BAB 2
Jenis-jenis dan Arsitektur Integrasi

Arsitektur integrasi biasanya dibangun secara sistematis dalam beberapa lapisan. Ide dibalik ini adalah untuk memecah problem ke dalam beberapa problem yang lebih kecil dan memecahkan setiap sub-problem setahap demi setahap. Karena itu integrasi dapat dipandang dalam beberapa lapisan. Biasanya dimulai dengan membangun arsitektur integrasi pada lapisan terendah dan menaik secara gradual. Beberapa jenis integrasi yang terpenting adalah (Juric, 2007):

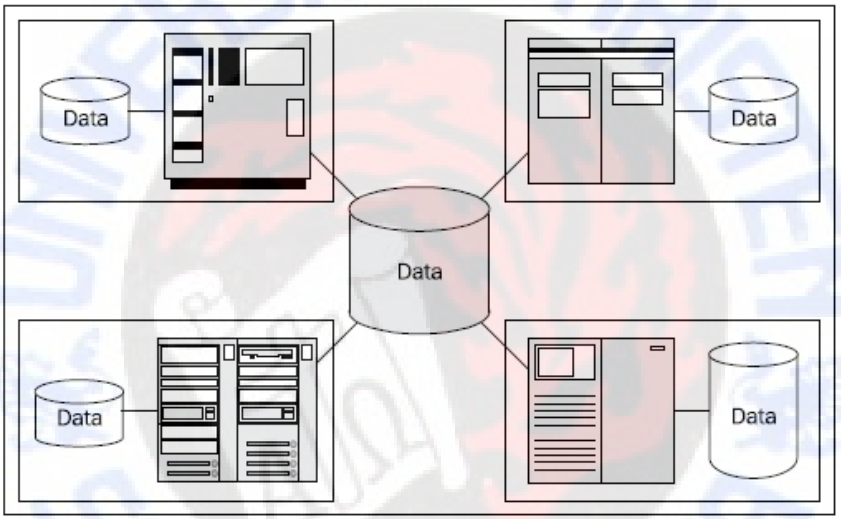
a. Integrasi Level Data
b. Integrasi Aplikasi
c. Integrasi Proses Bisnis
d. Integrasi Presentasi
e. Integrasi B2B

Arsitektur integrasi secara mendasar dapat dibedakan menjadi empat jenis yaitu (Binildas, 2008) : *Point-to-Point*, *Hub-and-Spoke*, *Enterprise Message Bus*, dan *Enterprise Service Bus*.

2.1 Jenis-jenis Integrasi

2.1.1 Integrasi Level Data

Data-level integration memusatkan pada perpindahan data antar aplikasi dengan tujuan membagi data yang sama ke beberapa aplikasi yang berbeda. Integrasi ini merupakan titik awal integrasi.



Gambar 2.1 Integrasi level data (Juric, 2007)

Dari sudut pandang teknis, integrasi level data ini secara relative lebih sederhana yang sudah sangat dikenal oleh kebanyakan pengembang. Mengakses basisdata secara relative lebih mudah dan ada beberapa tool yang memudahkan sharing data dan mempercepat. Selain itu, integrasi level data tidak memerlukan perubahan aplikasi.

