

MetaComputing

Naming, RPC, PSEs

CS 594 Spring 2001
Dr Graham E Fagg

Some Material from talks by Mark Baker, Henri Casanova and Keith Moore.

Contents - Naming Services

- Naming
 - What's in a name?
 - Naming, binding, and distribution.
 - Name consistency.
 - Scaling.
 - Capabilities.
 - Examples.

Naming Services - Emphasis

- Naming is intimately related to network (access and location) transparency and migration transparency.
- We need a separate name service in addition to the naming schemes used in the context of specific services.

Naming Services - Emphasis

- There are separate concerns to be addressed:
 - Designing the name space.
 - Meeting administrative requirements to partition the name space.
 - Creating an implementation that scales.

Naming Services - Emphasis

- A system such as the DNS that resolves names on a world-wide scale represents a considerable engineering achievement - as does the system of routing daemons that resolves IP addresses.
- Should understand the engineering problems and strong motivation for getting naming right.
- **Caching and replication** are key to naming service implementation.
 - Compare Internet2, IBP & RCDS

Things that are named...

- People.
- Files.
- Machines.
- Services.
- Programs.
- Variables.
- Datatypes.
- Mailing lists.

What Names Do

- Uniquely identify something.
- Can name a not-yet-created object.
- Specify membership in a group.
- Provide information about location, function.
- Convey knowledge of a secret.

Embedding Information in Names

- Names can be **location-independent** or **location-dependent**.
 - `http://www.cs.utk.edu/` is location-dependent.
 - `/user/me` is not.
 - Is PVM Tid 0x40001 ????
- At an extreme, a pure name conveys no information about the named object, except by convention:
 - `/u/pvm/bin/foo` is probably some sort of executable.

Embedding Information in Names

- Impure names imply a commitment not to change the embedded information:
 - 423 - 9745790 *must* stay in the 423 area code.
- What happens when the area code changes to 865??
 - Do we change all the Names? If so WHO does it and makes sure its correct, how do we know everybody has changed including those outside our domain?
 - What about calls that use it now?
 - Are the updates synchronous?

Pure Names and Location

- How do we find an object with a pure name?
 - Example system that provide pure names;
 - Legion and Globe.
 - SNIPE provides location information relating to which RCDS server in which DNS domain to ask first....
- Use name servers.
 - Server remembers where objects are.
 - Three ways to use them:
 - Hash name to a name server; object tells server where it is... a matrix of names...
 - Name contains location of name server.
 - Hierarchical naming.

Pure Names and Location

- But how to find name servers?
 - Hacks...
 - ENV, hard coded..
 - Alternative - broadcast name request to everybody...
 - Cache location hints to avoid lookups.
 - (Maybe just ask portmap on port 111)
 - 111 is a hard coded value.

Hierarchical Naming

- Like UNIX files.
- Name is a sequence of components, separated by dividers.
 - Example:
`fagg/cs/utk/u/server`
- Can implemented by a hierarchy of name servers.
 - Find each name server from its parent.
 - How to find parent?
 - Answer - hacks...

Relative versus Absolute Naming

- Absolute - every process sees the same namespace.
 - Like UNIX.
 - **Advantage** - same name means same thing to everybody.
- Relative - every process has its own root
 - Like World Wide Web.
 - Complex tangle of directories.
 - **Advantages** - easy to customise, no need for central root server.

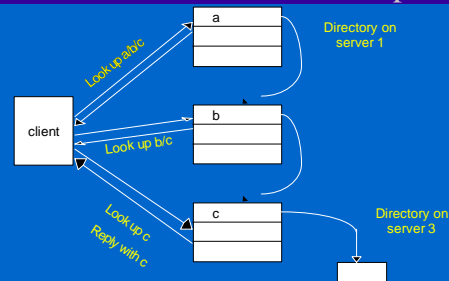
Errors and Inconsistencies

- What if the requested name didn't exist?
 - Maybe it was not created yet.
 - Maybe it was deleted.
 - Maybe a server was down.
 - Maybe name was misspelled.
- With caching, or replicated name servers, inconsistencies can arise.
- Deal with them by:
 - Not caching.
 - Ignoring the problem.
 - Using a real distributed database.

Scaling and Uniqueness Control

- In a large system, how to make sure names are unique?
- **Solution** - central registry.
 - Problem - Doesn't scale.
- **Solution** - local management of sub-trees of hierarchical name space.
 - Problem - This can result in inefficiencies.

Distributed Name Lookup



Naming and Security

- Naming and security are closely related.
- Names are “trusted” to refer to certain things.
- The name server(s) must be trusted.
- **Capabilities** combine naming and security.

Capabilities

- A capability is a secret name for an object, complete with access permissions.
- Anybody who knows the secret name can operate on the object as specified by those permissions.
 - An object might have many capabilities, each conferring different rights.

Capabilities

- Common implementation - capability is a digitally-signed statement from some authority that says *“The bearer has permission to perform operations X, Y, and Z on object O”*.
 - Often uses public-key cryptography.

Naming Methods

- Many fancy mechanisms have been proposed or implemented.
 - **Symbolic links** - indirection through names.
 - **Filter links** - a “view” of a directory that hides or renames some of the contents.
 - **Union directories** - a single directory that appears to contain all files in two or more other directories.
 - **Computed directories** - not a directory at all, but the output of some program.
 - CGI scripts or SQL query lookup results

Case Study - DNS

- The **Domain Name System (DNS)** is a name service whose principal naming database is used across the Internet.
- It replaced the original naming scheme in which all host names and addresses were held in a single central file and downloaded by **FTP** to all the computers that required them.

DNS

Shortcomings of the original scheme:

- It did not **scale** to large numbers of computers.
- Local organisations wished to administer their own naming schemes.
- General name service was needed, not just one that served to lookup computer addresses.

DNS

- The objects named by DNS are primarily computers - mainly IP numbers are stored as attributes.
- In principle any type of object named - its architecture gives rise to a variety of implementations.
- It also allows organisations or departments to manage their own naming data.

DNS

- DNS holds hundreds of thousands of names around the world.
- Any name can be resolved by any client.
- This is achieved by **hierarchical partitioning of the database**, by replication of naming data and caching.

DNS - Queries

- **Host name resolution** - generally applications use DNS to resolve host names into IP addresses.
- **Mail host location** - electronic mail software uses the DNS to resolve domain names into the IP addresses of mail hosts - computers that will accept mail for those domains
- **Reverse Resolution (RARP)** - some software applications requires a domain name to be returned given an IP address... why ?

