GridForce

(**Grid Fo**r **R**esearch **C**ollaboration and **E**ducation)

(Supported by : NSF CCLI A&I DUE -0311473)

Dr. Bina Ramamurthy

CSE486/586
Distributed Systems
Fall 2003

Bina Ramamurthy
127 Bell Hall, Tel: 645-3180, Ext.108
Office Hours: TTh 9.00 - 10.30am
Username: bina@cse.buffalo.edu
URL: http://www.cse.buffalo.edu/gridforce/index.htm

Course Objective

This course will address some of the fundamental challenges in the design, implementation and deployment of large scale distributed systems including connection establishment, event handling, interprocess communication, storage management, static and dynamic component configuration, concurrency and synchronization. It will also cover issues related to distributed objects such as mobility, security, naming and location. Possible solutions will be analyzed at various levels of granularity using objects, processes, services, components and frameworks. This course will focus on practical solutions over theoretical formalisms and server-side and middle-ware technology over client-side. *Special attention will be given the emerging technology of grid computing. The grid computing labs are supported by a grant from National Science Foundation DUE 0311473* Concepts studied will be applied to solve problems in various domains such as wireless world, embedded systems, electronic marketplace and application servers. Students will work in orchestrated groups of two with well-defined responsibilities on middleware-based projects. There will be a mid-semester assessment and a final exam.

On completion of this course, a student will be able to design and implement a distributed application, and will be able to analyze a distributed system for its architecture, algorithms, protocols and services. Students will have good undertanding and working knowledge of grid computing.

Class Meetings

```
MWF 1.00-1.50PM   Capen 260

Recitations will be held at the times indicated in the course catalog.
```

Required Textbooks

Distributed Systems: Concepts and Design, by G. Coulouris, J. Dollimore and T. Kindberg, Third Edition, Addison-Wesley Inc., 2001.

Recommended optional supplements

1. Grid Computing: Making Global Infrastructure a Reality Editors: Fran Berman, Jeffrey Fox and Tony Hey, Jojn Wiley and Sons, April 2003.
2. *Fundamentals of Distributed Object Systems: The CORBA Perspective*, by Z. Tari, O. Burkres, Wiley Inter-Science Pub., 2001.

3. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects: Volume 2* by Schmidt, Stal, Rohnert, and Buschmann, John-Wiley and Sons, 2000.
4. *Java: How to Program*, by Deitel and Deitel, Prentice-Hall Inc., 2000.

Prerequisites

1. CSE505/CSE305 or equivalent; Good foundation in problem solving, design representation, and object-oriented design methodology and application. Passion for design and development in Java.
2. Working knowledge of C++ and Java programming languages.
3. As Seniors and Graduates participating in a 500 level Computer Science and Engineering class it is assumed that you are capable of learning new programming languages and libraries in minimal amounts of time. You should also be familiar with object-oriented modeling, modern code design and debugging practices.

Grading

Midterm Exams (2 X 100)	200 points
Final Exam	200 points
Projects (3)	300 points

Final letter grades will be based on the (combined) overall percentage of all the items listed above. A (95 –), A- (90 – 94), B+ (85 – 89), B (80 – 84), B- (75 – 79), C+ (70 – 74), C (65 – 69), C- (60 – 64), D+ (55 – 59), D (50 – 54), F (less than 50). This policy is subject to change. If needed, the individual components and the overall grades will be appropriately curved. **In order to pass the course you must have a passing grade in every component of the course listed above.**

Exams

Two midterm exams, one of the exams will be held before the last date to **R**esign from the course. Final exam will be held during the regularly scheduled final exam week. Midterm exam will cover approximately 25% of the material and the final exam will be comprehensive. No make up exam will be given unless otherwise there is an extraordinary reason.

Projects

The due date for each project will be announced when it is assigned. All the source code, documentation, makefile, data files, and README files are to be submitted on-line. The details of how to submit given along with your first project. You will have to follow the rules for the other projects too. Projects will be graded for 100 points each and the total project points inclduing any bonuses will be scaled to 300 points as indicated in the grading policy.

I reserve the right to change the project specifications at any point before the due date to answer the problems that may arise during the course of the project. If your design is modular the changes will not be difficult to implement. A detailed grading guideline will be given to you along with the project specification. Use this as a

guide for your design and implementation. It is absolutely necessary to keep up with the programming projects in the class. There will be a 25 point deduction for each day the project is late after the due date.

Incomplete Policy

Incompletes will not given in this course, unless under the most dire circumstances. By definition, an incomplete is warranted if the student is capable of completing the course satisfactorily, but some traumatic event has interfered with his/her capability to finish within the timeframe of the semester. Incompletes are not designed as stalling tactic to defer a poor performance in a class.

Academic Dishonesty

You are required to work on your own unless the project specification clearly states that it is a group project with responsibility for each member of the group. Students who collaborate on homework, projects and/or the exams will be penalized with an 'F' for the course. CSE department has a strict policy on Academic Dishonesty. This will be strictly followed in this course. See: http://www.cse.buffalo.edu/undergrad/academicintegrity.html

There is very fine line between conversation between your peers about the concepts in the course and academic dishonesty. You are allowed to converse about the general concepts, but in no way are you allowed to share code or one person do the work for others. Remember that items taken from the web are also covered by the academic dishonesty policy. Also projects from this class cannot be used as projects for other courses and vice versa (projects from other courses cannot be passed in as work for this class).

Attendance and Participation

It is very important that you attend all the lectures and the recitation. You are strongly encouraged to participate in the lecture by asking relevant questions and taking part in useful discussion. This helps break the monotony of the lecture format. But if I find a discussion digressing from the topic of the lecture I may defer the discussion to after the regular lecture period or to the newsgroup meant for this class. There is newsgroup for the class: sunyab.cse.486. Understand that the newsgroup is a public forum and try to be civil and discuss only class related matter with professional courtesy.

FINAL EXAM – DISTRIBUTED SYSTEMS CSE486/586
December 12, 2003 (Fall 2003)

NAME : _____

STUDENT NUMBER : _____-_____
INSTRUCTIONS

This is a closed book but you are allowed two sheets of information to refer to. You have 180 minutes to complete 10 questions. *Please write neatly and clearly.* To receive partial credit, you must show all work for your answers. You should have 11 pages in this exam book, some of which are blank to allow room for your answers.

Question	Grade
I	____/20
II	____/20
III.1	
2.	
3.	
4.	
5.	
6.	
7.	
8.	
9.	
10.	
11.	
12.	
Total	____/100

I. (20 points) Design of a Grid Service

A business process is a fundamental component of a business system. In order for a grid service to be used to support business applications, it should be possible to model a business process as a grid service. Show the feasibility of this claim.

Hints:

- Examples for business process: Inventory control, Order Management, and Billing.
- Define a business process, list its requirements.
- Then identify the grid services capabilities that will satisfy the requirements. The capabilities of a grid service discussed in the GT3 tutorial will help.
- Bring all these together as an application implementing the business process.

II. (20 points) Design and Implementation of grid-based application

An e-commerce site is a very common application used in discussing large scale multi-tier distributed systems. An example of an e-commerce site is amazon.com. Explain with diagrams, the requirements of such an application and how it can be implemented using specific features of the grid computing framework.

Hint: You may have to use VO, virtualization and other such system level concepts.

III. Answer the following questions using few sentences. Assume Grid computing context for all the questions. Each question is worth 5 points. A good answer would have a simple explanation, an example and a diagram.

1. What is meant by virtualization?
2. What is a virtual organization (VO)?
3. What is federation of information?
4. What are the two approaches to designing a grid service?
5. Describe a Grid Service-based Application model. Use a block diagram.
6. What is the difference between transient and persistent services?
7. What is a portType?
8. What is a service EndPoint?
9. What is a service data? How can it be used by applications?
10. What is Notification? How can be used by applications?
11. What is a (i) Facory and (ii) Registry? How are they related?
12. What is a service handle, service reference and a handleMap? How are they related?

DRAFT - Nov 16, 2003

Course Evaluation

CSE 486: Distributed Systems

This course evaluation is part of an effort to evaluate the courses that are being developed as part of a grant from the National Science Foundation. Your participation in this course evaluation will provide important information to help improve the course. In addition, your comments will benefit students taking this course in the future.

We appreciate your taking the time to read each question carefully and answer them as fully as possible.

Instructions for Completing the Course Evaluation

- Do not put your name on any form. Survey responses will remain anonymous.
- Please respond to items 1–35 on this survey by circling the appropriate number. Responses to items 36–39 should be reported in the spaces provided.
- When you have completed the survey, please place the forms in the envelope supplied by your instructor.

Course Evaluation Student Questionnaire
CSE 486 — Distributed Systems I — Fall, 2003

Please respond to of the following questions by circling the number between one and five which most nearly represents your feelings. As indicated below, we use the scale: (1) Strongly Agree, (2) Agree, (3) Neutral, (4) Disagree, (5) Strongly Disagree. **Please read each question carefully**.

Course Information Please indicate the degree to which you feel	**Strongly Agree**	**Agree**	**Neutral**	**Disagree**	**Strongly Disagree**
1. the objectives of this course were clearly stated.	1	2	3	4	5
2. this course increased your interest in distributed systems.	1	2	3	4	5
3. this course increased your interest in grid computing.	1	2	3	4	5
4. you learned a lot about distributed systems, including both concepts and implementation.	1	2	3	4	5
5. you learned a lot about grid computing and its future potential.	1	2	3	4	5
6. adequate time was allotted to cover the course content.	1	2	3	4	5
7. the topic areas were sequenced in an appropriate manner.	1	2	3	4	5
8. the instructions for exercises and assignments were clear and easy to understand.	1	2	3	4	5
9. the lab exercises and assignments reflected the content of the course.	1	2	3	4	5
10. the lab exercises and assignments helped you learn the course material.	1	2	3	4	5
11. the grading of the lab exercises and assignments was fair.	1	2	3	4	5
12. the questions on tests reflected the content of the course.	1	2	3	4	5
13. the grading of the tests was fair.	1	2	3	4	5
14. adequate time was given to complete the tests.	1	2	3	4	5
15. the textbook was helpful and a good information resource.	1	2	3	4	5
16. the textbook, course materials and handouts were sufficient for you to understand all the topics covered.	1	2	3	4	5
17. the course website was useful for obtaining course materials and information.	1	2	3	4	5
18. the instructor or TA provided help when you needed it.	1	2	3	4	5
19. you are prepared for an advanced course on distributed systems.	1	2	3	4	5
20. the topics covered will be useful to you in the future, beyond CSE 486-586.	1	2	3	4	5
21. the course met your expectations.	1	2	3	4	5
22. Overall, how would you rate this course? (1=excellent, 2= good, 3=average, 4=poor, 5=bad)	1	2	3	4	5

Course Objectives Please indicate the degree to which you feel you	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
23. understand the fundamental components and operation of a distributed system.	1	2	3	4	5
24. can design and implement a distributed application.	1	2	3	4	5
25. are able to analyze a distributed system for its architecture, algorithms, protocols and services.	1	2	3	4	5
26. have good understanding and working knowledge of grid computing.	1	2	3	4	5
27. are able to program using Web services.	1	2	3	4	5
28. are able to program using the Globus grid computing framework.	1	2	3	4	5
29. are able to demonstrate the ability to design, implement, and deploy distributed systems based on Java technology and Grid Technology.	1	2	3	4	5

Computer Resources (Hardware and Software) Please indicate the degree to which you feel	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
30. the type of hardware computer resources provided by UB were appropriate for the course.	1	2	3	4	5
31. the type of software computer resources provided by UB were appropriate for the course.	1	2	3	4	5
32. the computer resources provided by UB were adequate to do the lab exercises and assignments.	1	2	3	4	5
33. the computer resources were available and accessible when you needed or wanted to use them.	1	2	3	4	5
34. the computer resources enabled you to gain "hands on" experience with distributed systems.	1	2	3	4	5
35. the computer resources enabled you to gain "hands on" experience with grid computing.	1	2	3	4	5

Please take the time to answer each of the following questions.

36. Why did you take this course?

37. What was the most valuable aspect of the Distributed Systems course? What did you like about it?

38. What was the poorest aspect of the course? In what ways could this course be improved?

39. What other comments would you like to make regarding any aspect of this course?

System Models and Networking Chapter 2,3

Bina Ramamurthy

Fundamental Issues

- There is no global time.
- All communications are by means of messages.
- Message communication may be affected by network delays and can suffer from a variety of failures and security attacks.
- How does one express a solution/process for handling an issue? One of the ways is to establish a model.

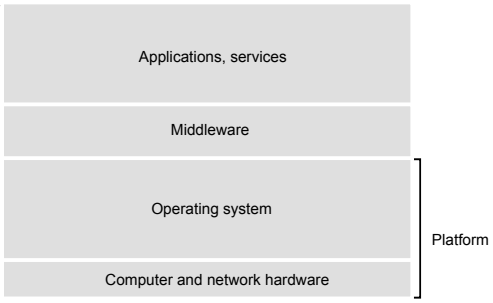
System Models

- **Interaction model** deals with performance and setting time limits in a distributed system, say, for message delivery.
- **Failure model** gives specification of faults and defines reliable communication and correct processes.
- **Security model** specifies possible threats and defines the concept of secure channels.
- **Architectural model** defines the way in which the components of the system interact with one another and the way in which they are mapped onto the underlying network of computers.

Architectural Model

- Abstracts the functions of the individual components.
- Defines patterns for distribution of data and workload.
- Defines patterns of communication among the components.
- Example: Definition of server process, client process and peer process and protocols for communication among processes; definition client/server model and its variations.

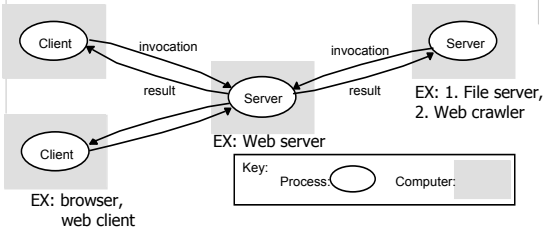
Software and hardware service layers in distributed systems



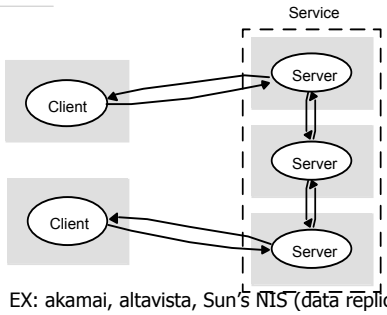
Middleware

- Layer of software whose purpose is to mask the heterogeneity and to provide a convenient programming model for application programmers.
- Middleware supports such abstractions as remote method invocation, group communications, event notification, replication of shared data, real-time data streaming.
- Examples: CORBA spec by OMG, Java RMI, MS's DCOM.

Clients invoke individual servers

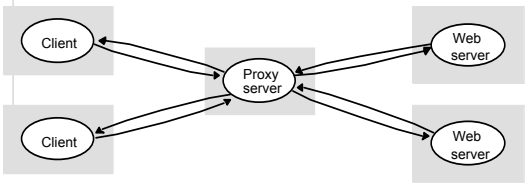


A service provided by multiple servers



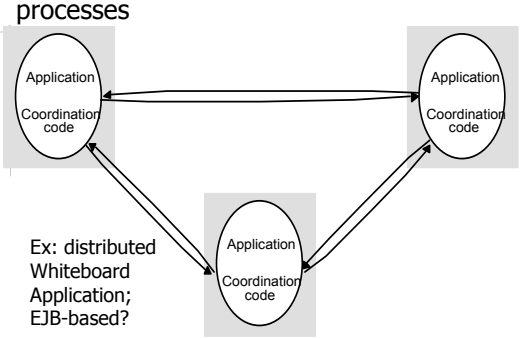
EX: akamai, altavista, Sun's NIS (data replication)

Web proxy server and caches



Proxy servers + cache are used to provide increased Availability and performance. They also play a major role Firewall based security. http://www.interhack.net/pubs/fwfaq/

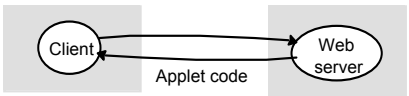
A distributed application based on peer processes



Ex: distributed Whiteboard Application; EJB-based?

Web applets

a) client request results in the downloading of applet code



b) client interacts with the applet

EX: Look at Object by value in CORBA

Networking (Chapter 3)

- Distributed systems use local area networks, wide area networks and internet for communication.
- Performance, reliability, scalability, mobility, and quality of service (qos) impact the design.
- Changes in user requirements have resulted in emergence of wireless and qos guarantees.
- Principles: protocol layering, packet switching, routing, data and behavior streaming.
- Coverage: Ethernet, Asynchronous Transfer Mode (ATM), IEEE 802.11 wireless network standard.

Networking Issues

- Performance:
  - Latency: delays at the switches and routers.
  - Data transfer rate (bits/sec) : raw data
  - Bandwidth: total volume of traffic that can be transferred across the network in a given time.
- Scalability:
  - How does a system handle increase in the number of users? Increase in the size of the system? Increase in load and traffic?

Networking Issues (contd.)

- Security: requirements and techniques for achieving security. Firewall, Virtual Private Network (VPN).
- Mobility: Support for moving devices. Not necessarily wireless.
- QoS: Bandwidth and latency bounds.

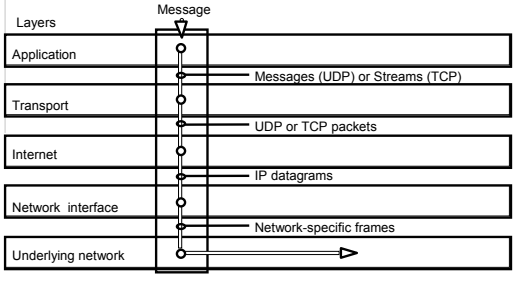
Types of Networks

- Characterized by speed, communication medium, size, geographical distances, bandwidth, latency, technology.
- LAN :
  - Single medium such as twisted pair of copper wires, coaxial cables, or optical fibers.
  - Technology: Ethernet, token rings, slotted rings.
- WAN:
  - Set of comm circuits (coax, satellite) linked by dedicated computers called routers.
  - Technology: Switching.

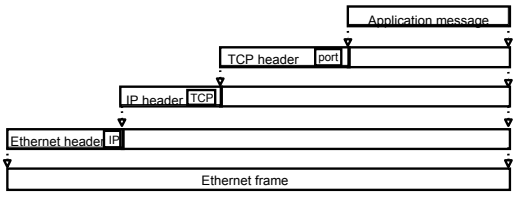
Types of Networks

- MAN:
  - High bandwidth copper or fiber optic cables. (phone lines, DSL, cable modem)
  - Technology: Ethernet, IEEE802.6, ATM
- Wireless:
  - Radio frequency, infrared,
  - Technology: IEEE 802.11 (wavelan), CDPD, GSM, bluetooth (proximity)

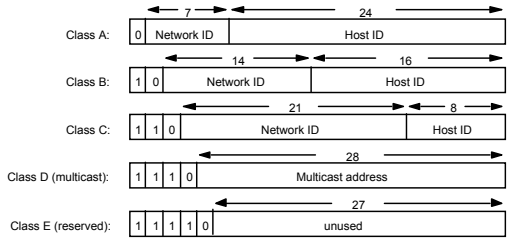
TCP/IP layers



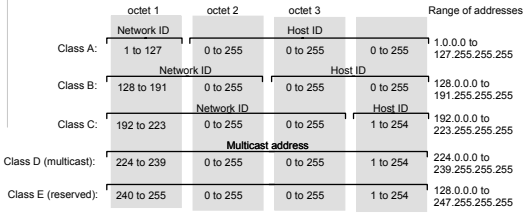
Encapsulation in a message transmitted via TCP over an Ethernet



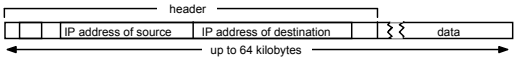
Internet address structure, showing field sizes in bits



Decimal representation of Internet addresses



IP packet layout and IPV4 Issues



- Address limitations
- Scarcity of Class B addresses
- Managing entries in routing tables
- Ad hoc measures such as allocation Class C to Class B address ranges (CIDR – classless interdomain routing).

