BAB 1
Service Oriented Architecture

1.1 Evolusi SOA

Dengan melakukan penelusuran evolusi pola-pola integrasi, maka dapat ditunjukkan bahwa SOA merupakan teknik integrasi yang dibangun berdasarkan teknologi integrasi untuk sistem komputasi terdistribusi. Menurut Roshen (2009) beberapa teknologi komputasi terdistribusi yang berkaitan dengan SOA adalah pemrograman soket, *Remote Procedure Call (*RPC*)*, *Object Request Broker* (ORB) dan *Asynchronous Messaging*. Teknologi komputasi terdistribusi tersebut memberikan kontribusi terhadap berbagai konsep integrasi berbasis SOA. Pembahasan mengenai soket, RPC, ORB dan *Asynchronous Messaging* akan mengacu pada Juric et al (2007), Roshen (2009), dan Kumar et al (2010).

Integrasi perusahaan sendiri merupakan perancangan aplikasi baru atau modifikasi aplikasi yang sudah ada sehingga aplikasi ini dapat saling berbagi data dan fungsionalitas. Sebelum munculnya pemrograman soket sudah ada upaya-upaya pengintegrasian dalam arti berbagi data yaitu metode berbagi data berbasis file dan basisdata.

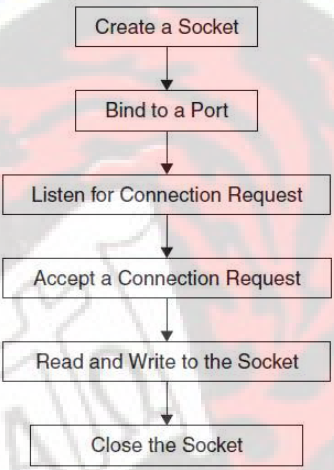
Metode berbagi data berbasis file merupakan metode paling lama. Metode berbagi data berbasis file dan menggunakan basisdata merupakan

metode berbagi data yang bukan untuk waktu-nyata. Jika memerlukan secara waktu-nyata maka harus dibantu dengan metode soket yang menyediakan koneksi waktu-nyata antar aplikasi.

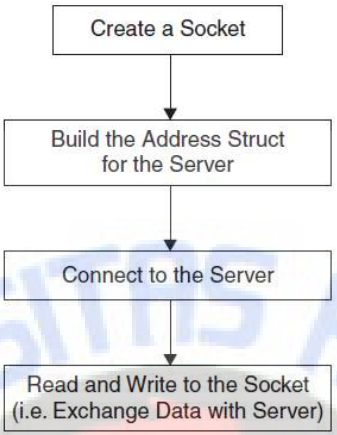
Metode pertama berbagi data melalui file ini merupakan metode yang paling umum karena penyimpanan data di file sudah mendunia. Jenis penyimpanan ini diperbolehkan oleh semua sistem perangkat keras dan sistem operasi. Pada metode berbagi data ini, sebuah aplikasi menuliskan data ke file, sedangkan aplikasi lain yang berjalan pada mesin yang sama dapat melakukan baca dan tulis di file tersebut. Walaupun metode berbagi data melalui file merupakan metode paling umum, namun metode ini mempunyai kelemahan yaitu data tidak dapat dibagi secara waktu-nyata.

Kemudian disusul dengan metode berbagi data melalui basisdata. Metode ini mirip dengan metode sebelumnya metode file. Pada metode basisdata ini, suatu aplikasi menuliskan data ke basisdata, dan aplikasi lain membaca data dari basisdata tersebut. Perbedaan penting dengan metode file adalah basidata berjalan pada mesin yang terpisah. Hal ini berarti transfer data antar aplikasi selalu terjadi melalui jaringan, sedangkan metode file membagi data pada mesin yang sama. Oleh karena itu, metode ini lebih lambat dari pada metode file. Keunggulan metode ini adalah koneksi antar aplikasi tidak secara *point-to-point*, seperti metode file. Sejumlah aplikasi dapat membagi data yang dituliskan pada basisdata. Penggunaan basisdata untuk integrasi aplikasi cukup terkenal karena penggunaan relasi basisdata berbasis SQL. Kebanyakan platform pengembangan mendukung SQL, sehingga tidak perlu mengkawatirkan adanya beberapa format file yang berbeda. Karena itu, sekali mempelajari SQL, akan dapat digunakan untuk platform yang berbeda.