DNS - Queries

- **Host Information** - the DNS can store the machine architecture type and operating system against the domain names of the hosts.
- **Well know services** - a list of the services run by a computer (`ftp/telnet`) and the protocol used to obtain them (`UDP/TCP`) can be returned, given the computer's domain name.

DNS - Name Servers

- The problems of scale are treated by the combination of partitioning the naming database and by replicating and caching parts of it close to the points of need.
- The DNS database is distributed across a logical network of servers.
- Each server holds part of the naming database - **primarily data** for the local domain.

DNS - Name Servers

- Most queries concern computers in the local domain, and are satisfied by servers within that domain.
- However, each server records the domain names and addresses of other name servers, so that queries pertaining to objects outside the domain or in a separately administered sub-domains can be satisfied.

DNS Name Servers

- Naming data is divided into zones - each zone contains the following:
 - Attribute data for names in a domain.
 - The names and addresses of servers that provide authoritative (reliable) data for the zone.
 - The names and addresses of servers that hold data for delegated sub-zones.
 - Zone management parameters, such as those governing the caching and replication of zone data.

Types of DNS query/response

Name	Description	Remarks
A	IP address	To query / response the IP address of a host name.
NS	name server	To query / response the authoritative name server for a domain.
CNAME	canonical name	To query / response the alias of a host name.
PTR	pointer control	To query / response the host name of an IP address.
HINFO	host info	To query / response the host information. The response is two arbitrary strings specifying the CPU and operating system.
MX	mail exchange record	To query / response the mail relay domain of a particular domain.

DNS Name Servers

- The DNS architecture specifies that each zone must be replicated authoritatively in at least two failure-independent servers.
- There are two types of server that are considered to provide authoritative data:
 - Primary (*master*) servers are the ones that read zone data directly from a local master file.
 - Secondary (*slave*) servers download zone data from a primary server.

DNS Name Servers

- Servers communicate periodically with the primary server, to check whether their stored version matches that held by the primary server.
- If a secondary's copy is out of date, the primary sends it the latest version.
- The frequency of the secondary's check is set by administrators as a zone parameter, and its value is typically once or twice a day.

DNS Name Servers

- Any server is free to cache data from any other server, this avoids having to contact them again when the same data is required - this *cache is set as non-authoritative data*.
- Entries in a zone have a **time-to-live value**.
- When a non-authoritative server obtains data from an authoritative server, it notes the time-to-live.

DNS Name Servers

- It will only provide its cached data to clients for up to this time; when queried after the time period has expired, it recontacts the authoritative server to check its data.

DNS - Navigation and Query Processing

- In DNS, the user agent is called a **resolver**.
- It accepts queries, formats them into messages expected under DNS protocols, and communicates with one or more name servers in order to satisfy the queries.
- A simple request-reply protocol is used, typically using **UDP** (using port 42/53).

DNS - Navigation and Query Processing

- The resolver times out and resends its query if necessary - primary/secondary name servers.
- DNS protocol allows multiple queries to be packed into a single request packet - saves network bandwidth.

DNS - BIND

- The Berkeley Internet Name Domain (BIND) is an implementation of DNS for systems running BSD UNIX.
- Client programs link in library software to act as the DNS resolver.
- DNS “name server” computers run the *named* daemon.
- BIND allows for three categories of name server: primary, secondary and caching-only servers.

Case Study RCDS

- Resource Cataloging and Distribution System (RCDS)
- Used to publish “metadata” or just web pages on the internet
 - Used as a publishing tool for documents
 - Used by SNIPE for internal and external data
 - External = how to contact me, what I offer, and more on how, who and why
 - Used by HARNESS for repository location

“It doesn’t work when I click here!”

- URLs become stale
- Web servers fail or become overloaded
- Network links fail or become congested
- The server might be halfway across the world

Why Good URLs Go Bad

- Web pages move from one host to another
- Directory structures get re-organized
- Hostnames change or are re-assigned
- Sometimes the resource no longer exists

Sources of Failure

- A URL is tied to a particular server, and...
 - not a location independent name
- Everyone uses the same server...
- The network fails... or
- The server becomes overloaded...
- Links to the server become congested ...
- Users cancel and restart their downloads...
- Can you say “snowball effect”?

Inefficient Network Utilization

- TCP has over a 10× overhead (short xfers)
 - TCP connect send to machine and disconnect = 1.4mSec under Linux
 - UDP packet to and from machine 100uSec
- TCP does not degrade gracefully during connection establishment and slow-start
- HTTP cannot be multicast
- Web browsers do not exploit mirror sites

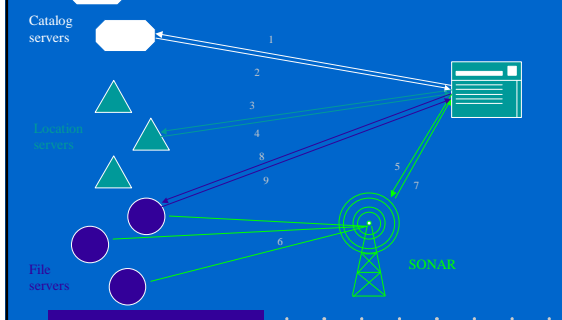
The RCDS Approach

- Resources are replicated to multiple servers (which may be geographically dispersed)
- Location database is updated when replicas are created or deleted
- SONAR helps users and apps find which replicas are nearby
- Resource descriptions contain integrity checks and are cryptographically signed

RCDS Components

- Publication Tools
- Replication (“mirroring”) Tools
- Collection Managers
- Catalog Servers
- Location Servers
- File Servers (http, ftp, etc.)
- Clients (web browser, proxy server, etc.)