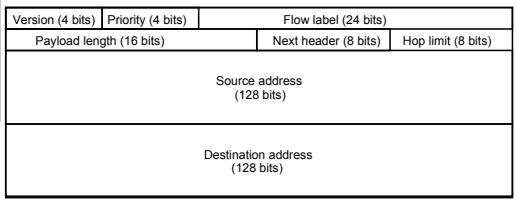
Issues in IPV4

- Address limitations
- Scarcity of Class B addresses
- Managing entries in routing tables
- Ad hoc measures such as allocation Class C to Class B address ranges (CIDR – classless interdomain routing).

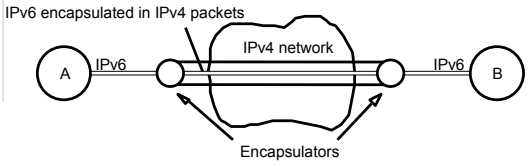
IPV6 Features

- Addresses are 128 bits (double that of IPV4)
- Address space is partitioned
- Routing speed improved by removing some operations such as checksum.
- Accommodates real-time and special services. (streams and devices)
- Future evolution possible (next header field).
- IPV6 support "anycast" (message delivered to at least one of the hosts).
- Built-in security.

IPv6 header layout



Tunnelling for IPv6 migration



Introduction to Web Services

Bina Ramamurthy

Literature Surveyed

- IBM's alphaworks site:
http://www-106.ibm.com/developerworks/webservices/

http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf

Web Services

- *Web Services* is a technology that allows for applications to communicate with each other in a standard format.
- A *Web Service* exposes an interface that can be accessed through XML messaging.
- A Web service uses XML based protocol to describe an operation or the data exchange with another web service. Ex: SOAP
- A group of web services collaborating accomplish the tasks of an application. The architecture of such an application is called Service-Oriented Architecture (SOA).

Web Services Suite of Protocols

- A suite of protocols define the Web Services Technology.
- These are used to describe, publish, discover, deliver and interact with services.
- The information about the protocols is from IBM's developerworks.

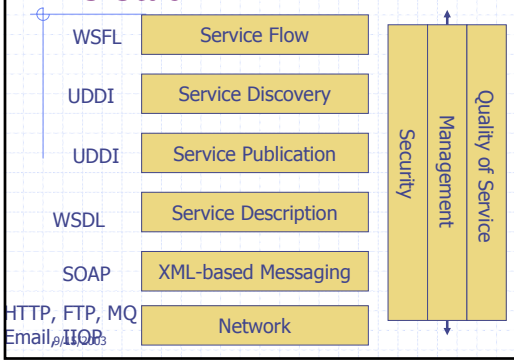
WS Suite of Protocols

- Messaging protocol Simple Object Access Protocol (SOAP) encodes messages so that they can be delivered over the transport protocols HTTP, SMTP or IIOP.
- Web Services Definition Language (WSDL) is used to specify the service details such as name, methods and their parameters, and the location of the service. This facilitates the registering and discovery of the service.
- For services to locate each other, the Universal Description, Discovery and Integration (UDDI) protocol defines a registry and associated protocols for locating and accessing services.

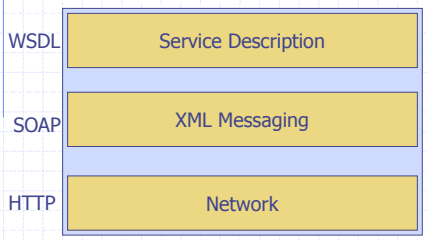
WS Suite of Protocols (contd.)

- The WS-Transaction and WS-Coordination protocols work together to handle distributed transactions.
- The Business Process Execution Language for Web Services (BPEL4WS) defines workflow operations.
- WS-Security is a family of protocols that cover authentication, authorization, federated security, secure communications, and delivery.
- WS-Policy is another group of protocols that define the policy rules behind how and when Web services can interact with each other.
- WS-Trust defines how trust models work between different services.
- These protocols are for e-business. Are there any available for e-science?

WS Stack



WS Interoperability Infrastructure

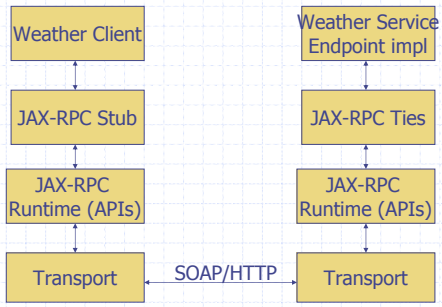


Do you see any platform or language dependencies here?

JAX-RPC

- **JAX-RPC: Java API for XML-based Remote Procedure Call (RPC).**
- **An API for building Web Services and Web Services clients.**
- **Some important concepts in JAX-RPC are:**
  - **Type-mapping system (WSDL to Java)**
  - **Service endpoint**
  - **Exception handling**
  - **Service endpoint context**
  - **Message handlers**
  - **Service clients and service context**
  - **SOAP with attachments**
  - **Runtime services**
  - **JAX-RPC client invocation models**

Application Architecture



Approaches to Web Service Implementation

- Top down: Start with WSDL and map onto Java
- Bottom up: Start with Java and end up all the supporting classes needed.
- We used the second approach for our RMI example.

WS Development Lifecycle

- Build:
  - Definition of service interface
  - Definition of service implementation
    - New services
    - Existing application into WS
    - Composing a WS out of other WS and applications
  - Source compiled and Stubs and Ties are generated.
- Deploy:
  - Publication of the service interface and service implementation to service registry or service requestor.
  - Deployment of executables in an execution environment.

WS Development Lifecycle (contd.)

- Run: A WS is available for invocation. Requestor can perform find and bind operation.
- Manage: on going management and administration for security, availability, performance, QoS, and business processes.

A Simple Example from Sun Microsystem

- HelloWorld distributed application:
- Files of interest:
  - **HelloIF.java: service definition interface**
  - **HelloImpl.java: Service definition implmentation.**
  - **HelloClient.java: remote client to invoke the service.**
  - **config-interface.xml: configuration file used by wscompile**
  - **jaxrpc-ri.xml: a configuration file used by wsdeploy**
  - **web.xml: a deployment descriptor for the web component that dispatches to the service.**
  - **build.xml for running the various steps such as compile, wscompile, deploy etc. Used by the ant tool.**
  - **build.properties: contains the details of varuious context roots or paths.**

Building and Deploying the Service

- Code the service definition interface and implementation class.
- Compile the service definition code written above.
- Package the code in a WAR (web archive) file.
- Generate the ties and WSDL files.
- Deploy the service.

Coding the interface and implementation classes

- The interface extends java.rmi.Remote interface.
- No constant declarations allowed.
- Methods must throw java.rmi.RemoteException
- Method parameters and return types must be supported by JAX-RPC types.

Compiling and Packaging

- To compile:
  - ant compile-server
- To pacakge:
  - ant setup-web-inf
  - ant package
- These two commands will generate and place the executables in appropriate directories. (Details will be given to you later in another handout).

Generating Ties and WSDL file and deploy the service

- To generate Ties and WSDL:
  - ant process-war
  - Will invoke *wsdeploy* to generate the tie classes and the WDSL file MyHello.wsdl
- To deploy the service:
  - ant deploy
- To verify deployment:
  - http://localhost:8080/hello-jaxrpc/hello
  - The details of the web service will be displayed.
- To undeploy:
  - ant undeploy

Building and Running the client

- Generate the stubs.
- Code the client.
- Compile the client code.
- Package the client classes into a JAR file.
- Run the client.

Client steps

- To generate stubs:
  - ant generate-stubs
  - This will call *wscompile* to generate the stubs.
- Coding the client: is a stand alone program. It calls the service through the generated stub which acts as a proxy for the remote service.

Clients steps (contd.)

- Compile the client code:
  - ant compile-client
- Package the client:
  - ant jar-client
- Running the client:
  - Ant run

Iterative Development

- Test the application.
- Edit the source files.
- Execute *ant build* to create and deploy war files.
- Execute *ant redeploy* to undeploy and deploy the service.
- Execute *ant build-static* to create jar files with static stubs.
- Execute *ant run* to run the client.

Other features

- Read about the types supported by JAX-RPC
- An advanced feature of interest is the dynamic proxy.
- Read about the directory structure and paths.

Reading Material

- Introduction to Web Services Ch.1.
- Building Web Services with JAX-RPC Ch. 11.
- Ant build tool details.
- XML, XML Schema and SOAP1.1.

WebServices Using JAX-RPC

Based on the presentation in O'Rielly's Webservices in a NutShell by Kim Topley

JAX-RPC

- JAX-RPC (The Java API for XML-based RPC) is designed to provide a simple way for developers to create Web services server and Web services client.
- Based on remote procedure calls; so the programming model is familiar to Java developers who have used RMI or CORBA.
- Major difference between RMI and JAX-RPC is that messages exchanged are encoded in XML based protocol and can be carried over a variety of transport protocols such as HTTP, SMTP etc.
- You can use JAX-RPC without having to be an expert in XML, SOAP, or HTTP.

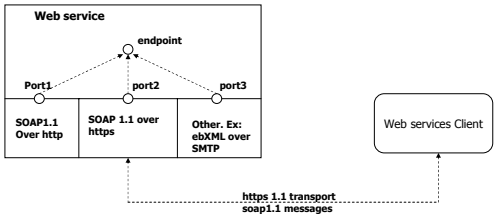
The JAX-RPC Programming Model

- Services, ports and bindings
- JAX-RPC web service servers and clients
- JAX-RPC service creation
- JAX-RPC client and server programming environments
- Stubs and ties
- Client invocation modes
- Static and dynamic stubs and invocation

Services, ports and bindings

- Service endpoint interface or service endpoint that defines one or more operations that the web service offers.
- Access to an endpoint is provided by binding it to a protocol stack through a port.
  - A port has an address that the client can use to communicate with the service and invoke its operations.
- An endpoint can be bound to different ports each offering a different suite of protocols for interaction.

Endpoint, Port and binding



Web Service Clients and Servers

- JAX-RPC maps a
  - web service operation to a java method call.
  - service endpoint to a Java Interface.
- Thus one way to begin implementation of a web service in JAX-RPC is to define a Java interface with a method for each operation of the service along with a class that implements the interface. Of course, following the rules of remote invocation etc.
- Now visualize client/server invocation in the same address space and lets compare it with remote invocation.

Local Date Service

```
//server
public class DataService {
    public Data getDate() {
        return new Date();}
//client
Public class Appln {
   public static void main (..) {
      DateService instance = new DateService();
       Date date = instance.getDate();
       System.out.println (" The date is" + date);
}
```

- In the case of the remote call a layer of software is used to convey the method call from client to server. This layer of software is provided by JAX-RPC runtime.

JAX-RPC service creation

- A service definition describes the operations that it provides and the data types that they require as argument and provide as return values.
- This definition can be made available as a document written in WSDL.
- From a WSDL document, JAX-RPC can generate the Java code required to connect a client to a server leaving one to write only the logic of the client application itself.
- Since WSDL is language independent the server can be in .net, Jax-rpc or any other compatible platform.

JAX-RPB service creation (contd.)

- Define the service a Java interface.
- Generate WSDL using the tools provided with JAX-RPC package.
- Advertise it in a registry for the client to lookup and import it.
- For publication and lookup any other technology such as J2EE can be used.

Client and Server Programming Environment

- JAX-RPC API is distributed over a set of packages:
  - javax.xml.rpc
  - javax.xml.rpc.encoding
  - javax.xml.rpc.handler
  - javax.xml.rpc.handler.soap
  - javax.xml.rpc.holders
  - javax.xml.rpc.server
  - javac.xml.rpc.soap

Stubs and Ties

- Client Side: Stub object has the same methods as the service implementation class.
  - Client application is linked with the stub.
  - When it invokes a method stub delegates the call to the JAX-RPC runtime so that appropriate SOAP message can be sent to the server.
  - On completion the result return back in the reverse path as above.
- Server side:
  - Message received must be converted into a method call on actual service implementation. This functionality is provided by another piece of glue called tie.
  - Tie extracts method name and parameter from SOAP message.
  - Tie also converts the result of the method call back into a response message to be returned to client JAX-RPC runtime.
- Developer need not write these classes (tie and stub) since JAX-RPC comes with tools to generate them.

Client Invocation Modes

- Synchronous request-response mode (tightly coupled).
- One-way RPC (loosely coupled): no value returned, no exception thrown, need to bypass stub layer, use Dynamic Invocation Interface (DII).

Client Invocation Modes

Distributed File Systems

B.Ramamurthy

Introduction

- Distributed file systems support the sharing of information in the form of files throughout the intranet.
- A distributed file system enables programs to store and access remote files exactly as they do on local ones, allowing users to access files from any computer on the intranet.
- Recent advances in higher bandwidth connectivity of switched local networks and disk organization have lead high performance and highly scalable file systems.

Storage systems and their properties

	Sharing	*Persis-tence*	*Distributed cache/replicas*	*Consistency maintenance*	*Example*
Main memory	×	×	×	1	RAM
File system	×	✓	×	1	UNIX file system
Distributed file system	✓	✓	✓	✓	Sun NFS
Web	✓	✓	✓	×	Web server
Distributed shared memory	✓	×	✓	✓	Ivy (Ch. 16)
Remote objects (RMI/ORB)	✓	×	×	1	CORBA
Persistent object store	✓	✓	×	1	CORBA Persistent Object Service
Persistent distributed object store	✓	✓	✓	✓	PerDiS, Khazana

File system modules

Directory module:	relates file names to file IDs
File module:	relates file IDs to particular files
Access control module:	checks permission for operation requested
File access module:	reads or writes file data or attributes
Block module:	accesses and allocates disk blocks
Device module:	disk I/O and buffering

File attribute record structure

File length
Creation timestamp
Read timestamp
Write timestamp
Attribute timestamp
Reference count
Owner
File type
Access control list (ACL)

UNIX file system operations

filedes = open(name, mode) *filedes = creat(name, mode)*	Opens an existing file with the given *name*. Creates a new file with the given *name*. Both operations deliver a file descriptor referencing the open file. The *mode* is *read*, *write* or both.
status = close(filedes)	Closes the open file *filedes*.
count = read(filedes, buffer, n) *count = write(filedes, buffer, n)*	Transfers *n* bytes from the file referenced by *filedes* to *buffer*. Transfers *n* bytes to the file referenced by *filedes* from buffer. Both operations deliver the number of bytes actually transferred and advance the read-write pointer.
pos = lseek(filedes, offset, whence)	Moves the read-write pointer to offset (relative or absolute, depending on *whence*).
status = unlink(name)	Removes the file *name* from the directory structure. If the file has no other names, it is deleted.
status = link(name1, name2)	Adds a new name (*name2*) for a file (*name1*).
status = stat(name, buffer)	Gets the file attributes for file *name* into *buffer*.

Distributed File System Requirements

- Many of the requirements of distributed services were lessons learned from distributed file service.
- First needs were: access transparency and location transparency.
- Later on, performance, scalability, concurrency control, fault tolerance and security requirements emerged and were met in the later phases of DFS development.

Transparency

- Access transparency: Client programs should be unaware of the the distribution of files.
- Location transparency: Client program should see a uniform namespace. Files should be able to be relocated without changing their path name.
- Mobility transparency: Neither client programs nor system admin program tables in the client nodes should be changed when files are moved either automatically or by the system admin.
- Performance transparency: Client programs should continue to perform well on load within a specified range.
- Scaling transparency: increase in size of storage and network size should be transparent.

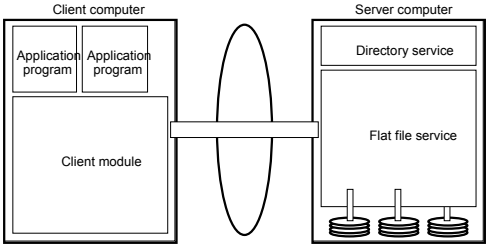
Other Requirements

- Concurrent file updates is protected (record locking).
- File replication to allow performance.
- Hardware and operating system heterogeneity.
- Fault tolerance
- Consistency : Unix uses on-copy update semantics. This may be difficult to achieve in DFS.
- Security
- Efficiency

General File Service Architecture

- The responsibilities of a DFS are typically distributed among three **modules:**
  - **Client module** which emulates the conventional file system interface
  - **Server modules**(2) which perform operations for clients on **directories** and on **files**.
- Most importantly this architecture enables stateless implementation of the server modules.

File service architecture



Flat file service Interface

Read(FileId, i, n) -> Data — throws *BadPosition*	If $1 \le i \le Length(File)$: Reads a sequence of up to *n* items from a file starting at item *i* and returns it in *Data*.
Write(FileId, i, Data) — throws *BadPosition*	If $1 \le i \le Length(File)+1$: Writes a sequence of *Data* to a file, starting at item *i*, extending the file if necessary.
Create() -> FileId	Creates a new file of length 0 and delivers a UFID for it.
Delete(FileId)	Removes the file from the file store.
GetAttributes(FileId) -> Attr	Returns the file attributes for the file.
SetAttributes(FileId, Attr)	Sets the file attributes (only those attributes that are not shaded in).

Primary operations are reading and writing.

Directory service Interface

Lookup(Dir, Name) -> FileId — throws *NotFound*	Locates the text name in the directory and returns the relevant UFID. If *Name* is not in the directory, throws an exception.
AddName(Dir, Name, File) — throws *NameDuplicate*	If *Name* is not in the directory, adds (*Name*, *File*) to the directory and updates the file's attribute record. If *Name* is already in the directory: throws an exception.
UnName(Dir, Name) — throws *NotFound*	If *Name* is in the directory: the entry containing *Name* is removed from the directory. If *Name* is not in the directory: throws an exception.
GetNames(Dir, Pattern) -> NameSeq	Returns all the text names in the directory that match the regular expression *Pattern*.

Primary purpose is to provide a service for translation text names to UFIDs.

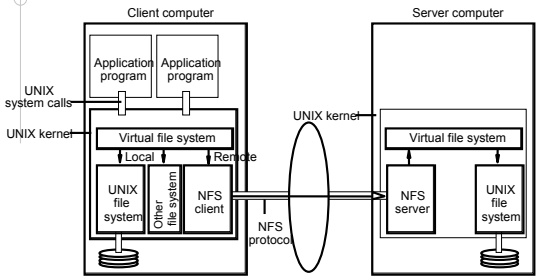
Case Studies in DFS

- We will look into architecture and operation of SUN's Network File System (NFS) and CMU's Andrew File System (AFS).

