya. Menurut [Koto98] setelah selesainya tahap idealnya ini akan berlaku:

- Berbagai *requirement* dari masing-masing *stakeholder* tidak bertentangan.
- Informasi di dalam semua *requirement* harus lengkap.
- Berbagai *requirement* yang ada harus selaras dengan anggaran yang dimiliki.

Walaupun dengan adanya batasan-batasan tersebut, seluruh *requirement* sebaiknya mudah diubah ataupun disesuaikan.

2.2.3 *Requirements documentation*

Tahap ini adalah penulisan dari *requirements document*, yang terkadang disebut dokumen *Software Requirements Specification* (SRS). Menurut [Hen80], dokumen ini sebaiknya:

- Hanya menetapkan perilaku sistem sebagaimana terlihat dari luar
- Menetapkan batasan-batasan (*constraints*) yang diberikan kepada implementasinya.
- Mudah diubah.
- Berguna sebagai alat referensi untuk pemeliharaan sistem.
- Memuat gambaran akan siklus kehidupan sistem di masa yang akan datang.

Untuk meningkatkan *readability*, beberapa standar dokumentasi SRS telah dikembangkan. Namun menurut [Kov99], serangkaian standar dan *template* apabila berdiri sendiri tidak dapat digunakan sebagai cara yang mandraguna untuk memberi struktur bagi sekumpulan *requirement*; tetapi struktur yang digunakan haruslah dikembangkan sendiri-sendiri tergantung dari masalah yang sedang ditangani. Masalah standarisasi notasi dan pendokumentasian *requirement* membuat pendekatan sistematis terhadap RE menjadi sulit. [McDe94] memberikan sebuah daftar praktis ciri-ciri yang dinginkan pada sebuah *requirements document*:

- *Unambigous*. Idealnya, hanya ada satu interpretasi terhadap sebuah *requirements document*.
- *Complete*. Semua aspek yang bersangkutan haruslah dijelaskan secara lengkap di dalam *requirements document*.

- *Consistent*. Tidak ada pernyataan yang bertentangan dalam *requirements document*.
- *Verifiable*. Setelah sebuah sistem diimplementasikan, sebaiknya dapat dipastikan bahwa sistem tersebut memenuhi *requirement* awal.
- *Validatable*. Suatu *requirement* sebaiknya dapat diperiksa oleh pelanggan untuk memastikan bahwa *requirement* tersebut memang memenuhi kebutuhannya.
- *Modifiable*. Perubahan sebaiknya mudah dilakukan dan efek dari perubahan ini terhadap bagian-bagian lain sebaiknya minimal.
- *Understandable*. Semua *stakeholder* sebaiknya dapat mengerti *requirement* seperti ditetapkan di dalam dokumen.
- *Testable*. Semua requirement sebaiknya cukup kuantitatif untuk digunakan sebagai titik tolak pengujian sistem.
- *Traceable*. Harus dimungkinkan adanya pengacuan (*reference*) antar berbagai bagian di dokumen *requirement* ataupun ke bagian-bagian lain dari proses pembuatan perangkat lunak.

2.2.4 *Requirements validation*

Dalam tahap ini, dokumen dari tahap sebelumnya diperiksa agar memenuhi kriteria-kriteria sebagai berikut [Koto98]:

- Lengkap.
- Konsisten.
- Tunduk pada keputusan-keputusan yang diambil pada tahap *requirements analysis*.

Apabila ada *requirement* yang tidak memenuhi kriteria-kriteria tersebut, mungkin ada baiknya bagi proses RE untuk kembali ke tahap-tahap sebelumnya. Beberapa contoh masalah *requirement* yang terungkap pada tahap validasi antara lain [Koto98]:

- Kurang/tidak cocok dengan bakuan-bakuan kualitas.
- Kata-kata yang digunakan kurang baik sehingga *requirement* menjadi ambigu.
- Berbagai kesalahan yang terdapat pada model-model baik – model sistem ataupun model permasalahan yang hendak dipecahkan.
- Pertentangan antar *requirement* yang tidak ditemukan pada tahap analisis.

2.2.5 *Requirements management and evolution*

Sebuah *software* yang sukses pasti akan berevolusi mengikuti perubahan lingkungannya. Sebaliknya, *software* yang sudah tidak diperbaharui berarti telah ditinggalkan oleh para penggunanya. Dalam perjalanan evolusi sebuah *software*, *requirement* akan *software* tersebut akan bertambah, berubah, atau terkadang berkurang. Agar perubahan ini terkendali, perlu adanya aktivitas *requirements management* (RM).

Walaupun berbagai aktivitas RE dalam **Gambar 2.1** digambarkan secara terpisah, namun sebenarnya pemisahan antara aktivitas-aktivitas ini tidaklah terlalu kentara. Proses *elicitation*, *analysis*, *documentation*, dan *validation* dalam RE berjalan secara berkesinambungan tanpa adanya batasan-batasan yang definitif. Fungsi lain dari *requirements management* adalah untuk memastikan agar berbagai aktivitas dalam proses RE ini berjalan dengan baik – agar iterasi demi iterasi dalam RE dilakukan secara terkendali dan (diharapkan) menuju suatu kemajuan. Maka dari itu *requirements management* digambarkan secara paralel dengan pengulangan dari aktivitas-aktivitas RE lainnya.

Dalam aktivitasnya mengendalikan perubahan *requirement* maupun mengendalikan proses RE itu sendiri, pada *requirements management* juga dilakukan penyusunan berbagai informasi *traceability* – yaitu keterhubungan antara berbagai artifak di dalam proses RE (termasuk perubahan *requirement*). *Traceability* itu sendiri merupakan konsep yang penting di dalam RE yang akan selanjutnya dibahas berikutnya.

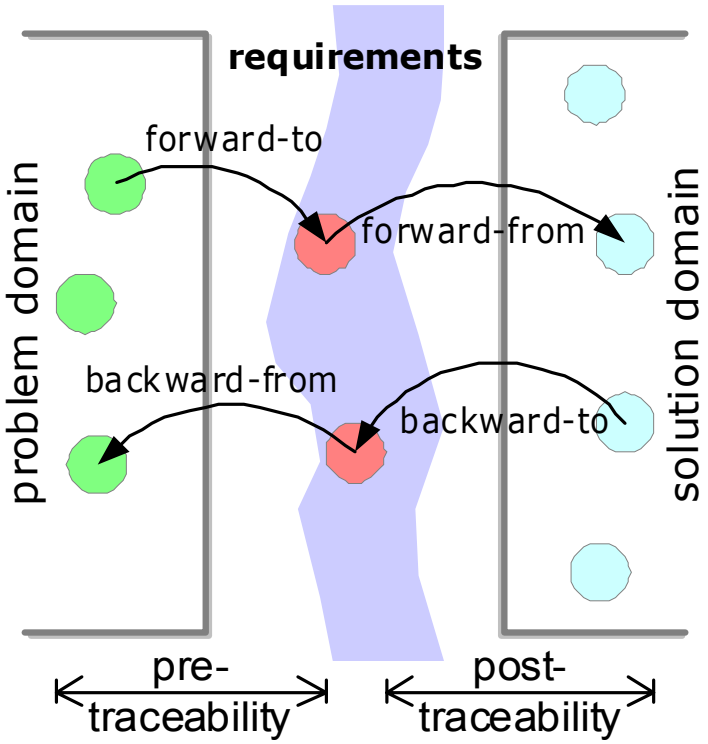
2.3 *Traceability*

Traceability adalah cara untuk membentuk hubungan-hubungan antara manusia dan sistem dengan model-model di mana mereka dikelola [Jark98]. Kutipan berikut memberikah sebuah garis besar mengenai pentingnya traceability di dalam rancangan sistem secara keseluruhan:

> *"Requirement tracing is emerging as an effective bridge that aligns system evolution with changing stakeholder needs. It also helps uncover unexpected problems and innovative opportunities, and lays the groundwork for corporate knowledge management." [Jark98]*

Secara tradisional ada empat jenis informasi *traceability* ([Koto98], [Jark98]):

- ***Forward-from traceability*** menghubungkan *requirement* ke rancangan komponen-komponen dan penerapannya serta memberikan tanggungjawab kepada komponen-komponen sistem tersebut untuk pencapaian *requirement-requirement* tertentu.
- ***Backward-to traceability*** menghubungkan rancangan serta komponen-komponen penerapannya kembali kepada *requirement* untuk memungkinkan pengujian sistem terhadap pemenuhan *requirement* tersebut.
- ***Forward-to traceability*** menghubungkan informasi *domain* beserta informasi lain yang bersangkutan yang mendahului rancangan sistem ke berbagai *requirement*. *Traceability* ini memungkinkan penaksiran pengaruh perubahan kebutuhan *stakeholder* maupun perubahan berbagai asumsi.
- ***Backward-from traceability*** menghubungkan *requirement* ke sumber-sumbernya pada dokumen-dokumen ataupun orang-orang lain. *Traceability* ini memungkinkan diketahuinya struktur kontribusi yang mendasari berbagai *requirement* yang ada – yang sangat penting di dalam validasi *requirement*.



Gambar 2.2 Empat jenis informasi *traceability* tradisional

Dua jenis *trace* yang pertama sering disebut *post-traceability* sedangkan dua yang terakhir disebut *pre-traceability*. Seperti ditunjukkan di [Jark98], saat ini *post-traceability* cenderung lebih dimengerti ketimbang *pre-traceability*, walaupun *pre-traceability* mempunyai kemungkinan untuk menyediakan hubungan antara bisnis dengan IT[3] yang saat ini sangat diperlukan.
Walaupun empat jenis *trace* tersebut sangat populer di dalam berbagai literatur, mereka bukanlah satu-satunya jenis *traceability* yang ada. Bahkan pada [DP98], nampak bahwa untuk setiap proyek mungkin diperlukan jenis-jenis *trace*-nya sendiri sesuai dengan kebutuhan proyek tersebut; kemampuan ini disediakan di banyak alat bantu RE (hal ini akan ditelusuri lebih lanjut pada bab ini juga).

2.4 *Requirements Patterns*

Dalam [Gamm95] telah ditunjukkan bahwa *pattern* dapat membawa pengaruh positif dalam bidang *software engineering*. Walau buku ini membahas *design pattern* – yang notabene adalah artifak dari *solution space*, semangat yang sama juga dapat diterapkan pada *problem space*. Dengan mengumpulkan serta mengkategorisasikan berbagai pengalaman kolektif dalam bidang RE, tentulah akan meningkatkan kualitas RE sebagai ilmu praktis.

Sebuah *pattern* adalah pemecahan yang telah ditemukan untuk suatu masalah yang umum. Praktis diresmikan oleh Gamma dkk. dalam [Gamm95], suatu *pattern* digambarkan sebagai kesatuan dari niatan, alias, motivasi, penggunaan, akibat, serta contoh penerapannya (lihat subbab 5.1 di halaman 55).

Berbagai keuntungan yang bisa didapatkan dengan mempelajari *pattern* menjadi pendorong riset ke arah *RE patterns*. Sedikit berbeda dengan *design patterns*, tujuan dari setiap *RE pattern* adalah untuk mengurangi ambiguitas yang dapat terjadi serta pada saat yang bersamaan memberikan kebebasan terhadap pemenuhan *requirement* tersebut [Cree99].

Memang studi *RE patterns* masih ketinggalan ketimbang yang telah dilakukan di dalam *solution space*. Walaupun demikian, [Cree99] telah memberikan tiga *pattern* sebagai satu permulaan bagi sebuah katalog yang dapat menandingi [Gamm95]. Ketiga

[3] IT: *Information Technology*

pattern ini telah digunakan dengan sukses di dalam empat SRS yang berbeda di mana masing-masing menangani *problem domain* yang berbeda pula:

- *Specify (Locate)* – menggambarkan bagaimana seorang (sebuah) aktor dapat menemukan suatu obyek di dalam aplikasi.
- *Present (Display)* – menggambarkan data yang harus ditampilkan oleh aplikasi.
- *Prioritize (Sort)* – untuk mengkomunikasikan bagaimana suatu aspek aplikasi didahulukan dari yang lain.

Ketiga pattern di atas terutama difokuskan untuk pembuatan berbagai *requirement statement*. Tetapi karena RE juga meliputi *requirements management*, maka [Ambl00] memberikan dua *pattern* berikut untuk mengatasi berbagai tantangan politis di dalam RE:

- *Requirement As [a] Shield* – ketika seseorang mengkritik pekerjaan yang dilakukan, untuk menunjukkan bahwa kritik yang dilontarkan tidak berhubungan dengan masalah yang sedang ditangani.
- *Requirement As [a] Stick* – untuk meniadakan berbagai pendekatan atau strategi yang diusulkan namun tidak menguntungkan bagi organisasi; dengan mengevaluasi pendekatan-pendekatan dan strategi ini terhadap *requirement* yang telah diterima, kelemahan-kelemahannya akan cepat nampak.

2.5 Alat Bantu RE

Tipe alat bantu RE yang menjadi fokus proyek ini adalah alat bantu untuk *requirements management*. Alat bantu ini berguna untuk mengumpulkan *system requirements* di dalam sebuah tempat penyimpanan serta menyediakan berbagai fasilitas untuk mendapatkan informasi mengenai berbagai *requirement* tersebut. Menurut [Koto98], sebuah alat bantu *requirements management* umumnya memiliki fasilitas-fasilitas berikut:

- Sebuah *requirements browser* sehingga para pembaca dapat mengambil suatu *requirement* yang spesifik dari *database* ataupun *repository*.
- Sebuah *query system* sehingga para pengguna dapat mengambil berbagai *requirement* yang berhubungan.
- Sebuah *requirements converter* dan *processor linker* yang dapat mengubah berbagai *requirement* di dalam sebuah dokumen pengolah kata ke dalam bentuk

basis data *requirement* dan dapat memelihara hubungan antara *database* dengan perwakilan bahasa alami dari *requirement* tersebut.

- Sebuah *change control system* yang dapat memelihara informasi mengenai perubahan-perubahan *requirement* dan hubungan-hubungan dengan berbagai *requirement* yang terpengaruhi oleh perubahan tersebut.

2.5.1 Arsitektur Alat Bantu

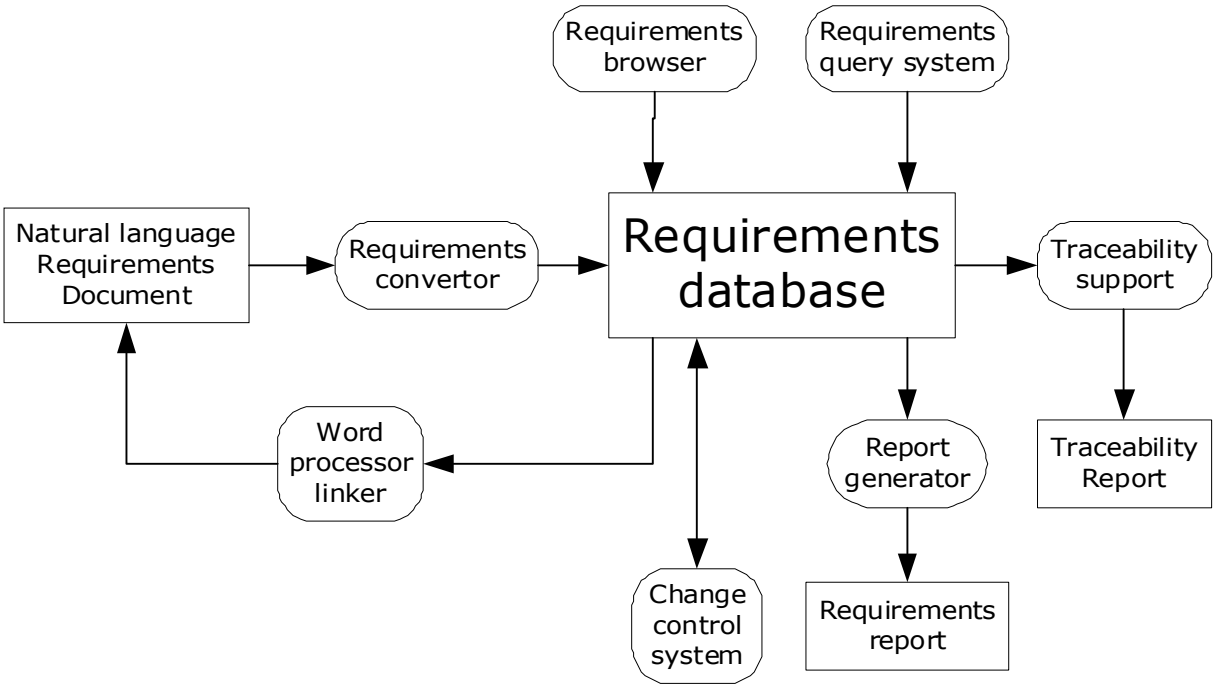
Secara tradisional, alat bantu RE menyimpan berbagai *requirement* di dalam bentuk *database* atau dokumen. Menurut [Wieg99], pendekatan berbasis dokumen untuk menyimpan *requirement* mempunyai beberapa keterbatasan:

- Sulit untuk menjaga agar semua dokumen tetap tersinkronisasi dan terkini.
- Mengkomunikasikan perubahan-perubahan kepada semua anggota tim yang terpengaruh merupakan sebuah proses manual.
- Sulit untuk membuat hubungan-hubungan antara *functional requirement* dengan berbagai *use case*, rancangan, *kode program*, pengujian, dan berbagai tugas untuk menghasilkan artifak-artifak tersebut.
- Tidak praktis untuk melacak status dari setiap *requirement*.

Selain masalah-masalah di atas, masalah umum yang terjadi dengan bahasa alami sudah umum diketahui. Sebuah pemecahan kepada masalah-masalah ini adalah dengan menyimpan berbagai *requirement* di dalam sebuah basis data *multi-user* serta mengendalikan akses kepadanya dengan alat bantu *requirement management*. Namun juga ada beberapa batasan dengan pendekatan ini [Gudg00]:

- *Format file database* seringkali *proprietary* dan tidak dapat dibaca oleh manusia.
- *Format file database* tidak komplementer; apabila sebuah dokumen *requirement* disimpan di dalam suatu database menggunakan satu alat bantu maka tidaklah mungkin untuk menambah informasi dengan suatu alat bantu lain yang berbeda. Hal ini hanya mungkin dengan dokumen, di mana lebih banyak informasi dapat diberikan dengan menambahkan lebih banyak teks.
- Karena sebenarnya tidak ada dokumen *requirement*, maka proses *requirement* yang ada sangat tergantung dengan alat bantu yang digunakan.

Banyak alat bantu RE yang menggabungkan kedua pendekatan, baik yang berbasis dokumen maupun dengan *database*. Dengan pendekatan itu, sebuah dokumen



Gambar 2.3 Sebuah *requirement management system* tradisional, di mana dokumen-dokumen dalam bahasa alami, laporan-laporan, dan basis data requirement adalah kesatuan-kesatuan yang terpisah [Koto98]

requirement dimanipulasi dengan sebuah alat bantu, tetapi sebuah basis data dipelihara di sampingnya untuk menyimpan berbagai attribut, tabel *traceability*, dan sebagainya. Sebuah gambaran luas mengenai alat seperti itu ada pada **Gambar 2.3**.

Sedangkan INCOSE [Jone95] memberikan empat jenis arsitektur alat bantu *requirements management*:

- **Arsitektur A – document based**. Arsitektur ini menggunakan komponen utama sebuah program *spreadsheet* (seperti QuattroPro™, Lotus 1-2-3™, Excel™) atau *word processor* (seperti WordPerfect™, AmiPro™, atau MS Word™) dengan diberi fungsi-fungsi tambahan untuk melakukan RM dengan menggunakan fasilitas pemrograman yang diberikan oleh program paket tersebut (umumnya bahasa *macro*). Teks di dalam dokumen diberi tanda-tanda khusus yang menandakan teks tersebut sebagai sebuah *requirement*.
- **Arsitektur B – hybrid**. Seperti pada Arsitektur A namun dengan tambahan validasi automatis dari berbagai *requirement* yang terdapat di dalam dokumen-dokumen. Alat-alat ini dapat menghasilkan sebuah daftar yang memuat berbagai *requirement* yang terlalaikan ataupun tidak perlu.

- **Arsitektur C – special-purpose-database-based**. Alat-alat RM yang termasuk di dalam kategori ini menyimpan seluruh teks *requirement* di dalam sebuah basis data, namun program *database* yang digunakan adalah bagian dari program RM itu sendiri.
- **Arsitektur D – general-purpose-database-based**. Alat bantu yang termasuk kategori ini menyimpan teks requirement menggunakan database yang terpisah. Program DBMS yang digunakan adalah suatu program – umumnya komersial – tersendiri.

2.5.2 Alat bantu komersial

Tabel berikut ini memuat beberapa alat bantu RM yang banyak digunakan, sebagaimana dituliskan oleh Wiegers [Wieg00]. Dalam tabel berikut juga diberikan kelas arsitektur alat bantu RM menurut klasifikasi INCOSE [Jone95].

Tabel 2.1 Beberapa alat bantu *requirements management* komersial

Alat Bantu	**Penyedia**	**Arsitektur**
Caliber-RM	Technology Builders, Inc. www.tbi.com	D (version 3.0)
DOORS	Quality Systems and Software, Inc. www.qssinc.com	C (version 2.1)
RequisitePro	Rational Software Corporation www.rational.com	B, D (version 2002)
RTM Workshop	Integrated Chipware, Inc. www.chipware.com	D (version 2.3)
Vital Link	Compliance Automation, Inc.; www.complianceautomation.com	C (version 2.0)

Ulasan mengenai perbandingan alat-alat bantu *requirements management* bisa didapatkan di [Inco02] dan [Achr03].

Bab 3

XML

Sebelum memperkenalkan XML, ada baiknya untuk membahas dasar-dasarnya di dalam ilmu komputer. XML adalah sebuah versi penyederhanaan dari *Standard Generalized Markup Language* (SGML), sebuah sistem untuk mengatur dan menandai elemen-elemen dari sebuah dokumen. SGML dikembangkan dan distandarisasi oleh International Organization for Standards (ISO) pada tahun 1986. SGML masih digunakan secara luas di dalam industri penerbitan, namun karena besar dan kerumitannya, SGML menjadi kurang ideal untuk penggunaan secara luas di era Internet. Untuk memecahkan masalah-masalah ini, World-Wide Web Consortium (W3C) [Cons] membuat XML sebagai sebuah subset dari SGML – walaupun kini XML mempunyai fitur-fitur yang tidak didukung SGML sehingga XML tidak lagi merupakan sebuah subset.

Seperti dijelaskan di [Marc99], XML awalnya dirancang sebagai HTML yang lebih baik. Sejalan dengan perkembangan popularitasnya, kegunaan XML sekarang menjadi lebih luas ketimbang tujuan awalnya. Nampak bahwa XML telah menjadi sebuah standar untuk pertukaran data, di mana penggunaannya tidak hanya untuk halaman web saja, namun juga untuk data yang lain. Melalui berbagai inisiatif seperti BizTalk, RosettaNET dan ebXML – yang bertujuan untuk memajukan penggunaan XML untuk memungkinkan perniagaan elektronis dan penggabungan aplikasi – serta standar seperti *Simple Object Access Protocol* (SOAP) [Cons00a] – yang merupakan protokol untuk pertukaran informasi di sebuah lingkungan terdesentralisir dan terdisitributif – perkembangan XML telah melampaui kegunaan awalnya. Pada saat standar integrasi dan pertukaran data seperti IDEF0 [Fede] dan OpenDoc [IBM] telah gagal, atau setidaknya tidak dapat mencapai potensinya secara penuh, XML akan sangat mungkin menjadi sukses.

Ciri kunci dari XML adalah sifat keterbukaannya. Pada [Oxfo90], *open system* didefinisikan sebagai berikut:

> *Any system in which the components conform to non-proprietary standars rather then to the standards of specific supplier of hardware or software.*

Karena standar XML serta berbagai standar pendukungnya didefinisikan oleh sebuah konsorsium (W3C) ketimbang satu *vendor* tertentu, maka XML memenuhi kriteria *open system*.

3.1 Markup

Markup adalah nama yang diberikan pada keterangan yang menjelaskan teks:

Igor ← Name
Asisten Laboratorium ← Job

XML adalah sebuah *markup language* yang menetapkan suatu cara untuk menulis *markup*. Pada XML markup di atas mungkin berbentuk seperti ini:

```
<name>Igor</name>
<job>Asisten Laboratorium</job>
```

XML juga *extensible* karena memungkinkan Anda untuk menambah jenis-jenis markup Anda sendiri. Ini tercermin dari nama XML, e**X**tensible **M**arkup **L**anguage. Ringkasnya, XML adalah sebuah *meta-language* yang memungkinkan Anda untuk membuat dan membentuk sebuah *markup* untuk dokumen-dokumen Anda sendiri. XML dikembangkan oleh W3C [Grah99], awalnya untuk menanggulangi keterbatasan-keterbatasan HTML. Akan tetapi kegunaannya telah berkembang melewati maksud awalnya. Sebuah hasil sampingan pada rancangan XML – seperti disebutkan di [Ecks99] – adalah bahwa *XML memungkinkan pemisahan isi dari penampilan*.

Ulasan XML ini dipusatkan untuk memberi perkenalan sangat singkat pada dasar-dasar XML dan terus untuk menjelaskan lebih dalam hasil-hasil XML yang secara istimewa mempunyai sangkut paut dengan proyek ini.

3.2 Terminologi XML

Sebuah dokumen XML terdiri dari satu atau lebih *elemen* (unsur). Sebuah elemen ditandai dengan bentuk sebagai berikut:

```
<contoh>
Ini adalah data menurut susunan unsur contoh.
</contoh>
```

Elemen ini terdiri dari dua *tag* (etiket), sebuah *tag* pembuka yang melingkupi unsur di antara sebuah tanda kurang-dari (<) dan tanda lebih-dari (>) dan sebuah *tag*

penutup yang serupa kecuali untuk garis-miring-depan (/) yang nampak sebelum nama elemen. Teks di antara *tag* pembuka dan penutup dianggap sebagai bagian dari elemen tersebut. Tidak seperti HTML, semua *tag* pembuka harus mempunyai tag penutup yang bersesuaian, kecuali untuk elemen kosong. Suatu elemen dapat disarangkan (*nested*) di dalam unsur lain, seperti berikut:

```
<a><b>Diperbolehkan</b></a>
```

namun penyarangan tidak dapat saling meliputi, seperti berikut:

```
<a><b>Tidak diperbolehkan</a></b>
```

Selanjutnya, setiap dokumen XML harus mempunyai hanya satu unsur akar (*root element*). Hal ini berarti dokumen XML dapat diwakili oleh struktur data pohon.

Unsur dapat mempunyai *atribut* (perlengkapan), seperti:

```
<name age="45">Igor</name>
```

"usia" adalah atribut dari unsur "nama" dan nilainya adalah 45, yang diberikan di dalam tanda kutip.

Selain elemen biasa, XML juga mendukung *elemen kosong* di mana tidak ada teks di antara tag pembuka dan penutup. Maka dari itu kedua etiket dapat digabungkan. Kedua elemen berikut ini adalah serupa:

```
<kosong></kosong>
```

```
<kosong/>
```

3.3 Konsep XML Secara Luas

Walaupun sebuah keberadaan dokumen XML hanya dapat menampung teks yang diberi *markup* dengan berbagai *tag* seperti telah dibahas sebelumnya, sebuah dokumen XML juga dapat mengandung hal-hal lain. Nyatanya ada tiga kesatuan yang dapat diuraikan oleh sebuah aplikasi XML (seperti digambarkan di [Ecks99]):

- **XML Document** Mengandung data dokumen, dietiketkan dengan unsur-unsur XML yang mempunyai arti, beberapa di antaranya dapat mengandung perlengkapan.

- **Document Type Definition** (DTD) Menentukan aturan-aturan bagaimana unsur-unsur dan perlengkapan-perlengkapan, dan penentapan data lainnya dari sebuah dokumen yang tunduk pada standar XML.
- **Stylesheet** Memerintahkan bagaimana sebuah dokumen sebaiknya dibentuk ketika mereka ditampilkan. Berbagai stylesheet yang berbeda dapat diterapkan pada dokumen yang sama, sehingga mengubah penampilannya tanpa memempengaruhi data yang mendasarinya.

Maka dari itu, DTD menentukan informasi apa saja yang dapat disimpan di dalam sebuah berkas XML dan *stylesheet* mengubah bentuk satu dokumen XML menjadi bentuk yang lain. DTD dan *stylesheet* ditetapkan di dalam standarnya masing-masing, yang akan ditinjau pada dua bagian berikut ini.

3.4 *Document Type Definition*

Keberadaan-keberadaan dari dokumen XML adalah *well-formed* (bentuk-baik) atau *valid* (sah). Sebuah keberadaan dokumen XML dikatakan *well-formed* apabila dokumen tersebut mengikuti sintaksis XML, yaitu bagian-bagian paling penting yang ditetapkan sebelumnya di *Terminologi XML* (subbab 3.2). Kebentukbaikan didefinisikan secara lengkap di [Grah99], namun pada umumnya sebuah dokumen adalah *well-formed* jika dokumen tersebut memiliki dari campuran tag awal dan akhir yang benar, semua attribut diberi tanda kutip secara benar, *character set* digunakan secara benar, dan seterusnya. Kebentukbaikan adalah satu hal yang harus dimiliki oleh semua dokumen XML; tanpa menjadi *well-formed*, sebuah dokumen XML bukanlah dokumen XML.

Hal lain yaitu sebuah keberadaan dokumen XML dapat *valid* (sah). Keabsahan secara tidak langsung menyatakan bahwa dokumen tersebut juga *well-formed*. Perbedaan di antara dokumen yang *valid* dan dokumen yang *well-formed* adalah sebuah dokumen *valid* harus memenuhi sebuah *Document Type Definition* (DTD), tetapi sebuah dokumen *well-formed* tidak memerlukannya. Maka dari itu dokumen *valid* dikatakan lebih ketat. Peran dari DTD adalah untuk menentukan elemen-elemen apa saja yang dapat berada di dalam suatu dokumen berikut susunannya. Berikut ini adalah contoh dari sebuah dokumen *well-formed*:

```
<?xml version="1.0" ?>
<henchman>
```

```
    <name age="45">Igor</name>
    <job>Asisten Laboratorium</job>
</henchman>
```

Seseorang dapat saja menambahkan berbagai *tag* dan/atau attribut lainnya ke dokumen ini dan akan masih *well-formed*:

```
<?xml version="1.0" ?>
<henchman>
    <name age="45">Igor</name>
    <job>Asisten Laboratorium</job>
    <salary>Tidak ada!</salary>
</henchman>
```

Walaupun demikian, ekstensibilitas ini tidak selalu diinginkan. Katakanlah contoh-contoh di atas diciptakan oleh seorang ilmuwan gila yang ingin menggunakan XML untuk melacak kakitangannya (yaitu *Henchmen Tracking Markup Language* – atau HTML). Jika ia juga ingin untuk menulis program-program komputer untuk mengolah bahasa tersebut, sangat mungkin ia akan ingin membatasi cara-cara *markup* yang dapat digunakan (sehingga program-programnya dapat bekerja dengan *instance* mana pun dari dokumennya). Hal ini dapat dicapai menggunakan DTD sebagai berikut:

```
<!-- DTD dari Henchmen Tracking Markup Language -->
<!ELEMENT henchman (name, job)* >
<!ELEMENT name (#PCDATA) >
<!ELEMENT job (#PCDATA) >
<!ATTRIBUTE name age CDATA>
```

Dokumen XML pertama di atas adalah sah menurut DTD ini, namun dokumen yang kedua tidak. Sebuah ciri penting dari DTD adalah kemampuannya untuk menetapkan jenis-jenis attribut yang berbeda. Sebuah daftar lengkap jenis-jenis attribut yang mungkin ada diberikan di [Marc99], namun jenis-jenis berikut ini adalah yang relatif sering digunakan:

- `CDATA` untuk attribut *string*.
- `ID` untuk pengenal, yaitu sebuah nama yang khas di dalam dokumen.
- `IDREF` adalah pengacuan kepada sebuah attribut `ID` yang ditentukan di tempat lain pada dokumen yang sama.
- `IDREFS` adalah sebuah daftar `IDREF` yang dipisahkan oleh spasi-spasi.
- `NMTOKEN` pada dasarnya adalah sebuah kata tanpa spasi.
- `NMTOKENS` adalah sebuah daftar attribut `NMTOKEN` yang dipisahkan oleh spasi-spasi.

Sebuah nilai awal dapat ditentukan untuk sebuah attribut. Jika dokumen tidak mempunyai attribut itu, maka nilanya dianggap nilai awal yang dapat mengambil salah satu dari nilai-nilai berikut:

- `#REQUIRED` berarti sebuah nilai harus disediakan di dokumen.
- `#IMPLIED` berarti jika tidak ada nilai yang diberikan, aplikasi yang mengolah dokumen XML harus menggunakan nilai awalnya sendiri.
- `#FIXED` diikuti oleh sebuah nilai berarti perlengkapan yang bersangkutan adalah tetap pada nilai tersebut.

Sebuah nilai *literal* perarti attribut tersebut akan diberikan nilai itu bila tidak ada yang ditentukan di dokumen.

3.4.1 Masa depan DTD

DTD merupakan warisan dari SGML. Karena alasan sejarah, kemampuan DTD menjadi agak terbatas. Keterbatasan terbesar adalah sintaks yaitu DTD bukanlah XML *well-formed*, yang berarti dokumen-dokumen DTD tidak dapat diolah oleh peralatan XML. Untuk memecahkan masalah ini, W3C sedang mengerjakan beberapa usulan untuk mengendalikan isi dokumen kemudahan dan daya yang lebih ketimbang yang dimungkinkan sekarang.

3.5 eXtensible Stylesheet Language (XSL)

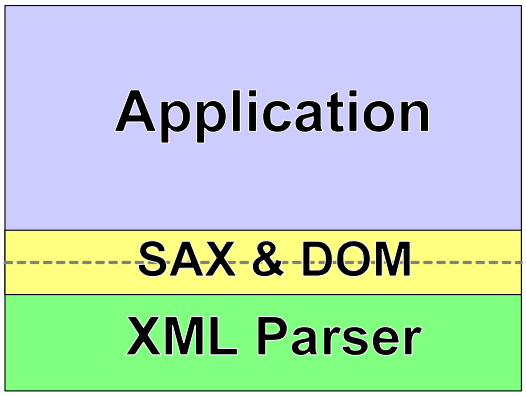
eXtensible Stylesheet Language (XSL) terdiri dari serangkaian *markup* yang dapat digunakan untuk menerapkan aturan-aturan pembentukan untuk setiap elemen di dalam sebuah dokumen XML. XSL bekerja dengan menerapkan berbagai aturan *style* pada isi dari sebuah dokumen XML, berdasarkan unsur-unsur yang ditemukannya [Ecks99]. XSL diatur menjadi dua bagian:

- XSLT; untuk XSL Transformation
- XSLFO; untuk XSL Formatting Objects

Sebuah XSL Processor menerima sebuah dokumen XSL dan menggunakannya untuk mentransformasikan suatu bentuk dokumen XML menjadi bentuk dokumen XML lainnya.

3.6 Pengurai XML

Walaupun struktur dokumen XML termasuk sederhana, pembuatan program untuk menguraikan dokumen-dokumen XML tidak dapat dikatakan suatu tugas yang sepele. Oleh karena itu muncul suatu jenis perangkat lunak yang disebut Pengurai XML (*XML Parsers*).



Gambar 3.1 Arsitektur pengurai XML secara umum

Sebuah pengurai XML umumnya berbentuk pustaka (*software library*) tingkat menengah yang memberikan layanan-layanan bagi aplikasi pada tingkat yang lebih tinggi untuk membaca serta mengambil data yang terletak di dalam dokumen XML. Pengurai-pengurai XML ini menetapkan API tertentu untuk berinteraksi dengan program aplikasi yang menggunakannya. API ini mendefinisikan *data model* dari sebuah dokumen XML kepada aplikasi yang menggunakan pengurai tersebut (**Gambar 3.1**).

Terdapat dua standar API untuk kegunaan ini: *Simple API for XML* (SAX) dan *Document Object Model* (DOM). SAX tercipta awalnya melalui proses informal yang terjadi di *mailing list* XML-DEV kemudian diterbitkan oleh David Megginson di http://www.megginson.com/SAX [Marc99]. Sedangkan DOM ditetapkan oleh W3C – sebuah badan formal yang mengawasi dan memberikan acuan-acuan untuk perkembangan *world-wide web*. Karena menggunakan paradigma yang berbeda, kedua API ini tidak menjadi saingan satu sama lain, melainkan saling melengkapi. Tidak jarang pustaka-pustaka pengurai XML memberikan kedua pilihan API bagi program-program aplikasi.

3.6.1 Simple API for XML (SAX)

Standar SAX menggunakan gaya Hollywood untuk memproses dokumen XML, "*Don't call us, we'll call you.*" Pustaka pengurai XML menggunakan fungsi-fungsi ataupun antarmuka *callback* untuk memanggil aplikasi pada saat dokumen XML diuraikan. Pusat kendali terletak pada pustaka pengurai ketimbang program aplikasi.

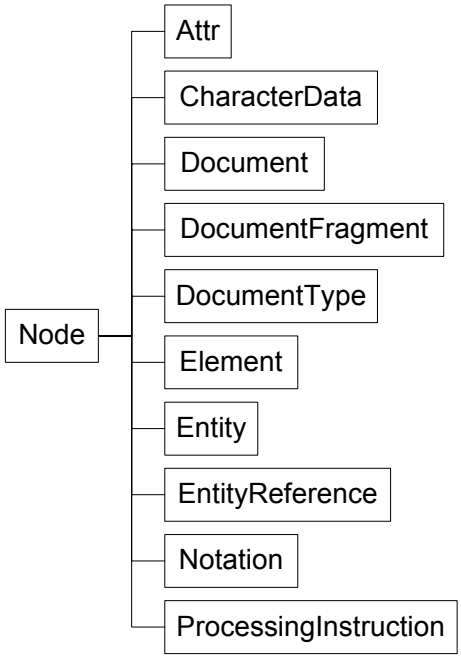
Penggunaan SAX lebih cocok pada aplikasi-aplikasi yang menggunakan XML sebagai representasi data eksternal dan menggunakan struktur datanya sendiri yang dapat berbeda jauh dari struktur dokumen-dokumen XML yang digunakannya. Sejalan dengan penguraian XML, fungsi-fungsi *callback* yang dipanggil pengurai XML mengisi struktur data aplikasi berdasarkan parameter-parameter yang diberikan.

Karena pengurai XML tidak perlu menyimpan data pada saat dokumen diuraikan, aplikasi-aplikasi yang menggunakan SAX cenderung memerlukan lebih sedikit *memory* untuk memproses dokumen XML. Selain itu, sifat penguraian SAX yang *on-the-fly* tidak memerlukan keberadaan seluruh dokumen sebelum penguraian dimulai. Hal ini kontras dengan DOM yang memerlukan seluruh dokumen diterima sebelum dapat memulai pemrosesan.

3.6.2 Document Object Model (DOM)

Standar DOM dikembangkan oleh W3C sebagai usaha untuk menyatukan model obyek dari *web browser* Netscape 3 dan Internet Explorer 3. Rekomendasi DOM mendukung dokumen XML maupun HTML.