Retrieval Flow



RCDS Innovative Features

- Resolution of URLs, URNs and LIFNs
- Scalable replication of Location and Catalog info
- Multiple transfer protocols for a single resource
- Multiple representations for a single resource
- Robustness/fault-tolerance features
- Support for correct caching
- Easy deployment

URLs, URNs, and LIFNs

- URL - string encoding of an access protocol
- URN - stable, resolvable, resource name
 - name-to-resource binding can change
- LIFN - stable, resolvable resource name
 - name-to-resource binding never changes
 - all replicas of the resource are identical

Location Independent File Name (LIFN)

- A LIFN is a kind of URN
- Names a particular sequence of octets
- Once assigned, it can never be changed
- Used for communication within RCDS
 - replication tools need an exact file name
 - location information is indexed by LIFN
 - resource descriptions contain LIFNs which precisely describe instances of the resource

URN Resolution

(works for URLs and LIFNs too!)

urn:inet:netlib.org:pvm3

How to resolve URNs of type **inet**? → **DNS**

HAPTN `h/(/*)(/*1.*/*1/)` ←

Where do I find servers for **netlib.org**? → **DN**

SRV `znds.cs.uk.ac.edu`
SRV `znds.epm.ornl.gov`
SRV `znds.cs.rice.edu`

Tell me about **urn:inet:netlib.org:pvm3** → **RCDS**

Title: Parallel Virtual Machine
Author: Robert Manchek
Content-Type: application/x-tar
MD5: 30a480cff1035ba0343a8..
LIFN: netlib.org:199606041143.3434

How LIFNs are Used

URN:netlib.org:pvm3

LIFN: netlib.org:199606041143.3434

Title: Parallel Virtual Machine
Author: Robert Manchek
Content-Type: application/x-tar
MD5: 30a480cff1035ba0343a8..
LIFN: netlib.org:199606041143.3434
Copying-Policy: free
Netlib.Language: C

Locations:
<http://netlib.cs.uk.ac.edu/pvm3/pvm3.11.tar.gz>
<http://netlib.att.com/pvm3/pvm3.11.tar.gz>
<http://netlib1.epm.ornl.gov/pvm3/pvm3.11.tar.gz>
<http://www.hensa.ac.uk/netlib/pvm3/pvm3.11.tar.gz>

resource description locations

How LIFNs are Used

URN:netlib.org:pvm3

LIFN: netlib.org:199606041143.3434

Title: Parallel Virtual Machine
Author: Robert Manchek
Content-Type: application/x-tar
MD5: 30a480cff1035ba0343a8..
LIFN: netlib.org:199606041143.3434
Copying-Policy: free
Netlib.Language: C

Locations:
<http://netlib.cs.uk.ac.edu/pvm3/pvm3.11.tar.gz>
<http://netlib.att.com/pvm3/pvm3.11.tar.gz>
<http://netlib1.epm.ornl.gov/pvm3/pvm3.11.tar.gz>
<http://www.hensa.ac.uk/netlib/pvm3/pvm3.11.tar.gz>

When a resource is changed, a new LIFN is issued...

How LIFNs are Used

URN:netlib.org:pvm3

LIFN: netlib.org:199606041143.3434

Title: Parallel Virtual Machine
Author: Robert Manchek
Content-Type: application/x-tar
MD5: 30a480cff1035ba0343a8..
LIFN: netlib.org:199606041143.3434
Copying-Policy: free
Netlib.Language: C

Locations:
<http://netlib.cs.uk.ac.edu/pvm3/pvm3.11.tar.gz>
<http://netlib.att.com/pvm3/pvm3.11.tar.gz>
<http://netlib1.epm.ornl.gov/pvm3/pvm3.11.tar.gz>
<http://www.hensa.ac.uk/netlib/pvm3/pvm3.11.tar.gz>

As replicas of the new resource are created, they are added to the list of locations for the new LIFN

How LIFNs are Used

URN:netlib.org:pvm3

LIFN: netlib.org:199606041143.3434

Title: Parallel Virtual Machine
Author: Robert Manchek
Content-Type: application/x-tar
MD5: 30a480cff1035ba0343a8..
LIFN: netlib.org:199606041143.3434
Copying-Policy: free
Netlib.Language: C

Locations:
<http://netlib.cs.uk.ac.edu/pvm3/pvm3.11.tar.gz>
<http://netlib.att.com/pvm3/pvm3.11.tar.gz>
<http://netlib1.epm.ornl.gov/pvm3/pvm3.11.tar.gz>
<http://www.hensa.ac.uk/netlib/pvm3/pvm3.11.tar.gz>

Copies of obsolete resources may be deleted as new resources are propagated to servers

Propagation of Location Information

LIFN: netlib.org:199606041143.3434

<http://netlib.cs.uk.ac.edu/pvm3/pvm3.11.tar.gz> ↔ <http://netlib1.epm.ornl.gov/pvm3/pvm3.11.tar.gz>

↕

<http://netlib.att.com/pvm3/pvm3.11.tar.gz> ↔ <http://www.hensa.ac.uk/netlib/pvm3/pvm3.11.tar.gz>

Ease of Deployment

- For providers:
 - update DNS servers (new version of named)
 - install RCDS servers
 - register RCDS servers in DNS
 - register resources in RCDS
- For repositories:
 - switch to RCDS-aware mirror tools
- For users:
 - install RCDS-aware proxy, SONAR

RCDS use within SNIPE

- Used to store information about all entities such as hosts, file servers, repositories, processes, classes of processes.

- Examples:

```
- x-snipe://cs.utk.edu/snipe_app/fagg/sol/29481/10674.183121/
- serial 7
- assertion 0 SNIPE_COMM_TCP_FBACK [latin1] = "sol.cs.utk.edu:7647" 10674.15821983
10681.15821983 1 5
- assertion 1 SNIPE_STATE [latin1] = "2" 10674.15821950 10681.15821950 1 4

- sol->query_name_test "*" x-snipe://cs.utk.edu/snipe-reg/app/class/
- urn x-snipe://cs.utk.edu/snipe-reg/app/class/
- serial 11
- assertion 0 class-count [latin1] = "2" 10641.3953375 10648.3953375 1 10
- assertion 1 class:0 [latin1] = "demo0" 10641.3203285 10648.3203285 1 9
- assertion 2 class:1 [latin1] = "benchmarking-dist" 10641.3953392 10648.3953392 1 11
```

RCDS use within HARNESS

- Used as a method of resolving requests for Java class libraries at run time

Harness at SC 98

- Built a replicated repository of class libraries that Emory's system used to dynamically download during execution
 - Had a few hiccups such as the system needed only specify a single URL as the Java loaded only expected the class libraries to be on a single web server or file system.
 - System was built from the old RCDS web-personal proxy

Harness at SC 98

- How it worked
 - The Emory system asked for class libraries from a specific web server
- `edu/Emory/mathcs/harness/services/H_RemoteRunnable.class`
- `http://proxy/rcds-urn/harness/edu/Emory/mathcs/harness/services/H_RemoteRunnable.class`
- The URN is then associated with multiple LIFNs (Location Independent File Names) which each point to a URL where each copy of the repository is.