Network File System

- The Network File System (NFS) was developed to allow machines to mount a disk partition on a remote machine as if it were on a local hard drive. This allows for fast, seamless sharing of files across a network.

NFS architecture



NFS server operations (simplified) – 1

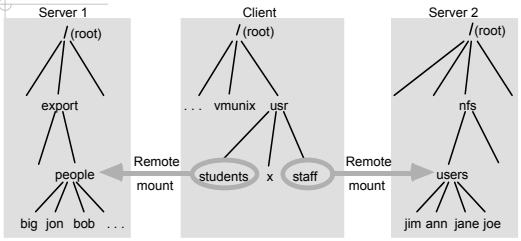
lookup(dirfh, name) -> fh, attr	Returns file handle and attributes for the file *name* in the directory *dirfh*.
create(dirfh, name, attr) -> newfh, attr	Creates a new file name in directory *dirfh* with attributes *attr* and returns the new file handle and attributes.
remove(dirfh, name) status	Removes file name from directory *dirfh*.
getattr(fh) -> attr	Returns file attributes of file *fh*. (Similar to the UNIX *stat* system call.)
setattr(fh, attr) -> attr	Sets the attributes (mode, user id, group id, size, access time and modify time of a file). Setting the size to 0 truncates the file.
read(fh, offset, count) -> attr, data	Returns up to *count* bytes of data from a file starting at *offset*. Also returns the latest attributes of the file.
write(fh, offset, count, data) -> attr	Writes *count* bytes of data to a file starting at *offset*. Returns the attributes of the file after the write has taken place.
rename(dirfh, name, todirfh, toname) -> status	Changes the name of file *name* in directory *dirfh* to *toname* in directory to *todirfh*
link(newdirfh, newname, dirfh, name) -> status	Creates an entry *newname* in the directory *newdirfh* which refers to file *name* in the directory *dirfh*.

Continues on next slide

NFS server operations (simplified) – 2

symlink(newdirfh, newname, string) -> status	Creates an entry *newname* in the directory *newdirfh* of type symbolic link with the value *string*. The server does not interpret the *string* but makes a symbolic link file to hold it.
readlink(fh) -> string	Returns the string that is associated with the symbolic link file identified by *fh*.
mkdir(dirfh, name, attr) -> newfh, attr	Creates a new directory *name* with attributes *attr* and returns the new file handle and attributes.
rmdir(dirfh, name) -> status	Removes the empty directory *name* from the parent directory *dirfh*. Fails if the directory is not empty.
readdir(dirfh, cookie, count) -> entries	Returns up to *count* bytes of directory entries from the directory *dirfh*. Each entry contains a file name, a file handle, and an opaque pointer to the next directory entry, called a *cookie*. The *cookie* is used in subsequent *readdir* calls to start reading from the following entry. If the value of *cookie* is 0, reads from the first entry in the directory.
statfs(fh) -> fsstats	Returns file system information (such as block size, number of free blocks and so on) for the file system containing a file *fh*.

Local and remote file systems accessible on an NFS client



Note: The file system mounted at */usr/students* in the client is actually the sub-tree located at */export/people* in Server the file system mounted at */usr/staff* in the client is actually the sub-tree located at */nfs/users* in Server 2.

NFS Revisited

- From A.Tannenbaum's text
- Three aspects of NFS are of interest: the architecture, the protocol, and the implementation.

NFS Architecture

- Allows an arbitrary collection of clients and servers to share a common file system.
- In many cases all servers and clients are on the same LAN but this is not required.
- NFS allows every machine to be a client and server at the same time.
- Each NFS server exports one or more directories for access by remote clients.
- See example enclosed.

NFS Protocol

- One of the goals o NFS is to support a heterogeneous system, with clients and servers running different operating systems on different hardware. It is essential the interface between clients and server be well defined.
- NFS accomplishes this goal by defining two client-server protocol: one for handling mounting and another for directory and file access.
- Protocol defines requests by clients and responses by servers.

Mounting

- Client requests a directory structure to be mounted, if the path is legal the server returns file handle to the client.
- Or the mounting can be automatic by placing the directories to mounted in the /etc/rc: automounting.

File Access

- NFS supports most unix operations except open and close. This is to satisfy the "statelessness" on the server end. Server need not keep a list of open connections. See the operations listed in slides 17, 18.
- (On the other hand consider your database connection… you create an object, connection is opened etc.)

Implementation

- After the usual system call layer, NFS specific layer Virtual File System (VFS) maintains an entry per file called vnode (virtual I-node) for every open file.
- Vnode indicate whether a file is local or remote.
  - For remote files extra info is provided.
  - For local file, file system and I-node are specified.
  - Lets see how to use v-nodes using a mount, open, read system calls from a client application.

Vnode use

- To mount a remote file system, the sys admin (or /etc/rc) calls the mount program specifying the remote directory, local directory in which to be mounted, and other info.
- If the remote directory exist and is available for mounting, mount system call is made.
- Kernel constructs vnode for the remote directory and asks the NFS-client code to create a r-node (remote I-node) in its internal tables. V-node in the client VFS will point to local I-node or this r-node.

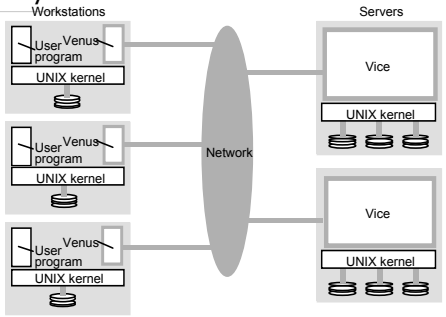
Remote File Access

- When a remote file is opened by the client, it locates the r-node.
- It then asks NFS Client to open the file. NFS file looks up the path in the remote file system and return the file handle to VFS tables.
- The caller (application) is given a file descriptor for the remote file. No table entries are made on the server side.
- Subsequent reads will invoke the remote file, and for efficiency sake the transfers are usually in large chunks (8K).

Server Side of File Access

- When the request message arrives at the NFS server, it is passed to the VFS layer where the file is probably identified to be a local or remote file.
- Usually a 8K chunk is returned. **Read ahead and caching** are used to improve efficiency.
- Cache: server side for disk accesses, client side for I-nodes and another for file data.
- Of course this leads to cache consistency and security problem which ties us into other topics we are discussing.

Distribution of processes in the Andrew File System



Summary

- Study Andrew Files System (AFS): how?
- Architecture
- APIs for operations
- Protocols for operations
- Implementation details

Name Services

Bina Ramamurthy

Chapter 9

Introduction

- You need to name an entity in order to use it.
- If you don't have a name or don't know a name you should be able to describe its characteristics in order to identify it.
- According to these two requirements we have two services:
  - Naming service
  - Directory service

Naming Service

- Given the name of a resource returns the information about the resource.
- For example consider the **white pages**: given the name of a person you get the address/telephone number of that person.
- Other examples: LDAP (Lightweight Directory Access Protocol) a person on UB computers gives you information about the person's email, campus address, phone number, position held etc.

Directory Service

- Given a description, find a service or resource that matches the description.
- For example consider the **yellow pages**: when you want to rent a car, it may give a list of car rental agencies.

Names, Attributes and Addresses

- Names: human readable names such as /etc/passwd, URLs such as http://www.cd3.net/
- An address is an attribute of a name. Ex: *castor* is the name, its address is 128.205.34.1
- Identifiers: refer to names that are interpreted by programs. Examples: remote object reference, NFS file handles.
- Name Resolution: a name is resolved when it is translated into data about the name's resource or object.

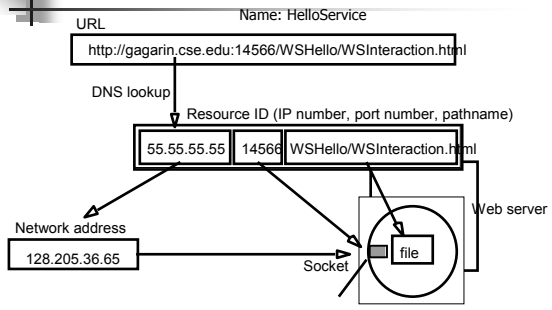
Names, Attributes and Addresses

- Binding: association between a name and an object. In general, names are bound to the attributes of the object rather than an implementation of an object.
- Attribute: Value or property of an object.
- Names and services: many of the names are specific to some services.
- URL: principle means of identifying web resources

Some Familiar Naming Services

- DNS: maps domain names to the attributes of a host computer.
- X500 maps person's name to personal information.
- CORBA naming service maps a name to remote object reference.

Example



Name Service

- A name service stores a collection of one or more naming contexts – set of bindings between names and attributes for objects such as users, computers, services and remote objects.
- Name Management is separated from other services because of the openness of the distributed system.
- Requirements:
  - Unification (EX: URLs, uniform names, UUID)
  - Integration: Share resources for resolving names.
  - Handle arbitrary number of names and domains
  - Long lifetime, High Availability, Fault isolation, Tolerance of mistrust

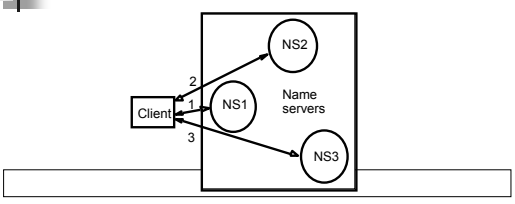
NameSpaces

- Namespace: is a collection of all valid names recognized by a particular service (context). Requires syntactic definition.
- Can be flat or hierarchical: Hierarchical is scalable and reusable and can be managed separately.
- May provide aliases for names.
- Can be broken down into domains.

Name Resolution

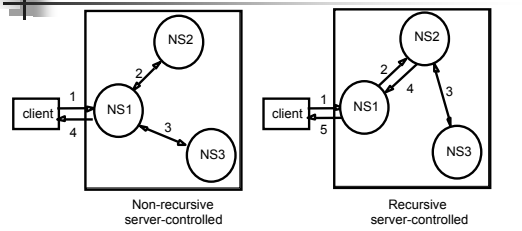
- Is a repetitive process in which a name is presented successively to naming contexts until its context is located or not locatable.
- When a context contains the name its attributes are returned.
- Navigation among the contexts can be iterative or recursive as shown in the next slides.

Figure 9.2 Name Resolution: Iterative navigation



A client iteratively contacts name servers NS1–NS3 in order to resolve a name

Figure 9.3 Non-recursive and recursive server-controlled navigation



A name server NS1 communicates with other name servers on behalf of a client

Navigation: Process of locating naming data from among more than one Name server in order to resolve a name. (iterative or multicast navigation)

Directory Services

- A more powerful service than naming where you look up for names using the attributes than the other way.
- Clients can Lookup for services by providing their attributes rather the name.
- A discovery service provides registry and lookup for spontaneous networking.
- Registry is used by server to publish a service and lookup is used by a client to locate a service.

Jini: A case study

- Jini (Jini Is Not Initials) is Java's solution to providing connectivity to services and devices.
- It is network-centric computing model as opposed to network-transparent model offered by CORBA and other earlier distributed system models. Software infrastructure that includes devices must be incredibly robust.
- The devices have to support true "plug and play".
- Devices and services should form spontaneous communities.

Jini and Name Servers

- Jini does serves the functionality of a name server. But it is much more than that.
- Jini differs from names servers such as LDAP (Light Weight Directory Access Protocol) or DNS (Domain Name Service) in two aspects:
  - Services can appear and disappear without much overhead. Interested parties can be notified when a service changes.
  - Jini is self-healing. It accepts partial failures and has mechanisms for taking care of this.

Five Key Concepts

1. Discovery
2. Lookup
3. Leasing
4. **Remote Events**
5. Transactions

Jini Services/Devices

- Service providers in Jini can be:

1. **Pure software component**
2. Pure hardware device
3. Combination of the two

For obvious reasons we will consider only software services.

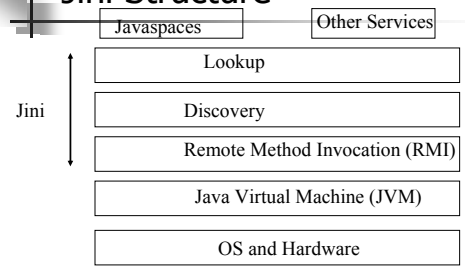
Discovery and Lookup

- Discovery is Jini service that finds communities on the network to join to form a spontaneous "federation".
- It basically searches and locates lookup services of communities.
- Lookup service has the details of services, their location, code, attributes etc.

Leasing

- Leasing is the technique that provides the self-healing characteristic of Jini.
- Every service provider keeps renewing its lease with the holder of the services (probably a lookup service) periodically. If it fails to update lease the service will be deleted from the community.
- This automatically removes failed or crashed server from the network thus carrying out the self-healing.

Jini Structure



Discovery

- To find and join a group of Jini services
- Sends out multicast packet and unicast packet
- Receives RMI reference to a lookup service where the requested service may be found.

Lookup

- Repository of available services.
- Stores proxy of object and its attributes.
- Proxy can be thin or fat.
- Lookup interface: Registration, Access, Search, Removal
- Note: Best way to study a service is through its interfaces.

Summary

- We studies the essential features of a Name Service.
- We also looked at some existing name servers.
- Jini extends the concepts of a simple name service to build a spontaneous networking distributed system model.
- Think about: How will you build a sophisticated name service using the common name service and the web services infrastructure?

Security

Chapter 7

Introduction

- There are two main issues:
  - Authentication
  - Authorization
- Authentication: is validating the user and the messages sent by by the authenticated user.
- Authorization: refers to access control of resources after a user/message has been authenticated.
- Security primarily refers to the authentication issue. This is discussed quite nicely in chapter 7 of your text.
- For access control models we will discuss Java Authentication and Authorization Service (JAAS).

Cryptography

- Cryptography is the basis for authentication of messages.
- We need security protocols to exploit it.
- Selection of cryptographic algorithms and management of keys are critical issues for effectiveness, performance and usefulness of security mechanisms.
- Public-key cryptography is good for key distribution but inadequate for encryption of bulk data.
- Secret-key cryptography is suitable for bulk encryption tasks.
- Hybrid protocols such as SSL (Secure Socket Layer) establish a secure channel using public-key cryptography and then use it exchange secret keys for subsequent data exchanges.

Historical context: the evolution of security needs

	1965-75	*1975-89*	*1990-99*	*Current*
Platforms	Multi-user timesharing computers	Distributed systems based on local networks	The Internet, wide-area services	The Internet + mobile devices
Shared resources	Memory, files	Local services (e.g. NFS), local network	Email, web sites, Internet commerce	Distributed objects, mobile code
Security requirements	User identification and authentication	Protection of service	Strong security for commercial transactions	Access control for individual objects, secure mobile code
Security management environment	Single authority, single authorization database (e.g. /etc/ passwd)	Single authority, delegation, repli-cated authorization databases (e.g. NIS)	Many authorities, no network-wide authorities	Per-activity authorities, groups with shared Responsibilities, **mass authentication**

Encryption

- Most schemes include algorithms for encrypting and decrypting messages based on secret codes called keys.
- Two common models:
  - Shared secret keys
  - Public/private key pairs: A message encrypted with the public key of the receiver can be decrypted only by the private key of the recipient.

Familiar names for the protagonists in security protocols

Alice	First participant
Bob	Second participant
Carol	Participant in three- and four-party protocols
Dave	Participant in four-party protocols
Eve	Eavesdropper
Mallory	Malicious attacker
Sara	A server

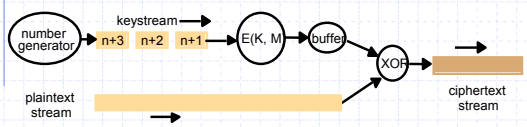
Cryptography notations

K_A	Alice's secret key
K_B	Bob's secret key
K_{AB}	Secret key shared between Alice and Bob
K_{Apriv}	Alice's private key (known only to Alice)
K_{Apub}	Alice's public key (published by Alice for all to read)
$\{M\}_K$	Message M encrypted with key K
$[M]_K$	Message M signed with key K

Cryptographic Algorithms

- Plain text Ą cipher text Ą Decipher text
- $E(K,M) = \{M\}_K$ where E is the encryption function, M is the message and K is the key.
- Decryption:
- $D(K,E(K,M)) = M$
- Same key is used in encrypting and decrypting. So it is called symmetric cryptography.

Stream cipher



Cryptographic algorithms

- Shannon's principles of cryptography: introduce "confusion" (XORing, bit shifting etc.) and "diffusion" (adding noise bits to diffuse the information)
- We will look at Tiny Encryption Algorithm (TEA) as an example of symmetric algorithm and Rivest, Shamir and Adelman (RSA) an an example for asymmetric algorithms.

TEA Encryption Function

```
void encrypt(unsigned long k[], unsigned long text[]) {
    unsigned long y = text[0], z = text[1];
    unsigned long delta = 0x9e3779b9, sum = 0; int n;
    for (n= 0; n < 32; n++) {
        sum += delta;
        y += ((z << 4) + k[0]) ^ (z+sum) ^ ((z >> 5) + k[1]);
        z += ((y << 4) + k[2]) ^ (y+sum) ^ ((y >> 5) + k[3]);
    }
    text[0] = y;  text[1] = z;      }
```


TEA decryption function

```
void decrypt(unsigned long k[], unsigned long text[]) {
    unsigned long y = text[0], z = text[1];
    unsigned long delta = 0x9e3779b9, sum = delta << 5;  int n;
    for (n= 0; n < 32; n++) {
        z -= ((y << 4) + k[2]) ^ (y + sum) ^ ((y >> 5) + k[3]);
        y -= ((z << 4) + k[0]) ^ (z + sum) ^ ((z >> 5) + k[1]);
        sum -= delta;
    }
    text[0] = y; text[1] = z;
}
```

TEA in use

```
void tea(char mode, FILE *infile, FILE *outfile, unsigned long k[]) {
/* mode is 'e' for encrypt, 'd' for decrypt, k[] is the key.*/
    char ch, Text[8]; int i;
    while(!feof(infile)) {
        i = fread(Text, 1, 8, infile);          /* read 8 bytes from infile into
Text */
        if (i <= 0) break;
        while (i < 8) { Text[i++] = ' ';}    /* pad last block with spaces */
        switch (mode) {
        case 'e':
            encrypt(k, (unsigned long*) Text); break;
        case 'd':
            decrypt(k, (unsigned long*) Text); break;
        }
        fwrite(Text, 1, 8, outfile);            /* write 8 bytes from Text to
outfile */
    }
}
```


RSA Encryption

To find a key pair e, d:

1. Choose two large prime numbers, P and Q (each greater than 10100), and form:
   $N = P \times Q$
   $Z = (P-1) \times (Q-1)$
2. For d choose any number that is relatively prime with Z (that is, such that d has no common factors with Z).
   We illustrate the computations involved using small integer values for P and Q:
   $P = 13$, $Q = 17$ –> $N = 221$, $Z = 192$
   $d = 5$
3. To find e solve the equation:
   $e \times d = 1 \bmod Z$

That is, $e \times d$ is the smallest element divisible by d in the series $Z+1$, $2Z+1$, $3Z+1$, ...
$e \times d = 1 \bmod 192 = 1, 193, 385, \ldots$
385 is divisible by d
$e = 385/5 = 77$

RSA Encryption (contd.)

