Mengenal Enterprise Application Integration (EAI)

Ida Bagus Adi Sudewa
deweu@yahoo.com

Pengantar

Pada artikel kali ini, kita akan membahas topik yang jarang atau bahkan tidak pernah dibahas dalam perkuliahan formal ilmu komputer. Pengajaran ilmu komputer selalu mengandaikan kondisi ideal dan sederhana. Kenyataannya, kondisi sistem informasi di dunia bisnis tidaklah ideal dan biasanya sangat kompleks. Kuliah Rekayasa Perangkat Lunak (*Software Engineering*) biasanya mengasumsikan bahwa mahasiswa membuat software baru dari awal (*building from scratch*). Di dunia bisnis, komponen perangkat lunak yang sudah ada sebisa mungkin terus digunakan untuk memaksimalkan investasi perangkat lunak yang sudah pernah ditanamkan. Di dalam dunia perkuliahan, isi dari komunikasi data antara dua komputer tidak pernah harus dicatat secara mendetil. Di dunia bisnis, setiap transaksi antara dua sistem harus dicatat dalam *audit logging* dan menjadi bagian dari laporan akunting.

Jadi, apa hubungan antara tulisan di atas dengan tema artikel kali ini tentang *Enterprise Application Integration* (EAI)? Pada waktu penulis masih duduk di bangku kuliah, penulis mengikuti kuliah pilihan tentang komunikasi antara sistem terdistribusi. Teknologi untuk membuat dua buah aplikasi dapat "berbicara" satu sama lain adalah RPC (*Remote Procedure Call*), CORBA (*Common Object Request Broker Architecture*) , atau RMI (*Remote Method Invocation*)/IIOP (*Internet Inter-ORB Protocol*). Sekarang setelah penulis berkecimpung di dunia kerja – dengan spesialisasi di bidang EAI –, teknologi yang dipakai adalah … WebSphere MQ, MSMQ, SonicMQ, Tibco, MQ Integrator. Sederetan nama produk yang TIDAK menggunakan teknologi RPC, CORBA, dan RMI/IIOP sama sekali. Sepertinya, dunia bisnis mempunyai teknik tersendiri dalam melakukan integrasi. Mari kita kupas satu per satu[1].

[1] Penulis memiliki pengalaman di beberapa proyek integrasi sistem di industri perbankan. Oleh karena itu, kebanyakan contoh yang digunakan berhubungan dengan aplikasi di industri perbankan. Di industri selain perbankan, kondisinya seringkali serupa dengan –walaupun biasanya tidak sekompleks– yang ada di industri perbankan,

Kebutuhan EAI Pada Perusahaan Berskala Menengah/Besar

Kondisi Saat Ini

Besar kemungkinan, sebuah perusahaan berskala besar akan memiliki lebih dari satu aplikasi perangkat lunak. Sebuah bank berukuran sedang paling tidak akan memiliki satu perangkat lunak untuk:

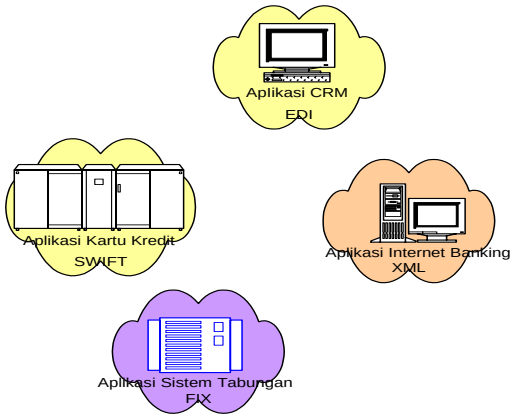
- CIS (Customer Information System)
- Sistem Branch Teller
- Sistem Credit Card
- Sistem Loan
- Sistem Giro (Corporate Banking)
- Sistem Tabungan (Consumer Banking)
- Internet Banking
- Customer Relationship Management (CRM)
- Accounting
- Procurement
- Human Resources

Sistem operasi dan platform yang digunakan pun seringkali tidak familiar untuk telinga kita. Mungkin ada beberapa server Windows, SUN, atau Linux (biasanya untuk DNS atau Mail Server saja). Aplikasi-aplikasi inti (*core application*) sering menggunakan platform seperti: OS/400, z/OS, Compaq True64 Unix, Tandem NonStop Kernel, dan berbagai platform lain.