Menurut Roshen (2009) komputasi terdistribusi dimulai dengan pengembangan pemrograman soket yang memungkinkan aplikasi membangun koneksi langsung dan berbagi data secara waktu-nyata (Gambar 1.1 dan 1.2). Membangun koneksi melalui soket merupakan dasar dari ide mengenai layanan-layanan dan SOA. Hal ini disebabkan pengembangan layanan dan SOA berikutnya menggunakan landasan soket. Jadi sulit membayangkan ide layanan dan SOA saat ini akan berkembang tanpa adanya soket.

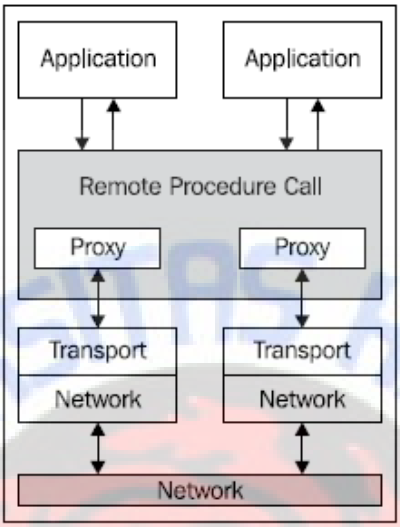


Gambar 1.1 Sisi server soket untuk berbagi data waktu-nyata (Roshen, 2009)



Gambar 1.2 Sisi *client* soket untuk berbagi data waktu-nyata (Roshen, 2009)

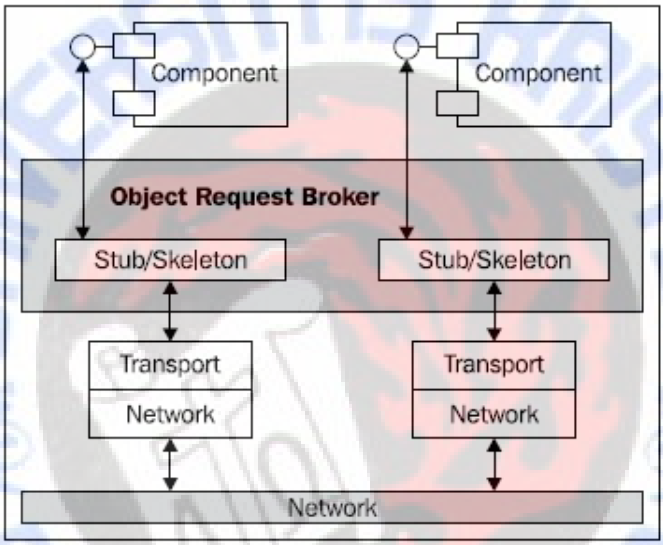
Soket hanya memungkinkan berbagi data saja. Soket tidak memungkinkan berbagi fungsionalitas secara langsung. Karena itu pengembangan selanjutkan diperlukan aplikasi yang dapat berbagi fungsionalitas. Dan teknologi berikutnya yang dapat berbagi fungsionalitas adalah RPC yang dikenal sebagai pemrograman *client-server*. RPC dibangun dengan landasan soket dan menyembunyikan pemrograman jaringan level rendah (lihat Gambar 1.3). Berkaitan dengan berbagi fungsionalitas antar aplikasi, RPC juga memperkenalkan cara elementer dalam mendeklarasikan antarmuka layanan dan ide platform independen melalu penggunaan *external data representation* (XDR).



