EMBEDED MIDDLEWARE PADA CORBA

I. Joko Dewanto
Fasilkom – Universitas INDONUSA Esa Unggul, Jakarta
Jl. Arjuna Utara Tol Tomang, Jakarta 11510
i_joko_d_2000@yahoo.com

Abstract

The era computer information at now, so far, we've only dealt with the issues of a single embedded middleware. What if we what to get embedded middleware to run application that communicate across the program?, and distributed program.Sometime, programmer think about middleware have 3 model: CORBA, Java (EJB) or DCOM. CORBA it's Common Object Request Broker Architecture it's Middleware that enables an object that resides on client to send message to a method that is encapsulated by an object that resides on server. CORBA have language transparency, location transparency, Interoperability and no support distributed garbage collection. Standard CORBA have DII (Dynamic Invocation Interface), DSI(Dynamic Skeleton Interface) and Object adapter

Keyword: *Embedded Middleware, Distributed Program, CORBA, DII, DSI, Object Adapter*

Pendahuluan

Embeded dapat dijelaskan sebagai *"hidden inside so one can't see it"* disembunyikan di dalam tentunya tidak dapat dilihat". Di dalam *embedded system* ada 4 karakter, antara lain: 1. merupakan sistem tunggal, konsekuensinya adalah keterbatasan dalam mengontrol system. Sebagai contoh: *embedded system microwave* harus melakukan konfigurasi untuk system yang berbeda, 2. *embedded system* memiliki keterbatasan, sebagai contoh: MP3 memiliki keterbatasan baterai selama 12 jam, 3. embedded system haruslah *real time*, kita dapat meminimumkan waktu didalam pengontrolan system, 4. *embedded system* membutuhkan sedikit memori, misalnya saja dalam menjalankan: video atau audio tentunya akan membutuhkan memori untuk dijalankan, ditahun 2005 aplikasi *embeded system* rata-rata membutuhkan memori 32 kbytes.

Middleware adalah perangkat lunak komputer yang dikoneksikan pada beberapa komponen perangkat lunak atau beberapa aplikasi. Pada bagian ini bisa terdiri dari server web, aplikasi server, cms (*content management system*), dan peralatan-peralatan yang sama di dalam mendukung aplikasi yang dikembangkan dan digunakan. *Middleware* adalah kerjasama special pada informasi yang dasarnya adalah teknologi pada XML, SOAP, *Web Service* dan arsitektur yang berorientasi pada *service.*

Definisi ***Middleware*** adalah teknologi yang memungkinkan terintegrasinya aplikasi secara *enterprise* atau *enterprise application integration*- (EAI). Ini tentunya dapat dijelaskan sebagai bagian perangkat lunak yang dikoneksikan dua atau lebih aplikasi-aplikasi perangkat lunak yang mana dapat melakukan pertukaran data. Pada **Objek Web middleware** didefinisikan sebagai " Layer perangkat lunak yang antara sistem operasi dan aplikasi-aplikasi di dalam satu bagian dengan menggunakan sistem komputer terdistribusi di dalam sebuah jaringan.

***Middleware* pada Simulasi Teknologi,** Pada simulasi, *"middleware"* adalah sebuah bagian yang umum menggunakan konteks *Hig Level Architecture* (HLA) yang diaplikasikan ke banyak simulasi-simulasi distribusi. Pada layer ini perangkat lunak bekerja antara kode aplikasi dan *Run Time Architecture* (RTI), dimana *"middle"* ini adalah gelar. Middleware secara umum terdiri atas fungsi-fungsi *library,* dan memungkinkan sebuah angka pada aplikasi-aplikasi (simulasi-simulasi, atau perkumpulan-perkumpulan di dalam terminologi HLA) untuk halaman fungsi-fungsi ini umumnya *library* lebih dari sekedar mengulang pembuatan di dalam bagian aplikasi.

Model-model distribusi komputer:

- *Sockets* – *Interface low-level* pada sistem operasi untuk TCP dan UDP
- *Streams* – Proses *Interface low-level* antara kernel (biasanya peralatan – *level driver*)
- *Shared Memory – Interface Low-level* untuk *clients* dan *servers* pada mesin yang sama untuk komunikasi menggunakan sebuah memori yang sedang *share.*
- *Shared files – Client* dan *servers* pertukaran informasi melalui sistem file yang di *share.*
- RPC – akan dijelaskan kemudian.