Oleh Pengurai XML, data XML diwakilkan di dalam suatu struktur data *DOM Document Tree*. Struktur pohon di *memory* ini serupa dengan dokumen XML yang sedang diurai. Terdapat satu *node* untuk setiap elemen XML dengan tipenya masing-masing. Perwakilan data *Document Tree* digunakan secara bersama oleh Pengurai XML maupun aplikasi yang menggunakannya.



Gambar 3.2 Hirarki tipe dalam DOM [Marc99]

Dalam DOM, dokumen XML memiliki tipe `Document`. Elemen-elemen di dalam dokumen tersebut umumnya bertipe `Element`. Berbagai attribut yang dimiliki oleh elemen diwakili oleh obyek-obyek bertipe `Attr`. Data tekstual seperti komentar dan elemen yang berisi teks diwakili oleh `CharacterData`. Sedangkan ada beberapa tipe khusus lainnya seperti `Entity`, `EntityReference`, `Notation`, dan `ProcessingInstruction`. Kesemua tipe ini merupakan turunan dari tipe `Node` (**Gambar 3.2**).

Elemen terluar atau *root node* dari suatu dokumen bukanlah juga merupakan dokumen itu sendiri. Sebuah keberadaan dari `Document` memiliki paling banyak satu keberadaan dari `Node` yang menjadi *root node*. Sedangkan `Document` yang tidak mempunyai *root node* dikatakan dokumen kosong (*blank document*).

Traversal untuk masing-masing `Node` dilakukan secara *random access* dari `Node` yang memilikinya (parent node). Masing-masing node memiliki sebuah daftar dari anakannya dan daftar ini dapat diakses melalui operasi *indexing*.

Operasi-operasi berikut ini diberikan oleh `Node` untuk operasi *traversal* [Marc99]:

- `nodeType` adalah sebuah kode yang mewakili tipe sebenarnya dari `Node` yang bersangkutan.
- `parentNode` adalah induk dari *node* ini, jika ada.
- `childNode` adalah daftar anakan dari *node* ini.
- `firstChild` adalah anak pertama dari *node* ini.
- `lastChild` adalah anak terakhir dari *node* ini.
- `previousSibling` bila diberikan sebuah *child node*, maka akan memberikan *child node* sebelumnya.
- `nextSibling` bila diberikan sebuah *child node*, maka akan memberikan *child node* setelahnya.
- `attributes` adalah daftar attribut yang dimiliki oleh *node* ini, jika ada.

W3C mendefinisikan API DOM dalam bentuk Interface Definition Language (IDL) – sebuah standar yang digunakan oleh Object Management Group (OMG) untuk mendefinisikan antarmuka berorientasi obyek untuk komponen-komponen perangkat lunak namun tidak bergantung pada suatu bahasa pemrograman tertentu. Didefinisikannya DOM dalam IDL memungkinkan penerapan XML Parser untuk berbagai bahasa pemrograman – terdapat XML parser untuk C++, Java, SmallTalk, Ada, bahkan COBOL.

Walaupun IDL ditujukan untuk pemrograman terdistribusi menggunakan CORBA, namun tidak berarti bahwa semua pengurai XML harus diterapkan dalam bentuk obyek-obyek CORBA. Bahkan tidak ada pengurai XML yang diterapkan sebagai obyek-obyek CORBA [Marc99]. W3C hanya menggunakan fasilitas multi-bahasa dari IDL namun meninggalkan aspek-aspek distribusinya.

3.7 Standar Pendukung Lainnya

3.7.1 Namespaces

Namespace ditetapkan untuk memberikan kenunikan di antara elemen-elemen XML. Penggunaan *namespace* tidak mutlak, namun sangat disarankan. Di dalam dokumen XML, *namespace* ditetapkan dengan menggunakan sebuah *string* yang unik secara global – penggunaan Universal Resource Identifier (URI) sangat disarankan untuk menjamin keunikannya. URI ini dapat berupa Uniform Resource Locator (URL) yang mengacu pada suatu dokumen ataupun tidak mengacu pada apapun. Agar dokumen

tetap ringkas, sebuah URI *namespace* dapat diberikan suatu *prefix* yang lebih pendek dan keunikannya cukup dalam dokumen itu saja. Suatu deklarasi *namespace* tanpa prefix menandakan sebuah *default namespace* bagi dokumen XML yang memuatnya.

Katakanlah ilmuwan gila kita tadi takut orang lain akan menggunakan tag yang menggunakan nama yang sama dengan yang ada pada *Henchmen Tracking Markup Language*. Lagipula, tag `name` memang sangat umum. Hal ini dapat dipecahkan dengan menetapkan sebuah *namespace* untuk *markup language* tersebut. Misalkan nama *domain* dari ilmuwan tersebut adalah http://www.example.com, maka ia dapat membuat suatu namespace dengan URI http://www.example.com/dtd/html-0.1.dtd dan menggunakannya sebagai berikut:

```
<xml version="1.0">
<dd:henchman xmlns:dd="http://www.example.com/dtd/html-0.1.dtd">
<dd:name age="45">Igor</dd:name>
<dd:job>Asisten Laboratorium</dd:job>
</dd:henchman>
```

Atau versi lainnya seperti ini

```
<?xml version="1.0" ?>
<henchman xmlns:doomsday="http://www.example.com/dtd/html-0.1.dtd">
     <name age="45">Igor</name>
     <job>Asisten Laboratorium</job>
</henchman>
```

Dalam versi di atas, URI `http://www.example.com/dtd/html-0.1.dtd` menyatakan suatu *default namespace* dari dokumen XML secara keseluruhan. Ilmuwan ini dapat memberikan DTD untuk diperoleh dari URL tersebut ataupun tidak sama sekali.

3.7.2 XLink dan XPointer

XLink [Con00b] dan XPointer [Con00c] merupakan bakuan-bakuan untuk membuat hubungan antar sumber-sumber daya. Karena XLink menggunakan XPointer maka XPointer akan diperkenalkan terlebih dahulu.

Pada 3.4 telah diperkenalkan suatu attribut `ID` yang memberikan suatu nama yang unik di dalam suatu dokumen XML. Attribut `ID` ini dapat digunakan untuk mengacu suatu elemen di dalam sebuah dokumen XML. Katakanlah di dalam file XML `contoh.xml` terdapat suatu elemen dengan `ID smurf`. Maka elemen ini dapat diakses dengan XPointer berikut:

```
contoh.xml#smurf
```

Sebuah contoh yang lebih rumit yaitu:

```
contoh.xml#root().child(6,magic)
```

XPointer di atas mengakses elemen `<magic>` keenam dari `contoh.xml`. Sebuah penjelasan yang lebih lengkap mengenai XPointer bisa didapatkan di [Con00c].

Seperti telah dijelaskan di [Marc99], XLink memungkinkan penghubungan antara dokumen-dokumen XML. Selain hubungan antar dokumen, XLink juga memungkinkan penghubungan yang terjadi di antar elemen-lemen dalam satu dokumen (intra-dokumen).

Dalam XLink ada dua jenis hubungan:

- *Simple links* yang mirip seperti *link* yang terdapat di dalam HTML.
- *Extended links* yang memungkinkan penghubungan ke beberapa sumber daya dari satu *link* yang sama.

Berikut ini adalah contoh dari *simple link*. *Namespace* `xlink` digunakan untuk menandakan attribut-attribut yang menjadi bagian dari bakuan XLink. Contoh ini menggunakan XPointer yang digunakan sebagai contoh pada bagian sebelumnya.

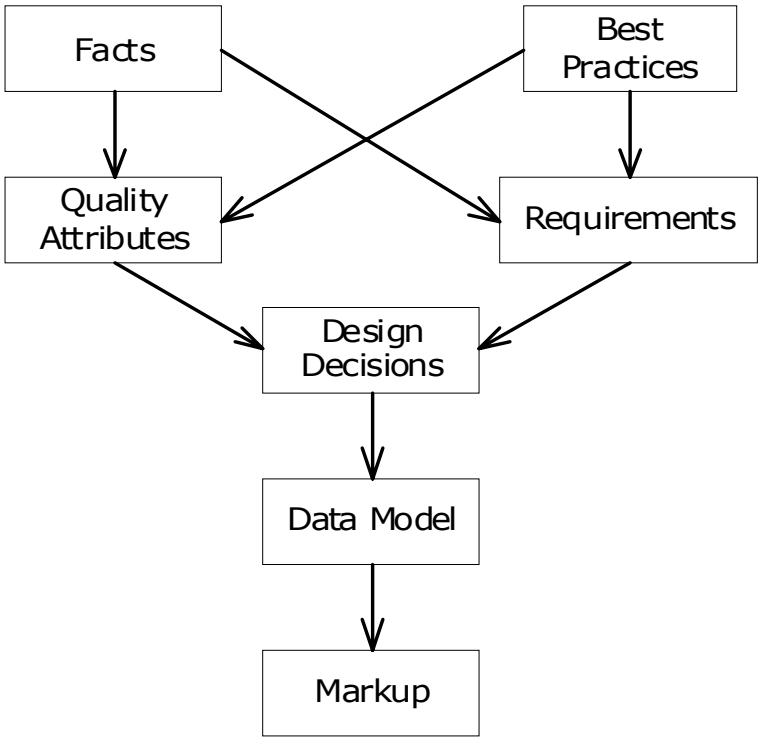
```
<example-element    xmlns:xlink="http://www.w3.org/1999/xlink"
xlink:type="simple" xlink:href="example.xml#root().child(6,magic)" />
```

Walaupun XLink menggunakan sintaks XML, namun XLink tidak mempunyai DTD. Tetapi DTD sebuah *markup* dapat dibuat untuk memaksakan sintaks XLink yang benar pada dokumen-dokumennya.

Bab 4

REQUIREMENTS MARKUP LANGUAGE

RQML adalah suatu usulan standar untuk penulisan dokumen *system requirements specification* di dalam *file* XML. Dikembangkan oleh Gardar Gudgeirsson, University of York [Gudg00], sebuah dokumen RQML bertujuan untuk menampung berbagai *requirement* yang ditulis dalam bahasa alami.



Gambar 4.1 Tahap-tahap perancangan RQML

Dalam mengembangkan RQML, Gudgeirsson memulai dengan penelaahan terhadap berbagai literatur dalam bidang *requirements engineering* (RE) dan juga *software engineering* secara umum. Dari studi literatur ini didapat serangkaian fakta yang menggambarkan praktek-praktek *software requirements engineering* pada masa kini. Selain itu dari salah satu literatur yang dipelajari, [Wieg99], didapatkan serangkaian *best practices* dalam RE.

Sebagai *domain information* untuk RE, berbagai fakta dan *best practices* ini mendasari dibuatnya sekumpulan *requirements* dan juga *quality attributes* yang menuntun perancangan RQML. Setelah beberapa keputusan dasar ("*design decisions*") diambil,

rancangan RQML dibuat dalam bentuk *data model* yang menjadi dasar dibentuknya sebuah *document type definition* (DTD) yang menentukan sintaks RQML.

Dalam bab ini diberikan pembahasan singkat mengenai fakta-fakta, *best practices*, dan *requirements* yang membentuk RQML. Selain itu juga akan dibahas mengenai *markup* RQML dengan mengutip bagian-bagian dari RQML DTD.

4.1 Fakta-fakta

Bagian ini memuat sebuah *subset* dari fakta-fakta yang dikemukakan oleh [Gudg00] di dalam studi literaturnya. Berbagai fakta yang dikemukakan ulang di sini adalah yang dapat dilacak balik (*back-trace*) dari rancangan RQML. Penomoran fakta sengaja disamakan dengan yang ada pada [Gudg00] sehingga nomor ini nampak tidak terurut – Gudgeirsson mengemukakan 16 fakta namun hanya enam di antaranya dapat dilacak balik secara eksplisit dari *data model* RQML.

4.1.1 Fact 1

Dengan banyaknya jenis *requirement* maka tidaklah mungkin untuk memberikan suatu ketetapan untuk mengkategorisasikannya ataupun untuk menggariskan cara-cara yang 'terbaik' di dalam penggambarannya; kendati cara-cara tertentu memang lebih baik dari yang lain.

4.1.2 Fact 2

Banyaknya teknik *elicitation* yang digunakan berakibat ganda. Pertama, karena himpunan *requirement* masih limbung maka berbagai *requirement* yang ada bisa saling meliputi dan/atau bertentangan. Kedua, seiring dengan jalannya pekerjaan menyebabkan munculnya berbagai *requirement* baru ataupun hilangnya *requirement* yang sudah ada.

4.1.3 Fact 3

Dengan cara-cara yang ada pada masa kini masih sulit untuk merekam berbagai *requirement* di seluruh aktivitas-aktivitas analisis dan negosiasi, dan pada saat yang bersamaan mengelola pertentangan antar *stakeholder* ataupun berhadapan dengan *requirement* yang tidak lengkap.

4.1.4 Fact 4

Seperti dituliskan di [Nuse00], semakin dirasa penting sekali kemampuan untuk tidak hanya menulis berbagai *requirement*, namun juga untuk melakukannya sedemikian rupa sehingga dapat dilacak oleh banyak orang untuk pengelolaan evolusinya sejalan dengan waktu.

4.1.5 Fact 7

Karena secara teoritis banyak sekali data pelacakan yang dapat dikumpulkan di suatu proyek, organisasi-organisasi harus dapat memilih dengan cermat data mana yang akan dipelihara. Menurut [Koto98], pada prakteknya sangatlah mahal sekali untuk mengumpulkan dan mengelola semua jenis informasi *traceability*.

4.1.6 Fact 8

Menurut [Ram98], pada saat ini tidak ada cara yang efisien di dalam menyimpan informasi *traceability* yang rumit.

4.2 Best Practices RE

Bagian ini memuat sebuah *subset* dari serangkaian *best practices* dalam bidang *requirements engineering* yang dikemukakan oleh [Wieg99]. *Subset* ini digunakan oleh Gudgeirsson untuk membuat berbagai *requirement* terhadap rancangan RQML. Berbagai *best practices* yang tidak dipergunakan umumnya karena tidak terkait dengan penulisan dokumen SRS – karena itu di luar cakupan RQML.

Penomoran *best practices* pada bagian ini disamakan dengan yang ada di [Gudg00] sehingga nampak tidak terurut.

- BP4 Membuat suatu daftar kata-kata.
- BP8 Melacak setiap perubahan pada semua hasil kerja yang terpengaruh.
- BP9 Membuat garis pangkalan serta mengendalikan versi-versi dari *requirements documents*..
- BP11 Melacak status berbagai *requirement*.
- BP12 Mengukur kemantapan *requirement*.
- BP18 Melacak usaha-usaha yang dihabiskan untuk berbagai *requirement*.

- BP19 Menulis visi serta cakupan.
- BP21 Mengenali kelas-kelas pengguna.
- BP22 Memilih *product champion*.
- BP23 Membentuk *focus groups*.
- BP24 Mengenali *use-cases*.
- BP27 Menetapkan *quality attributes*.
- BP29 Menggunakan kembali berbagai *requirement*.
- BP33 Memprioritaskan *requirement*.
- BP34 Memodelkan berbagai *requirement*.
- BP38 Mengenali sumber-sumber dari berbagai *requirement*.
- BP39 Memberi nama pada setiap *requirement*.
- BP40 Mencatat aturan-aturan bisnis.
- BP41 Membuat matriks *traceability*.

4.3 *Quality Attributes*

Bagian ini memuat tiga attribut kualitas yang membentuk rancangan RQML. Sebagai analogi dari nilai-nilai moral, attribut-attribut kualitas ini menuntun berbagai pengambilan keputusan terhadap hasil akhir dari sintaks RQML.

- ***Integratibility***. Dapat digabungkan berarti kemampuan komponen-komponen yang dibuat secara terpisah untuk dapat bekerja sama dengan benar. Hal ini bergantung pada kompleksitas eksternal dari komponen-komponen tersebut, mekanisme-mekanisme interaksi serta berbagai protokol yang dimilikinya, dan juga tingkat pemisahan tanggung jawab dari masing-masing komponen.
- ***Extensibility***. Dapat dikembangkan mengukur seberapa mudah untuk menambah fungsionalitas kepada sistem. Ini juga merupakan suatu indikasi mengenai seberapa mudah untuk menambahkan komponen-komponen yang ada di pasaran terhadap sistem yang sudah dipergunakan.
- ***Portability***. Dapat dipindahkan yaitu kemampuan sistem untuk dijalankan di dalam berbagai lingkungan komputasi yang berbeda. Sebuah sistem disebut *portable* bila sedemikan rupa sehingga semua anggapan memgenai lingkungan

komputasi tertentu dibatasi pada satu komponen atau pada beberapa komponen yang sangat berhubungan.

4.4 Requirements

Bagian ini memuat seluruh *requirement* yang diminta dari RQML. Beberapa *requirement* ini dapat dilacak secara langsung kepada *best practices* maupun fakta-fakta yang telah dikemukakan sebelumnya. *Traceability* ada pada nomor *requirement* yang dicetak tebal dan diberikan sebagai nomor *best practice* (BP*x*) ataupun nomor fakta (Fact *x*) pada akhir kalimat *requirement*. Persis setelah nomor *requirement* ditulis kategori dari requirement tersebut dalam kurung siku, yaitu **[fungsional]** atau **[non-fungsional]**.

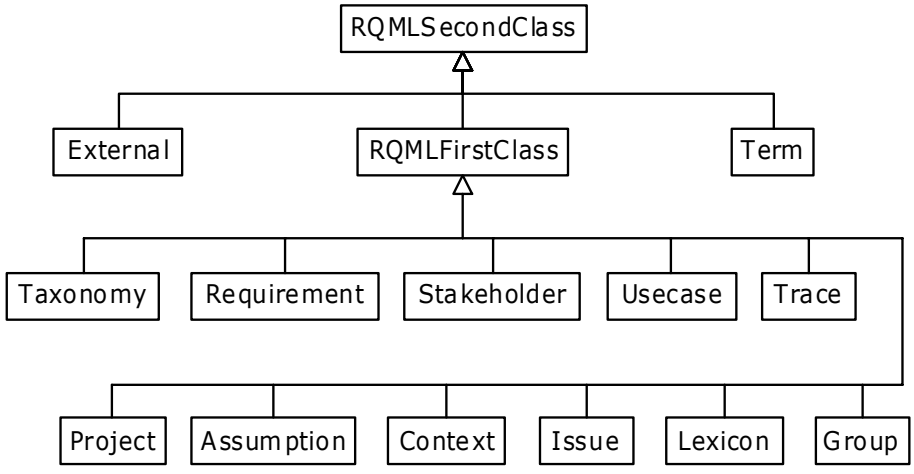
- REQ1 [fungsional] RQML akan menjadi sebuah data format untuk menyimpan *software requirements specification*.
- REQ2 [non-fungsional] RQML akan menggabungkan berbagai kekuatan metode-methode masa kini yang menggunakan dokumen maupun database untuk menyimpan berbagai *requirement*.
- REQ3 [fungsional] RQML akan dapat dikembangkan menuju metode-metode pemodelan baru. *Requirement* ini adalah fungsional dan bukanlah attribut kualitas *extensible*.
- REQ4 [non-fungsional] RQML akan diarahkan untuk digunakan sebagai standar penyimpanan data untuk alat bantu *requirement*.
- REQ5 [fungsional] RQML akan mendukung *best practices* untuk *requirements engineering* (RE); namun tidak mendukung suatu metode RE tertentu.
- **REQ5-1** [fungsional] RQML akan mendukung pendokumentasian daftar istilah (BP4).
- **REQ5-2** [fungsional] RQML akan memungkinkan pelacakan setiap perubahan pada hasil-hasil kerja (BP8).
- **REQ5-3** [fungsional] RQML akan mendukung pembuatan garis pangkal serta pengendalian versi dari berbagai *requirements document* (BP9).
- **REQ5-4** [fungsional] RQML akan memungkinkan pelacakan status *requirement* (BP11).

- **REQ5-5** [fungsional] RQML akan membuat mungkin pengukuran kemantapan *requirement* (BP12).
- **REQ5-6** [fungsional] RQML akan membuat mungkin untuk mendokumentasikan usaha yang dilakukan dalam penerapan masing-masing *requirement*. (BP18)
- **REQ5-7** [fungsional] RQML akan membuat mungkin pendokumentasian visi dan cakupan untuk proyek-proyek tertentu (BP19).
- **REQ5-8** [fungsional] RQML akan mendukung pengenalan kelas-kelas pengguna (BP21).
- **REQ5-9** [fungsional] RQML akan mendukung pemilihan berbagai *product champion* (BP22).
- **REQ5-10** [fungsional] RQML akan membuat mungkin untuk membentuk berbagai *focus group* (BP23).
- **REQ5-11** [fungsional] RQML akan memungkinkan untuk menetapkan berbagai attribut kualitas (BP27).
- **REQ5-12** [fungsional] RQML akan mendukung penggunaan kembali berbagai *requirement* (BP29).
- **REQ5-13** [fungsional] RQML akan memungkinkan pemrioritasan *requirement* (BP33).
- **REQ5-14** [fungsional] RQML akan medukung pendokumentasian requirement dalam bahasa alami. (BP24, BP34)
- **REQ5-15** [fungsional] RQML akan memungkinkan untuk mengenali sumber-sumber dari berbagai *requirement* (BP38).
- **REQ5-16** [fungsional] RQML akan mendukung penamaan berbagai *requirement* (BP39).
- **REQ5-17** [fungsional] RQML akan mendukung perekaman aturan-aturan bisnis (BP40).
- **REQ5-18** [fungsional] RQML akan memberikan suatu cara untuk menyimpan informasi *traceability* yang merupakan sebuah peningkatan dari cara-cara yang digunakan sekarang (BP41, Fact 7, Fact 8).
- **REQ6** [fungsional] RQML akan menyediakan suatu cara untuk mengklasifikasikan jenis-jenis dari berbagai requirement (Fact 1).

- **REQ7** [fungsional] RQML akan mendukung pendokumentasikan berbagai *requirement* yang berubah ataupun tidak lengkap (Fact 2, Fact 3, Fact 4).
- **REQ8** [fungsional] RQML akan mendukung *traceability* ke dalam dan ke luar dari bagian-bagian teks dalam bahasa alami (Fact 7).
- REQ9 [fungsional] RQML akan memungkinkan penamaan, penggambaran, serta pemberian alasan untuk berbagai kesatuan terkait yang sedang didokumentasi-kan.
- REQ10 [fungsional] RQML akan mendukung pendokumentasian informasi *stakeholder*.
- REQ10-1 [fungsional] RQML akan membuat mungkin pendokumentasian berbagai anggapan.
- REQ10-2 [fungsional] RQML akan membuat mungkin pendokumentasian masalah-masalah yang belum terselesaikan.

4.5 *Data Model*

Bagian ini membahas *data model* yang digunakan sebagai dasar dibuatnya markup RQML. Paradigma *data model* yang digunakan adalah data berorientasi obyek, tetapi semua kelas hanya terdiri dari attribut (tanpa adanya *method*). Dari *data model* ini nampak jenis-jenis dari berbagai data yang dapat dimuat di dalam suatu dokumen RQML.



Gambar 4.2 Hirarki *data model* RQML

4.5.1 Kelas **RQMLSecondClass**

Kelas `RQMLSecondClass` adalah *base class* bagi semua kelas di dalam RQML. Kelas ini memuat attribut-attribut umum yang harus dimiliki oleh semua elemen. Berbagai attribut tersebut adalah:

- ID merupakan identitas unik dari obyek ini di dalam suatu file RQML.
- URI mereferensikan obyek lain di luar file RQML lewat suatu URI (*Universal Resource Identifier*).

4.5.2 Kelas **External**

Kelas **External** mewakili obyek-obyek yang ada di luar *file* XML (dokumen RQML). Pada dasarnya, obyek apapun yang mempunyai URI dapat diwakili oleh suatu obyek **External**.

4.5.3 Kelas **RQMLFirstClass**

Kelas **RQMLSecondClass** adalah *base class* bagi elemen-elemen yang disebut elemen *first-class.* Berbagai elemen *first-class* ini adalah *top-level elements* – yaitu elemen teratas sebelum elemen akar (*root element*) – di dalam file RQML. Attribut-attribut yang harus dimiliki oleh sebuah elemen *first-class* adalah:

- Versi dari obyek *first-class* ini (REQ5-3).
- Status dari obyek *first-class* ini (REQ5-4, REQ 5-7).
- Kemantapan dari obyek *first-class* ini (REQ5-5).
- Kesulitan yang ada dalam mewujudkan informasi yang ada dalam obyek *first-class* ini (REQ5-6).
- Prioritas dari obyek *first-class* ini (REQ5-13).
- Nama dari obyek *first-class* ini (REQ9).
- Keterangan dari obyek *first-class* ini (REQ9)
- Alasan yang mendasari adanya masukan obyek *first-class* ini (REQ9).

4.5.4 Kelas **Term**

Obyek-obyek dari kelas **Term** digunakan sebagai elemen tujuan (*target* atau *anchor*) dalam melakukan *traceability* ke dalam dan ke luar dari teks bahasa alami. Kelas ini akan diturunkan langsung dari **RQMLSecondClass** (REQ8, REQ5-1).

4.5.5 Jenis-jenis obyek *first-class*

Dalam bagian ini diberikan jenis-jenis obyek *first-class* yang ada dalam RQML. Berbagai obyek *first-class* ini adalah *subclass* dari `RQMLFirstClass`.

- **Taxonomy** Menyediakan suatu cara untuk membuat berbagai klasifikasi di mana obyek-obyek *first-class* lainnya dapat diasosiasikan dengannya. Umumnya obyek ini digunakan untuk mengklasifikasikan berbagai *requirement* menjadi *functional requirement*, *non-functional requirement*, *business rules*, dan sebagainya, namun juga dapat digunakan untuk mengklasifikasikan obyek-obyek *first-class* lainnya (REQ6, REQ5-11, REQ5-10, REQ5-8, REQ5-17).
- **Requirement** Merupakan suatu *requirement* di dalam bahasa alami (REQ5-14).
- **Stakeholder** Menggambarkan seorang *stakeholder* (REQ5-9, REQ10).
- **Usecase** Adalah gambaran sebuah *use-case* (REQ5-14).
- **Project** Merupakan gambaran luas sebuah proyek ataupun sub-proyek (REQ5-7).
- **Assumption** Menggambarkan sebuah anggapan (REQ10-1).
- **Context** Menyediakan sebuah konteks di mana obyek-obyek *first-class* lainnya dapat diasosiasikan dengannya. Salah satu contoh konteks adalah *domain information* (REQ5-15).
- **Issue** Mendokumentasikan sebuah permasalahan yang belum terpecahkan (REQ10-2)[4].
- **Lexicon** Menyimpan definisi dari sebuah konsep (REQ5-1).
- **Group** Mengelompokkan beberapa obyek *first-class* (REQ5-10).
- **Trace** Menetapkan pelacakan dari satu obyek *first-class* ke obyek *first class* yang lain.

4.5.6 Jenis-jenis *trace*

Berikut ini adalah jenis-jenis *traceability* yang terdapat di dalam RQML. Dimuat sebagai suatu obyek *first-class* **Trace**, berbagai *traceability* ini menghubungkan dua

[4] Ada kekeliruan di dalam [Gudg00], halaman 46, yang merunutkan obyek ini ke REQ10-1.

obyek *first-class* di dalam satu *file* XML. Untuk masing-masing jenis *traceability* diberikan nomor *requirement* yang mendasarinya.

- **parent-type** Adalah pelacakan antara dua obyek *taxonomy* (REQ5-6).
- **instantiation** Memungkinkan pembuatan suatu keberadaan dari satu obyek *first-class* (selain dari *taxonomy*) ke obyek *first-class* yang lain (REQ5-6).
- **contextual-scope** Memungkinkan sebuah obyek *first-class* untuk diasosiasikan dengan konteks tertentu (REQ5-10, REQ5-15).
- **baseline** Menetapkan sebuah garis pangkal untuk obyek-obyek *first-class* dari sebuah proyek (REQ5-3, REQ5-7).
- **subproject** Menetapkan sebuah proyek sebagai bagian dari proyek lain (REQ5-3, REQ5-7).
- **project-data** Menghubungkan suatu proyek ke obyek *second-class* yang berhubungan dengannya (REQ5-3, REQ5-7).
- **refinement** Perincian dari satu *requirement* ataupun *use case* ke yang lainnya (REQ5-2, REQ5-3).
- **supersede** Memungkinkan satu obyek *first-class* untuk digantikan dengan yang lain (REQ5-2, REQ5-3).
- **parent** Menetapkan suatu obyek *first-class* sebagai induk dari yang lain (REQ5-2, REQ5-3).
- **conflict** Menunjukkan bahwa dua *requirement* atau *use case* bertentangan (REQ5-2, REQ5-3).
- **forward-from** Adalah suatu pelacakan tradisional *forward-from* (REQ5-2, REQ15).
- **backward-to** Adalah suatu pelacakan tradisional *backward-to* (REQ5-2, REQ15).
- **forward-to** Adalah suatu pelacakanpelacakan tradisional *forward-to* (REQ5-2, REQ15).
- **backward-from** Adalah suatu pelacakanpelacakan tradisional *backward-from* (REQ5-2, REQ15).
- **stakeholder-view** Memungkinkan seorang *stakeholder* untuk dihubungkan dengan suatu obyek *first-class* yang menandakan kepemilikan, ketertarikan, atau jenis-jenis hubungan lainnya (REQ10-1).

- **trace-to-trace** Menghubungkan antara dua *trace* (REQ10-1).
- **general-conflict** Menandakan adanya konflik antara dua obyek *first-class* (REQ5-2).
- **lexref** Menciptakan suatu hubungan antara sebuah **term** dengan masukan **lexicon** untuk **term** tersebut (REQ5-1).
- **see-also** Menghubungkan antara dua masukan **lexicon** (REQ5-1).
- **universal-trace** Adalah pelacakan untuk penggunaan umum (REQ5-2).

4.6 Markup

Pada bagian ini akan dibahas *document type definition* (DTD) yang menjadi penentu sintaks RQML. DTD ini dibuat dari *data model* RQML yang dibahas pada bagian sebelumnya.

Karena XML tidak *object-oriented* maka pemetaan dari *data model* ke DTD dilakukan dengan cara meratakan ("*flatten*") pohon hirarki *data model* (**Gambar 4.2**) kemudian membuat definisi *tag* XML dari berbagai *leaf node* yang dimiliki oleh pohon tersebut. Tipe-tipe data sederhana (terutama *string* dan enumerasi) dipetakan menjadi attribut dari suatu elemen. Sedangkan jenis-jenis data kompleks dijadikan sub-elemen dari elemen yang memilikinya. Cara pemetaan ini analogis dengan cara meng-*instantiate* semua class kemudian membuat elemen-elemen dari obyek-obyek yang merupakan *instance* dari *leaf class*.

4.6.1 *Root Element*

Elemen `rqml` merupakan elemen akar di dalam suatu dokumen RQML. Dokumen ini memuat hanya memuat berbagai elemen *first-class* sebagai *child element* langsung.

Perlu diketahui bahwa URL http://www.rqml.org yang digunakan sebagai basis *namespace* bukanlah *website* RQML[5]; situs web RQML yang terakhir diketahui adalah http://www.raqoon.is/rqml/rqml-resources.htm.

```
<!ELEMENT rqml (taxonomy | requirement | stakeholder | use-case | project |
assumption | context | issue | lexicon | group | trace)*>
<!ATTLIST rqml
     xmlns CDATA #FIXED "http://www.rqml.org/namespaces/rqml"
>
```

[5] Nampaknya domain `rqml.org` telah diambil alih oleh pihak lain.

4.6.2 Elemen *term*

Elemen `term` digunakan di banyak elemen lain untuk memungkinkan pemberian attribut kepada sekelompok kata di dalam sebuah blok teks bahasa alami. Selain itu elemen ini juga digunakan untuk melakukan *traceability* ke dalam bagian teks bahasa alami.

Ada suatu ketidakkonsistenan di dalam [Gudg00] pada definisi elemen ini. Walau seharusnya elemen ini adalah turunan dari `RQMLSecondClass`, namun attribut URI tidak diwariskannya (lihat subbab 4.5.1 untuk keterangan kelas `RQMLSecondClass`).

```
<!ELEMENT term (#PCDATA)>
<!ATTLIST term
     id ID #REQUIRED
     priority (low | normal | high) "normal"
     difficulty (low | normal | high) "normal"
     status (proposed | approved | incorporated | validated) #IMPLIED
     stability (low | normal | high) "normal"
     version CDATA #IMPLIED
>
```

4.6.3 Kelas RQMLFirstClass

Kelas `RQMLFirstClass` merupakan suatu *abstract class* yang menjadi dasar bagi elemen-elemen lainnya. Walau sebenarnya tidak ada di dalam DTD, namun kelas ini berguna untuk mendefinisikan berbagai attribut serta *child elements* yang umum ada di semua elemen-elemen *first-class*. Elemen-elemen `taxonomy`, `requirement`, `stakeholder`, `use-case`, `project`, `assumption`, `issue`, `lexicon`, `group`, dan `trace` semua memiliki sekelompok attribut dan elemen anakan yang sama; kesamaan itu didefinisikan di `RQMLFirstClass`.

Kelas ini mengalami ketidakkonsistenan yang sama dengan elemen `term`, yaitu tidak adanya attribut URI yang seharusnya diwariskan dari `RQMLSecondClass`.

```
<!-- ========== RQMLFirstClass - NOT included in RQML DTD ========== -->
<!ELEMENT RQMLFirstClass (name?,description*,rationale?)>
<!ATTLIST RQMLFirstClass(
     id ID #REQUIRED
     priority (low | normal | high) "normal"
     difficulty (low | normal | high) "normal"
     status (proposed | approved | incorporated | validated) #IMPLIED
     stability (low | normal | high) "normal"
     version CDATA #IMPLIED
>
<!ELEMENT name (#PCDATA)>
<!ELEMENT description (#PCDATA | term)*>
<!ATTLIST description
```

```
    headline CDATA #IMPLIED
>
<!ELEMENT rationale (#PCDATA | term)*>
<!ELEMENT term (#PCDATA)>
<!ATTLIST term
    id ID #REQUIRED
    priority (low | normal | high) "normal"
    difficulty (low | normal | high) "normal"
    status (proposed | approved | incorporated | validated) #IMPLIED
    stability (low | normal | high) "normal"
    version CDATA #IMPLIED
>
```

4.6.4 Elemen-elemen *first-class*

Pada bagian ini diberikan penjelasan singkat mengenai elemen-elemen *first-class* yang semuanya merupakan turunan dari `RQMLFirstClass`. Pada beberapa definisi elemen ditemukan ketidakkonsistenan yang ada di dalam [Gudg00]. Berbagai ketidak-konsistenan tersebut akan ditunjukkan di dalam keterangan elemen yang bersang-kutan.

4.6.4.1 Requirement

Elemen `requirement` digunakan untuk mendokumentasikan sebuah *requirement* dalam bahasa alami. Elemen ini tidak menambahkan informasi apapun dari yang didapatnya dari kelas `RQMLFirstClass`; namun hanya merupakan penamaan ulang dari *superclass*-nya.

```
<!-- requirement  -->
<!ELEMENT requirement (name?, description*, rationale?)>
<!ATTLIST requirement
    id ID #REQUIRED
    priority (low | normal | high) "normal"
    difficulty (low | normal | high) "normal"
    status (proposed | approved | incorporated | validated) #IMPLIED
    stability (low | normal | high) "normal"
    version CDATA #IMPLIED
>
<!-- "name" was already declared -->
<!-- "description" was already declared -->
<!-- "rationale" was already declared -->
```

4.6.4.2 Stakeholder

Seperti `requirement`, elemen `stakeholder` tidak menambah informasi apapun kepada `RQMLFirstClass`. Elemen ini menampung nama, keterangan, dan alasan untuk seorang *stakeholder*. Walaupun informasi tambahan seperti alamat, e-mail, dan lain-lain akan menguntungkan, tambahan ini tidak disertakan agar rancangan RQML tetap sederhana. Lagipula, informasi data pribadi seperti ini umumnya diletakkan di dalam suatu *relational database* dan dapat dihubungkan dengan menggunakan *traceability*.

```
<!-- stakeholder-->
<!ELEMENT stakeholder (name?, description*, rationale?)>
<!ATTLIST stakeholder
     id ID #REQUIRED
     priority (low | normal | high) "normal"
     difficulty (low | normal | high) "normal"
     status (proposed | approved | incorporated | validated) #IMPLIED
     stability (low | normal | high) "normal"
     version CDATA #IMPLIED
>
<!-- "name" was already declared -->
<!-- "description" was already declared -->
<!-- "rationale" was already declared -->
```

4.6.4.3 Assumption

Elemen `assumption` adalah suatu penamaan ulang dari `RQMLFirstClass` yang ditujukan untuk mendokumentasikan sebuah anggapan.

Ternyata ada ketidakkonsistenan di [Gudg00] untuk spesifikasi elemen ini, yaitu spesifikasi elemen-elemen anakan yang dimiliki oleh elemen `assumption` berbeda dengan yang digariskan oleh *superclass*-nya, `RQMLFirstClass`. Perbedaan ini dicetak tebal dalam penggalan berikut.

```
<!--assumption-->
<!ELEMENT assumption (name?, description*, rationale?)*>
<!ATTLIST assumption
     id ID #REQUIRED
     priority (low | normal | high) "normal"
     difficulty (low | normal | high) "normal"
     status (proposed | approved | incorporated | validated) #IMPLIED
     stability (low | normal | high) "normal"
     version CDATA #IMPLIED
>
<!-- "name" was already declared -->
<!-- "description" was already declared -->
<!-- "rationale" was already declared -->
```

4.6.4.4 Issue

Seperti `requirement`, `stakeholder`, dan `assumption`, elemen `issue` adalah penamaan ulang dari `RQMLFirstClass`. Kegunaan elemen ini adalah untuk mendokumentasikan permasalahan yang belum terpecahkan.

```
<!-- issue -->
<!ELEMENT issue (name?, description*, rationale?)>
<!ATTLIST issue
     id ID #REQUIRED
     priority (low | normal | high) "normal"
     difficulty (low | normal | high) "normal"
     status (proposed | approved | incorporated | validated) #IMPLIED
     stability (low | normal | high) "normal"
     version CDATA #IMPLIED
>
<!-- "name" was already declared -->
```

```
<!-- "description" was already declared -->
<!-- "rationale" was already declared -->
```

4.6.4.5 Use-Case

Elemen `use-case` didasari oleh template *use-case* yang diambil dari [Dary00]. Elemen ini mendokumentasikan seorang aktor, nol atau beberapa *precondition* dan *postcondition*, nol atau beberapa *alternative course*, dan nol atau beberapa *exception*.

```
<!-- use-case -->
<!ELEMENT use-case (name?, description*, rationale?, actor, precondition*,
postcondition*, normal-course, alternative-course*, exception*)>
<!ATTLIST use-case
     id ID #REQUIRED
     priority (low | normal | high) "normal"
     difficulty (low | normal | high) "normal"
     status (proposed | approved | incorporated | validated) #IMPLIED
     stability (low | normal | high) "normal"
     version CDATA #IMPLIED
>
<!-- "name" was already declared -->
<!-- "description" was already declared -->
<!-- "rationale" was already declared -->
<!ELEMENT actor (#PCDATA)>
<!ELEMENT precondition (#PCDATA)>
<!ELEMENT postcondition (#PCDATA)>
<!ELEMENT normal-course (description*, (actoraction | systemresponse)*)>
<!ELEMENT alternative-course (description*, (actoraction | systemresponse)*)>
<!ELEMENT exception (description*, (actoraction | systemresponse)*)>
<!ELEMENT actoraction (#PCDATA)>
<!ELEMENT systemresponse (#PCDATA)>
```

4.6.4.6 Project

Elemen `project` digunakan untuk mendokumentasikan suatu proyek atau subproyek.