Gambar 1.3 *Remote Procedure Call* (Juric et al, 2007)

Setelah RPC, disusul dengan teknologi *Object Request Broker* (ORB) yang memperkenalkan ide pemrograman berorientasi obyek ke dalam komputasi terdistribusi. Secara khusus, teknologi ORB (lihat Gambar 1.4) memperluas ide obyek dalam pemrograman berorientasi obyek ke obyek jarak jauh. Pada obyek jarak jauh ini, obyek dapat berada di aplikasi berbeda yang berjalan di komputer yang berbeda pula. Teknologi ORB telah membuat obyek jarak jauh ini dapat berkomunikasi satu sama lain. Obyek jarak jauh ini dapat berbagi fungsionalitas dan data dalam banyak cara seperti teknologi RPC. Contoh teknologi ORB yang sangat terkenal adalah CORBA dan Java RMI. Secara khusus, CORBA memperkenalkan sejumlah ide baru yang berkaitan dengan layanan-layanan dan SOA seperti antarmuka layanan yang bebas bahasa pemrograman, konsep awal tentang *registry* dan pemisahan aplikasi

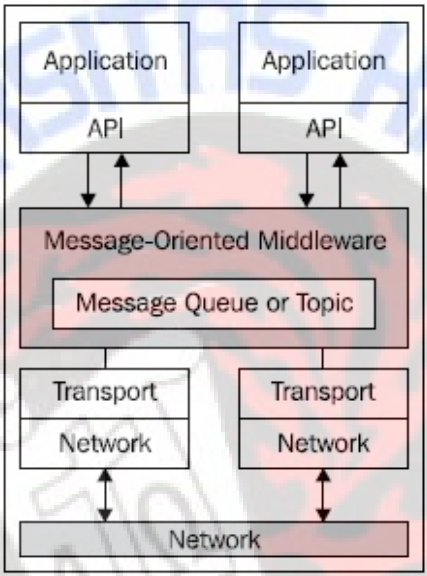
yang berbeda dari fungsionalitas yang berkaitan jaringan, dan kode untuk *marshalling* dan *unmarshaling* yang sangat meningkatkan penggunaan kembali kode karena kode yang sama dapat digunakan oleh sejumlah aplikasi yang berbeda. Server aplikasi semacam *WebSphere Application Server* dan JBoss menggunakan teknologi ORB.



Gambar 1.4 *Object Request Broker* (Juric et al, 2007)

Paralel dengan pengembangan teknologi ORB, *Asynchronous Messaging* juga dikembangkan, yang dikenal sebagai teknologi *Message Oriented Middleware* (MOM). Teknologi MOM ini (lihat Gambar 1.5) juga dilandasi oleh soket tetapi menyediakan beberapa kelebihan seperti skalabilitas integrasi aplikasi. Skalabilitas ini dihasilkan dari sifat *messaging* yang *asynchronous*, yang memungkinkan aplikasi pengirim tetap melanjutkan kerjanya tanpa

menunggu respon dari aplikasi penerima. *Method* pertukaran pesan antar aplikasi menggunakan *queues* untuk pengiriman dan penerimaan pesan. *Method* pertukaran pesan yang tidak langsung ini menyediakan ikatan longgar antara aplikasi pengirim dan penerima. Kelebihan lain adalah bahwa pengiriman pesan dapat dijamin dengan penyimpanan di dua sisi jaringan.



Gambar 1.5 *Message Oriented Middleware* (Juric et al, 2007)

Sinkronisasi pesan dapat disimulasikan menggunakan ID korelasi untuk membandingkan *request* pesan dengan *response* pesan. Pengembangan yang terkait erat adalah pengembangan *router* atau perantara pesan yang dapat merutekan pesan berbasis isi atau konteks.

Kemudian hadir WS, yang memperkenalkan standar untuk mengurangi heterogenitas yang disebakan oleh penggunaan beberapa teknologi (seperti

RPC, ORB, dan *messaging*). Selain itu juga diperkenalkan standard format data independen yang disebut *Extensible Markup Language*, atau XML. Kemudian definisi antarmuka layanan sebelumnya diperbaiki dengan pengenalan WSDL, yang memungkinkan antarmuka layanan dideklarasikan dalam bentuk bahasa pemrograman, platform dan *middleware* yang independen. Sama seperti WSDL, ide sebelumnya tentang pendaftaran layanan juga disempurnakan dengan pengenalan antarmuka *Universal Description, Discovery, and Integration* (UDDI). Akhirnya, format standard untuk pertukaran pesan telah dikenalkan dalam bentuk SOAP.

