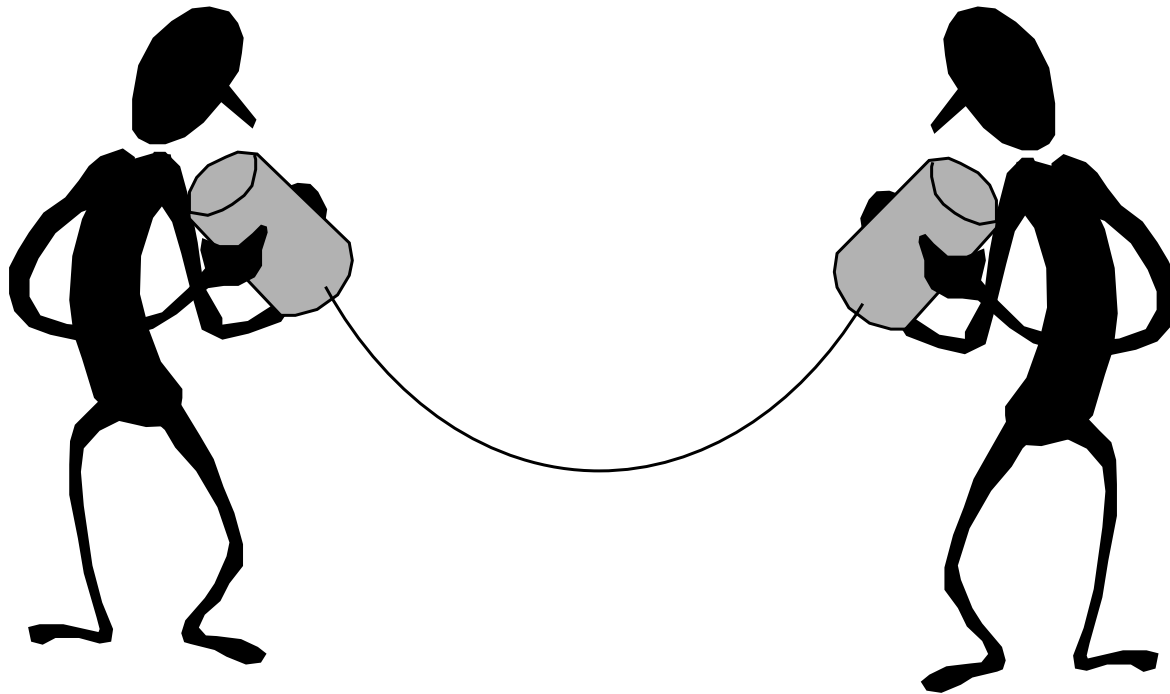


Interoperability / Distribution



Interoperability - Introduction

- ◆ This section introduces some concepts and issues about interoperability, commonly used client server architectures, etc...
- ◆ Although not directly related to C, C++ or Java most of the topics in this section have to be taken into account when designing and implementing interoperable systems.

Interoperability - Definitions

- ◆ **Interoperability** is the capability that a software system has to interact (work together) with other software systems.
- ◆ A piece of code requesting a service (I.e. calling a function) is called **client**.
- ◆ The piece of code executing the service (I.e. the function) is called **server**.
- ◆ A **local service** is a service belonging to the same address space (I.e. process) of the client.
- ◆ A **remote service** is a service not belonging to the same address space of the client (it can be on another process, on another machine, on internet...)