To encrypt text using the RSA method, the plaintext is divided into equal blocks of length k bits where $2^k < N$ (that is, such that the numerical value of a block is always less than N; in practical applications, k is usually in the range 512 to 1024).
$k = 7$, since $2^7 = 128$

The function for encrypting a single block of plaintext M is: (N = P X Q = 13X17 = 221), e = 77, d = 5:
$E'(e,N,M) = M^e \bmod N$
for a message M, the ciphertext is $M'^{77} \bmod 221$

The function for decrypting a block of encrypted text c to produce the original plaintext block is:
$D'(d,N,c) = c^d \bmod N$

The two parameters e,N can be regarded as a key for the encryption function, and similarly d,N represent a key for the decryption function.

So we can write K_e = <e,N> and K_d = <d,N>, and we get the encryption function:
$E(K_e, M) = \{M\}_K$ (the notation here indicating that the encrypted message can be decrypted only by the holder of the private key K_d) and $D(K_d, \{M\}_K) = M$.

<e,N> - public key, d – private key for a station

Application of RSA

- Lets say a person in Atlanta wants to send a message M to a person in Buffalo:
- Atlanta encrypts message using Buffalo's public key B Ą E(M,B)
- Only Buffalo can read it using it private key b: E(p, E(M,B)) Ą M
- In other words for any public/private key pair determined as previously shown, the encrypting function holds two properties:
  - Ã E(p, E(M,P)) Ą M
  - Ã E(P, E(M,p)) Ą M

How can you authenticate "sender"?

- (In real life you will use signatures: the concept of signatures is introduced.)
- Instead of sending just a simple message, Atlanta will send a signed message signed by Atlanta's private key:
  - Ã E(B,E(M,a))
- Buffalo will first decrypt using its private key and use Atlanta's public key to decrypt the signed message:
  - Ã E(b, E(B,E(M,a)) Ą E(M,a)
  - Ã E(A,E(M,a)) Ą M

Digital Signatures

- Strong digital signatures are essential requirements of a secure system. These are needed to verify that a document is:
- Authentic : source
- Not forged : not fake
- Non-repudiable : The signer cannot credibly deny that the document was signed by them.

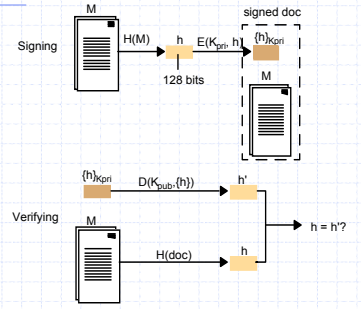
Digest Functions

- Are functions generated to serve a signatures. Also called secure hash functions.
- It is message dependent.
- Only the Digest is encrypted using the private key.

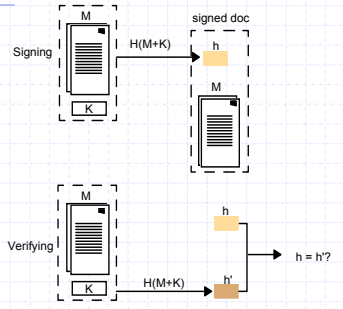
Alice's bank account certificate

1. *Certificate type*	Account number
2. *Name*	Alice
3. *Account*	6262626
4. *Certifying authority*	Bob's Bank
5. *Signature*	$\{Digest(field\ 2 + field\ 3)\}_{K_{Bpriv}}$

Digital signatures with public keys



Low-cost signatures with a shared secret key



X509 Certificate format

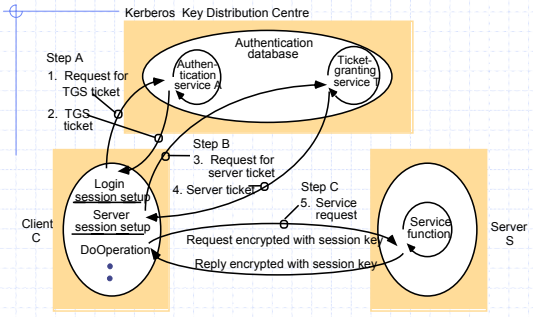
Subject	Distinguished Name, Public Key
Issuer	Distinguished Name, Signature
Period of validity	Not Before Date, Not After Date
Administrative information	Version, Serial Number
Extended Information	

Certificates are widely used in e-commerce to authenticate Subjects.
A Certificate Authority is a trusted third party, which certifies Public Key's do truly belong to their claimed owners.
Certificate Authorities: Verisign, CREN (Corp for Educational Research Networking), Thawte
See also Netscape SSL2.0 Certificate format:
http://wp.netscape.com/eng/security/ssl_2.0_certificate.html#SSL2cert

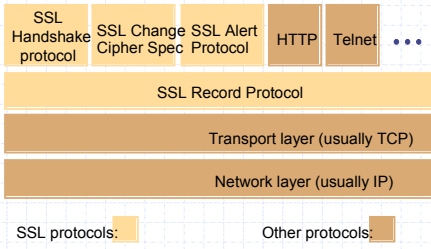
The Needham–Schroeder secret-key authentication protocol

Header	*Message*	*Notes*
1. A->S:	A, B, N_A	A requests S to supply a key for communication with B.
2. S->A:	$\{N_A, B, K_{AB}, \{K_{AB}, A\}K_B\}K_A$	S returns a message encrypted in A's secret key, containing a newly generated key K_{AB} and a 'ticket' encrypted in B's secret key. The nonce N_A demonstrates that the message was sent in response to the preceding one. A believes that S sent the message because only S knows A's secret key.
3. A->B:	$\{K_{AB}, A\}K_B$	A sends the 'ticket' to B.
4. B->A:	$\{N_B\}K_{AB}$	B decrypts the ticket and uses the new key K_{AB} to encrypt another nonce N_B.
5. A->B:	$\{N_B - 1\}K_{AB}$	A demonstrates to B that it was the sender of the previous message by returning an agreed transformation of N_B.

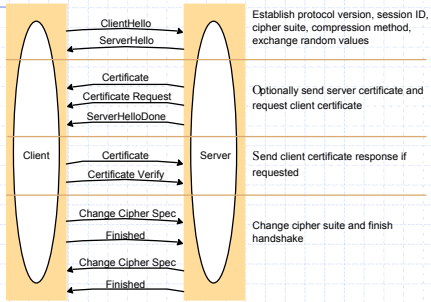
System architecture of Kerberos



SSL protocol stack



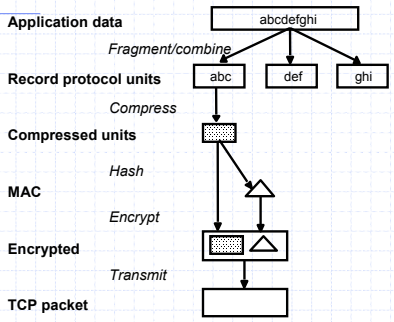
SSL handshake protocol



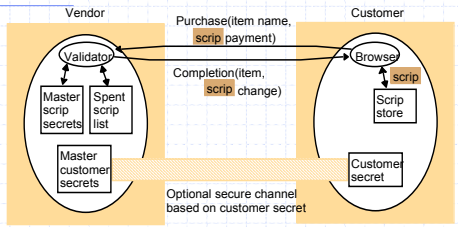
SSL handshake configuration options

Component	*Description*	*Example*
Key exchange method	the method to be used for exchange of a session key	RSA with public-key certificates
Cipher for data transfer	the block or stream cipher to be used for data	IDEA
Message digest function	for creating message authentication codes (MACs)	SHA

SSL record protocol

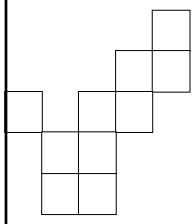


Millicent architecture



Scrip layout

Vendor	Value	Scrip ID	Customer ID	Expiry date	Properties	Certificate



Grid Technology

B. Ramamurthy

Reference Material

- Grid Computing, Making the Global Infrastructure a Reality by F. Berman, G. Fox and A. Hey, Wiley and Sons Ltd., 2003, ISBN: 0-470-85319-0
- Publications from the site:
- http://www.globus.org/research/papers.html

Introduction

- The Grid
- The History
- Building blocks of the global grid
- Layered Grid Model
- Grid Applications
- Categories of applications
- Future of Grid

The Grid

- *The grid is a a computing and data management infrastructure that provides us ability to dynamically link together resources to support execution of large-scale, resource-intensive, and distributed applications. (paraphrased from Fran Berman et al's text)*
- Grids are intrinsically distributed, heterogeneous and dynamic.

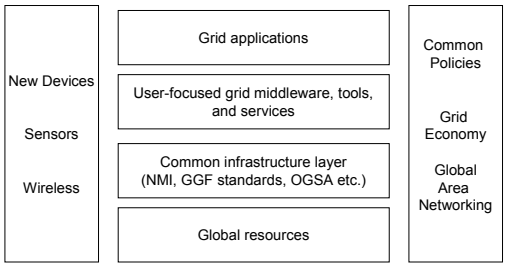
History of the Grid

- 1980s parallel computing was used as a means of achieving high performance. Examples: Parallel virtual Machine (PVM), Message Passing Interface (MPI), and High Performance Fortran (HPF).
- 1990s the focus shifted into coordination, distribution and collaboration, the fundamentals concepts of grid computing.
- I-Way, the precursor modern day grid was demonstrated in the year 1995 in SC conference.
- This lead to the development of
  - grid software in Globus, Condor, Legion, and others
  - services such as Network Weather Service (NWS), Storage Resource Broker (SRB)
  - Protocols such as Open Grid Services Architecture (OGSA), Grid Security Infrastructure (GSI)

Building Blocks

- **Networks**: grids are built on ubiquitous high-performance networks such as Internet2 Abilene, and intra-Europe GEANT network. Networks connect resources on the grid, such as the computers (nodes) and the storage.
- **Computational nodes**: Nodes are high performance parallel machines or clusters.
- **Infrastructure software**: This focuses on pulling together the network and the nodes and provides a development environment and execution platform for the applications.
- **Standards**: Development of key standards is critical for the successful management of the grid complexity. OGSA that provides the standard for the services on the grid is a fine example of such an effort.

The Layered Model



Grid Applications

- Life science Applications
- Engineering-oriented applications
- Data-oriented applications
- Physical science applications
- E-science collaboratory
- Commercial applications

Categories of applications

- Minimal communication applications: embarrassingly parallel applications. Ex; SETI@home
- Staged/linked application: access to remote instruments
- Adaptive applications: run where you find resources satisfying criteria.
- Real-time and on-demand application: do something right now.
- Coordinated applications: dynamic and brand and bound applications
- Poly-applications: choice of resources for different components of the application.

Trends

- Development of models of interaction between users and grid: Grid Computing environments and portals
- Access technologies: non-computer means of access.
- Policies: grid resources are in different domains. Developing policies is a challenge.
- Grid economies: Building a business model around it is another interesting challenge.
- Grid will serve as the enabling technology for a broad set of applications in science, business, entertainment, health and other areas.

From Prototype to Production Grid

B. Ramamurthy

Introduction

- In the last lectures we looked at the design of a prototype test bed for the grid based on the paper
  - http://www-library.lbl.gov/docs/LBNL/511/92/PDF/LBNL-51192.pdf
- This lecture we will look into the details of transition from the test bed to a production grid.

First steps

- Issue host certificates for all the computing and data resources and establish procedures for installing them.
- Issue user certificates.
- You may revoke the certificates to make sure of the operations and reissue them.
- Using certificates issued by your CA validate correct operation of GSI, GSS libraries, GSISSH and GSIFTP and/or GRIDFTP at all sites.
- Read: Certification Systems:X.509,CA, PGP at http://mcg.org.br/cert.htm
- Another URL to look at to get an overall picture:
- http://www-library.lbl.gov/docs/LBNL/511/92/PDF/LBNL-51192.pdf

Defining and Understanding the Extent of the Grid

- Boundaries are primarily defined by:
  - Interoperability of the grid software
  - What CAs you must trust: This is explicitly configured in each Globus environment on per CA basis.
  - How you scope the searching of the GIS or control the information that is published in them. It depends on the model you choose to structure your directory services.

Model of the GIIS

- GIIS (Resource Information Servers) and directory servers are needed.
- Use a X.500 style hierarchical name component space directory structure. VO roots can be attached to the hierarchy extending the scope.
- Index server directory structure: Use Globus MDS for information directory hierarchy.

Local Authorization

- A Globus mapfile is an ACL that maps from Grid identities to local user identification numbers (UIDs) on the systems where jobs are to be run.
- A Globus Gatekeeper replaces the usual login authorization mechanism for Grid-based access and uses mapfile to authorize access to resources after authentication.

Site Security Issue

- Any distributed application requires use of many IP communication ports. If the server is behind firewall these ports may not be accessible. Typical application may require several 10s of ports.
- Globus can be configured to use mid-700 range ports and make sure the sysadmin knows about the block usage.
- Proxies can help manage intra-service component communication.

High Performance Communication Issue

- If high data rate distributed applications are anticipated, enlist the help of WAN networking people to refine network bandwidth end-to-end using large packet size data streams.
- Network monitors and Loggers can help in monitoring and identifying low rate problems.

Batch Schedulers

- Job initiation and resource management are very important functions closer to the application level.
- Parallel Batch Scheduler (PBS) , Condor-G are examples of schedulers.
- PBS provides time-of-the-day based advanced resource reservation.
- Schedulers also maintain queues and implement access control.
- PBS also has full preemption capabilities that combined with existing access control mechanisms can provide full disaster response or scheduling of high priority job preempting a lower priority one.

Preparing for the Deployment

- Identify some sample problems to test the working of the grid.
- Read a sample “Quick Start Guide” available at http://www.globus.org/toolkit/documentation/QuickStart.pdf
- At this point Globus, GIS/MDS, security infrastructure should all be operational.
- Deploy and build Globus on at least two production platforms at two different facilities.
- Configure job submission and schedulers and verify them.

Grid Service Model

- Establish a model for moving data> For example: GridFTP.
- Check the operation using a sample service such as MyProxy service: provides for creating and storing intermediate lifetime proxies that can accessed by Web-based portals, job schedulers, and so forth.

Summary

- We outlined the installation of prototype grid.
- We also sketched the details of moving from a prototype grid to a production grid.
- Your task is to read the main paper and the related material referenced in the presentation.

Globus User's Guide and Programmer's Guide:

1. User's guide tells you about the software and tools needed and how to install, configure and verify these.

2. Programmer's guide goes through the details of designing a service and implementing it.

    a. Provide service interface
    b. Generate Grid service support code
    c. Implement the service
    d. Deploy the service

More Details:

a. Provide service interface:

Two approaches:

interface in Java Ą generate WSDL interface

WSDL portType interface Ą generate SOAP binding (Define it in gwsdl)

(PortType is an element defined in WSDL that defines a set of operation and the messages needed for the operations).

b. Generate Grid Service Support Code:

--All the tools for stub and support code generation are centered around generateWSDL and generateStubs.

--Ant task and xml batch files are provided to generate the required stub and code for hosting the service as an OGSI compliant Grid Service.

Bottom up:

--used when the service is available as legacy code in Java and we want to grid enable it.

Top down:

-- Used when service is in available in someother language other than Java and you want a Java implementation. Or when when a new grid service is defined.

-- From GWSDL interface: Use GWSLD2WSDL tool to generate WSDL 1.1 portType, run generateBinding tool to generate wsdl:binding and wsdl:service parts for the portType definition; generateStubs for generating stubs.

c. Implement the service:

--See the Figure 2 Server Programming Model we discussed in the core white paper.

-- Two approaches: Inheritance approach and Operation provider approach.

--Inheritance extends GridServiceImpl but is tightly coupled with the implementations in the container.

-- Operation Provide approach makes it easy to plug in various implementations at deployment time.

-- OGSI defined implementations of NotificationSource and Factory have been implemented as OperationProviders in the framework. These can be readily configured into the service using deployment descriptors.

-- QName : Qulaified name: conatins namespace and a name as in wsdl.

-- * specifies all operations in a certain namespace

d. Deploy the service:

--write a deployment descriptor configuring your service

-- create a "gar" package of the configuration along with your implementation

-- deploy the gar package into a Grid service hosting env: from OGSA installation directory run the deploy command.

e. Writing a client

1. Get OGSiGridServiceLocator
2. Resolve GridServiceFactory
3. Resolve CounterServiceGridLocator
4. Make proxy/stub
5. Invoke operation on stub

Toward a Framework for Preparing and Executing Adaptive Grid Programs

Ken Kennedy[α], Mark Mazina, John Mellor-Crummey, Keith Cooper, Linda Torczon
Rice University

Fran Berman, Andrew Chien, Holly Dail, Otto Sievert
University of California, San Diego

Dave Angulo, Ian Foster
University of Chicago

Dennis Gannon
Indiana University

Lennart Johnsson
University of Houston

Carl Kesselman
USC/Information Sciences Institute

Ruth Aydt, Daniel Reed
University of Illinois, Urbana-Champaign

Jack Dongarra, Sathish Vadhiyar
University of Tennessee

Rich Wolski
University of California, Santa Barbara

Abstract

This paper describes the program execution framework being developed by the Grid Application Development Software (GrADS) Project. *The goal of this framework is to provide good resource allocation for Grid applications and to support adaptive reallocation if performance degrades because of changes in the availability of Grid resources. At the heart of this strategy is the notion of a* configurable object program, *which contains, in addition to application code, strategies for mapping the application to different collections of resources and a resource selection model that provides an estimate of the performance of the application on a specific collection of Grid resources. This model must be accurate enough to distinguish collections of resources that will deliver good performance from those that will not. The GrADS execution framework also provides a* contract monitoring *mechanism for interrupting and remapping an application execution when performance falls below acceptable levels.*

Introduction

The recently-published volume *The Grid: Blueprint for a New Computing Infrastructure* [5] has established a compelling vision of a computational and information resource that will change the way that everyone, from scientist and engineer to business professional, teacher, and citizen uses computation [5,12]. Just as the Internet defines fundamental protocols that ensure uniform and quasi-ubiquitous access to communication, so the Grid will provide uniform access to computation, data, sensors, and other resources. Grid concepts are being pursued aggressively by many groups and are at the heart of major application projects and infrastructure deployment efforts, such as NASA's Information Power Grid (IPG) [7], the NSF PACI's National Technology Grid [12] and Distributed Terascale Facility, the NSF's Grid Physics Network, and the European Union's EU Data Grid and Eurogrid projects. These and many other groups recognize the tremendous potential of an infrastructure that allows one to conjoin disparate and powerful resources dynamically to meet user needs.

Despite the tremendous potential, enthusiasm, and commitment to the Grid paradigm, as well as the sophistication of the applications being discussed, the dynamic and complex nature of the Grid environment poses daunting challenges. Few software tools exist. Our understanding of algorithms and methods is extremely limited. Middleware exists, but its suitability for a broad class of applications remains unconfirmed. Impressive applications have been developed, but only by teams of specialists [3, 4, 5, 6, 8, 9, 11].