Fungsi *Middleware*

Di dalam seluruh situasi-situasi, aplikasi-aplikasi digunakan sebagai perangkat lunak tingkat tinggi yang menentukan bagian atas system operasi dan komunikasi protokol-protokol yang memiliki beberapa fungsi :

- Menyembunyikan distribusi, contoh: faktanya bahwa sebuah aplikasi biasanya membuat banyak interkoneksi bagian di dalam lokasi terdistribusi
- Menyembunyikan heterogenitas adalah dimungkinkannya untuk komponen-komponen perangkat keras, system operasi dan komunikasi protocol-protokol.
- Menentukan kelompok (uniform), standard, interfaces tingkat tinggi *(high level interactives (HLA))* yang digunakan pengembang-pengembang aplikasi dan orang yang menghubungkan, yang aplikasi-aplikasinya dapat digabungkan secara mudah, digunakan kembali (reused), dan membuat bisa beroperasi secara internal *(interoperate)*
- Suplai adalah setup pada pelayanan umum yang memungkinkan memiliki fungsi tujuan secara umum, yang di jalankan membutuhkan dukungan duplikasi dan menfasilitasi kolaborasi antara aplikasi.

Ini adalah tingkat tinggi layers perangkat lunak yang disebut *middleware.* Ketentuan *middleware* adalah membuat mudah pengembang aplikasi, didalam menentukan abstraksi program yang umum, dengan penggunaan heterogenitas dan distribusi pada perangkat keras dan sistem operasi dan memungkinkan menyembunyikan detail *low level* program.

Tipe-tipe *Middleware*

Hurwitz mengklasifikasikan sistem organisasi dari beberapa tipe middleware yang terdiri atas:

1. ***Remote Procedure Call* (RPC)** – Membuat pemanggilan klien di dalam prosedure-prosedure yang dijalankan pada sistem remote. Dapat sebagai *asynchronous* atau *synchronous*
2. ***Message Oriented Middleware* (MOM)**- Pesan yang dikirim melalui klien yang dikoleksikan dan disimpan sampai dengan proses dijalankan, dimana klien melanjutkan dengan proses yang lain.
3. ***Object Request Broker* (ORB)** – Tipe ini pada *middleware* memungkinkan untuk mengirim objek-objek dan permintaan-permintaan *service* di dalam sistem berbasis objek.
4. ***SQL – oriented Data Access*** – *middleware* antara aplikasi dan *server database*

Sumber-sumber lain juga menambahkan klasifikasi-klasifikasi :

- ***Transaction Processing* (TP) monitors** – menentukan peralatan-peralatan dan lingkungan untuk membangun dan membangun kembali aplikasi terdistribusi
- ***Application Servers*** – perangkat lunak yang diinstal pada sebuah komputer untuk memfasilitasi pelayanan (running) pada aplikasi-aplikasi lain.
- ***Enterprise Service Bus*** – Sebuah abstraksi layer pada tingkat tinggi di dalam *Enterprise Messaging System.*

RPC adalah sebuah teknologi yang terdiri atas sebuah program komputer yang memiliki sebuah subrutin atau prosedur yang dapat dieksekusi di dalam address space lain (umumnya komputer didalam sebuah *share network)* tanpa programmer mengeksplisit detail *coding* untuk melakukan interaksi *remote.* Bahwa ini, adalah programmer akan menuliskan essensi kode yang sama di dalam subrutin local untuk mengeksekusi program, atau remote. Ketika perangkat lunak di dalam menjawab dengan menuliskan penggunaan prinsip-prinsip *object oriented*, RPC bisa juga disebut sebagai ***remote invocation*** atau ***remote method invocation***.