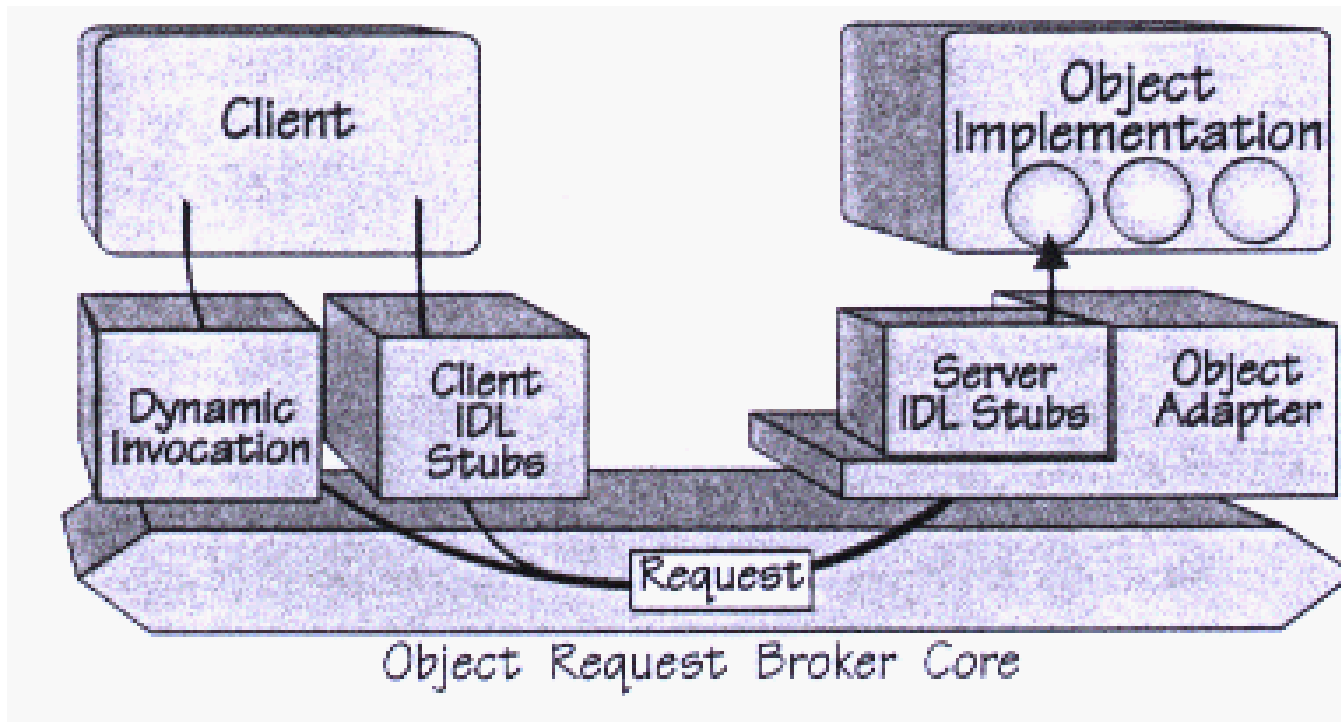
Interoperability - Issues

- ◆ Interfaces between server and client are often not very well defined.
- ◆ The client and server may run on different hardware and software platform (different machine, operating system, language, language implementation) => different data representation, language virtual machine, computational model etc...
- ◆ The server is not guaranteed to be running (available, etc...) at the time when a service is requested.

Interoperability - Distributed Architectures

- ◆ The most common Distributed Architectures are:
 - **CORBA** (Common Object Request Broker Architecture) from the Object Management Group;
 - **OLE/DCOM** (Object Linking and Embedding / Distributed Component Object Model) from Microsoft;
 - **Java Beans** (the Java based components) from Sun;
 - **Jini / UPnP / Salutation**
 - The **Web** (XML-Protocols/SOAP?).

Interoperability - CORBA



Interoperability - CORBA

(cont)

- ◆ Interfaces are specified in **IDL** (see examples CPPCorba, JavaCorba).
- ◆ The **client IDL stubs** provide static interface to object services.
- ◆ The **Dynamic Invocation Interface** (DII) allows to discover and use the methods to be invoked at run time.
- ◆ The **Interface Repository APIs** allow to obtain and modify the descriptions of all registered component interfaces, the methods they support and the parameters they require.

[Mars]

Interoperability - CORBA

(cont)

- ◆ The **server IDL stubs** (skeletons for OMG) provide static interfaces to each service exported by server.
- ◆ The **Dynamic Skeleton Interface (DSI)** provides a run-time binding mechanism for servers that need to handle incoming method calls that do not have IDL-based compiled skeletons.
- ◆ The **Object Adapter** sits on top of ORB's core communication services and accepts requests on behalf of the server objects.
- ◆ The **Implementation Repository** provides a run-time repository of information about the classes a server supports, the objects that are instantiated and their IDs.

[Mars]

Interoperability - CORBA - Services (cont)

- ◆ An ORB by itself doesn't have all it takes for objects to interoperate at the system level, it only provide the basic mechanism for brokering object requests.
- ◆ All other services are provided by objects with IDL interfaces that reside on top of the ORB.
- ◆ The IDL and ORB provide the function of a “software bus”; the CORBA object services plug in into the bus and augment it. The end-user object components make use of both the bus and its services.

[Mars]

Interoperability - CORBA - Services (cont)

◆ Object Services are:

- life cycle,
- persistence,
- naming (names → references),
- event,
- concurrency control,
- transaction,
- externalization,
- query service,
- licensing,
- properties.

Interoperability - CORBA - Facilities (cont)

- ◆ Common Facilities are collections of IDL-defined components that provide services of direct use to application objects.
- ◆ Facilities are divided into:
 - horizontal:
 - » User Interface
 - » Information Management (OpenDoc, etc.)
 - » System Management (Conf, Inst, etc.)
 - » Task Management (WF, Trans., Scripting, etc.)
 - vertical:
 - » domain application dependent (health, retail, telcos, etc.)