Entirely new approaches to software development and programming are required for Grid computing to become broadly accessible to ordinary scientists, engineers, and other problem solvers. In particular, it must be relatively easy to develop new Grid applications. Currently applications are developed atop existing software infrastructures, such as Globus, by developers who are experts on Grid software implementation. Although many useful applications have been produced this way, this approach requires a level of expertise that

[α] Corresponding author: ken@rice.edu

will make it difficult for Grid computing to achieve widespread acceptance.

The *Grid Application Development Software (GrADS) Project* was established with support from the NSF Next Generation Software Program to help address this challenge. In the GrADS vision, the end user should be able to specify applications in high-level, domain-specific problem-solving languages and expect these applications to seamlessly access the Grid to find required resources when needed. Using such environments, users would be free to concentrate on how to solve a problem rather than on how to map a solution onto available Grid resources.

To realize this vision we must solve two fundamental technical problems. First, we must understand how to build programming interfaces that insulate the end user from the underlying complexity of the Grid execution environment without sacrificing application execution efficiency. Second, we must provide an execution environment that automatically adapts the application to the dynamically-changing resources of the Grid. To address this second problem, the GrADS project has designed an execution framework for adaptive Grid applications. The goal of this paper is to elaborate the design of this framework and the motivation behind it.

The GrADS Framework

Initial efforts within the GrADS project have demonstrated the complexity of writing applications for the Grid and managing their execution. To deal with this complexity, the GrADS project has adopted a strategy for program preparation and execution that revolves around the idea that a program must be *configurable* to run on the Grid. To be configurable in the sense intended by GrADS, a program must contain more than just code—it must also include a portable strategy for mapping the program onto distributed computing resources and a mechanism to evaluate how well that mapped program will run on a given set of resources. The notion of a *configurable object program* is thus at the heart of the GrADS execution framework. Later in this paper, we will discuss tools to help construct mapping strategies and performance models that are part of the configurable object program. For now, we will simply assume that these components exist in executable form.

Once a configurable object program, plus input data, is provided to the GrADS execution system, there must be a process that initiates the resource selection, launches the problem run, and sees its execution through to completion. In the GrADS execution framework, the *Application Manager* is the process that is responsible for these activities—either directly or through the invocation of other GrADS components or services. In this scenario, individual GrADS components only need to know *how* to accomplish their task(s); the question of *when* and with *what* input or state becomes the Application Manager's responsibility.

The application launch and execution process is illustrated in Figure 1. We will step through this process discussing the role of each component in the execution launch sequence.

Application Execution Scenario

A Grid user, or a problem solving environment (PSE) on behalf of the user, provides source code (which may be annotated with resource selection or run-time behavior information) or a handle to an existing IR Code object previously created for the user. This is given to a component called the *Builder*, which is the part of the program preparation system responsible for producing a configurable object program (COP). An overview of how the Builder accomplishes its task will be provided in a later section.

The Builder will construct any required objects and return a handle to a configurable object program, which includes the IR Code, the mapping strategy (or *Mapper)*, and the performance model, which we will refer to as the *Resource Selection Evaluator (RSE).* In addition, the Builder will provide a model of the resource space needed for execution of the application. This is called an *Application Abstract Resource and Topology (AART) Model*. An AART Model provides a structured method for encapsulation of application characteristics and requirements in an input-data-independent way. This information is in the form of a collection of descriptive and parametric resource characteristics along with a description of the topology connecting these resources. The purpose of the AART Model is to kick-start the resource selection process and to provide part of the information needed by the Mapper and the Resource Selection Evaluator.

Next, the user starts the Application Manager. This may be a standard GrADS Application Manager or a specialized manager designed by the user. The Application Manager needs the handle to the COP, I/O location information, the problem run size information (specifically, information to allow calculation of memory requirements), plus any desired resource selection criteria and other run-specific parameters desired or required.

The Application Manager retrieves the pieces of the COP. The AART Model is combined with the problem run information, resulting in the Resource Selection Seed Model. This produces the preliminary state necessary for the Mapper and the Resource Selection Evaluator to start being useful.

Once these components are available, the application manager invokes the *Scheduler/Resource*

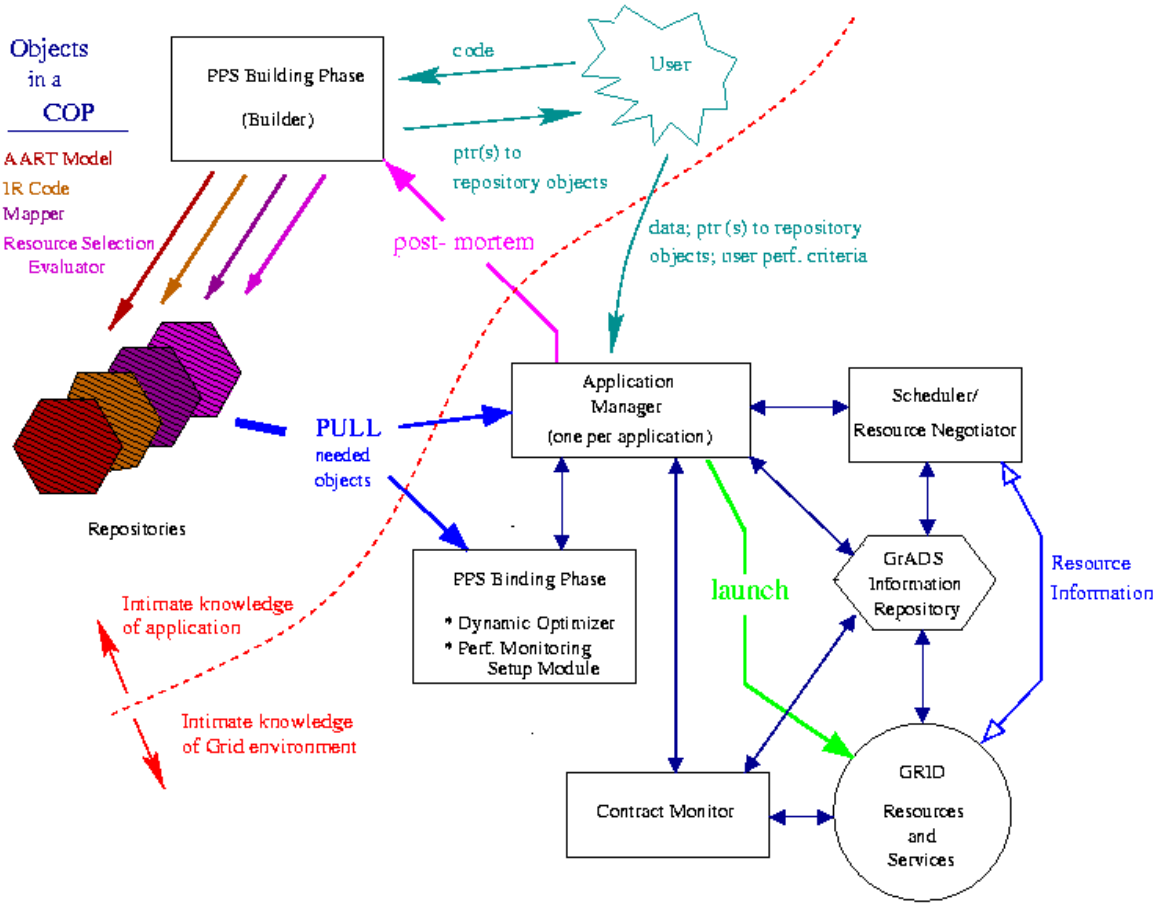


Figure 1: The GrADS Application Launch and Execution Process

Negotiator (S/RN) and provides it with the Resource Selection Seed Model. The Scheduler/Resource Negotiator is the component responsible for choosing Grid resources appropriate for a particular problem run based on that run's characteristics and organizing them into a proposed virtual machine. In GrADS, the S/RN is basically an optimization procedure that searches the space of acceptable resources looking for the best fit according to the application's needs as determined by using the Resource Selection Evaluator as an objective function.

The Scheduler/Resource Negotiator then invokes the Grid Information Service to determine the state of Grid resources and determine what resources are available that satisfy the characteristics required by the Resource Selection Seed Model. In other words, the Resource Selection Seed Model defines a feasible resource space for application execution. Once sets of feasible sets of resources are identified, they are organized into a collection according to the proposed Grid virtual machine. The Scheduler/Resource Negotiator then searches the collection of feasible sets of resources to find the one with the best performance on the given application, using the Resource Selection Model provided by the Application Manager as the objective function.

Once a collection of resources has been identified, the Application Manager begins the launch sequence. First, it stores state (basically a checkpoint) on the impending problem run (i.e. application + data) in the GrADS *Program Execution System (PES) Repository*, which is used to keep track of where each component of the application is executing and provide sufficient information to restart the application in the case of a catastrophic component failure. The Application Manager then invokes the *Program Preparation System (PPS)*

Binding Phase, passing it the COP handle, selected virtual machine, and the user's run-time information.

The PPS Binding Phase invokes the Mapper to perform the actual data layout and creates optimized binaries using a component called the Dynamic Optimizer, which performs tailoring of the program components to the specific computational resources on which they will run. The Binding Phase also inserts monitoring sensors needed by the performance-monitoring component of the execution environment, which is referred to as the *Contract Monitor*. The Contract Monitor is responsible for identifying egregious violations of the performance assumptions that led to the original resource mapping and initiation a reallocation of resources if necessary. The definition of what sensors are needed is provided by the Performance Monitoring Setup Module, which is invoked from within the PPS Binding Phase.

For some Grid-aware libraries, the PPS Binding Phase may need to arrange for dynamic linking to pre-built libraries for specific platforms. Handles to the optimized problem run binaries are passed back to the Application Manager, which again checkpoints its state to the GrADS PES Repository.

The Application Manager starts the Contract Monitor and then launches the binaries by invoking the GrADS Launcher, a service that is constructed on top of the Globus middleware layer. While the Contract Monitor is initializing, code inserted by the PPS in the application binaries may be positioning data on the resources making up the virtual machine.

As the code runs, the Contract Monitor gathers sensor data and uses the contract monitoring performance model(s) and violation thresholds provided by the Performance Monitoring Setup Module to determine if the application is delivering an acceptable level of performance. In addition, the Contract Monitor may try to make some determination of the cause of the poor performance. It reports its findings, together with summary monitoring information, to the Application Manager.

The evaluation of acceptable levels of performance and determination of the cause of violations is the shared responsibility of the Contract Monitor Component and the Application Manager, with the final decision to signal a violation coming under the domain of the Application Manager. The distribution of the decision making effort between the components will vary as appropriate for the given application structure, contract monitoring performance model granularity, and violation type.

Concurrently, the Contract Monitor output, as well as the original sensor output, can be archived for later use to refine models, adjust thresholds, or guide future executions. In addition, the application, Contract Monitor, and Application Manager may adjust the contract monitoring performance models and violation thresholds throughout the application lifetime in response to evolving application patterns and resource volatility.

If the Application Manager determines that the application is not making reasonable progress (or alternately, if the system becomes aware of more suitable execution resources), the *Rescheduler* is invoked. Using knowledge of the current execution, the Rescheduler determines the best course of action in order to improve progress. Examples of rescheduling actions are replacing particular resources, redistributing the application workload/tasks on the current resources, and adding or removing resources; or doing nothing (continuing execution with the current VM).

If the Rescheduler constructs a revised VM, the Application Manager builds new optimized executables, checkpoints the application, reconfigures and re-launches the application. The application reads in the checkpoint information and continues program execution. Once the application finishes, the Application Manager makes certain that the relevant collected performance data is fed back (i.e. archived) into the Program Preparation System and shuts down the Contract Monitor.

Constructing Configurable Object Programs

Clearly, for this execution scenario to work, we must have a reasonable performance model and mapping strategies for each application. In fact, the performance model depends on a preliminary mapping provided by the mapping strategy, so these two components are intimately tied together. In our preliminary research [10], we discovered that performance models for non-homogeneous collections are extremely difficult for even sophisticated developers to construct.

As a result, we have adopted a strategy of providing within the program preparation system a collection of components and tools to assist in the development of the requisite performance models and mapping strategies. These tools will use three general strategies for constructing reasonably accurate performance models:

1. Expert knowledge about performance of components, particularly on different classes of homogeneous parallel processors.
2. Trial execution to determine run times of important components, with estimates of communications costs based on information from the Grid Information Service.
3. Integration of whole-application performance models from accurate models for individual components, based on the topology of the application.

The design and evaluation of these tools is a subject of ongoing research. However, our preliminary studies

indicate that there is strong promise that these three strategies can combine to provide enough accuracy to make the resource selection process effective [1,10].

Project Status

Preliminary versions of the execution model described in this paper have been prototyped in the context of two demonstration applications: ScaLAPACK [10] and Cactus [1]. We are currently working toward an implementation that includes generic versions of these components that can be used with any configurable object program.

Bibliography

1. G. Allen, D. Angulo, I. Foster, G. Lanfermann, C. Liu, T. Radke, E. Seidel, and J. Shalf. The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Environment. *International Journal of High Performance Applications and Supercomputing* 15(4), Winter, 2001.
2. F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, and R. Wolski. The GrADS Project: Software Support for High-Level Grid Application Development. *International Journal of High Performance Applications and Supercomputing* 15(4), Winter, 2001.
3. T. DeFanti, I. Foster, M. Papka, R. Stevens, and T. Kuhfuss. Overview of the I–WAY: Wide--Area Visual Supercomputing. *The International Journal of Supercomputer Applications and High Performance Computing* 10(2):123–130, Summer/Fall 1996.
4. I. Foster, J. Geisler, W. Nickless, W. Smith, and S. Tuecke. Software Infrastructure for the I–WAY Metacomputing Experiment. To appear in *Concurrency: Practice & Experience*.
5. I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, 1998.
6. E. Gabriel, M. Resch, T. Beisel, and R. Keller. Distributed Computing in a Heterogenous Computing Environment. In *Proc. EuroPVMMPI'98*. 1998.
7. W. E. Johnston, D. Gannon, and B. Nitzberg. Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid. In *Proceedings of the 8th IEEE Symposium on High-Performance Distributed Computing (HPDC)*, IEEE Computer Society Press, 1999.
8. T. Kimura and H. Takemiya. Local Area Metacomputing for Multidisciplinary Problems: A Case Study for Fluid/Structure Coupled Simulation. In *Proc. Intl. Conf. on Supercomputing*, pages 145–156. 1998.
9. P. Lyster, L. Bergman, P. Li, D. Stanfill, B. Crippe, R. Blom, C. Pardo, and D. Okaya. CASA Gigabit Supercomputing Network: CALCRUST Three–Dimensional Real–Time Multi–Dataset Rendering}. In Proceedings of Supercomputing '92, Minneapolis, Minnesota, November 1992 (Poster session).
10. A.Petitet, S.Blackford, J.Dongarra, B.Ellis, G.Fagg, K.Roche, and S.Vadhiyar. Numerical Libraries and the Grid: The GrADS Experiment with ScaLAPACK. *International Journal of High Performance Applications and Supercomputing* 15(4), Winter, 2001.
11. T. Sheehan, W. Shelton, T. Pratt, P. Papadopoulos, P. LoCascio, and T. Dunigan. Locally Self Consistent Multiple Scattering Method in a Geographically Distributed Linked MPP Environment. *Parallel Computing* 24, 1998.
12. R. Stevens, P. Woodward, T. DeFanti and C. Catlett. From the I–WAY to the National Technology Grid. *Communications of the ACM* 40(11): 50–60, November 1997.

CSE 4/586 Distributed Systems Fall2003
Project 1: Designing and deploying a Web Service

Purpose:

1. To understand the components, the core technologies, the architecture and the protocols enabling a Web Services-based distributed system.
2. To design and implement a Web Service.
3. To understand the elaborate and complex process of preparing and deploying a remote service.

Preparation before lab:

1. Read and study the Web Services architecture and the associated protocols:
http://www.w3.org/2002/ws/
There is another article available as a first chapter of the Web Services tutorial offered by Sun Microsystems at http://java.sun.com/webservices/tutorial.html

2. Learn how to use the XML-based build tool Ant at http://ant.apache.org/

3. Understand the role of deployment descriptors. The deployment descriptors are XML files used to configure runtime properties of an application thus relieving application to deal only with the programmatic details. A simple reading on deployment descriptors can be found at
http://www.systinet.com/doc/wasp_jserver/waspj/deployment_descriptors.htm

4. Learn to use the application interface to the Oracle database using embedded SQL and JDBC.

5. Finally, you must have a clear understanding of a client-server system operation.

Web Services Technology:

Web Services technology provides a standard means (SOAP, XML over HTTP) of buidling a distributed system over the Internet. In simple terms, it provides a means for a sophisticated remote procedure call. The sophistication arises out of the elegant mechanisms it supports for enabling (i) various transparencies (platform, language, and hardware) (ii) application to application data exchange and interoperability, and (iii) composability of complex web services from a set of simple web services.. The significant difference between the regurlar HTTP-based technologies and Web Servcies is the standardization realized through the XML and SOAP. Web Services Definition Lanaguage (WSDL) is an important standard supported that allows for standard definition of services. All these make Web Services technology ideally suited for large-scale enterprise level application integration.

Web Servcies specification is defined by World Wide Web (W3) consortium in terms of (i) Web Servcies architecture requirements, (ii) Web Services architecture, (iii) Web Services glossary, and (iv) Web Services architecture usage services. Many vendors including Sun Microsystems (Sun One) and Microsoft (.net) have frameworks for building and deploying Web Services.

Assignment:

Build a multi-tier distributed system comprising two major sub-systems (i) an RMI and simple data acquisition system and (ii) a Web Services based web application processing and serving the data collected. The two sub-systems are loosely coupled via a database. The block diagram of the system you will implement is given in Figure 1. The RMI part of the project is adapted from the Weather service problem described in the fourth edition of *Java: How to Program?* By Deitel and Deitel.