The five-layer TCP/IP model
5. Application layer
DHCP • DNS • FTP • Gopher • HTTP • IMAP4 • IRC • NNTP • XMPP • POP3 • SIP • SMTP • SNMP • SSH • TELNET • **RPC** • RTP • RTCP • RTSP • TLS/SSL • SDP • SOAP • BGP • PPTP • L2TP • GTP • STUN • NTP • …
4. Transport layer
TCP • UDP • DCCP • SCTP • RSVP • …
3. Internet Layer
IP (IPv4 • IPv6) • IGMP • ICMP • OSPF • ISIS • IPsec • ARP • RARP • RIP • …
2. Data link layer
802.11 • ATM • DTM • Token Ring • Ethernet • FDDI • Frame Relay • GPRS • EVDO • HSPA • HDLC • PPP • …
1. Physical layer
Ethernet physical layer • ISDN • Modems • PLC • SONET/SDH • G.709 • Optical Fiber • WiFi • WiMAX • Coaxial Cable • Twisted Pair • …

Jenis-jenis RPC adalah :

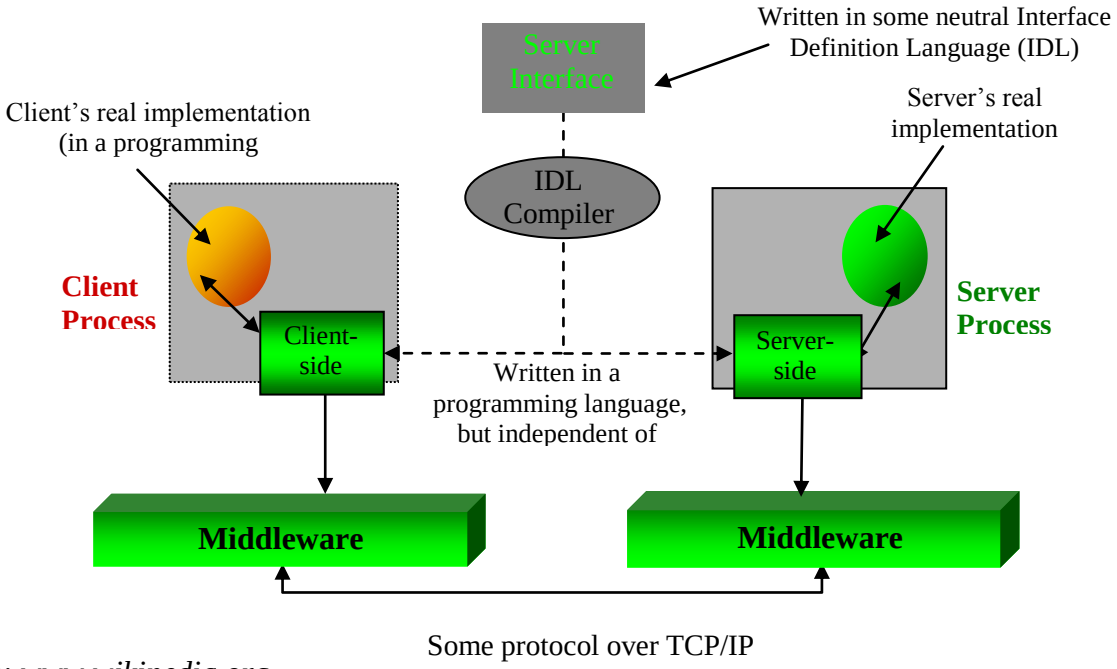
- Java's **Java Remote Method Invocation (Java RMI)** API yang menentukan fungsi persamaan untuk metode standar pada RPC system Unix.
  - **XML-RPC** ini adalah protokol RPC dengan menggunakan XML yang dikode dengan Calls dan HTTP sebagai sebuah mekanisme transport.
  - ***Microsoft.NET*** *Remote* adalah memfasilitasi RPC untuk sistem terdistribusi yang dimplementasikan pada platform Windows.

Middleware CORBA

Message-oriented middleware menentukan sebuah kategori pada inter perangkat lunak aplikasi komunikasi yang secara umum dijelaskan pada *asynchronous message – passing* yang dioposisikan untuk sebuah permintaan/respon. *Most message-oriented middleware* (MOM) lebih dalam lagi menjelaskan sebuah sistem *message queue system*, selanjutnya diimplementasikan jika terjadi *broadcast* atau pada *multicast messaging systems.*

Object Request Broker **(ORB)** di dalam komputer terdistribusi ini adalah bagian pada perangkat lunak middleware yang menentukan *programmer-programmer* untuk membuat panggilan-panggilan dari satu komputer ke lainnya melalui via network. Ada berbagai model, yaitu: EJB, DCOM dan ***Common Object Request Broker Architecture*** **(CORBA)**. Pada tulisan ini hanya membahas ORB. ORB menghandel transformasi dalam proses struktur data dan pengurutan byte, dimana dapat ditransmisikan melalui jaringan. Bagian ini disebut ***marshalling*** atau ***serialization***. Beberapa CORBA sistem dikompilasikan, dengan menggunakan ***Interface Description Language*** **(IDL)** untuk menjelaskan data dimana dapat ditransmisikan pada *remote calls.*