Ada beberapa ketidakkonsistenan di dalam [Gudg00] mengenai definisi elemen ini. Perbedaan pertama yaitu spesifikasi *child elements* yang diperbolehkan bukan merupakan *superset* dari yang dimiliki oleh `RQMLFirstClass`, yang seharusnya menjadi *superclass* dari elemen ini. Perbedaan kedua yaitu adanya ketidaksamaan di antara bagian pembahasan dengan bagian lampiran untuk elemen ini.

Di [Gudg00], halaman 57 (*Chapter 3. Design*), elemen `project` didefinisikan sebagai:

```
<!ELEMENT project (product | problem | vision | scope)* >
<!ATTLIST project
     id ID #REQUIRED
     priority (low | normal | high) "normal"
     difficulty (low | normal | high) "normal"
     status (proposed | approved | incorporated | validated) #IMPLIED
     stability (low | normal | high) "normal"
     version CDATA #IMPLIED
>
<!-- "name" was already declared -->
```

```
<!-- "description" was already declared -->
<!-- "rationale" was already declared -->
<!ELEMENT product (name, description*)>
<!ELEMENT problem (#PCDATA | term)*>
<!ELEMENT scope (#PCDATA | term)*>
<!ELEMENT vision (#PCDATA | term)*>
```

Sedangkan di halaman 90 (*RQML DTD*), spesifikasinya adalah:

```
<!ELEMENT project (name?, description*, rationale?, product?, client?,
purpose?, goal?, problem?, background?)>
<!ATTLIST project
     id ID #REQUIRED
     priority (low | normal | high) "normal"
     difficulty (low | normal | high) "normal"
     status (proposed | approved | incorporated | validated) #IMPLIED
     stability (low | normal | high) "normal"
     version CDATA #IMPLIED
>
<!-- "name" was already declared -->
<!-- "description" was already declared -->
<!-- "rationale" was already declared -->
<!ELEMENT product (name, description*)>
<!ELEMENT client (name, description*)>
<!ELEMENT purpose (#PCDATA | term)*>
<!ELEMENT goal (#PCDATA | term)*>
<!ELEMENT problem (#PCDATA | term)*>
<!ELEMENT background (#PCDATA | term)*>
```

4.6.4.7 Taxonomy

Elemen `taxonomy` hanya menambahkan satu buah informasi kepada kelas `RQMLFirst-Class`. Tambahan ini adalah attribut `type-element` yang merupakan jenis-jenis elemen *first class* lainnya kecuali `term` dan `taxonomy`.

```
<!-- taxonomy -->
<!ELEMENT taxonomy (name?, description*, rationale?)>
<!ATTLIST taxonomy
     id ID #REQUIRED
     priority (low | normal | high) "normal"
     difficulty (low | normal | high) "normal"
     status (proposed | approved | incorporated | validated) #IMPLIED
     stability (low | normal | high) "normal"
     version CDATA #IMPLIED
     type-element (requirement | stakeholder | use-case | project | assumption
| context | issue | lexicon | group | trace) #REQUIRED
>
<!-- "name" was already declared -->
<!-- "description" was already declared -->
<!-- "rationale" was already declared -->
```

4.6.4.8 Context

Elemen `context` memungkinkan pendokumentasian *domain information* ataupun berbagai informasi lainnya yang menyusun konteks untuk bagian-bagian sistem yang lain. Tambahan yang diberikan kepada kelas `RQMLFirstClass` adalah elemen-elemen `origin` dan `textual-data`.

```
<!-- context -->
<!ELEMENT context (name?, description*, rationale?, origin?, textual-data?)>
<!ATTLIST context
     id ID #REQUIRED
     priority (low | normal | high) "normal"
     difficulty (low | normal | high) "normal"
     status (proposed | approved | incorporated | validated) #IMPLIED
     stability (low | normal | high) "normal"
     version CDATA #IMPLIED
>
<!-- "name" was already declared -->
<!-- "description" was already declared -->
<!-- "rationale" was already declared -->
<!ELEMENT origin (#PCDATA)>
<!ELEMENT textual-data (#PCDATA | term)*>
<!-- "term" was already declared -->
```

4.6.4.9 Lexicon

Elemen `lexicon` memungkinkan berbagai definisi diberikan untuk elemen-elemen `term` yang nampak di bagian-bagian lain. *Traceability* dapat digunakan untuk menghubungkan elemen-elemen ini dengan elemen `lexicon`.

```
<!-- lexicon -->
<!ELEMENT lexicon (name?, description*, rationale?, definition)>
<!ATTLIST lexicon
     id ID #REQUIRED
     priority (low | normal | high) "normal"
     difficulty (low | normal | high) "normal"
     status (proposed | approved | incorporated | validated) #IMPLIED
     stability (low | normal | high) "normal"
     version CDATA #IMPLIED
>
<!-- "name" was already declared -->
<!-- "description" was already declared -->
<!-- "rationale" was already declared -->
<!ELEMENT definition (#PCDATA)>
```

4.6.4.10 Group

Elemen `group` digunakan untuk mengelompokkan beberapa elemen lainnya. Mekanisme pengelompokkan yang digunakan yaitu XLink. Pengelompokan ini ditujukan untuk membantu bekerjanya sebuah *focus group* (REQ5-10).

```
<!-- group -->
<!ELEMENT group (name?, description*, rationale?, member*)>
<!ATTLIST group
     id ID #REQUIRED
     priority (low | normal | high) "normal"
     difficulty (low | normal | high) "normal"
     status (proposed | approved | incorporated | validated) #IMPLIED
     stability (low | normal | high) "normal"
     version CDATA #IMPLIED
     xmlns:xlink CDATA #FIXED "http://www.w3c.org/1999/xlink"
     xlink:type (extended) #FIXED "extended"
>
<!-- "name" was already declared -->
<!-- "description" was already declared -->
```

```
<!-- "rationale" was already declared -->
<!ELEMENT member (#PCDATA)>
<!-- must be the location of a first class object -->
<!ATTLIST member
     xlink:type (location) #FIXED "location"
     xlink:href CDATA #REQUIRED
     xlink:label NMTOKEN #REQUIRED
>
```

4.6.4.11 Trace

Elemen `trace` digunakan untuk melakukan *traceability* antara elemen-elemen first-class lainnya. Fungsionalitas *traceability* yang diberikan oleh elemen ini lebih dari hanya empat jenis *trace* tradisional (*forward-to*, *backward-to*, *forward-from*, dan *backward-from*) namun juga untuk hal-hal seperti hubungan *parent-child*, pengkategorisasian elemen, pembuatan garis pangkal, *konflik*, dan berbagai hubungan lainnya.

```
<!-- trace -->
<!ELEMENT trace (name?, description*, rationale?, source, sink, (parent-type
| instantiation | contextual-scope | baseline | subproject | project-data |
refinement | supercede | parent | conflict | forward-from | backward-to |
backward-from | forward-to | stakeholder-view | trace-to-trace | gen-
eral-conflict | lexref | see-also | universal-trace))>
<!ATTLIST trace
     id ID #REQUIRED
     priority (low | normal | high) "normal"
     difficulty (low | normal | high) "normal"
     status (proposed | approved | incorporated | validated) #IMPLIED
     stability (low | normal | high) "normal"
     version CDATA #IMPLIED
     xmlns:xlink CDATA #FIXED "http://www.w3c.org/1999/xlink"
     xlink:type (extended) #FIXED "extended"
>
<!ELEMENT source EMPTY>
<!ATTLIST source
     xlink:type (locator) #FIXED "locator"
     xlink:href CDATA #REQUIRED
     xlink:label NMTOKEN #IMPLIED
>
<!ELEMENT sink EMPTY>
<!ATTLIST sink
     xlink:type (locator) #FIXED "locator"
     xlink:href CDATA #REQUIRED
     xlink:label NMTOKEN #IMPLIED
>
<!-- ========== -->
<!-- Trace arcs -->
<!-- ========== -->
<!ELEMENT parent-type EMPTY>
<!-- Source: <taxonomy> Sink: <taxonomy> -->
<!-- The sink is the parent type of the source -->
<!ATTLIST parent-type
     xlink:type (arc) #FIXED "arc"
     xlink:from NMTOKEN #REQUIRED
     xlink:to NMTOKEN #REQUIRED
>
<!ELEMENT instantiation EMPTY>
<!-- Source: * except <taxonomy> Sink: <taxonomy> -->
<!-- The source is is an instance of the sink -->
<!ATTLIST instantiation
```

```
     xlink:type (arc) #FIXED "arc"
     xlink:from NMTOKEN #REQUIRED
     xlink:to NMTOKEN #REQUIRED
>
<!ELEMENT contextual-scope EMPTY>
<!-- Source: <context> Sink: * -->
<!-- Establishes a common context for an object or group of objects -->
<!ATTLIST contextual-scope
     xlink:type (arc) #FIXED "arc"
     xlink:from NMTOKEN #REQUIRED
     xlink:to NMTOKEN #REQUIRED
>
<!ELEMENT baseline EMPTY>
<!-- Source: <project> Sink: * -->
<!-- a baseline for the source project -->
<!ATTLIST baseline
     time CDATA #IMPLIED
     date CDATA #IMPLIED
     xlink:type (arc) #FIXED "arc"
     xlink:from NMTOKEN #REQUIRED
     xlink:to NMTOKEN #REQUIRED
>
<!ELEMENT subproject EMPTY>
<!-- Source: <project> Sink: <project> -->
<!-- The sink is a subproject of the source -->
<!ATTLIST subproject
     xlink:type (arc) #FIXED "arc"
     xlink:from NMTOKEN #REQUIRED
     xlink:to NMTOKEN #REQUIRED
>
<!ELEMENT project-data EMPTY>
<!-- Source: <project> Sink: * except <project> -->
<!-- The sink holds all data that is relevant to the source project -->
<!ATTLIST project-data
     xlink:type (arc) #FIXED "arc"
     xlink:from NMTOKEN #REQUIRED
     xlink:to NMTOKEN #REQUIRED
>
<!-- Requirement and use-case traces -->
<!ELEMENT refinement EMPTY>
<!-- Source: * Sink: Same type as source -->
<!-- The sink is a refinement of the source -->
<!ATTLIST refinement
     xlink:type (arc) #FIXED "arc"
     xlink:from NMTOKEN #REQUIRED
     xlink:to NMTOKEN #REQUIRED
>
<!ELEMENT supercede EMPTY>
<!-- Source: * Sink: Same type as source -->
<!-- The sink supercedes the source -->
<!ATTLIST supercede
     xlink:type (arc) #FIXED "arc"
     xlink:from NMTOKEN #REQUIRED
     xlink:to NMTOKEN #REQUIRED
>
<!ELEMENT parent EMPTY>
<!-- Source: * Sink: Same type as source -->
<!-- The sink is the parent the source -->
<!ATTLIST parent
     xlink:type (arc) #FIXED "arc"
     xlink:from NMTOKEN #REQUIRED
     xlink:to NMTOKEN #REQUIRED
>
<!ELEMENT conflict EMPTY>
<!-- Source: * Sink: Same type as source -->
<!-- The sink and source are in conflict -->
```

```
<!ATTLIST conflict
     xlink:type (arc) #FIXED "arc"
     xlink:from NMTOKEN #REQUIRED
     xlink:to NMTOKEN #REQUIRED
>
<!ELEMENT forward-from EMPTY>
<!-- Source: <requirement> or <use-case> Sink: second class object -->
<!-- The sink object traces forward from the source object -->
<!ATTLIST forward-from
     xlink:type (arc) #FIXED "arc"
     xlink:from NMTOKEN #REQUIRED
     xlink:to NMTOKEN #REQUIRED
>
<!ELEMENT backward-to EMPTY>
<!-- Source: second class object Sink: <requirement> or <use-case> -->
<!-- The source object traces backward to the sink object -->
<!ATTLIST backward-to
     xlink:type (arc) #FIXED "arc"
     xlink:from NMTOKEN #REQUIRED
     xlink:to NMTOKEN #REQUIRED
>
<!ELEMENT backward-from EMPTY>
<!-- Source: <requirement> or <use-case> Sink: second class object -->
<!-- The sink object traces backward from the source object -->
<!ATTLIST backward-from
     xlink:type (arc) #FIXED "arc"
     xlink:from NMTOKEN #REQUIRED
     xlink:to NMTOKEN #REQUIRED
>
<!ELEMENT forward-to EMPTY>
<!-- Source: second class object  Sink: requirement or use-case -->
<!-- The source object traces forward-to the sink object -->
<!ATTLIST forward-to
     xlink:type (arc) #FIXED "arc"
     xlink:from NMTOKEN #REQUIRED
     xlink:to NMTOKEN #REQUIRED
>
<!-- Stakeholder related traces -->
<!ELEMENT stakeholder-view (administration?, origination?, responsibility?,
shcomment?, access?)>
<!-- Source: <stakeholder>,<taxonomy> or <group> Sink: * -->
<!-- The source is linked with the view defined by the sink. -->
<!-- If the source is a <taxonomy>, it must describe a type of stakeholder.
-->
<!-- If the source is a <group>, it must only contain objects of type -->
<!-- <stakeholder> or <taxonomy> (again describing a stakeholder type). -->
<!-- The rights that the source has in context with the sink are -->
<!-- defined via subelements (see below) -->
<!ATTLIST stakeholder-view
     xlink:type (arc) #FIXED "arc"
     xlink:from NMTOKEN #REQUIRED
     xlink:to NMTOKEN #REQUIRED
>
<!ELEMENT administration EMPTY>
<!-- The source administrates the sink -->
<!-- This will probably mean that he can give away access rights to other -->
<!-- stakeholders -->
<!ELEMENT origination EMPTY>
<!-- The source is the originator of the sink -->
<!ELEMENT responsibility EMPTY>
<!-- The source is responsible for the sink -->
<!ELEMENT shcomment (#PCDATA | term)*>
<!-- A comment from the source stakholder about the sink data -->
<!ELEMENT access EMPTY>
<!ATTLIST access
     rights (none | read | readandwrite) "none"
```

```
>
<!-- The access rights of the source in context with the sink -->
<!-- Trace to trace -->
<!ELEMENT trace-to-trace EMPTY>
<!-- Source: <trace> Sink: <trace> -->
<!-- Provides traceability from source trace to sink trace -->
<!ATTLIST trace-to-trace
     xlink:type (arc) #FIXED "arc"
     xlink:from NMTOKEN #REQUIRED
     xlink:to NMTOKEN #REQUIRED
>
<!-- General conflict -->
<!ELEMENT general-conflict EMPTY>
<!-- Source: * Sink: * -->
<!-- The source and sink are in conflict -->
<!ATTLIST general-conflict
     xlink:type (arc) #FIXED "arc"
     xlink:from NMTOKEN #REQUIRED
     xlink:to NMTOKEN #REQUIRED
>
<!-- lexicon traces -->
<!ELEMENT lexref EMPTY>
<!-- Source: <term> Sink: <lexicon> -->
<!-- The source is defined in the sink -->
<!ATTLIST lexref
     xlink:type (arc) #FIXED "arc"
     xlink:from NMTOKEN #REQUIRED
     xlink:to NMTOKEN #REQUIRED
>
<!ELEMENT see-also EMPTY>
<!-- Source: <lexicon> Sink: <lexicon> -->
<!-- The sink lexicon entry is relevant to the source entry -->
<!ATTLIST see-also
     xlink:type (arc) #FIXED "arc"
     xlink:from NMTOKEN #REQUIRED
     xlink:to NMTOKEN #REQUIRED
>
<!-- universal trace -->
<!ELEMENT universal-trace EMPTY>
<!-- Source: * Sink: * -->
<!-- Provides traceability between any two objects -->
<!ATTLIST universal-trace
     xlink:type (arc) #FIXED "arc"
     xlink:from NMTOKEN #REQUIRED
     xlink:to NMTOKEN #REQUIRED
>
```

Bab 5

DESIGN PATTERNS

Design patterns adalah unsur-unsur rancangan yang seringkali muncul pada berbagai sistem yang berbeda. Setiap kemunculan ini menguji pattern tersebut di berbagai situasi. Semakin terujinya suatu unsur rancangan berarti semakin matangnya unsur tersebut – sedemikian sehingga beberapa dapat dikatakan sebagai *best practices* dalam perancangan sistem.

Istilah *design patterns* dimulai di bidang perancangan bangunan oleh Christopher Alexander. Dalam bukunya *A Pattern Language* [Alex77], ia menerangkan pola-pola yang terdapat di dalam berbagai rancangan arsitektur bangunan. Arti *design pattern* diterangkannya dalam kalimat berikut:

> *Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice [Alex77].*

Dengan kata lain, *design pattern* adalah gambaran formal dari suatu masalah berikut pemecahannya. Masing-masing *pattern* harus mempunyai nama yang sederhana dan deskriptif yang dapat langsung digunakan untuk mengacu pada pola tersebut. Sebuah *pattern* harus mendokumentasikan permasalahan, pemecahan, serta akibat-akibat penggunaannya [Wall00].

Dalam bidang *software engineering*, debut design patterns diawali oleh Erich Gamma, Richard Helm, Ralph Johnson dan John Vlissides dalam bukunya *Design Patterns: Elements of Reusable Object-Oriented Language* [Gamm95]. Setelah buku ini, banyak praktisi dan akademisi lainnya yang mulai mengkategorikan pola-pola rancangan lainnya dalam berbagai *software-intensive systems* [Sun01a] [Wall00].

Pembahasan design patterns tentu tidak terlepas dari pembahasan *arsitektur software* secara keseluruhan. Sebagai *best practices* dari perancangan software, *design patterns* muncul sebagai komponen-komponen dari arsitektur *software*. Dalam bab ini juga

dibahas dua arsitektur software dalam keterhubungannya dengan *design patterns*: *model-view-controller* dan diikuti dengan salah satu variannya yaitu *separable model architecture*.

5.1 Gang of Four Patterns

Pola-pola rancangan yang terdapat pada [Gamm95] secara kolektif disebut *Gang of Four Patterns* [Coop98] – bukan karena mereka yang membuatnya, namun empat sekawan ini membuat daftar pola-pola rancangan yang telah cukup stabil dan digunakan di dalam berbagai sistem berorientasi obyek. [Gamm95] membagi katalog *design patterns* mereka dalam tiga kategori: *creational*, *structural*, dan *behavioral*.

Creational patterns berhubungan dengan penciptaan obyek. Pola-pola ini berkisar seputar obyek mana yang diciptakan, siapa yang menciptakannya, serta berapa banyak obyek diciptakan.

Structural patterns berhubungan dengan struktur statis obyek dan kelas. Pola-pola dalam *structural patterns* dapat dilihat pada saat program di-compile melalui struktur *inheritance*, properties, serta agregasi obyek-obyek.

Behavioral patterns lebih berkenaan terhadap perilaku *run-time* program. Pola-pola ini berkaitan dengan algoritma serta interaksi antar obyek saat program berjalan. Penekanan *behavioral patterns* lebih pada komposisi obyek ketimbang *inheritance*.

Sedangkan masing-masing *pattern* dibahas oleh [Gamm95] sebagai kesatuan dari:

- **Name** – Merupakan nama yang diberikan pada pola ini.
- **Intent** – Merupakan pernyataan ringkas yang memberikan permasalahan yang terjadi serta maksud yang hendak dicapai.
- **Also Known As** – Berbagai alias untuk pola ini, jika ada.
- **Motivation** – Sebuah skenario yang menerangkan sebuah permasalahan rancangan dan bagaimana pola ini dapat memecahkannya.
- **Applicability** – Berbagai situasi di mana pola ini dapat diterapkan.
- **Structure** – Sebuah gambar yang menerangkan hubungan kelas dan obyek dalam pattern ini.

- **Participants** – Berbagai kelas dan/atau obyek yang turut serta dalam pola ini beserta peranannya.
- **Collaborations** – Bagaimana kerja sama dari para peserta untuk melaksanakan peranannya masing-masing.
- **Consequences** – Bagaimana pola ini mencapai tujuannya serta kompromi (*trade-off*) yang harus dilakukan dalam penerapannya.
- **Implementation** – Petunjuk, peringatan, serta berbagai teknik yang digunakan dalam penerapan pola ini.
- **Sample Code** – Contoh program yang mengilustrasikan penerapan pola ini.
- **Known Uses** – Contoh-contoh dari penggunaan pola ini pada sebuah sistem sungguhan.
- **Related Patterns** – Pola-pola lain yang berhubungan dengan pola ini.

Dalam bagian ini, pembahasan *design pattern* diringkas dalam bentuk tabular yang diusulkan oleh [Gust99]. Teks yang menjadi keterangan niatan (*intent*) dari masing-masing *pattern* diurai dalam bentuk kolom-kolom yang memberikan tambahan struktur kepada gambaran *pattern* tersebut. Penguraian ini juga memungkinkan keseluruhan pattern yang digunakan dapat dilihat dalam selayang pandang. Tidak semua *pattern* dibahas di sini; namun hanyalah pola-pola yang digunakan di *Rambutan*. Pembaca yang tertarik dapat mengacu pada [Gamm95], [Coop98], dan [Sun01a] untuk pembahasan yang lebih lengkap.

5.1.1 *Creational Patterns*

Niatan dari masing-masing pola dalam *creational patterns* diuraikan menjadi lima kolom:

- **Penciptaan** – apakah membuat atau membatasi pembuatan suatu obyek.
- **Kesatuan** – obyek apa yang diciptakan.
- **Tindakan** – cara yang diambil dalam menciptakan obyek.
- **Tekanan** – batasan-batasan yang ada dalam penciptaan obyek.
- **Informasi Tambahan** – keterangan yang memperjelas niatan dari pola yang bersangkutan.

Tabel 5.1 *Creational Patterns*

Nama Pattern	Penciptaan	Kesatuan	Tindakan	Tekanan	Informasi Tambahan
Abstract Factory	Menciptakan	sekumpulan obyek yang berhubungan atau saling tergantung	dengan menyediakan suatu antarmuka untuk melakukannya	tanpa menentukan suatu kelas yang nyata.	
Factory Method	Menciptakan	obyek-obyek	dengan menyediakan suatu antarmuka yang memungkinkan subkelas-subkelas untuk memutuskan obyek mana yang akan diciptakan.		Factory method memungkinkan sebuah kelas untuk menunda pembuatan keberadaan obyek kepada subkelas-subkelas.
Singleton	Jangan menciptakan	lebih dari satu keberadaan dari suatu kelas	dengan menyediakan suatu titik akes global kepadanya.		

5.1.2 *Structural Patterns*

Niatan dari masing-masing pola dalam *structural patterns* diuraikan menjadi empat kolom:

- **Tindakan** untuk diterapkan – apa yang dilakukan oleh pola yang bersangkutan.
- **Kesatuan** – perihal yang terkait dengan pola ini.
- **Akibat** – Dampak yang terjadi dari penerapan pola ini.
- **Hasil** – Apa saja yang didapatkan dari penerapan pola ini.

Tabel 5.2 *Structural Patterns*

Nama Pattern	Tindakan untuk diterapkan	Kesatuan	Akibat	Hasil
Adapter	Mengubah	antarmuka sebuah kelas	kepada antarmuka kelas lain yang diharapkan klien.	Adapter memungkinkan kelas-kelas untuk bekerja sama yang sebaliknya tidak mungkin karena antarmuka-antarmuka yang tidak cocok.

Tabel 5.2 *Structural Patterns*

Nama Pattern	Tindakan untuk diterapkan	Kesatuan	Akibat	Hasil
Bridge	Memisahkan satu dari yang lainnya	sebuah abstraksi dan penerapannya		sehingga keduanya dapat berbeda dan tidak saling tergantung.
Composite	Menyusun	obyek-obyek	ke dalam struktur pohon untuk mewakili hirarki seluruh-sebagian.	Composite memungkinkan klien-klien untuk memperlakukan obyek tunggal maupun komposisi dengan cara yang sama.
Decorator	Secara dinamis memberikan tambahan tanggungjawab-tanggungjawab kepada	sebuah obyek.		Decorator memberikan mekanisme perluasan yang mudah disesuaikan.
Façade	Memberikan sebuah kesatuan antarmuka kepada	suatu himpunan antarmuka-antarmuka di dalam sebuah subsistem.		Façade menetapkan antarmuka dengan tingkat yang lebih tinggi yang mempermudah penggunaan sebuah subsistem.
Flyweight	Penggunaan secara bersama-sama dari	obyek-obyek.		Flyweight memungkinkan sejumlah besar dari obyek-obyek yang berukuran kecil untuk ditangani secara tepatguna.

5.1.3 *Behavioral Patterns*

Niatan dari masing-masing pola dalam *behavioral patterns* diuraikan menjadi tiga kolom:

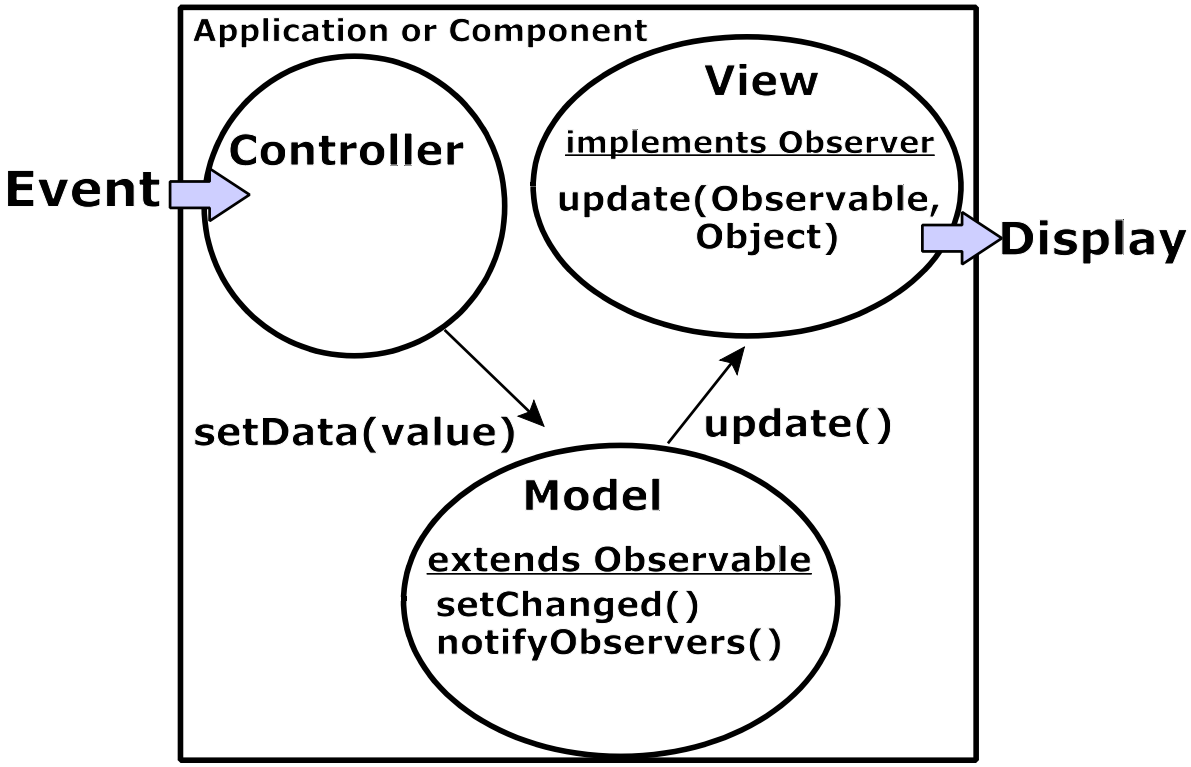
- **Tujuan** – Hal yang ingin dicapai dari penerapan pola.
- **Rangkaian tindakan** – cara pola ini diterapkan.
- **Hasil** – Apa saja yang didapatkan dari penerapan pola ini.

Tabel 5.3 *Behavioral Patterns*

Nama pattern	**Tujuan**	**Rangkaian tindakan / langkah-langkah**	**Hasil**
Command	Memparameterkan klien dengan permintaan-permintaan yang berbeda	dengan cara membungkus sebuah permintaan di dalam sebuah obyek.	Command memungkinkan Anda untuk mengantrikan atau mencatat permintaan-permintaan dan mendukung operasi yang dapat dibalikkan.
Iterator	Mengakses unsur-unsur dari sekumpulan obyek secara berurutan.		Iterator memberikan suatu jalan untuk mengakses unsur-unsur tanpa menyingkap implementasi yang mendasarinya.
Observer	Semua obyek yang bergantung diberitahukan serta diperbaharui ketika suatu obyek berganti keadaan	dengan menetapkan hubungan satu-ke-banyak antara obyek-obyek.	
Memento	Menangkap dan mengeluarkan keadaan dalam suatu obyek sehingga keadaan ini dapat dikembalikan belakangan.		Memento melakukan hal ini tanpa melanggar pembungkusan.
Strategy	Membuat algoritma-algoritma dari suatu keluarga algoritma dapat dipertukarkan	dengan cara membungkus masing-masing algoritma.	Strategy memungkinkan algoritma berubah-ubah terlepas dari klien-klien yang menggunakannya.
Template Method	Menunda langkah-langkah suatu algoritma ke subkelas-subkelas	dengan cara menetapkan rangka dari suatu algoritma di satu operasi.	Template Method memungkinkan subkelas-subkelas untuk menetapkan-ulang langkah-langkah tertentu dari suatu algoritma tanpa mengubah susunan algoritma tersebut.

5.2 Model-View-Controller (MVC)

Arsitektur MVC memisahkan suatu aplikasi interaktif ke dalam tiga kategori kelas: *model*, *view*, dan *controller*. Kelas-kelas yang termasuk dalam *model* merepresentasikan data yang diolah aplikasi tersebut. Sedangkan kelas-kelas di dalam *view* dan *controller* adalah penerapan antarmuka pemakai; *view* merupakan tampilan aplikasi sedangkan *controller* menerima masukan dari pemakai.



Gambar 5.1 Arsitektur MVC dan hubungannya dengan Java [Stan97]

Dipopulerkan oleh SmallTalk-80 [Burb92], arsitektur MVC beserta varian-variannya masih digunakan sampai saat ini. Arsitektur Doc/View yang ada pada *framework* Borland ObjectWindows [Borl93] bisa dikatakan cukup mirip dengan MVC. Penerapannya juga tidak terbatas pada aplikasi GUI yang menjadi tujuan awalnya; *framework* seperti Jakarta Struts menerapkan arsitektur MVC pada lingkungan berbasis web.

Umumnya suatu kelas *controller* dirancang untuk digunakan bersama dengan kelas *view* tertentu, karena sulit untuk membuat *controller* umum yang tidak tergantung pada *view* tertentu [Fowl02]. Karena itu hubungan antara *view* dan *controller* bisa dikatakan *tightly coupled*.

Bertugas menerima masukan dari pemakai, kelas-kelas *controller* mengubah data di dalam *model* berdasarkan masukan ini. Perubahan data dilakukan dengan memanggil *method-method* tertentu pada kelas *model* – agar tidak melanggar enkapsulasi. Perubahan data di dalam *model* menyebabkan *view* diberitahukan akan adanya perubahan ini.

Di antara *view* dengan *model* terdapat hubungan *publish/subscribe*. Sebuah *model* memiliki daftar *view* yang *subscribe* kepadanya. Pada saat data di dalam *model* berubah, semua *view* diberitahukan akan perubahan ini. Dengan diterimanya pemberitahuan, *view* akan secara otomatis memperbaharui tampilannya untuk merefleksikan perubahan ini. [Gamm95] menyebut hubungan antara *view* dengan *model* ini sebagai penerapan dari *Observer pattern*.

Java 2 Standard Edition menyediakan dua kelas untuk mempermudah penerapan hubungan *publish/subscribe* [Sun00a]. Di dalam *package* `java.util` terdapat pasangan `class Observable` dan `interface Observer`. Suatu kelas *model* dapat di-*extend* dari `Observable` sedangkan *view* dapat meng-*implement* `Observer`. Suatu *view* dapat mendaftarkan dirinya untuk menerima pemberitahuan dari suatu *model* melalui method `Observable.addObserver()`. Apabila data berubah, *model* yang bersangkutan perlu memanggil *method* `setChanged()` untuk menandakan perubahan ini. Pengumuman perubahan data dilakukan dengan memanggil *method* `notifyObservers()`. Kedua method tersebut ada di kelas `Observable`; dengan method `setChanged()` bersifat `protected` sehingga hanya dapat dipanggil oleh subkelasnya.

5.3 Separable Model Architecture

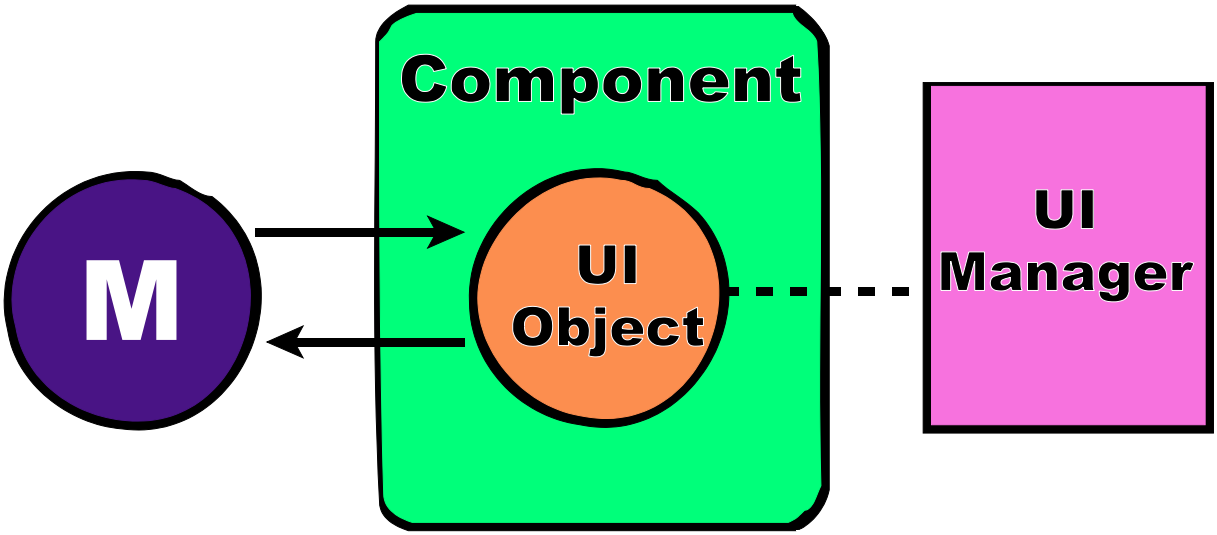
Swing adalah nama panggilan ("*code name*") untuk sekumpulan kelas antarmuka-pemakai grafis (GUI) yang tersedia pada *platform Java 2 Standard Edition*. Pada *release* sebelumnya platform ini menyediakan yang disebut *Abstract Windowing Toolkit* (AWT) untuk membangun antarmuka pemakai. Walaupun AWT masih disediakan, aplikasi-aplikasi baru untuk Java 2 disarankan untuk menggunakan Swing ketimbang AWT [Sun03a].

Perbedaan yang paling mendasar antara Swing dengan AWT yaitu Swing merupakan pustaka yang *pure Java* [Sun00a]. Keseluruhan logika Swing dijalankan oleh JVM tanpa memerlukan *native code*. Sedangkan pada AWT, fungsionalitas setiap komponen antarmuka-pemakai sebenarnya didelegasikan kepada komponen-komponen *native* dari *host platform*. Namun Swing menggunakan beberapa komponen AWT untuk interaksi-interaksi primitifnya.

Suatu aplikasi dikatakan *pure Java* apabila sama sekali tidak menggunakan *native code* – yaitu kode-kode program yang *pribumi* terhadap mesin yang menjalankannya

(*host machine/platform*). Program-program Java mendapatkan kemampuan *cross-platform* karena program-program tersebut tidak di-*compile* ke bahasa mesin yang menjadi tujuan dijalankannya program. Namun program Java di-*compile* ke suatu bahasa mesin maya yang disebut *Java bytecode*. Pada setiap mesin yang akan menjalankan program Java dipasang suatu perangkat lunak yang dinamakan *Java Virtual Machine* (JVM). JVM ini secara *software* bertindak sebagai mesin maya yang menerjemahkan *bytecode* menjadi *machine code* pada saat program berjalan.

Tight coupling yang terjadi di antara view dan controller pada arsitektur MVC berakibat tidak praktisnya pembuatan komponen-komponen umum antarmuka-pemakai berbasis murni MVC. Hal ini ditemukan pada saat pengembangan arsitektur Swing sehingga para pengembangnya memutuskan untuk tidak mengikuti arsitektur MVC secara penuh [Fowl02].



Gambar 5.2 Separable Model Architecture [Fowl02]

Arsitektur yang digunakan Swing dinamakan *Separable Model Architecture* (SMA). Berakar dari MVC, arsitektur ini menggabungkan *view* dan *controller* menjadi satu obyek. Sedangkan *model* masih merupakan obyek tersendiri. Perbedaan yang lain yaitu penampilan (*painting*) komponen-komponen Swing tidak dilakukan oleh komponen itu sendiri, namun didelegasikan ke suatu *UI Manager*.

Karena tidak tergantung pada *problem domain* tertentu, *model* yang digunakan lebih disesuaikan pada kebutuhan komponen yang bersangkutan. Misalkan komponen `JScrollBar` dan `JSlider` (dua komponen *scroll bar* dan *slider*, secara berurutan, dari Swing) menggunakan jenis model yang sama, yaitu *Bounded Range Model* [Sun00a]

karena kedua komponen ini menampilkan data yang sejenis, yaitu suatu nilai yang terletak di antara dua batas (*bounds*).

Obyek *model* di dalam Swing terdiri atas dua kategori, yaitu *GUI-state model* dan *Application-data model* [Fowl02].

GUI-state model adalah sekumpulan `interface` yang merepresentasikan keadaan penampilan suatu komponen antarmuka grafis. Misalkan apakah suatu tombol sedang ditekan atau tidak, *item* mana saja di dalam sebuah *list* yang sedang dipilih, ataupun posisi sebuah *slider*. Suatu aplikasi dapat saja menggunakan suatu *GUI-state model* sendiri. Namun umumnya interaksi dengan *model* tidak diperlukan; aplikasi dapat saja memanggil method dari komponen yang bersangkutan tanpa menyentuh *model* yang digunakannya. Komponen akan mendelegasikan permintaan tersebut ke *model* yang dimilikinya.