Harness at SC 98

- Each repository is a copy of the class libraries
 - may only be a partial copy if disk space is to be conserved or if only the commonly used plugins (class libraries) are to be replicated
- 3 Replicated copies at SC98 + 2 else where
 - one on the Emory machine (Solaris)
 - one on the ORNL machine (sunu6f? Solaris)
 - one on a UTK laptop (Apache under Linux)
 - + sol.utk.edu and another at mathcs.Emory.edu

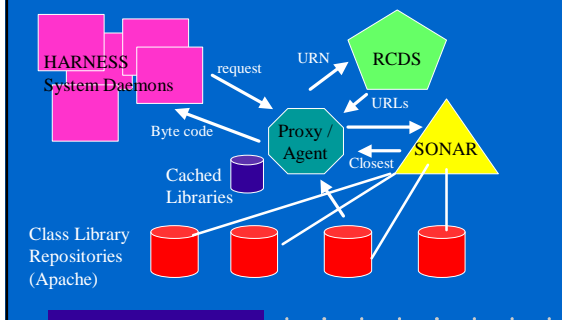
Harness at SC 98

- Once the URL had been requested, the proxy/agent would get the list of other URLs from the RCDS server.
- The list could then be sorted via connectivity using SONAR

Harness at SC 98

- As the Java run-time was unable to understand a redirect, and only expected the binary class library, the proxy would get the class and then forward it.
 - Allowing caching... which is very very useful as if one part of the system needs a certain class library (plug-in) then so will the rest if we are maintaining a comprehensive synchronous system.

Harness at SC 98



RPC

- RPC makes it possible for a client to access a remote service by simply calling a local procedure.
- The RPC interface makes it possible for client programs to be written in a simple way, familiar to most programmers.
- RPC makes it easy to run existing codes in distributed environments with few, if any, changes.

Writing a Client and a Server

- The RPC system consists of a number of components, including languages, libraries, daemons and utility programs.
- Together these make it possible to write client and servers.
- In the client/server system, the glue that holds everything together is the interface definition.

Interface Definition Language

- Interface Definitions are written in a language named IDL.
- It permits procedure declarations in a form closely resembling function prototypes in ANSI C.
- IDL files can also contain type definitions, constant declarations, and other information needed to correctly marshal parameters and unmarshal results.

Interface Definition Language

- A crucial element in every IDL file is an identifier that uniquely identifies the interface - the client sends this identifier in the first RPC message and the server verifies that it is correct.
- In this way, if a client in-advertently tries to bind to the wrong server, or even to an older version of the right server, the server will detect the error and the binding will not take place.

Interface Definition Language

- The first step in writing a client/server application is usually calling the *uuidgen* program, asking it to generate a prototype IDL file containing an interface identifier guaranteed never to be used again in any interface anywhere generated by *uuidgen*.
 - Universal Unique Identifier - 128 bits.
 - Contains a timestamp and a host identifier.
 - Used to uniquely identify a service.
 - e.g., **uuid**

Interface Definition Language

- The next step is editing the IDL file, filling in the names of the remote procedures and their parameters.
- When the IDL file is complete, the IDL compiler is called to process it - this outputs three files:
 - A header file (e.g. **interface.h** in C terms).
 - The client stub.
 - The server stub.

Interface Definition Language

- The header file contains the **uuid**, **type definition**, **constant definition**, and **function prototypes**.
- The client stub contains the actual procedures that the client program will call - these procedures are responsible for collecting and packing the parameters into the outgoing message and then calling the runtime system to send it.

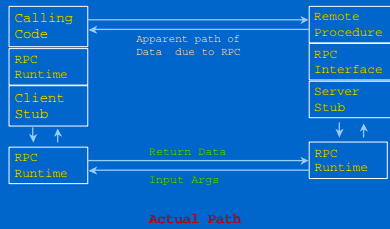
Interface Definition Language

- The server stub contains the procedures called by the **runtime system on the server** machine when an incoming message arrives - these, in turn, call the actual server procedures that do the work.
- The next step is for the application writer to write the client and server code - both of these are then compiled, as are the two stub procedures.

Interface Definition Language

- The resulting **client code and stub object files** are then linked with the runtime library to produce the **client executable**.
- A similar procedure happen with the server side code.
- At runtime, the client and server are started to make the application run.

Effect of the IDL and RPC Runtime



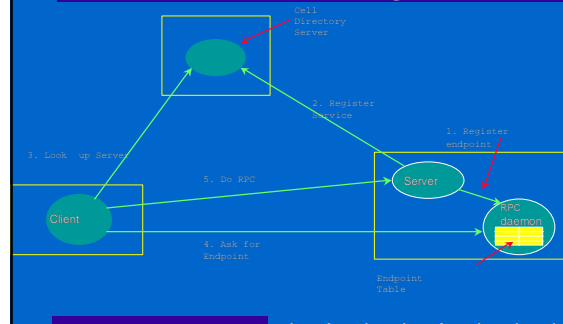
DCE RPC

- Interface Definition Language - The usual...
- Uuidgen:
 - Generates a prototype IDL file containing unique ID (embedding location and timestamp)
 - ONC version is just called `rpcgen`
 - Timestamps help with security
- IDL compiler produces - Header file, Client and Server stubs.
- Semantics:
 - At-most-once (default).
 - At-least-once (for important operations, such as read).

RPC Binding

- Server registers with local **RPC daemon**
 - Passes endpoint (port) to daemon (1).
- Server registers with **Cell directory** (2).
- Client looks up service's location with **directory server** (3).
- Client goes to well-known port to ask RPC daemon for **endpoint of service** (4).
- Client performs RPC using endpoint (5).

Client-to-Server Binding in DCE



Other Interface Designs

- Encapsulate the whole server in such a way that you can manipulate it as a complete entity (object/component)
 - Requires more daemons for naming and resolution services
 - communications may have to be indirect to allow for security and translation
 - interfaces may be *dynamic*, I.e. not known until run-time (linking) or even at call (invoke) time.
 - Not compiled in.

Component Based Distributed Computing

- OO Distributed Technologies
- CORBA
- DCE vs CORBA
- JavaBeans

Distributed Technologies

- Rather it is the distributed object technology which can build general multi-tiered enterprise intranet and internet applications.
- CORBA (Common Object Request Broker Architecture) is turning from a sleepy heavyweight standards initiative to a major competitive development activity that battles with COM and JavaBeans to be the core distributed object technology.