Di dalam penambahan data *marshalling*, ORBs sering diekspos isikan lebih banyak fitur-fitur, bisa sebagai *distribution transaction*, *directory services* atau *real time schedulling*. Di dalam bahasa orientasi objek, ORB menentukan form pada objek yang memungkinkan koneksi objek dapat dilayani. Setelah sebuah objek dikoneksikan pada ORB, maka metode-metode objek tersebut dijalankan berdasarkan remote di dalam lapangan pekerjaan, ORB meminta beberapa alamat jaringan sebuah objek yang diremote. Tipe ORB juga menentukan banyak metode-metode lainnya.



Source: www.wikipedia.org

SQL – *oriented Data Access* – di dalam *middleware* hubungan aplikasi dan server database diwakilkan pada SQL data akses, ada beberapa tipe antara lain: http://en.wikipedia.org/wiki/Category:SQL_data_access. There are 12 pages in this section of this category.

A
- *ADO vs ADO.NET*
- *ADO.NET*

D
- *DataReader*
- *DataSource*

D cont.
- *Java Database Connectivity*
- *Dbclient*

I
- *IODBC*

O
- *OLE DB*
- *Open*

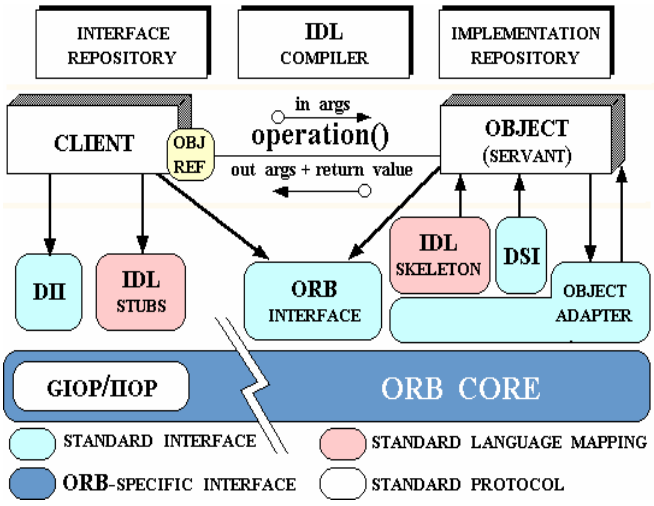
S
- *SQLJ*
- *Structured Query Language Interface*

U
- *UnixODBC*

Middleware CORBA memiliki beberapa alasan antara lain, adalah :

1. ***Language transparency***. Klien dan servers dapat menulis di bahasa pemrogramman yang berbeda
2. ***Location transparency***. Klien dan server tidak usah ragu kira-kira dapat dijalankan pada lokasi yang berbeda.
3. ***Interoperability***. Klien dan server dapat dijalankan pada hardware yang berbeda, sistem operasi yang berbeda dan berlangsung ketika dijalankan dari beberapa protokol atau dapat disebut IIOP. Atau disebut juga ***GIOP*** (*General Inter-ORB Protocol)* adalah abstraksi protokol yang dipilih melalui *Object Reguest Group (OMG).*
4. Tidak mendukung untuk *distributed garbage collection*

Selanjutnya kita dapat melihat standar CORBA dalam menjalankan model komputer objek terdistribusi.



Source: www.wikipedia.org

IDL (Interface Description Language) adalah bahasa komputer yang digunakan untuk menjelaskan sebuah hubungan komponen perangkat lunak bahwa tidak untuk share sebuah bahasa, misalnya antara komponen-komponen yang dituliskan dalam C++ dam komponen-komponen yang dituliskan dalam Java.

IDL biasanya digunakan untuk perangkat lunak pemanggilan *prosedure remote.* Pada kasus ini biasanya mesin-mesin yang di*link* dengan menggunakan sistem operasi yang berdan dan bahasa komputer yang berbeda. IDL adalah sebuah *bridge* yang menghubungkan dua sistem yang berbeda.