Mengapa menggunakan platform-platform langka seperti tersebut di atas? Beberapa alasan yang sering dilontarkan antara lain karena faktor sejarah (misalnya: aplikasinya sudah di-*install* sejak tahun 1975) atau karena faktor ekonomi (misalnya: aplikasi termurah dan performanya paling bagus hanya jalan di platform tertentu saja).

Selain masalah platform, isu lain dalam EAI berhubungan dengan *format* komunikasi. Jika satu aplikasi menggunakan EDI (*Electronic Data Interchange*) saja, aplikasi lain hanya mengerti SWIFT, dan aplikasi ketiga –yang baru diluncurkan – sudah menggunakan XML (*eXtensible Markup Language*), maka bagaimana integrasi dapat dilakukan? Dan tentu saja masalah yang paling besar adalah ada beberapa aplikasi yang tidak mampu melakukan komunikasi sama sekali, dengan semua interaksi hanya dapat dilakukan melalui antarmuka pengguna saja[2].

[2] Catatan untuk para programmer aplikasi: dalam membuat aplikasi baru, biarpun sangat sederhana, pisahkan komponen UI anda dengan komponen *business logic* (dan juga dengan komponen *data logic*). Dengan demikian, maka aplikasi anda akan lebih mudah dimodifikasi di kemudian hari, misalnya jika anda ingin melakukan *revamp* terhadap komponen UI-nya saja.



Gambar 1 Pulau-Pulau Aplikasi

Integrasi yang Diinginkan

Dengan kondisi yang telah dijelaskan di bagian sebelumnya, sistem multi-platform dengan format komunikasi yang berbeda-beda, diharapkan integrasi antar aplikasi yang mampu melakukan proses-proses seperti berikut:

Sharing data, misalnya aplikasi Internet Banking ingin mengetahui saldo rekening tabungan dari seorang nasabah. Aplikasi Internet Banking harus melakukan *inquiry* ke Aplikasi Sistem Tabungan.

Update yang konsisten pada beberapa sistem, misalnya seorang nasabah pindah alamat dan melaporkan alamat yang baru ke sebuah cabang. Sistem branch teller harus melakukan propagasi alamat baru ini ke sistem CRM dan CIS. Transaksi ini bersifat *repeatable*, karena jika transaksi ini gagal dilakukan, maka transaksi ini boleh diulangi lagi tanpa mempengaruhi integritas sistem.

Transaksi yang melibatkan lebih dari satu sistem. Jika seorang nasabah telah meminta agar pembayaran kartu kreditnya di-auto debet dari rekening tabungan, maka sistem kartu kredit akan meminta agar sistem tabungan melakukan debet sejumlah yang sama dengan jumlah tagihan kartu kredit dari nasabah. Transaksi ini bersifat *non-repeatable*. Jika transaksi ini gagal dilakukan, transaksi tidak boleh diulang secara gegabah. Bisa jadi, Aplikasi Sistem Tabungan telah melakukan debet akan tetapi Sistem Kartu Kredit belum me-*reset* status kartu kredit nasabah. Kalau transaksi diulang begitu saja, maka rekening tabungan nasabah akan didebet dua kali! Tentu saja nasabah akan marah karena merasa ditipu. Oleh karena itu, transaksi jenis ini merupakan transaksi *atomic*. Artinya, semua proses *update* harus dilakukan, jika satu sistem gagal melakukan update, maka semua sistem lain harus melakukan *rollback* terhadap update yang terlanjur dilakukan.