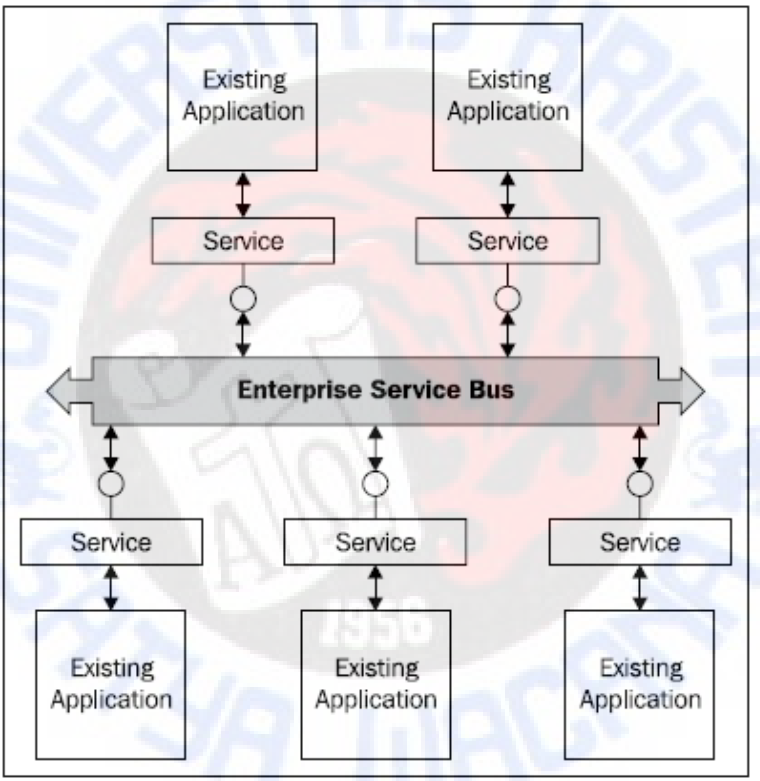
Masalah integrasi in terletak pada kompleksitas basisdata dan jumlah data. Untuk memindahkan data antar basisdata teknologinya belum begitu dikenal. Salah satu teknologi yang dapat digunakan untuk integrasi level data ini dikembangkan oleh konsorosium OGSA yang dikenal sebagai proyek OGSA-DAI (*Open Grid Services Architecture – Data Access and Integration*). OGSA-DAI merupakan framework yang diperluas untuk akses data dan integrasi data. OGSA-DAI digunakan untuk mengekspose sumber daya yang heterorgen untuk grid melalui *web services*. OGSA-DAI berinteraksi dengan sumber daya data dengan *query & update*, transformasi/kompresi data, *delivery* data dan fungsionalitas aplikasi tertentu.

2.1.2 Integrasi Aplikasi

Integrasi aplikasi memusatkan pada *sharing* fungsinalitas logika bisnis, dan tidak hanya data murni seperti pada integrasi level data. Integrasi aplikasi biasanya dicapai melalui penggunaan *application programming interfaces* (APIs). Aplikasi yang mengekspose fungsionalitasnya melalui API dapat mengakses ke fungsionalitas secara programatik tanpa menggunakan *user interface*.

Sebelumnya pengembang belum merealisasikan penggunaan API. Namun aplikasi yang lebih baru telah menerima konsep *service*s yang menyediakan untuk aplikasi lain. Melalui penggunaan API, fungsionalitas sistem yang ada dapat diakses. Namun API yang diekspose oleh aplikasi yang berbeda akan berbeda pula cara aksesnya.

Tujuan integrasi aplikasi ada dua yaitu : memahami dan menggunakan API untuk mengakses fungsionalitas yang dibutuhkan, dan membungkus teknologi yang berbeda yang digunakan untuk API dan aksesnya. Akses ini menggunakan *service* untuk mengekspose *interface* (API).

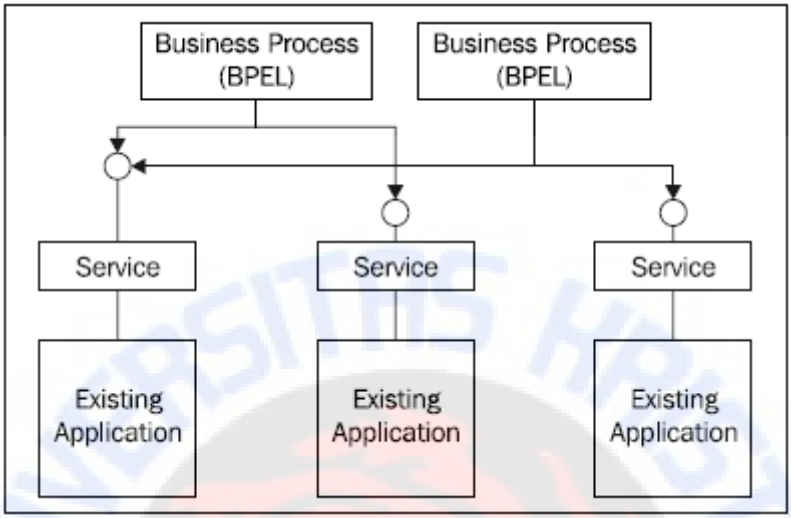


Gambar 2.2 Integrasi aplikasi (Juric, 2007)

2.1.3 Integrasi Proses Bisnis

Integrasi proses bisnis memungkinkan dukungan proses bisnis dalam *enterprise* dimana solusi yang ada merupakan bagian dari langkah proses bisnis. Integrasi ini mengekspose fungsionalitas sebagai abstraksi dari metode bisnis melalui *interface*.

