

BAB II

LANDASAN TEORI

II.1 Tinjauan Pustaka

Beberapa pustaka baik buku terbitan maupun artikel-artikel yang dipublikasikan di internet telah digunakan sebagai bahan referensi dalam penulisan tugas akhir ini. Usaha untuk menganalisa dan mengimplementasikan *prototype* integrasi sistem informasi yang dibangun di Universitas Atma Jaya Yogyakarta (SIMKA, SIPEM, SIATMA, dan CSIPAKAD) dapat dilakukan dengan menggunakan *Service-Oriented Architecture* (SOA) sebagai dasar integrasi sistem informasi yang ada.

Integrasi sistem informasi merupakan suatu mekanisme yang banyak digunakan sebagai pen jembatan antar aplikasi sehingga dapat berkomunikasi dan bertukar data/informasi satu sama lain. Terdapat beberapa mekanisme integrasi aplikasi, salah satunya adalah dengan *Service-Oriented Architecture*. (Hohpe, G., 2005)

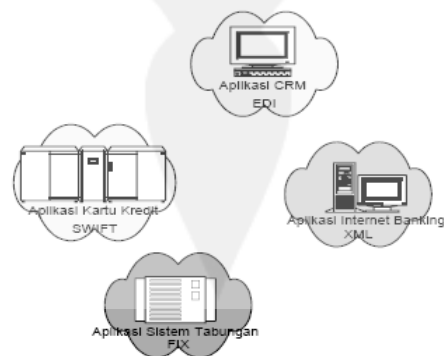
Dari berbagai integrasi yang ada seperti *File Transfer*, *Shared Database*, *Shared Business Function*, *Distributed Business Process* dan *Business to Business Integration*, SOA merupakan salah satu mekanisme integrasi aplikasi yang dapat berjalan pada *multiplatform*, *intranet/internet*, *loosely coupled*, interoperabilitas dan *indenpendent*. (Hohpe, G. et all, 2003)

SOA merupakan salah satu mekanisme integrasi yang sesuai untuk menjembatani aplikasi yang akan

dikembangkan dalam beberapa sistem di Universitas Atma Jaya Yogyakarta, memandang aplikasi yang ada dibangun dengan *multiplatform* (OS, bahasa pemrograman, pengembang aplikasi).

II.2 *Enterprise Application Integration (EAI)*

Dalam suatu perusahaan ataupun instansi pendidikan, sistem informasi memegang peran yang sangat penting dan digunakan sebagai suatu alat bantu kerja pada setiap bagian perusahaan atau instansi yang bersangkutan. Seperti halnya pada sebuah perusahaan berskala besar dan pasti memiliki lebih dari satu sistem informasi yang digunakan dan sistem - sistem tersebut dibangun pada waktu, tempat, *platform* dan bahasa pemrograman yang berbeda. Sebagai contohnya, sebuah Bank yang berskala sedang, setidaknya memiliki sebuah aplikasi perangkat lunak untuk bagian CIS(*Customer Information System*), Sistem Branch Teller, Sistem Credit Card, Sistem Loan, Sistem Giro(*Corporate Banking*), Sistem Tabungan(*Customer Banking*), Internet Banking, Customer Relationship Manager(CRM), Accounting, Procurement dan Human Resources.



Gambar 2.1. Contoh aplikasi - aplikasi yang dibangun pada sebuah Bank (Sumber: Bagus, I. dan Sudewa A. (2003))

Kompleksitas akan bertambah jika data dan bisnis proses tersebut diperlukan oleh bagian atau unit yang lainnya. Pertukaran data menjadi pokok persoalan utama saat manajemen ingin mengetahui data-data yang berasal dari masing-masing unit. *Manager* dapat mengambil keputusan berdasarkan data yang riil dan bukan berdasarkan kepada spekulasi semata.

Aplikasi *interface* ini mengambil masukkan data dari aplikasi yang sudah ada kemudian menampilkannya kembali di aplikasi yang memintanya dengan *format* yang sesuai. Sehingga data-data yang sudah dimasukkan di *level* unit bisa digunakan di *level* yang lebih atas dan data tersebut data merepresentasikan proses yang terjadi secara keseluruhan.

Apakah *Enterprise Application Integration* itu? Ada beberapa definisi yang dapat diperoleh, yaitu:

- 1). Sumber dari ComLabs - USDI Institut Teknologi Bandung, *Enterprise Application Integration* merupakan suatu proses memadukan aplikasi dan sumber data enterprise sehingga dapat dengan mudah berbagi bisnis proses dan data.
- 2). Sumber dari Ebizzasia, *Enterprise Application Integration* merupakan sebuah proses untuk mengintegrasikan sejumlah aplikasi berbeda di dalam perusahaan agar keseluruhannya dapat bekerja sama membentuk sebuah sistem tunggal yang holistik (utuh dan menyeluruh).
- 3). Sumber dari Wikipedia, *Enterprise Application Integration* didefinisikan sebagai suatu penggunaan prinsip dasar arsitektural pada

perangkat lunak dan sistem komputer untuk mengintegrasikan sejumlah aplikasi.

4). Sumber dari Iwaysoftware, *Enterprise Application Integration* merujuk pada perencanaan, metode, dan perangkat-perangkat yang digunakan untuk proses konsolidasi, modernisasi dan koordinasi aplikasi yang ada pada perusahaan.

5). Sumber dari Webopedia, *Enterprise Application Integration* adalah kebebasan pertukaran data dan bisnis proses di seluruh aplikasi atau *data source* yang terhubung pada sebuah organisasi.

Ada beberapa alasan yang membuat sebuah perusahaan perlu mengintegrasikan berbagai aplikasi yang berdiri sendiri ke dalam sebuah sistem yang terpadu, yaitu:

1). Adanya sejumlah rangkaian proses di dalam sebuah perusahaan yang melibatkan aliran data dan informasi dari satu entitas bisnis ke entitas bisnis lainnya (lintas sektoral), sehingga seluruh aplikasi yang terkait dengan entitas bisnis tersebut harus dapat bekerja sama.

2). Menghindari proses terjadinya aktivitas berulang yang tidak perlu (redudansi), misalnya memasukkan data yang sama ke dalam beberapa aplikasi yang berbeda, karena masing-masing aplikasi tersebut terkait dengan atau membutuhkan data tersebut.

- 3). Mengurangi terjadinya banyak kesalahan akibat tersebarnya berbagai data dan informasi yang sama pada sejumlah sistem berbeda.
- 4). Mereduksi total biaya investasi maupun pemeliharaan berbagai aplikasi yang berbeda.
- 5). Memudahkan proses konsolidasi data dan informasi yang dibutuhkan oleh berbagai pihak di dalam perusahaan.
- 6). Keperluan untuk menerapkan berbagai konsep bisnis terkini, seperti *Supply Chain Management*, *Enterprise Resource Planning*, *Customer Relationship Management*, dan lain sebagainya.

Namun dalam melakukan proses integrasi terhadap aplikasi-aplikasi tersebut, ada beberapa hal yang harus diperhatikan, yaitu:

- 1). Biaya yang dikeluarkan tidak terlalu mahal.
- 2). Tidak memerlukan banyak perubahan pada aplikasi yang telah ada.
- 3). Dapat menangani kepadatan lalu lintas dengan kehandalan yang tinggi.
- 4). Fleksibel, sehingga mudah dalam proses manajemen dan rekonfigurasi.

Ada beberapa cara untuk melakukan integrasi dengan dilakukannya beberapa metoda yang terbagi menjadi beberapa bagian yaitu:

1). *File transfer*

Sebuah perusahaan mempunyai banyak aplikasi yang dibangun secara terpisah, dengan bahasa pemrograman dan *platform* yang berbeda. Bagaimana cara mengintegrasikan aplikasi-aplikasi yang ada

sehingga dapat berkerja sama dan bertukar informasi?

Aplikasi-aplikasi yang ada pada suatu perusahaan merupakan suatu aplikasi yang dibangun pada waktu, teknologi, bahasa pemrograman, *platform* dan *programmer* yang berbeda. Oleh karena itu, hal ini menimbulkan kesulitan dalam berbagi informasi antar aplikasi.

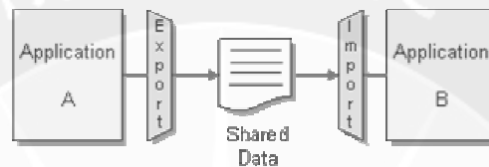
File merupakan *universal storage mechanism* yang tertanam pada semua sistem operasi dan dapat diakses oleh semua bahasa pemrograman. Ini merupakan salah satu cara yang paling sederhana untuk mengintegrasikan aplikasi yang ada. (Hohpe, G. and Woolf, B., 2003)

Hal yang harus diperhatikan dalam penggunaan *file* adalah *format* yang akan digunakan. Tidak semua aplikasi memerlukan *file* tersebut, jadi harus ditentukan prosesnya secara jelas.