DII *(Dynamic Invocation Interface)* adalah sebuah API yang memiliki konstruksi dinamis pada penyampai objek CORBA. Ini biasanya digunakan pada waktu kompilasi dimana sebuah klien tidak membutuhkan pengetahuan mengenai objek yang ingin digunakan. Dengan *interface* ini daftar argumentasi di *serialization,* sebuah fungsi nama, dan sebuah permintaan pelayanan pengiriman objek ke server. DII biasanya akan mengeluarkan mode *asynchronous* pada pengoprasian.
Ada tiga model DII :

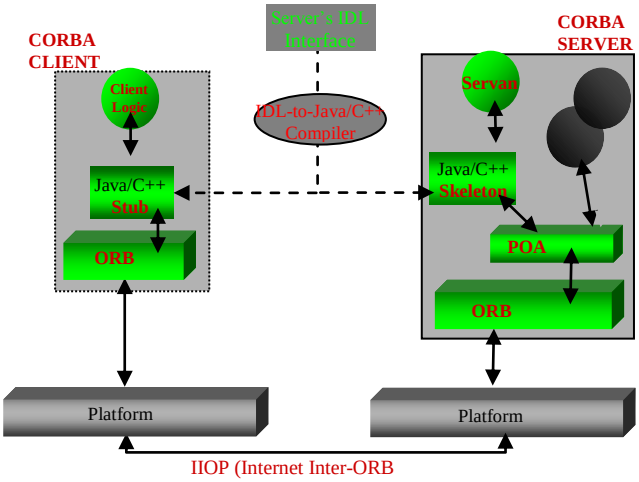
- ***Twoway syncronous*** – pemilihan tipe permintaan/respon model dihubungkan dengan RPC toolkits, dimana client memblok setelah pengiriman dan selanjutnya melakukan permintaan respon yang datang dari server
- ***Oneway syncronous*** – dimana sebuah "hanya permintaan" model pesan dimana klien tidak melakukan respon
- ***Deferred syncronous*** – model panggilan dua arah dimana permintaan dari respon dimana klien dapat dijawab setelah server mengirim respon.

IDL stubs (pilihan hanya menggunakan RPC – model permintaan-permintaan), DII biasanya juga melakukan klien untuk dibuat tidak memblok *Deffered synchronous* pada pengoperasian-pengoperasian (pengiriman dan penerimaan).

DSI (Dynamic Skeleton Interface): ini adalah merupakan server side's analog di dalam client side's DII. Pada DSI mengikuti ORB yang diminta atas permintaan implementasi objek bahwa tidak memiliki pengetahuan waktu compile dalam mengimplementasikan objek. Client membuat permintaan berdasarkan bukan dari ide implementasi dari spesifikasi tipe IDL Skeleton atau menggunakan Dynamic Skeletons.

Object Adapter: menghubungkan implementasi objek dengan ORB dam terdiri atas ORB yang mengirim permintaan-permintaan pada objek dan dengan aktifasi objek. Yang perlu digarisbawahi, dengan menggunakan CORBA secara relative memungkinkan dukungan yang mudah pada jaringan dengan kecepatan tinggi. Secara umum, CORBA diimplementasikan pada DII dan DSI.

Pemikiran kritis CORBA



Source: www.wikipedia.org

Interface IDL

Interface IDL didefinisikan sebagai *interface* atau perilaku (tetapi bukan atribut-atribut atau state) pada sebuah *server object,* sedangkan modul: paket yang direlasikan dalam bentuk hubungan-hubungan dan tipe-tipe. IDL bukanlah bahasa pemrograman dengan menggunakan konstruksi seperti *if, else, then, while, do, etc. IDL Compiler* dengan menentukan setiap ORB, input = server interface di dalam IDL, output= skeleton dan stubs di dalam bahasa pemrograman (Java, C++, atau yang lainnya). Beberapa tipe IDL: tipe dasar: *octet, char, short, long, double,* struktur *tipe : string, sequence <someOherType>, union, struct.*

Tipe

- Apa saja tipe IDL ?
  - Tipe-tipe dasar: *octet, char, short, long, double*
  - Tipe-tipe struktur: *string, sequence <someOtherType>, union, struct*
  - Tipe-tipe arbitary : *any*
  - *User-defined types: e.g., interfaces*

Contoh IDL Interface

```
module Bank
{
 exception ZeroBalance {};
 typedef sequence<double> DepositHistory;
     interface Account
{
string getBalance(in string accountNumber);
void depositMoney(in double amount,
ut double balance,inout DepositHistory latest);
double withdrawMoney (in double amount)
raises (ZeroBalance);
      };
};
```

Enterprise Application Integration (EAI)

Didefinisikan sebagai perangkat lunak dan prinsip-prinsip arsitektur sistem komputer yang diintegrasikan dengan sebuah set pada aplikasi sistem enterprise.