Integrasi proses bisnis menampilkan sistem informasi *enterprise* seperti yang diinginkan atau seperti yang akan dibangun jika dapat membangunnya secara baru, dengan kebutuhan yang jelas untuk apa sistem terintegrasi ini diinginkan dengan dukungan teknologi modern. Ini berarti bahwa *interface* sistem informasi didasarkan pada arsitektur yang didesain baru. Namun, fungsionalitasnya tidak diimplementasikan ulang, namun menggunakan aplikasi yang ada. Aplikasi yang sudah ada dimodelkan ulang dengan cara yang dapat mengekspose fungsionalitas lapisan proses bisnis dan sesuai dengan arsitektur aplikasi modern. Akhirnya potongan-potongan yang berbeda direkatkan bersama, biasanya menggunakan pemodelan proses bisnis dan bahasa eksekusi (*execution langage*), semacam BPEL (*Business Process Execution Language*).



Gambar 2.3 Integrasi proses bisnis (Juric, 2007)

SOA, BPEL, dan teknologi terkait menyediakan kesempatan baru untuk membuat sistem informasi terintegrasi lebih fleksibel dan beradaptasi dengan perubahan proses bisnis. SIstem informasi dengan cara ini dapat lebih kokoh, menyediakan dukungan yang lebih baik untuk perubahan kebutuhan, dan lebih dekat dengan kebutuhan bisnis.

2.1.4 Integrasi Presentasi

Setelah mencapai integrasi proses bisnis, biasanya dilanjutkan dengan integrasi presentasi. Sebab aplikasi yang ada sekarang dimodelkan dan dibungkus pada lapisan tengah (*middle tier*), yang mengekspose *service*s melalui *interface* level tinggi. Ini menjadi penting dimana pengguna dapat melihat sebagai sistem informasi yang menyatu.

Integrasi presentasi menghasilkan sistem terintegrasi yang menyediakan lapisan presentasi yang menyatu, yang melalui ini pengguna dapat mengakses fungsionalitas dari sistem terintegrasi. Karena menggunakan lapisan presentasi yang dibangun baru, pengguna tidak menyadari keragaman aplikasi yang ada yang mengeksekusi di balik sistem. Lapisan presentasi juga mengakses fungsi melalui *interface* umum, yang disediakan oleh lapisan bisnis, yang dikembangkan dalam fase integrasi proses bisnis. Karena itu lapisan presentasi dipisah dan tidak sadar adanya rincian aplikasi yang ada dibaliknya.

Dengan pengembangan lapisan presentasi yang menyatu, maka fakta dibalik aplikasi yang berbeda disembunyikan. Dengan cara ini maka dapat dicapai efisiensi dan merupakan cara emngganti sebagian dari sistem *legacy* tanpa mempengaruhi bagian sistem yang lain.

Integrasi presentasi dipertimbangkan sebagai langkah definisi dan implementasi dari *user interface* umum dari sebuah portal untuk sistem informasi. Juga dipertimbangkan sebagai cara mengekstraksi data dari aplikasi yang ada melalui *user interface*.

2.1.5 Integrasi B2B

Saat ini, integrasi aplikasi didalam perusahaan tidak lagi mencukupi. Terdapat kebutuhan pertumbuhan yang memungkinkan integrasi antar perusahaan, yang sering diacu sebagai integrasi business-to-business (B2B) atau e-business. Kebutuhan saat ini tidak lagi sekedar melakukan *publish* catalog *off-line* pada halaman web.

Yang diharapkan adalah *online*, informasi *up-to-date* (terkini), efisien, handal dan berkualitas.

Tentu saja, prasyarat untuk e-business yang efisien atau integrasi B2B adalah sebuah sistem informasi *enterprise* yang terintegrasi, pada level proses bisnis. Hanya pada level integrasi ini yang memungkinkan pemrosesan *request* secara *on-demand*. Pelanggan saat ini mengharapkan respon segera dan tidak puas dengan pemrosesan *batch* dengan *delay* beberapa hari untuk konfirmasi order. Namun yang sering menjadi kasus adalah ketika *e-business* tidak didukung oleh sistem informasi *enterprise* yang terintegrasi secara efisien. Respon cepat, yang diraih melalui integrasi dari sistem *back-end* (*enterprise information*) dan *front-end* (*presentation*) merupakan kunci factor sukses.