Sebuah bank dengan aplikasi-aplikasinya yang canggih, tapi tanpa sistem EAI yang canggih pula, akan mengalami kesulitan dalam menambah inovasi layanan baru terhadap para nasabahnya. Di sini akan dijelaskan dua contoh kesulitan yang akan dialami dalam menambahkan layanan baru:

Proses penambahan layanan mobile banking akan menjadi beberapa bulan lebih lambat jika implementor sistem mobile banking harus melakukan pemrograman untuk mengirim transaksi dengan format SWIFT ke sistem kartu kredit dan FIX ke sistem tabungan. Sistem mobile banking ini menggunakan XML (misalnya), dan proses implementasi akan menjadi jauh lebih sederhana, jika semua komunikasi dapat dilakukan dengan XML

Bank ini ingin menambahkan fasilitas untuk memudahkan nasabahnya membeli saham perdana pada IPO melalui ATM. Untuk itu, perlu dilakukan integrasi antara aplikasi sistem tabungan dengan aplikasi dari perusahaan pialang. Aplikasi Sistem Tabungan menggunakan FIX, sedangkan aplikasi dari perusahaan pialang menggunakan EDI. Maka dibutuhkan waktu untuk memodifikasi salah satu sistem agar mengerti format dari sistem lainnya.

Oleh karenanya, dibutuhkan suatu arsitektur EAI yang mampu mengakomodasi semua transaksi yang membutuhkan koordinasi dari lebih dari satu sistem aplikasi. Idealnya, arsitektur EAI akan memiliki karakteristik-karakteristik sebagai berikut:

Memiliki *message format* yang distandarkan. Sistem yang lama harus dimodifikasi untuk mendukung format yang standar, sedangkan sistem yang baru harus diimplementasi dengan format standar ini. Format message standar yang saat ini paling banyak dipakai adalah XML.

Menggunakan *middleware* sebagai kanal komunikasi. Middleware harus mampu menangani aspek-aspek berikut:

Memastikan bahwa message sampai pada tujuannya. Kecuali message yang bersifat *inquiry*, message tidak boleh sampai "hilang" di tengah jalan. Biarpun sistem mengalami *crash* sehingga harus di-*restart*, message harus tetap *survive*. Bahkan ada beberapa bank yang melakukan implementasi *disaster recovery* untuk infrastruktur EAI mereka. Artinya, jika terjadi gempa bumi atau kebakaran pun, message akan dapat dipulihkan di server lain yang ada di lokasi lain.

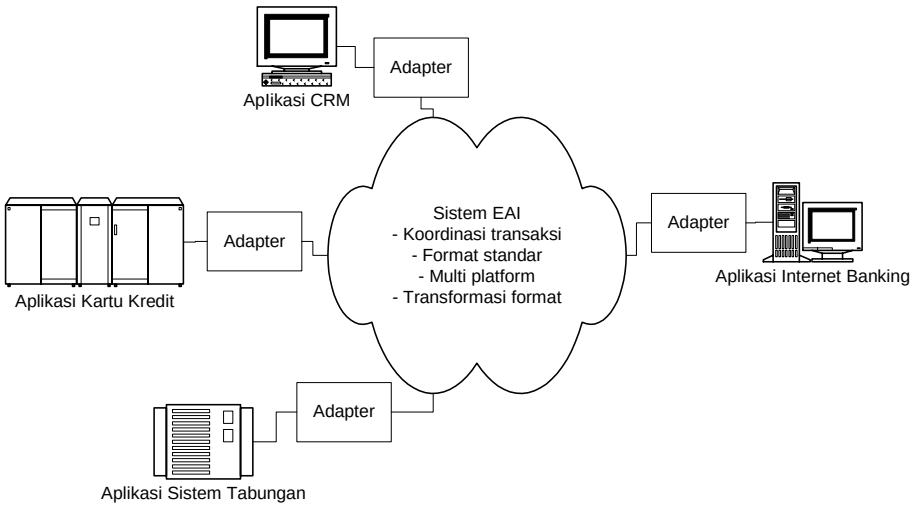
Melakukan transformasi message, misalnya dari XML ke EDI, SWIFT ke FIX, dan lain sebagainya.