Keuntungan pendekatan ini adalah tidak diperlukannya pengetahuan secara mendetail dari aplikasi tersebut, karena aplikasi yang dibangun sudah menyediakan hasil keluaran berupa *file* dengan *content* dan *format* yang sudah ditentukan sebelumnya, sehingga aplikasi yang membutuhkannya dapat langsung memproses *file* tersebut.

Alasan mengapa dengan *File transfer* merupakan metoda yang cukup sederhana adalah tidak diperlukannya *tools* tambahan ataupun paket integrasi, namun demikian developer harus bekerja lebih dalam pengembangan aplikasi tersebut dan penamaan *file* dan direktori yang ditentukan harus

valid dan unik. Aplikasi perlu mengimplementasikan sebuah mekanisme penguncian sebagai tanda bahwa *file* sedang diproses atau dibaca oleh aplikasi yang bersangkutan, sehingga aplikasi yang lain tidak dapat melakukan proses pembacaan *file* tersebut. (Hohpe, G. and Woolf, B., 2003)



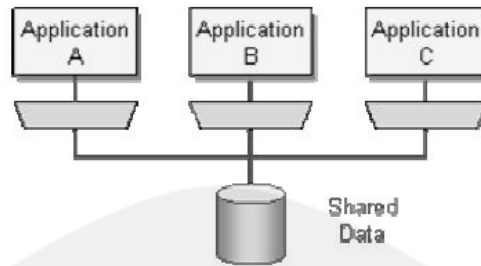
Gambar 2.2. File transfer (Sumber: Hohpe, G. and Woolf, B. (2003))

Hal penting yang harus diperhatikan dalam pendekatan ini adalah bagaimana mengatur semua *file* yang telah dibentuk dan memastikan bahwa *file* tersebut tidak hilang ataupun sedang dibaca.

2) . Shared Database

File transfer memberikan keleluasaan aplikasi untuk berbagi data, tetapi waktu yang diperlukan cukup lama. Selain itu, adanya keterbatasannya *format data*.

Hal tersebut dapat ditangani dengan *Shared Database*, yang mana integrasi aplikasi ini akan menggunakan sebuah *database* yang akan digunakan bersama - sama. *Error* yang muncul lebih mudah untuk ditemukan dan waktu yang diperlukan untuk meng-*update* program lebih cepat dan mudah.



Gambar 2.3. Shared Database (Sumber: Hohpe, G. and Woolf, B. (2003))

Shared Database dibuat menjadi lebih mudah digunakan oleh fungsi dari SQL yang merupakan dasar dari RDBMS (*Relational Databases Management System*) dan semua aplikasi yang dikembangkan dari berbagai macam *platform* dapat bekerja dengan menggunakan SQL, sehingga tidak perlu khawatir akan beragamnya *format file*. .(Hohpe, G. and Woolf, B., 2003)

Salah satu kesulitan dengan penggunaan *Shared Database* adalah penggabungan skema yang diperlukan oleh semua aplikasi.

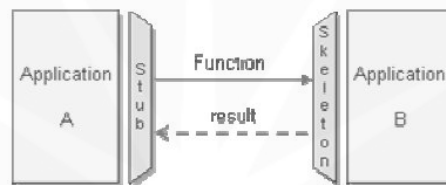
3) . Remote Procedure Invocation

Dengan *File transfer* dan *Shared Database*, aplikasi - aplikasi yang ada dapat berbagi data. Namun, hal itu tidaklah cukup. Keseringan terjadinya perubahan data mengakibatkan perubahan pada semua aplikasi yang terhubung. Perubahan alamat mungkin merupakan suatu perubahan yang sederhana atau pada registrasi *trigger* dan proses.

Salah satu kehandalan dari mekanisme terstruktur dalam *application design* adalah enkapsulasi. Pendekatan *Shared Database* tidak melakukan proses enkapsulasi data, sehingga akan menyulitkan dalam *me-maintain* aplikasi - aplikasi yang terintegrasi.

Mekanisme apa yang dibutuhkan oleh suatu aplikasi untuk memanggil suatu fungsi yang terdapat pada aplikasi yang lain, *passing* data yang akan di-*share* dan memanggil fungsi yang ada pada *receiver*. (Hohpe, G. and Woolf, B., 2003)

Hal tersebut dapat dipecahkan dengan mengembangkan setiap aplikasi sebagai objek atau komponen dalam skala besar dengan data yang terenkapsulasi dan menyediakan sebuah *interface* yang memungkinkan aplikasi yang lain untuk berinteraksi dengan aplikasi yang sedang berjalan.



Gambar 2.4. Remote Procedure Invocation (Sumber: Hohpe, G. and Woolf, B. (2003))

Metoda *Remote Procedure Invocation* menerapkan prinsip enkapsulasi untuk mengintegrasikan aplikasi. Jika sebuah aplikasi membutuhkan informasi yang dimiliki oleh aplikasi yang lain, maka akan meminta informasi tersebut secara langsung dari aplikasi yang diinginkan. Jika suatu aplikasi ingin memodifikasi data dari aplikasi yang lain, maka hal tersebut akan dilakukan oleh aplikasi *receiver*.

Selama pengembang *software* menggunakan *procedure call*, *Remote Procedure Invocation* merupakan metoda yang tepat untuk mengintegrasikan aplikasi yang ada. Namun, ada satu hal yang harus

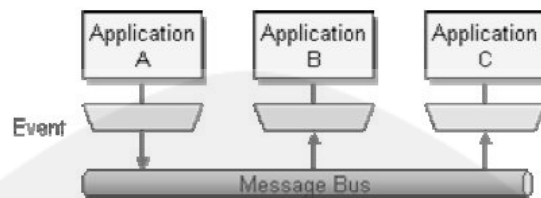
diperhatikan pada kinerja dan kehandalan antara *remote procedure call* dan *local procedure call*, jika pengembang tidak mengetahui akan hal tersebut, maka metoda dengan *Remote Procedure Invocation* akan mengakibatkan sistem menjadi lambat dan tidak handal. (Hohpe, G. and Woolf, B., 2003)

4). **Messaging**

File transfer dan *Shared Database* memungkinkan aplikasi untuk berbagi data, tetapi tidak untuk fungsionalitasnya. *Remote Procedure Invocation* memungkinkan aplikasi untuk berbagi fungsionalitas, tetapi masih *tightly couple* dalam prosesnya.

Yang diperlukan sekarang adalah suatu pendekatan yang seperti *File transfer* yang mana paket data yang dapat dibentuk dengan cepat, proses transfer yang mudah dan aplikasi *receiver* secara otomatis memberitahukan ketika sebuah paket data tersedia untuk digunakan, *Shared Database* yang mana skema dan detailnya mudah untuk diubah, *Remote Procedure Invocation* yang mana dapat menggunakan fungsionalitas yang ada pada aplikasi yang lain. Transfer data seharusnya bersifat asinkron, jadi pengirim tidak perlu menunggu penerima.

Pendekatan *Messaging* digunakan saat *transfer* paket data berulang kali, langsung, handal, asinkron dan *format* data yang mudah untuk diubah.



Gambar 2.5. *Messaging* (Sumber: Hohpe, G. and Woolf, B. (2003))

Assynchronous Messaging adalah suatu proses pengiriman pesan yang mana sistem yang mendapat pesan tidak perlu dalam keadaan *ON* atau *READY* pada waktu yang bersamaan. (Hohpe, G. and Woolf, B., 2003)

Dalam pendekatan ini memungkinkan terjadinya *decoupling*, sama halnya dengan *File transfer*. Namun, pesan tersebut dapat diubah pada saat pengiriman yang mana pengirim dan penerima tidak mengetahui akan perubahan tersebut.

Ketika memutuskan untuk menggunakan metoda *Messaging* sebagai integrasi sistem, ada beberapa hal yang harus dipertimbangkan dan melaksanakan segala sesuatu yang dapat diterapkan. Misalnya: (Hohpe, G. and Woolf, B., 2003)

- a) Bagaimana mentransfer paket data? Pengirim mengirimkan data ke penerima dengan mengirimkan sebuah pesan dengan *via Message Channel* yang menghubungkan pengirim dan penerima.
- b) Bagaimana cara mengetahui ke mana data dikirimkan? Jika pengirim tidak mengetahui alamat data yang akan dikirimkan, maka data tersebut dapat dikirimkan ke *Messenger Router*, yang akan merujuk ke penerima yang tepat.

- c) Bagaimana cara untuk mengetahui *format* data yang akan digunakan? Jika pengirim dan penerima tidak menentukan *format* data, maka pengirim dapat merujuk data ke *Message Translator*, yang mengubah *format* data yang sesuai di penerima.
- d) Bagaimana menghubungkan aplikasi yang ada ke sistem *Messaging*? Aplikasi yang akan menggunakan *Messaging* akan mengimplementasikan *Messaging Endpoints* untuk proses pengiriman dan penerimaan pesan.