Fakta penting lain adalah bahwa kebanyakan aplikasi *front-end* dapat menggunakan sistem yang ada (*legacy*) sebagai solusi *back-end*. Membuat integrasi antar sistem semacam ini akan menjadi kunci factor sukses. Respon cepat dan propagasi data cepat ke seluruh aplikasi yang terkait akan menjadi masalah utama.

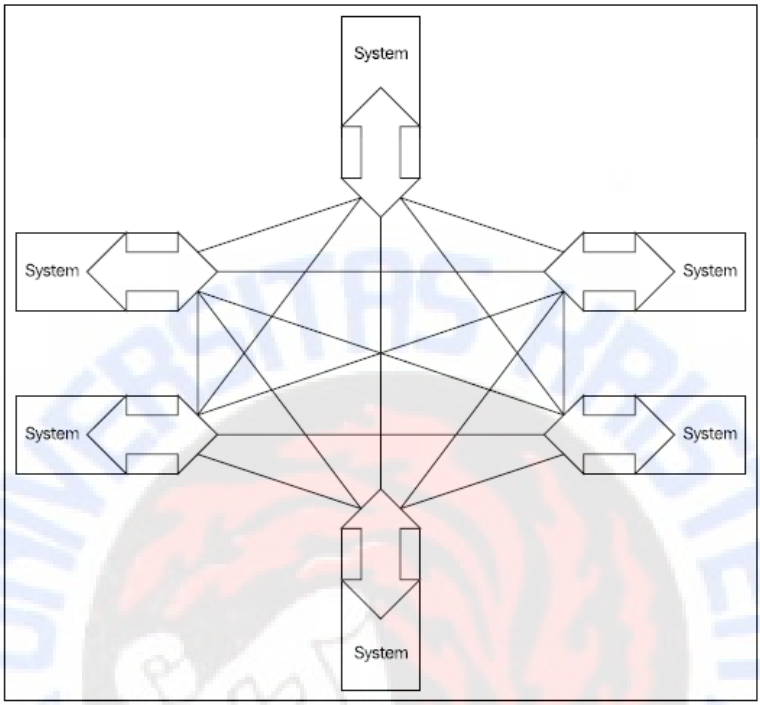
2.2 Arsitektur Integrasi

Untuk dapat memahami arsitektur integrasi adalah dengan memahami topologi integrasi yang berbeda. Arsitektur integrasi secara mendasar dapat dibedakan menjadi empat jenis yaitu (Binildas, 2008) : *Point-to-Point*, *Hub-and-Spoke*, *Enterprise Message Bus*, dan *Enterprise Service Bus*.

2.2.1 *Point-to-Point*

EAI secara tradisional telah dikerjakan dengan menggunakan integrasi secara *point-to-point*. Dalam integrasi *point-to-point*, maka terlebih dahulu didefinisikan solusi integrasi untuk sepasang aplikasi, sehingga ada dua *end point* yang akan diintegrasikan. Kemudian membangun protokol dan atau format *adapter*, atau *transformer* pada satu atau kedua *end points*. Ini merupakan cara termudah untuk integrasi, sepanjang jumlah yang diintegrasikan masih sedikit. Teknologi yang digunakan untuk integrasi ini biasanya adalah FTP, IIOP, *remoting* atau *batch interfaces*. Keunggulan arsitektur ini adalah integrasi secara *tight coupling*, sehingga kedua *end points* saling mengetahui pasangannya.