Melakukan koordinasi transaksi. Misalnya saja Aplikasi Internet Banking melakukan inisiasi sebuah proses auto debet. Proses ini juga melibatkan Aplikasi Sistem Tabungan dan Aplikasi Kartu Kredit. Aplikasi Internet Banking hanya perlu mengirimkan satu message ke middleware. Middleware akan mengkoordinasikan transaksi yang terjadi pada Aplikasi Sistem Tabungan dan Aplikasi Kartu Kredit, dan melaporkan status akhirnya saja kepada Aplikasi Internet Banking.

Melakukan *audit logging* untuk membuat laporan akunting.

Middleware yang disebutkan pada point 0 harus mendukung semua platform yang ada. Beberapa middleware, seperti misalnya MSMQ dari Microsoft, hanya berjalan di atas sistem operasi Windows.

Aplikasi yang sudah ada hanya boleh dimodifikasi sesedikit mungkin. Implementasi EAI biasanya dilakukan dengan membangun sebuah program *adapter* untuk setiap aplikasi. Adapter inilah yang bertugas "menterjemahkan" antara format standar (misalnya XML) dengan format *proprietary* dari aplikasi.



Gambar 2 Contoh Arsitektur EAI pada Bank

Komponen-Komponen dari EAI

Sebuah arsitektur EAI yang tipikal terdiri dari komponen-komponen berikut:

Adapter. Program ini bertugas untuk mentransformasikan message dari format yang non-standar ke format standar atau sebaliknya.

Message Router and Transformer. Program ini memeriksa validasi message dan "mengarahkan" message ke tujuan yang sesuai. Program ini juga dapat melakukan transformasi message dari format yang non-standar ke format standar atau sebaliknya, sebagai pengganti adapter.

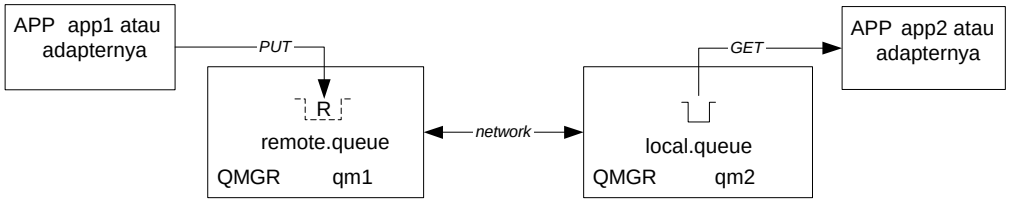
Gatekeeper. Program ini berfungsi untuk memeriksa apakah message yang sama sudah pernah diproses ataukah belum. Jika suatu message sudah pernah diproses, maka message tersebut akan ditolak. Komponen ini biasanya diimplementasikan dalam program yang sama dengan Komponen Adapter.

Compensation Engine. Biar bagaimanapun tahan bantingnya arsitektur EAI kita, tetap saja ada kemungkinan bahwa suatu transaksi berada dalam kondisi "in-doubt". Jika suatu transaksi memiliki status "in-doubt", itu berarti transaksi tersebut berhasil dieksekusi di beberapa aplikasi, akan tetapi gagal dieksekusi di aplikasi-aplikasi lain. Untuk meluruskan status ini, maka aplikasi-aplikasi yang sebelumnya berhasil mengeksekusi transaksi tersebut harus melakukan proses *rollback*. *Compensantion Engine* berfungsi untuk melakukan *rollback* transaksi. Kemungkinan besar, intervensi manual dari system admin juga diperlukan untuk melakukan *rollback*.