Application-data model adalah sekumpulan `interface` yang mewakili data yang terutama berarti pada konteks aplikasi yang bersangkutan (berhubungan erat dengan *problem domain* aplikasi). Penggunaan *application-data model* sangat disarankan untuk aplikasi-aplikasi yang sangat berhubungan erat dengan data [Fowl02]. Interaksi yang erat dengan data model lebih disarankan untuk beberapa komponen yang sangat *data-centric* seperti `JTable` dan `JTree`.

Tabel berikut menunjukkan korelasi antara komponen-komponen Swing, interface dari model yang digunakan serta tipe dari model yang diinginkan [Fowl02].

Tabel 5.4 Korelasi komponen-komponen Swing dengan *data model* yang digunakannya [Fowl02]

Component	Model Interface	Model Type
`JButton`	`ButtonModel`	GUI
`JToggleButton`	`ButtonModel`	GUI/data
`JCheckBox`	`ButtonModel`	GUI/data
`JRadioButton`	`ButtonModel`	GUI/data
`JMenu`	`ButtonModel`	GUI
`JMenuItem`	`ButtonModel`	GUI
`JCheckBoxMenuItem`	`ButtonModel`	GUI/data
`JRadioButtonMenuItem`	`ButtonModel`	GUI/data
`JComboBox`	`ComboBoxModel`	data
`JProgressBar`	`BoundedRangeModel`	GUI/data
`JScrollBar`	`BoundedRangeModel`	GUI/data

Component	Model Interface	Model Type
JSlider	BoundedRangeModel	GUI/data
JTabbedPane	SingleSelectionModel	GUI
JList	ListModel	data
JList	ListSelectionModel	GUI
JTable	TableModel	data
JTable	TableColumnModel	GUI
JTree	TreeModel	data
JTree	TreeSelectionModel	GUI
JEditorPane	Document	data
JTextPane	Document	data
JTextArea	Document	data
JTextField	Document	data
JPasswordField	Document	data

Salah satu kelebihan Swing bila dibandingkan dengan pustaka antarmuka pemakai lainnya, yaitu masing-masing komponen Swing mendelegasikan penampilannya (*painting*) pada kelas-kelas lain. Pendelegasian ini memungkinkan Swing untuk mempunyai penampilan yang berbeda-beda dan dapat diubah pada saat program dijalankan [Fowl02]. Fungsionalitas ini disebut *pluggable look-and-feel* (PL&F) dan diatur oleh sebuah *UIManager*. Untuk masing-masing komponen, *painting* didelegasikan lebih lanjut ke kelas-kelas *look-and-feel*. Kelas-kelas inilah yang menggambar setiap komponen ke layar. PL&F memungkinkan suatu program Swing mempunyai tampilan yang berbeda-beda.

Pada Java Development Kit 1.3 untuk *platform* Windows, ada tiga *look-and-feel* yang disediakan:

- **Metal** – Juga disebut *cross-platform look-and-feel*, ini merupakan *look-and-feel default* yang diambil apabila tidak dipilih lainnya.
- **Motif** – Meniru penampilan aplikasi-aplikasi Motif yang umum terdapat pada *platform* Unix.
- **Windows** – Meniru penampilan Microsoft Windows.

Selain ketiga *look-and-feel* di atas, beberapa vendor pihak ketiga juga menyediakan *look-and-feel* alternatif untuk digunakan sebuah aplikasi Swing.

Bab 6

ANALISIS DAN PERANCANGAN

Bab ini membahas analisa yang dilakukan serta rancangan awal untuk sistem yang akan dibuat. Analisa meliputi *user story*, serangkaian *requirement* serta berbagai batasan (*constraints*) yang ditentukan. Perancangan yang dibuat termasuk ide umum rancangan, rencana *data model*, serta gambaran kasar antarmuka pemakai.

Analisis dilakukan atas dasar *user story* yang menggambarkan sebuah contoh situasi di mana Rambutan akan digunakan. Cerita ini menuturkan seorang *system analyst* yang sedang mewawancarai dua orang *stakeholder* untuk membuat *draft requirement* pada saat *project initiation* (pra-kontrak). Sang *analyst* berasal dari suatu perusahaan perangkat lunak yang berbasis proyek. Sedangkan dua orang *stakeholder* yang diwawancarainya bekerja pada sebuah perusahaan *dotcom* yang tidak mempunyai *IT department* sendiri – mayoritas *new development* dikontrakkan ke luar.

Bagian berikutnya membahas berbagai spesifikasi kebutuhan (*requirement*) untuk dipenuhi oleh *software* yang akan dibuat. Umumnya rangkaian *requirement* ini disarikan dari *user story* pada bagian sebelumnya.

Spesifikasi kebutuhan diikuti oleh batasan-batasan (*constraints*) yang dibebankan pada rancangan hasil akhir dari sistem. Batasan-batasan ini terutama didorong oleh pertimbangan-pertimbangan politis maupun bisnis untuk kebaikan pengembangan sistem selanjutnya yang bertolak dari proyek ini.

Setelah itu akan dibahas rancangan-rancangan awal yang dibuat sebagai modal dasar untuk implementasi Rambutan. Sebagai sebuah *one-person prototyping project*, berbagai spesifikasi ini sangat *high-level*; perincian dari rancangan-rancangan ini dibuat *on-the-fly* secara iteratif sejalan dengan proses *coding* seperti cara kerja seorang *hacker* pada umumnya.

6.1 *User Story*

Latar belakang cerita ini adalah pada dasawarsa pertama dari abad ke-21, di sebuah kota metropolitan yang sibuk. Karena seluruh pelaku adalah para pekerja muda – atau terkadang disebut sebagai *Generation-X* [Robb01] – maka bahasa percakapan tetap dipertahankan dalam bentuk informal. Nama, merek dagang, maupun perusahaan sebagian besar adalah fiktif – namun beberapa nama produk nyata digunakan untuk menambah realisme cerita, tetapi tidak untuk mempromosikan produk tersebut. Berikut ini adalah *user story* dalam bentuk narasi dialog.

Pada suatu pagi yang cerah, Rinoa, seorang *system analyst*, sedang bekerja di kantornya. Seperti biasanya, ia memulai hari kerjanya di kantor pada jam delapan dengan segelas *cappucino* beserta sederet *e-mail* dan IT *newsletter*. Selagi *surfing*[6] di dalam *inbox*-nya, tiba-tiba telepon berdering...

Operator: "Bu Rinoa, panggilan dari Binaraga di line satu."

Rinoa: "Saya terima." *(Menjawab telepon di line satu...)*

Telepon: "Halo, Rinoa? Masih inget gua?"

Rinoa: "Elu Vi, ada apa?"

Rupanya yang menelepon adalah kenalannya, Vivian, R & D Manager dari PT Binaraga Maya Persada. Perusahaan ini merupakan klien lama dari PT Arcle Technologies, tempat Rinoa bekerja.

Vivian: "Gini nih... gue – eh, kami – pengen bikin website baru.... Tempat lo bisa handle ga?"

Rinoa: "Web apaan nih?"

Vivian: "Ga jauh lah, e-commerce juga... Kali ini kita mau jualan kamera, namanya kamerafoto dot com.... Lo bisa ke tempat gua ga?"

Rinoa: "Hmm..." *(Memeriksa jadwal pada aplikasi Palm Desktop di komputernya)* "Gimana kalo besok after lunch?"

Vivian: "Besok jam 13:30, OK?"

---

[6] Apabila kata *skimming* yang digunakan – walaupun lebih tepat – nilai rasa yang diberikan menjadi berkurang sehingga tidak mencerminkan keasyikan yang dialami.

Rinoa: *(Membuat appointment di Palm Desktop untuk keesokan harinya)* "OK."

Rinoa kemudian menjalankan program Rambutan Desktop untuk membuat dokumen *System Requirements Specification* (SRS). Ia membuat dokumen SRS baru, kemudian menambahkan proyek "kamerafoto" ke dalamnya. Di dalam *project description* ditulisnya 'e-commerce kamera buat binaraga.' Ia juga menambahkan seorang *stakeholder*, 'Vivian'. Kemudian ia mengambil PDA-nya, sebuah Clié model tidak terlalu baru, dan melakukan sinkronisasi data dengan komputer desktop di kantornya. Dengan sinkronisasi ini, dokumen SRS yang baru ia buat disalin ke dalam PDA-nya.

Keesokan harinya, Rinoa datang ke kantor Binaraga….

Vivian: "Hai… udah lama ga ketemu nih, gimana kabar Cowo Lo… eh, Bos…"

Rinoa: "Hus, jangan diomongin di sini… bisa runyam."

Vivian: "Iya deh, kita ke meeting room yuk."

Mereka berdua berjalan ke suatu ruangan tertutup di dalam kantor. Walaupun tertutup, sekat ruangan itu adalah kaca kedap suara – sehingga orang-orang kantor masih dapat melihat ke dalam dan sebaliknya.

Rinoa: *(Sambil mengeluarkan PDA-nya dan mengaktifkan program Rambutan)* "So, kita bisa mulai?"

Vivian: *(Penasaran, sebagai pemilik PDA juga)* "Program baru ya?"

Rinoa: "Iya nih, buat ngumpulin requirement… Bos nyuruh gua nyoba, jadi kelinci percobaan nih, mentang-mentang gua…"

Vivian: *(Menyindir, kemudian batuk palsu)* "Orang kepercayaannya? (ehem)"

Rinoa: "Ya sudah lah, back to business. Lo mo bikin website e-commerce? Jual kamera?"

Vivian: "Iya, ekspansi, biasa…."

Rinoa: "Siapa aja yang ikutan proyek ini?"

Vivian: "Gua, Bos gua, Sugi…"

Rinoa: "Sugianto?"

Vivian: "Iya, sama Kristin, anak baru di marketing – nanti gua kenalin."

Rinoa: *(Menambah dua stakeholder baru selain Vivian yang telah dicatat kemarin: Sugianto dan Kristin)* "Udah? Ada lagi?"

Vivian: "Ya nanti paling anak-anak CS sama marketing, tapi itu Kristin yang urus."

Rinoa: *(Mencatat stakeholder: customer service dan marketing)* "T'rus mo nampilin apa aja?"

Vivian: "Produk, shopping cart – biasa lah, cuma kita mau ada sejarah pesanan buat business intelligence."

Rinoa: *(Mencatat requirement: 'shopping cart', 'daftar produk', dan 'sejarah pesanan'. Untuk requirement 'sejarah pesanan', ia menambahkan rationale 'business intelligence').*

Vivian: "Lalu nanti mau bikin komunitas penggemar fotografi, jadi kudu ada tampilan info-info atau artikel yang bisa di-update."

Rinoa: *(Mencatat requirement, 'tampilan artikel' beserta rationale-nya: 'untuk membuat komunitas').* "Pake content management?"

Vivian: "Iya lah, lo tau kan kita ga punya IT staff khusus.. Jadi artikel, produk, [dan] pesanannya kudu bisa diurus dari web... dan bisa gampang di-update."

Sementara mereka berbicara, Vivian melihat seseorang berjalan melewati ruang pertemuan. Ia kemudian bangkit, keluar, dan memanggil orang tersebut. Menjawab panggilan Vivian, orang itu ikut masuk kembali ke ruang pertemuan bersamanya.

Vivian: "Ini Kristin, anak yang gua bilang tadi."

Kemudian Kristin dan Rinoa saling berkenalan.

Rinoa: "Bisa minta nomor telepon?"

Kristin: *(Melihat Clié milik Rinoa)* "Gua beam ya... Bentar deh...."

Kristin kemudian mengeluarkan telepon genggamnya, sebuah 7650, *standard issue* bagi pegawai *marketing* Binaraga. Rinoa mengarahkan PDA-nya pada telepon genggam

Kristin dan sebaliknya. Mereka lalu bertukaran *virtual business card* (*VCard*) lewat jalur inframerah.

Kristin: "Lagi pada ngapain nih?"

Vivian: "Ini nih, lagi bicarain kamerafoto... Rinoa itu system analyst dari Arcle Technologies, yang bakal ngebuatin web baru kita itu. Lo bisa bantuin kan?"

Kristin: "Oh gitu... bisa lah, jelas udah tugas gua." *(Kemudian duduk untuk mengikuti rapat).*

Vivian: "Sampai mana tadi... Oh ya, bisa integrasi dengan web yang lama ga? Terus pake PHP dan MySQL biar bisa ditaruh di webhosting yang sama."

Rinoa: *(Mencatat masalah: 'integrasi dgn legacy sys'. Kemudian mencatat batasan 'PHP & MySQL'. Batasan ini diberi alasan, 'webhosting yg sama dgn legacy sys').*

Kristin: "Bisa dibuat untuk nyaring Pak Ogah dan Paman Gober ga?"

Rinoa: "Pak Ogah? Paman Gober? Siapa mereka?"

Kristin: "Istilah marketing sini – Pak Ogah itu orang-orang yang daftar sebagai member tapi ga pernah login lagi atau lama sekali ga aktif. Kalo Paman Gober itu member udah lama, sering login, tapi ga pernah beli apa-apa."

Rinoa: *(Mencatat istilah: Pak Ogah dan Paman Gober. Kemudian membuat dua requirement 'filter Pak Ogah' dan 'filter Paman Gober'. Lalu ia membuat hubungan antara kedua requirement dengan definisi artinya masing-masing).*

Kristin: *(Setengah memaksa)* "Jadi bisa ga?"

Rinoa: "Nanti gua tanyain anak-anak[7] deh, yang penting sekarang keperluan-keperluannya dikumpulin dulu."

Vivian: *(berbicara perlahan kepada Kristin)* "Tenang aja, kita udah sering kerja sama dengan mereka, koq."

---

[7] Anak-anak: para programmer.

Rinoa: "Visi proyeknya apa nih?"

Vivian: "Visinya…" *(diam sejenak)* "Eh, nanti gua tanya Bos dulu boleh bilang sekarang apa engga – maklum, rada classified, takut bocor."

Rinoa: "Tapi proyeknya jadi kan?"

Vivian: "Jadi lah, kita udah beli domain name-nya koq. So, berapa duwit bikinnya nih?"

Rinoa: "Kayaknya ini rada sulit…" *(menghitung requirement yang sudah dicatat kemudian berpikir sejenak)* "Mungkin sekitar satu setengah dari proyek lo yang terakhir itu – tapi ini masih kira-kira lho."

Tiba-tiba telepon genggam Rinoa berdering. Ternyata Yenny, sekretaris kantor, yang memanggil.

Yenny: "Rin, besok kan kita ada JAD[8], jadi si Bos mau kumpulin kita-kita sore ini buat rapat – lembur dikit lah. Abis ini gua kirimin agenda rapatnya via SMS."

Rinoa: *(kepada telepon)* "OK. Btw, gua kayaknya dapet proyek nih, draft SRS-nya ntar gua kirim. Tolong kasi tau bos sama anak-anak."

Yenny: "Tentu, Bos…."

Rinoa: *(menutup telepon)* "Kawan-kawan, gua dipanggil nih, sorry. Ada yang mau diomongin lagi tidak?"

Vivian: "Kayaknya segitu dulu deh."

Kristin: *(Mengangguk diam).*

Rinoa: "Ya udah, gua balik dulu ya. Nanti gua kontak lo lagi sama ngasi draft kontraknya."

Vivian: "Oke deh. Good luck ya. Ati-ati di jalan."

Rinoa: *(bangkit untuk meninggalkan ruang rapat)* "Makasih banyak ya."

---

[8] JAD: Joint Application Development.

Di dalam *subway*, Rinoa mengirimkan *draft* SRS yang sudah ditulisnya pada saat pertemuan tadi ke kantor via telepon genggam. Ia kemudian menerima satu halaman agenda rapat yang kemudian dibacanya dengan menggunakan PDA.

6.2 Spesifikasi Kebutuhan

6.2.1 Sistem membantu dalam pengumpulan informasi *requirements*

Sebagai sebuah *requirements management tool*, tentulah fungsi utamanya adalah untuk membantu mengumpulkan berbagai *requirement* yang ada. Pengumpulan ini dilakukan terutama pada saat *system analyst* berinteraksi dengan *stakeholder*. Interaksi dapat dilakukan dalam sesi-sesi formal maupun informal – seperti wawancara, *brainstorming*, *workshop*, *rapat*, maupun pada saat diskusi lewat telepon. Dalam beberapa kesempatan ini, mungkin sang *analyst* sedang tidak dapat menggunakan sebuah komputer *desktop* maupun *notebook* – untuk alasan kepraktisan ataupun alasan-alasan yang lain.

6.2.2 Sistem membantu dalam pengelolaan informasi *requirements*

Data maupun informasi yang telah dikumpulkan haruslah dapat dikelola agar menjadi suatu informasi yang berkualitas. Sistem harus dapat mengkategorikan berbagai data *requirement* yang ada untuk mempermudah akses maupun analisis lebih lanjut.

Dari *user story* pada bagian 6.1, sebuah dokumen SRS yang menjadi penampung dari berbagai informasi *requirement* terdiri atas sekumpulan *entry* yang dapat dikategorikan sebagai berikut:

- proyek
- *stakeholder*
- *requirement*
- istilah
- masalah
- batasan

Selain itu juga ada informasi *traceability* yang menghubungkan berbagai *entry* di dalam sebuah SRS.

Berbagai *entry* yang telah dimasukkan ke dalam sistem haruslah dapat diubah (*edit*) untuk memfasilitasi perubahan requirement pada analisa lebih lanjut.

6.2.3 Sistem mempunyai sebuah modul untuk digunakan di komputer genggam

Berhubungan dengan *requirement* 6.2.1, di saat absennya keberadaan komputer *desktop* atau *notebook*, *system analyst* dapat juga menggunakan sebuah *handheld computer* (yang beberapa di antaranya disebut *Personal Digital Assistant* – PDA). Sebuah komputer berukuran kecil dan *software* pendukungnya akan memungkinkan *data entry* dilakukan langsung di lokasi klien.

Sebagai staff *Information Technology* (IT), penggunaan sebuah PDA tentu tidak terlalu menjadi masalah. Hal ini kontras dengan staff *marketing* yang terkadang mengalami *technophobia*[9].

6.2.4 Sistem mempunyai sebuah modul untuk digunakan di komputer desktop

Fungsi-fungsi yang diberikan oleh komputer genggam tentulah lebih terbatas bila dibandingkan dengan komputer *desktop* atau *notebook*. Walaupun kekuatan prosesor dan ukuran memory dari *gadgets* ini terus meningkat, namun ukuran fisik layar tidak akan bertambah besar – apabila ukurannya bertambah tentulah tidak dapat lagi disebut *handheld computer*, karena sudah melampaui ukuran genggaman manusia. Belum lagi dengan minim atau absennya sebuah *keyboard* yang membatasi kecepatan maupun kemudahan *data entry*.

Karena itu perlu adanya sebuah modul untuk dijalankan di komputer *desktop* yang fungsi-fungsinya merupakan *superset* dari modul yang terdapat di komputer *handheld*. Modul ini terutama berfungsi untuk *editing* yang lebih intensif pada saat *requirement* dianalisa.

6.2.5 Informasi di aplikasi *desktop* dan *handheld* dapat dipertukarkan

Agar aplikasi *desktop* dan *handheld* dapat digunakan secara bergantian, data dokumen SRS haruslah dapat dipertukarkan antar aplikasi – secara transitif, antar *device*. Pergantian data ini haruslah terjadi semudah dan setransparan mungkin,

---

[9] Tokoh Kristin pada *user story* di bagian 6.1 bukanlah merupakan contoh dari pegawai *marketing* yang *technophobic*.

seperti yang ada pada aplikasi PIM (*Personal Information Management*) yang disertakan di semua komputer genggam yang dijual sebagai PDA.

6.3 Batasan Rancangan

6.3.1 Aplikasi harus dirancang untuk mempermudah pengembangan selanjutnya (*extensible*)

Sebagai suatu proyek *prototyping* dengan jatah waktu yang sangat singkat, tentulah tidak semua *requirement* maupun fitur yang diinginkan dapat diterapkan. Oleh karena itu, program-program yang dibuat haruslah dirancang sedemikian rupa untuk memfasilitasi pengembangan lebih lanjut.

Aspek-aspek rancangan ini termasuk dokumentasi yang lengkap serta sebuah *open architecture* yang diharapkan akan relatif stabil. Pencapaian sebuah arsitektur yang stabil dapat dilakukan dengan penerapan *design patterns* yang telah terbukti manfaatnya serta teruji oleh waktu.

6.3.2 Menggunakan [Gamm95] *design patterns*

Batasan ini berhubungan dengan *requirement* 6.3.1. Karena pola-pola rancangan yang dikategorikan di [Gamm95] telah teruji di banyak *object-oriented systems*, maka penggunaannya diharapkan dapat menghasilkan sebuah arsitektur yang baik. Fakta pendukung lainnya yaitu buku ini telah melakukan evangelisasi *design patterns* di bidang pengembangan perangkat lunak.

6.3.3 Aplikasi handheld harus *multi-platform*

Setidaknya saat ini terdapat dua sistem operasi PDA: Palm OS dan Pocket PC. Walaupun Palm OS masih memegang mayoritas, namun pangsa pasarnya telah banyak dikurangi oleh Pocket PC dari Microsoft. Belum lagi sistem operasi Symbian OS yang banyak digunakan di telepon genggam *high-end* Nokia – yang memiliki berbagai fungsi tambahan menyerupai PDA.

Untuk mengurangi resiko terjebak di tengah *platform wars* ini, modul yang diperuntukkan bagi komputer *handheld* jangan sampai tergantung pada sistem operasi tertentu. Program harus dapat dijalankan di setidaknya dua sistem operasi *handheld*; atau minimal dapat dibuat dua versi untuk masing-masing sistem operasi hanya dengan

cara kompilasi ulang (analogis dengan portabilitas antar Unix dan Windows yang ditawarkan oleh pustaka Qt untuk program C++).

6.3.4 Aplikasi desktop harus *multi-platform*.

Sebagaimana dengan batasan yang diberikan pada aplikasi untuk *handheld*, ketergantungan pada sistem operasi *desktop* tertentu juga perlu dihindari. Walaupun dominasi Microsoft Windows saat ini cukup signifikan, namun pengguna MacOS tidak bisa dibilang sedikit – terutama di Amerika Serikat. Belum lagi dengan mulai bangkitnya Linux untuk penggunaan di komputer desktop. Ketergantungan pada sistem operasi tertentu dapat memperkecil pangsa pasar sebuah *RE Tool* – yang pada awalnya juga tidak bisa dibilang besar.

6.3.5 Terdapat kode program pada *handheld* dan *desktop* yang digunakan bersama

Sehubungan dengan *requirement* 6.2.5, *data transfer* antar-aplikasi haruslah difasilitasi dengan adanya sejumlah *common code* pada kedua belah sisi pada antarmuka aplikasi *desktop* dengan *handheld*. Keuntungan yang didapatkan dengan adanya *common code* ini adalah:

- Meningkatkan konsistensi representasi data antar aplikasi.
- Mempermudah *maintenance* pada kedua aplikasi.
- Meminimalkan *recoding* maupun *reimplementation* pada kedua aplikasi.

6.3.6 Aplikasi yang dibuat tidak tergantung pada metodologi RE tertentu

Sebagai suatu bidang yang relatif baru, industri perangkat lunak masih belum setuju pada satu metodologi requirements engineering tertentu. Walaupun ada beberapa metodologi yang cukup populer, namun berbagai organisasi yang melakukan *software development* masih cenderung untuk menggunakan metodologi kesukaannya sendiri – malahan beberapa organisasi[10] tidak menggunakan metodologi RE yang baku. Selain itu, pertanyaan “Metodologi RE mana yang terbaik” juga masih diperdebatkan [Gudg00].

[10] Terutama yang masih termasuk CMM Level 1(*initial*)

Mengingat keadaan-keadaan tersebut, ketergantungan pada metodologi RE tertentu akan dapat menghambat perkembangan suatu *RE Tool* yang masih *infant* seperti Rambutan. Sehingga suatu *RE Tool* yang masih baru ini sebaiknya dapat digunakan untuk beberapa metodologi RE yang berbeda.

6.3.7 Tidak ada pembayaran kontinu kepada pihak ketiga

Pembayaran *license fee* ataupun biaya-biaya pihak ketiga lainnya secara berkelanjutan tentu dapat mengurangi keuntungan dari penjualan sebuah perangkat lunak. Apalagi bila *license fee* ini merupakan *variable cost* yang menambah biaya jual untuk setiap salinan aplikasi yang dijual.

Sebagai sebuah *prototype* yang belum mempunyai sponsor yang jelas, komponen-komponen komersial – terutama yang bisa mendatangkan biaya-biaya tambahan seperti disebutkan sebelumnya – sebaiknya dihindari. Ini ditujukan agar Rambutan tetap mempunyai banyak alternatif *development branches* yang mungkin.

6.4 Perancangan

6.4.1 Gambaran umum sistem

Sistem terdiri atas dua aplikasi yang berdiri sendiri: satu untuk komputer *handheld* dan satu lagi untuk komputer *desktop*. Aplikasi *handheld* diterapkan dengan SuperWaba sedangkan aplikasi *desktop* dibuat untuk platform Java 2 Standard Edition (J2SE). Kedua aplikasi diprogram untuk API yang berbeda, namun keduanya dibuat dengan bahasa pemrograman Java. Hal ini memungkinkan beberapa komponen untuk digunakan secara bersama di aplikasi *handheld* ataupun *desktop*.

Berkat SuperWaba, aplikasi *handheld* dapat dijalankan di sistem operasi Palm OS dan Pocket PC tanpa perlu dilakukan kompilasi ulang. Karena – seperti Java – SuperWaba adalah sebuah sistem berbasis *virtual machine*, maka *executable file* yang persis sama dapat digunakan untuk kedua platform. Selain itu berkat suatu *compatibility layer*, program-program SuperWaba dapat dijalankan dengan menggunakan sebuah Java Virtual Machine (JVM) yang memenuhi standar JDK 1.1. Pilihan kompatibilitas ini dapat digunakan untuk mendukung *handheld* dengan sistem

operasi lain yang mendukung Java 1.1, seperti misalnya komputer genggam berbasis Linux[11].

Sebagai program J2SE, aplikasi *desktop* dapat digunakan di semua sistem operasi yang mendukung Java 2 – termasuk Windows, Linux, Macintosh, OS/390, bahkan Novell Netware. Selain portabilitas, *platform* J2SE juga memberikan suatu pustaka serta API yang kaya yang akan memudahkan pengembangan aplikasi selanjutnya.

Dukungan SuperWaba terhadap Palm OS dan Pocket PC memenuhi *requirement* 6.3.3. Penggunaan J2SE memenuhi *requirement* 6.3.4. Bahasa pemrograman Java yang digunakan oleh SuperWaba dan J2SE memungkinkan pemenuhan *requirement* 6.3.5 dan 6.2.5. Fakta bahwa program-program SuperWaba dapat dijalankan di J2SE juga mendukung pencapaian *requirement* 6.2.5. Selain itu, bahasa Java yang *object-oriented* serta *platform* J2SE yang kaya akan pustaka kelas mendukung pencapaian *requirement* 6.3.1 dan 6.3.2. SuperWaba yang *open-source* serta J2SE yang dapat di-*download* secara gratis tidak bertentangan dengan batasan 6.3.7.

6.4.2 Rancangan umum

Rancangan *problem domain* sistem didasarkan terutama pada RQML. Pada dasarnya, aplikasi *desktop* maupun *handheld* adalah *editor* untuk dokumen-dokumen RQML. Aplikasi-aplikasi ini memberikan cara untuk membuat berbagai *entry* di mana masing-masing *entry* merupakan salah satu dari *first-class RQML element* (subbab 4.3 di halaman 36). RQML dipilih terutama karena jenis-jenis *first-class element* yang ada adalah *superset* dari jenis-jenis *entry* dokumen SRS yang terdapat pada *requirement* 6.2.2. Selain itu karena RQML berdasarkan XML, maka *format* dokumen SRS yang dihasilkan oleh Rambutan adalah *non-proprietary*.

Aplikasi *desktop* melakukan *load/save* terhadap data RQML secara langsung (dalam bentuk *file* XML). Sedangkan aplikasi *handheld* menggunakan suatu *format* data tersendiri (*proprietary*) yang dapat dibaca oleh aplikasi *desktop* dan kemudian disimpan dalam bentuk XML. Aplikasi *desktop* juga dapat menyimpan data RQML ke dalam *format* yang digunakan oleh aplikasi *handheld*. Bentukan *proprietary* dari data di aplikasi *handheld* dipilih agar memudahkan implementasi aplikasi tersebut serta

[11] Misalnya Sharp Zaurus™.

menghemat tempat yang diperlukan untuk menyimpan data di dalam komputer genggam.

Sebagai sebuah *RQML editor*, kedua aplikasi terdiri dari beberapa *dialog* yang masing-masing digunakan untuk menyunting sebuah *first-class RQML element*. Misalkan untuk elemen `stakeholder` terdapat *editor*-nya sendiri yang berbeda dengan *editor* untuk elemen `project`.

Kedua aplikasi ditujukan untuk satu pengguna saja (*single-user*) dan dapat menangani lebih dari satu dokumen RQML. Masalah pengumpulan dan analisis *requirement* secara kolaboratif belum ditangani pada prototipe ini.

Dokumen-dokumen di aplikasi *handheld* disimpan di dalam *device* dan ditransfer ke komputer desktop pada saat sinkronisasi[12]. Untuk tahap pertama pengembangan, aplikasi *desktop* cukup dapat membaca dan menyimpan data yang dimiliki oleh aplikasi handheld – namun satu dokumen belum dapat di-*edit* secara bersamaan di *desktop* dan *handheld* dengan perubahan-perubahannya akan digabungkan pada saat sinkronisasi[13].

Pada kedua aplikasi dilakukan pemisahan antara *user-interface* dan *data model*. Aplikasi *desktop* menggunakan Swing untuk antarmuka pemakainya. Sedangkan aplikasi *handheld* tidak mempunyai banyak piliha standar antarmuka pemakai selain yang telah disediakan oleh SuperWaba. Diharapkan bahwa komponen-komponen data yang diimplementasikan oleh aplikasi *desktop* dapat digunakan kembali (re-*use*) untuk pengembangan lebih lanjut – seperti misalnya aplikasi untuk *server*.

6.4.3 Rancangan kelas representasi data

Kelas representasi data dirancang dengan memetakan *data model* RQML yang diberikan oleh [Gudg00] hampir secara langsung ke dalam suatu *interface inheritance*.

---

[12] Konsep sinkronisasi ini umum terdapat pada komputer genggam yang dijual sebagai PDA dan dilakukan secara periodik oleh penggunanya untuk melakukan *backup* data ke *desktop*. Istilah Palm OS untuk hal ini adalah 'HotSync' sedangkan Pocket PC menamakannya 'ActiveSync'.

[13] Para programmer untuk *platform* Palm OS mungkin akan lebih memahami kalimat, "Belum ada *conduit* yang dibuat untuk HotSync."

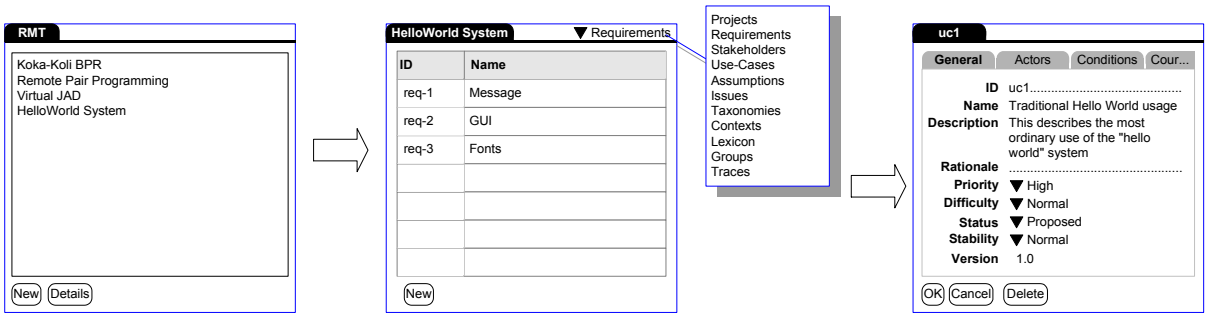
Kemudian *interface tree* ini diimplementasikan oleh dua data model, masing-masing untuk aplikasi *desktop* dan *handheld*.

Data model pada aplikasi *desktop* akan dibuat sebagai sekumpulan *adapter* yang menggunakan komponen-komponen dari pustaka pengurai XML untuk menyimpan datanya baik di *memory* maupun sebagai *file* dalam disk. Berbagai *method call* implementasi dari `interface` RQML di-*forward* ke *method call* untuk *XML parser*. Penggunaan pustaka pengurai XML secara eksklusif sebagai *back-end* memungkinkan pengembangan lebih lanjut yang menyimpan data RQML bukan di *file* – misalnya penyimpanan data XML di dalam sebuah basis data relasional[14].

Sedangkan *data model* pada aplikasi *handheld* menyimpan datanya sendiri dalam bentuk *instance variables*. Ini dilakukan karena berbagai keterbatasan yang ada pada *platform* yang digunakan – sebuah XML parser yang *fully-featured* masih terlalu besar untuk diterapkan secara efisien di dalam sebuah komputer genggam. Selain itu untuk mengikuti salah satu filsafat komputer *handheld* yaitu sebagai *satellite viewer* untuk komputer *desktop* [Palm].

6.4.4 Rancangan antarmuka pemakai

Rancangan antarmuka pemakai dibuat awalnya untuk aplikasi *handheld*. Sebagai sebuah *superset*, rancangan antarmuka pemakai untuk aplikasi *desktop* akan mengikuti semangat dari antarmuka aplikasi *handheld*.

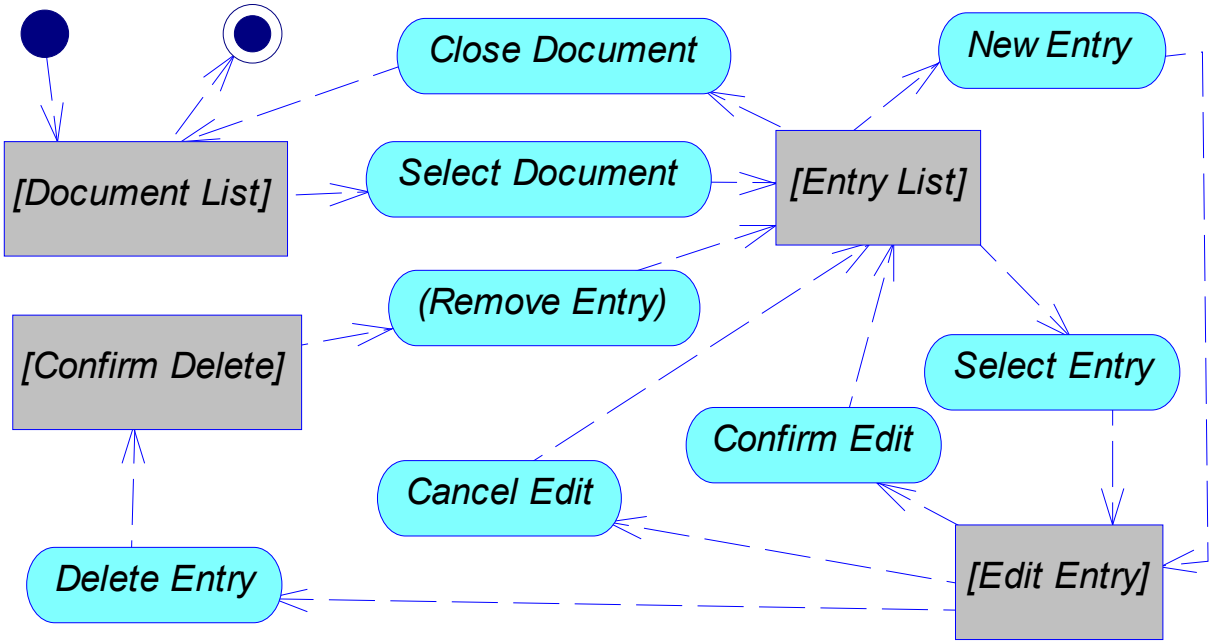


Gambar 6.1 Rancangan layar aplikasi *handheld*. Dari kiri ke kanan: *Document List*, *Entry List*, dan *Edit Entry*.

Terdapat tiga jenis layar di aplikasi *handheld*: sebuah *Document List*, *Entry List* dan akhirnya *Edit Entry*. Layar *Document List* memungkinkan pengguna untuk memilih sebuah dokumen SRS yang akan di-*edit*. Layar *Entry List* berguna untuk memilih

[14] RDBMS seperti Oracle 9i memberikan fasilitas ini.

sebuah elemen *RQML first-class* untuk disunting. Sedangkan layar *Edit Entry* digunakan untuk mengubah sebuah elemen *RQML first-class*. Pada layar *Entry List*, pengguna dapat memilih jenis dari elemen-elemen *RQML first-class* yang ditampilkan.



Gambar 6.2 *State diagram* rancangan antarmuka pemakai aplikasi *handheld*.

Gambar 6.2 menampilkan *state diagram* untuk berbagai tampilan pada aplikasi *handheld*. Status-status *Document List*, *Entry List*, dan *Edit Entry* bersesuaian dengan layar-layar yang dilukiskan sebelumnya. Karena status *Confirm Delete* hanyalah sebuah *dialog box* pertanyaan sederhana, maka ilustrasi tampilannya tidak diberikan.

Pada awalnya program menampilkan layar *Document List*. Di layar ini, pengguna dapat memilih salah satu dokumen yang akan di-edit ataupun membuat dokumen baru (status tidak ditampilkan). Kedua pilihan akan membawa pengguna pada layar *Entry List* yang menampilkan daftar berbagai *entry* yang terdapat di dalam dokumen. Di dalam layar ini, pengguna dapat membuat *entry* baru maupun menyunting *entry* yang telah ada. Kedua pilihan akan menampilkan layar *Edit Entry* yang berguna untuk mengubah *entry* yang telah dipilih. Pada layar ini, operasi *editing* dapat kemudian disimpan ataupun untuk menghapus *entry* yang sedang aktif. Setelah selesai melakukan *editing*, pengguna akan dikembalikan ke layar *Entry List*. Dari layar *Entry List*, pengguna dapat menutup dokumen yang sedang aktif untuk melakukan *editing* terhadap dokumen SRS lainnya.

Bab 7

IMPLEMENTASI

Bab ini akan membahas rincian-rincian implementasi dari program yang telah dihasilkan. Sebelumnya didahului oleh uraian singkat mengenai berbagai teknologi yang digunakan dalam pembuatan program, pembahasan dimulai pada tinjauan luas arsitektur masing-masing aplikasi. Kemudian dilanjutkan pada cara yang telah ditempuh untuk mengekspresikan data RQML – yang didefinisikan dengan sebuah XML DTD – ke dalam bentuk *object-oriented*. Dari sini pembahasan diteruskan ke antarmuka pemakai berikut beberapa contoh *screenshot* kedua aplikasi. Setelah itu akan dibahas beberapa komponen yang dominan secara arsitektural berikut *design patterns* yang digunakannya. Pembahasan pada bab ini ditutup dengan memberikan hitungan *source code metric* pada Rambutan.