Dalam banyak kasus, WS sendiri belum cukup untuk mengurusi segala masalah heterogenitas. Secara khusus, WS belum dapat menangani adanya ketidaksesuaian protokol komunikasi antara penyedia layanan dengan pemakai layanan. Selain itu, WS juga belum dapat menyediakan solusi memuaskan dalam mengatasi ketidaksesuaian format data/pesan antara penyedia layanan dan pemakai layanan. Kemudian muncul ESB yang dapat mengatasi masalah tersebut. ESB menyediakan beberapa fungsi yang mencakup transformasi protokol dan pesan, rmerutekan pesan didasarkan pada isi dan konteks, transparansi lokasi, kualitas layanan, pengayaan data, dan fungsi lain.

Selain WS yang menggunakan aplikasi baru dan ESB, SOA harus menyediakan cara untuk mengintegrasikan aplikasi yang sudah ada untuk menawarkan solusi integrasi lengkap. Hal ini berarti memerlukan pembungkusan (*wrapping*) aplikasi yang sudah ada ke dalam WS atau menggunakan adapter yang memungkinkan aplikasi yang ada berkomunikasi satu sama lain. Mengenai perkembangan teknologi integrasi berbasis SOA ini dapat dilihat pada Gambar 1.6.

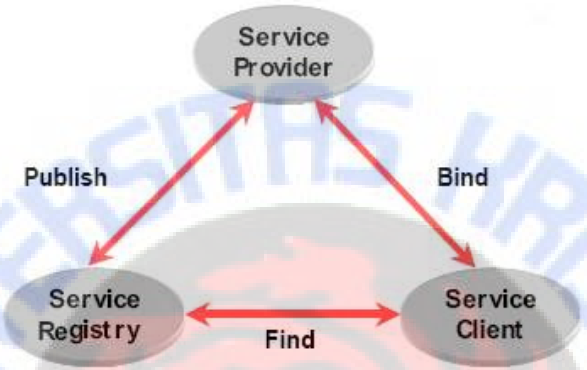
Gambar 1.6 Perkembangan teknologi integrasi berbasis SOA (Roshen, 2009)

1.2 Konsep SOA

Menurut Erl, SOA merupakan arsitektur sofware yang dibangun menggunakan prinsip-prinsip perancangan berorientasi service, sedangkan orientasi service merupakan konsep dalam rekayasa sofware yang merepresentasikan pendekatan berbeda untuk memisahkan kepentingan (Erl, 2005). Hal ini berarti bahwa fungsionalitas sistem dipecah ke dalam unit logik yang lebih kecil yang dinamakan service. Service-service ini lepas satu sama lain, tetapi mempunyai kemampuan untuk berinteraksi satu sama lain melalui mekanisme komunikasi tertentu. Karena itu, Erl (2005) mendefinisikan komponen SOA sebagai service, descriptions, dan messages. Service berkomunikasi dengan yang lain melalui message yang memungkinkan interaksi antar services, yang ditetapkan oleh description. Dua service berkomunikasi satu sama lain yang diacu sebagai service requestor dan service provider. Service requestor adalah service yang memanggil service lain, sedangkan yang dipanggil disebut service provider.

Service sendiri dapat dipandang sebagai enkapsulasi lojik dari satu atau sekumpulan aktivitas tertentu. Otomasi bisnis merupakan sekumpulan aktivitas yang disusun dalam langkah-langkah sebagai implementasi proses bisnis. Setelah seluruh permasalahan dapat dibagi dalam beberapa *service*, solusi dari permasalahan tersebut harus bisa diselesaikan dengan memungkinkan seluruh *service* berpartisipasi dalam sebuah orkestrasi. Untuk itu ada beberapa permasalahan yang harus dimiliki oleh *service*, yaitu bagaimana *service* berhubungan, bagaimana *service* berkomunikasi, bagaimana *service* didesain, dan bagaimana pesan antar *service* didefinisikan (Erl, 2005).

Selain definisi SOA oleh Erl, masih ada beberapa definisi SOA yang mirip dari sumber lain. Kadang kala definisi SOA ini dinamakan sebagai framework eb services, yang secara umum dapat dilihat pada gambar xxx.