Integrasi berarti menghubungkan sistem komputer, perusahaan dan manusia. Menurut Gregor Hoppe dan Booby Woolf dengan bukunya yang berjudul "*INTEGRATION PATTERNS Designing, Building and Deploying Messaging Solutions*", untuk membantu dalam merancang dan mengimplementasikan integrasi, ada 6 tipe yang dapat digunakan, antara lain:

1) .Information Portal

Pengguna harus mengakses lebih dari satu sistem yang ada untuk memperoleh informasi yang detail dan valid. Misalnya, memeriksa status pemesanan, seorang *customer service* harus mengakses sistem manajemen pemesanan ditambah dengan *log on* ke dalam sistem untuk mengatur pesanan yang kemudian informasinya akan ditampilkan di Web. *Information Portal* mengumpulkan informasi dari berbagai sumber yang

kemudian akan ditampilkan dalam satu *form* tersendiri. (Hohpe, G. and Woolf, B., 2003)



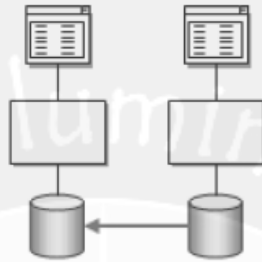
Gambar 2.6. Information Portal (Sumber: Hohpe, G. and Woolf, B. (2003))

Hal ini berguna untuk menghindari pengguna untuk mengakses sistem - sistem yang terhubung guna memperoleh informasi tersebut. *Information Portal* terbagi menjadi beberapa zona dimana setiap zona menampilkan informasi dari sistem yang berbeda. Untuk lebih canggihnya, sistem menyediakan *limited interaction* antar zona, misalnya seorang pengguna memilih sebuah *item* yang ada pada zona A, kemudian zona B di-*refresh* dan menampilkan informasi yang dipilih oleh user di zona A.

2) .Data Replication

Banyak sistem perlu mengakses data yang sama. Misalkan, alamat pelanggan yang digunakan di *customer care system* (ketika pelanggan ingin mengubahnya), *accounting system* (ketika menghitung biaya penjualan), *shipping system* (untuk label pengiriman), *billing system* (untuk mengirim faktur). *Database* sistem tersebut perlu menyimpan informasi yang berhubungan dengan pelanggan. Ketika pelanggan ingin mengubah alamat rumahnya, maka sistem yang membutuhkan data alamat tersebut

perlu diubah juga. Hal tersebut dapat diselesaikan dengan menerapkan integrasi sistem dengan replikasi data.



Gambar 2.7. Data Replication (Sumber: Hohpe, G. and Woolf, B. (2003))

Banyak cara untuk menerapkan integrasi dengan replikasi data, misalnya:

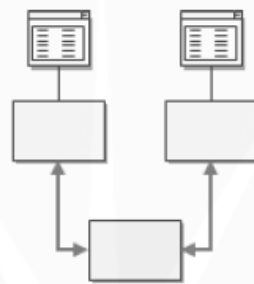
- Berapa vendor *database* membangun fungsi replikasi di *database*,
- Export data menjadi suatu *file* tertentu, kemudian di-*import* lagi ke sistem yang lain,
- Menggunakan *message-oriented*, dengan membawa data yang tersimpan dalam suatu pesan.

3) .Shared Business Function

Kebanyakan aplikasi menyimpan data redundan, yang cenderung juga mengimplementasikan fungsionalitas yang redundan. Sistem mungkin perlu memeriksa apakah *security number valid* atau tidak, apakah alamat yang diinputkan sesuai dengan kode pos dan apakah *item* tertentu ada atau tidak. (Hohpe, G. and Woolf, B., 2003) Sehingga

mengakibatkan untuk berbagi fungsionalitas yang diimplementasikan sebagai suatu layanan.

Misalkan permasalahan yang dihadapi sama seperti permasalahan pada replikasi data yaitu alamat pelanggan. Sehingga dapat membentuk suatu fungsionalitas, misalkan *"Get Customer Address"* yang dapat digunakan oleh sistem yang lain dari pada harus menyimpan data yang redundan.

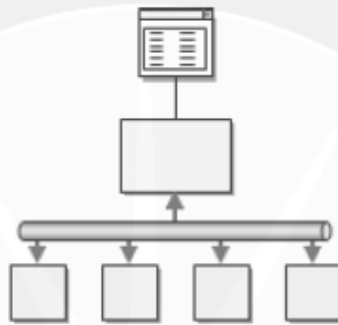


Gambar 2.8. *Shared Business Function* (Sumber: Hohpe, G. and Woolf, B. (2003))

4) *Service-Oriented Architecture*

Shared business function sering disamakan sebagai suatu layanan atau *service*. Namun, layanan adalah suatu fungsi *well-defined* yang dapat digunakan secara *universal* dan merespon permintaan tergantung dari permintaan pengguna. Misalkan, sebuah perusahaan mengumpulkan sekumpulan layanan yang akan digunakan kemudian mengelola layanan tersebut. Hal apa saja yang akan dilakukan. Pertama, layanan - layanan tersebut disimpan dalam sebuah direktori tertentu, kemudian setiap layanan perlu menggambarkan relasi dengan sistem atau aplikasi.

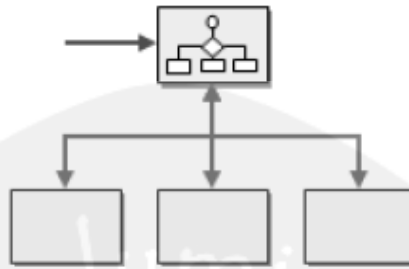
Sebuah aplikasi dapat dibangun dengan layanan yang sudah ada dari aplikasi yang lain. Oleh karena itu, pemanggilan layanan dibatasi oleh integrasi di antara kedua aplikasi. Selain itu, *Service-Oriented Architecture* biasanya menyediakan *tools* untuk pemanggilan layanan eksternal yang sederhana dengan pemanggilan fungsi lokal.



Gambar 2.9. *Service-Oriented Architecture* (Sumber: Hohpe, G. and Woolf, B. (2003))

5) *Distributed Business Process*

Salah satu hal penting dalam proses integrasi adalah transaksi bisnis tunggal yang dibangun, digunakan secara terdistribusi pada sistem yang berbeda. Berdasarkan contoh yang sebelumnya yaitu "place order" yang digunakan oleh sistem yang lain, semua fungsi yang relevan dimasukkan ke dalam aplikasi yang ada. Namun terdapat kegagalan dalam koordinasi antar aplikasi, sehingga dapat ditambahkan sebuah komponen manajemen bisnis proses yang mengelola eksekusi fungsi bisnis yang berjalan di sistem yang ada. (Hohpe, G. and Woolf, B., 2003)

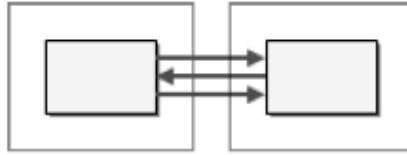


Gambar 2.10. *Distributed Business Process* (Sumber: Hohpe, G. and Woolf, B. (2003))

6) *Business-to-Business Integration*

Dalam beberapa kasus, fungsi bisnis mungkin tersedia dari *outside supplier* atau *business partner*. Misalnya, perusahaan perkapalan yang menyediakan layanan untuk pelanggan untuk menghitung biaya keberangkatan atau jalur pengiriman atau sebuah bisnis yang mungkin menggunakan *provider* dari luar untuk menghitung rata - rata biaya penjualan. Demikian juga, pada integrasi yang sering terjadi antar *business partner*. Pelanggan dapat menghubungi *retailer* untuk mendapatkan informasi mengenai biaya dan ketersediaan barang. Sebagai responnya, *retailer* dapat menanyakan ke *supplier* mengenai status dari pengiriman yang berisi barang diperlukan.