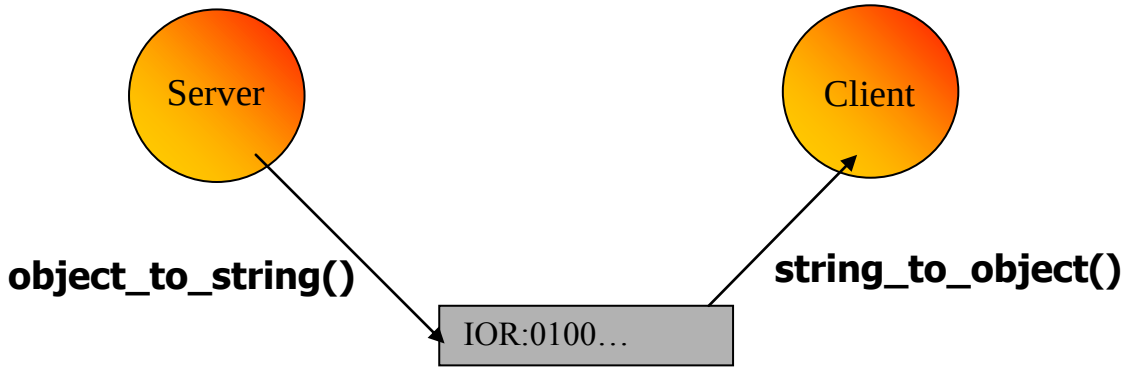
POA

- Hanya berjalan pada CORBA server
- Pengiriman yang datang atas permintaan merupakan target utama yang dilayani
- Biasanya aktifitas berdasarkan sebuah spasi nama yang dikoleksi berdasarkan pengaturan pelayanan
- Penambahan kebijakan yang memungkinkan anda dapat melakukan kostumisasi sebuah POA, misalnya saja :
  - *Multithreading*
  - *Activation on demand*
  - *Persistent references (lifespan of objects)*
  - *Unique object identifiers*
- Sebuah proses server dapat memiliki banyak POA's, yang dikonfigurasikan secara berbeda, dan memiliki tanggung jawab dalam sebuah kemungkinan set pada pelayanan
- General Inter-ORB Protocol (GIOP)
  - Spesifikasi sebuah protocol, dan bukanlah implementasi
  - Delapan pesan format yang disampaikan antara lain: *Request, Reply, Locate Request, LocateReply, CancelRequest, MessageError, CloseConnection, Fragment*
  - Sebaiknya didefinisikan *headers* untuk setiap pesan
  - Membawa pengiriman dalam bentuk *order bit*
  - Dapat dilakukan pemetaan antara protocol sebenarnya dengan orientasi konektifitas.
- *Internet Inter-ORB Protocol* (IIOP)
  - Kongkrit implementasi pada GIOP yang dispesifikasikan melalui TCP/IP
  - Harus dapat mendukung setiap ORB
  - Biasanya objek dikomunikasi tanpa mengenai OS, arsitektur *hardware* dan lainnya
- *Impelementation Repository*
  - Lokasi penyimpanan pada sebuah *executables* di dalam sistem *file*
  - *Register Run-Time* pada server dan klien-klien
  - Sangat spesifikasi terhadap vendor
- *Interface Repository*
  - Menyimpan tipe informasi pada seluruh objek CORBA di dalam aplikasi
  - *Interface Dynamic invocation* bahwa sebenarnya dapat diimplikasikan untuk tidak mengetahui tipe-tipe yang ditampilkan dalam waktu, serta kamu dapat meningkatkan pada konsumsi waktu yang dijalankan
- *Naming Service*
  - Mendaftarkan secara nama yang user-friendly untuk diasosiasikan dengan referensi objek
  - Pelayanan dapat dilapisi dengan sebuah nama
  - Klien dapat melihat referensi berdasarkan nama

Interoperable Object References (IORs)