Gambar 1.7 : Service-Oriented Architecture (Erl, 2005)

Service menyediakan service ke publik, berperan sebagai service provider yang menyediakan deskripsi service nya ke service registry. Service requestor, meminta pengiriman service dari service provider menggunakan service registry ini. Dua service ini diikat satu sama lain, untuk pertukaran data dan menggunakan fungsionalitas lain. Ini sesuai dengan definisi Erl yang diperluas dengan service registry. Pada service registry ini, service dapat didaftarkan dengan service description nya, sehingga service ini dapat ditemukan oleh service requestor. Dalam hal ini Erl memperluas definisinya tentang SOA yang masih sangat mendasar dengan menggunakan framework web services, yang merupakan definisi SOA yang konkrit yang didasarkan pada web services.

Framework web service ini merupakan framework teknologi yang didasarkan secara standard, yang dipetakan ke dalam model SOA primitif sebagai berikut :

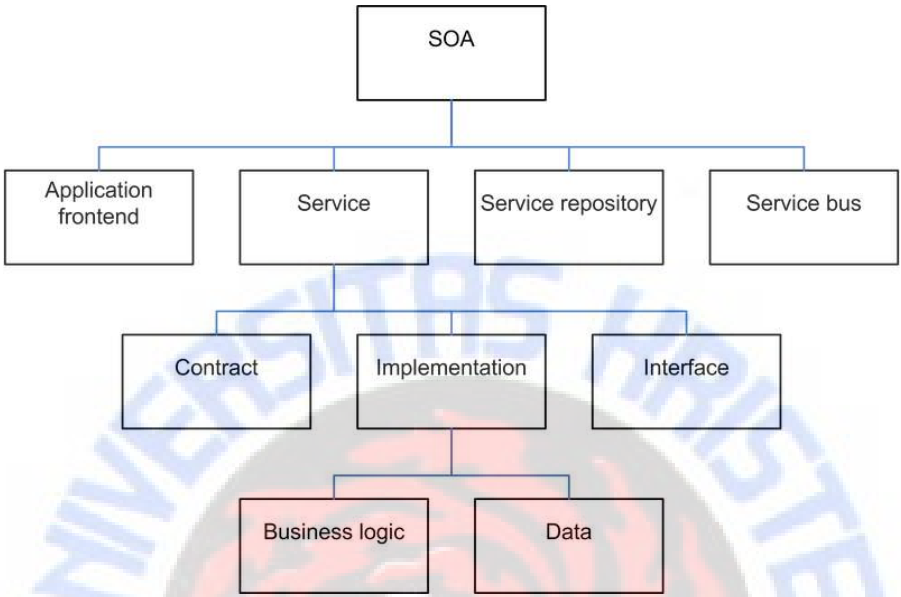
- Services direalisasikan sebagai web services
- Message dideskripsikan oleh protokol SOAP
- Deskripsi ditetapkan oleh WSDL
- Pada model primitif, service registry menggunakan UDDI

Hal ini pada dasarnya sesuai dengan korelasi ditunjukkan pada Gambar 2.1. Pengertian umum tentang SOA ini digunakan di banyak artikel. Namun definisi SOA ini hanya berurusan dengan aspek teknologi SOA. Hal ini sangat berkaitan erat dengan solusi berbasis web services dan kebutuhannya, tetapi konsep ini dapat diabstraksikan untuk membangun fondasi SOA secara umum.

Krafzig et al, mendefinisikan SOA dari sudut pandang enterprise, sehingga definisinya tidak se abstrak definisi Erl. Namun Krafzig menjelaskan bagian konkrit SOA yang digunakan untuk memecahkan masalah yang berkaitan dengan enterprise. Krafzig et al. bahkan memberikan definisi eksplisit dari SOA dan komponen masing-masing: "Service-Oriented Architecture (SOA) adalah sebuah arsitektur software yang didasarkan pada konsep-konsep kunci dari frontend aplikasi, service, service repository, dan service bus. Service terdiri dari contract, satu atau lebih interface, dan implementasi. Elemen-elemen SOA dan ketergantungan mereka seperti yang dijelaskan diatas dapat dilihat pada gambar 2.2. Setiap entitas pada diagram ini disajikan sebagai segiempat.