Gambar 2.4 berikut ini adalah merupakan gambar diagram dari integrasi *point-to-point*:



Gambar 2.4 Arsitektur integrasi *point-to-point*

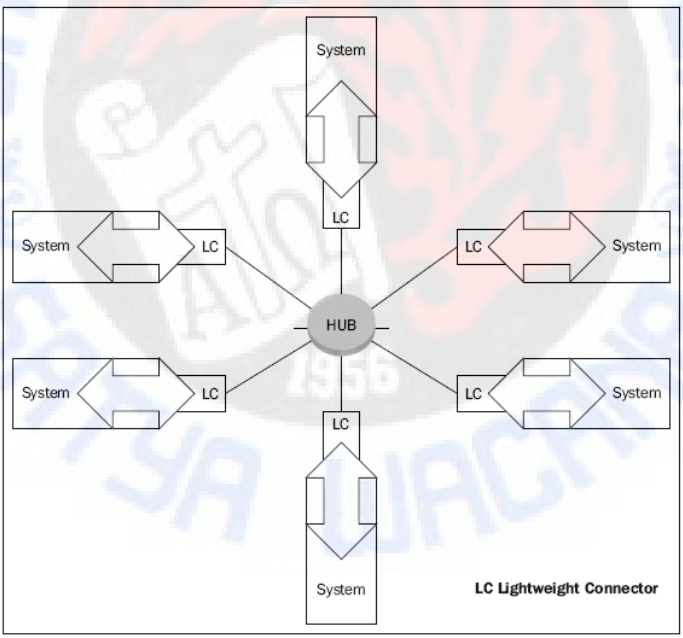
2.2.2 *Hub-and-Spoke*

Arsitektur *hub-and-spoke* juga dinamakan ***message broker*** dan mirip dengan arsitektur *point-to-point* ditambah dengan sebuah hub (broker) yang menghubungkan seluruh aplikasi. Fitur lain dari arsitektur *hub-and-spoke* adalah bahwa setiap aplikasi dihubungkan dengan *central hub* melalui konektor ringan (*lightweight connectors*). *Lightweight connectors* memudahkan integrasi aplikasi dengan sedikit perubahan atau tanpa perubahan pada aplikasi yang ada.

Transformasi *message* dan *routing* terjadi di dalam hub. Arsitektur ini merupakan peningkatan dari solusi *point-to-point* dengan mengurangi jumlah koneksi yang diperlukan untuk integrasi. Karena aplikasi tidak terkoneksi secara langsung dengan aplikasi lain, maka aplikasi dapat dihilangkan dari topologi integrasi dengan menghilangkan dari hub. Hal ini akan mengurangi kekacauan dalam setup integrasi.

Namun ada kelemahan dalam arsitektur *hub-and-spoke*, yaitu terletak pada sifat hub yang terpusat. Jika hub gagal maka keseluruhan integrasi juga gagal.

Gambar 2.5 berikut ini adalah gambar diagram dari integrasi *hub-and-spoke*:



Gambar 2.5 Arsitektur integrasi *hub-and-spoke*

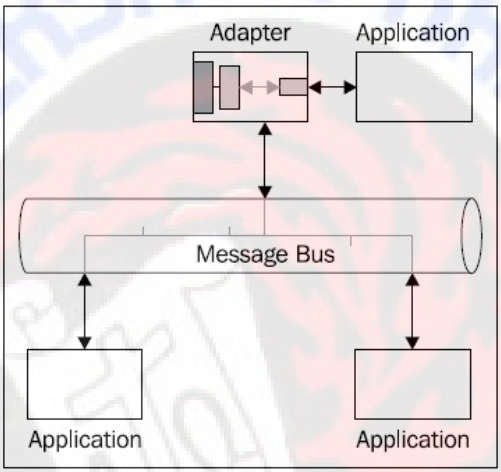
2.2.3 *Enterprise Message Bus*

Arsitektur *hub-and-spoke* menggunakan *lightweight connectors* untuk mengintegrasikan aplikasi melalui sebuah central hub, sehingga aplikasi dapat dengan mudah ditambahkan atau dihilangkan tanpa mempengaruhi yang lain. *Enterprise message bus* menyediakan infrastruktur komunikasi umum yang bertindak sebagai adapter antar aplikasi yang netral-platform dan netral-bahasa-pemrograman.

Infrastruktur komunikasi ini mencakup *message router* dan atau saluran *Publish-Subscribe*. Sehingga aplikasi berinteraksi satu sama lain melalui *message bus* dengan bantuan antrian *request-response*. Jika aplikasi konsumer ingin melakukan invoke ke *service* tertentu pada aplikasi provider yang lain, maka aplikasi konsumer mengirimkan *request message* dengan format yang cocok pada antrian *request* untuk *service* yang bersangkutan. Aplikasi provider kemudian mendengarkan untuk *response message* pada antrian *service's reply*. Aplikasi provider mendengarkan untuk *request* pada antrian *service's request*, menampilkan *service*, kemudian mengirimkan beberapa *response* untuk antrian *service's reply*.