7.1 Teknologi yang Digunakan

Aplikasi *handheld* diterapkan untuk *platform* SuperWaba, sedangkan aplikasi *desktop* diprogram untuk Java 2 Standard Edition (J2SE). Untuk implementasi GUI pada aplikasi *desktop* digunakan Swing, suatu pustaka dan *framework* alternatif dari J2SE di samping Abstract Windowing Toolkit (AWT). Pengurai (*parser*) XML yang digunakan adalah Apache Xerces. Sedangkan *build tool* yang digunakan untuk meng-*compile* program serta membuat paket-paket distribusi adalah Apache Ant. Dokumentasi *source code* dibuat menggunakan Doxygen.

7.1.1 Java

Java adalah suatu bahasa pemrograman berorientasi obyek yang dibuat oleh Sun Microsystems. Tidak seperti bahasa pemrograman pada umumnya, program-program Java di-*compile* ke dalam bentuk *bytecode*, yaitu sebuah bahasa mesin untuk prosesor abstrak. Sebuah perangkat lunak yang dinamakan *Java Virtual Machine* (JVM) mengimplementasikan mesin abstrak ini secara *software*. Sebagai *platform*, Java juga mendefinisikan sekelompok API untuk digunakan oleh program-programnya. Karena

di-compile menjadi bahasa mesin abstrak serta adanya standarisasi API membuat Java menjadi bahasa pemrograman yang *multi-platform*.

Sun juga menyediakan beberapa implementasi JVM berikut API-nya untuk berbagai sistem operasi yang dapat di-*download* secara gratis. Software ini secara resmi disebut Java Software Development Kit, dan secara tradisional disebut sebagai JDK. Beberapa *vendor* independen juga menyediakan berbagai implementasi JDK lainnya sebagai alternatif dari yang disediakan oleh Sun.

Sampai saat ini, *platform* Java yang ditetapkan Sun dinamakan Java 2. *Platform* ini dibagi dalam tiga edisi [Sun02a]:

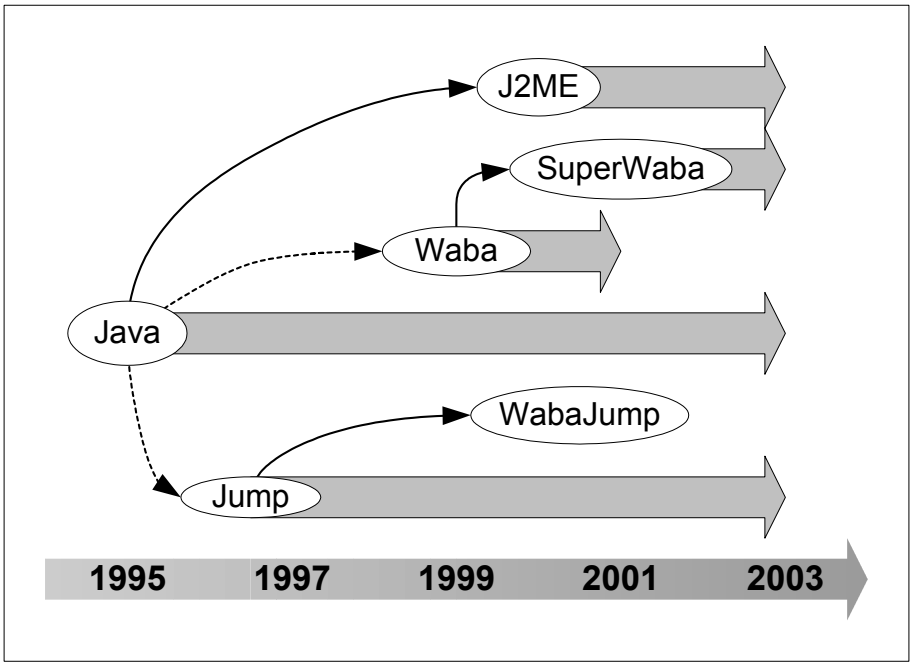
- **Java 2 Standard Edition (J2SE)** – ditujukan untuk pengembangan aplikasi bagi komputer-komputer *client* maupun *personal computer*.
- **Java 2 Enterprise Edition (J2EE)** – ditujukan bagi komputer-komputer *server* untuk pengembangan *enterprise applications*.
- **Java 2 Micro Edition (J2ME)** – ditujukan untuk *consumer electronic devices* seperti telepon genggam, *smart card*, *pager*, dan *set-top box*.

7.1.2 SuperWaba

SuperWaba (http://www.superwaba.org/) merupakan salah satu implementasi alternatif Java yang ditujukan untuk komputer genggam (*handheld*) berbasis Palm OS dan Pocket PC. Dikembangkan oleh Guilherme Campos Hazan, SuperWaba merupakan pengembangan lanjutan dari Waba 1.0g [Fari03]. Penggunaan SuperWaba diatur oleh lisensi *open-source* LGPL dan dapat di-*download* secara gratis.

Waba merupakan sebuah implementasi Java Virtual Machine (JVM) yang ditujukan untuk komputer-komputer dengan arsitektur yang sangat terbatas. Dikembangkan oleh Rick Wild pada tahun 1999, Waba ditargetkan sebagai *subset* yang hanya mendukung sebagian dari *language features* yang dimiliki oleh Java. Tidak semua *byte-code* yang didukung oleh JVM juga didukung oleh Waba VM. Walaupun merupakan implementasi JVM, namun standar API yang digunakan berbeda jauh dengan yang digariskan oleh pencipta Java, Sun Microsystems. Perbedaan-perbedaan ini memungkinkan Waba menjadi jauh lebih ramping dibandingkan Java sehingga dapat diterapkan di komputer *handheld* (yang pada masa itu lebih terbatas dibandingkan sekarang).

Salah satu keuntungan utama dari menggunakan Waba adalah penggunaan *tools* Java untuk pengembangan program. *Learning curve* untuk para developer Java relatif landai karena bahasa pemrograman yang digunakan benar-benar *subset* dari Java. Walaupun API yang digunakan berbeda, Waba memberikan suatu *compatibility layer* yang menjadi perantara antara Waba API dengan Java API sehingga program-program Waba dapat dijalankan di komputer desktop dengan menggunakan Java 1.1.



Gambar 7.1 Evolusi bahasa pemrograman Java untuk komputer genggam [Dick02] [Fari03] [Haza03] [Sun00b] [Wild99].

Kesemua ini terjadi ketika standar Java 2 Micro Edition (J2ME) masih dalam proses tinggal landas [Sun00b] – juga pada waktu itu masih sedikit *development tools* yang cukup *viable* untuk *cross-platform handheld programming*. Salah satu yang ada untuk kegunaan tersebut, MobileBuilder (http://www.penright.com/), sangat mahal dengan harga lisensi $1595 per *developer seat* [Adib02]. Dengan Waba didistribusikan secara gratis dalam lisensi LGPL membuatnya menjadi *platform* yang sangat menarik.

Implementasi Waba 1.0 mendukung banyak sekali *platform* seperti Palm OS, Windows CE, Newton, TI (kalkulator *scientific*), dan DOS [Adib02]. Namun sayangnya kemampuannya sangat terbatas – seperti sedikitnya *user-interface controls*, tidak mendukung warna, dan tidak mendukung *exception*. Selain itu nampaknya Rick Wild berhenti meneruskan pengembangan Waba pada tahun 2000 [Haza03].

Perkembangan selanjutnya dari Waba adalah SuperWaba dan WabaJump. WabaJump dikembangkan di http://www.wabajump.org/ yang merupakan penggabungan dari Waba dan Jump. WabaJump memungkinkan untuk menulis program-program Jump dengan API dari Waba. Namun perkembangan proyek ini sejak tahun 2001 nampaknya tidak banyak yang tercermin dari rendahnya aktivitas di *website*-nya.

Dikembangkan secara estafet oleh Greg Hewgill, Ralf Kleberhoff, dan terakhir Peter Dickerson, Jump (http://jump.sourceforge.net/) merupakan salah satu *effort* untuk menghasilkan *development tools* berbasis Java bagi Palm OS [Dick02]. Dengan Jump, program dalam bahasa Java di-compile ke *native code* prosesor keluarga Motorolla DragonBall yang menjadi prosesor *handheld* berbasis Palm OS 2.0 – 4.1 (Palm OS 5 menggunakan prosesor jenis lain [Palm02]). API yang digunakan oleh Jump bukanlah API standar Java, namun API Palm OS yang di-*wrap* ke bahasa Java.

Pembuat SuperWaba mengklaim bahwa SuperWaba adalah *extension* dari Waba. Namun pada kenyataannya, untuk menjalankan program-program Waba di SuperWaba terkadang diperlukan modifikasi pada *source code* program tersebut – tidak hanya di-*compile ulang* [Haza03]. Walaupun modifikasi ini relatif kecil dan tidak perlu dilakukan untuk beberapa program, namun keberadaan inkompatibilitas ini berarti bahwa SuperWaba bukanlah murni superset dari Waba.

Platform yang didukung SuperWaba juga lebih terbatas dibandingkan dengan Waba. Sampai pada versi 3.51, program-program SuperWaba hanya dapat dijalankan di Windows CE (HPC 2.11 dan PPC 2.11, untuk prosesor ARM, MIPS, maupun SH3), Pocket PC (ARM, MIPS, SH3), Palm OS (2.0–4.1 dan 5) serta JVM (minimal Java 1.1).

Namun fasilitas pemrograman yang diberikan oleh SuperWaba jauh lebih banyak ketimbang Waba pendahulunya. Dukungan layar warna, kaya akan *user-interface controls*, dukungan *exception*, pemrosesan bilangan *floating-point*[15], komunikasi via *TCP/IP network*, *serial port*, *infrared* maupun *bluetooth*, serta adanya dukungan untuk *cooperative thread* (tidak *preemptive* seperti *thread* pada umumnya) ada pada SuperWaba.

[15] Sebagai perbandingan, J2ME MIDP tidak mendukung *floating-point*.

Performa (kecepatan) yang diberikan oleh SuperWaba juga jauh di atas Waba pada *device* yang sama [Haza03]. Bahkan kecepatan SuperWaba lebih dari dua kalinya Sun MIDP (salah satu sub-*platform* dari J2ME) ketika keduanya diujikan pada Sony Clié PEG-770C (33Mhz DragonBall) [Duve02].

7.1.3 Apache Xerces

Xerces merupakan pustaka pengurai (*parser*) XML yang dikembangkan di bawah naungan Apache Software Foundation (ASF). Separti perangkat lunak ASF lainnya, Xerces disebarluaskan secara *open-source* dengan lisensi Apache Software License. Ada dua versi dari implementasi Xerces, masing-masing untuk C++ dan Java.

Sampai pada versi 2.4.0, fitur-fitur yang ditawarkan oleh Xerces adalah:

- eXtensible Markup Language (XML) 1.0 Second Edition Recommendation
- Namespaces in XML Recommendation
- Document Object Model (DOM) Level 2 Core, Events, and Traversal and Range Recommendations
- Simple API for XML (SAX) 2.0 Core, and Extension
- Java APIs for XML Processing (JAXP) 1.2
- XML Schema 1.0 Structures and Datatypes Recommendations

Pembahasan mengenai pengurai XML telah diberikan pada subbab 3.6 (halaman 27). Informasi lengkap mengenai Xerces bisa didapatkan di http://xml.apache.org/.

7.1.4 Apache Ant

Apache Ant adalah suatu *build tool* seperti *make*, namun tanpa kekurangan-kekurangannya. Ant memecahkan masalah *tab* yang sering terjadi pada para pengguna *make*. Diimplementasikan untuk Java 1.1 (untuk Ant versi 1.x), maka Ant menjadi suatu alat yang *cross-platform* untuk melakukan *build* pada program-program Java.

Awalnya Ant merupakan bagian dari Jakarta Tomcat, suatu *web server* serta *JSP container* yang dibuat sebagai bagian dari Jakarta Project di bawah naungan Apache Software Foundation (ASF). Pada waktu itu, Ant hanya digunakan untuk melakukan *build* pada *source code* Tomcat. Seiring dengan perjalanan waktu, ternyata Ant dapat

berguna untuk proyek-proyek Jakarta lainnya. Maka pada 19 Juli 2000, versi 1.1 dari Ant menjadi suatu proyek yang berdiri sendiri [Apac02a].

Seperti layaknya produk ASF lainnya, Ant merupakan perangkat lunak *open-source* yang disebarluaskan dengan lisensi Apache Software License. Berbeda dengan lisensi *open-source* GNU General Public License (GPL), produk ASF dapat dikembangkan menjadi *software* komersial yang tidak *open-source*.

Visi dari Ant adalah *simplicity*, *understandability*, dan *extensibility*. *Simplicity* berarti Ant haruslah mudah digunakan oleh seorang programmer yang kompeten. *Understandability* berarti mudah bagi seorang programmer yang sama sekali baru untuk mempelajari dan kemudian menggunakan Ant. *Extensibility* berarti Ant haruslah mudah untuk dikembangkan dengan cara menambahkan berbagai kelas Java kepadanya [Apac00a].

Menurut pembuat awalnya, James Duncan Davidson, Ant adalah sebuah akronim dari *Another Neat Tool*. Selanjutnya nama Ant lebih diasosiasikan dengan semut yang merupakan pekerja keras dan dapat mengangkat beban yang lebih daripada berat tubuhnya sendiri [Apac02a].

Karena *extensible*, kegunaan awal Ant sebagai *build tool* telah terlampaui dengan banyaknya fasilitas yang diberikan. Selain untuk meng-*compile file-file* Java, Ant juga dapat melakukan *packaging*, *deployment*, *javadoc*, operasi-operasi CVS, *automated testing*; bahkan ftp, telnet, dan e-mail. Ant juga dapat digunakan sebagai *build tool* untuk lingkungan pemrograman lainnya, seperti misalnya platform .NET dari Microsoft.

Unit kerja Ant adalah satu ***project***, yang dimuat di dalam satu file XML. Biasanya dinamakan `build.xml`, file *project* ini memuat beberapa ***target*** dan ***property set***. Pada setiap *target* ada serangkaian terurut ***task*** yang merupakan perintah-perintah yang perlu dijalankan untuk melakukan *build* pada *target* tersebut. Di antara *target* dimungkinkan adanya *dependency* yang menyatakan bahwa suatu *target* tertentu membutuhkan *target-target* lain untuk di-*build* terlebih dahulu. Sedangkan *property set* merupakan serangkaian konstanta yang mengendalikan proses *build*. Keterangan lengkap untuk menggunakan Ant dapat ditemukan di *website* http://ant.apache.org/.

7.1.5 Doxygen

Doxygen (http://www.doxygen.org/) adalah sebuah alat bantu untuk membuat dokumentasi *source code*. Dokumentasi dibuat sebagai *comment* bertanda khusus yang diletakkan persis sebelum *class*, *method*, *function*, ataupun *variables* untuk memberikan keterangan mengenai bagian kode yang bersangkutan. Doxygen akan memproses *source code* ini dan kemudian mengambil *comment-comment* khusus tersebut untuk diformat dan dijadikan sebuah dokumentasi tersendiri.

Dibuat oleh Dimitri Van Heesch, Doxygen awalnya ditujukan untuk memproses *source code* C++ untuk proyek-proyek pribadi yang ditulis menggunakan pustaka Qt. Kini *source code* yang dapat diproses oleh Doxygen juga meliputi Java, PHP, C#, dan juga IDL.

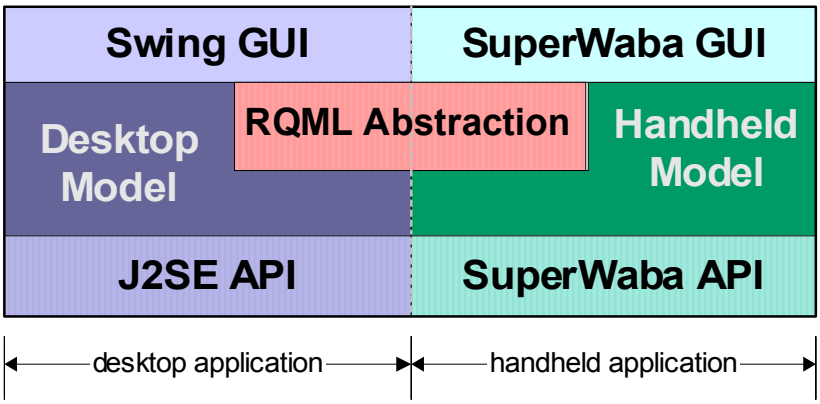
Kelebihan-kelebihan Doxygen antara lain:

- Mampu membuat dokumentasi dalam berbagai format: HTML, RTF, LaTeX, *man page*, XML, dan bahkan *Compiled HTML* (alias *Windows 98 help*).
- Kompatibel dengan JavaDoc 1.1, Qt-Doc dan KDOC.
- Dapat membuat *class diagram* dan *collaboration diagram* secara automatis.
- Dapat membuat tampilan *source code* secara *hypertext*, lengkap dengan *syntax highlighting*.

Doxygen diimplementasikan dalam bahasa C++ dengan pustaka Qt sehingga dapat digunakan di *platform* Unix dan Windows. Selain itu Doxygen juga disebarluaskan dengan lisensi GNU GPL dan dapat di-*download* secara gratis.

7.2 Arsitektur

Rambutan terdiri dari dua aplikasi: sebuah aplikasi *desktop* dan aplikasi *handheld*. Aplikasi *desktop* diprogram untuk API Java 2 Standard Edition (J2SE) sedangkan aplikasi *handheld* diprogram untuk API SuperWaba. Kedua aplikasi ditulis dalam bahasa pemrograman Java. Penggunaan bahasa pemrograman yang sama memungkinkan kedua aplikasi untuk menggunakan beberapa komponen yang sama.



Gambar 7.2 Selayang pandang arsitektur kedua aplikasi

Secara umum aplikasi *desktop* maupun *handheld* masing-masing terbagi dalam dua lapisan utama: lapisan antarmuka pemakai dan lapisan model data. Lapisan antarmuka pemakai bertanggungjawab atas tampilan grafis serta interaksi dengan pengguna. Sedangkan lapisan model data mengatur data *requirement* yang diproses oleh masing-masing aplikasi.

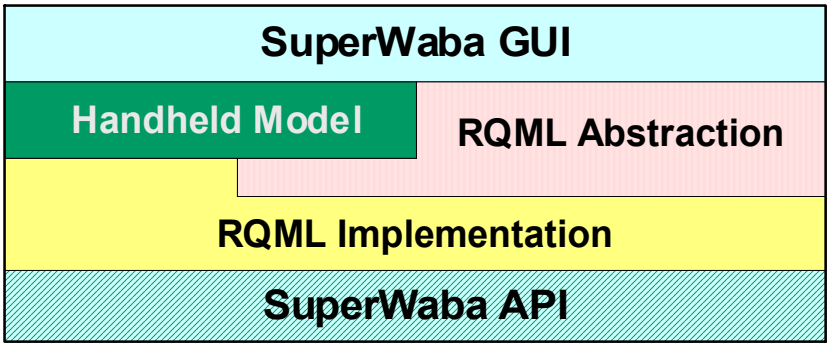
Kedua aplikasi ini menggunakan satu komponen yang sama, yaitu *RQML Abstraction*. Komponen ini adalah serangkaian `interface` yang berfungsi untuk mendefinisikan abstraksi data RQML. Pada gilirannya, kumpulan antarmuka ini diimplementasikan oleh kelas-kelas *data model* pada masing-masing aplikasi *desktop* maupun *handheld*. Akses terhadap penerapan RQML di kedua aplikasi dilakukan sebanyak mungkin melalui abstraksi RQML ini. Dengan adanya komponen bersama ini maka konsistensi representasi data antara aplikasi desktop dan aplikasi *handheld* akan lebih terjamin – perubahan *interface* RQML akan beriak ke kedua *data model* di *desktop* maupun *handheld*.

Rambutan terbagi atas empat *package* utama yang kesemuanya dinaungi dalam *package* `com.arcle.rmt`:

- `com.arcle.rmt.j2se` – Aplikasi *desktop*
- `com.arcle.rmt.superwaba` – Aplikasi *handheld*
- `com.arcle.rmt.rqml` – Antarmuka-antarmuka abstraksi RQML
- `com.arcle.rmt.xplat` – Kelas-kelas multiguna lintas-*platform*

7.2.1 Aplikasi Handheld

Aplikasi *handheld* sebenarnya terdiri dari tiga lapisan besar: lapisan *SuperWaba GUI*, *Handheld Model*, dan *RQML Implementation*. Sedangkan *RQML Abstraction* merupakan komponen yang digunakan bersama di desktop maupun handheld. Kesemua lapisan bergantung pada API SuperWaba.



Gambar 7.3 Selayang pandang arsitektur aplikasi *handheld*

Lapisan *SuperWaba GUI* merupakan implementasi dari antarmuka pemakai. Kelas-kelas dalam lapisan ini umumnya merupakan turunan dari kelas `Window` yang dimiliki oleh SuperWaba. Akses kepada data dilakukan hanya melalui lapisan-lapisan *Handheld Model* ataupun *RQML Abstraction*; kelas-kelas di dalam *RQML Implementation* tidak pernah diakses secara langsung oleh kelas-kelas GUI.

Kelas-kelas di dalam lapisan *Handheld Model* menyediakan layanan pengaturan elemen-elemen RQML kepada GUI. Sebagian besar operasi yang dilakukan oleh lapisan ini untuk menangani elemen-elemen RQML hanya bergantung kepada antarmuka-antarmuka yang ditetapkan oleh komponen *RQML Abstraction*. Salah satu perkecualian utama adalah pada saat pembuatan obyek dari elemen-elemen RQML (di mana lapisan ini bertindak sebagai *abstract factory* kepada penggunanya).

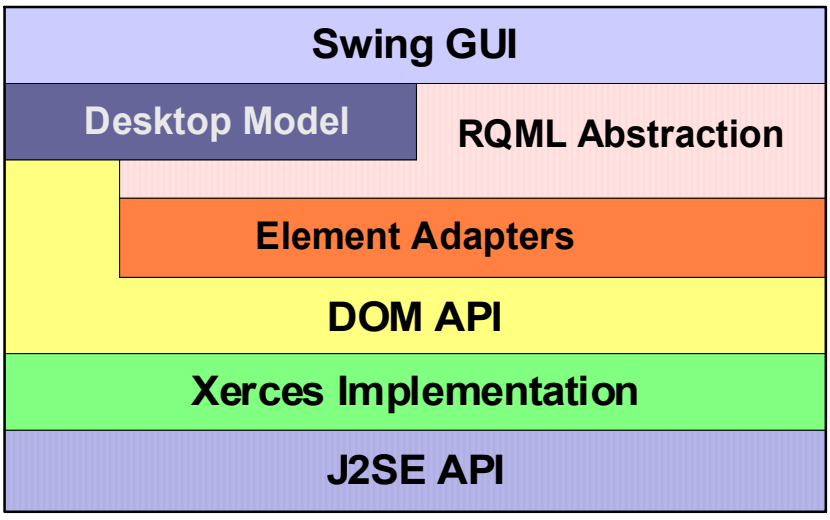
Karena kelas-kelas di dalam *RQML Abstraction* semuanya adalah *interface* yang abstrak, agar berguna mereka harus dibuat penerapannya. Lapisan *RQML Implementation* menerapkan antarmuka-antarmuka ini untuk digunakan di dalam aplikasi handheld. Karena lapisan ini terlindung dari GUI oleh lapisan *Handheld Model* maupun *RQML Abstraction*, perubahan implementasi padanya tidak akan mempengaruhi lapisan GUI.

Aplikasi *handheld* terletak di dalam package `com.arcle.rmt.superwaba`, dan diatur dalam beberapa *sub-package* berikut:

- `com.arcle.rmt.superwaba.vc`
  Kelas-kelas antarmuka pemakai
- `com.arcle.rmt.superwaba.model`
  Kelas-kelas *data model* serta penerapan RQML.
- `com.arcle.rmt.superwaba.ui`
  Kelas-kelas antarmuka pemakai multiguna
- `com.arcle.rmt.superwaba.util`
  Kelas-kelas multiguna
- `com.arcle.rmt.superwaba.framework`
  Antarmuka-antarmuka untuk *framework* MVC

7.2.2 Aplikasi *Desktop*

Aplikasi *desktop* juga terdiri dari tiga lapisan utama: *Swing GUI*, *Desktop Model*, dan *RQML Implementation*. Lapisan *RQML Abstraction* adalah komponen yang sama dengan yang digunakan di aplikasi *handheld*. Kesemua lapisan bergantung pada API J2SE.



Gambar 7.4 Selayang pandang arsitektur aplikasi *desktop*

Antarmuka dari aplikasi *desktop* menggunakan pustaka GUI Swing yang tersedia di J2SE. Walaupun fungsi-fungsinya serupa dengan aplikasi *handheld*, GUI pada aplikasi *desktop* meraup keuntungan layar yang lebih besar serta kemampuan antarmuka pemakai yang lebih kaya yang umum terdapat di sebuah komputer

desktop. Akses terhadap data juga dilakukan hanya melalui kelas-kelas model dan *RQML Abstraction*.

Karena sebuah komputer *desktop* menawarkan kemampuan pemrosesan yang lebih besar serta antarmuka pemakai yang lebih kaya, *data model* di desktop mempunyai lebih banyak fitur untuk mengambil manfaat dari kenyataan ini. Maka dari itu, *data model* pada aplikasi *desktop* mempunyai *method signature* serta implementasi yang berbeda dengan *data model* pada aplikasi *handheld*. Seperti aplikasi *handheld*, *data model* ini juga bergantung pada antarmuka-antarmuka pada *RQML Abstraction* untuk memanipulasi elemen-elemen RQML dan bertindak sebagai kelas *factory* untuk membuat obyek implementasinya.

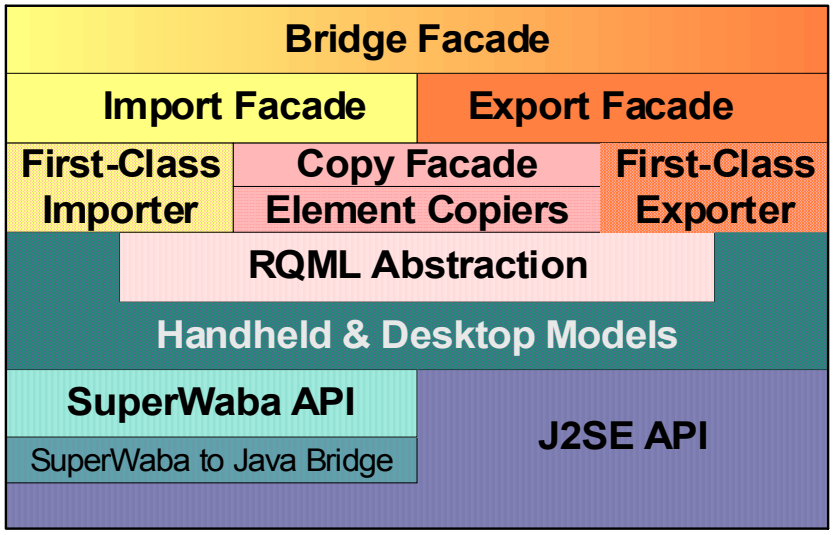
Lapisan *RQML Implementation* pada aplikasi *desktop* berbeda dengan rekannya di *handheld*. Ketimbang menyimpan datanya sendiri, kelas-kelas ini bergantung pada obyek-obyek DOM untuk mewakili data RQML di *memory*. Maka dari itu kelas-kelas dalam lapisan ini secara kolektif disebut *Element Adapters*. Karena adapter-adapter ini berkomunikasi dengan *XML parser* hanya melalui antarmuka DOM yang disediakan, sebuah pengurai XML lainnya dapat digunakan asalkan memenuhi standar DOM.

Aplikasi desktop terdapat dalam *package-package* ini:

- `com.arcle.rmt.j2se.swing.vc`
  Kelas-kelas antarmuka pemakai.
- `com.arcle.rmt.j2se.model`
  Kelas-kelas *data model* and *element adapter*.
- `com.arcle.rmt.j2se.swing.ui`
  Kelas-kelas umum antarmuka pemakai
- `com.arcle.rmt.j2se.util`
  Kelas-kelas multiguna
- `com.arcle.rmt.j2se.framework`
  Antarmuka-antarmuka *framework*

7.2.3 Komponen Jembatan

Ilustrasi aplikasi *desktop* yang ditunjukkan pada **Gambar 7.4** hanya menampilkan sebagian dari arsitekturnya secara keseluruhan. Sebuah komponen jembatan sengaja disembunyikan agar gambar ini lebih mudah dicerna. Komponen jembatan ini digunakan oleh aplikasi *desktop* untuk mengakses data yang dimiliki oleh aplikasi *handheld*. Ia menangani pengubahan antara representasi data *desktop* dengan *handheld*.



Gambar 7.5 Selayang pandang arsitektur komponen *bridge*

Ketika seorang pengguna membuka ataupun menyimpan dokumen *handheld* menggunakan aplikasi *desktop*, antarmuka pemakai dari aplikasi *desktop* memanggil komponen *Bridge Facade* untuk membaca data dalam format *handheld* dan memuatnya ke dalam *data model desktop* serta sebaliknya. Pada gilirannya, *facade* ini mendelegasikan permintaan tersebut ke *Import Facade* atau *Export Facade* yang sesuai dengan operasi yang dilakukan. Kedua *facade* ini kemudian menangani operasi *import* ataupun *export* yang diminta.

Komponen *Import Facade* menggunakan suatu himpunan kelas *First-Class Importer* untuk memuat data dari representasi *handheld* ke *desktop*. Setiap kelas pengimpor ini mengimpor satu jenis elemen *RQML First-Class*.

Dengan cara yang sama, komponen *Export Facade* menggunakan suatu himpunan *First-Class Exporter* untuk menyimpan data dari *desktop* ke handheld. Mirip dengan pengimpor, ada pemetaan satu-satu antara kelas pengekspor dengan tipe elemen *RQML First-Class*.

Kedua facade bergantung pada sebuah *Copy Facade* untuk menyalin data antar elemen-elemen RQML. Pada gilirannya, facade ini menggunakan sehimpunan kelas *Element Copier* untuk menyalin data-data tersebut. Ada pemetaan satu-satu antara sebuah kelas penyalin dan elemen RQML (untuk elemen *first-class* maupun *second-class*). Para penyalin ini hanya bergantung pada antarmuka *RQML Abstraction* – mereka tidak mengetahui implementasi yang mendasari antarmuka-antarmuka ini. Maka dari itu para penyalin ini dapat digunakan untuk menerapkan pengimpor dan pengekspor RQML untuk berbagai implementasi *data model* lainnya di masa depan.

Baik *facade* pengimpor maupun pengekspor berinteraksi secara langsung dengan *data model* di desktop maupun handheld. Bertindak sebagai *client* terhadap kedua model, mereka mengambil serta menyimpan data RQML antara keberadaan-keberadaan dari kedua model.

Kode program untuk *data model* aplikasi *handheld* dijalankan di *desktop* sehingga data RQML dalam format *handheld* dapat dibaca dan dimuat ke dalam *data model* aplikasi desktop. Walaupun *data model* di *handheld* diprogram untuk SuperWaba API, hal ini dimungkinkan karena adanya lapisan simulasi (*SuperWaba to Java Bridge*) yang disediakan oleh SuperWaba agar program-programnya dapat dijalankan di Java 1.1. Karena itu tidaklah perlu untuk menduplikasikan logika program dari aplikasi *handheld* untuk membaca datanya.

Komponen jembatan ini terletak di dalam *package* `com.arcle.rmt.j2se.bridge` dan dibagi lagi menjadi beberapa sub-*package* berikut:

- `com.arcle.rmt.j2se.bridge.copy`
  Kelas-kelas penyalin beserta *facade*-nya
- `com.arcle.rmt.j2se.bridge.swimport`
  Kelas-kelas pengimpor untuk memuat data dari aplikasi *handheld*.
- `com.arcle.rmt.j2se.bridge.swexport`
  Kelas-kelas pengekspor untuk menyimpan data ke dalam format *handheld*.

7.3 Pemetaan dari RQML ke class

Pemetaan data model RQML ke kelas dilakukan dengan cara mengubah hirarki *data model* pada **Gambar 4.2** (halaman 39) menjadi hirarki `interface`. Untuk setiap

elemen RQML dibuatkan satu `interface` yang mewakilinya, baik bagi elemen *first-class* maupun *second-class* – termasuk berbagai sub-elemen dari elemen-elemen *first-class*.

Dalam proses pembuatan hirarki interface RQML dilakukan *refactoring*. Bila ada dua elemen yang mempunyai sekelompok attribut yang sama, maka dibuat satu *superinterface* dari kedua *interface* yang mewakili elemen-elemen tersebut. *Super-interface* ini memuat *common attributes* yang dimiliki oleh kedua elemen. Proses ini juga dilakukan untuk kasus umum di mana ada lebih dari dua elemen yang mempunyai attribut yang sama: untuk elemen-elemen yang mempunyai sekelompok attribut yang sama dibuat satu atau lebih *superinterface* untuk *interface-interface* yang mewakili elemen-elemen itu.

Dalam rangka menyederhanakan hirarki *interface* serta kelas-kelas yang akan meng-implementasinya, suatu elemen yang dapat mempunyai lebih dari satu *instance* dari *child element* untuk suatu jenis elemen yang sama – yaitu elemen-elemen anakan yang ditulis dengan *suffix* '*' atau '+' dalam DTD [Marc99], maka *interface* yang mewakilinya hanya mendukung satu *instance* dari *child element* tersebut. Ada tiga hal yang memotivasi penyederhanaan ini:

- Ditemukan beberapa ketidakkonsistenan pada spesifikasi RQML dalam [Gudg00][16] yang menandakan bahwa (spesifikasi) RQML masih belum stabil.
- RQML belum menjadi bakuan yang disahkan oleh suatu badan independen (seperti misalnya ISO, IETF atau W3C) sehingga tidak perlu 100% *compliant* dengan spesifikasi RQML untuk interoperabilitas dengan alat bantu berbasis RQML lainnya[17].
- Kesulitan teknis dalam pengimplementasian antarmuka pemakai yang efisien pada aplikasi *handheld* untuk mendukung *multiple child element*s ini.

Berikut ini adalah penggalan DTD untuk elemen `<requirement>`, ditulis kembali dari subbab 4.6.4.1:

[16] Berhubung proyek ini tidak bertujuan untuk memvalidasi [Gudg00], kemungkinan masih ada lagi ketidakselarasan yang ada selain yang sudah ditemukan.

[17] Bahkan satu-satunya software *requirements management* berbasis RQML lainnya yang diketahui adalah [Dhar02].

```
<!-- requirement  -->
<!ELEMENT requirement (name?, description*, rationale?)>
<!ATTLIST requirement
     id ID #REQUIRED
     priority (low | normal | high) "normal"
     difficulty (low | normal | high) "normal"
     status (proposed | approved | incorporated | validated) #IMPLIED
     stability (low | normal | high) "normal"
     version CDATA #IMPLIED
>
<!-- "name" was already declared -->
<!-- "description" was already declared -->
<!-- "rationale" was already declared -->
```

Elemen ini dipetakan menjadi beberapa `interface` yaitu `RQMLSecondClass`, `RQMLFirst-Class`, dan `Requirement`:

```
public interface RQMLSecondClass {
   public abstract java.lang.String getURI();
   public abstract void setURI(java.lang.String URI);
   public abstract java.lang.String getID();
   public abstract void setID(java.lang.String ID);
}
```

Attribut-attribut `priority`, `difficulty`, `status`, dan `stability` semuanya adalah jenis enumerasi di mana satu nilai dapat dipilih dari sekumpulan pilihan nilai yang terbatas. Attribut-attribut ini dipetakan menjadi tipe data `int` di mana nilai dari tipe enumerasinya dispesifikasikan dengan menggunakan konstanta-konstanta. Misalkan untuk memberikan attribut `status` nilai `approved` digunakan konstanta `STATUS_APPROVED`.

```
public interface RQMLFirstClass extends RQMLSecondClass {
   public static final int PRIORITY_LOW = -1;
   public static final int PRIORITY_NORMAL = 0;
   public static final int PRIORITY_HIGH = 1;
   public static final int DIFFICULTY_LOW = -1;
   public static final int DIFFICULTY_NORMAL = 0;
   public static final int DIFFICULTY_HIGH = 1;
   public static final int STATUS_PROPOSED = 0;
   public static final int STATUS_APPROVED = 1;
   public static final int STATUS_INCORPORATED = 2;
   public static final int STATUS_VALIDATED = 3;
   public static final int STABILITY_LOW = -1;
   public static final int STABILITY_NORMAL = 0;
   public static final int STABILITY_HIGH = 1;

   public abstract Name getName();
   public abstract int getPriority();
   public abstract int getDifficulty();
   public abstract int getStatus();
   public abstract int getStability();
   public abstract java.lang.String getVersion();
   public abstract Description getDescription();
   public abstract Rationale getRationale();
   public abstract void setName(Name NAME);
   public abstract void setDescription(Description DESC);
   public abstract void setRationale(Rationale RATIONALE);
```

```
   public abstract void setPriority(int PRIORITY);
   public abstract void setDifficulty(int DIFFICULTY);
   public abstract void setStatus(int STATUS);
   public abstract void setVersion(java.lang.String VERSION);
   public abstract void setStability(int stability);
}
```

Karena *tag* `<requirement>` tidak menambah informasi apapun dari *superclass*-nya, maka `interface` yang mewakilinya menjadi kosong (tidak mendefinisikan *method* apapun).

```
public interface Requirement extends RQMLFirstClass {
}
```

Ketidakkonsistenan pada elemen `<project>` (subbab 4.6.4.6, di halaman 47) ditanggulangi dengan menurunkan `interface Project` dari `interface RQMLFirstClass` dan menggunakan *property-property* berikut sebagai informasi tambahan yang dimiliki oleh elemen `project`:

- *product*
- *problem*
- *scope*
- *vision*

Sedangkan ketidakkonsistenan pada definisi elemen `<assumption>` (subbab 4.6.4.3 di halaman 46) dianggap tidak ada – *interface* `Assumption` diturunkan langsung dari `RQMLFirstClass` dan tidak mendefinisikan *method* tambahan apapun.

Dalam implementasinya, ternyata untuk mengakses *property-property* yang bertipe obyek hanya method *getter* (dengan pola menamaan `getXxx()` pada definisi *interface*) yang dipergunakan. Berbagai method *setter* (dengan pola penamaan `setXxx()`) hanya digunakan untuk mengakses *property-property* dengan tipe sederhana, seperti `String` atau `int`. Karena aplikasi hanya diberi *interface* dari kelas-kelas elemen RQML[18], maka kesulitan yang ditemui pada penerapan *method setter* bagi *property* yang juga *interface* RQML adalah bagaimana caranya aplikasi dapat menciptakan obyek baru untuk di-*set* ke dalam obyek RQML yang bersangkutan. Salah satu alternatif adalah dengan memberikan sebuah *factory method* ataupun *factory object* bagi *application*

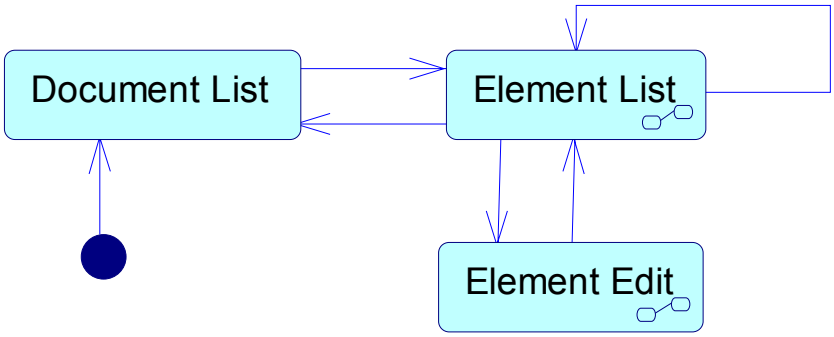
[18] Sesuai dengan salah satu nasihat dari [Gamm95], "*Program to an interface, not to an implementation.*"

code, namun hal ini akan mempersulit keadaan. Maka diputuskan bahwa *factory* tetap hanya digunakan oleh obyek-obyek penerapan *interface* RQML dan obyek untuk sub-elemen dibuat secara dinamis pada akses pertama dari elemen tersebut dengan menggunakan method `getXxxx()`. Modifikasi data pada sub-elemen dilakukan langsung pada obyek elemen tersebut dengan *method setter* untuk mengubah *property* dengan tipe sederhana pada sub-elemen itu. Dengan kata lain, pengubahan data dilakukan pada *leaf property* yang notabene memiliki tipe data sederhana.