Distributed Technologies

- The commercial architecture is evolving rapidly and is exploring several approaches which co-exist in today's (and any realistic future) distributed information system.
- The most powerful solutions involve *distributed objects*.
- There are three important commercial object systems - CORBA, COM and JavaBeans.

OO Programming Languages

- An object-oriented program, for example in Smalltalk, Java or C++ consists of a collection of interacting objects, each of which provides a service specified by its interface.
 - Like a server interfaced by RPC

OO Programming Languages

- **Objects communicate with one another by sending messages.**
 - A message is a request for an object to perform one of its methods (operations).
 - It is the object that receives a message – not the sender – that determines how to carry out the method.
 - The set of messages to which an object can respond is called its **interface**.

OO Programming Languages

- **Objects “encapsulate” data and the “code” of their methods.**
 - In other words, the sender of a message requesting a method does not know how it is carried out.

Object Identity

- Each object has an **identifier** in the underlying object system, for example the value of a Java object reference is an **object identifier** or **OID**.
- The identifier of an object enables it to be indicated as the target of a message.
- Object-oriented programming systems support enquiries as to whether one OID refers to the same object as another. (e.g. *equals* in Java).

Actions

- An action in an object-oriented program is initiated by an object sending a message to another requesting a *method invocation*.
- The object invoked executes the appropriate method and then returns control to the invoking object.
- An invocation of a method can have two effects:
 - The state of the object may be changed.
 - Further invocations on methods in other objects may take place.

Actions

- As an invocation can lead to further invocations of methods in other objects, an action is a chain of related invocations each of which eventually returns.
- A message is a request for an action and can have additional information (arguments) needed to carry out the action.

Actions

- The receiver will perform some method (execute its code) to carry out the request.
- The interpretation of a message can vary with different recipients:
 - e.g. a message to an object to draw itself would result in a different action according to whether the object is a square or a circle.

Classes and Instances

- Since classes are used to create new objects, each object is an instance of a class.
- All the instances of a class use the same methods and can share the code of the implementation, but each instance has its own set of instance variables with their own values.
- Classes in programs can be organised as a hierarchy in which one class can make use of the code of another.

Classes and Instances

- Single inheritance allows a class to make use of the code of one other class - that is it can be a sub-class.
- A sub-class specifies that all of its instances will be the same as instances of another class (its superclass) except for differences explicitly stated.

Classes and Instances

- These differences may consist of simple extensions (extra data and methods) or they may consist of re-definitions (overrides) of the methods of the parent class.
- For example, a class *Shape* may define the properties common to all graphical objects and the classes *Circle*, *Square*, etc. will define the properties specific to circles and squares.

Interface versus Implementation

- The users of an object just see the interface view of its class, whereas the implementers see the details of how the data is represented and manipulated.
- Provided that the two views are independent, the implementer is free to improve the implementation at any time, either by adding functionality or by altering the data representation.

Interface versus Implementation

Interface of stack

```
int pop ()
void push (int n)
```

Implementation of stack

```
int stack [];
int stackpointer;
public final int STACKSIZE = 20;
public push(int n) {
    stack[stackpointer++] = n;
};
```

Multiple Inheritance

- Multiple inheritance allows a class to make use of the code of several other classes.
- Java does not provide multiple inheritance - but allows classes to implement several *interfaces* as well as inheriting from one class.
 - In Java, an interface is an abstract definition of the signatures of a set of methods.
 - Java interfaces are allowed to extend one or more other interfaces.

Distributed Technologies

- These have similar approaches and it is not clear if the future holds a single such approach or a set of inter-operable standards.
 - **CORBA** is a distributed object standard managed by the OMG (Object Management Group) comprised of **700+** companies.

Distributed Technologies

- **COM** is Microsoft's distributed object technology initially aimed at Window machines.
- **JavaBeans** (augmented with **RMI** and other Java 1.2 features) is the "pure Java" solution - cross platform but unlike CORBA, not really cross-language!

CORBA

- The OMG (Object Management Group), formed in 1989:
 - Aimed to encourage the adoption of distributed objects systems to gain the benefits of OO for software development and to make use of distributed systems which were becoming widespread.
- By this time, client-server architectures were very widely used.
- The OMG advocated the use of open systems based on standard OO interfaces.

CORBA

- The OMG introduced a metaphor - the *object request broker* whose role is to help a client to invoke a method on an object.
- This role involves:
 - Locating the object.
 - Activating the object if necessary.
 - Communicating the client's request to the object which carries it out and replies.
- The CORBA specification includes an IDL (Interface Definition Language).

CORBA

- Object interfaces are defined in **IDL**, thus enabling clients and objects to be implemented in different languages.
- **Note** - that interface languages were already widely used to enable clients to communicate by means of **RPC** with servers in different languages.

CORBA

- A major difference between CORBA and earlier RPC systems is that CORBA IDL is intended for defining interfaces to remote objects whereas the earlier IDLs were designed for specifying interfaces to servers.
- Therefore CORBA IDL provides a type allowing Remote-Object IDs (ROID) to be passed as arguments and results in RMI.

CORBA

- Another difference is that CORBA IDL provides a special client interface which allows client messages to be bound dynamically at run time.
- In 1991, a specification for an object request broker known as **CORBA (Common Object Request Broker Architecture)** was agreed by a group of companies including DEC, HP, Hyperdesk, NCR, Object Design and Sun.

CORBA

- The CORBA 2.0 and 3.0 Specifications are available on the OMG Web site Web at the following URL:
<http://www.omg.org/corbask.htm>
- The OMG home page and a list of technical reports can be accessed from there.

CORBA

- The CORBA object bus defines the shape of the components that live within it and how they inter-operate.
- What makes CORBA so important is that it defines **middleware** that has potential to subsume all other client/server middleware technologies.

CORBA

- Motivation:
 - Simplifies application interworking.
 - Foundation for distributed objects.
 - To be to distributed computing (C++ is to C).
 - Encapsulation, Inheritance, Exceptions, ...
- Components:
 - Object Request Broker (ORB).
 - Interface Definition Language (IDL).
 - Static Invocation Interface.
 - Dynamic Invocation Interface.
 - Object adapters.

Cobra's Object model

- The CORBA standard tries to obviate problems resulting from **boundaries** such as networks, programming languages, and operating systems.
- CORBA can be viewed as an environment to support the development of a complete new system or an environment for integrating legacy systems and sub-systems.

Cobra's Object model

- An implementation of the standard defines a language and platform-independent object bus called an ORB (Object Request Broker), which lets objects transparently make requests to, and receive responses from, other objects located locally or remotely.
- Each implementation of the CORBA standard is able to communicate with any other implementation of the standard, the protocol used to achieve this is the **Internet Inter-ORB Protocol (IIOP)**.