Vendor terbaik untuk solusi antrian *message* untuk integrasi aplikasi dalam topologi *message bus* biasanya adalah IBM's Websphere MQ (WMQ) dan Microsoft MQ (MSMQ). Seperti yang ditunjukkan pada gambar 10 berikut ini, kadangkala aplikasi harus menggunakan *adapter* yang menangani skenario seperti invokasi

transaksi CICS. Adapter semacam ini akan menyediakan konektivitas antar aplikasi dan *message bus* menggunakan bus API dan aplikasi API berlisensi. *Message bus* juga memerlukan struktur perintah umum yang menyajikan operasi yang berbeda pada bus. Perintah ini mengatur invoke bus yang mencakup *listening* ke sebuah address, membaca *bytes* dari sebuah *address*, dan menuliskan *bytes* ke sebuah *address*.



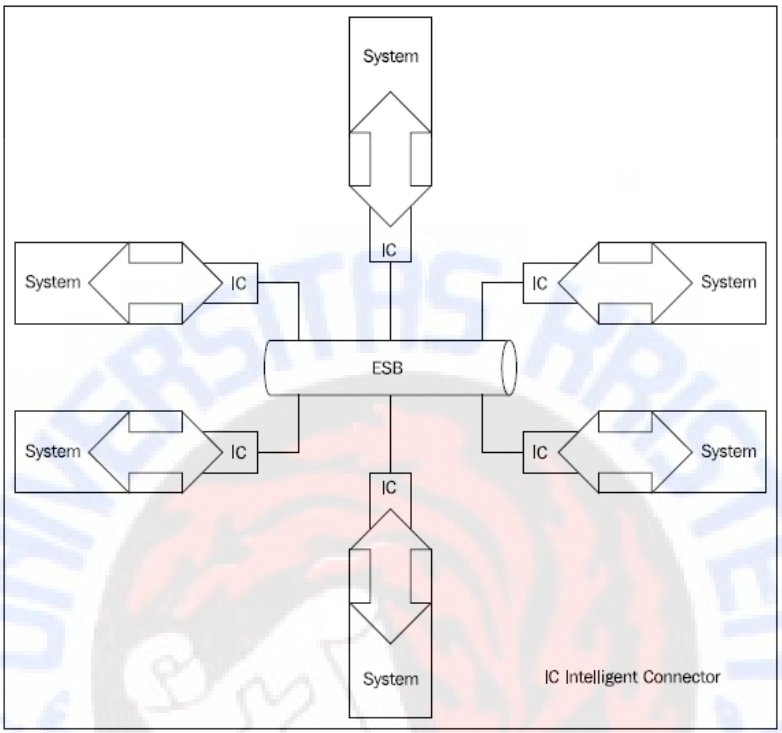
Gambar 2.6 Arsitektur integrasi *Enterprise Message Bus*

2.2.4 *Enterprise Service Bus*

Pendekatan *service*s bus untuk integrasi adalah menggunakan teknologi yang menyediakan bus untuk integrasi aplikasi. Aplikasi-aplikasi yang berbeda tidak berkomunikasi satu sama lain secara langsung melainkan berkomunikasi melalui *backbone middleware Service Oriented Architecture* (SOA). Fitur arsitektur ESB yang paling membedakan adalah sifat terdistribusi dari topologi integrasi.

Kebanyakan solusi ESB berbasis pada teknologi *Web Services Description Language* (WSDL) dan menggunakan format Extensible Markup Language (XML) untuk translasi dan transformasi *message*.

ESB merupakan sekumpulan *middleware services* yang menyediakan kemampuan integrasi. *Middleware services* ini merupakan jantung arsitektur ESB yang menempatkan *message* untuk dapat diroutekan dan ditransformasikan. Mirip dengan arsitektur hub-and-spoke, dalam arsitektur ESB, aplikasi melakukan koneksi ke ESB melalui ***intelligent connectors***. Connectors ini abstrak karena hanya mendefinisikan *transport binding protocols* dan *service interface*, dan bukan merupakan rincian implementasi real. Connectors ini intelegent karena mempunyai *logic built-in* pada ESB untuk melakukan bind *service* secara selektif pada saat run time. Kemampuan ini meningkatkan kecerdasan untuk aplikasi dengan memungkinkan *late binding* dari *service* dan melakukan penundaan pilihan *service*.