Pertimbangan tersebut merujuk pada *business-to-business integration*. Jadi, dengan kata lain *business-to-business integration* sering digunakan untuk integrasi dengan aplikasi *partner* bisnis atau aplikasi *supplier* luar. (Hohpe, G. and Woolf, B., 2003)



Gambar 2.11. *Business-to-Business Integration* (Sumber: Hohpe, G. and Woolf, B.(2003))

Dari keenam tipe integrasi tersebut, *Service-Oriented Architecture* merupakan salah satu metode integrasi yang sering digunakan pada perusahaan menengah atau besar. Hal ini dikarenakan oleh beberapa hal yaitu:

- 1). Mendukung klien dan *platform* yang heterogen
- 2). Dapat berjalan diatas *platform* internet
- 3). Mudah pengimplementasiannya
- 4). Murah

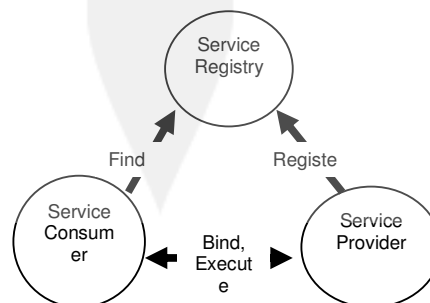
II.3 *Service-Oriented Architecture*(SOA)

Terdapat beberapa definisi mengenai *Service-Oriented Architecture*, antara lain:

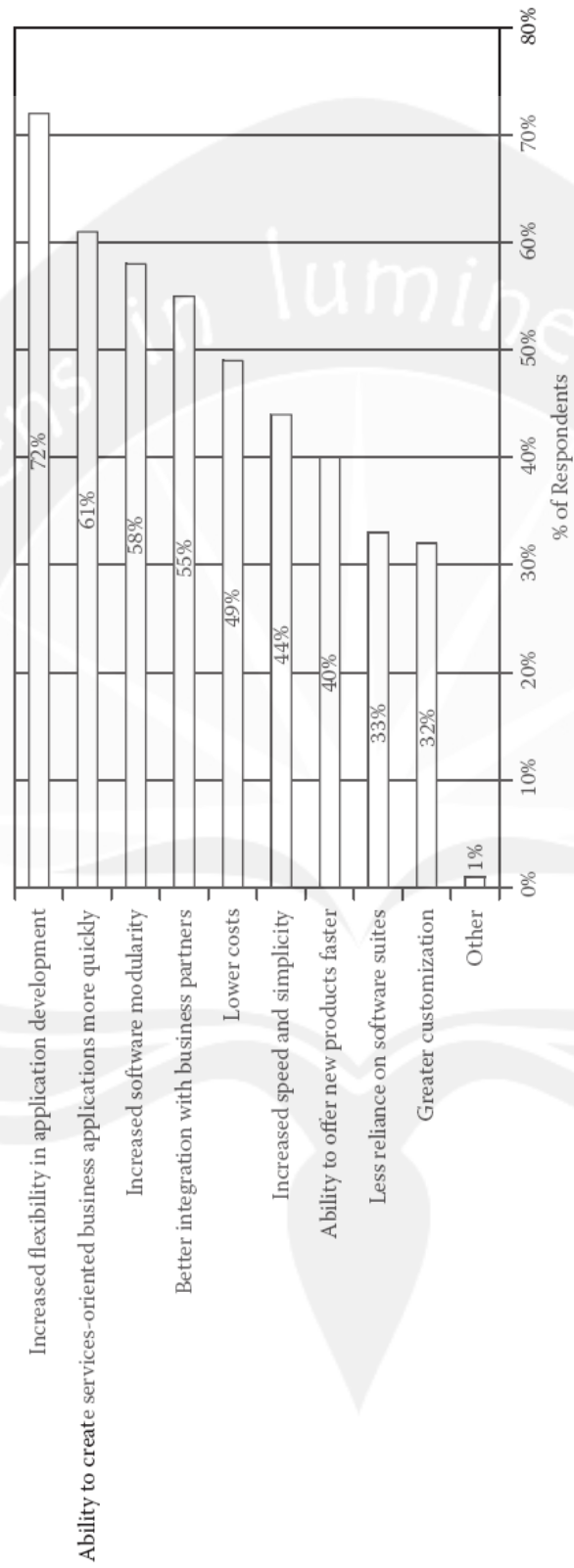
- 1). SOA merupakan salah satu arsitektur yang digunakan untuk membangun suatu aplikasi dengan menggunakan layanan yang tersedia di jaringan seperti pada Web. SOA bersifat *loose coupling* dan aplikasi yang berbasis SOA dibangun dengan layanan yang mengimplementasikan sebuah fungsionalitas bisnis yang *well-defined* dan dapat digunakan oleh aplikasi atau bisnis proses yang berbeda (Qusay, 2005).
- 2). SOA merupakan suatu arsitektur yang merepresentasikan fungsi dalam bentuk layanan. Fungsi menunjukkan abstraksi aktivitas yang secara alami dilakukan oleh aplikasi / program, individu dan organisasi. Layanan

mengabstraksikan fungsi dan dapat menunjukkan bentuk hubungan yang bermakna antara 2 pihak yang berkomunikasi (peminta dan penyedia layanan) (Edi, 2006).

- 3). SOA merupakan suatu cara untuk membangun sistem terdistribusi yang handal, menyediakan fungsionalitas berupa layanan dan *loose coupling* (Srinivasan, 2005).
- 4). SOA merupakan suatu arsitektur yang digambarkan dan diatur untuk mendukung *Web Service's dynamic, automated description, publication* dan *discovery*. SOA bertanggung jawab untuk menggambarkan bagaimana suatu *Web Service* digabungkan menjadi layanan yang lebih besar (Niemann, 2004).
- 5). SOA merupakan suatu pendekatan yang bersifat *loose coupling, protocol independent*, komputasi terdistribusi dimana *resource software* tersedia di jaringan sebagai suatu layanan (Hanson, 2003).
- 6). SOA merupakan suatu mode arsitektur yang bersifat *loose coupling* di antara beberapa integrasi sistem dengan menggunakan layanan (FAMIS Software.Inc, 2005).



Gambar 2.12. Proses Layanan SOA (Sumber: Brooks, M. (2004))



Gambar 2.13. Keuntungan dan tujuan dari SOA (Sumber: Information Week Research SOA/Web Service pada 200 perusahaan pada bulan September 2006)

Dari hal tersebut, dapat kita klasifikasikan keuntungan yang dapat diperoleh dalam menggunakan teknologi SOA, antara lain:

SOA dapat menggabungkan IT dengan misi dari suatu organisasi			
Lebih canggih, tidak ada redudansi, interaksi antar sistem dan mengurangi biaya operasi and perawatan.			
Kecanggihan	Heterogenitas Proses	Heterogenitas Data	Costs
Membuat sebuah jaringan untuk pemakai dan penyedia layanan Mudah untuk beradaptasi dengan perkembangan perubahan kebutuhan. Meningkatkan fleksibilitas bisnis melalui <i>plug and play architecture</i> dan menggunakan kembali layanan yang telah ada.	Dapat berinteraksi dengan sistem lain tanpa kostumisasi penggunaan waktu dan integrasi <i>point to point</i> Memfasilitasi integrasi dengan beragam solusi <i>via open IT standars</i>	Meningkatkan semantik pertukaran data selama proses eksekusi bisnis Menjaga konsistensi perpindahan data dari sistem yang berbeda. Menggunakan <i>best-of-breed software packages</i> untuk menghilangkan keberagaman IT(seperti <i>platform</i>, bahasa pemrograman, dan vendor)	Mengurangi biaya pengembangan fungsionalitas yang baru Biaya perawatan yang lebih murah.

Gambar 2.14. Keuntungan yang diperoleh dalam penggunaan Teknologi SOA (Sumber: Mocan,A. et all (2006))

Teknologi SOA merupakan salah satu teknologi yang berbasis layanan atau *Web Service*. Dalam membangun suatu *Web Service*, kita dapat menggunakan beberapa cara yaitu dengan *Representational State Transfer* (REST), WS-BPEL dan SOAP (*Simple Object Access Protocol*).

II.3.1 Web Service

Beberapa definisi mengenai *Web Service* :

1. Berdasarkan definisi W3C, *Web Service* adalah suatu sistem perangkat lunak yang didesain untuk mendukung interaksi mesin ke mesin pada suatu jaringan. *Web Service* mempunyai suatu *interface* yang diuraikan dalam suatu *format machine - processible* seperti WSDL. Sistem lain yang berinteraksi dengan *Web Service* dilakukan melalui *interface* / antar muka menggunakan pesan seperti pada SOAP. Pada umumnya pesan ini melalui HTTP dan XML yang merupakan salah satu standar *Web*. Perangkat Lunak aplikasi yang ditulis dalam berbagai bahasa pemrograman dan berjalan pada berbagai platform dapat menggunakan *Web Service* untuk pertukaran data pada jaringan komputer seperti *Internet* dalam cara yang serupa dengan komunikasi *inter-process* pada komputer tunggal.
2. Sedangkan menurut Michael C. Daconta, *Web Service* adalah aplikasi perangkat lunak yang dapat ditemukan, diuraikan dan diakses berdasarkan pada XML dan protokol standar *Web* pada intranet, ekstranet dan *Internet*. *Web Service* adalah aplikasi perangkat lunak yang tersedia pada *Web* yang melaksanakan fungsi yang spesifik. Pengertian dari "ditemukan, diuraikan dan diakses berdasarkan pada XML dan protokol standar *Web*". Dibangun pada XML, suatu standar yang didukung dan diterima oleh beribu-ribu vendor di seluruh dunia, *Web Service* pertama

fokus pada interoperabilitas. XML adalah sintaksis dari pesan dan HTTP, bagaimana aplikasi mengirimkan pesan XML, ke *Web Service* dalam rangka berkomunikasi. Teknologi *Web Service*, seperti UDDI dan ebXML registri, memungkinkan aplikasi untuk secara dinamis menemukan informasi tentang *Web Service*. Pesan sintaksis untuk suatu *Web Service* diuraikan di dalam WSDL. Ketika kebanyakan teknologi berpikir tentang *Web Service*, mereka berpikir tentang SOAP, bagian yang diakses dari *Web Service*. SOAP, yang dikembangkan sebagai *Simple Object Access Protocol* adalah protokol pesan berbasis XML untuk berkomunikasi dengan *Web Service*. SOAP adalah dasar untuk *Web Service*, karena SOAP merupakan suatu protokol yang dapat digunakan oleh banyak bahasa pemrograman.