Cobra's Object model

- The IIOP specification defines a set of data formatting rules, called **CDR (Common Data Representation)**, which is tailored to the data types supported in the CORBA **Interface Definition Language (IDL)**.
- Using the CDR data formatting rules, the IIOP specification also defines a set of message types that support all of the ORB semantics defined in the CORBA core specification.

Cobra's Object model

- The messaging formats and the CDR constitute the **General Inter-ORB Protocol (GIOP)**.
- IIOP is the GIOP message format sent over the **Transmission Control Protocol/ Internet Protocol (TCP/IP)**.
- GIOP messages could be sent over any data transport protocol.
- Using IIOP any CORBA client can speak to any other CORBA Object.

Cobra's Object model

- A specialised ORB may use home-grown protocols to fine tune access between objects within the same ORB, however to be CORBA compliant, the ORB must use IIOP to communicate with objects on other ORB.
- The CORBA 2.0 specification dictates a common format for object references exchanged over IIOP, called **IOR (Interoperable Object Reference)** format.
- An IOR contains one or more profiles.

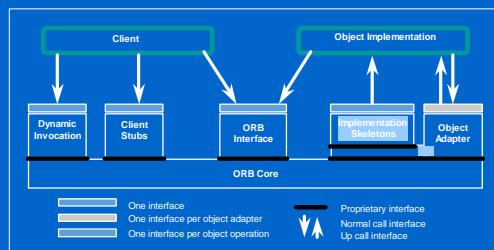
Cobra's Object model

- Each profile describes how a client can contact and send requests to the object using a particular protocol.
- All legal IORs must have at least one IIOP profile, thus ensuring that wherever that reference goes, any CORBA-compliant ORB will be able to locate the object and send requests to it.

Cobra's Object model

- The IIOP profile contains the **Internet address of the object's server** and a **key value** used by the server to find the specific object described by the reference.
- Object references can be converted into character strings, which can be published arbitrarily, like URLs, in email messages, files, databases, directories, and so on.
- Any CORBA-compliant application can convert the string into an IOR and use it to locate and invoke the object.

Object Request Broker (ORB)



CORBA Components

- **Client stub** - Each stub represents an object operation (a possible request) which a client invokes in a language - dependent manner (e.g., by calling a subroutine which represents the operation).
- **Dynamic Invocation** - Alternatively, a client may dynamically construct and invoke request objects which can represent any object operation.
- **Implementation Skeleton** - Each skeleton provides the interface through which a method receives a request.
- **Object Adapter** - Each object adapter provides access to those services of an ORB (such as activation, deactivation, object creation, object reference management) used by a particular ilk of object implementation.
- **ORB interface** - The interface to the small set of ORB operations common to all objects, e.g., the operation which returns an object's interface type.

Writing a CORBA Application

- The development process for writing a CORBA based application comprises of the following steps (typical):
 - Define IDL.
 - Generating client stub and server skeleton using IDL compiler
 - Object implementation (server).
 - Client implementation.
 - Register Server with an ORB daemon.
 - Run client by contacting an ORB daemon.

CORBA's Object Model

- CORBA does not provide transactions or any form of concurrency control or recovery.
- Nor does it provide any form of object replication, which limits the effectiveness of interactive programs.
- Client software can be designed with caches, but the design of a cache is difficult when the semantics of server objects are unspecified.
- In particular, CORBA IDL does not support the ability to provide a copy of an object.

CORBA's Object Model

- Currently OMG has defined 16 services of which 11 now have agreed standards.
- One of these services (POS - the persistent Object Service) allows objects to be suspended and activated using traditional or object-oriented database systems.
- Other services are described for transactions (which may be nested) and for concurrency control and recovery.
- Another service is specified for security.

CORBA's Object Model

- It seems likely that these services may be implemented in a variety of different ways, but they will provide a common set of IDL interfaces.
- The proposals and specifications for some of these services are available via the OMG Web pages.
- CORBA objects are generally fairly large-grain objects due to the considerable performance overhead for RMI.

CORBA's Object Model

- For example, the CORBA remote objects in a server may be implemented in an object-oriented language such as C++.
- In this case each CORBA remote object may be represented by a C++ class with an interface that matches the IDL interface.
- The C++ program that implements the objects for a CORBA server would generally include several classes, only one of which corresponds to each IDL interface.

Architecture of CORBA

- The architecture is designed to support the role of an "Object Request Broker" that enables clients to invoke methods in remote objects that have been implemented in a variety of languages.
- A *Server* in CORBA is a process that is executing the implementation of one or more remote objects.
- Although CORBA does allow servers to be run continuously, it is designed to allow servers to be activated on demand.

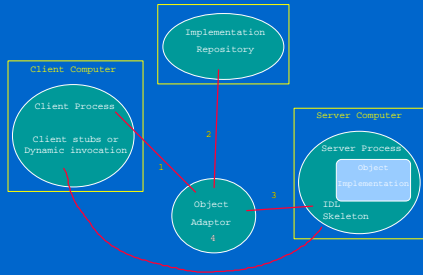
Architecture of CORBA

- Although the simplest form of the client-server model assumes that server processes run continuously it is not necessary to start all services when a system starts up.
- In particular, the standard set of *TCP* services such as *ftp* are not actually run unless clients require them.
- This is achieved by running a service called *inetd* which starts other servers whenever clients need them.

Architecture of CORBA

- Main components of the CORBA architecture are illustrated in next slide.
- It is similar to the client-server RPC architecture in its use of clients stubs.
- The server stubs are included in "IDL skeletons" in OMG CORBA specifications.
- If the server is implemented in an OO language (e.g. C++) then the IDL skeleton.
- If the server is implemented in a non OO (e.g. C), then the IDL skeleton consists of a set of server stub procedures.

Architecture of CORBA



Architecture of CORBA

- In CORBA, the term *object implementation* is used to refer to an executable file containing a program that will be executed as a server process.
- For example, in C++, the *object implementation* will be compiled from a *main* and all the necessary supporting classes.
- In the CORBA architecture, object implementations are registered with implementation repositories and activated when they are needed.

Object Adapters

- Glue between implementation and ORB:
 - Caller calls into Adaptor, which invokes object.
- Functions:
 - Object registration (as implementation of CORBA objects).
 - Object reference generation.
 - Server process activation.
 - Object activation.
 - Request demultiplexing.
 - Upcalls.