7.4 Antarmuka pemakai

7.4.1 Aplikasi *Handheld*

Ada tiga layar utama di aplikasi *handheld*. Mereka adalah *Document List*, *Element List*, dan *Element Edit*. Karena melingkupi seluruh daerah tampilan layar pada komputer genggam, maka hanya salah satu dari layar-layar ini yang ditampilkan pada suatu saat tertentu.



Gambar 7.6 Diagram status dari antarmuka aplikasi *handheld*

Aplikasi *handheld* dimulai pada tampilan *Document List* yang menampilkan daftar dokumen *requirement* yang ada di dalam komputer genggam. Ketukan pada sebuah nama dokumen akan membuka dokumen tersebut. Juga terdapat sebuah tombol perintah *New* pada bagian bawah layar yang digunakan untuk membuat suatu dokumen baru. Ketika pengguna mengetuk tombol ini, sebuah kotak dialog akan ditampilkan yang menanyakan nama dari dokumen yang akan dibuat. Setelah si pengguna memberikan sebuah nama, dokumen baru ini akan diciptakan kemudian dibuka – seolah-olah dokumen itu telah ada sebelumnya dan sang pengguna mengetuk namanya.

Membuka sebuah dokumen akan menampilkan layar *Element List* serta menampilkan nama dokumen tersebut pada *title-area* di bagian atas layar. Sebenarnya layar ini

terdiri dari suatu himpunan tampilan di mana masing-masing menampilkan daftar elemen-elemen RQML *first-class* dengan tipe-tipe tertentu. Masing-masing daftar elemen ini menampilkan attribut-attribut RQML *ID* dan *Name*. Perpindahan antara tipe elemen *first-class* dilakukan melalui sebuah *combo box* pada bagian kanan-atas layar. Ketukan pada *title-area* akan menampilkan sebuah *pull-down menu* di mana pengguna dapat menciptakan sebuah elemen RQML *first-class* baru dengan tipe yang diinginkan. Sedangkan ketukan pada nama sebuah elemen *first-class* akan menyunting elemen tersebut.

Tiga tombol perintah disediakan pada bagian bawah layar *Element List*:

- **New** – untuk menciptakan sebuah elemen *first-class* baru dengan tipe yang sama dengan yang ditampikan pada saat itu.
- **Close** – untuk menutup dokumen yang sedang ditampilkan dan kembali ke layar *Document List*.
- **Delete** – untuk menghapus dokumen yang sedang ditampilkan dan kembali ke layar *Document List*.

Pengaturan antarmuka pemakai ini meniru gaya dari beberapa aplikasi yang disertakan oleh Palm OS. Aplikasi-aplikasi *Address Book*, *To Do List*, *Memo Pad*, dan *Mail* semuanya memiliki sebuah *combo box* pengubah kategori pada bagian kanan atas dan suatu baris tombol perintah pada bagian bawah layar.

Ketika sang pengguna membuat sebuah elemen baru atau membuka elemen yang telah ada, maka sebuah layar *Element Edit* akan ditampilkan. Layar ini juga merupakan suatu himpunan layar di mana masing-masing layar digunakan untuk menyunting suatu kelompok tertentu dari attribut-attribut yang dimiliki oleh sebuah elemen RQML *first-class*. *Combo box* pengubah kategori digunakan untuk berpindah antara pandangan-pandangan dari elemen *first-class* yang sama yang sedang di-*edit*. Daerah judul dari layar-layar ini menampilkan tipe dari elemen yang sedang aktif.

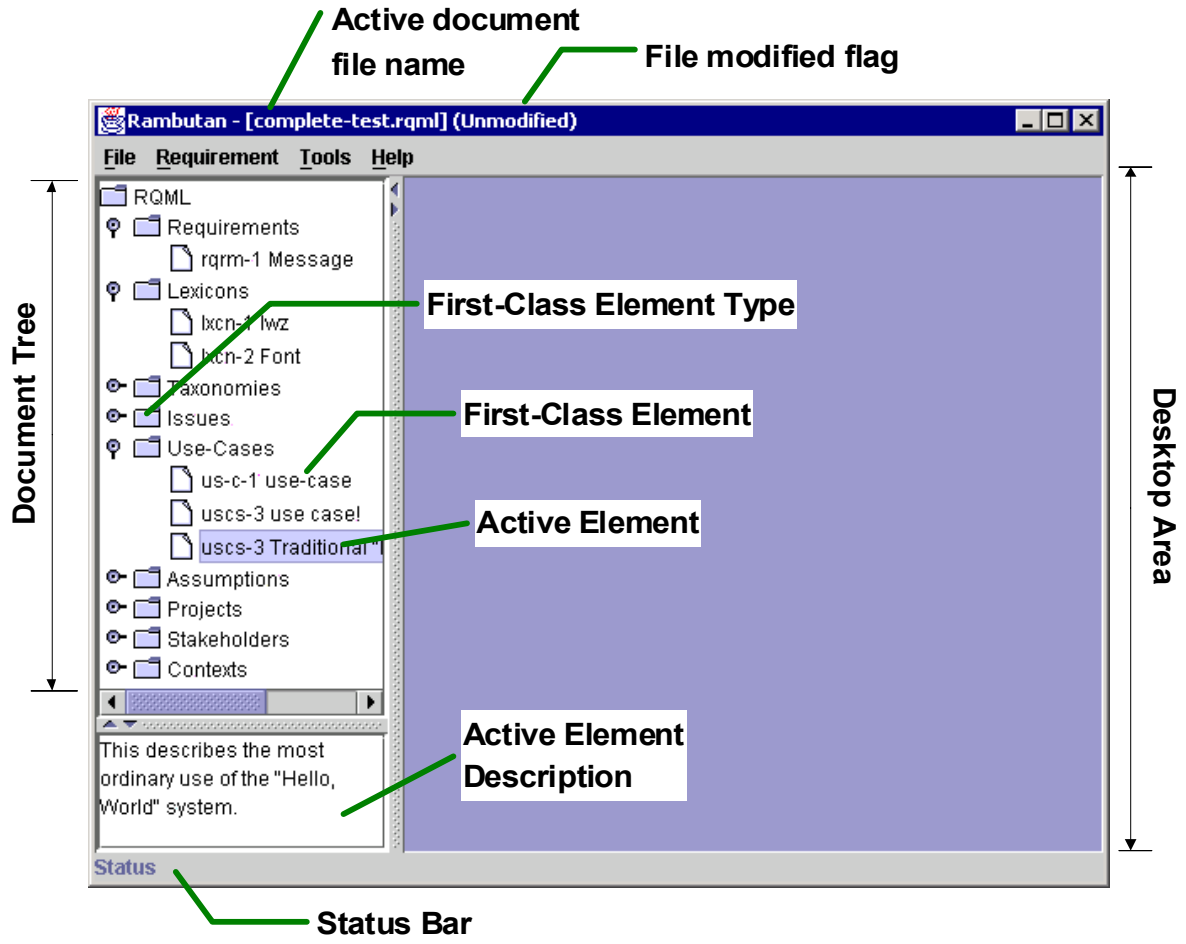
Gambar 7.7 Layar *Document List*.

Gambar 7.8 Salah satu layar *Element List*; yang ini menampilkan daftar elemen dengan tipe *Use-case*.

Gambar 7.9 Salah satu layar *Element Edit*; yang ini menampilkan sebagian dari attribut-attribut sebuah *Requirement*.

7.4.2 Aplikasi Desktop

Susunan antarmuka aplikasi *desktop* terutama terinspirasi oleh tampilan Rational RequisitePro. Pada dasarnya tampilan adalah sebuah jendela *Multiple Document Interface* (MDI) dengan sebuah *tree view* yang diletakkan di bagian kiri. Daerah *desktop* digunakan untuk penampilkan jendela-jendela editor elemen *first-class*.



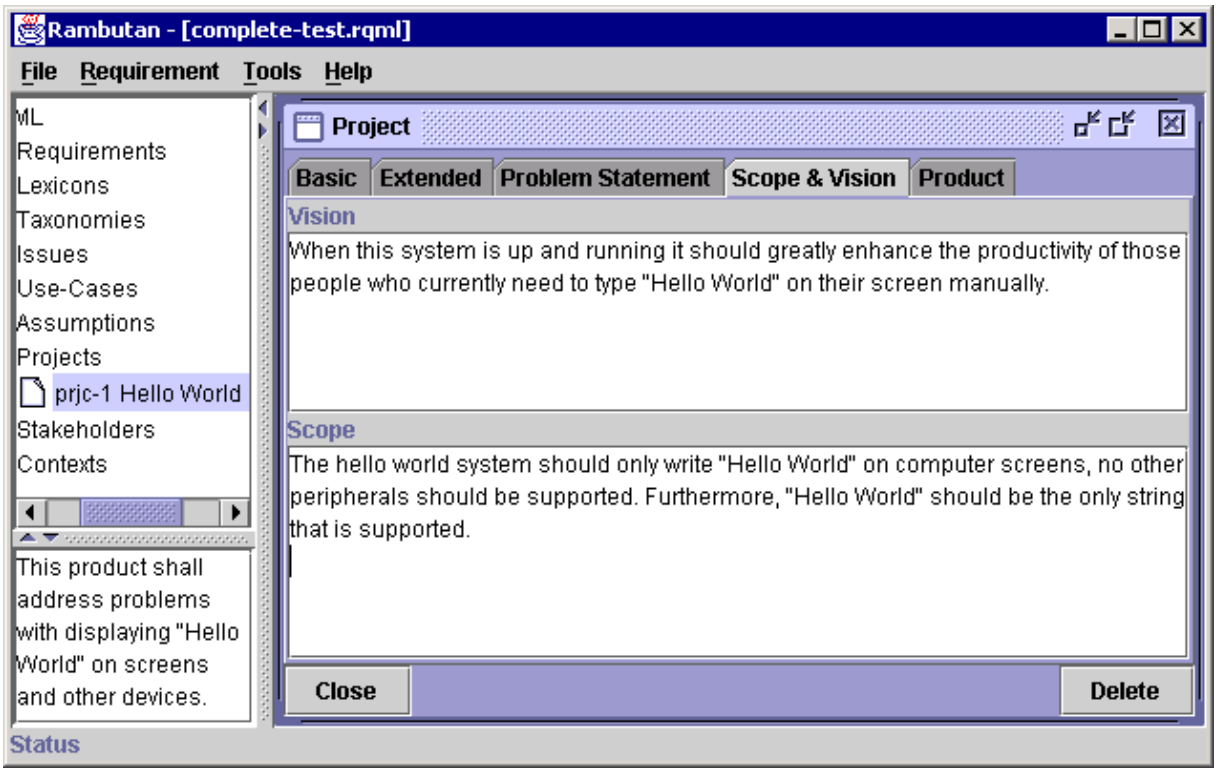
Gambar 7.10 Selayang pandang tampilan aplikasi *desktop*

Tampilan ***Document Tree*** pada bagian kiri dari jendela menampilkan jenis-jenis elemen *first class* sebagai *folder* di dalam *tree*. Klik ganda pada sebuah *folder* ini akan membuat sebuah elemen *first-class* dengan tipe dari *node* tersebut. Elemen-elemen baru juga dapat diciptakan melalui menu *Requirement*.

Kotak ***Active Element Description*** yang terletak persis di bawah *Document Tree* menampilkan attribut *description* dari elemen di dalam pohon yang sedang disorot. Kotak ini berfungsi untuk memberikan pandangan sekilas ketika pengguna sedang melihat-lihat berbagai elemen *first-class* yang ada di dalam dokumen – sehingga atribut ini dapat dilihat tanpa perlu meng-*edit* elemen yang bersangkutan.

Bagian ***Desktop Area*** digunakan untuk menampung jendela-jendela penyunting elemen *first-class*. Meraup keuntungan dari antarmuka pemakai yang lebih kaya di komputer desktop, daerah ini mampu menampilkan lebih dari satu jendela *editor*.

Status Bar adalah suatu daerah di bagian bawah jendela yang menampilkan informasi status. Daerah ini disiapkan untuk pengembangan selanjutnya; pada saat ini tidak berfungsi.



Gambar 7.11 Contoh tampilan yang menampilkan jendela penyunting elemen *first-class*.

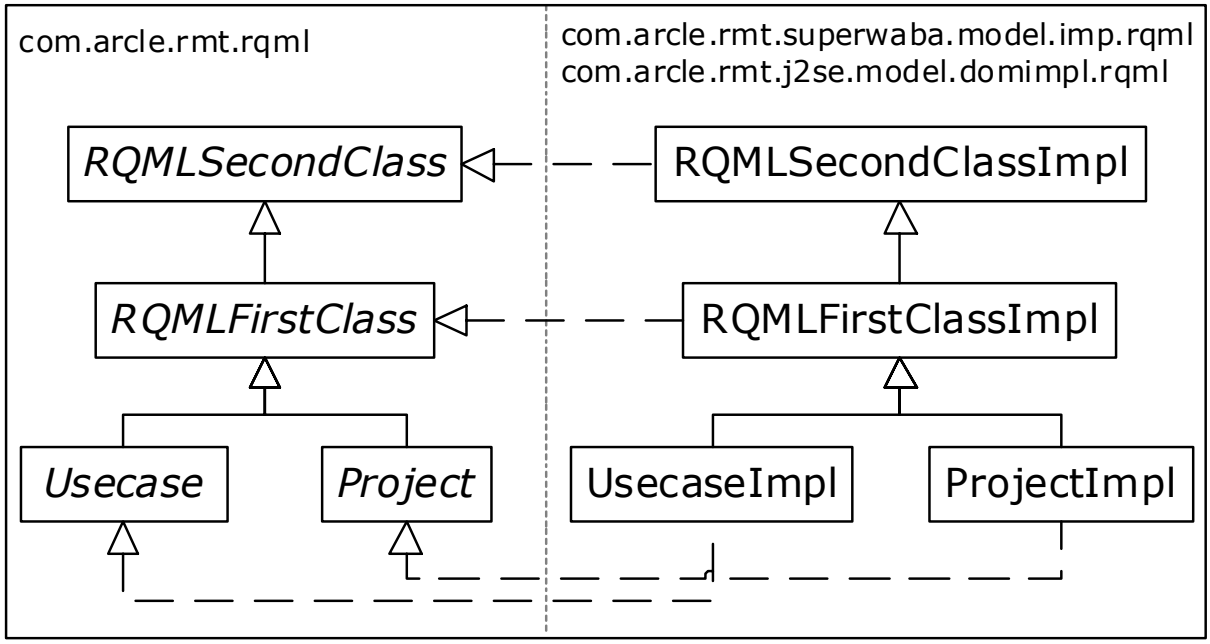
Jendela-jendela penyunting elemen *first-class* semuanya adalah *internal frame* yang diletakkan di *Desktop Area*. Seperti yang terdapat pada aplikasi *handheld*, setiap jendela *editor* mempunyai beberapa sub-bagian di mana masing-masing mempunyai himpunan komponennya sendiri untuk menyunting bagian-bagian dari elemen *first-class* yang bersangkutan. Pada aplikasi *desktop*, sub-bagian ini ditaruh di dalam sebuah *tabbed pane* di dalam bingkai penyunting. Ada dua tombol perintah yang terletak pada baris bawah dari masing-masing jendela penyunting:

- **Close** – menutup jendela penyunting dan memasukkan perubahan-perubahan yang dilakukan.
- **Delete** – menghapus elemen yang sedang di-edit dari dokumen dan kemudian menutup jendela penyuntingnya.

7.5 Penerapan *Design Patterns*

7.5.1 Representasi data RQML

Sebagaimana telah digariskan di subbab 7.2, *data model* RQML didefinisikan oleh sebuah hirarki `interface`. Kumpulan antarmuka ini kemudian diimplementasikan oleh kelas-kelas representasi data pada aplikasi *desktop* maupun *handheld*. Untuk masing-masing kelas antarmuka RQML terdapat satu kelas penerapannya sendiri, sehingga terjadi korespondensi satu-satu antara kelas-kelas *interface* RQML dengan kelas-kelas penerapannya.



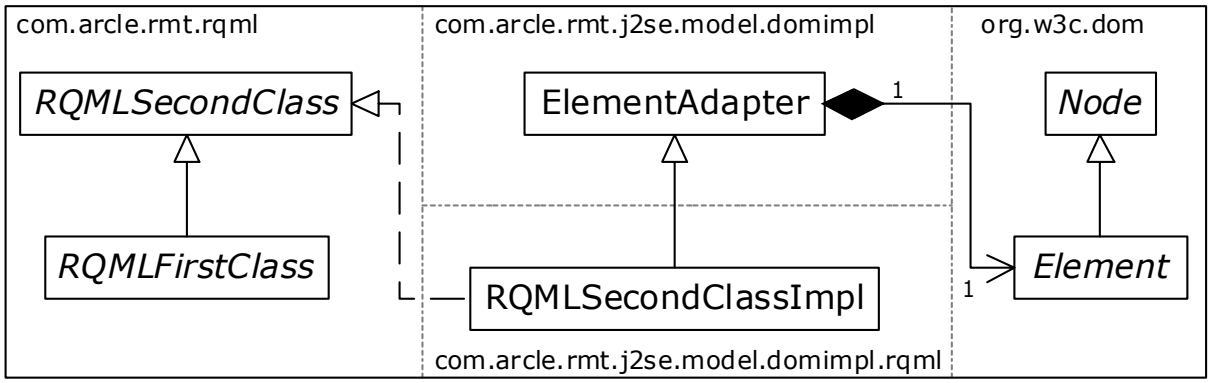
Gambar 7.12 Antarmuka serta implementasi *data model* RQML.

Nampak pada **Gambar 7.12**, hirarki paralel ini merupakan suatu penerapan dari *bridge pattern*. Berbeda dengan pola rancangan *bridge* yang termuat di [Gamm95], penjembatanan antar hirarki kelas tidak dilakukan dengan cara *composition* maupun *acquaintance* oleh *root class* dari salah satu hirarki. Penghubungan kedua hirarki dicapai oleh salah satu hirarki yang menjadi kelas-kelas implementasi dari hirarki yang lain, dengan hirarki yang kedua ini murni terdiri dari `interface`.

Data model aplikasi *desktop* terletak dalam *package* `com.arcle.rmt.j2se.model.domimpl.rqml`. Sedangkan *data model* aplikasi *handheld* terletak di `com.arcle.rmt.superwaba.model.imp.rqml`. Dalam hubungannya dengan himpunan antarmuka RQML pada *package* `com.arcle.rmt.rqml`, bentuk hirarki *data model* aplikasi *desktop* dengan rekannya di *handheld* tidaklah berbeda.

7.5.2 *Data model* aplikasi *desktop*

Kelas-kelas *data model* pada aplikasi *desktop* tidak menyimpan datanya sendiri, namun penyimpanan data RQML didelegasikan kepada obyek-obyek DOM yang dimiliki oleh *XML Parser*. Obyek-obyek ini adalah *instance* dari `interface` Element (di dalam *package* `org.w3c.dom`). Karena itu maka penerapan RQML di *desktop* adalah kelas-kelas *adapter* untuk antarmuka ini.



Gambar 7.13 *Adapter pattern* pada kelas representasi data untuk aplikasi *desktop*

Penerapan *adapter pattern* [Gamm95] dilakukan oleh kelas akar dari *data model* aplikasi *desktop* yaitu `ElementAdapter`. Kelas ini memiliki sebuah obyek DOM `Element` yang diadaptasikannya. *Instance* obyek DOM ini digunakan oleh kelas-kelas turunan untuk menyimpan data RQML. Masing-masing *derived class* mengadaptasikan suatu tipe elemen RQML tertentu sesuai dengan antarmuka RQML yang diterapkannya. Selain itu `ElementAdapter` memuat beberapa *method* pembantu untuk digunakan oleh turunan-turunannya.

Pada gilirannya, kelas `ElementAdapter` ini menjadi *base class* dari `RQMLSecondClassImpl`, yaitu penerapan dari *root interface* `RQMLSecondClass` untuk aplikasi *desktop*. Pararel dengan hirarki antarmuka RQML, semua penerapan elemen RQML adalah turunan dari kelas ini.

Sebagai bagian dari *document tree*, sebuah elemen XML dapat mempunyai satu atau lebih elemen-elemen anakan. Demikian pula dengan elemen RQML yang juga mempunyai *child elements*. Kelas `Element` menyediakan suatu mekanisme akses terhadap elemen-elemen anakannya. Umumnya *child nodes* ini juga merupakan instance dari kelas `Element`. Penerapan *composite pattern* ini dibawa dari konsep dokumen XML.

Agar *extensible*, aplikasi haruslah diabstraksikan dari implementasi *data model* yang menggunakan DOM. Akses terhadap data RQML haruslah melalui hirarki antarmuka RQML yang didefinisikan di *package* `com.arcle.rmt.rqml`. Sehingga apabila penerapan representasi data berubah, lapisan-lapisan aplikasi yang lebih tinggi tidak terkena riak dari perubahan ini.

Namun sebagai *adapter*, turunan-turunan dari `ElementAdapter` hanya dapat mengakses *instance* DOM dari elemen anakan yang dimilikinya; tetapi tidak dapat mengakses *adapter* dari *child element* tersebut. Kesulitan ini menjadi masalah ketika *client code* mengakses sebuah *child element*: obyek yang diberikan kepada aplikasi haruslah merupakan *instance* dari `RQMLSecondClass`, bukan *instance* dari kelas `Element`.

Karena itu, untuk setiap *child element* yang ditemui dibuatlah suatu obyek *adapter* baru. Obyek inilah yang diberikan kepada *client code*. Perilaku ini nampak pada implementasi *method* `getFirstChild()` di kelas `ElementAdapter`. *Method* ini digunakan untuk mendapatkan elemen anakan pertama yang mempunyai nama tertentu. Apabila *child element* itu tidak ditemukan maka akan dibuat secara otomatis sehingga *method* ini selalu berhasil. Terlihat pada **Gambar 7.14** di baris 247 dan 252 bahwa setiap kali sebuah *child element* ditemukan maka suatu obyek *adapter* baru diciptakan untuknya.

```
242 protected RQMLSecondClass getFirstChild(String name, boolean forceCreate) {
243     NodeList children = getElement().getChildNodes();
244     for (int i=0; i<children.getLength(); i++) {
245         Node child = children.item(i);
246         if (child instanceof Element && name.equals(child.getNodeName())) {
247             return getFactory().createRQMLSecondClass((Element) child);
248         }
249     }
250     // child not found
251     if (forceCreate) {
252         return createChild(name);
253     }
254     return null;
255 }
```

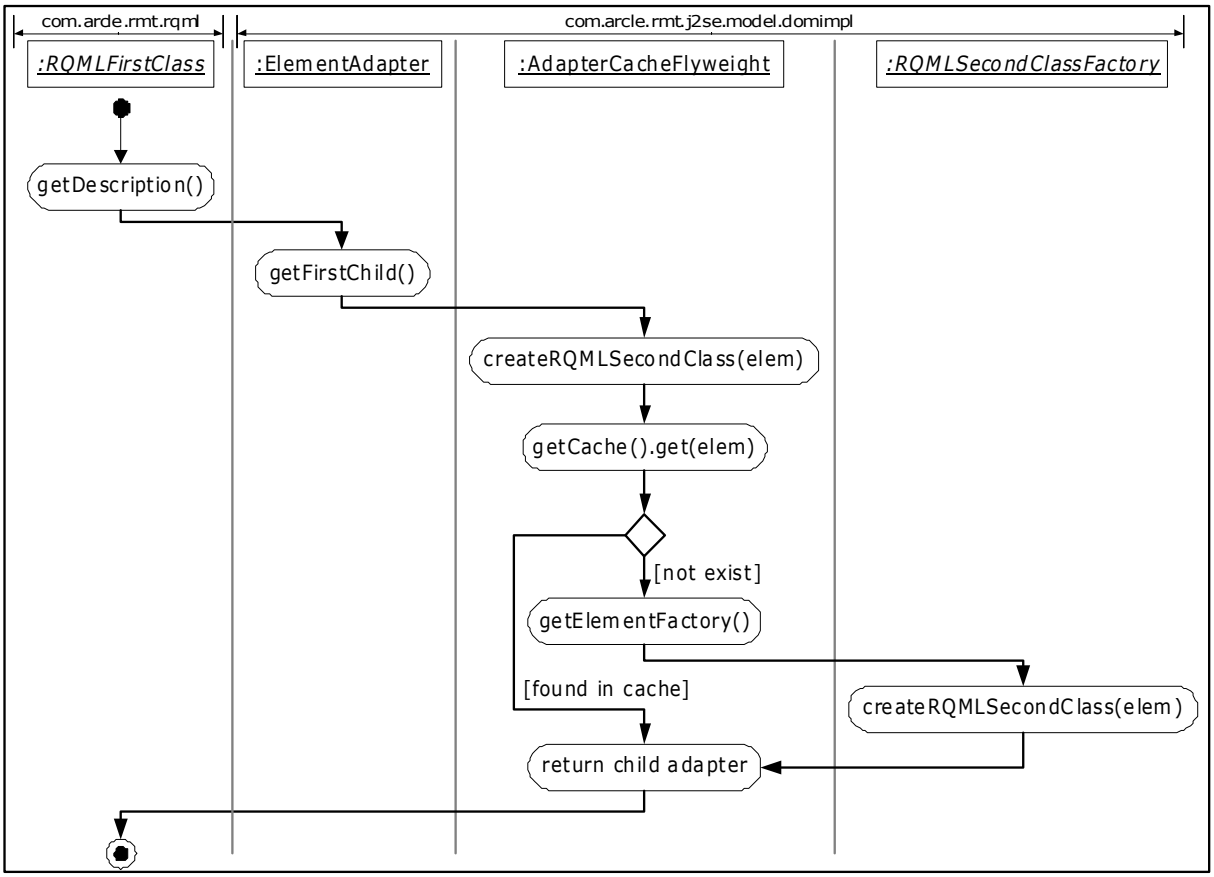
Gambar 7.14 File `ElementAdapter.java`, baris 242 – 255

Sayangnya cara ini menimbulkan suatu permasalahan baru. Penciptaan *instance adapter* secara *on-the-fly* ini akan mengakibatkan satu *instance* dari elemen DOM untuk mempunyai beberapa *instance adapter* yang berbeda. *Redundancy* ini tentulah akan meningkatkan kebutuhan *memory* secara signifikan. Belum lagi dengan adanya hubungan *one-to-many* antara obyek `Element` dengan *adapter*-nya, maka aplikasi tidak

dapat membandingkan kesamaan antara dua elemen hanya dengan membandingkan *instance* dari masing-masing *adapter*-nya.

Masalah ini dipecahkan oleh kelas `AdapterCacheFlyweight`. Kelas ini bertindak sebagai *abstract factory* kepada kelas-kelas *adapter* elemen DOM dengan menerapkan `interface RQMLSecondClassFactory`. Namun pada saat diminta untuk menciptakan suatu obyek *adapter*, kelas ini terlebih dahulu memeriksa apakah sebuah *adapter* untuk elemen yang bersangkutan telah pernah dibuat sebelumnya. Pemeriksaan dilakukan terhadap suatu *cache* yang merupakan pemetaan antara elemen-elemen DOM dengan *adapter*-nya masing-masing. *Instance* adapter yang terdapat di dalam *cache* akan diberikan ketimbang menciptakan suatu obyek *adapter* yang baru.

Letak dari penerapan *flyweight pattern* adalah pada penggunaan bersama dari setiap *instance adapter*. Karena para *client* dari *data model* melihat `AdapterCacheFlyweight` sebagai sebuah implementasi *abstract factory* lewat antarmuka `RQMLSecondClassFactory`, maka mereka 'merasa' menciptakan obyek-obyek *adapter* baru dengannya. Namun pada kenyataannya obyek *adapter* yang diberikan oleh `AdapterCacheFlyweight` lewat method `createRQMLSecondClass()` yang dideklarasikan oleh `RQMLSecondClassFactory` bukanlah selalu obyek baru.

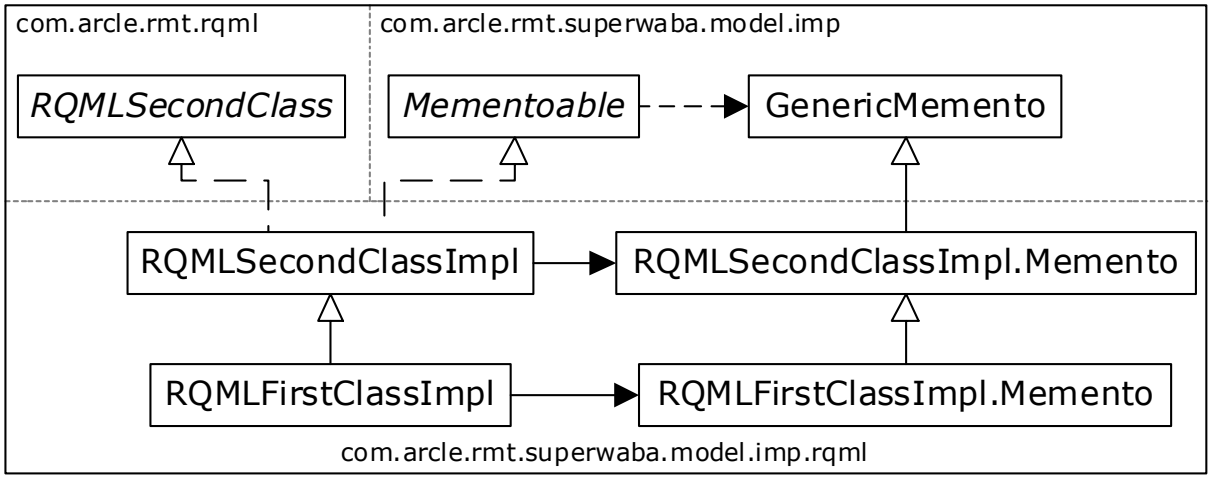


Gambar 7.15 Jalannya penciptaan obyek *adapter* oleh `AdapterCacheFlyweight`

Contoh proses ini dapat dilihat di **Gambar 7.15**. *Interface* `RQMLFirstClass` mempunyai suatu *child element* `Description` yang diakses lewat method `getDescription()`. Pada saat method ini dipanggil, kelas implementasinya akan memanggil *method* `getFirstChild()` yang diwariskan dari *superclass*-nya, `ElementAdapter`. Method `getFirstChild()` kemudian akan menggunakan *method* `createRQMLSecondClass()` dari *abstract factory* yang telah diberikan kepadanya untuk membuat *adapter* dari elemen anakan yang ditemukan. Agar dapat menciptakan *adapter*, maka *method* `createRQMLSecondClass()` diberikan obyek `Element` yang akan diadaptasikan. Sebagai *abstract factory* penerap *method* `createRQMLSecondClass()`, kelas `AdapterCacheFlyweight` akan memeriksa apakah sebuah *adapter* untuk elemen yang diminta telah ada di dalam *cache* untuk kemudian memberikan *instance* ini. Apabila belum ada maka sebuah *instance adapter* baru akan dibuat, ditambahkan ke dalam *cache*, dan kemudian diberikan ke pemanggilnya. Untuk operasi pembuatan *adapter* baru, kelas `AdapterCacheFlyweight` mempunyai sekumpulan *instance* `RQMLSecondClassFactory` yang melakukan operasi *instantiation* sebenarnya.

7.5.3 Representasi data aplikasi *handheld*

Penerapan *data model* RQML pada aplikasi *handheld* dilakukan dengan obyek-obyek biasa. Data RQML disimpan oleh obyek-obyek ini di dalam *instance variables*. Akses terhadap data ini dilakukan lewat berbagai *method* yang dideklarasikan oleh antarmuka-antarmuka RQML.



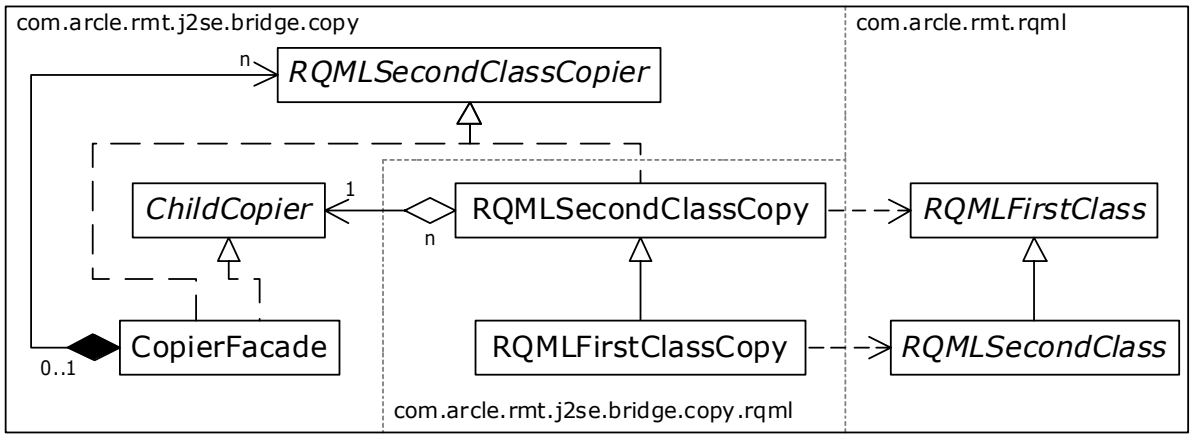
Gambar 7.16 *Memento pattern* pada kelas representasi data untuk aplikasi *handheld*.

Karena tidak menggunakan pustaka pengurai XML, maka penyimpanan data ke *persistent storage* di aplikasi *handheld* dicapai oleh *persistent objects* dengan penerapan pola rancangan *memento*. Masing-masing kelas implementasi RQML bertindak sebagai *originator* yang akan menciptakan sebuah obyek *memento* untuk operasi *load/save*. Obyek-obyek *memento* ini yang akan menyimpan atau memanggil kembali data *instance variables* obyek *originator*-nya dari/ke *persistent storage*. Dengan ini semua *instance variables* – baik yang memiliki *accessor method* maupun tidak – dapat diakses secara implisit tanpa melanggar enkapsulasi.

Pada **Gambar 7.16** terlihat bahwa penerapan *memento pattern* [Gamm95] pada *data model handheld* juga merupakan suatu variasi dari *bridge pattern*. Ada hirarki paralel antara kelas-kelas *memento* dengan *originator*-nya. Perbedaan dengan *bridge* biasa yaitu kelas-kelas *memento* ini masing-masing adalah *nested class* dari kelas *originator*-nya. Sebagai *member*, kelas-kelas tersarang ini memiliki akses penuh terhadap berbagai *method* maupun *instance variable* dari kelas yang melingkupinya – bahkan yang `protected` atau `private` sekalipun.

7.5.4 Penyalin data RQML

Seperti telah disebutkan di subbab 7.2.3, ada sekelompok kelas *Element Copiers* yang digunakan dalam proses *import/export* antara implementasi *data model* aplikasi *desktop* dengan *handheld*. Kelas-kelas penyalin ini digunakan untuk mentransfer data antara elemen-elemen RQML. Namun proses penyalinan hanya dapat mengandalkan *method-method* yang diberikan oleh antarmuka-antarmuka RQML tanpa mengakses kelas-kelas penerapannya. Batasan ini diberikan agar kelas-kelas penyalin ini dapat digunakan untuk berbagai penerapan representasi data RQML dan tidak terbatas pada implementasi kelas-kelas *data model* yang ada sekarang.



Gambar 7.17 Subsistem *Element Copiers*.

Masing-masing kelas *Element Copiers* berfungsi untuk menyalin satu jenis elemen RQML saja – dengan kata lain, kelas penyalin *x* hanya dapat mengoperasikan obyek yang mengimplementasikan `interface` RQML *y*. Karena suatu kelas penyalin juga harus memiliki operasi-operasi untuk menyalin semua *property* dari semua super-*interface* daripada *interface* yang diterimanya, maka kelas-kelas penyalin ini dibuat dalam suatu hirarki kelas yang paralel dengan hirarki antarmuka RQML.

Banyaknya elemen-elemen RQML mengakibatkan banyaknya pula kelas-kelas penyalinnya. Hal ini membuat proses penyalinan menjadi lebih sulit. Agar dapat mentransfer data antara dua obyek RQML perlu didapatkan kelas *copier* yang sesuai untuk menyalinnya. Untuk mempermudah penyalinan kelas-kelas implementasi dari `RQMLSecondClass` ini maka dibuatlah suatu kelas *facade* yang akan memilih *instance* penyalin yang sesuai dengan tipe RQML yang diberikan.

Penyederhanaan penggunaan subsistem *Element Copiers* oleh kelas `CopierFacade` adalah satu penerapan dari *facade pattern*. Seperti ditampilkan di **Gambar 7.17**, kelas ini memiliki obyek-obyek `RQMLSecondClassCopy` di mana masing-masing obyek mempunyai kemampuan untuk menyalin suatu tipe elemen RQML tertentu. Kelas `CopierFacade` sendiri juga menerapkan `RQMLSecondClassCopy`. Pada saat diminta untuk menyalin satu elemen RQML ke elemen yang lainnya, *facade* ini akan memeriksa tipe elemen RQML tersebut kemudian memilih *instance* kelas penyalin yang sesuai. Sehingga kelas-kelas yang menggunakan subsistem ini hanya perlu menggunakan `CopierFacade` tanpa perlu mengetahui keberadaan masing-masing kelas penyalin. Selain itu *facade* ini juga berlaku dua arah: setiap kelas penyalin menggunakan suatu *instance* `ChildCopier` untuk menyalin *child elements* dari suatu elemen; yang tidak lain adalah obyek `CopierFacade` itu sendiri.

Dengan adanya hirarki paralel maka kelas-kelas *Element Copiers* juga menerapkan *bridge pattern*. Namun karena hirarki penyalin ini tidak memiliki *reference* ke kelas-kelas antamuka RQML dan tidak pula menjadi kelas implementasinya, penjembatanan yang dilakukan tidak nampak secara struktural. Garis-garis *dependency* pada **Gambar 7.17** yang menghubungkan suatu kelas penyalin dengan antarmuka RQML yang dioperasikannya lebih menyatakan hubungan perilaku *run-time*.