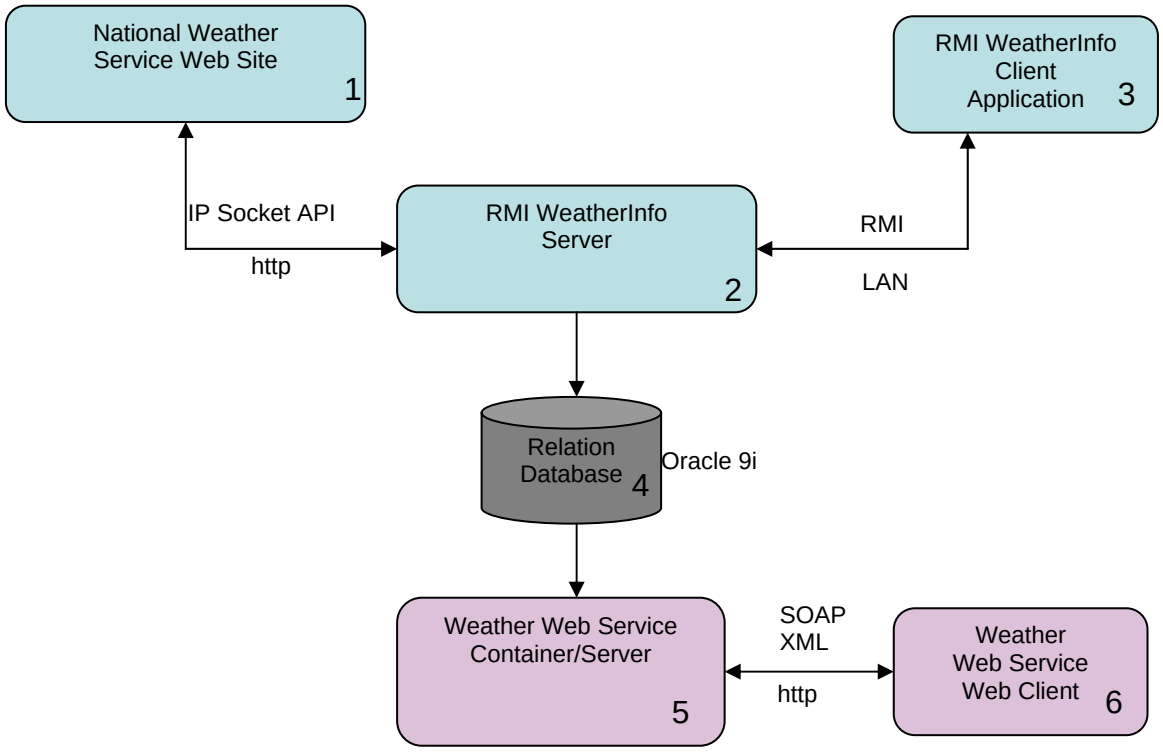


Figure 1: System Architecture of the Weather Service

The national weather bureau updates the weather conditions at various cities once every day on its web site (box 1 in Figure 1) at http://iwin.nws.noaa.gov/iwin/us/traveler.html . The RMI Server (box 2) streams in the page and parses it for the relevant data and stores it in a persistent storage. (The details of the RMI and existing code base and a simple frame work were discussed during the lecture and are available at http://www.cse.buffalo.edu/gridforce/courses.htm) The persistent storage in the sample code is a simple file and the data stored is just the weather data for one day. You are required to update the code to accumulate the data for a period of over at least 1 week (or

any 7 days). The data collected will be stored in a relational data base (box 4) on Oracle 9i. The daily data in the sample code is served to an RMI client (box 3) which simply renders the ASCII data provided by the weather bureau in the form of visually appealing graphics.

In the Web Services part of the system in Figure 1, the data collected in the data base will be processed by the server (box 5) for such information as average temperature for a given city, and the temperature for a particular date for a city. The Web Services client (box 6) will be able to query the server for various information related to the data collected. Your task is to design and implement the complete Web Services-based system indicated by boxes 4, 5 and 6 of Figure 1.

Project Implementation Details and Steps:

1. **Getting used to building client-server systems**: When you implement a simple client side application program there are just two steps involved: compile and execute the code. In a client-server system, you will have to take care of the server side as well as the client side. On the server side, you will compile the code, generate stubs or proxies using special compilers, deploy the service, register and publicize the service for the clients to use. On the client side you will prepare the client code with appropriate stubs, and during excution lookup the service needed and use it. To understand the process study the RMI-based system code and implementation. Deploy it and make sure it works and you understand the various operations. You will notice that besides simple compile and execute, configuration and deployment of a service are important issues to be reckoned with.

2. **Working with the relational database and embedded SQL:** In this project you will store the data in a relational table and access it using SQL statements embedded in Java lanaguage. Work on a simple java program to refresh your knowledge about accessing the Oracle database. See http://www.cse.buffalo.edu/local/Consulting/Orcale.html for examples and access details.

3. **Building systems using build tools such as Ant:** In order to tackle complexities in configuration and deploying server-side applications, you will need to use special build tools. Apache Ant is a XML-based build tool which similar to "make" utility that most of you are familiar with. This topic will be covered during the recitation this week. Work on simple files to familiarize yourself with the Ant build tool.

4. **Study and understand the Web Services building and deployment details:** For the Web Servcies part we will use the Sun Microsystems implementation of the Web Servcies specification. We have a version of the HelloWorld (in Web Services) available at /projects/bina/cse486. This will provide framework to develop your Web Services client-server system. It has clear directory structure

which your are expected to follow. It has the source code for the server and the client, a build file and a configuration file in XML. Copy the code WSSample.tar into your project space. Unzip it, build the server and deploy the server. Build the client which is a web application and access the service provided by the server. For this step you need to download and install JWSDP 1.2 http://java.sun.com/webservices/downloads/webservicespack.html

5. **Design, implement and test your weather Web Service:** Using the frame work given in the Step 4 above design the Web Service for dispensing and answering user queries about the weather information of various cities. This is expected to be the most time consuming part of the project due to the novelty of the topic.

6. **Deploy the integrated system:** The various components listed above were deployed and tested individually. In this step you will run the entire integrated system. The RMI part can be scheduled to acquire data once a day to update the database that will be used by the Web Service part.

Submission Details:

Create a project1 directory and use that as the working space. Let the code directory tree be located in this directory. Let the design be represented by an integrated class diagram and presented in a file project1.pdf. Provide internal documentation using javadoc style comments. You will create a README file and also a file that contains the questions and answers for the questions pertaining to the topic of the project that will be given to your later.

Zip the project1 directory and submit the resulting zip file, project1.zip. Making sure that you current directory contains your project1 directory, you can create this file as follows:

```
zip -r project1.zip project1
```

Use the electronic submission program that corresponds to your section. For instance students in section A will submit by typing

```
submit_cse486 proejct1.zip
```

at the Unix prompt.

Due Date: 10/10/2003 by midnight.

Project 2 A Simple Java-based Framework for Grid Computing Fall2003
Based on http://www.javaworld.com/javaworld/jw-04-2003/jw-0425-grid.html

1. Introduction

Grid computing is a natural evolution of the information infrastructure successfully realized using the Internet. It provides an infrastructure for the flow of services by exploiting the vast pool of resources networked by the Internet. Early beginning of the grid computing can be observed in the SETI@home project. Currently many toolkits such as Globus Toolkit 3.0 and Condor 6.5.5 are available to implement the grid framework. However these frameworks are production-quality and are quite complex for us to understand, deploy and take apart to study and experiment with the code. (For example, in the CSE421 Operating Systems course we can study, understand and extend the code whereas with a complete Unix BSD or Solaris you may not be to do as easily.) So we have decided to let you build a minimal grid framework based on the article "A do-it-yourself framework for grid computing" by Anthony Karre in Java World. Our focus in this project will be on the client-side of the grid computing (though we will run a server). The framework given in paper satisfies the following grid requirements:

1. Machine independence (The paper uses Java, Apache Tomcat servlet container and Apache Axis SOAP implementation)
2. Secure and scalable infrastructure achieved through the use of SOAP-based Web services for client-server communication.
3. Task abstraction achieved through the use of jar files, and Java classloader. A classloader capable of identifying and executing an arbitrary Java class is also provided.

In this project we will use *SOAP with Attachments API for Java* (SAAJ) instead of the Apache Axis.

2. Purpose (Goal of the Project)

Implement a simple framework for the grid clients to retrieve, load and execute a specified task given the Jar file of the job submitted to the grid. The client will also return at a later time a representative result of the execution to the server. The executing task could use another webservice to submit the result to the server.

3. Technology Requirements

Java 2 platform standard edition (J2SE 1.4.1 or later), Apache Tomcat servlet container, SAAJ, and JAX-RPC based webservices.

4. Assignment

4.1 Architectural Model

The block diagram of the overall system is given Figure 1. Computing task is submitted to the grid server as a JAR file with the manifest indicating the task thread. The client

side contains a main class and a custom classloader. The client side operations are as follows:

1. The main class instantiates the custom classloader.
2. The custom classloader uses a SOAP service to fetch the JAR file containing the task thread classes. The JAR file stored locally upon retrieval.
3. Then the main client class uses the classloader to load and instantiate the primary compute thread identified by the manifest in the JAR file.
4. The compute thread is then started by the main class.
5. The result of the computation is returned to the server using other methods such as Webservices. (Observe that server and client are loosely coupled.)

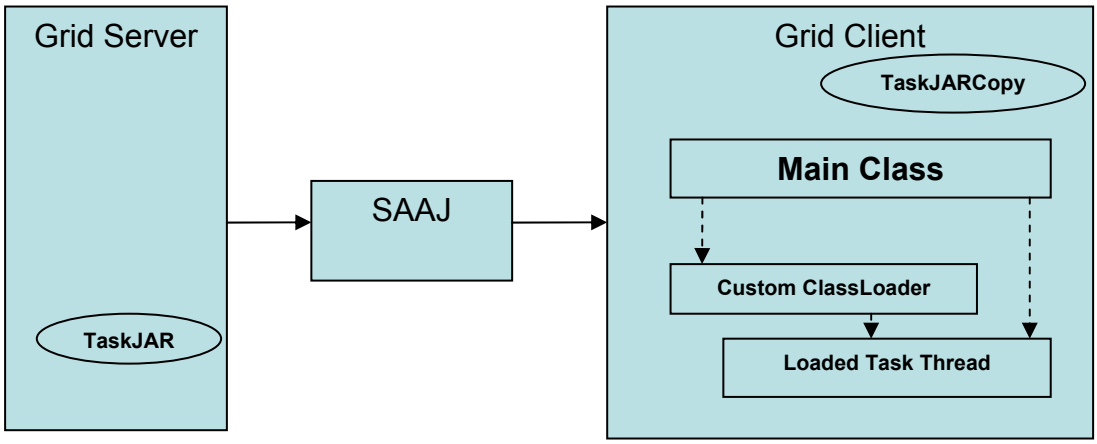


Figure 1: Components of the Grid Server and Grid Client of the Project1

4.2 Implementation Details

1. You will design and implement a simple server that will allow users to submit computing tasks in the form of JAR files.
2. Initially a trivial compute thread similar to the one discussed in the paper referenced will be coded and packaged into the JAR file and stored on the server.
3. A SOAP service will be used to transfer the task JAR file to the client on request. Use SAAJ for building and sending the SOAP request and response messages.
4. On the client side you will build a custom classloader to retrieve the JAR file using the SOAP service. It will also load the class containing the task thread.
5. Design and implement the main client application for instantiating the custom classloader, then using it to load the task thread.
6. Design and implement a means for returning the results back to the server.
7. Use the infrastructure build to create and run any non-trivial application.

5. Report and Submission

See Project 1 for the Report that you need to prepare and for the submission details.
Due Date: November 15, 2003 by midnight. No extension will be given.

Project 3 Design, Implementation and Deployment of a Grid Service Fall2003
Due Date: 12/8/2003

1. Introduction

In this project we will design, implement, deploy and test various versions of a single grid service from a basic service to sophisticated service with features such as logging and notification enabled. During the lecture classes we examined the description, architecture of and infrastructure supporting a Grid Service. For details of a grid service, grid service architecture (Open Gird Services Architecture: OGSA), grid services infrastructure (Open Grid services Infrastructure: OGSI) and the hosting environments see the comprehensive paper on this topic in [1]. The software that we will be using is the core of the Globus Toolkit 3.0.2. The core of the Globus can be downloaded from http://www-unix.globus.org/toolkit/download.html#core . The details of the core are available in a white paper on the core services at http://www-unix.globus.org/core/ . This white also contains a javadoc-style Grid Services API description, User's Manual and a Programmer's Manual. The user's manual provides the instructions to compile, build, convert, deploy and test a grid service. The programmer's manual provides the details of writing a grid service, the various programming choices available, and deployment description. A samples directory in the core package provides a numerous examples illustrating the various grid services features.

[1] The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. I. Foster, C. Kesselman, J. Nick, S. Tuecke, Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002. (extended version of Grid Services for Distributed System Integration) [Citation, PDF]

2. Purpose (Goal of the Project)

Implement a suite of grid services ranging in complexity from a basic service to a sophisticated one for the weather service you implemented in the Project 1. The range of services in some ways represents the different qualities of service that a grid service can offer. Present the suite of services in a GUI interface for the user/client to choose, activate and execute. Prepare a GAR file of the deployment for later submission into CSE, CCR or Geneseo Grid.

3. Technology Requirements

Java 2 platform standard edition (J2SE 1.4 or later), Globus Tool Kit, core only.

4. Assignment

Write the weather service in Project 1 as a grid service. Let the weather service offer at least three functions/operations. The samples directory of the Core Globus download has many examples such as *counter* and *google*. The Counter example has sample code for a variety of implementations of the same counter. Among the features illustrated we are

interested in: basic, delegation, generate, notification, logging, secure and persistent. For the corresponding implementation of the weather service use this naming convention: <feature><service_name>Impl.java

For example: SecureWeatherImpl.java, LoggingWeatherImpl.java etc.
Reuse the GUI interface the core package provides to present your weather service.

4.1 Implementation Details

1. Download, install, configure and verify the core of the Globus Toolkit.
2. See the details of programming and building a grid service in a fine tutorial offered at http://www.casa-sotomayor.net/gt3-tutorial/
3. Build and test all the sample services that are provided in the downloaded package.
4. Implement the basic weather service and deploy it.
5. Test it with a simple command line client.
6. Modify the GUI to incorporate your basic weather service by adding GUI buttons and boxes. Test it.
7. Repeat the steps 3-5 for other improved versions of the weather service.
8. Prepare the GAR (grid archive file) for job submission into real grid. We will provide you the details later.

5. Report and Submission

See Project 1 for the Report that you need to prepare and for the submission details.

Due Date: December 8, 2003 by midnight. No extension will be given.

University At Buffalo

COURSE OUTLINE

A. Course Title: **CSE 487/587 Information Structures**

B. Curriculum: Computer Science and Engineering Offering

Program: BS, BA, MS and Ph.D

C. Catalog Description: 4 Credits Elective

The objective of the course is to give an overview of the use of information technology in large-scale commercial and scientific systems, with emphasis on the use of state of the art computing in realizing various services and the frameworks supporting these services. Concepts covered include: enterprise modeling, process modeling, process automation and streamlining, workflow management, messaging, persistent message queues, transaction monitoring, document exchange, application servers, service definition (web services, web services definition language: WSDL), connection and resource reservation protocols (TCP, grid computing), integration technologies and architectures (Java 2 Enterprise Edition: J2EE, eXtensible Markup Language: XML, and Globus toolkit).

D. Class Timing and Location: Tu/Th 8.00-9.20AM , Baldy 200G

E. Suggested Text: The Grid: Blueprint for a New Computing Infrastructure by I. Foster and C. Kesselman. Second Edition, Morgan Kaufmann Publishers, 2003.

Tools/Languages: Java SDK1.4, Java 2EE, XML, SOAP, Globus/Grid development environment

F. Instructor Information:

Instructor: B. Ramamurthy (Bina)
127 Bell Hall
bina@cse.buffalo.edu
http://www.cse.buffalo.edu/~bina
716-645-3180 (108)
Office Hours: MWF: 10-10.50AM

G. Program Competencies: ACM Curriculum 2001 suggests a model curriculum. Among the topics specified in that curriculum, this course will cover:

1. NC1: Net-centric Computing: Distributed Systems: Describe emerging technologies in the net-centric computing area and assess their current capabilities, limitations, and near-term potential. (Addressed by objective 1 through 7).

2. NC5: Building Web Applications: The Role of middleware, enterprise-wide web applications: Implement a distributed system using any two distributed object frameworks and compare them with regard to performance and security issues. (Addressed by objectives 2,3,4 and 6)

3. SE2: Using APIs: Design, implement, test, and debug programs that use large-scale API packages. (Addressed by objectives 2 and 6)

4. SE3: Software Tool and environments: Programming environments, testing tools, scripting and configuration management tools. (Addressed by objective 4)

5. SE9: Component-based programming: Apply component-oriented approaches to the design of a range of software including those required for concurrency and transactions, reliable communication services, database interaction including services for remote query and database management, secure communication and access. (Addressed by objective 2)

6. SE12: Specialized System Development: Distributed Systems: Grid Systems: Outline the central technical issues associated with the implementation of specialized systems development. (Addressed by course objectives 1 through 4)

7. CN4: High Performance computing: Recognize problem areas where computational modeling enhances current research methods (such as grid computing). (Addressed by course objectives 1, 3 and 5).

H. Course Objectives: At the completion of the course, the student should be able to:

1. Understand the application context of large scale distributed systems. For example: Scientific and industrial applications, from biomedical to astrophysics.
2. Work with the current application models: component-oriented, and grid-oriented.
3. Analyze and design the infrastructure needs of a large scale distributed system. For example: computing elements, configuration, deployment details, peer-to-peer communication.
4. Work with development tools (Ex: Eclipse integrated development environment, Apache Ant, and JUnit).
5. Understand and work with open grid services architecture and infrastructure.
6. Design and implement using Globus grid computing framework.
7. Technology Objectives: Students will be able to demonstrate the ability to design, implement, and deploy distributed systems based on Java technology and Grid Technology.

I. Assessment of Student Learning:

1. A mid-semester exam will be given during the semester and a three-hour final exam during the exam week scheduled by the university.
2. We will also use additional methods of evaluation to include: Graded programming assignments/projects, and lab exercises.

J. Learning Resources and other Support: It is required that all students attend the recitation that will be conducted by a Graduate Teaching Assistant (GTA). The instructor for the courses and the GTA will have each 3 hours of open office hours each per week to help with any question related to the course. Students are encouraged to use the open hours of Computer Science Lab at 338 Bell Hall and the Grad lab. Each student will be allocated individual project space to install the appropriate software needed for the labs.

K. Tentative Schedule:

Week of	Topics Covered	Reading Material
1/12	General Introduction; Course Outline; Cleint/server systems.	Firstday handout
1/19	Distributed System using Java RMI. Enterprise Computing: technology landscape.	Class notes: Will be posted
1/26	Project 1 discussion; Setting enterprise integration design objectives. Recitation: Distributed system using servlets.	java.sun.com/j2ee
2/2	Designing the enterpirse architecture; establishing the enterpise infrastructure; JDBC and using Oracle server	Department web page
2/9	Java 2 Enterprise Edition (J2EE). Techologies, enterprise application model, design and implementation of a sample application.	java.sun.com/j2ee
2/16	Grids in context: Fundamantal of a grid	Ch. 1
2/23	The scientific grid and the industrial grid	Ch. 2 and Ch.3
3/1	Project 2 discussion: Basic Grid Application; Application level tools and runtime systems.	Project 2 handout;
3/8	Globus Toolkit discussion: Open Grid Services Architecture and Infrastructure	www.globus.org; Ch. 4
3/15	Spring Break	
3/22	Grid architecture: Resource and service management	Ch. 17, 18
3/29	Building grid clients and services; Project 3 discussion	Ch. 19-21
4/3	Globus toolkit usage. Project 3 discussion: Grid based enterprise resource management	Project 3 handout; www.globus.org
4/10	Grid applications: Selected applications from Ch. 5-16.	
4/17	Data access, integration and management	Ch. 22
4/24	Review for the final exam.	

Date	Item Due
2/24	Project 1
3/31	Project 2
4/26	Project 3
2/26	Exam 1
Finals Week	Final Exam

Lab (Project) Topics

Note: Each lab will involve complete installation of all the necessary toolkits, software packages and servers by each student (or group of students) in their workspace. Students will also write a detailed technical report on the project they implement.