Interoperability – CORBA – BOA vrs. POA

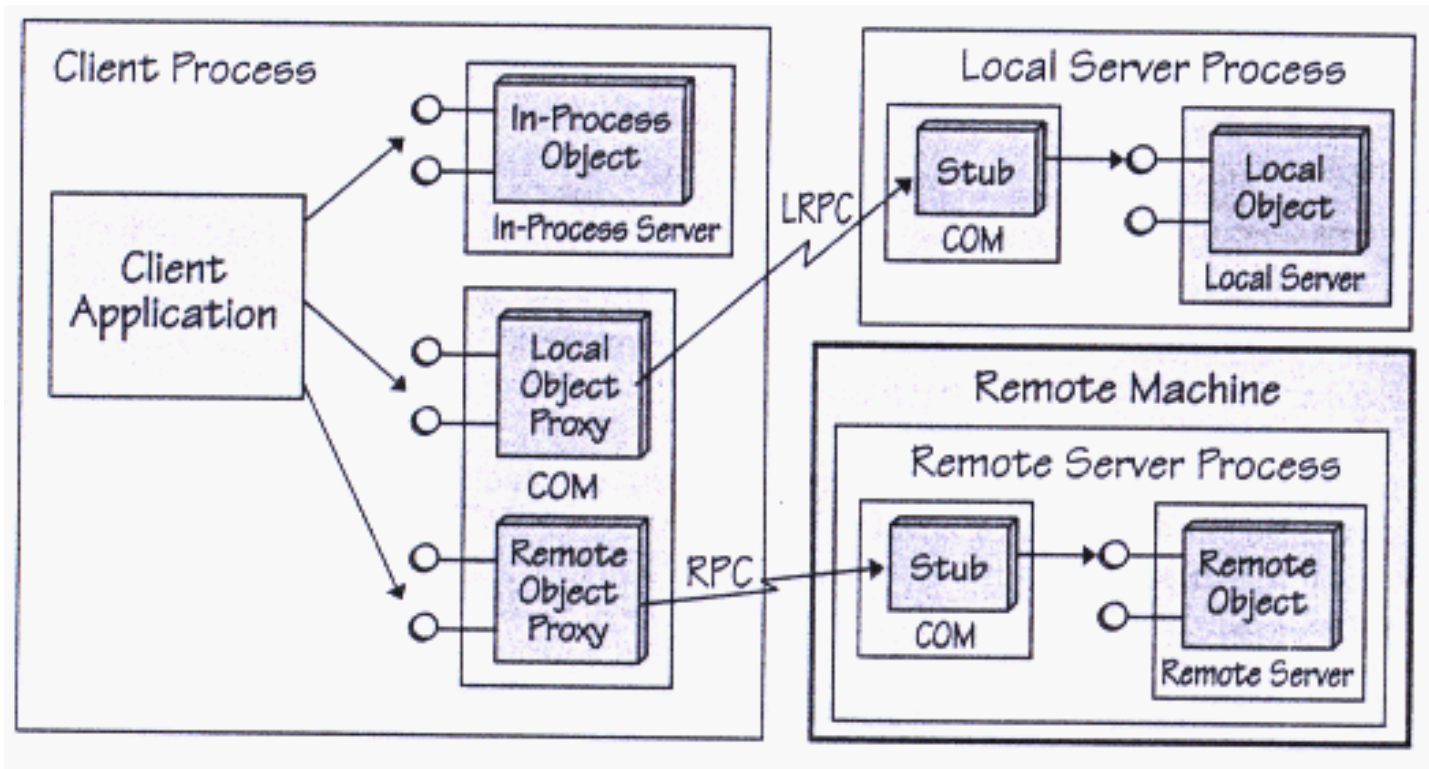
- ◆ With CORBA 3.0 the **POA** (Portable Object Adapter) superceeds the BOA (Basic Object Adapter) as the primary way of making implementations objects available to the ORB for servicing requests. POA was added because of the following reasons:
 - BOA was not portable across different ORBs. **POA**, being completely specified in IDL, **is fully portable**.
 - **POA provides support implementing large scale systems.** Servers can better handle connections and requests from multiple clients through **automatic, policy driven behaviour**.
 - Opposite to BOA, **POA is completely location transparent.** With POA a call to a local object (in the same process) will still go through the POA. POA acts like an EJB “container”.

Interoperability – CORBA – BOA vrs. POA

(c)

- ◆ The **POA distinguishes** between the CORBA object reference (**IOR**) and the implementation object that does the work. This implementation object is called a **servant**. A BOA-based approach has the IOR and servant existing at the same time. A POA-based approach can support this, but can also support IORs existing without being associated with servants, and also servants existing without being associated with IORs.
- ◆ Obviously, the association between an IOR and a servant has to be made at some point, to make the servant a useable CORBA object. But **this association can be done on-demand**. Consider the following example scenarios to motivate the advantages of on-demand association:
 - A pool of servants can be instantiated, and then associated in turn with IORs, as needed.
 - A set of IORs can be created for the purposes of publishing the references to the Name Service, without going through the work to actually instantiate the servants.
- ◆ Moreover, the POA allows a single servant to simultaneously support several IORs.
- ◆ All of the above significantly contribute to scalable applications.

Interoperability - OLE/DCOM



Interoperability - OLE/DCOM

(cont)

- ◆ Interfaces are specified in **IDL** (implementation repository) and **ODL** (type/interface repository) [see ADCOMServer, ADCOMClient and MSJDCOM examples].
- ◆ An OLE **component** can support one or more **interfaces**; an interface defines a contract between components.
- ◆ An OLE component is defined by a **class** that implements one or more interfaces and a **class factory** - this is the interface that knows how to produce a component instance of that class.
- ◆ A COM interface is a collection of function calls (no variables).
- ◆ OLE/DCOM IDL/ODL **don't support implementation inheritance**.