3. Berdasarkan IEEE (*Institute of Electrical and Electronics Engineers*), *Web Service* merupakan salah satu *software system* yang dibangun untuk mendukung interaksi *inter-operability* antar aplikasi pada jaringan dan *loosely coupled*. *Interoperability* merupakan sebuah kemampuan dari dua atau lebih sistem atau komponen untuk bertukar informasi dan menggunakan informasi yang telah ditukarkan.

Dalam membangun *Web Service* terdapat 4 komponen utama yaitu UDDI, WSDL, SOAP dan XML.

- Provide a Directory of Services on the Internet
- Web Services are defined in terms of the formats and msg ordering
- Web Services consumers can send and receive messages using XML
- Built using open Internet protocols

UDDI
*Universal Description,
Design, and Integration*

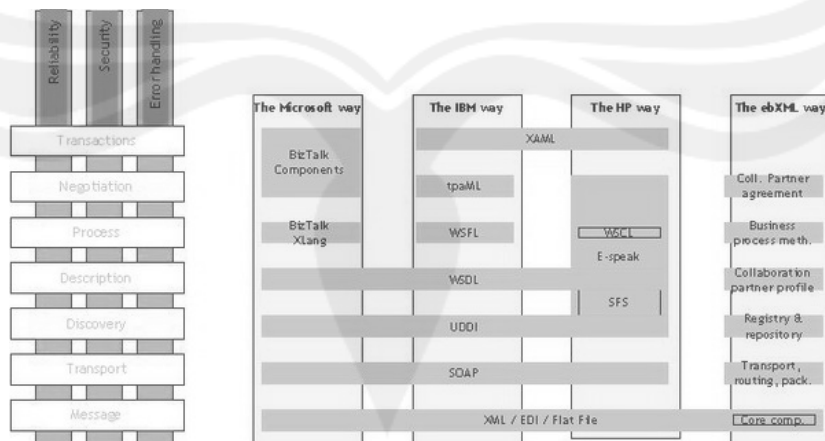
WSDL
*Web Services
Description Language*

SOAP

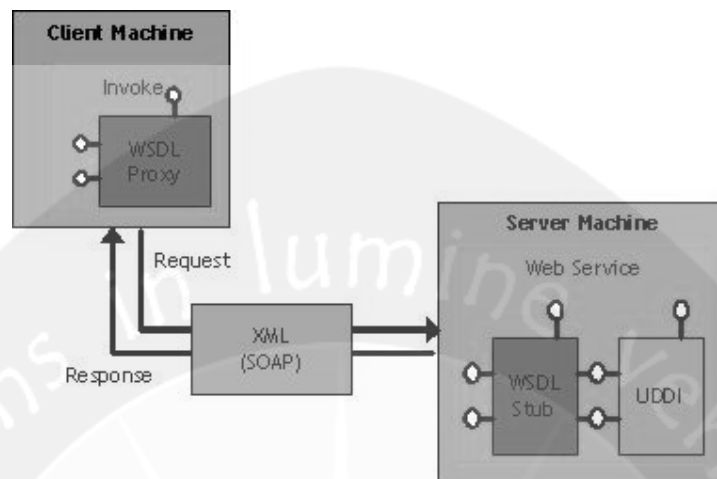
XML & HTTP

Gambar 2.15. Empat Komponen Utama *Web Service* (Sumber: Shentivel, G. (2006))

Keempat komponen utama tersebut dibangun oleh vendor yang berbeda-beda seperti yang ditunjukkan pada gambar di bawah.



Gambar 2.16. Vendor – vendor yang membangun keempat komponen utama *Web Service* (Sumber: Shentivel, G. (2006))



Gambar 2.17. Arsitektur Web Service (Sumber: Shentivel, G. (2006))

Berikut ini beberapa keuntungan dari penggunaan Web Service adalah

1. *Web Service* menyediakan interoperabilitas antar berbagai aplikasi perangkat lunak yang berjalan pada *platform* yang berbeda.
2. *Web Service* menggunakan standar dan protokol yang tersedia. Jika *format* protokol dan data berbasiskan teks, maka akan memudahkan pengembang untuk memahaminya.
3. Dengan memanfaatkan HTTP, *Web Service* dapat berjalan pada berbagai *firewall security* tanpa memerlukan perubahan pada *filtering firewall*.
4. *Web Service* mengizinkan perangkat dan layanan dari suatu perusahaan dan lokasi yang berbeda untuk digabungkan dalam penyediaan sebuah layanan yang terintegrasi.
5. *Web Service* mengizinkan untuk penggunaan kembali layanan dan komponen pada sebuah infrastruktur. *Web Service* adalah *loosely coupled* dengan

demikian akan memudahkan pendekatan yang terdistribusi ke integrasi aplikasi.

Berikut ini beberapa kerugian dari penggunaan *Web Service* adalah

1. Karakteristik standar *Web Service* saat ini masih dalam tahap perkembangan awal dibandingkan dengan *distributed computing open standards* seperti CORBA.
2. *Web Service* memiliki kinerja yang kurang dibandingkan dengan pendekatan komputasi terdistribusi lainnya seperti RMI, CORBA atau DCOM. Ini merupakan salah satu *trade-off* ketika memilih penggunaan *text-based format*.

Berikut contoh dua situs terkenal dalam penggunaan *Web Service*, yaitu Yahoo dan Amazon:

	Yahoo	Amazon
Protokol yang digunakan dalam Web Service	HTTP	HTTP
Standard pertukaran message	Menggunakan SOAP dan REST	Menggunakan SOAP
Cara mendaftarkan service	NA	NA
Cara mencari Service	belum lintas service	belum lintas service
Sudah mempertimbangkan semantic	belum	belum

Gambar 2.18. Penggunaan *Web Service* pada Yahoo dan Amazon

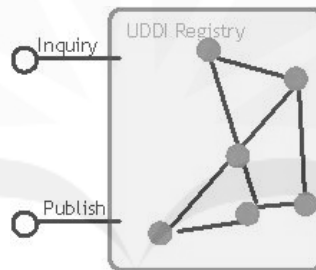
Jadi, *Web Service* merupakan:

1. Suatu aplikasi yang *programmable*, dapat diakses sebagai komponen melalui protokol standar web.
2. Menggunakan protokol standar web seperti HTTP, XML, dan SOAP.

3. Bekerja melalui *proxy* dan *firewall* yang ada.
4. Dapat mengambil keuntungan dari HTTP *authentication*.
5. Tersedia untuk berbagai klien (*platform independent*).

II.3.1.1 UDDI (*Universal Description, Discovery and Integration*)

UDDI merupakan salah satu infrastruktur dari *Web Service* yang berguna sebagai penyedia layanan, membuat *platform-independent*, *open framework for describing service, discovering bussines*, dan integrasi *business service*. (Shentivel, G., 2006)



Gambar 2.19. Universal Description, Discovery and Integration
(Sumber: Shentivel, G. (2006))

UDDI dapat mendaftarkan dan mencari sejumlah layanan yang ada dan menjelaskan bagaimana layanan atau aplikasi dapat saling berinteraksi satu sama lain di Internet. UDDI *business registration* terdiri dari 3 komponen, yaitu:

1. *White Pages* menunjukkan *address, contact* dan identifikasi.
2. *Yellow Pages* menunjukkan kategori berdasarkan hasil klasifikasi.
3. *Green Pages* menunjukkan informasi mengenai layanan yang dilakukan dalam proses bisnis.

UDDI merupakan salah satu bagian utama dari suatu *Web Service*. UDDI dibentuk untuk interaksi dengan SOAP *message* dan menyediakan akses ke dokumen WSDL.

Misalnya fungsi *Web Service* yang berhubungan (*get*, *set*) akan diregister dalam UDDI server. Kemudian server akan mempublikasikan *result set*. FYI, Windows Server 2003 terdiri dari *Enterprise Universal Description, Discovery, and Integration (UDDI) Services* yang merupakan suatu infrastruktur yang dinamis dan fleksibel untuk *Web Service*. UDDI dapat digunakan secara intranet maupun ekstranet. *Enterprise UDDI Service* membantu perusahaan dalam mengelola dan mendaftarkan *Web Service*.

II.3.1.2 WSDL (*Web Service Definition Language*)

Web Service secara keseluruhan dituliskan dengan dokumen berbasis XML atau sering disebut dengan dokumen *Web Service Description Language (WSDL)*. Ada beberapa bahasa pemrograman yang menghasilkan dokumen WSDL secara otomatis seperti Visual Studio .NET, PHP dengan Nusoap dan lain - lain.