Lab 1: Design, implement and deploy a web application with component-oriented middleware 4 weeks

Lab 2: Design, implement and deploy a basic grid-oriented application. 4 weeks

Lab 3: Design and implement a sophisticated applications with groups within the class interacting. 4 weeks

L. Grading Policy:

Grades will consist of the following components:

Projects (3 projects) : 55%

Midterm Exam : 20%

Final Exam : 20%

Class Participation : 5%

Grade Range	Letter Grade
95 - 100	A
90 - 94.99	A-
85 - 89.99	B+
80 - 84.99	B
75 - 79.99	B-

70 - 74.99	C+
65 - 69.99	C
60 - 64.99	C-
55 - 59.99	D+
50 - 54.99	D
0 - 49.99	F

The ***Minimal*** point distribution guideline will be as above. We reserve the right to alter component weighting or provide a "curve" on an assignment as warranted. In order to pass this course you must have passing average in the Exam component of the course. All assignments will be graded and returned in a timely manner. When an assignment is returned, you will have a period of one week to contest any portion of the grade. Grading conflicts will be first resolved with the TA who graded your assignment. If the conflict cannot be resolved, the instructor will mediate the dispute. When contesting a grade, you must be able to demonstrate how your particular solution is correct. Also, when contesting a grade, the instructor or TA reserves the right to re-evaluate the entire exam, not just the question in dispute.

Projects

Projects constitute a major component of the course. Students will apply the concepts studied during the lecture in three group projects: an introductory project in enterprise systems, and two large projects in grid computing.

You will be given approximately four weeks to complete each project. *Do not be lulled into a safe sense of security thinking you have a lot of time to implement each project!* Much of your project development time will be spent in the design phase of your code. When implementing your solution, you should plan on using an *incremental development* path. You should plan your project in achievable stages such that you can get parts of your solution working a little at a time. This will help maximize partial credit during grading. Late assignments will be penalized at a rate of 25% of the achieved grade for each day overdue. The penalty will be assessed from the due date and time indicated on the assignment.

Exams

There will be a midterm exam that will be administered and graded before the course resign date. Midterm material will cover all lecture and reading assignments before the exam, as well as

concepts from homework and lab assignments. The final exam is comprehensive, covering all lecture, lab, and homework areas. Make-up exams are not administered! If you miss the midterm exam or final, you will be assigned a grade of 0 points for that component.

Lecture and Recitation Attendance

Attendance is required for all lectures and assigned recitations. You are responsible for all materials presented in lecture and recitation, as well as handouts and/or other supplemental material. I do not give **incomplete**s in the course, unless under the most dire of circumstances. By definition, an incomplete is warranted if the student is capable of completing the course satisfactorily, but some traumatic event has interfered with their capability to finish within the timeframe of the semester.

M. Academic Integrity:

UB's definition of Academic Dishonesty in part is, "Students are responsible for the honest completion and representation of their work". You should also read the departmental academic honesty policy located at http://www.cse.buffalo.edu/academics-academic_integrity.shtml.

You must abide by the UB Academic Integrity policy at all times. Remember that items taken from the web are also covered by the academic honesty policy. If you are caught violating the academic integrity policy, you will minimally receive a ZERO in the course. We will also place the incident in your permanent record. If it is your second violation, we will recommend to the Undergraduate/Graduate committee Chair that formal proceedings be filed against you, which would mean either you could be expelled, or your degree progress will be terminated within the Computer Science and Engineering department.

N. Help:

When asking questions, please try and talk with the TA first. He has probably covered the question many times with other students. Take advantage of my office hours and theirs. We have about 6 hours amongst us. Attend the recitations regularly.

If you have special needs due to a **disability,** and are registered with the Office of Disability Services, we need to know as soon as possible! Do not assume that we have received the paperwork! (Although it is your responsibility to make sure we receive the paperwork as soon as possible from Disability Services).

O. TOPICS

1. Introduction to enterprise systems — 1 week
   - 1.1 Client server systems
   - 1.2 Enterprise computing technology landscape
   - 1.3 Application models
   - 1.4 Naming Service
   - 1.5 Location transparency
2. Component programming — 2 weeks
   - 2.1 Enterprise Java bean (EJB)
   - 2.2 Session bean and entity beans
   - 2.3 Resources and access methods
3. Introduction Java 2 Enterprise edition — 1 week
   - 3.1 J2EE application model
   - 3.2 Web tier
   - 3.3 EJB tier
   - 3.4 Resources Tier
4. Development environment Macromedia's JRun4 — 1 week
   - 4.1 Application model
   - 4.2 Environment context
   - 4.3 Java Naming and Directory Interface (JNDI)
   - 4.4 Resources (database, message queue)
5. Introduction to Grid Computing — 1 week
   - 5.1 Grid in context
   - 5.2 Scientific application domain
   - 5.3 Industrial application domain
6. Grid Architecture (anatomy) and Functions (physiology) — 3 weeks
   - 6.1 Web Services framework
   - 6.2 Open Grid Services Architecture
   - 6.3 Open Grid Services Infrastructure
   - 6.4 Globus toolkit
   - 6.5 Grid application model
   - 6.6 Grid Core Tutorial and Globus installation
7. Virtual Organization concept — 1 week
   - 7.1 Virtualization
   - 7.2 Federation
   - 7.3 Provisioning
   - 7.4 Logging
   - 7.5 Events and Notification
   - 7.6 Service Data
8. Resource and service management — 1 week
   - 8.1 Resource Descriptions
   - 8.2 Resource management framework
   - 8.3 Resource discovery and selection
   - 8.4 Policies and scheduling
   - 8.5 Resource brokers

9. Reliable Grid clients and services	1 week
9.1 A service-oriented architecture	
9.2 Data-intensive applications	
9.3 Data integration and management services	
10. Instrumentation and monitoring	1 week
10.1 Event monitoring	
10.2 Instruments for monitoring	
10.3 Sensor and sensor management	
10.4 Performance diagnosis	
10.5 Network weather service	
11. Security for Virtual organizations	1 week
11.1 Security requirements	
11.2 Dynamic trust domain	
11.3 Emerging security technologies	
11.4 Certification Authority	
11.5 Webservices security	

We designed and developed large scale distributed systems using three-tier client-server acrhitectures, component programming, J2EE application model and grid technology. Two representative large scale distributed problems are described below. Study the description and provide your software solution for the problems. Your response to each problem will have these components:

a. (10 %) A block diagram showing your understanding of the overall system providing the various organizations involved.

b. (10 %) Anlayze the problem and provide the requirements in the form of a use case diagram.

c. (30 %) Design the client-server (3-tier) systems and the class diagrams for the the business logic.

d. (40 %) Assume thst we are implmenting the system in J2EE. Describe the application model using the JRun4 application directory structure. Describe the resources needed and their access methods.

e. (10 %) Give a stepwise description of how you will deploy the system and use the services offered by the system (web access etc.).

Do not write an essay. Try to represent as many details as possible in the diagrams themselves. Provide brief explanation or justification for the design decisions you make.

Question 1: Application domain: Enterprise Integration [40 points]

A large furniture store chain TomVille Inc. has many different types of furniture (Ex: chairs, tables, bedroom sets etc.) in its product line. But all the processes in the business, sales, order entry, inventory control, shipping, buying and corporate planning use manually entered and maintained data. The management of TomVille wants to computerize their operations and also want to provide an online store front end for customers to place their orders online. Orders received in a day are collected to generate a production order (message) for the production systems. The monthly data on orders is archived in a warehouse for corporate office o generate reports for planning. Orders are also used to collect the furniture ordered and ship them to the appropriate address. Email confirmation is sent to the customer at each stage of order flow.

Question 2: application domain: Large scale simulation [60 points]

KayJay World is a small vacation theme park that operates a circular, six-train mono-rail system connecting a parking lot, a theme park, a hotel, and a concert hall/restaurant complex. There are four stations, each with a ticket booth and a boarding queue. Passengers obtain tickets for one of the possible destination stations and enter the boarding queue. The boarding queue is arranged in such a way that every seat on a train will be filled up before any passenger can be left at a station. Trains consist of an engine and one to six-passenger cars, each car with carrying capacity of 50 passengers.

The rail system consists of fifteen safety segments of track and each train in service occupies one of these segments. A train may not enter a segment occupied by another train. Each station counts as one segment, and each link between stations is divided into two to three segments. There is a barn capable of storing all six trains with two access segments: one exit segment leading to the barn and yard, and one entrance segment departing from the barn and yard. There are other details such as junction switch and fixed junction, which we will not worry about in this context.

The capacity manager initializes the system by ordering one train to leave the barn and thus be placed in service. At least one train remains in service until the capacity manager shuts down the system ordering the last train out of service. The capacity manager is in charge of adding or removing trains from the system depending on the crowd and using a Predetermined policy. There is a yard manager, which checks the trains and declares them operable or inoperable. Inoperable trains will go into service yard and get repaired.

There are station managers to take care the passengers, ticketing and queuing. Kayjay World is completely automated system. That is, computers manage all the operations. You have been assigned the task of simulating the KayJay World (Enterprise) on a computer so that the problems in automating such an elaborate system can be checked out before physically building the system.

Analyze and come up with a complete design for your simulation. Here are some hints for you get going:

a. Identify the modules in the application.

b. For each module identify the services offered.

c. For each module identify users (clients).

d. For each module identify entities, processes, and rules.

e. Provide a class diagram for each module, which shows the classes and relationships among the classes. Give the classes meaningful names

f. Define each of the classes: properties and behavior.

g. Provide list of data that is to be persisted.

h. Partition the features into client-side, middleware and server-side ones.

i. Provide an algorithm explaining the simulator that will control all the modules.

STATE UNIVERSITY OF NEW YORK AT BUFFALO
DEPARTMENT OF COMPUTER SCIENCE

Name, Last: _____ First: _____ M.I.: ____

Person Number: _____ Section: _____

CSE487/587 Information Structures
Spring 2004
Exam 1
Instructor: Bina Ramamurthy

Time: 75 minutes

Instructions:

1. This exam consists of 2 questions, and 5 pages. PLEASE CHECK.
2. Write your name, person number and course section number on this page NOW.
3. This exam is OPEN BOOK, OPEN NOTES, and CLOSED NEIGHBORS.
4. For full credit, show ALL of your work, not just the results.
5. DO NOT WRITE AN ESSAY. BE PRECISE. IF YOU DO NOT FOLLOW THIS RULE I WILL DEDUCT 10 POINTS FROM YOUR FINAL SCORE.
6. Write your answers in the spaces provided. You may use the backs of pages if you find it necessary.
7. Raise your hand if you have any questions.
8. There are severe penalties for academic dishonesty.
9. Please RETURN the ENTIRE EXAM before leaving the room.

Question	Points	Grade
1	40	_____
2	60	_____
Total	100	_____

Course Evaluation

CSE 487: Information Structures

This course evaluation is part of an effort to evaluate the courses that are being developed as part of a grant from the National Science Foundation. Your participation in this course evaluation will provide important information to help improve the course. In addition, your comments will benefit students taking this course in the future.

We appreciate your taking the time to read each question carefully and answer them as fully as possible.

Instructions for Completing the Course Evaluation

- Do not put your name on any form. Survey responses will remain anonymous.
- Please respond to items 1–42 on this survey by circling the appropriate number. Responses to items 43–46 should be reported in the spaces provided.
- When you have completed the survey, please place the forms in the envelope supplied by your instructor.

Course Evaluation Student Questionnaire
CSE 487 — Information Structures — Spring 2004

Please respond to of the following questions by circling the number between one and five which most nearly represents your feelings. As indicated below, we use the scale: (1) Strongly Agree, (2) Agree, (3) Neutral, (4) Disagree, (5) Strongly Disagree. **Please read each question carefully**.

Course Information Please indicate the degree to which you feel	**Strongly Agree**	**Agree**	**Neutral**	**Disagree**	**Strongly Disagree**
1. the objectives of this course were clearly stated.	1	2	3	4	5
2. this course increased your interest in enterprise (Ex: J2EE) systems.	1	2	3	4	5
3. this course increased your interest in grid computing.	1	2	3	4	5
4. you learned a lot about enterprise systems, including both concepts and implementation.	1	2	3	4	5
5. you learned a lot about grid computing and its future potential.	1	2	3	4	5
6. adequate time was allotted to cover the course content.	1	2	3	4	5
7. the topic areas were sequenced in an appropriate manner.	1	2	3	4	5
8. the instructions for exercises and assignments were clear and easy to understand.	1	2	3	4	5
9. the lab exercises and assignments reflected the content of the course.	1	2	3	4	5
10. the lab exercises and assignments helped you learn the course material.	1	2	3	4	5
11. the grading of the lab exercises and assignments was fair.	1	2	3	4	5
12. the questions on tests reflected the content of the course.	1	2	3	4	5
13. the grading of the tests was fair.	1	2	3	4	5
14. adequate time was given to complete the tests.	1	2	3	4	5
15. the textbook was helpful and a good information resource.	1	2	3	4	5
16. the textbook, course materials and handouts were sufficient for you to understand all the topics covered.	1	2	3	4	5
17. the course website was useful for obtaining course materials and information.	1	2	3	4	5
18. the instructor or TA provided help when you needed it.	1	2	3	4	5
19. you are prepared for applying grid concepts to research and development.	1	2	3	4	5
20. the topics covered will be useful to you in the future, beyond CSE 487-587.	1	2	3	4	5
21. the course met your expectations.	1	2	3	4	5
22. Overall, how would you rate this course? (1=excellent, 2= good, 3=average, 4=poor, 5=bad)	1	2	3	4	5

Course Objectives Please indicate the degree to which you feel you	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
23. understand the fundamental components and operation of an enterprise system (J2EE).	1	2	3	4	5
24. can design and implement an enterprise application.	1	2	3	4	5
25. are able to analyze a distributed system for its architecture, algorithms, protocols and services.	1	2	3	4	5
26. have good understanding and working knowledge of grid services and grid computing.	1	2	3	4	5
27. are able to program using Enterprise Java Beans(EJB).	1	2	3	4	5
28. have good understanding and working knowledge of the components of Grid Services architecture (Ex: OperationProvider, ServiceData etc.)	1	2	3	4	5
29. have good understanding and the working knowledge of the Grid Services infrastructure (Ex: Notification Service, Logging service etc.)	1	2	3	4	5
30. have a good understanding Virtual Organization concept.	1	2	3	4	5
31. are able to program using Grid Services and Globus core.	1	2	3	4	5
32. are able to program using the Globus grid computing framework.	1	2	3	4	5
33. are able to demonstrate the ability to design, implement, and deploy distributed systems based on Java technology and Grid Technology.	1	2	3	4	5

Computer Resources (Hardware and Software) Please indicate the degree to which you feel	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
34. the type of hardware computer resources provided by UB were appropriate for the course.	1	2	3	4	5
35. the type of software computer resources provided by UB were appropriate for the course.	1	2	3	4	5
36. the computer resources provided by UB were adequate to do the lab exercises and assignments.	1	2	3	4	5
37. the computer resources were available and accessible when you needed or wanted to use them.	1	2	3	4	5
38. the computer resources enabled you to gain "hands on" experience with distributed systems.	1	2	3	4	5
39. the computer resources enabled you to gain "hands on" experience with grid computing.	1	2	3	4	5
40. able to work with Condor grid (CSECCR grid) supported by CSE and CCR.	1	2	3	4	5
41. able to work with Linux grid supported by CSE department.	1	2	3	4	5

Application Development Environment and Tools

Please indicate the degree to which you feel	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
39. use of development environment (Jrun4) was helpful in developing J2EE-based enterprise applications.	1	2	3	4	5
40. used deployment description and container managed resources in JRun.	1	2	3	4	5
41. In general, development environment similar to JRun4 will streamline development of grid services.	1	2	3	4	5
42. A graphical grid development environment will provide systematic approach to designing grid services.	1	2	3	4	5
43. able to understand and use declarative features over programmatic alternatives (ex: JDBC) where ever applicable.	1	2	3	4	5
44. able to understand and use Apache Ant tool.	1	2	3	4	5

Please take the time to answer each of the following questions.

43. Why did you take this course?

44. What was the most valuable aspect of the Information Structures course? What did you like about it?

45. What was the poorest aspect of the course? In what ways could this course be improved?

46. What other comments would you like to make regarding any aspect of this course?

Enterprise Computing: An Overview

B. Ramamurthy

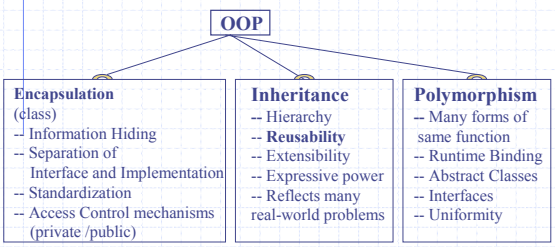
Introduction

- In this lecture we will trace through all the important developments leading to enterprise computing.
- During this process I will review many fundamental concepts such as object-oriented principles and request-reply model, distributed objects, remote method invocations, Java technology etc.
- Your task is to identify the concepts that you further need to study and work on them in the next two weeks.
- Those who are familiar with any of the concepts, share your experiences with the students in the class.

Topics of Discussion

- Object-Orientation (OO) Principles
- Unified Modeling Language (UML)
- Beyond objects
- Enterprise systems
- Middleware
- J2EE Components and Application Model

Object-Oriented Principles (OOP)



Why OO paradigm?

- OO Models let you structure your thoughts.
- Convenient for large software development
- Systematic approach to analyzing large problems
- Reuse through classes and inheritance
- Supports Application programmer Interface (API) concept
- Standardization (standard interface)
- Facilitates security , protection and access control

Unified Modeling Language

The Unified Modeling Language™ (UML) was developed jointly by Grady Booch, Ivar Jacobson, and Jim Rumbaugh with contributions from other leading methodologists, software vendors, and many users. The UML provides the application modeling language for:

- •Business process modeling/ Requirement Analysis with use cases.
- •Static Design with Class modeling and object modeling.
- •Dynamic Design with sequence, collaboration and activity diagrams.
- •Component modeling.
- •Distribution and deployment modeling.

•See

http://www.rational.com/uml/resources/whitepapers/index.jsp
http://www.cetus-links.org/oo_uml.html

Phases of System Development

- Requirement Analysis
  - Functionality users require from the system
  - Use case model
- OO Analysis
  - Discovering classes and relationships
  - UML class diagram
- OO Design
  - Result of Analysis expanded into technical solution
  - Sequence diagram, state diagram, etc.
  - Results in detailed specs for the coding phase
- Implementation (Programming/coding)
  - Models are converted into code
- Testing
  - Unit tests, integration tests, system tests and acceptance tests.

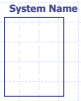
Use-Case Modeling

- In use-case modeling, the system is looked upon as a black box whose boundaries are defined by its functionality to external stimulus.
- The actual description of the use-case is usually given in plain text. A popular notation promoted by UML is the stick figure notation.
- We will look into the details of text representation later. Both visual and text representation are needed for a complete view.
- A use-case model represents the use-case view of the system. A use-case view of a system may consist of many use case diagrams.
- An use-case diagram shows (the system), the actors, the use-cases and the relationship among them.