Interoperability - OLE/DCOM

(cont)

- ◆ OLE/DCOM IDL/ODL don't support exceptions.
- ◆ CORBA references point to object services instances, DCOM CLASSIDs to classes.
- ◆ Some CORBA/DCOM analogies:
 - Life Cycle \leftrightarrow Iunknown (QueryInterface, AddRef, Release)
 - Version Control/Licensing \leftrightarrow IFactory/IFactory2
 - Events \leftrightarrow Connectable Objects
 - OpenDoc OSA \leftrightarrow OLE Automation
 - Dynamic Skeleton Interface \leftrightarrow Dispinterface
 - Persistence \leftrightarrow Structured Storage and Monikers
 - OpenDoc Part \leftrightarrow OCX (ActiveX)

Interoperability - OLE/DCOM - OCX (cont)

- ◆ An OCX is a combination of an OLE in-process server and an OLE automation server.
- ◆ It is a in-process server that supports embedding, in-place editing, inside-out activation, OLE automation, event notification and connectable objets.
- ◆ **OCXs support licensing, property editing, and they define standard properties and events.**
- ◆ The closest thing to OCXs (I.e. ActiveXs) are JavaBeans (with CORBA 3.0 CORBABeans will appear on the scene).

Interoperability - JavaBeans

- ◆ Beans are **independent, reusable software modules**. Beans may be visible objects, like AWT components, or invisible objects, like queues and stacks. A builder/integration tool manipulates Beans to create applets and applications (see BangBean example).
- ◆ Beans consist of three things:
 - **Events**
 - **Properties**
 - **Methods**
- ◆ Also, since Beans rely on their state, they need to be able to be persistent over time.

Interoperability - JavaBeans - Events (cont)

- ◆ Events are for notifying others when something is happening. The delegation-event model of AWT, introduced with Java 1.1, demonstrates the Beans event model. It has three parts:
 - EventObjects
AWTEvent in the AWT world.
 - Event Listeners
ActionListener, ItemListener, ...
 - and Event Sources (Beans)
The different AWT Components.

Interoperability - JavaBeans - Events (cont)

- ◆ Anyone can register an `EventListener` with a `Component`, provided the `Component` understands the event set. (For instance, you cannot register an `ActionListener` with a `TextArea`, but you can with a `TextField`). When something happens within the `Component`, it notifies any listener(s) by sending each an `EventObject`, through the appropriate method of the listener.

Interoperability - JavaBeans - Properties (C)

- ◆ Properties define the characteristics of the Bean. For instance, when examining an AWT TextField for its properties, you will see properties for the caret position, current text, and the echo character, among others. In the simplest case, a method or methods in the following design pattern defines a property:

```
public void setPropertyName(PropertyType value);  
public PropertyType getPropertyName();
```

- ◆ Where *PropertyName* is the name of the property, and *PropertyType* is its datatype. If only one method is present, the property is read-only (set missing) or write-only (get missing).

Interoperability - JavaBeans - Methods (C)

- ◆ Bean methods are available for anyone to call by just making each public. However, you can restrict which methods are visible to the Bean builder/integration tool by providing a `getMethodDescriptors` method along with your Bean's `BeanInfo`.
- ◆ Every Bean can provide a supporting `BeanInfo` class to customize a Bean's appearance to an integration tool.

Interoperability - JavaBeans - Persistence

- ◆ Persistence is the ability of an object to store its state, for recreation later.
- ◆ Beans use Java's object serialization capabilities for persistence.
- ◆ Serialization saves all non-static and non-transient instance variables of an object.

Interoperability – Jini / UPnP / Salutation

- ◆ Coordination between devices has become a serious research issue.
- ◆ A number of architectures addressing mobile and specialized devices have emerged recently.
- ◆ These architectures are essentially coordination frameworks that propose certain ways and means of device interaction with the ultimate aim of simple, seamless and scaleable device interoperability.
- ◆ The most important architectures are:
 - **Jini** – <http://www.jini.org>
 - **UPnP** – <http://www.upnp.org>
 - **Salutation** – <http://www.salutation.org>

[from a paper published by California SW Labs, <http://www.cswl.com>]

Interoperability – Jini / UPnP / Salutation (c)

- ◆ Device coordination essentially means providing a subset of the following capabilities to a device:
 - Ability to **announce its presence** to the network.
 - **Automatic discovery** of devices in the neighbourhood and even those located remotely.
 - Ability to **describe its capabilities** as well as **query/understand the capabilities of other devices**.
 - **Self configuration** without administrative intervention
 - **Seamless inter-operability** with other devices wherever meaningful

Interoperability – Jini / UPnP / Salutation (c)

- ◆ It should be borne in mind that for any coordination framework to work, **it must introduce some standards** into the operations of the devices. Otherwise the devices simply cannot coordinate.
- ◆ The essential problem here is maintaining a **balance** between **standardization requirements** and **device autonomy**.
- ◆ By device autonomy we mean the use of proprietary protocols and techniques and the need (market-based or otherwise) to continue using them..

Interoperability – Jini / UPnP / Salutation (c)

◆ Jini

- **Jini** from Sun Microsystems, underneath all the hype, is but a coordination framework evolved and adapted from academic research and tailored specifically to Java.
- The original inspiration comes from the works of David Gelernter (Yale University) and Nick Carriero who built the **Linda** coordination model using Tuple Spaces, about a decade ago. An inspired result was the **JavaSpaces** technology with its later evolution into Jini.
- Jini uses the term **federation** to imply **coordination between devices**. A federation is a collection of autonomous devices which can become aware of one another and cooperate if need be.
- To facilitate this, a Jini **subsystem contains a set of lookup services** that maintain dynamic information about available devices. These services are key to the proper functioning of the Jini subsystem.