Dalam SOA, dokumen WSDL merupakan dokumen yang sangat penting dan perlu dimengerti secara mendalam sehingga dapat menggunakan *Web Service* yang telah dibangun secara optimal.

Web Service dapat bertukar pesan melalui beberapa protokol yang berbeda pada SOAP yaitu HTTP POST, HTTP GET, dan SMTP (*Simple Mail Transfer Protocol*). (Vasiliev, 2007) Namun, haruslah diingat bahwa SOAP merupakan protokol yang paling tepat untuk pertukaran pesan berbasis XML. Dokumen WSDL menggambarkan secara

keseluruhan mengenai *Web Service*, termasuk operasi, pesan, dan tipe data. Pendekatan yang terbaik untuk memahami tentang WSDL adalah pemahaman mengenai elemen XML.

Dokumen WSDL merupakan dokumen XML, jadi harus mengikuti aturan dalam penulisan dokumen XML pula. Pertama akan dimulai dengan *schema namespace definition* sebagai element dasar dalam dokumen WSDL yang menggunakan elemen `<definition>`. Sebuah dokumen WSDL terdiri dari beberapa *schema definition*, namun yang paling penting adalah

```
<definition xmlns=http://schemas.xmlsoap.org/wsdl/>
```

Elemen dasar `<definition>` menyertakan isi dari dokumen WSDL. Kemudian, seluruh elemen berikutnya merupakan anak dari elemen `<definition>`.

Dokumen WSDL berisi 7 elemen utama yang terbagi menjadi 2 bagian, yaitu:

1. *Abstract Description*: Elemen XML yang berupa metode, parameter input, dan tipe data yang dikembalikan.
2. *Concrete Implementation*: Elemen XML yang menunjukkan kepada *client* bagaimana membangun dan menggunakan *Web Service*.

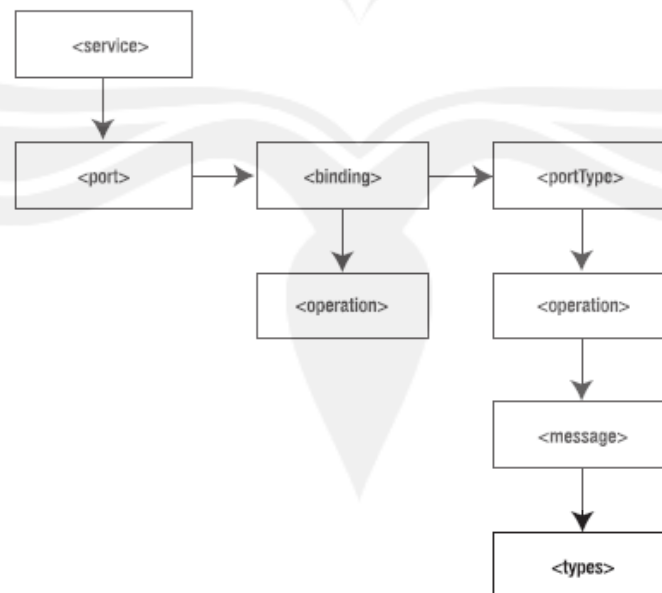
Terdapat 4 elemen XML untuk *abstract description*, yaitu:

1. `<types>` : Elemen ini akan mendaftarkan semua tipe data yang akan digunakan dalam pertukaran data. Elemen ini sama seperti XSD *schema definition file*.

2. `<message>` : Elemen ini menggambarkan pesan SOAP, yang mungkin berupa masukan, keluaran atau *fault message* untuk operasi *Web Service*.
3. `<operation>` : Elemen ini berupa pendefinisian masukan, keluaran dan *fault message* yang terhubung dengan operasi.
4. `<portType>` : Elemen ini akan mendaftarkan semua operasi pada *Web Service*.

Terdapat 3 elemen XML untuk *Concrete Implementation*, yaitu:

1. `<binding>` : Elemen ini menghubungkan elemen *abstract* dan *concrete* dalam dokumen WSDL.
2. `<port>` : Elemen ini mendefinisikan *Uniform Resource Identifier* (URI) di mana *Web Service* ditempatkan dan juga mengimplementasikan elemen `<binding>`.
3. `<service>` : Elemen ini menyertakan satu atau lebih elemen `<port>`.



Gambar 2.20. Struktur WSDL (Sumber: Vasiliev(2007))

1) . Elemen **<types>**

Elemen ini akan mendaftarkan seluruh tipe data yang akan digunakan dalam pertukaran data melalui pesan berbasis XML sebagai parameter inputan atau mengembalikan suatu tipe data. Elemen ini sama seperti *XSD schema definition file*. Jadi, kita dapat menentukan tipe data yang akan dipertukarkan dalam bentuk *file* XSD. Dalam dokumen WSDL, *file* tersebut dapat dimasukkan dengan menambahkan elemen dasar **<import>**.

```
<import
namespace=http://www.bluestonepartners.com/schemas
/StockTrader/
location="http://www.bluestonepartners.com/schemas
/StockTrader.xsd" />
```

Elemen ini tidak boleh dihilangkan biarpun tidak digunakan, karena elemen ini akan *generate parsing error* dalam dokumen WSDL.

2) . Elemen **<Message>**

Elemen ini menggambarkan pesan SOAP yang dapat berupa sebuah masukan, keluaran, atau sebuah pesan *fault message* untuk operasi *Web Service*. Pesan SOAP dibagi menjadi beberapa bagian yang direpresentasikan oleh elemen **<part>**.

Misalkan terdapat suatu fungsi *RequestQuote*. Klien ingin menggunakan fungsi *RequestQuote* tanpa peduli akan cara mengimplementasikannya. Tapi, klien perlu mengetahui bagaimana struktur pesan yang akan digunakan dalam menggunakan fungsi tersebut.

Fungsi *RequestQuote* menggunakan *request message* sebagai masukan dan *response message* sebagai keluaran.

Request message:

```
<message name="RequestQuoteSoapIn">
  <part name="Symbol" element="s1:Symbol" />
</message>
```

Response message:

```
<message name="RequestQuoteSoapOut">
  <part name="RequestQuoteResult" element="s1:Quote" />
</message>
```

3). Elemen <Operation>

Elemen ini akan mendefinisikan masukan, keluaran dan *fault message*. Selain itu, akan ditentukan parameter inputan dan tipe balikkannya.

Pada bagian sebelumnya kita telah menjelaskan penggunaan elemen <message> untuk pemanggilan fungsi *RequestQuote*.

```
<operation name="RequestQuote">
  <input message="tns:RequestQuoteSoapIn" />
  <output message="tns:RequestQuoteSoapOut" />
  <fault message="tns:ExceptionMessage" />
</operation>
```

Dalam dalam hal ini, proses *operation* dapat dibagi menjadi 4 bagian yaitu:

- a) *Request/Response* merupakan suatu proses di mana klien mengirimkan *request message* ke *Web Service* dan kemudian *Web Service* mengirimkan *response message*.

```

<operation name="MyOperation">
  <input message="MyInputMessage" />
  <output message=" MyOutputMessage" />
</operation>

```

- b) *One Way* merupakan suatu proses di mana klien mengirimkan *request message* ke *Web Service* namun *Web Service* tidak mengirimkan *response message*.

```

<operation name="MyOperation">
  <input message="MyInputMessage" />
</operation>

```

- c) *Solicit/Response* merupakan kebalikan dari *Request/Response* di mana *Web Service* mengirimkan *request message* dan kemudian klien mengirimkan *response message*.

```

<operation name="MyOperation">
  <output message=" MyOutputMessage" />
  <input message="MyInputMessage" />
</operation>

```

- d) *Notification* merupakan kebalikan dari *One Way* di mana *Web Service* mengirimkan sebuah pesan ke klien di mana tidak ada balasan dari klien.

```

<operation name="MyOperation">
  <output message=" MyOutputMessage" />
</operation>

```

4) . Elemen <portType>

Elemen ini berupa daftar semua operasi dari *Web Service* yang akan digunakan. Misalnya, nama *portType*-nya adalah *StockTraderServiceSoap* yang

mendukung dua operasi yaitu *RequestQuote* dan *PlaceTrade*.

```
<portType name="StockTraderServiceSoap">
  <operation name="RequestQuote">
    <input message="tns:RequestQuoteSoapIn" />
    <output message="tns:RequestQuoteSoapOut" />
    <fault message="tns:ExceptionMessage" />
  </operation>
  <operation name="PlaceTrade">
    <input message="tns:PlaceTradeSoapIn" />
    <output message="tns:PlaceTradeSoapOut" />
  </operation>
</portType>
```

Jika operasi yang diinginkan banyak, maka kita perlu menuliskan setiap operasi secara detail dan hal tersebut dapat membingungkan. Untuk menjadi lebih efisien maka proses operasi tidak perlu dimasukkan.

```
<portType name="StockTraderServiceSoap">>
  <operation name="RequestQuote" />
  <operation name="PlaceTrade" />
</portType>
```