Architecture of CORBA

- The Object Adaptor deals with everything that a client needs at run time in order to invoke a method in a remote object - that is:
 - It registers object implementations in an implementation repository and accesses them as needed by clients.
 - It activates an object implementation in a server process whenever a client needs it.

Architecture of CORBA

- It registers the servers currently offering activated objects and accesses them as needed by clients.
- It creates ROIDS and translates between local OIDs and ROIDS.
- It includes a dispatcher that passes RMIs to the appropriate objects.

Architecture of CORBA

- The client (1) contacts an object adaptor, (2).
- The object adaptor contacts the implementation repository and (3) activates the object implementation in the server process.
- Finally (4) the client invokes methods using the ROID translation and dispatching functionalities of the object adaptor and the IDL skeleton.

Architecture of CORBA

- The components of CORBA can be implemented as libraries linked with client and server.
- Some of them may be implemented as a separate services.

Future of CORBA

- Been incorporated into Netscape - accessed via Internet Inter-Orb Protocol (IIOP).
- Object Management Group looking at higher-level components of an Object Management Architecture - not just CORBA

CORBA vs DCE

- **Procedural v. Object-Oriented:**
 - DCE IDL has no inheritance, parameterised exceptions.
- **Access to distribution infrastructure:**
 - CORBA has fewer facilities so far
 - Authentication, File services.
- DCE and CORBA can inter-operate.

Java - Introduction

- One of Java's advantages is that it is platform-neutral.
- Programs written in Java can run on a variety of operating systems and hardware configurations.
- Another advantage of Java is its compact format.
- Java **bytecode** files are easily transported across the Internet.

Java - Introduction

- They are then played by an interpreter on a local machine.
- Browsers, like **Netscape Navigator** and **Microsoft Internet Explorer**, automatically load and run Java applets directly from Web pages.

JavaBeans - Introduction

- JavaBeans technology makes it easier to develop applets for Web pages.
- With JavaBeans components, it is possible to create applets, and even full Java applications, without writing any Java code.
- JavaBeans are software components that can be connected to one another and reused in different combinations.

JavaBeans - Introduction

- The JavaBeans mission statement explains the goal of the technology - "Write once, run anywhere, reuse everywhere."
- JavaBeans technology combines the advantages of the Java language with the benefits of component architecture.
- As an extension of the Java language, Java Beans are fully portable, compact and networkable.

JavaBeans - Introduction

- JavaBeans is a component model that provides a framework for software components to be developed, delivered, and deployed using Java.
- Component technology is revolutionising software development, just as the assembly line revolutionised manufacturing.
- Components are small software modules that application developers can use, reuse, and combine to create complex programs with a minimum of effort.

JavaBeans - Introduction

- The component approach to building applications is efficient and less error-prone than traditional programming.
- JavaBeans extends the Java platform and makes it the basis for a complete software component system.
- Using visually-oriented Application Builder tools, designers can assemble JavaBeans components interactively and customise them easily.

JavaBeans - Introduction

- JavaBeans technology is appropriate for software components that can be visually manipulated and customised.
- Many beans, including the button, slide, and graph beans, are essentially graphic components.

JavaBeans - Introduction

- Java Beans, however, are not limited to visual objects.
- A timer bean, for instance, might fire time events at specific intervals, without any kind of visible interface.
- Even if a bean has no GUI appearance at run time, it can be visually manipulated in a builder tool by having an associated icon.

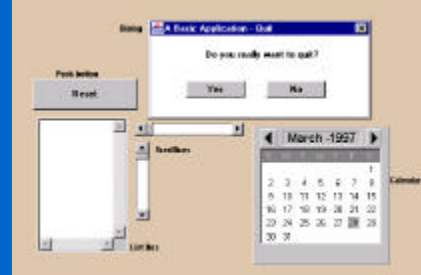
JavaBeans - Basics

- Beans can vary in size and complexity from simple GUI elements, such as buttons, to full-scale applications, such as database editors.
- Beans can act as containers for other beans.
- JavaBeans builds on the strengths of the component assembly development model.
- A component model is an architecture and set of APIs that allow you to define software components that can be combined together to create an application.

JavaBeans - Basics

- Software components can be simple like push buttons, text fields and list boxes, or more complex, like calendars and spreadsheets.
- Components can also work behind the scenes without presenting a visual interface.

JavaBeans - Basics



JavaBeans - Basics

- All components need containers in order to function.
- The **Application Builder** is the container for JavaBeans components.
- It supplies the "glue" in the form of code that holds the components together.
- The Application Builder also provides a visual interface for working with JavaBeans components.

JavaBeans - Basics

- All component architectures must solve the problem of how to make components interact, regardless of who wrote them or when they were written.
- Components must somehow reveal their interfaces so that they can be driven by calls from other components or application scripts.

JavaBeans - Basics

- Some component architectures have implemented registration mechanisms to meet this requirement.
- Java, on the other hand, has a built-in class discovery mechanism that allows objects to interact with each other dynamically.
- The Application Builder that connects JavaBeans components utilises **reflection** and **introspection** technology to find out about the internals of components.

JavaBeans - Services

- The Services provided by component models include:
 - Component Interface Exposure and Discovery,
 - Component Properties,
 - Event Handling,
 - Persistence,
 - Application Builder Support, and Component Packaging.

Component Interface Exposure and Discovery

- This is the mechanism for components to expose their interfaces so that they can be driven dynamically by calls and event notifications from other components or application scripts.
- Introspection is the mechanism through which Application Builders discover JavaBeans interfaces.

Component Interface Exposure and Discovery



Component Properties

- A component's properties are the public attributes of a component that either directly reflect or effect the current state of that component.
- For example, **the font typeface of a pushbutton is the property of that component.**
- The component model exposes these properties to a component's environment allowing the state of a component to be interrogated or modified through standard mechanisms.

Event Handling

- Event Handling is the mechanism for components to broadcast events and have those events delivered to the appropriate component or components that need or want to be notified.
- In other words, this mechanism defines how components interact with each other.

Application Builder Support

- Application Builder Support interfaces enable components to expose their properties and behaviors to Application Builder tools.
- Using these interfaces, tools can determine the properties and behaviors of arbitrary components.
- You can also modify the state and appearance of components as well as establish relationships between components by using tool palettes, inspectors, and editors.

Component Packaging

- Component packaging is the technology used to save the constituent resources of components, so that these components may be reassembled and redistributed over a network.
- The Java JAR file format combines and compresses class files and other component resources, such as sounds, images, and help files, into a single physical entity for distribution.