Interoperability – Jini / UPnP / Salutation (c)

◆ Jini Lookup Services

- Jini Lookup services must be running on a network for the concept to really work.
- **Every device must discover one or more such lookup services** before it can enter a federation.
- **It is possible to assign group names to lookup services** so that a device may look for a specific group in its vicinity.
- Once **a device** has located a lookup service of interest it **can now tell the service** about itself (register), or **query the service for information on other devices** in the federation.
- When registering itself, **it can associate a set of properties** (name/value pairs) **with this information** which can be matched against queries from other devices.

Interoperability – Jini / UPnP / Salutation (c)

◆ Exposing Interfaces

- During the registration process it is possible for a device to **upload some Java code to the lookup service.**
- **This code is essentially a "proxy"** that can be used to contact an interface on the device and invoke methods of that interface.
- So a querying device can automatically download this proxy and call methods inside our device. This is accomplished using **Remote Method Invocation (RMI) or a proprietary (null) protocol.**
- The Jini lookup services can form an ensemble, passing on queries up a hierarchy for resolution and trying to **make sure that they have an accurate snapshot of the current set of active devices.** This is done by providing **registration leases with expiry.**

Interoperability – Jini / UPnP / Salutation (c)

◆ UPnP

- UPnP, pushed primarily by Microsoft, is a framework defined at a much lower level than Jini.
- **UPnP is not an extension of PnP.**
- **UPnP works primarily with TCP/IP network protocols suite,** implementing standards at this level instead of at the application level.
- By providing a set of defined network protocols, **UPnP allows devices to build their own APIs that implement these protocols** - in whatever language or platform they choose.

Interoperability – Jini / UPnP / Salutation (c)

◆ Service Discovery

- UPnP uses a special protocol known as **SSDP** (Simple Service Discovery Protocol) that enables devices to announce their presence to the network as well as discover available devices. This can work with or without a lookup (directory) service in between.
- This discovery protocol (SSDP) will use HTTP over both unicast (**HTTTPU**) and multicast UDP (**HHTPMU**).
- The registration/query process will send and receive data in HTTP format, but having special semantics. An **announcement message called ANNOUNCE** and a **query message called OPTIONS** are embedded in HTML to facilitate this.

Interoperability – Jini / UPnP / Salutation (c)

- A device joining the network can then send out a **multicast ANNOUNCE**.
- The ANNOUNCE message must also contain a **URI that identifies the resource** (e.g. "dmtf:printer") and a **URL to an XML file that provides a description of the announcing device**.
- A query for device discovery can also be multicast (an OPTIONS message).
- **Queries** however seem not to be targeted at the XML descriptions, and **just use the URIs** like "/ietf/ipp/printer". The XML description is looked at only after discovery takes place.

Interoperability – Jini / UPnP / Salutation (c)

◆ Configuration

- UPnP also addresses the problem of automatic assignment of IP addresses and DNS names to a device being plugged in. For these, a few more protocols are introduced.
- One way to assign an IP address is to **look for a DHCP server** on the network.
- If not present, then the device can **choose an IP from a reserved IP address range known as the LINKLOCAL net** (address range 169.254.x.x ?). It must use the ARP protocol to find an unused address in this range locally and assign to itself. This is known as **AutoIP**.
- To address issues in naming, a **multicast DNS proposal** has been drafted.
- Once the discovery process is through and the XML description of a device is received, **proprietary protocols can take over** in communicating with the devices.

Interoperability – Jini / UPnP / Salutation (c)

◆ Salutation

- The Salutation coordination framework seems to be a **more rigorous and useful framework** from a coordination perspective than either Jini or UPnP.
- It seems to have drawn more from **research on intelligent agents** than other frameworks mentioned here.
- Salutation **trods a middle way between device autonomy and standardization**. This would allow many vendors to adapt many of their products more or less easily to the specification and inter-operate with one another.

Interoperability – Jini / UPnP / Salutation (c)

◆ Service Discovery

- In Salutation, a **device almost always talks directly to a Salutation Manager**, which may be in the same device or located remotely.
- What we see is an ensemble of **Salutation Managers (SLMs)** that coordinate with one another. They **act like agents that do everything on behalf of their clients**. Even data transfer between devices, including those across different media and transports, is mediated through them.
- The **framework provides call-backs into the devices to notify of events** like data arriving or devices becoming unavailable.
- **All registration is done with the local or nearest available SLM**. SLMs discover other nearby SLMs and exchange registration information, even with those on different transport media. The latter is done using appropriate transport-dependent modules called Transport Managers which may use broadcast internally. Broadcast RPC is optionally used for this.

Interoperability – Jini / UPnP / Salutation (c)

- The concept of a **'service'** is broken down into a collection of **Functional Units**, each unit representing some essential feature (e.g., Fax, Print, Scan or even sub-features like Rasterize). A **service description is then a collection of functional unit descriptions**, each having a **collection of attribute records** (name, value).
- **These records can be queried and matched against during the service discovery process.** Certain well defined comparison functions can be associated with a query that searches for a service.
- The **discovery request is sent to the local SLM which in turn will be directed to other SLMs.** SLMs talk to one another using Sun's ONC RPC (Remote Procedure Call). Salutation defines APIs for clients to invoke these operations and gather the results.