5). Elemen **<binding>**

Elemen ini menghubungkan elemen *abstract* dan *concrete*. Elemen ini diasosiasikan dengan elemen **<portType>** dan juga terdapat alamat *Web Service* yang diasosiasikan dengan elemen **<portType>**. Kemudian, elemen ini mendaftarkan protokol-protokol yang digunakan untuk berkomunikasi dengan

Web Service. Misalnya untuk pemanggilan fungsi *RequestQuote* dengan protokol SOAP.

```
<binding name="StockTraderServiceSoap"
type="tns:StockTraderServiceSoap">
  <soap:binding
transport="http://schemas.xmlsoap.org/soap/http" style="document" />
  <operation name="RequestQuote">
    <soap:operation
soapAction="http://www.bluestonepartners.com/schemas/StockTrader/RequestQuote"
style="document" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
</binding>
```

6) . Element **<port>**

Elemen ini memeberikan URL dari lokasi *Web Service* berada dan juga mengimplementasikan elemen **<binding>**. Seperti yang kita ketahui bahwa kita sudah mendefinisikan elemen **<binding>** untuk sebuah *Web Service*, tetapi tidak mengindikasikan pada lokasi *Web Service*. Inilah kegunaan dari elemen ini, di mana akan diberikan lokasi dari *Web Service* tersebut berada.

```

<port name="StockTraderServiceSoap"
binding="tns:StockTraderServiceSoap">
    <soap:address
        location="http://localhost/StockTrader/StockT
        rader.asmx" />
</port>

```

7). Elemen <service>

Elemen ini menyertakan satu atau lebih elemen <port>.

```

<service name="StockTraderService">
    <port name="StockTraderServiceSoap"
        binding="tns:StockTraderServiceSoap">
        <soap:address
            location="http://localhost/StockTrader/S
            tockTrader.asmx" />
    </port>
</service>

```

Dalam penulisan dokumen WSDL, kita dapat menambahkan suatu *documentation* yang berisikan penjelasan mengenai proses yang akan dijalankan. Misalkan pada elemen *operation*:

```

<operation name="RequestQuote">
    <documentation>
        Returns a delayed 30-minute quote for a
        given stock ticker symbol.
        This operation returns a Quote XML type
        as defined in the XSD schema at:
        http://www.bluestonepartners.com/schemas
        /StockTrader.xsd
    </documentation>

```

```

    <input message="s1:RequestQuoteSoapIn" />
    <output message="s1:RequestQuoteSoapOut" />
  </operation>

```

Jadi, bentuk umum dalam membangun dokumen WSDL adalah

```

<definitions>
  <types />
  <message />
  <operation>
    <message />
  </operation>
  <portType>
    <operation />
  </portType>
  <binding>
    <operation />
  </binding>
  <service>
    <port>
      <binding />
    </port>
  </service>
</definitions>

```

II.3.1.3 SOAP (*Simple Object Access Protocol*)

SOAP merupakan suatu *Messaging protocol* yang digunakan untuk transfer data aplikasi dalam *format XML* pada *transport protocol*, seperti HTTP. Sekarang, aplikasi *Web Service* menerapkan SOA sebagai protokol standar untuk pertukaran informasi yang bersifat terpusat maupun terdistribusi. (Vasiliev, 2007)

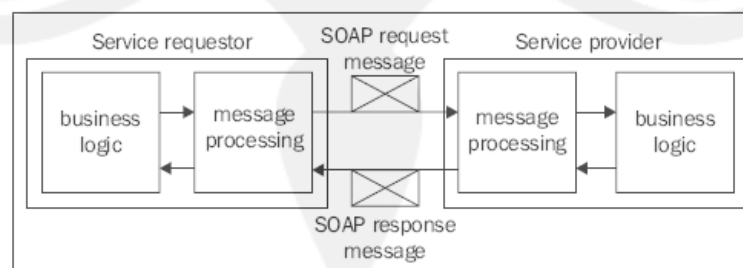
Struktur SOAP message terdiri dari *envelope*, *header* dan *body*.

```
<SOAP-ENV:Envelope ...>
  <SOAP_ENV:Header>
    ...
  </SOAP_ENV:Header>
  <SOAP_ENV:Body>
    ...
  </SOAP_ENV:Body>
</SOAP-ENV:Envelope ...>
```

Dari stuktur di atas, SOAP message terdiri dari beberapa elemen yaitu:

1. *Envelope* : Sebagai pembungkus dari keseluruhan pesan.
2. *Header* : Berupa *subelement* yang membawa pesan berupa metadata. Elemen ini bersifat *optional*.
3. *Body* : Berupa isi pesan yang dihasilkan oleh *Web Service* dan terkadang berupa *fault message* yang digunakan bila terjadi *error*.

Pattern yang sering digunakan dalam SOAP message adalah *request/response*.

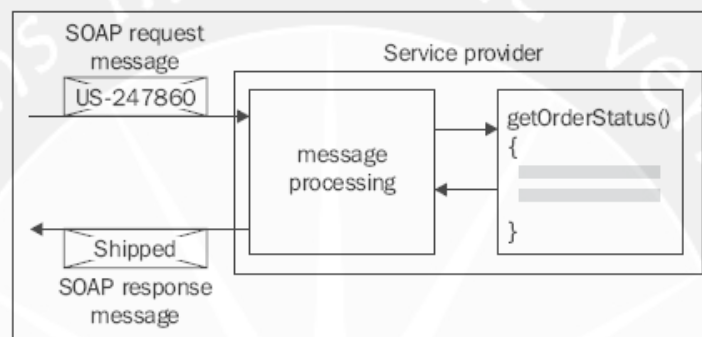


Gambar 2.21. Struktur SOAP (Sumber: Vasiliev(2007))

Seperti yang ditunjukkan oleh gambar di atas, *service requestor*, *service provider* dan *message processing logic* memerlukan proses *send/receive* dan

seperti yang kita ketahui bahwa hal ini sama seperti dari proses *pattern request/response*.

Misalkan sebuah *Web Service* mempunyai suatu fungsi yang bernama *getOrderStatus* yang mengambil jumlah *purchase order* sebagai parameter dan mengembalikan status *order* sebagai hasilnya.



Gambar 2.22. Proses Layanan dengan SOAP (Sumber: Vasiliev(2007))

Keterangan: (Vasiliev, 2007)

1. *Service requestor* mengirimkan *request message* berupa jumlah *purchase order* ke *service provider*.
2. *Service provider* memproses *request message* dan mengambil nilai PO dari *SOAP envelope*.
3. *Service provider* memanggil fungsi *getOrderStatus* dan nilai PO yang diambil, dijadikan sebagai parameter.
4. *Service provider* mengenkapsulasi nilai yang dihasilkan oleh fungsi *getOrderStatus* menjadi *response message*.
5. *Service provider* mengirimkan *response message* ke *requestor*.

Contoh penulisan request message:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
```

```

xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:getOrderStatus>
      <body>US-247860</body>
    </SOAP-ENV:getOrderStatus>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Dari contoh di atas, dapat dilihat bahwa SOAP message berisi dari nilai *purchase order* yang kemudian akan digunakan sebagai parameter untuk fungsi *getOrderStatus*. Berikut contoh *response message* yang akan dihasilkan:

```

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:getOrderStatusResponse>
      <body>Shipped</body>
    </SOAP-ENV:getOrderStatusResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Jika terjadi suatu kesalahan, maka diperlukan suatu *exception* untuk menanggulangnya. Seperti yang dijelaskan sebelumnya, bahwa pada elemen *body* bisa berupa suatu *fault message*. Misalkan terjadi kegagalan dalam koneksi ke *database*, maka *fault message* yang akan dibentuk adalah

```

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-
  ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>SOAP-ENV:Server</faultcode>
      <faultstring>Failed to determine the
        order status</faultstring>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

II.3.1.4 XML Schema Definition (XSD)

XSD merupakan salah satu dokumen yang berguna untuk mendefinisikan tipe dari suatu data dan biasanya dibentuk karena tipe data yang akan digunakan ataupun dikembalikan merupakan tipe data yang kompleks dan dengan hal tersebut dapat memudahkan *developer* dalam menuliskan dokumen WSDL. (Vasiliev, 2007)

```

<?xml version="1.0" encoding="utf-8"?>
<definitions name ="poService"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd1="http://localhost/Webservices/schema/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace=
  "http://localhost/Webservices/ch1/po.wsdl">
  <types>

```

```

<xsd:schema
  targetNamespace="http://localhost/Webservices
  /schema/">
  <xsd:element name="poInfo">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="pono"
          type="xsd:string" />
        <xsd:element name="shippingDate"
          type="xsd:string" />
        <xsd:element name="status"
          type="xsd:string" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
</types>
<message name="getOrderStatusInput">
  <part name="poNumber" element="xsd:string"/>
  <part name="poDate" element="xsd:string"/>
</message>
<message name="getOrderStatusOutput">
  <part name="poStatus" element="xsd1:poInfo"/>
</message>
...
</definitions>

```

Dalam contoh di atas, maka *response message* yang dikirimkan oleh *Web Service* ke *requestor* adalah <SOAP-ENV:Envelope

```

xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:getOrderStatusResponse>
      <poStatus>
        <pono>US-247860</pono>
        <shippingDate>21-jan-07</shippingDate>
        <status>Shipped</status>
      </poStatus>
    </SOAP-ENV:getOrderStatusResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Dari dokumen WSDL terdapat beberapa baris yang menunjukkan alamat dari XSD yang digunakan yaitu `http://localhost/WebService/schema/po.xsd`.