- Format Stringfield sebagai sebuah referensi
  - Dapat disimpan di dalam sebuah file, dikirim melalui jaringan, dan lainnya
  - Server-server terdiri atas hostname, port number, object key, etc.
- Contoh IOR dapat dilihat pada:

IOR:010000001000000049444c3a4163636f756e743a312e3000020000000000000030000000010100001300000752d6d61792e7468696e6b6f6e652e636f6d00007b0900000c000000424f410a20b0530000055030100000024000000010000000100000001000000140000000100000001000100000000000901010000000000

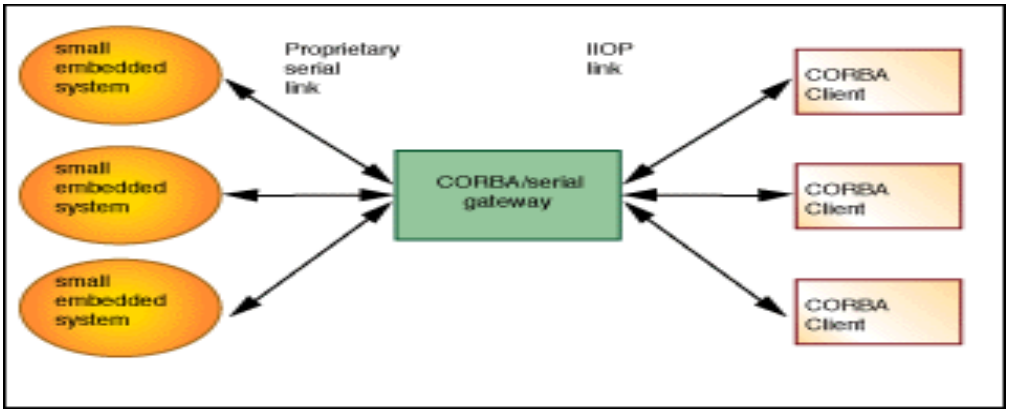


Source: www.wikipedia.org

Middleware di dalam *Embedded Systems*

- *Requirements*
  - Sumber-sumber terbatas: *Smaller foot-print (requires less code-bloat)*
  - Dukungan *Real-time* (memungkinkan di dalam *Real-Time* spesiifikasi CORBA)
- Implementasi Full CORBA: 150 Kbytes – 5 Mbytes!
- Apa saja yang dimungkinkan ?
  - Minimal CORBA: *Throw out* biasanya pada mekanisme heavyweight CORBA (~ 30-60 Kbytes)
  - Engine IIOP Engine: digunakan sebuah library untuk mempersiapkan hanya pada bagian protocol IIOP (~ 15 Kbytes)
  - CORBA *Gateways:* TCP bridge melalui *prosesor embedded* pada sebuah CORBA gateway bahwa dibicarakan sebagai istira-hat pada dunia

Menggunakan CORBA gateway



Source: www.wikipedia.org

Kesimpulan

1. Konsep *Middleware* sangat dibutuhkan oleh para pengembangan komputer tentunya untuk kebutuhan pengembagan inter pe-rangkat lunak, sehinga juga memungkinkan melakukan interkoneksi bagian dilokasi terdistribusi baik dari segi perangkat kerat, sistem operasi dan komunikasi protokol-protokol.
2. Konsep *middleware* juga mampu mela-kukan pengemlompokan, standar, interface yang memungkinkan menggabungkan apli-kasi secara mudah, serta mensuplai setup fungsi yang memberikan dukungan dupli-

kasi dan manfasilitasi kolaborasi antar aplikasi.

3. CORBA adalah merupakan object contoh yang digunakan dalam pengembangan *middleware*

Saran

1. Memungkinkan menggunakan object lain untuk dikembangkan, seperti: Java dan DCOM
2. Sebaiknya sebelum mengembangkan *embedded middleware* anda perlu membandingkan beberapa object middleware lain.

Daftar Pustaka

Jonathan W. Valvano, *"Embedded Microcomputer Systems Real Time Interfacing"*, second edition, Thomson, US, 2007.

Peter Marwedel, "Embedded System Design".

Roger S. Pressman, *"Software Engineering A Practitioner's Approach"*, fifth edition, Mc GrawHill, Singapore, 2001.

Victor, Felix dan Herwig, *"Advanced Distributed System"*, Springer, Argentina, 2004.

www.primstech.com

www.wikipedia.org