Interoperability – Jini / UPnP / Salutation (c)

◆ Communication

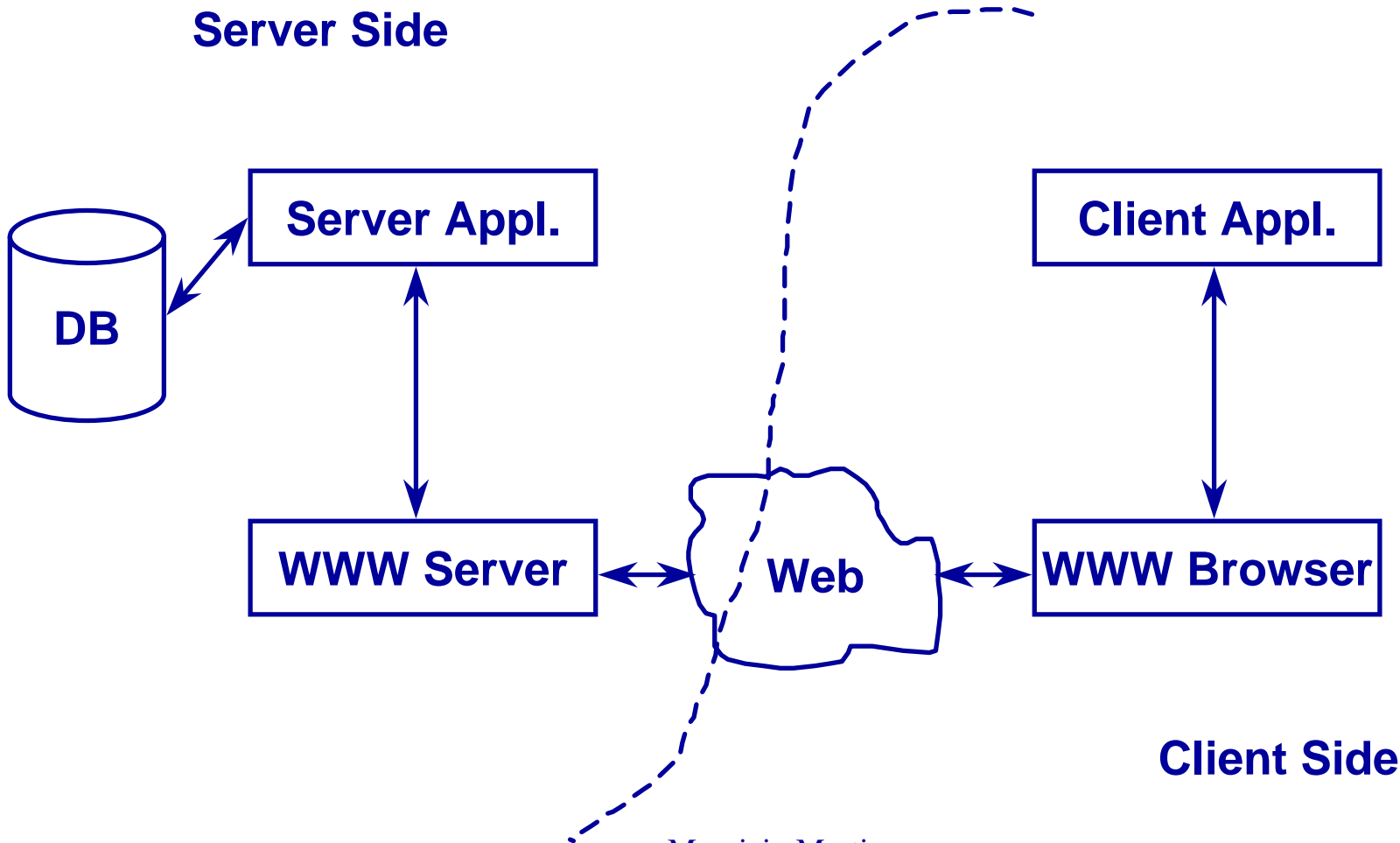
- One of the key features that Salutation tries to provide is **transport protocol independence**.
- The communication between clients and functional units (services) can be in a number of ways.
 - » In **the native mode** these may use **native protocols** and talk to one another directly without the SLMs getting involved in data transfer.
 - » In the **emulated mode, the SLMs will manage the session and act as a conduit for the data**, delivering them as messages to either party.
 - » In the **salutation mode, SLMs not only carry the data, but also define the data formats to be used in the transmission**. In this mode well-defined standards of interoperation with functional units are to be followed, allowing generic inter-operability.
- All communication in the latter two modes involve APIs and events which are well defined.