Gambar 2.7 Arsitektur integrasi *Enterprise Services Bus*

Fitur dan fungsionalitas utama yang didukung ESB akan membantu dalam memahami arsitektur ESB ini yaitu (Binildas, 2008):

a. *Addressing* dan *routing*
b. Jenis *synchronous* dan *asynchronous*
c. *Multiple transport* dan *protocol bindings*
d. *Content transformation* dan *translation*
e. *Business process orchestration*
f. *Event processing*

g. *Adapter* untuk berbagai platform
h. Integrasi dari *tool* perancangan, implementasi dan *deployment*
i. Fitur *Qualtiy of Services* seperti transaksi, keamanan dan *persistence*
j. *Auditing*, *logging*, dan *metering*
k. Manajemen dan monitoring

Enterprise Service Bus (ESB) merupakan infrastruktur software yang bertindak sebagai lapisan *intermediary* dari *middleware*. ESB menambahkan fleksibilitas yang memungkinkan koneksi *service* diimplementasikan dalam teknologi yang berbeda (seperti EJB, *messaging sistems*, komponen CORBA, dan aplikasi *legacy*) dalam cara yang mudah. ESB dapat bertindak sebagai mediator antar protokol dan produk *middleware* yang berbeda atau bahkan tidak kompatibel (Juric, 2007).

ESB menyediakan infrastruktur komunikasi antar *service* yang kuat, dapat diandalkan, aman dan dapat diperluas. ESB juga menyediakan kendali komunikasi dan kendali atas penggunaan *service*s yang mencakup (Juric, 2007) :

a. Kemampuan menangkap pesan (*message interception*), yang memungkinkan untuk menangkap pesan *request* untuk *service*s dan pesan *response* dari *service*, serta memberikan pemrosesan tambahan. Dengan cara ini, ESB dapat bertindak sebagai *intermediary*.
b. Kemampuan routing, yang memungkinkan ESB melakukan routing pesan ke *service*s yang berbeda didasarkan pada isi (*content*), asal, atau atribut lain.

c. Kemampuan transformasi, yang memungkinkan transformasi pesan sebelum dikirimkan ke *service*s. Untuk pesan format XML, transformasi semacam ini dilakukan menggunakan XSLT (*Extensible Stylesheet Language for Transformations*) atau mesin XQuery.
d. Kendali atas deployment, penggunaan dan pemeliharaan *service*s. Hal ini memungkinkan adanya *logging*, *profiling*, *load balancing*, *performance tuning*, ongkos penggunaan *service*s, *distributed deployment*, *on-the-fly reconfiguration*, dsb.
e. Fitur manajemen lain yang mencakup definisi korelasi antar pesan, definisi *path* komunikasi yang handal, definisi *security constraints* yang berkaitan dengan pesan dan *service*s, dsb.

ESB menyediakan dukungan untuk keragaman teknologi atas *service* yang diimplementasikan. ESB juga menyediakan konektor untuk range penggunaan teknologi yang luas, seperti komponen J2EE dan .NET, messaging *middleware*, aplikasi *legacy*, dan monitor *Transactions Processing*. ESB menyediakan fleksibilitas dalam melakukan *bind* beberapa kombinasi *service* tanpa adanya hambatan teknologi. ESB mendukung kombinasi dari model interaksi yang berbeda, seperti *queuing*, *routing*, dsb., tanpa perubahan *service* atau penulisan kode lagi.

ESB menyediakan *service* secara meluas. Ini berarti bahwa ESB akan mudah ditemukan, dan dikoneksikan. Dengan *service* yang tersedia secara meluas, ESB dapat meningkatkan penggunaan ulang dan menggabungkan *service* secara lebih mudah. Akhirnya, ESB

menyediakan kemampuan pengelolaan, seperti *message routing*, interaksi, dan transformasi.

ESB yang menyediakan fitur-fitur ini menjadi bagian esensial dari SOA. ESB ini menyediakan beberapa manfaat, termasuk fleksibilitas yang meningkat, mengurangi ongkos *deployment*, pengembangan dan pemeliharaan, dan meningkatkan kehandalan dan pengelolaan.