```

<?xml version="1.0"?>
  <schema
    targetNamespace="http://localhost/WebServices/sche
    ma/"
    xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="poInfo">
      <complexType>
        <sequence>
          <element name="pono" type="string"
            />
          <element name="shippingDate"
            type="string" />
          <element name="status"
            type="string" />
        </sequence>
      </complexType>
    </element>
  </schema>

```

```

        </complexType>
    </element>
</schema>
</schema>

```

Dengan adanya XDS akan membuat proses pembangunan integrasi sistem menjadi lebih fleksibel, *reusable* dan *modular solution*.

II.4 Contoh Implementasi SOA berbasis .NET dan PHP untuk layanan Penjumlahan Dua Bilangan

Dalam contoh ini, layanan akan dibangun pada *platform* .NET dengan bahasa C# dan akan digunakan oleh aplikasi lain dengan bahasa pemrograman PHP.

Layanan yang akan dibangun adalah sebuah layanan untuk memproses suatu perhitungan penjumlahan dua bilangan. Jika dituliskan dalam bahasa pemrograman C# adalah

```

public int Hitung(int a, int b)
{
    return a+b;
}

```

Fungsi tersebut akan dijadikan sebuah layanan / *service*, yang kemudian akan digunakan oleh aplikasi yang berbasis PHP.

Misalkan kita buat suatu *Web Service* project pada .Net dengan bahasa pemrograman C# dan nama *filenya* adalah SOAHitung2Bilangan. Kemudian tambahkan suatu fungsi yang bernama Hitung dengan parameter a dan b sebagai masukan dan nilai balikkannya adalah a+b.

```

[WebMethod(EnableSession= true)]
public int Hitung(int a, int b)

```

```

{
    return a + b;
}

```

Untuk .Net dokumen WSDL akan dibentuk secara otomatis setelah program dijalankan. Jika dijalankan, dokumen WSDL yang akan diperoleh adalah

```

<?xml version="1.0" encoding="utf-8" ?>
<wsdl:definitions
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  xmlns:tns="http://tempuri.org/"
  xmlns:s="http://www.w3.org/2001/XMLSchema"
  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  targetNamespace="http://tempuri.org/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
      targetNamespace="http://tempuri.org/">
      <s:element name="Hitung">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="a"
              type="s:int" />
            <s:element minOccurs="1" maxOccurs="1" name="b"
              type="s:int" />

```

```

        </s:sequence>
    </s:complexType>
</s:element>
<s:element name="HitungResponse">
<s:complexType>
<s:sequence>
    <s:element minOccurs="1" maxOccurs="1"
        name="HitungResult" type="s:int" />
</s:sequence>
</s:complexType>
</s:element>
<s:element name="int" type="s:int" />
</s:schema>
</wsdl:types>
<wsdl:message name="HitungSoapIn">
    <wsdl:part name="parameters" element="tns:Hitung" />
</wsdl:message>
<wsdl:message name="HitungSoapOut">
    <wsdl:part name="parameters"
        element="tns:HitungResponse" />
</wsdl:message>
<wsdl:message name="HitungHttpGetIn">
    <wsdl:part name="a" type="s:string" />
    <wsdl:part name="b" type="s:string" />
</wsdl:message>
<wsdl:message name="HitungHttpGetOut">
    <wsdl:part name="Body" element="tns:int" />
</wsdl:message>
<wsdl:portType name="ServiceSoap">
<wsdl:operation name="Hitung">
    <wsdl:input message="tns:HitungSoapIn" />

```

```

        <wsdl:output message="tns:HitungSoapOut" />
    </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="ServiceHttpGet">
<wsdl:operation name="Hitung">
    <wsdl:input message="tns:HitungHttpGetIn" />
    <wsdl:output message="tns:HitungHttpGetOut" />
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="ServiceSoap"
    type="tns:ServiceSoap">
    <soap:binding
        transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="Hitung">
        <soap:operation
            soapAction="http://tempuri.org/Hitung"
            style="document" />
    <wsdl:input>
        <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
        <soap:body use="literal" />
    </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="ServiceSoap12"
    type="tns:ServiceSoap">
    <soap12:binding
        transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="Hitung">

```

```

    <soap12:operation
      soapAction="http://tempuri.org/Hitung"
      style="document" />
  <wsdl:input>
    <soap12:body use="literal" />
  </wsdl:input>
  <wsdl:output>
    <soap12:body use="literal" />
  </wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:binding name="ServiceHttpGet"
  type="tns:ServiceHttpGet">
  <http:binding verb="GET" />
  <wsdl:operation name="Hitung">
    <http:operation location="/Hitung" />
  </wsdl:operation>
  <wsdl:input>
    <http:urlEncoded />
  </wsdl:input>
  <wsdl:output>
    <mime:mimeXml part="Body" />
  </wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="Service">
  <wsdl:port name="ServiceSoap"
    binding="tns:ServiceSoap">
    <soap:address
      location="http://localhost:1041/SOAHitung2Bilangan/
      Service.asmx" />
    </wsdl:port>
  </wsdl:service>

```

```

<wsdl:port name="ServiceSoap12"
  binding="tns:ServiceSoap12">
  <soap12:address
    location="http://localhost:1041/SOAHitung2Bilangan/
    Service.asmx" />
  </wsdl:port>
<wsdl:port name="ServiceHttpGet"
  binding="tns:ServiceHttpGet">
  <http:address
    location="http://localhost:1041/SOAHitung2Bilangan/
    Service.asmx" />
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

Untuk SOAP message yang dihasilkan jika terjadi penggunaan layanan tersebut adalah

Request Message:

POST /SOAHitung2Bilangan/Service.asmx HTTP/1.1

Host: localhost

Content-Type: text/xml; charset=utf-8

Content-Length: **length**

SOAPAction: <http://tempuri.org/Hitung>

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<soap:Envelope
```

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance"
```

```
  xmlns:xsd=http://www.w3.org/2001/XMLSchema
```

```
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelo
  pe/>
```

```
<soap:Body>
```

```
<Hitung xmlns="http://tempuri.org/">
```

```

        <a>int</a>
        <b>int</b>
    </Hitung>
</soap:Body>
</soap:Envelope>
Response Message:
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <HitungResponse xmlns="http://tempuri.org/">
      <HitungResult>int</HitungResult>
    </HitungResponse>
  </soap:Body>
</soap:Envelope>

```

Jika nilai $a = 100$ dan $b = 6$ maka nilai yang akan dikembalikan oleh fungsi Hitung adalah 106.

```

<?xml version="1.0" encoding="utf-8" ?>
<int xmlns="http://tempuri.org/">106</int>

```

Untuk program PHP dapat dituliskan sebagai berikut
(nama *file* hitung.php)

```

<html>
<head>
<title>PHP Hitung Dua Bilangan </title>
</head><body><?php
//Define the request

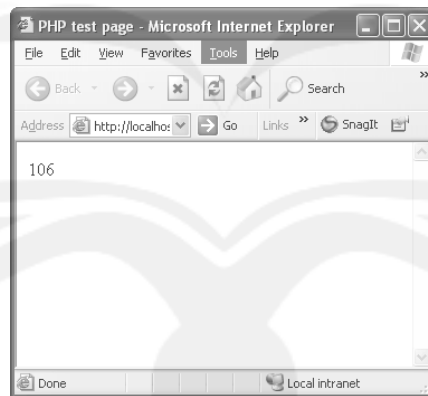
```

```

$a=100; $b=6;
$wsdl="http://localhost:1041/SOAHitung2Bilangan/Service.asmx?wsdl";
$client = SoapClient($wsdl);
$vem = $client->Hitung('a'=>$a,'b'=>$b);
print"<pre>";
print_r($vem);
print"</pre>";
?></body></html>

```

Variabel *request* merupakan suatu variabel yang berisikan *file* XML yang merupakan hasil *generate* dari layanan Hitung2Bilangan pada .NET dengan parameter masukkan a dan b yang masing - masing bernilai 100 dan 6. Dan variabel \$vem merupakan *storage* untuk hasil penjumlahan a dan b dalam bentuk *file* xml dan hasil yang diperoleh adalah 106.



Gambar 2.23. Hasil Pemanggilan Layanan pada platform .Net dengan pemrograman berbasis PHP