Interoperability – Jini / UPnP / Salutation (c)

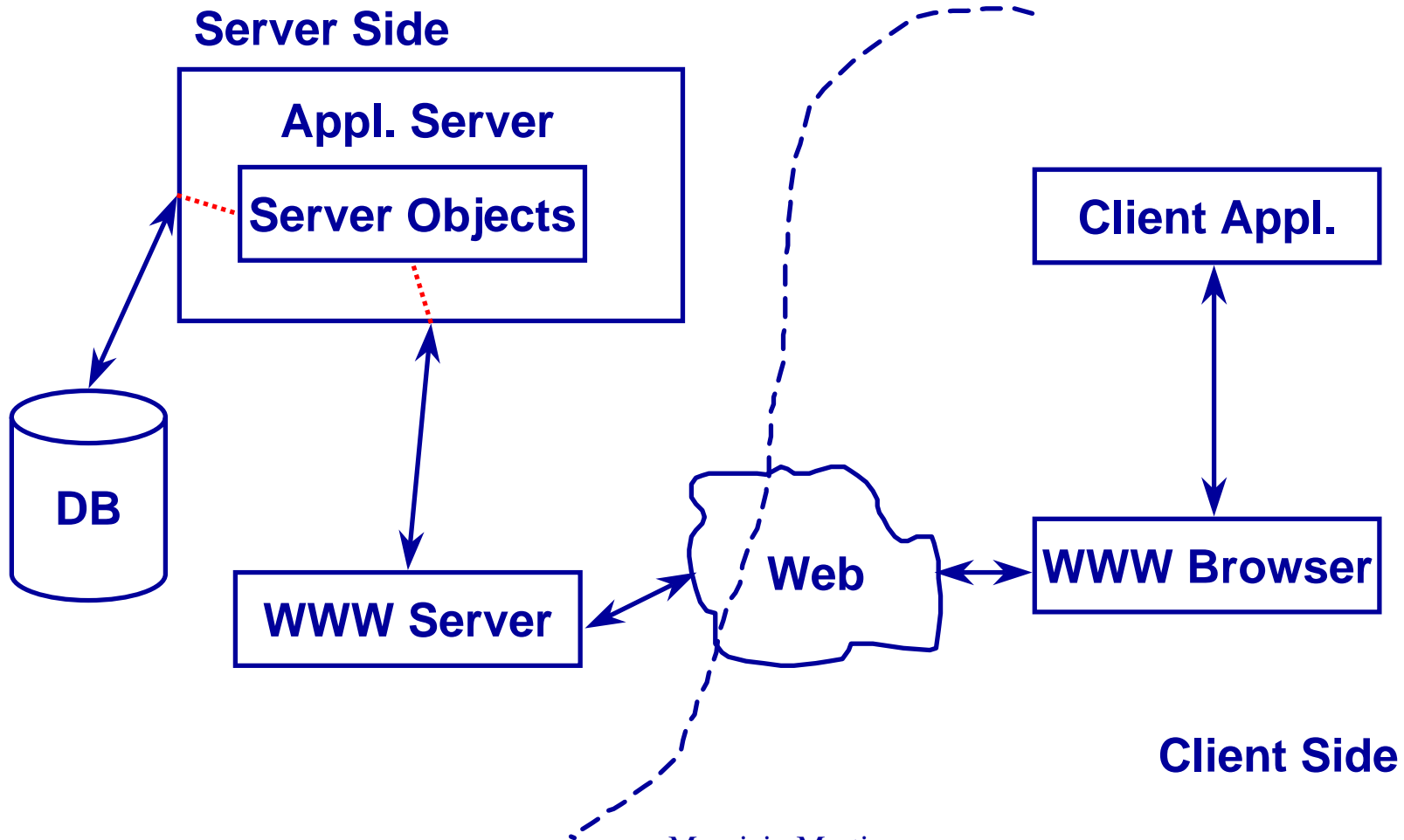
- Data descriptions follow a popular notation/encoding called **ASN.1** (Abstract Syntax Notation One) from OSI.
- Instead of lease management, SLMs can be asked to **periodically check the availability of functional units and report the status**. This allows a client or functional unit (service) to become aware when either have left the scene.
- **Salutation has already defined a number of Functional Units** for various purposes like faxing and voice mail. These also allow the use of vendor specific features.

Interoperability - WWW

Server Side



Interoperability - WWW



Interoperability - WWW

- ◆ CGI/SSI
- ◆ Applets/Servlets
- ◆ DHML / JavaScript / Java / CORBA / DCOM
- ◆ (Web) Application Servers
- ◆ HTTP + XML = XML-Protocols (SOAP)

Interoperability - Before WS

- ◆ Every distribution / interoperability infrastructure is based on:
 - a communication protocol
 - an agreement, a standard on how to represent data

- ◆ DCE-RPC (Distributed Computing Environment – Remote Procedure Call)
 - RPC Protocol
 - NDR (Network Data Representation)

- ◆ DCOM (Distributed Common Object Model)
 - OBJECT Protocol (on top of DCE-RPC Protocol)
 - NDR (Network Data Representation)

- ◆ CORBA (Common Object Request Broker Architecture)
 - GIOP (General Inter ORB Protocol) / IIOP (Internet Inter ORB Protocol)
 - CDR (Common Data Representation)

- ◆ Java RMI (Remote Method Invocation)
 - JRMP (Java Remote Method Protocol)
 - None (Why?)