7.5.5 Penerapan MVC di *Handheld*

Mengikuti paradigma MVC, aplikasi *handheld* dipecah menjadi tiga kelompok kelas: *Model*, *View*, dan *Controller*. Kelas-kelas yang termasuk *model* adalah kelas-kelas yang mengatur representasi data RQML di dalam *device*. Sedangkan *view* adalah kelas-kelas antarmuka pemakai dan *controller* adalah pengendali yang mengubah keadaan-keadaan dari suatu *view*.

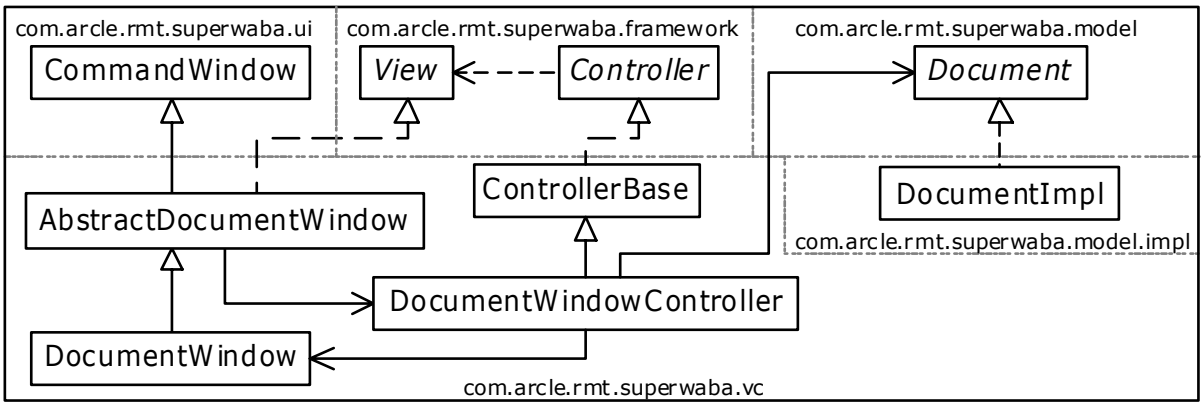
Pada [Gamm95] disebutkan bahwa hubungan antara *model* dengan *view* menerapkan *observer pattern*. Apabila data di dalam *model* berubah, maka masing-masing *view* yang menampilkan datanya akan dipanggil agar tampilannya merefleksikan perubahan data tersebut. Berhubung di dalam aplikasi *handheld* hanya ada satu *view* yang aktif (sedang menampilkan data di layar) dan sebuah komputer genggam umumnya *single-user*, pola rancangan *observer* dinilai tidak banyak berguna di situasi ini – karena itu tidak diimplementasikan. Setelah melakukan perubahan data maka *view*

hanya akan me-*refresh* tampilannya untuk merefleksikan data yang baru diperbaharui.

Pada SuperWaba, komponen-komponen antarmuka pemakai adalah kelas-kelas turunan (langsung ataupun tidak langsung) dari kelas `Control` (di dalam *package* `waba.ui`). Input dari pengguna diterima dalam bentuk *event* yang diterima oleh *method-method* tertentu di kelas ini. Kelas-kelas aplikasi harus meng-*override* berbagai *method* ini untuk dapat menangani berbagai *event* yang ada. Sebuah *event* bisa mewakili masukan yang *low-level* seperti *stylus tap* dan *keystroke*, tetapi juga dapat mewakili input yang bersifat *high-level* seperti *button click* ataupun pemilihan sebuah menu.

Misalkan pengguna mengetuk layar dengan *stylus*. SuperWaba VM akan mendeteksi kejadian ini kemudian akan membuat sebuah obyek `PenEvent` yang juga memuat koordinat layar tempat ketukan ini terjadi. Kemudian method `onEvent()` dari obyek `Control` yang mendapat ketukan akan dipanggil dengan parameter obyek `PenEvent` tersebut. Definisi method `onEvent()` pada kelas `Control` sendiri tidak melakukan apa-apa; method di-*override* oleh aplikasi untuk memproses *event* yang terjadi.

Pada arsitektur MVC yang asli sebagaimana diterapkan pada bahasa pemrograman SmallTalk [Burb92], sebuah *event* tidak diberikan kepada *view* di mana *event* tersebut terjadi. Melainkan sebuah obyek lain yang disebut *controller* akan menjadi penerima obyek *event* itu. Obyek *controller* inilah yang akan melakukan tindakan-tindakan yang akan mengubah keadaan dari *view*.



Gambar 7.18 Sebagian hirarki MVC pada aplikasi handheld.

Mengingat perbedaan arsitektur ini, maka di dalam aplikasi *handheld*, berbagai *method* penangan *event* seperti `onEvent()` akan memeriksa obyek `Event` yang diteri-

manya kemudian mendelegasikan sebagian besar pemrosesan terhadap obyek *controller* yang dimilikinya. Kelas-kelas antarmuka pemakai dari aplikasi *handheld* adalah turunan tidak langsung dari `Control` – umumnya turunan dari kelas `Container` atau `Window` (keduanya terletak di `waba.ui`). Masing-masing kelas antarmuka pemakai ini mempunyai satu obyek dari kelas *controller* yang khusus dibuat untuknya.

Seperti dapat di lihat di **Gambar 7.18**, kelas antarmuka pemakai `DocumentWindow` mempunyai kelas *controller*, `DocumentWindowController` yang obyeknya didapatkan dari *superclass*-nya, `AbstractDocumentWindow`. Sedangkan kelas `AbstractDocumentWindow` merupakan turunan dari `CommandWindow`, yang pada gilirannya diturunkan dari `Window` (kelas tidak digambarkan). Karena *controller* `DocumentWindowController` mempunyai reference ke obyek data model, `Document`, maka `DocumentWindow` juga dapat mengakses obyek *model* ini melalui *controller*-nya.

Kelas `DocumentWindow` menerapkan tampilan layar *Element List*, yang menampilkan daftar dari elemen-elemen RQML dengan tipe tertentu. Seperti telah disebutkan di bagian 7.4.1, layar ini mempunyai beberapa sub-layar yang masing-masing digunakan untuk menampilkan daftar elemen dengan tipe tersendiri. Hanya ada satu dari sub-layar ini yang aktif sehingga pada setiap saat hanya elemen-elemen RQML *first-class* yang sejenis yang ditampilkan.

```
00141 public void onEvent(Event event) {
00142     DocumentWindowController ctrl = getController();
00143     switch(event.type) {
00144     case ControlEvent.PRESSED:
00145         if (event.target == btnNew) {
00146             doNewElement();
00147             event.consumed = true;
00148         } else if (event.target == btnClose) {
00149             ctrl.cmdCloseWindow();
00150         } else if (event.target == btnDelete) {
00151             ctrl.cmdDeleteDocument();
00152         }

00241 protected void doNewElement() {
00242     ElementListController ctrl = ((ElementList)getActivePanel())
00243         .getController();
00244     ctrl.cmdNewElement();
00245 }
```

Gambar 7.19 *File* `AbstractDocumentWindow.java`, baris 141 – 152 dan 241 – 245.

Pada cuplikan di **Gambar 7.19** dapat dilihat bahwa *method* `onEvent()` dari `Abstract-DocumentWindow` menerjemahkan *event-event* dari SuperWaba menjadi berbagai *method call* untuk *controller*-nya. Pada baris 244 di dalam *method* `doNewElement()`, *controller*

yang digunakan untuk menindaklanjuti event bukanlah miliknya sendiri, namun *controller* dari salah satu *panel* yang dimilikinya. Obyek turunan dari `ElementList` inilah yang menampilkan daftar elemen RQML first-class dengan tipe tertentu.

7.5.6 Penerapan *Separable Model* di *Desktop*

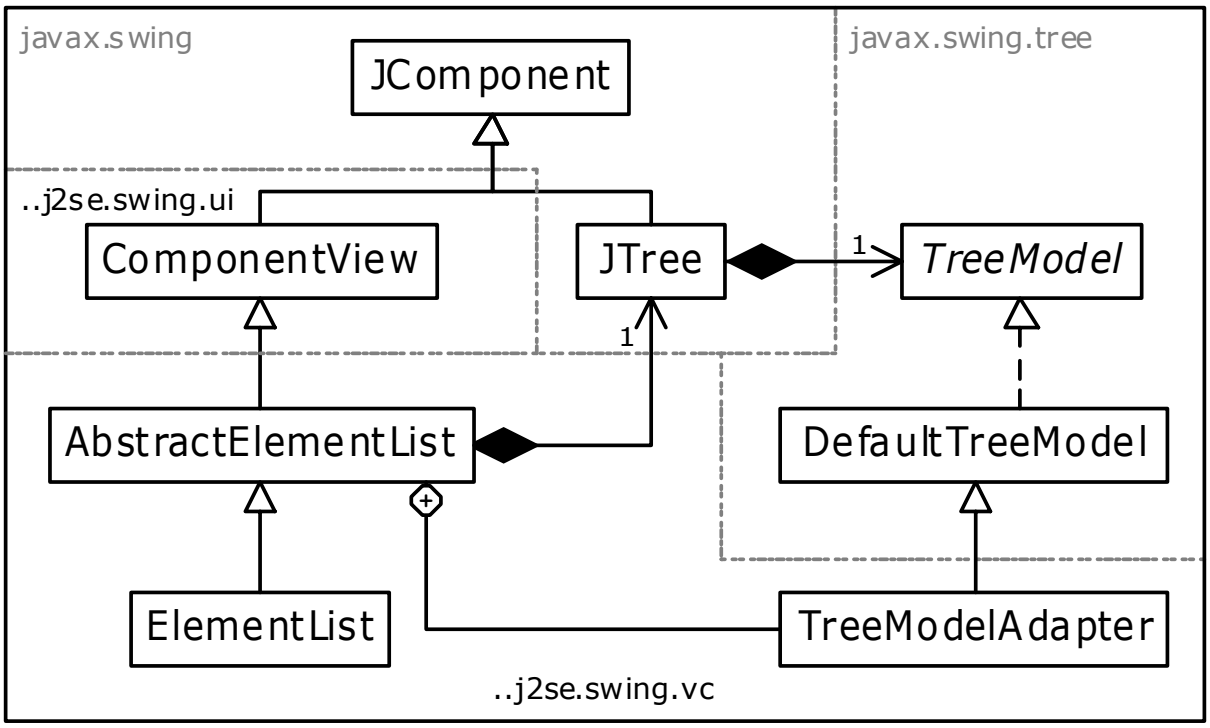
Di dalam *separable model architecture* yang diterapkan oleh Swing, masing-masing komponen Swing mempunyai obyek *model* tersendiri [Fowl02]. Obyek-obyek *model* ini adalah suatu kelas yang mengimplementasikan `interface` tertentu, tergantung dari komponen Swing yang menggunakannya. Untuk setiap *interface* obyek model ini ada kelas *default* yang menjadi implementasinya. Demikian halnya juga ada obyek data model *default* yang dimiliki oleh masing-masing obyek Swing. Umumnya obyek *default* ini dapat diganti lewat method `setModel()`.

Karena mengikuti *problem domain* dari *requirements engineering*, maka *data model* RQML tentulah berbeda dengan yang diinginkan oleh Swing. Berbagai antarmuka RQML, beserta kelas-kelas pendukungnya dirancang tanpa menggunakan kelas maupun *interface* Swing. Sebagian faktor yang mendorong keputusan rancangan ini yaitu agar kelas-kelas representasi data bisa digunakan untuk jenis *user interface* lain-nya[19]. Selain itu juga agar mengikuti salah satu filsafat rancangan perangkat lunak: *data model* yang *user-interface agnostic*.

Perbedaan representasi data ini membuat suatu kebutuhan bagi kelas-kelas *adapter* untuk menyesuaikan representasi data RQML menjadi *data model* yang dibutuhkan oleh Swing. Sebagai pustaka *user interface*, kelas-kelas yang menggunakan Swing umumnya adalah kelas-kelas *view* – yang notabene juga memegang satu atau lebih *instance* dari kelas representasi data RQML. Karena itu kelas-kelas *adapter* ini umumnya diimplementasikan sebagai *inner class* [Sun03b] dari kelas *view* yang bersangkutan. Karena satu *view* dapat mempunyai beberapa komponen Swing dengan kebutuhan *data model*-nya masing-masing, maka kelas *view* tersebut dapat mempunyai lebih dari satu *inner class* yang menjadi adapter representasi data untuk komponen-komponen Swing yang dimilikinya. *Adapter data model* sebagai *inner class* ini merupakan salah satu variasi lagi dari penerapan *adapter pattern* [Gamm95].

---

[19] Antarmuka pemakai seperti JSP atau bahkan antarmuka mesin seperti EJB dan Web Services.



Gambar 7.20 Penerapan *separable model architecture* pada aplikasi desktop.

Namun tidak semua kelas *view* mempunyai *adapter* untuk masing-masing komponen Swing di dalamnya. Pada beberapa kasus lebih sederhana untuk menggunakan *data model default* yang diberikan oleh Swing. Misalnya saja pada komponen *text field* yang hanya memuat satu string. Implementasi kelas *adapter* untuk menterjemahkan satu attribut RQML menjadi *data model* bagi *text field* ini tentulah lebih kompleks ketimbang hanya menggunakan *method-method* `setText()` dan `getText()` yang diberikan oleh sebuah *text field* untuk mengakses datanya. Karena itu kelas *adapter* hanya dibuat untuk representasi data yang kompleks, seperti misalnya untuk komponen `JTree`.

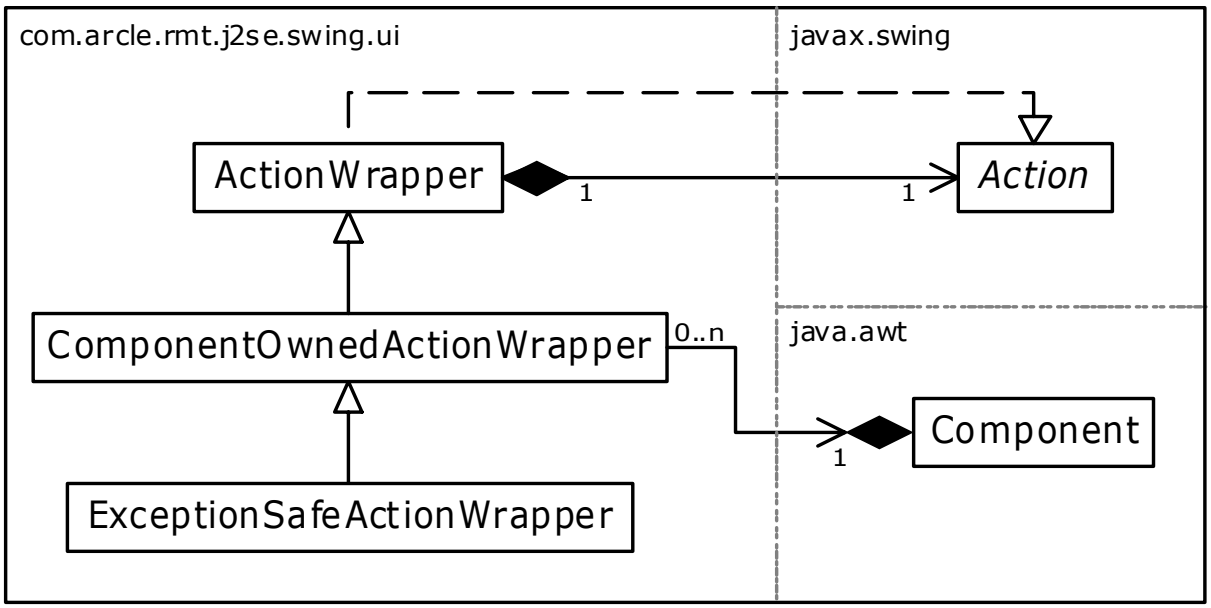
Pada **Gambar 7.20** dapat dilihat sebagian dari penerapan *adapter* representasi data pada aplikasi *desktop*. Kelas `ElementList` menerapkan komponen *Document Tree* beserta *Active Elemen Description* yang diletakkan di sebelah kiri jendela aplikasi *desktop* (lihat subbab 7.4.2). Untuk komponen *Document Tree* digunakan sebuah *instance* dari `JTree` (dari *package* `javax.swing`) yang menampilkan elemen-elemen RQML beserta tipenya. Tampilan yang dihasilkan oleh `JTree` ini dihasilkan oleh kelas representasi data `TreeModelAdapter`, yang merupakan *inner class* dari `AbstractElementList` (Notasi $\oplus$ untuk *inner class* diadaptasi dari [Syba02]). Sedangkan `AbstractElementList` adalah *superclass* dari `ElementList` yang mewariskan komponen *tree* yang dimilikinya. Lewat *superclass*-nya, kelas `TreeModelAdapter` mewarisi

interface `TreeModel`. Antarmuka inilah yang diharapkan oleh `JTree` dari implementasi *data model*-nya.

7.5.7 *Command Decorator*

Beberapa komponen Swing seperti *button* dan *menu* menerapkan *command pattern* [Gamm95] di dalam hubungannya dengan *application code*. Setiap obyek-obyek penerima perintah ini memiliki suatu obyek `Action` di mana method `actionPerformed()` yang dimilikinya akan dipanggil pada saat komponen yang bersangkutan di-*trigger* oleh pengguna.

Perilaku *default* Swing pada saat sebuah *exception* terlempar keluar (*unhandled exception*) dari *event handler* pada sebuah aplikasi – termasuk di dalam *method* `actionPerformed()` – adalah menampilkan *exception* tersebut beserta *stack trace*-nya pada *standard output*. Berhubung aplikasi Swing umumnya adalah program-program GUI yang tidak mempunyai *console*, hal ini dapat menjadi masalah: tampilan *exception* tidak nampak oleh pengguna dan seakan-akan perintah yang mengalami *exception* akan berhenti secara misterius tanpa ada *feedback* pada tampilannya – tampilan yang diberikan di *console* tidak terlihat pada *user interface* program GUI.



Gambar 7.21 Penerapan *decorator pattern* oleh kelas `ExceptionSafeActionWrapper`

Masalah penampilan *exception* ini ditangani dengan penerapan *decorator pattern* [Gamm95]. Kelas `ExceptionSafeActionWrapper` merupakan pembungkus untuk sebuah

obyek `Action` yang menangani *unhandled exception* yang mungkin terjadi. Pada saat sebuah obyek `Exception` terlempar ke luar dari method `actionPerformed()` dari obyek yang dibungkus, *method* `actionPerformed()` dari obyek `ExceptionSafeActionWrapper` yang membungkusnya akan menangani *exception* tersebut kemudian menampilkan sebuah *dialog box* untuk menampilkan *exception* yang terjadi.

Kelas `ExceptionSafeActionWrapper` adalah subclass dari `ComponentOwnedAction-Wrapper`. *Superclass* ini berfungsi untuk menyimpan obyek `Component` yang akan menjadi *parent* dari *dialog box* yang akan ditampilkan oleh `ExceptionSafeAction-Wrapper`.

Pada gilirannya, `ComponentOwnedActionWrapper` adalah *subclass* dari `ActionWrapper`, yang merupakan *decorator* umum untuk obyek-obyek yang menerapkan *interface* `Action`. Kelas *decorator* ini tidak berfungsi apa-apa dengan sendirinya selain untuk menjadi *base class* bagi kelas-kelas *decorator* lainnya.

```
00093 public void actionPerformed(ActionEvent e) {
00094     try {
00095         super.actionPerformed(e);
00096     } catch(Exception ex) {
00097         MoreSwingUtilities.showException(getOwnerComponent(), ex,
00098             "Unhandled exception caught in action: " + getValue(NAME));
00099     }
00100 }
```

Gambar 7.22 *File* `ExceptionSafeActionWrapper.java`, baris 93 – 100.

Pada **Gambar 7.22** dapat dilihat bahwa *method* `actionPerformed()` dari `Exception-SafeActionWrapper` menangkap suatu *exception* yang mungkin terjadi pada saat menjalankan suatu *command*. Di dalam blok `try..catch`, *method* ini hanya memanggil *method* yang sama dari *superclass*-nya. Sebagai sebuah *wrapper* kosong, method *superclass* kemudian hanya meneruskan *method call* ke obyek yang dibungkusnya. Apabila sebuah *exception* terjadi maka *exception* tersebut beserta *stack trace*-nya akan ditampilkan di sebuah kotak dialog.

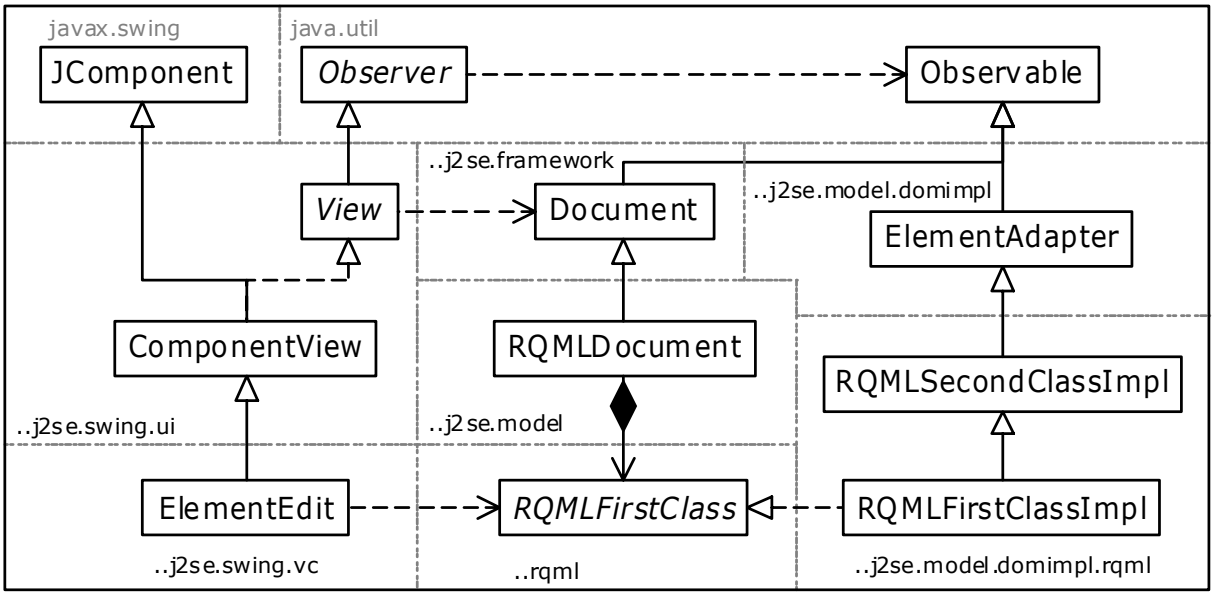
Namun masalah penampilan *exception* tadi masih belum terpecahkan seluruhnya. Mengingat tidak hanya subclass dari `Exception` yang dapat dilempar, namun kelas `ExceptionSafeActionWrapper` tidak menangkap *exception object* selain tipe ini – misalkan subclass dari `Error`. Sehingga sebuah *unhandled exception* dengan tipe ini tetap akan mendapatkan perilaku *default* Swing: ditampilkan di *console*. Sayangnya, dokumentasi JDK API mengatakan bahwa *subclass* dari `Error` tidak boleh ditangkap

oleh program aplikasi [Sun00a]. Larangan ini menjadi alasan bahwa hanya obyek `Exception` yang ditangkap oleh `ExceptionSafeActionWrapper`. Apakah dalam kasus khusus ini diperbolehkan untuk menangkap obyek `Error` masih menjadi pertanyaan.

7.5.8 *Editor* elemen-elemen *first-class*

Layar komputer *desktop* yang umumnya lebih luas memungkinkan aplikasinya untuk secara wajar mengaktifkan beberapa *view* pada saat yang sama. Penampilan beberapa *view* secara bersamaan ini menjadi alasan yang baik bagi penerapan *observer pattern* [Gamm95]. Apalagi dengan adanya dukungan dari pustaka J2SE – pada package `java.util` terdapat kelas `Observable` serta *interface* `Observer` yang telah mengimplementasikan sebagian besar algoritma yang diperlukan bagi penerapan pola rancangan ini.

Semua kelas *view* pada aplikasi desktop mengimplementasikan *interface* `Observer`. Sedangkan kelas-kelas *model* diturunkan dari obyek `Observable`. Ketika sebuah *view* sedang menampilkan data dari obyek *model* tertentu, *view* tersebut mendaftarkan diri sebagai pengamat dari obyek model yang ditampilkan itu. Sehingga apabila data dari *model* berubah – terutama akibat manipulasi dari komponen-komponen antarmuka pemakai lainnya – *view* tersebut akan mendapat pemberitahuan tentang perubahan itu sehingga dapat memperbaharui tampilannya.



Gambar 7.23 *Observer pattern* pada dialog penyunting RQML

Dapat dilihat dari **Gambar 7.23** bahwa obyek `Observable` meliputi kelas-kelas penerapan RQML – turunan dari `ElementAdapter` (subbab 7.5.2) – dan juga kelas `RQMLDocument` yang mewakili dokumen RQML secara keseluruhan. Sebagaimana sebuah dokumen RQML dengan elemen-elemennya, kelas `RQMLDocument` memiliki obyek-obyek `RQMLFirstClass` yang mewakili elemen-elemen pada dokumen tersebut.

Sebagai sebuah editor RQML, rancangan aplikasi dipusatkan pada sekelompok *dialog* yang berfungsi untuk meng-*edit* berbagai first-class RQML element yang ada (subbab 7.4.2). Apalagi dengan dukungan layar yang lebih besar yang memungkinkan beberapa *dialog* penyunting ini untuk ditampilkan secara bersamaan. Penerapan *observer pattern* memungkinkan hasil penyuntingan dari dialog yang satu untuk ditampilkan pada dialog yang lain.

Pada **Gambar 7.23** juga ditampilkan sebuah kelas *view*, yaitu `ElementEdit` yang menjadi *base class* dari semua jendela editor RQML. Kelas ini menerapkan *interface* `Observer`. Pada saat sebuah jendela *editor* sedang menyunting suatu elemen RQML, ia akan mendaftarkan diri sebagai pengamat dari elemen yang sedang disuntingnya. Sehingga pada saat suatu *editor* meng-*commit* perubahan pada elemen yang sama, *editor* yang lain akan langsung menampilkan perubahan ini.

Semua *dialog editor* RQML ini diturunkan dari kelas `ElementEdit`. Untuk setiap jenis elemen *first-class* terdapat satu kelas *editor* yang digunakan untuk menyunting elemen-elemen dengan tipe tersebut. Misalkan untuk elemen `Requirement` ada kelas `RequirementEdit`; sedangkan untuk elemen *first-class* dengan jenis `Usecase`, *editor*-nya adalah `UsecaseEdit`. Baik `RequirementEdit` maupun `UsecaseEdit` diturunkan secara tidak langsung dari `ElementEdit` (hirarki ini tidak ditampilkan untuk menghemat tempat). Garis *dependency* antara `ElementEdit` dan `RQMLFirstClass` menunjukkan hubungan antara suatu obyek penyunting RQML dengan elemen yang sedang di-*edit* dilakukan di *base class* obyek penyunting tersebut, yaitu `ElementEdit`.

7.5.9 *Class Map*

Adanya hirarki paralel antara antarmuka-antarmuka RQML beserta kelas-kelas aplikasi menciptakan suatu kebutuhan untuk menjalankan suatu *method* dari obyek tertentu berdasarkan *interface* yang diimplementasikan oleh obyek lain. Dengan kata lain, untuk suatu himpunan *interface* $A = \{a_1, a_2, ..., a_n\}$, suatu himpunan obyek

$B = \{b_1, b_2, ..., b_n\}$ yang semua obyek ini adalah dari kelas C atau kelas turunannya, dan sebuah obyek X yang menerapkan salah satu *interface* $a_i \in A$, seringkali dibutuhkan untuk menjalankan suatu *method* $m \in C$ terhadap obyek $b_j \in B$ di mana $i = j$.

Karena kebutuhan ini, maka pada pertengahan pengembangan aplikasi *desktop* banyak dijumpai penggalan kode yang berbentuk seperti berikut ini:

```
if (x instanceof A)
    p.doSomething(x);
else if (x instanceof B)
    q.doSomething(x);
else if (x instanceof C)
    r.doSomething(x);
...
```

Suatu masalah yang tidak kecil timbul karena untaian `if` seperti di atas tidak hanya melibatkan tiga atau empat obyek saja; terkadang malahan sampai tujuh atau lebih pernyataan `if` yang terlibat. Belum lagi himpunan *interface* yang digunakan seringkali sama di beberapa tempat yang berbeda. Hal ini tentu saja menjadi *redundant code* dan menurunkan modularitas program.

Karena itu penggalan di atas kemudian di-*refactor* ke dalam kelas `GenericClassMap`. Kelas ini digunakan untuk memetakan suatu himpunan kelas (ataupun *interface*) kepada suatu himpunan obyek. Dengan demikian, kode di atas dapat dipersingkat menjadi seperti ini:

```
GenericClassMap cm = new ThingMap();
y = cm.findFirstInstance(x);
y.doSomething(x);
```

Method `findFirstInstance()` dari `GenericClassMap` berfungsi untuk menemukan obyek yang dipetakan oleh salah satu *interface* yang diimplementasikan oleh obyek *x*. Apabila ditemukan, maka obyek terpetakan tersebut akan diberikan kepada pemanggilnya – yang pada contoh di atas adalah obyek *y*. Ditemukannya obyek yang dipetakan ini memungkinkan pemanggilan berbagai *method* yang dimilikinya.

Dalam operasinya, `findFirstInstance()` bergantung pada sebuah pemetaan antara kelas dan obyek. Pemetaan ini diberikan oleh *abstract factory method* `createMappings()` pada `GenericClassMap` yang diharuskan untuk menghasilkan sebuah obyek `Map` yang terdiri dari pasangan-pasangan terurut (`Class`, `Object`). Karena `createMappings()` tidak hanya diharapkan untuk menciptakan sebuah obyek namun

juga untuk mengisinya (*initialize*), maka *method* ini dapat dikatakan juga menerapkan *strategy pattern* [Gamm95].

Kelas-kelas turunan `GenericClassMap` mendefinisikan *method* ini untuk menciptakan pemetaan yang diinginkan. Untuk masing-masing himpunan pasangan (`Class`, `Object`) perlu diciptakan sebuah *subclass* dari kelas ini. Pada contoh di atas, kelas turunan yang digunakan adalah `ThingMap`, yang mendefinisikan pemetaan seperti berikut ini untuk menggantikan contoh untaian `if` yang pertama tadi:

```
public class ThingMap extends GenericClassMap {
    protected Map createMappings() {
        Map m = new Hashtable();
        m.put(A.class, p);
        m.put(B.class, q);
        m.put(C.class, r);
        return m;
    }
}
```

Memang nampaknya panjang definisi *method* `createMappings()` akan hampir sama dengan untaian `if` yang sebelumnya. Namun penghematan LoC[20] akan dicapai apabila perlu dibuat dua atau lebih untaian `if` yang keduanya memproses suatu obyek terhadap sekelompok antarmuka/obyek yang sama. Selain itu ada tambahan keuntungan di mana himpunan *interface*/*object* yang digunakan dapat di-*configure* pada saat *run-time* – suatu hal yang sulit dilakukan dengan untaian `if`.

Kelas `GenericClassMap` digunakan di beberapa tempat, seperti:

- `ElementEditFactoryFacade` – Sebuah *factory class*, *facade*, dan juga *singleton* yang digunakan untuk menciptakan *instance first-class editor* (*subbab* 7.5.8) yang sesuai dengan obyek *first-class element* yang akan disunting.
- `ElementTypeNodeFacade` – *Nested facade class* dari `AbstractElementList` (subbab 7.5.6) yang perlu mendapatkan sebuah *node* yang dimiliki oleh sebuah `JTree` berdasarkan tipe dari suatu *first-class element*.

7.5.10 *First-Class Importer*

Kelas *data model* yang mewakili data di aplikasi *desktop* berbeda dengan yang ada di aplikasi *handheld*. Pada kelas `RQMLDocument` di *desktop*, semua elemen *first-class* dapat

[20] LoC: Lines of Code

diakses secara acak. Sedangkan pada kelas `Document` di aplikasi *handheld*, elemen-elemen *first-class* dikelompokkan menurut tipenya masing-masing dan akses acak hanya dapat dilakukan terhadap suatu elemen dengan tipe tertentu.

Karena perbedaan ini, maka proses impor dari data *handheld* ke *desktop* dilakukan oleh beberapa kelas, di mana setiap kelas pengimpor berfungsi untuk memindahkan data dari salah satu jenis elemen *first-class*. Namun proses yang dilakukan oleh kelas-kelas ini umumnya sama. Untuk setiap jenis elemen *first-class* di *handheld* dilakukan iterasi berikut pada semua elemen dengan tipe tersebut:

1. Ambil elemen dari *instance* `Document`.
2. Buat elemen dengan tipe yang sama di dalam *instance* `RQMLDocument`.
3. Salin isi dari elemen yang didapat pada langkah 1 ke dalam elemen yang dibuat di langkah 2.

Walau pada dasarnya proses di atas sama bagi tiap jenis elemen *first-class*, namun *method-method* yang dipanggil oleh setiap langkah tersebut berbeda. Misalkan untuk elemen `Requirement`, langkah 1 memanggil method `getRequirementAt()` dan langkah 2 memanggil method `createRequirement()`. Sedangkan untuk elemen `Usecase`, langkah 1 memanggil method `getUsecaseAt()` dan langkah 2 memanggil *method* `createUsecase()`.

Kesamaan algoritma ini diterapkan menjadi suatu *template method* [Gamm95] pada *base class* untuk kelas-kelas pengimpor. *Template method* ini menggunakan beberapa *strategy method* yang abstrak untuk melakukan langkah-langkah spesifik yang dibutuhkan.

```
063 import com.arcle.rmt.superwaba.model.Document;
064 import com.arcle.rmt.j2se.model.RQMLDocument;
065
073 public abstract class FirstClassImport implements FirstClassImporter {
074
089     public boolean importFirstClass(Document srcDoc, RQMLDocument dstDoc) {
090         int numElems = getTotalElements(srcDoc);
091         boolean ok = true;
092         for (int i=0; i<numElems; i++) {
093             RQMLFirstClass srcElem = getElementAt(i, srcDoc);
094             RQMLFirstClass dstElem = createDestinationElement(dstDoc);
095             ok &= copyElement(srcElem, dstElem);
096         }
097         return ok;
098     }
116     protected boolean copyElement(RQMLFirstClass src, RQMLFirstClass dst) {
117         return CopierFacade.getInstance().copy(getSupportedClass(), src, dst);
118     }
132     protected abstract Class getSupportedClass();
141     protected abstract int getTotalElements(Document doc);
151     protected abstract RQMLFirstClass getElementAt(int index, Document srcDoc);
160     protected abstract RQMLFirstClass createDestinationElement(RQMLDocument doc);
161 }
```

Gambar 7.24 Cuplikan `FirstClassImport.java`

Pada **Gambar 7.24** dapat dilihat cuplikan *base class* dari kelas-kelas pengimpor, yaitu `FirstClassImport`. *Method* `importFirstClass()` adalah *template method* yang melakukan langkah-langkah umum pengimporan sebuah elemen *first-class*. *Method* ini bergantung pada beberapa *strategy method*, `getSupportedClass()`, `getTotalElements()`, `getElementAt()`, dan `createDestinationElement()`.

```
074 public class RequirementImport extends FirstClassImport {
079     protected Class getSupportedClass() {
080         return Requirement.class;
081     }
087     protected int getTotalElements(Document doc) {
088         return doc.getRequirementCount();
089     }
097     protected RQMLFirstClass getElementAt(int index, Document doc) {
098         return doc.getRequirementAt(index);
099     }
106     protected RQMLFirstClass createDestinationElement(RQMLDocument doc) {
107         return doc.createRequirement();
108     }
109 }
```

Gambar 7.25 Cuplikan `RequirementImport.java`.

Pada **Gambar 7.25** dapat dilihat salah satu turunan dari `FirstClassImport`. Kelas `RequirementImport` digunakan untuk mengimpor elemen-elemen dengan tipe `Requirement` dari *data model* handheld ke desktop. Dari cuplikan kode ini terlihat implementasi dari berbagai *strategy method* yang dibutuhkan oleh `importFirst-Class()`. *Method* `getSupportedClass()` menyebutkan tipe elemen *first-class* yang didukung oleh kelas pengimpor ini. *Method* `getRequirementCount()` memberikan jumlah *requirement* yang ada. *Method* `getElementAt()` mendapatkan suatu elemen

requirement pada indeks tertentu. Sedangkan *method* `createDestinationElement()` membuat suatu *instance* obyek `Requirement` pada *data model* aplikasi desktop.

7.6 Data Statistik

Pada bagian ini diberikan data *source code metric* yang dihasilkan dari Rambutan versi terakhir tertanggal 19 Mei 2003. Hitungan ini dilakukan oleh perangkat lunak JavaNCSS [Lee02]. JavaNCSS menggunakan besaran-besaran berikut dalam menghasilkan *source code metric*:

- **NCSS: Non-Commenting Source Statements** – Jumlah *statement* di dalam *source code* yang bukan komentar. Pernyataan yang dihitung bukan hanya *executable statements*, namun juga deklarasi *method*, deklarasi *class*, dan pernyataan `import`. Pada dasarnya, NCSS menghitung jumlah karakter '`{`' dan '`;`' pada *source code*.
- **CCN: Cyclomatic Complexity Number** – Juga dikenal sebagai *McCabe metric*, merupakan jumlah percabangan yang ada di dalam suatu *method*. Untuk suatu *method*, awalnya CCN bernilai satu. Kemudian nilai ini akan ditambahkan untuk setiap *statement* percabangan seperti `if`, `while`, `do`, dan case. Terminasi fungsi secara prematur (statement `return` di tengah-tengah *method*) juga akan menambah nilai CCN – namun statement `return` di akhir *method* tidak dihitung.
- **JVDC: JavaDoc Comments** – Untuk setiap *method* nilai ini bernilai satu apabila *method* tersebut didokumentasikan dengan JavaDoc dan bernilai nol apabila tidak didokumentasikan.