Aplication frontend merupakan pemilik proses bisnis dan dikoneksikan ke service melalui service bus. Service didefinisikan melalui interface yang memungkinkan akses ke service, contract yang menetapkan fungsionalitas dan implementasi service. Implementasi service menyediakan logika bisnis dan data yang relevan. Service menyediakan fungsionalitas bisnis yang dapat digunakan oleh application frontend atau service lain. Service repository digunakan untuk menyimpan service contract dari service sehingga dapat menggunakan service.

Jelas ada kesamaan antara definisi SOA menurut Erl dan Krafzig. Keduannya berisi service sebagai entitas dasar di dalam konsep mereka, dan service description yang mendeskripsikan service yang disediakan. Keduanya juga mengandung komponen komunikasi antar service, tetapi Erl menggunakan message sebagai konsep abstrak, disisi lain Krafzig memanfaatkan sebuah service bus. Dalam definisi Erl dalam SOA dasar, tidak ada elemen service registry, sebaliknya pada Krafzig, service registry merupakan komponen SOA. Tetapi dalam definisi web services berbasis SOA, menurut Erl, service registry merupakan bagian penting SOA. Selain itu, definis Krafzig berisi beberapa elemen tambahan yang tidak dapat ditemukan pada SOA dasar ataupun web service SOA pada Erl.



Gambar 1.8 : Komponen SOA

SOA adalah sebuah bentuk teknologi arsitektur yang mengikuti prinsip-prinsip *serviceorientation* (berorientasi *service*). Konsep *service-orientation* ini melakukan pendekatan dengan membagi masalah besar menjadi sekumpulan *service* kecil yang bertujuan untuk menyelesaikan permasalahan tertentu. Setelah seluruh permasalahan dapat dibagi dalam beberapa service, solusi dari permasalahan tersebut harus bisa diselesaikan dengan memungkinkan seluruh service berpartisipasi dalam sebuah orkestrasi. Untuk itu ada beberapa permasalahan yang harus dimiliki oleh service, yaitu bagaimana service berhubungan, bagaimana service berkomunikasi, bagaimana service didesain, dan bagaimana pesan antar service didefinisikan (Erl, 2005).

Pembagian berdasarkan *service* ini sesungguhnya bukan sesuatu yang baru, karena telah banyak diterapkan. Namun hal baru dari pendekatan *service-oriented* ini terkait dengan sifat- sifat yang dimilikinya (Erl, 2005), yaitu:

1. *Loosely coupled*, yaitu setiap *service* berdiri sendiri secara independen dan tidak tergantung *service* lain untuk berjalan. Ketergantungan diminimalisir sehingga hanya butuh mekanisme komunikasi satu sama lain.
2. *Service contract*, yaitu setiap *service* memiliki kesepakatan mengenai cara untuk komunikasi
3. *Autonomy*, yaitu *service* memiliki hak penuh terhadap semua lojik yang dienkapsulasi
4. *Abstraction*, yaitu *service* tidak memperlihatkan bagaimana lojik diimplementasi di dalamnya.
5. *Reusability*, yaitu lojik dibagi menjadi sekumpulan *service* yang dapat memudahkan *reuse*.
6. *Statelessness*, yaitu *service* tidak memiliki status tertentu terkait dengan aktivitas yang dilakukannya.
7. Discoverability, yaitu service didesain untuk deskriptif sehingga bisa ditemukan dan diakses melalui mekanisme pencarian tertentu.

SOA terdiri atas sekumpulan *service*. Namun sekumpulan *service* tidak cukup untuk membentuk sebuah arsitektur ini. Menurut (Erl, 2005), SOA terdiri atas empat komponen, yaitu:

1. *Message*, yaitu data yang dibutuhkan untuk menyelesaikan sebagian atau sebuah unit kerja, yang dipertukarkan antara satu *service* dengan yang lainnya
2. *Operation*, yaitu fungsi-fungsi yang dimiliki oleh sebuah *service* untuk memproses *message* hingga menghasilkan sesuatu. Fungsi-fungsi inilah yang nantinya akan saling berinteraksi untuk menyelesaikan sebuah unit kerja
3. *Service*, merepresentasikan sekumpulan *operation* yang berhubungan untuk menyelesaikan sekumpulan unit kerja yang berhubungan
4. *Process*, merupakan *business rule* yang menentukan operasi mana yang digunakan untuk mencapai tujuan tertentu.