Beans and Class Libraries

- Programmers can extend the functionality of class libraries by subclassing and overriding methods.
- Class libraries may provide more than one way to solve a particular problem.
- Beans, on the other hand, perform specific, well-defined tasks.
- While you can customise the appearance of beans, you can not extend or change their functionality.

Beans and Class Libraries

- Creating an application with class libraries is like building a custom-made home, whereas creating an application with beans is like building a modular structure from prefabricated blocks.
- Some types of buildings and applications are suitable for modular assembly, and some are not.

Beans, Applets and Applications

- Unlike Java applets, Java Beans do not run inside a browser.
- Nor are they stand-alone executables, like applications.
- Rather, beans are the components that make up applets or applications.

Beans, Applets and Applications

- After the designer finishes customising and visually manipulating the beans, the Application Builder generates the class files and JAR file used to run the application or applet.

Component-ware -comments

- A component model can be described as a set of APIs and an architecture which allow developers to define and create software components and applications.
- These components can range from small GUI widgets to large applications such as a fully functional word processor.
- All of these components have the ability to be combined together dynamically.

Componentware - Summary

- Components models also provide many different services and mechanisms by which the components inter-operate.
- These services include self-discovery, properties, customised and event control.
- Have not discussed COM from Microsoft - similar to CORBA - but currently only available on Win32 platforms.
- Interoperable componentware seems to be the future of distributed computing.

RDG Interface Compiler

- A software description that is part way between RPC and Corba
 - Defines the interfaces of different functions that can be called between systems
 - Provides multiple methods for distributing data between parallel processors
 - Strongly typed
 - Requires strict design even down to communication structure.

RDG Interface Compiler

- RDG IC was designed to link closely coupled MPPs, or clusters of workstations
 - Array Processors
 - AMT DAP
 - Distributed Memory Multiprocessors
 - Meiko CS1 and CS2
- Supported heterogeneous languages
 - Array Fortran (DAP Fortran), f77 and C.

RDG Interface Compiler

```
• interface comp_chem
• message initmsg {
•   int nprocs;
•   int iterations;
•   int particles["I"];
•   double f_avg["f"];
• }
• message results {
•   integer who; /* used to check receiver id for faster sorting */
•   int result;
• }
• message timevalues {
•   real ctime;
•   double wtime;
• }
• message updates {
•   int size;
•   int delta_particles["E"];
•   int particles;
• }
• group work is 1 in N pems_k
•   >> initmsg;
•   >> results, timevalues;
•   pass_k;
• group workers is m in N pems_k;
•   >> updates;
• end comp_chem
```

RDG Interface Compiler

- Produced either:
 - Custom communications code
 - used for high performance on/off the MPPs
 - PVM code
 - MPI code
 - ELWMCP code
 - Experimental LightWeight MultiCast Protocol

RDG Interface Compiler

- For example:
- message **initmsg** {
- **int** runs;
- **int** loops;
- **double** indata [1000];
- **double** indata2 [*];
- **double** results [10][10],average;
- }

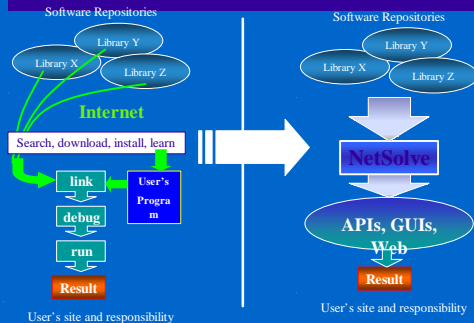
RDG Interface Compiler

```
• int pack_initmsg (mptr)
• initmsg = *mptr;
• {
•   int err=0;
•   err += pvm_pkint(&(mptr->runs), 1, 1);
•   err += pvm_pkint(&(mptr->loops), 1, 1);
•   err += pvm_pkdouble(mptr->indata, 1000, 1);
•   err += pvm_pkint(&(mptr->indata2_size), 1, 1);
•   if (mptr->indata2_size <= 0)
•     fprintf(stderr, "Warning: indata2 of initmsg_1 has not been packed as its
•     size is %d in %s\n", mptr->indata2_size);
•   else
•     err += pvm_pkdouble(mptr->indata2, mptr->indata2_size, 1);
•   err += pvm_pkdouble(mptr->results, 100, 1);
•   err += pvm_pkdouble(&(mptr->average), 1, 1);
•   return(err);
• }
```

Problem Solving Environments PSEs

- May use any of the technologies already discussed
- Provide a problem specific solution
 - I.e. an interface that suites a problem area very well, rather than a general API that may solve all most problems to some extent.
- Examples: Netsolve (UTK/UCSD) and ninf(Japan).

Motivation for NetSolve



NetSolve - Some Basic Concepts

- Client-server **agent-based** global computing
- Multiple and simple client interfaces
- **Load Balancing** and basic **Fault Tolerance**
- Heterogeneity
- Non-hierarchical software architecture

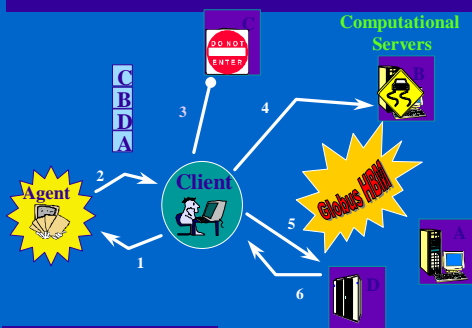
<http://www.cs.utk.edu/netsolve>

NetSolve - The big picture

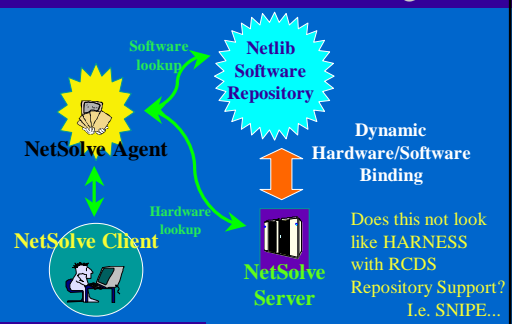


<http://www.cs.utk.edu/netsolve>

NetSolve - Failure Detection & Recovery



NetSolve-Future Paradigm



PSE / Computational Servers

- Performance is an issue...
 - People like better performance and if the end code/server is really just an MPI/PVM job then we do CARE how it is **scheduled**.