Tabel 7.1 Ikhtisar hitungan NCSS untuk seluruh program

Packages	Classes	Functions	NCSS	Javadocs	per
28.00	328.00	1354.00	7903.00	1002.00	Project
	11.71	48.36	282.25	35.79	Package
		4.13	24.09	3.05	Class
			5.84	0.74	Function

Tabel 7.2 Ikhtisar hitungan NCSS per *class*

Average Object NCSS	19.24
Average Object Functions	4.13
Average Object Inner Classes	0.34
Average Object Javadoc Comments	3.05

Program NCSS	7,903.00

Tabel 7.3 Ikhtisar hitungan NCSS per *method*

Average Function NCSS	3.34
Average Function CCN	1.29
Average Function JVDC	0.38
Program NCSS	7,903.00

Tabel 7.4 Hitungan NCSS per *package*

Nr.	Classes	Functions	NCSS	Javadocs	Package
1	2	11	73	11	com.arcle.rmt.j2se.bridge
2	3	8	85	12	com.arcle.rmt.j2se.bridge.copy
3	33	68	396	33	com.arcle.rmt.j2se.bridge.copy.rqml
4	2	6	36	7	com.arcle.rmt.j2se.bridge.swexport
5	10	23	108	33	com.arcle.rmt.j2se.bridge.swexport.rqml
6	2	6	41	8	com.arcle.rmt.j2se.bridge.swimport
7	10	43	158	53	com.arcle.rmt.j2se.bridge.swimport.rqml
8	2	9	24	7	com.arcle.rmt.j2se.framework
9	7	23	66	16	com.arcle.rmt.j2se.model
10	6	58	339	63	com.arcle.rmt.j2se.model.domimpl
11	34	86	561	74	com.arcle.rmt.j2se.model.domimpl.rqml
12	3	64	617	64	com.arcle.rmt.j2se.swing
13	14	79	360	68	com.arcle.rmt.j2se.swing.ui
14	6	64	633	93	com.arcle.rmt.j2se.swing.vc
15	24	100	951	52	com.arcle.rmt.j2se.swing.vc.rqml
16	8	31	141	35	com.arcle.rmt.j2se.util
17	38	46	180	68	com.arcle.rmt.rqml
18	2	4	27	3	com.arcle.rmt.superwaba
19	3	1	7	4	com.arcle.rmt.superwaba.framework
20	3	56	75	24	com.arcle.rmt.superwaba.model
21	5	86	435	49	com.arcle.rmt.superwaba.model.imp
22	33	116	892	44	com.arcle.rmt.superwaba.model.imp.rqml
23	6	48	228	44	com.arcle.rmt.superwaba.ui
24	1	19	63	2	com.arcle.rmt.superwaba.util
25	11	83	436	54	com.arcle.rmt.superwaba.vc
26	54	190	923	59	com.arcle.rmt.superwaba.vc.rqml
27	1	4	7	1	com.arcle.rmt.xplat.model
28	5	22	41	21	com.arcle.rmt.xplat.util
Total	328	1354	7903	1002	

Jangka waktu pembuatan: 2 bulan (6 April 2003 – 19 Mei 2003).

Programmer: 1 orang.

Bab 8

PENGUJIAN

Pada bab ini diberikan hasil-hasil pengujian terhadap Rambutan. Pengujian ini dilakukan lewat tiga sudut pandang:

- Pemenuhan terhadap *requirements* dan *constraints* yang ada di bab 6 (halaman 65).
- Kesesuaian terhadap standar RQML sebagaimana diusulkan di [Gudg00].
- Perbandingan dengan fitur-fitur yang umum dimiliki oleh berbagai alat bantu sejenis.

8.1 Terhadap Spesifikasi Kebutuhan

Bagian ini membahas pemenuhan Rambutan terhadap berbagai *requirement* yang dituliskan di subbab 6.2 (halaman 71). Seluruh *requirement* yang ada dituliskan kembali diikuti dengan hasil pengujiannya.

8.1.1 Sistem membantu dalam pengumpulan informasi *requirements*

Aplikasi *handheld* membantu pengumpulan informasi *requirements* yang dilakukan di luar kantor *system analyst* (pengguna Rambutan) di mana umumnya peralatan yang dapat digunakan lebih terbatas. Sedangkan aplikasi *desktop* dapat digunakan untuk operasi *editing* yang lebih intensif.

Pemenuhan: Penuh.

8.1.2 Sistem membantu dalam pengelolaan informasi *requirements*

Di dalam *user story* (subbab 6.1 di halaman 65), jenis informasi *requirement* yang dikelola adalah *proyek*, *stakeholder*, *requirement*, *istilah*, *masalah*, dan *batasan*. Sedangkan Rambutan mendukung elemen-elemen RQML **Taxonomy**, **Requirement**, **Stakeholder**, **Usecase**, **Project**, **Assumption**, **Context**, **Issue**, dan **Lexicon**. Berhubung *constraint* adalah salah satu jenis *requirement* [Grad92], maka jenis data yang

didukung oleh Rambutan adalah *superset* dari yang ada pada *user story* sehingga kemampuannya melebihi dari kebutuhan awal. Sayangnya *traceability* belum didukung oleh Rambutan sehingga *requirement* ini tidak seluruhnya terpenuhi.

Pemenuhan: Sebagian.

8.1.3 Sistem mempunyai sebuah modul untuk digunakan di komputer genggam

Adanya aplikasi *handheld* memenuhi *requirement* ini.

Pemenuhan: Penuh.

8.1.4 Sistem mempunyai sebuah modul untuk digunakan di komputer desktop

Adanya aplikasi *desktop* memenuhi *requirement* ini.

Pemenuhan: Penuh.

8.1.5 Informasi di aplikasi *desktop* dan *handheld* dapat dipertukarkan

Aplikasi *desktop* dapat membuka serta menyimpan data yang ditulis dalam *format* untuk aplikasi *handheld*. Tetapi operasi ini harus dilakukan secara manual. Bila seorang pengguna ingin meng-*edit* dokumen handheld di aplikasi desktop, ia harus mem-*backup* data di aplikasi *handheld*, membuka dokumen *handheld* tersebut dengan aplikasi *desktop*, melakukan modifikasi, menyimpannya, dan terakhir meng-*install* kembali dokumen itu ke dalam komputer genggam. Idealnya mayoritas operasi ini dilakukan secara transparan oleh sistem pada saat HotSync/ActiveSync[21].

Pemenuhan: Sebagian.

8.2 Terhadap Batasan Rancangan

Bagian ini membahas pemenuhan Rambutan terhadap berbagai *constraint* yang dituliskan di subbab 6.3 (halaman 73). Seluruh batasan yang ada dituliskan kembali diikuti dengan hasil pengujiannya.

[21] Sinkronisasi data antara komputer genggam dengan *desktop*-nya.

8.2.1 Aplikasi harus dirancang untuk mempermudah pengembangan selanjut nya (*extensible*)

Berbagai usaha telah dilakukan untuk mendukung ekstensibilitas Rambutan. Kedua aplikasi telah dirancang secara *modular* dan mempergunakan *design patterns*. Implementasi *data model* pada aplikasi *desktop* tidak tergantung pada Swing – yang menjadi penerapan antarmukanya – sehingga memungkinkan penerapan jenis-jenis antarmuka yang lain. Selain itu *data model* aplikasi *desktop* juga menggunakan API DOM sehingga penyimpanan data berbasis *file* yang digunakan sekarang berpotensi untuk ditingkatkan lewat integrasi dengan *software* DBMS berbasis XML[22]. Antarmuka-antarmuka RQML mengabstraksikan kode *user interface* kedua aplikasi agar konsisten serta untuk memisahkan komponen-komponen *user interface* dari implementasi *data model* kedua aplikasi tersebut. Kesemua itu ditambah dengan rasio JavaDoc per fungsi[23] yang mencapai nilai 0.74 – berarti untuk setiap empat *method*, maka tiga di antaranya didokumentasikan.

Walaupun demikian, hasil nyata dari usaha-usaha ini belumlah nampak sampai ada perkembangan lebih lanjut dari Rambutan dan digunakannya *software* ini di dalam industri.

Pemenuhan: Belum Terbukti.

8.2.2 Menggunakan [Gamm95] *design patterns*

Sebagaimana telah dibahas mengenai penerapan *design patterns* pada kedua aplikasi di subbab 7.5 (halaman 100) maka *requirement* ini terpenuhi.

Pemenuhan: Penuh.

8.2.3 Aplikasi *handheld* harus *multi-platform*

Aplikasi untuk komputer genggam diprogram untuk SuperWaba. Sedangkan SuperWaba menyediakan berbagai *runtime* masing-masing untuk sistem-sistem operasi PalmOS, Windows CE, dan PocketPC untuk berbagai jenis prosesor. Sebuah program

---

[22] Seperti misalnya Apache Xindice.

[23] Hitungan *source code metrics* ada pada subbab 7.6 di halaman 121.

SuperWaba setelah di-*compile* dapat dijalankan di semua sistem operasi yang didukung oleh SuperWaba tanpa perlu kompilasi ulang.

Pemenuhan: Penuh.

8.2.4 Aplikasi desktop harus *multi-platform*.

Aplikasi untuk komputer desktop diprogram untuk *platform* J2SE, versi 1.3. Sedangkan *platform* ini mendukung secara langsung setidak-tidaknya sistem-sistem operasi Windows, Solaris, dan Linux. Selain itu ada pihak ketiga yang menyediakan platform J2SE untuk MacOS X dan Novell Netware.

Pemenuhan: Penuh.

8.2.5 Terdapat kode program pada *handheld* dan *desktop* yang digunakan bersama

Package-package `com.arcle.rmt.rqml` dan `com.arcle.rmt.xplat` tidak bergantung pada API tertentu sehingga dapat digunakan untuk SuperWaba maupun J2SE. Package `com.arcle.rmt.rqml` memuat berbagai `interface` yang mengabstraksikan sekaligus menstandarisasi *data model* RQML bagi kedua aplikasi.

Pemenuhan: Penuh.

8.2.6 Aplikasi yang dibuat tidak tergantung pada metodologi RE tertentu

Karena Rambutan didasari pada RQML, yang pada gilirannya dibuat untuk mendukung sekumpulan *best practices* pada RE, maka secara implisit Rambutan dapat digunakan untuk beberapa metodologi RE.

Pemenuhan: Penuh.

8.2.7 Tidak ada pembayaran kontinu kepada pihak ketiga

Berbagai perangkat lunak yang menunjang pembuatan Rambutan, yaitu SuperWaba, Java Development Kit (JDK) 1.3, Doxygen, Apache Ant, dan CVS[24] dapat di-*download*

[24] CVS: Concurrent Versions System – http://www.cvshome.org

secara gratis. Semua *software* ini, kecuali JDK, adalah perangkat lunak *open-source* dengan lisensi yang telah disetujui oleh OSI[25].

Pemenuhan: Penuh.

8.3 Terhadap Standar RQML

Dari sebelas elemen *first-class* yang didefinisikan oleh RQML, dua diantaranya yaitu `<group>` dan `<trace>` tidak didukung oleh Rambutan. Pengabaian ini dilakukan karena sulit untuk membuat *user interface* yang efisien di aplikasi *handheld* yang mendukung kedua elemen ini. Faktor lain yang menjadi alasan adalah waktu yang tersedia untuk proyek ini sangatlah singkat.

Elemen lain yang tidak didukung adalah elemen `<term>` yang berfungsi untuk menandai sekelompok teks dengan attribut-attribut tertentu. Kesulitan yang ditemui dalam pengimplementasian elemen ini adalah karena SuperWaba tidak menyediakan *user interface control* untuk hal ini – sebuah *control* yang memiliki kemampuan menandai bagian-bagian teks dengan berbagai attribut dapat dikatakan memiliki fungsionalitas sebuah *word processor* mini. Pembuatan sebuah *custom control* untuk melakukan hal ini juga tidak dapat dilakukan mengingat waktu yang sangat terbatas.

Penyederhanaan lain adalah untuk setiap elemen hanya dapat memiliki satu *instance* dari setiap jenis *child element* yang dapat dimilikinya. Hal ini telah dibahas di subbab 7.3 (halaman 92).

Berikut ini dijabarkan pemenuhan Rambutan terhadap berbagai *requirement* yang aslinya dibebankan terhadap RQML. Untuk setiap *requirement* juga diberikan pemenuhan RQML terhadap *requirement* tersebut sebagaimana evaluasi terhadap hal ini dilakukan di [Gudg00]. Berbagai *requirement* ini telah dibahas di subbab 4.4 (halaman 37).

[25] OSI: Open Source Initiative – http://www.opensource.org

Tabel 8.1 Pemenuhan sistem terhadap *requirement* RQML

Requirement	**Pemenuhan**		**Keterangan**
	RQML	**Rambutan**	
REQ1	Penuh	Penuh	Aplikasi *desktop* menyimpan data SRS dalam bentuk file RQML.
REQ2	Sebagian	Sebagian	Kedayagunaan Rambutan perlu dievaluasi kembali di dalam penggunaannya oleh industri.
REQ3	Sebagian	Sebagian	Walau kemampuan Rambutan saat ini sangat terbatas, arsitektur serta *platform* yang digunakan sangat memungkinkan pengembangan lebih lanjut.
REQ4	Sebagian	Penuh	Rambutan merupakan salah satu perangkat lunak yang menggunakan RQML baik secara konseptual maupun sebagai *data format*.
REQ5	Sebagian	Sebagian	Rambutan tidak memaksakan metodologi RE tertentu.
REQ5-1	Penuh	Penuh	Dukungan atas elemen `<lexicon>` yang digunakan untuk membuat daftar istilah.
REQ5-2	Penuh	Tidak	Elemen `<trace>` belum didukung.
REQ5-3	Penuh	Tidak	Karena elemen `<trace>` digunakan untuk *baselining*; lihat REQ5-2.
REQ5-4	Penuh	Penuh	Semua obyek *first-class* mempunyai attribut **status**.
REQ5-5	Penuh	Penuh	Semua obyek *first-class* mempunyai attribut **stability**.
REQ5-6	T/T[26]	Sebagian	Semua obyek *first-class* mempunyai attribut **difficulty** yang dapat digunakan untuk memperkirakan kesulitan yang ditemui – atau antisipasi akan *effort* yang diperlukan.
REQ5-7	Penuh	Penuh	Dukungan atas obyek *first-class* **project** yang memiliki attribut-attribut **scope** dan **vision**.

[26] [Gudg00] tidak memberikan evaluasi untuk REQ5-6.

Tabel 8.1 Pemenuhan sistem terhadap *requirement* RQML

Requirement	**Pemenuhan**		**Keterangan**
	RQML	**Rambutan**	
REQ5-8	Penuh	Sebagian	Walaupun obyek-obyek **taxonomy** dan **stakeholder** didukung, obyek **trace** yang seharusnya menghubungkan dua obyek tersebut belum didukung.
REQ5-9	Penuh	Sebagian	Sama seperti REQ5-8.
REQ5-10	Penuh	Tidak	Elemen `<group>` belum didukung.
REQ5-11	Penuh	Sebagian	Walaupun obyek-obyek **taxonomy** dan **requirement** didukung, obyek **trace** yang seharusnya menghubungkan dua obyek tersebut belum didukung.
REQ5-12	Sebagian	Sebagian	Operasi *copy-paste* dapat dilakukan terhadap teks di dalam *requirement*, walaupun belum dapat dilakukan terhadap obyek *first-class* secara keseluruhan. *Reuse* juga terbatas karena *traceability* belum didukung.
REQ5-13	Penuh	Penuh	Semua obyek *first-class* memiliki attribut **priority**.
REQ5-14	Penuh	Penuh	Memang hanya *requirement* dalam bahasa alami yang didukung.
REQ5-15	Penuh	Tidak	Sama seperti REQ5-2.
REQ5-16	Penuh	Penuh	Setiap obyek **first-class** memiliki attribut ID yang di-*generate* secara *default* dan dapat diubah kemudian oleh pengguna.
REQ5-17	Penuh	Sebagian	Sama seperti REQ5-11.
REQ5-18	T/B	Tidak	Sama seperti REQ5-2.
REQ6	Penuh	Penuh	Obyek **taxonomy** didukung.
REQ7	Penuh	Sebagian	Sama seperti REQ5-2.
REQ8	Penuh	Tidak	Elemen `<term>` belum didukung.
REQ9	Penuh	Penuh	Semua obyek *first-class* memiliki attribut-attribut **name**, **description**, dan **rationale**.

Tabel 8.1 Pemenuhan sistem terhadap *requirement* RQML

Requirement	**Pemenuhan**		**Keterangan**
	RQML	**Rambutan**	
REQ10	Penuh	Penuh	Obyek **stakeholder** didukung.
REQ10-1	Penuh	Penuh	Obyek **assumption** didukung.
REQ10-2	Penuh	Penuh	Obyek **issue** didukung.
T/B: tidak terbukti. T/T: tidak diketahui.			

8.4 Terhadap Fitur dari Alat Bantu Sejenis

Bagian ini membandingkan fitur-fitur yang umum terdapat pada alat-alat bantu manajemen *requirement* dengan kemampuan yang dimiliki oleh Rambutan. Berbagai fitur yang digunakan sebagai perbandingan yaitu sebagaimana ditulis di [Koto98] dan telah dibahas di subbab 2.5 (halaman 17). Tujuan perbandingan ini adalah untuk mendapatkan gambaran posisi Rambutan di antara berbagai alat bantu sejenis yang telah ada di pasaran.

8.4.1 *Requirements Browser*

Pada dasarnya tampilan *Document Tree* (subbab 7.4.2, halaman 98) adalah sebuah *requirements browser* yang menampilkan obyek-obyek *first-class* yang ada di dalam suatu dokumen RQML. Dengan menggunakan tampilan *tree*, obyek-obyek *first-class* ini dikelompokkan sesuai dengan jenisnya masing-masing. Pengguna dapat memakai tampilan ini untuk mencari serta menguibah berbagai *requirement* yang ada di dalam suatu dokumen.

Pemenuhan: Penuh.

8.4.2 *Query System*

Pada tahap ini belum ada fasilitas *search* untuk mencari suatu *requirement* tertentu di dalam dokumen; walaupun dengan arsitektur yang ada, fungsi ini bisa ditambah-kan kemudian.

Pemenuhan: Tidak.

8.4.3 *Requirement Converter*

Karena aplikasi desktop menggunakan format file XML sebagai penyimpan data, sebuah *XML stylesheet* dapat dibuat untuk mengubah *file* XML yang dihasilkan oleh suatu program pengolah kata[27] menjadi *file* RQML. Dengan adanya transformasi antar file XML ini, maka *stylesheet* tersebut dapat disebut sebagai sebuah *requirement converter*. Walaupun demikian, cara-cara untuk memelihara hubungan antara dokumen XML sumber dengan dokumen RQML hasil transformasi masih menjadi pertanyaan.

Pemenuhan: Sebagian.

8.4.4 *Change Control System*

Rambutan tidak memiliki sistem kendali versi di dalamnya. Tetapi karena format *file* yang digunakan adalah *file* XML – yang notabene adalah *file* teks – maka dapat digunakan sistem *version control* yang umum digunakan untuk *source code*, seperti misalnya CVS. Alternatif lain daripada membuat dukungan sistem kendali versi sendiri adalah dengan menggunakan sistem *version control* yang ditujukan untuk *file-file* XML[28].

Pemenuhan: Sebagian.

---

[27] Contoh program-program pengolah kata yang mendukung XML adalah OpenOffice dan Corel WordPerfect™.

[28] Salah satu perangkat lunak *version control* untuk XML adalah DeltaXML – http://www.deltaxml.com/

Bab 9

PENUTUP

9.1 Rangkuman

Laporan proyek tugas akhir ini terdiri dari dua bagian besar yaitu landasan teori dan pembahasan. Bab-bab yang tergabung dalam landasan teori memperkenalkan berbagai teori yang digunakan dalam pembuatan program, yaitu *Requirements Management*, *XML*, *Design Patterns*, dan *Requirements Markup Language*. Sedangkan pembahasan terdiri dari laporan-laporan yang berkaitan dengan program yang dibuat yaitu *Analisis dan Peracangan*, *Implementasi*, dan *Pengujian*.

Bab *Requirements Engineering* (RE) membahas definisi *requirements*, proses RE, *traceability*, *requirements patterns*, serta alat bantu RE. Untuk memotivasi pembaca, pembahasan diawali dengan suatu dongeng yang mengilustrasikan perlunya suatu disiplin ilmu yang menangani *requirements*.

Bab *XML* (Extensible Markup Language) membahas sejarah singkat XML, berbagai terminologi seputar XML, XML stylesheet (XSL), Document Type Definition (DTD), pengurai XML, serta *namespaces* dan Xlink.

Bab *Requirements Markup Language* (RQML) memberikan langkah-langkah yang diambil dalam perancangan RQML dan diakhiri dengan pembahasan RQML DTD. Hal-hal yang mempengaruhi rancangan RQML adalah fakta-fakta dari studi literatur RE, *best practices* yang digariskan oleh Wiegers [Wieg99] serta suatu himpunan *requirement* yang dibuat atas kedua dasar ini. Selain itu juga ada tiga *quality attributes* yang diinginkan ada di dalam RQML yaitu *integratibility*, *extensibility*, dan *portability*.

Bab *Design Patterns* membahas sebuah *subset* dari berbagai *pattern* yang dimuat di [Gamm95] dan diikuti dengan pembahasan dua arsitektur perangkat lunak. Pembahasan *pattern* dilakukan dengan cara mendaftarkan nama dan niatan (*intent*) dari masing-masing *pattern* di dalam bentuk tabel. Sedangkan dua arsitektur yang dibahas adalah *Model-View-Controller* yang berasal dari SmallTalk dan *Separable Model Architecture* yang diterapkan di Swing.

Bab *Analisis dan Perancangan* memberikan uraian analisa yang dilakukan serta rancangan awal dari Rambutan, nama alat bantu yang dibuat. Analisa dilakukan terhadap *user story* yang menghasilkan sekelompok *requirement* untuk pedoman pembuatan perangkat lunak. Berbagai pertimbangan lain menghasilkan beberapa *constraints* yang dibebankan pada Rambutan. Bab ini diakhiri dengan rancangan umum program dan rancangan antarmuka pemakai.

Bab *Implementasi* membahas teknologi yang digunakan, arsitektur, implementasi RQML, *design patterns* yang diterapkan, serta memberikan beberapa data statistik. Berbagai teknologi yang dipakai dalam implementasi Rambutan adalah Java 2 Standard Edition, SuperWaba, Apache Ant, dan Doxygen. Pembahasan arsitektur meliputi tinjauan-tinjauan secara garis besar dari aplikasi *desktop*, aplikasi *handheld*, dan subsistem yang menghubungkan keduanya. Implementasi RQML membahas cara yang diambil untuk memetakan RQML DTD ke dalam hirarki kelas. Pembahasan *design patterns* mengambil beberapa komponen program yang cukup menarik serta membahas berbagai *pattern* yang diterapkannya. Sedangkan data statistik yang diberikan adalah *source code metrics* yang meliputi hitungan jumlah *statement*, jumlah percabangan, dan jumlah *JavaDoc comments*.

Bab *Pengujian* mengukur Rambutan dari tiga sudut pandang. Sudut pandang pertama adalah himpunan *requirements* dan *constraints* yang sebelumnya dibahas di dalam bab *Analisis dan Perancangan*. Sudut pandang kedua adalah *compliance* Rambutan terhadap standar RQML (dengan mengandaikan RQML sebagai sebuah standar). Sudut pandang ketiga adalah perbandingan fitur-fitur Rambutan dengan berbagai fitur yang umum dimiliki oleh alat bantu sejenis, sebagaimana digariskan di dalam [Koto98].

9.2 Kesimpulan

Selama pembuatan Rambutan dan penulisan laporan tugas akhir ini dicapai beberapa kesimpulan, yaitu:

- Penerapan arsitektur MVC di aplikasi *handheld* bisa dikatakan kurang memuaskan. Hal ini disebabkan terutama karena, tidak seperti SmallTalk yang merupakan cikal-bakal MVC [Burb92], pustaka antarmuka pemakai SuperWaba tidak secara spesifik dirancang untuk mengakomodasi arsitektur MVC. Harga

yang dibayar oleh pemisahan kelas *view* dengan *controller* tidak terimpaskan dengan *layering* serta enkapsulasi yang lebih baik. Langkah-langkah yang diambil untuk penerapan MVC pada aplikasi handheld dapat dilihat di subbab 7.5.5 (halaman 108).

- RQML mendefinisikan banyak tipe (jenis elemen) namun hanya sedikit dari elemen-elemen ini yang mendefinisikan informasi yang unik. Setelah dipetakan menjadi hirarki *interface* hal ini menghasilkan berbagai `interface` kosong (*interface* yang tidak mendeklarasikan *method* baru) yang tidak sedikit. Akibat lebih lanjut yaitu kelas-kelas penerapan *interface* ini menjadi kosong juga. Ledakan jumlah kelas ini menjadi masalah cukup serius di aplikasi *handheld* di mana ruang penyimpanan data lebih terbatas. Contoh `interface` kosong yang terdapat dalam *data model* RQML ada di subbab 7.3 (halaman 95).
- Banyaknya kelas yang diakibatkan oleh penerapan MVC, penerapan *data model* RQML (lengkap dengan berbagai `interface` kosongnya), serta kemampuan *multi-platform* harus dibayar dengan ukuran pada aplikasi *handheld* yang mencapai 192KB. Ukuran yang belum termasuk SuperWaba VM ini dirasa cukup besar bagi bagi sebuah program PalmOS – program-program *native* PalmOS umumnya berukuran di bawah seratus *kilobyte*.
- Aplikasi *desktop* yang lebih canggih serta adanya subsistem penghubung ke aplikasi *handheld* (subsistem *Bridge Facade* dengan berbagai pendukungnya) memungkinkan lebih banyak *design patterns* yang dapat diterapkan di aplikasi *desktop* ketimbang rekannya di *handheld*. Pembahasan arsitektur aplikasi desktop ada pada sub-sub bab 7.2.2, 7.2.3, 7.5.2, 7.5.4, 7.5.6, 7.5.7, 7.5.8, 7.5.9, dan 7.5.10 (halaman 89 – 118).
- Semua orang dapat membuat kesalahan. Penyusunan [Gudg00] yang sangat rapi dan lengkap dengan informasi *traceability* dari landasan teori ke perancangan dan terus ke evaluasi tetap tidak lolos dari beberapa ketidakkonsistenan. Contoh salah satu ketidakkonsistenan ini diberikan di subbab 4.6.4.6 (halaman 47).
- Usulan standar RQML yang dimuat di dalam [Gudg00] terbukti cukup *feasible* sebagai salah satu dasar pembuatan alat bantu *requirements management*. Namun masih ada beberapa hal yang perlu diselesaikan sebelum [Gudg00] dijadikan bakuan, terutama masalah konsistensi.

- Untuk beberapa fitur seperti *traceability* dan *grouping*, perumusan sebuah antarmuka pemakai yang praktis pada aplikasi *handheld* ternyata cukup sulit karena ukuran layar komputer genggam yang relatif kecil. Belum lagi pustaka *user interface controls* yang lebih terbatas akan memberikan tantangan yang cukup besar dalam implementasi fitur-fitur ini.
- Rambutan telah menjawab tantangan *market opportunity* adanya sebuah alat bantu *requirements management* berbasis komputer genggam. Namun, seperti telah dirinci pada subbab 8.4 (halaman 131), fitur-fitur yang dimiliki perangkat lunak ini masih jauh ketimbang pesaing-pesaing lainnya yang telah lebih dulu matang. Masih jauh perjalanan Rambutan sebelum layak dipasarkan sebagai perangkat lunak siap pakai.

9.3 Saran

Berikut ini adalah beberapa saran yang dapat dilakukan untuk penelitian lebih lanjut yang bertolak dari tugas akhir ini:

- Mengimplementasikan dukungan terhadap elemen-elemen `<group>` dan `<trace>` sehingga Rambutan dapat mengakomodasikan *traceability*, *focus group*, serta pengkategorisasian *requirements* yang lebih ekstensif.
- Mengimplementasikan fungsi *reporting* agar dokumen SRS yang dikelola oleh Rambutan dapat dicetak *hard copy*. Fungsi ini dapat dilakukan dengan cara menggunakan *XML stylesheet* untuk mengubah dokumen RQML menjadi HTML ataupun PDF untuk kemudian di-*print*. Lebih jauh lagi, fungsi *reporting* dapat dirancang untuk hanya mencetak sebuah *subset* dari keseluruhan *requirement* yang ada sesuai dengan pilihan pemakai.
- Melakukan *refactoring* untuk menurunkan jumlah baris kode program serta meningkatkan modularitas dan *maintainability*.
- Melakukan analisis *cross-cutting concerns* [Kicz97] terhadap kode program untuk penerapan *Aspect-Oriented Programming* (AOP). Berhubung gabungan kedua aplikasi telah melampaui batas lima ribu baris yang menjadi tolok ukur program besar, maka AOP diharapkan dapat membawa pengaruh positif terhadap modularitas Rambutan [Shah02]. Adalah suatu hal yang menarik untuk melihat apakah AspectJ – sebuah *compiler* AOP untuk Java – dapat diterapkan di aplikasi SuperWaba, yang notabene tidak menggunakan API standar Java.

- Diadakannya *usability research* untuk menguji Rambutan dan RQML di dalam situasi nyata. Hal ini perlu untuk mematangkan Rambutan agar dapat menjadi *software* siap pakai.
- Ekspansi cakupan Rambutan ke sistem informasi *requirements management* yang *multi-user*, mungkin dengan dukungan basis data *hybrid* yang berbasis XML dan relasional. Pada tahap ini aplikasi *handheld* dapat diberikan kemampuan untuk mengirim dan menerima dokumen SRS lewat jaringan nirkabel.
- Ekspansi Rambutan untuk mendukung *mobile devices* yang lain; mungkin dengan bantuan media *web*, WAP, mungkin bahkan di-*port* ke J2ME (bila *platform* ini ternyata *feasible* – mengingat fitur dan performa J2ME lebih terbatas ketimbang SuperWaba).
- Pengevolusian Rambutan berbarengan dengan RQML – alangkah baiknya jika RQML dapat dijadikan suatu bakuan yang diterima oleh berbagai pihak dengan Rambutan sebagai *reference implementation* dari standar tersebut.

A. DAFTAR PUSTAKA

[Achr03] Achrafi, Rabi. *Requirements Tools*. Volere Requirements Resources; Atlantic Systems Guild, Inc. http://www.volere.co.uk/tools.htm ; http://www.systemsguild.com/GuildSite/Robs/retools.html

[Adib02] Adibowo, Sasmito. *Developing Handhelds: Targeting Personal Digital Assistant Platforms*. Arcle Technologies Publications. 6 June 2002.

[Alex77] Alexander, Christopher, Ishikawa, Sara, et. al., *A Pattern Language*. Oxford University Press, New York. 1977.

[Ambl00] Ambler, Scott. *Requirements Engineering Patterns*. SD Magazine, CMP Media, LLC. 2000. http://www.sdmagazine.com/documents/s=744/sdm0005k/0005k.htm?temp=aK8YFuf9on

[Apac00a] *Ant Specification: Version 0.5*. The Apache Software Foundation. 20 April 2000

[Apac02a] *Apache Ant – Frequently Asked Questions*. The Apache Software Foundation. 2002. http://ant.apache.org/faq.html

[Burb92] Burbeck, Steve. *Applications Programming in Smalltalk-80™: How to use Model-View-Controller (MVC)*. ParcPlace Systems, Inc. 1987, 1992.

[Booc99] Booch, Grady, James Rumbaugh, Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[Benn00] Bennett, Simon, Steve McRobb, Ray Farmer. *Object-Oriented Systems Analysis and Design using UML*. McGraw-Hill International Editions. 2000.

[Borl93] et. al. Borland. *Borland ObjectWindows Programmer's Guide*. Borland International. 1993.

[Chun00] Chung, L, Nixon, B, Yu, E. & Mylopoulos, J. *Non-Functional Requirements in Software Engineering*. Boston: Kluwer Academic Publishers. 2000.

[Coop98] Cooper, James W. *The Design Patterns Java Companion*. Addison-Wesley Design Pattern Series. 1998.

[Cons] World-Wide Web Consortium. *World Wide Web Consortium (W3C)*.

[Cons00a] World-Wide Web Consortium. *Simple Object Access Protocol (SOAP) 1.1*. W3C Website. May 2000.

[Cree99] Creel, Christopher. *Requirements by Pattern*. SD Magazine, CMP Media, LLC. 1999. http://www.sdmagazine.com/documents/s=751/sdm9912d/9912d.htm

[Davi92] Davis, A. *Operational Prototyping: A New Development Approach*. Software, 9(5): 70–78. 1992

[Dick02] Dickerson, Peter. *Jump2 User's Manual*. 5 December 2002. http://sourceforge.net/projects/jump

[Dhar02] Dharmawan, Zulfikar S. *Alat Bantu Requirements Management Berbasis Web Dengan Memanfaatkan Dokumen Extensible Markup Language*. University of Indonesia Research Paper. 2002.

[Dary00] Daryl, Eamon Guiney; Kulak and Eric Lavkulich. *Use case: Requirements in Practice*. Addison-Wesley. 2000.

[Dorf90] Dorfman, Merlin, and Richard H Thayer. *Standards, Guidelines, and Examples of System and Software Requirements Engineering*. Los Alamitos, CA: IEEE Computer Society Press. 1990.

[Duve02] Duveau, Laurent. *Bench2*. aldweb Site. 12 December 2002. http://www.aldweb.com/articles.php?lng=en&pg=24&prt=1

[Ecks99] Eckstein, Robert. *XML Pocket Reference*. O'Reilly. 1999.

[Fari03] Farine, Arnaud. *SuperWaba: Java Language for PDA*. SuperWaba France. 2003.

[Fede] Federal Information Processing Standards Publications. *Integration Definition for Function Modeling (IDEF0)*.

[Fowl02] Fowler, Amy. *A Swing Architecture Overview: The Inside Story on JFC Component Design*. Sun Microsystems Technical Document. 2002. http://java.sun.com/products/jfc/tsc/articles/architecture/

[Gamm95] Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Publishing Company. 1995.

[Gogu94] Goguen, J. & Jirotka, M. *Requirements Engineering: Social and Technical Issues*. London: Academic Press. 1994.

[Grad92] Grady, R. *Practical Software Metrics for Project Management and Process Improvement*. Englewood Cliffs, NJ: Prentice-Hall. 1992.

[Gudg00] Gudgeirsson, Gardar. *Requirements Engineering and XML*. University of York Research Paper. 2000.

[Grah99] Graham, Ian S, Liam Quinn. *XML Specification Guide*. Wiley. 1999.

[Gust99] Gustavsson, Andreas, & Mattias Ersson. *Formalizing the Intent of Design Patterns: An Approach Towards a Solution to The Indexing Problem*. Uppsala Universitet, Computing Science Department, Institute of Technology. 1999.

[Haza03] Hazan, Guilherme C, Ed Crandell. *SuperWaba FAQ*. February 14, 2003. http://www.superwaba.com.br/faq.asp

[IBM] et. al., IBM. *OpenDoc*.

[Inco02] INCOSE RM Tools Working Group. *Tools Survey: Requirements Management (RM) Tools*. International Council on Systems Engineering. December 29, 2002. http://www.incose.org/tools/tooltax.html

[Jark98] Jarke, Matthias. *Requirements Tracing: Communications of the ACM*. 1998. 41(12):32-36.

[Jone95] Jones, David A., Donald M. York, John F. Nallon, Joseph Simpson. *Factors Influencing Requirements Management Toolset Selection*. INCOSE Requirements Working Group. 1995. http://www.incose.org/lib/rmtools.html

[Kicz97] Kiczales, Gregor, et. al. *Aspect-Oriented Programming*. ECOOP 1997. http://aspectj.org/documentation/papers/AndSlides/ECOOP1997-AOP.pdf

[Koto98] Kotonya, Gerald, Ian Sommerville. *Requirements Engineering - Processes and Techniques*. Wiley. 1998.

[Lee02] Lee, Chr Clemens. *JavaNCSS - A Source Measurement Suite for Java*. 2002. http://www.kclee.com/clemens/java/javancss/

[Lams98] van Lamsweerde, A. Darimont, R. & Letier, E. *Managing Conflicts in Goal-Driven Requirements Engineering*. IEEE Transactions on Software Engineering, 24(11): 908-926. 1998

[Leff00] Leffingwell, Dean & Don Widrig. *Managing Software Requirements: A Unified Approach*. Addison-Wesley. 2000.

[Maid96] Maiden, N. *CREWS-SAVRE: Scenarios for Acquiring and Validating Requirements*. Automated Software Engineering. 5(4): 419-446. 1996

[Marc99] Marchal, Benoît. *XML By Example*. Que. 1999.

[McDe94] McDermid, John. *Software Engineer's Reference Book*. Butterworth Heinmann. 1994.

[Micr] et. al. Microsoft. *BizTalk*.

[Nuse00] Nuseibeh, Bashar, & Steve Easterbrook. *Requirements Engineering: A Roadmap*. Department of Computing, Imperial College, London, UK and Department of Computer Science, University of Toronto, Ontario, Canada (respectively). 2000.

[Oxfo90] *Oxford Dictionary of Computing*. Oxford University Press. 1990

[Palm] et. al. Palm. *Zen of Palm*. Palm, Inc.

[Palm02] *Palm OS® 5 Development Overview*. PalmSource, Inc. 2002. http://www.palmos.com/dev/support/docs/palmos50/

[Pres01] Pressman, Roger S. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Higher Education. 2001.

[Robb01] Robbins, Stephen P. *Organizational Behavior 9th ed*. Prentice-Hall International, Inc. 2001.

[Saut03] Sauter, Vicky. *The King's Companion*. 2003.

[Shaw96] Shaw, M & Gaines B. *Requirements Acquisition*. Software Engineering Journal, 11(3): 149-165. 1996

[Shah02] Shahab, Quonita M. *Aspect-Oriented Programming (AOP) untuk Menangani Masalah pada Software Modularity*. University of Indonesia Research Paper. 2002.

[Stan97] *The Java Dependency Mechanism and the MVC Approach*. Stanford Genome. 1997. http://java.cnam.fr/public_html/Iagl99/dazy/UML_MVC/MVC/Steve/intro.html

[Sun00a] *Java™ 2 SDK, Standard Edition Documentation, version 1.3*. Sun Microsystems. 2000.

[Sun00b] *Java™ 2 Micro Edition (J2ME) Technology for Creating Mobile Devices*. Sun Microsystems. May 19, 2000.

[Sun01a] *J2EE Design Patterns*. Sun Microsystems. 2001.

[Sun02a] *What is Java™ Technology?* Sun Microsystems. 24 June 2002. http://java.sun.com/java2/whatis/

[Sun03a] *The Java Tutorial: Creating a GUI with JFC/Swing*. Sun Microsystems, Inc. 1995-2003. http://java.sun.com/docs/books/tutorial

[Sun03b] *The Java Tutorial: Learning the Java Language*. Sun Microsystems, Inc. 1995-2003. http://java.sun.com/docs/books/tutorial

[Syba02] *PowerDesigner OOM User's Guide*. Sybase PowerDesigner Help Files, version 9.5.1. December 2002.

[Wall00] Wallace, Nathan. *Design Patterns in Web Programming*. E-gineer article. March 8th, 2000. http://www.e-gineer.com/articles/

[Whit99] Whitten, Bentley, et. Al. *System Analysis and Design Methods*. McGraw-Hill. 1999.

[Wieg99] Wiegers, Karl E. *Software Requirements*. Microsoft Press. 1999.

[Wieg00] Wiegers, Karl E. *When Telepathy Won't Do: Requirements Engineering Key Practices*. Cutter IT Journal. May 2000.

[Wild99] Wild, Rick. *Happy Fingers*. WabaSoft. 1999. http://www.wabasoft.com/happyfingers.shtml

[Vill99] Viller, S & Sommerville, I. *Social Analysis in the Requirements Engineering Process: from ethnography to method*. 4th International Symposium on Requirements Engineering (RE '99), Limerick, Ireland, 7-11th June 1999.

[Zave97] Zave, P. *Classification of Research Efforts in Requirements Engineering*. ACM Computing Surveys, 29(4): 315-321. 1997.