◆ Problems

- Different Protocols
- (Very) Difficult Configuration Management
- Not suitable for Firewalls / Proxies
- Different Data Representations
- Every infrastructure imposes to buy/use a particular set of tools, systems, etc... (and this isn't dating, it's getting married)

Interoperability - Introduction to WS

◆ The Solution: KISS (Keep It Simple Stupid)

- Use very well known and accepted protocols for communication (e.g. HTTP – Hyper Text Transfer Protocol, SMTP – Simple Mail Transfer Protocol)
- Use the simplest possible data representation formats, that is ASCII, sorry XML (eXtensible Mark-up Language, actually XML-UTF-8 or XML-UTF-16)

Interoperability - Introduction to WS (cont)

- ◆ The KISS principle generated (is generating) various HTTP+XML actual implementations (see <http://www.w3.org/2000/03/29-XML-protocol-matrix>).

The most interesting are:

- **SOAP** (Ariba, Commerce One, Compaq, DevelopMentor, Hewlett Packard, IBM, IONA Technologies, Lotus Development, Microsoft, SAP, UserLand, Apache)
- **XMI** – XML Metadata Interchange (Unisys, Fujitsu, IBM, Recerca Informatica, Oracle, Daimler-Benz Platinum Technology)
- **XML-RPC** (Userland)

Interoperability - Introduction to WS (cont)

- ◆ XML-RPC is the simplest specification among the ones mentioned in the previous page: XML-RPC covers only RPC
- ◆ SOAP and XMI were supposed to be, on the contrary, “lightweight” protocols for the exchange of information (not only RPC).

Interoperability - Introduction to SOAP (cont)

◆ WS (SOAP) Request Example

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml;
charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
```

```
<SOAP-ENV:Envelope
xmlns:SOAPENV=http://schemas.xmlsoap.org/soap/envelope/
SOAPENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Interoperability - Introduction to WS (cont)

◆ WS (SOAP) Response Example

```
HTTP/1.1 200 OK
Content-Type: text/xml;
charset="utf-8"
Content-Length: nnnn
```

```
<SOAP-ENV:Envelope
xmlns:SOAPENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAPENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse xmlns:m="Some-URI">
      <Price>34.5</Price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Interoperability – WS Pros & Cons

◆ Pros

- Easy Interoperability
- No data representation problems
- No configuration problems (firewalls, proxies)
- No dependency on the hardware platform, on the operating system, on the language, on the compiler
- Standard (?)

◆ Cons

- (Very) High band-width usage
- The XML Protocols are evolving

Interoperability – WS Further Evolutions

- ◆ SOAP has been declared deprecated by Microsoft.
- ◆ The new fuzzy words are now: “Remoting” and “Indigo”.
- ◆ Web services are here to stay, even if the single enabling technologies may change.

- ◆ There’s a current evolution from the “Distributed Architecture Models” (e.g. CORBA, RMI, DCOM) to the “Services Oriented Applications” (I.e. Web Services / Indigo) [Indigo].

- ◆ And reality? Sockets as a basis?

Interoperability - 10 Golden Rules

- ◆ **Separate your tiers** - in particular, separate your user interface from your business logic.
- ◆ **Distinguish components from objects** - components are a packaging tool, objects are an implementation tool.
- ◆ **Distinguish interfaces from classes** - Interfaces define behaviours, classes implement behaviours.
- ◆ **Use tier-appropriate technology.**
- ◆ **Eliminate component state** - otherwise no pooling can be supported.
- ◆ **Enclose algorithms in transactions** - transactions make error handling easier.

Interoperability - 10 Golden Rules (cont)

- ◆ **Choose environments carefully** - it is a lifetime decision; you are not renting, you are buying; you are not dating, you are getting married.
- ◆ **Design decisively, implement incrementally** - don't start any implementation till the design is ready.
- ◆ **Write them right** - do it right, even if you don't think anybody will appreciate the effort; they will, in time.
- ◆ **Allow time** - this is tough work, most failures can be attributed to unrealistic expectations on how long it will take to implement the system.

[MSCDA]

Interoperability - Summary

- ◆ Definitions
- ◆ Issues
- ◆ Distributed Architectures
- ◆ CORBA
- ◆ OLE/DCOM
- ◆ Java Beans
- ◆ Jini / UPnP / Salutation
- ◆ WWW
- ◆ XML Protocols / WS
- ◆ 10 Golden Rules

Interoperability - Further Reading

- ◆ The CORBA documentation can be found at:
<http://www.omg.org>
- ◆ The OLE/DCOM documentation can be found at:
<http://www.microsoft.com>
- ◆ The Java related documentation at:
<http://java.sun.com>
- ◆ The XML / SOAP documentation can be found at:
<http://www.w3.org/2000/03/29-XML-protocol-matrix>.
- ◆ [Mars] R. Orfali, D. Harkey, J. Edwards, “The Essential Distributed Objects Survival Guide”, Wiley, 96, ISBN: 0-471-12993-3.
- ◆ R. Grimes, “DCOM Programming, Wrox, 97, ISBN: 0-861000-60-X.
- ◆ [MCDA] Roger Sessions, “COM and DCOM, Microsoft’s Vision for Distributed Objects”, Wiley Computer Publishing, 98, ISBN: 0-471-19381-X
- ◆ [Indigo] C. Peiris, “15 Seconds: Indigo Programming Model”, Internet.com.