Teknologi Middleware

Sayang sekali, beberapa teknologi yang kita kenal baik dalam dunia sistem terdistribusi, seperti RPC, RMI/IIOP, dan CORBA, bukan merupakan teknologi yang populer dalam EAI. Mengapa demikian? Baik RPC, RMI/IIOP, maupun CORBA merupakan teknologi komunikasi data dengan *remote procedure calling* sedangkan teknologi yang digunakan pada perusahaan adalah *message passing*. Berikut ini adalah beberapa teknologi EAI middleware yang populer:

Produk-produk untuk komunikasi *point-to-point*. Beberapa contoh produk dalam kategori ini adalah IBM WebSphere MQ, SonicMQ dari SonicSoftware, Microsoft MSMQ, dan TIBCO Enterprise. Semua produk tersebut menggunakan konsep *queue* (antrian) dalam implementasinya.



Gambar 3 Pengiriman Message dengan Queue

Gambar 3 mengilustrasikan proses pengiriman data dengan menggunakan *queue*. Aplikasi 1 cukup melakukan PUT message ke sebuah queue pada *queue manager* pada mesin lokal. Aplikasi 2 melakukan GET untuk mendapatkan message tersebut. Detail tentang proses pengiriman message melalui jaringan komputer diserahkan kepada middleware.

Produk-produk untuk arsitektur EAI yang bersifat *hub-and-spoke*. Beberapa contoh produk dalam kategori ini adalah IBM WebSphere MQ Integrator, Microsoft Biztalk, dan Crossworlds Interchange Server. Produk-produk ini berfungsi sebagai "hub" dalam arsitektur EAI.

Gambar 4 mengilustrasikan skenario di mana tidak ada *hub node* pada arsitektur EAI. Semua aplikasi terhubung satu sama lain secara *point-to-point*. Bayangkan keruwetan yang terjadi jika misalnya ada 10 aplikasi! Dibutuhkan total 45 koneksi point-to-point dan setiap adapter harus mampu mentranformasikan format message dari 9 adapter lain!

Gambar 5 adalah gambaran arsitektur *hub-and-spoke*. WebSphere MQI/Biztalk/Crossworlds bertindak sebagai hub. Hub berfungsi sebagai *rules engine*, *message validator*, dan *message formatter*. Hub juga bisa berfungsi sebagai pemecah transaksi (*fan-out*) dan penggabung transaksi *(fan-in*). Hub pada dasarnya juga memanfaatkan komunikasi *point-to-point*. Semua adapter mengirimkan message ke hub dengan menggunakan queue sebagai sarana komunikasi.

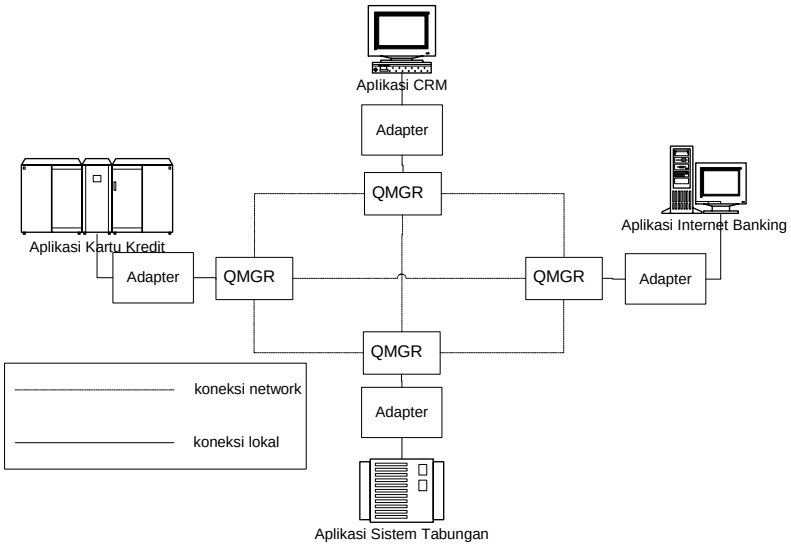
Berikut adalah contoh pengiriman message pada skenario hub-and-spoke:

Aplikasi A mengirimkan message untuk Aplikasi B. Aplikasi A melakukan PUT ke queue untuk dikirim ke hub

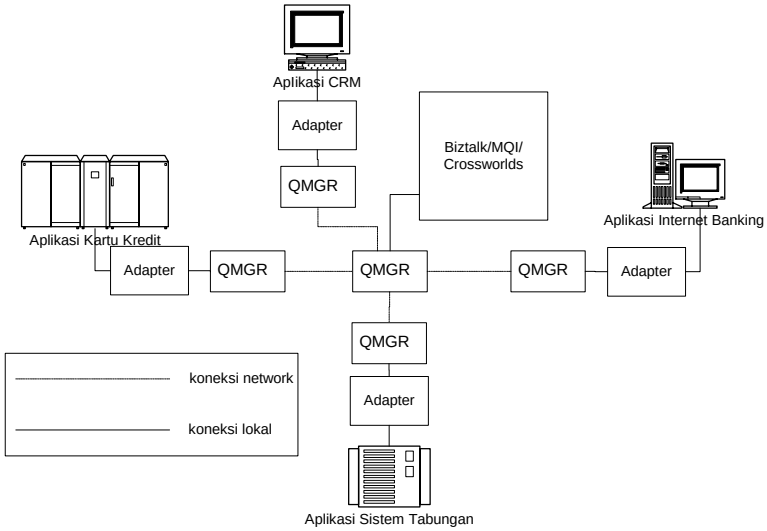
Hub melakukan GET dari queue, dan melakukan pemrosesan terhadap message

Hub melakukan PUT ke queue, untuk dikirim ke Aplikasi B

Aplikasi B melakukan GET untuk mendapatkan message.



Gambar 4 Arsitektur EAI Tanpa "Hub"



Gambar 5 Arsitektur EAI Dengan "Hub"

Jika pemrosesan yang akan dilakukan pada hub cukup sederhana, sebenarnya kita bisa menulis sendiri program hub untuk kebutuhan sendiri. Kemungkinan besar program hub yang komersial tersebut di atas (MQI, Biztalk, dan Crossworlds) memiliki banyak fitur yang akhirnya tidak kita gunakan.

Produk-produk untuk pembuatan adapter.

Adapter bersifat spesifik untuk setiap aplikasi. Karena sifatnya yang sangat spesifik, sebagian besar program adapter ditulis oleh programmer tanpa menggunakan paket aplikasi tertentu.

Akan tetapi, ada beberapa aplikasi yang tidak memiliki antarmuka selain antarmuka pengguna. Sebagian besar aplikasi seperti ini adalah aplikasi *green screen* untuk mainframe. Oleh karena itu, beberapa perusahaan membuat produk *screen scraping* komersial. Produk *screen scraper* bekerja dengan cara mengemulasikan kerja seorang pengguna komputer di depan terminal mainframe. Screen scraper secara otomatis melakukan *login* ke mainframe dan melakukan aktivitas sebagaimana pengguna biasa. Keluaran layar akan di-*capture* dan diproses menjadi message dengan format yang ditentukan.

Penutup

Penulis tergugah untuk menulis artikel ini karena melihat adanya gap antara ilmu yang kita pelajari pada masa perkuliahan komputer dengan apa yang sebenarnya dipraktekkan dalam dunia kerja. Semoga artikel ini dapat memberikan sekelumit gambaran tentang proses integrasi aplikasi (EAI) di dunia kerja.

Referensi

1. Biztalk: http://www.biztalk.org
2. EDI: http://www.wedi.org
3. SWIFT: http://www.swift.com
4. IBM WebSphere MQ Family:
5. http://www-306.ibm.com/software/info1/websphere/index.jsp?tab=products/businessint