Components of Use Case Model

- The components of a use case model are:
  - Use cases 

  - Actors 

  - System Modeled 

  - Stimulus

System

- As a part of the use-case modeling, the boundaries of the system are developed.
- System in the use-case diagram is a box with the name appearing on the top.
- Defining a system is an attempt to define the catalog of terms and definitions at an early stage of the development of a business model.

Actors

- An actor is something or someone that interacts with the system.
- Actor communicates with the system by sending and receiving messages.
- An actor provides the stimulus to activate an use case.
- Message sent by an actor may result in more messages to actors and to use cases.
- Actors can be ranked: primary and secondary; passive and active.
- Actor is a role not an individual instance.

Finding Actors

- The actors of a system can be identified by answering a number of questions:
  - Who will use the functionality of the system?
  - Who will maintain the system?
  - What devices does the system need to handle?
  - What other system does this system need to interact?
  - Who or what has interest in the results of this system?

Use Cases

- A use case in UML is defined as a set of sequences of actions a system performs that yield an observable result of value to a particular actor.
- Actions can involve communicating with number of actors as well as performing calculations and work inside the system.
- A use case
  - is always initiated by an actor.
  - provides a value to an actor.
  - must always be connected to at least one actor.
  - must be a complete description.
- Example?

Finding Use Cases

- For each actor ask these questions:
  - Which functions does the actor require from the system?
  - What does the actor need to do?
  - Could the actor's work be simplified or made efficient by new functions in the system?
  - What events are needed in the system?
  - What are the problems with the existing systems?
  - What are the inputs and outputs of the system?

Classes

- OO paradigm supports the view that a system is made up of objects interacting by message passing.
- Classes represent collection of objects of the same type.
- An object is an instance of a class.
- A class is defined by its properties and its behaviors.
- A class diagram describes the static view of a system in terms of classes and relationships among the classes.

Discovering Classes

- Underline the nouns in a problem statement.
- Using the problem context and general knowledge about the problem domain decide on the important nouns.
- Design and implement classes to represent the nouns.
- Underline the verbs. Verbs related to a class may represent the behavior of the class.
- You can also discover the classes from the use case diagram.

Designing Classes

- A class represents a class of objects.
- A class contains the data declarations ("parts") and methods ("behaviors" or "capabilities").

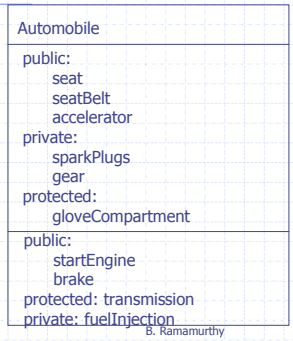
OO Design:

- Class properties or characteristics are answers to "What is it made of?" (It **has a** ____, ____, etc.)
- Behaviors, capabilities or operations are answers to "What **can** it **do**?" (verbs in the problem)

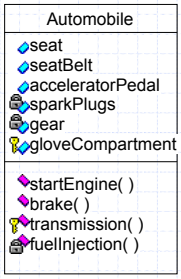
Classes are Blueprints

- A class defines the general nature of a collection of objects of the same type.
- The process creating an object from a class is called instantiation.
- Every object is an instance of a particular class.
- There can be many instances of objects from the same class possible with different values for data.
- A class structure implements encapsulation as well as access control: private, public, protected.

Class Diagram : Automobile



Automobile Class Using Rational Rose Tool



On to implementation

- You may define the methods of the class using sequence diagram and state diagram.
- Using these diagrams you can code the application.

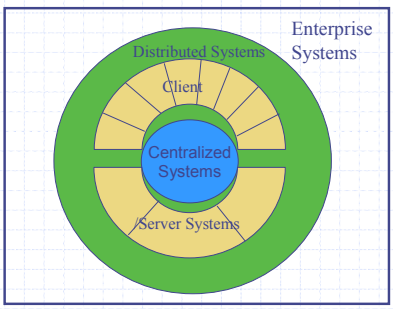
Beyond Objects

- Issues: Basic object-technology could not fulfill the promises such as reusability and interoperability fully in the context internet and enterprise level applications. Deployment was still a major problem and as a result portability and mobility are impaired.
- Solution: Middleware
- Common Object Request Broker Architecture (CORBA), Java 2 Enterprise Edition, .NET, computation grid

Enterprise Systems

- An enterprise is a very large organization.
- An enterprise system is a distributed system involving many large organizations.
- An example: AT&T, inktomi, amazon.com, UPS, and users operating in a supply chain model, make up an enterprise system.
- Inter .com ….

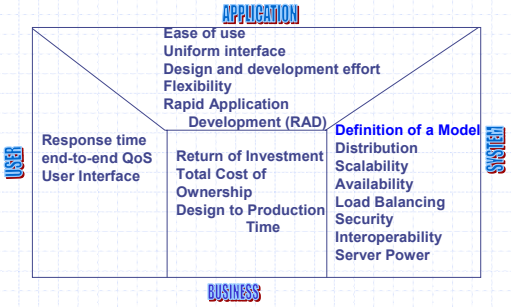
Evolution of Computing Systems



Distributed System as an Enterprise System

- There are many problems in using traditional distributed system model for enterprise computing. Look at "A Note on Distributing Computing" by Jim Waldo, Geoff Wyant, Ann Wollarth and Sam Kendall of Sun labs.
  -- current distributed system paradigm works well for small systems dealing with single address space but fails very badly for dynamically changing global address spaces.
- We have seen advances in code mobility, data mobility,etc. But the distributed system architecture/principles are yet to evolve in any significant way.
- Focus on distribution.

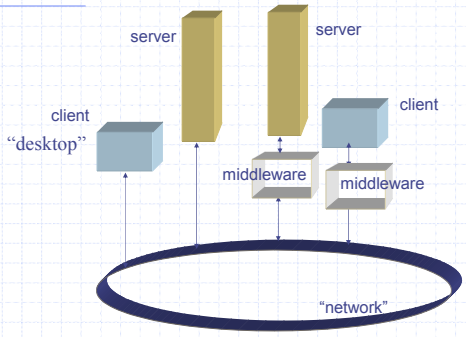
Issues in Enterprise Systems



Requirements for Enterprise Computing

- Accommodate changes gracefully - **scalability, dynamic reconfiguration**
- Maintain high **availability** at all times
- Offer good performance in terms of response time and end-to-end "QOS"
- Dependability and fault tolerance
- Simplicity
- ....

Enabling Technology



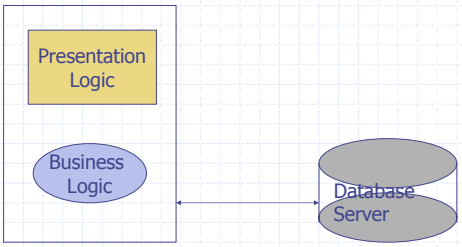
Middleware (as defined by NSF)

- Middleware refers to the software which is common to multiple applications and builds on the network transport services to enable ready development of new applications and network services.
- Middleware typically includes a set of components such as resources and services that can be utilized by applications either individually or in various subsets.
  - Examples of services: Security, Directory and naming, end-to-end quality of service, support for mobile code.
- OMG's CORBA defines a middleware standard.
- J2EE Java 2 enterprise edition is a middleware specification.
- Compute grid is middleware framework.

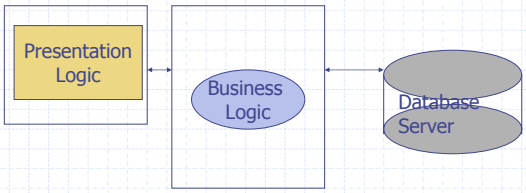
Component Technology

- We need an application architecture that works well in the new E-commerce age.
- Programmer productivity, cost-effective deployment, rapid time to market, seamless integration, application portability, scalability, security are some of the challenges that component technology tries to address head on.
- Enterprise Java Beans is Sun's server component model that provides portability across application servers, and supports complex systems features such as transactions, security, etc. on behalf of the application components.
- EJB is a specification provided by Sun and many third party vendors have products compliant with this specification: BEA systems, IONA, IBM, Oracle.

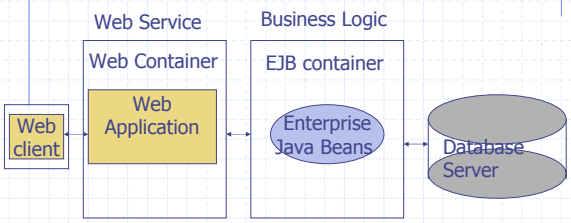
Two-tier applications



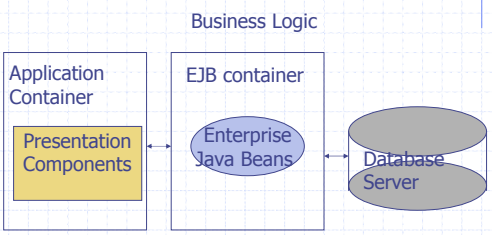
Three-tier Applications



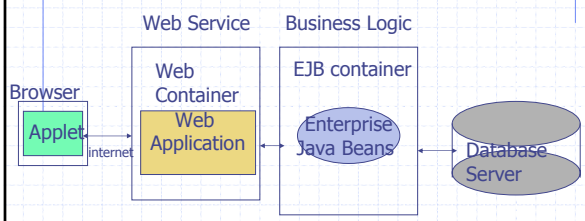
J2EE Application Programming Model for Web-based applications



J2EE Application Programming Model for Three-tier Applications



J2EE Application Programming Model for Web-based Applets



J2EE Application Model

- Study the introduction and the application model detailed in the discussion at the following URL:
  - Introduction to J2EE
  - Application Model
  - Components of J2EE

Distributed System Using Java 2 Enterprise Edition (J2EE)

B.Ramamurthy

Introduction

- Sun Microsystems provides specifications for a comprehensive suite of technologies to solve large scale distributed system problems.
- This suite is the Java 2 Enterprise Edition, commonly known as J2EE.
- In this discussion we will discuss the architecture of J2EE and how it can be used to develop distributed multi-tiered applications.
- This discussion is based on the tutorial by Sun Microsystems Inc.

J2EE Suite

- Core technology: Container infrastructure, language and environment support
- XML technology
  - The Java API for XML Processing (JAXP)
  - The Java API for XML-based RPC (JAX-RPC)
  - SOAP with Attachments API for Java (SAAJ)
  - The Java API for XML Registries (JAXR)
- Web Technology
  - Java Servlets
  - JavaServer Pages
  - JavaServer Pages Standard Tag Library
- Enterprise Java Bean (EJB) technology
  - Session beans
  - Entity beans
    - Enterprise JavaBeans Query Language
  - Message-driven beans
- Platform services
  - Security
  - Transactions
  - Resources
  - Connectors
  - Java Message Service

Distributed Multi-tiered Applications

- Services, clients (people and application) and data are distributed geographically across many platforms and many machines.
- Multiple tiers:
  - Client-tier (browser or client-application)
  - Web-tier (web-server: Java Server Pages)
  - Business-tier (logic; Examples: Enterprise Java Beans)
  - Enterprise-Information-System (EIS) tier (database)

J2EE clients

- Web clients
  - Dynamic web pages with HTML, rendered by web browsers.
  - Can include applets.
  - Communicates with server typically using HTTP.
- Application clients
  - User interface using GUI components such as Swing and AWT.
  - Directly accesses the business logic tier.

Web-tier Components

- Client can communicate with the business tier either directly or through servlets ot JSP that are located in the web-tier.
- Web-tier can help in pre-processing and allows distribution of the functionality.
- See Figure 2-1
- Servlets are special classes to realize the request-response model (get, post of HTTP).
- JSP is a developer-friendly wrapper over the servlet classes.

Business-tier Components

- This is defined by the logic that pertains to the (business) application that is being developed.
- Enterprise Java Beans (EJB) can be used to implement this tier.
- This tier receives the data from the web-tier and processes the data and sends it to the EIS-tier and takes the data from the EIS and sends it to the web-tier.
- See Figure 1-3, and Figure 1-4

Enterprise Java Beans

- Session beans
- Entity Beans
  - Bean-managed Persistence (BMP)
  - Container-managed Persistence (CMP)
  - Enterprise Javabeans Query Lanaguage
- Messaging Bean
  - Session bean with Java Messaging features

Session Beans

- For transient functions
- Represents “conversational” state
- Typically one per request
- Data is non-persistent
- Lifetime is limited by the client’s: once the client exits, the session bean and data are gone.
- Simple and easy to program.
- Light-weight.

Entity Bean

- “Transactional” in behavior
- Can be shared among clients
- Persistent: data exists permanently after client quits.
- Corresponds to a row a relational database.
- The persistence (storing into the database) can be automatically done by the “container” (CMP) or explicitly by the bean (BMP)

Enterprise Information System (EIS) Tier

- In general this corresponds to the database (relational database) and other information management system.
- The other information management systems may include Enterprise Resource Planning (ERP) and legacy system connected through open database connectivity.

Container Services

- A container interfaces the programmatic components such as EJBs to the declarative components.
- Container services include security, transaction management, naming services, and remote connectivity.
- The fact that the J2EE architecture provides configurable services means that application components can behave differently based on where they are deployed.
- The concept of “deployable units” and “containers” where they can be deployed is central to J2EE.

Enterprise Java Beans

Bina Ramamurthy

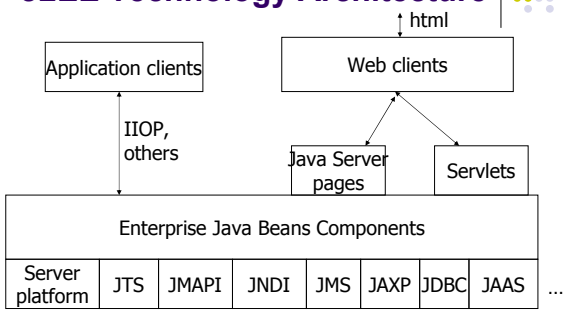
Introduction

- J2EE (Java2 Enterprise Edition) offers a suite of software specification to design, develop, assemble and deploy enterprise applications.
- It provides a distributed, component-based, loosely coupled, reliable and secure, platform independent and responsive application environment.

J2EE-based Application

- The J2EE APIs enable systems and applications through the following:
  - Unified application model across tiers with enterprise beans
  - Simplified response and request mechanism with JSP pages and servlets
  - Reliable security model with JAAS
  - XML-based data interchange integration with JAXP
  - Simplified interoperability with the J2EE Connector Architecture
  - Easy database connectivity with the JDBC API
  - Enterprise application integration with message-driven beans and JMS, JTA, and JNDI

J2EE Technology Architecture



Enterprise Java Bean(EJB)

- An *enterprise bean* is a server-side component that contains the business logic of an application. At runtime, the application clients execute the business logic by invoking the enterprise bean's methods.
- Main goal of Enterprise Java Bean (EJB) architecture is to free the application developer from having to deal with the system level aspects of an application. This allows the bean developer to focus solely on the logic of the application.

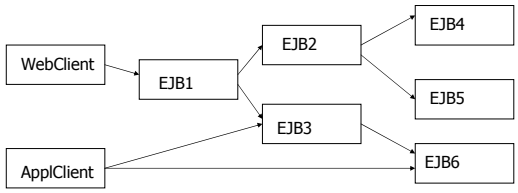
Roles in EJB Development

- Bean developer: Develops bean component.
- Application assembler: composes EJBs to form applications
- Deployer: deploys EJB applications within an operation environment.
- System administrator: Configures and administers the EJB computing and networking infrastructure.
- EJB Container Provider and EJB server provider: Vendors specializing in low level services such as transactions and application mgt.

Enterprise Java Bean (EJB)

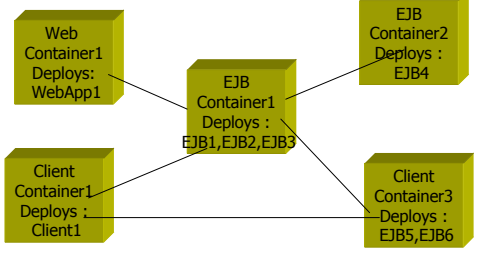
- Deployable unit of code.
- At run-time, an enterprise bean resides in an EJB container.
- An EJB container provides the deployment environment and runtime environment for enterprise beans including services such as security, transaction, deployment, concurrency etc.
- Process of installing an EJB in a container is called EJB deployment.

Enterprise Application with many EJBs



Lets consider a shopping front application and figure out the possible components (EJBs)

Deployment with Multiple EJB Clients



Business Entities, Processes and Rules

- EJB Applications organize business rules into components.
- Components typically represent a business entity or business process.
- Entity: is an object representing some information maintained in the enterprise. Has a “state” which may be persistent.
- Example: Customer, Order, Employee,
- Relationships are defined among the entities: dependencies.

Process

- Is an object that typically encapsulates an interaction of a user with business entities.
- A process typically updated and changes the state of the entities.
- A business process may have its own state which may exist only for the duration of the process; at the completion of the process the state ceases to exist.
- Process state may be transient or persistent.
- States ate transient for conversational processes and persistent for collaborative processes.

Rules

- Rules that apply to the state of an entity should be implemented in the component that represents the entity.
- Rules that apply to the processes should be implemented in the component that represents the processes.

EJB Types

- There are three types of EJBs:
  - Entity, session and message-driven
- We will discuss message-driven bean in a separate discussion.
- The syntax of the session bean and entity bean client-view API is almost identical.
- But they have different life cycle, different persistence management, etc.
- EJBs can be stateless and stateful beans.

Life Cycle Differences

Session Bean	Entity Bean
Object state: Maintained by container	Maintained by DB
Object Sharing: No sharing: per client	Shared by multiple client
State Externalization: State is inaccessible to other programs	Accessible to other programs
Transactions: Not recoverable	State changed transactionally and is recoverable.
Failure Recovery: Not guaranteed to survive failures	Survives failures and restored when the container restarts.

Choosing Entity or Session Bean

- Entity (business entity) is typically implemented as entity bean or a dependent object of an entity bean.
- Conversational (business) process as a session bean.
- Collaborative bean as an entity bean.
- Any process that requires persistence is implemented as an entity bean.
- When exposure to other applications are not needed for an entity or process (local/private process) then they are implemented as bean dependent objects.

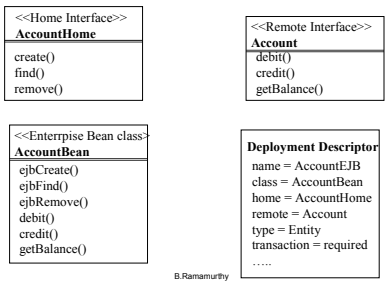
Parts of EJB

- EJB class that implements the business methods and life cycle methods; uses other helper classes and libraries to implement.
- Client-view API: consists of EJB home interface and remote interface.
  - Home interface: controls life cycle : create, remove, find methods
  - Remote interface: to invoke the EJB object methods

Parts of EJB (contd.)

- Deployment Descriptor: XML document for bean assembler and deployer;
  - A declaration about EJB environment needed for customizing the bean to the operating environment.
- Container Runtime services include: transactions, security,distribution,load balancing, multithreading, persistence, failure recovery, resource pooling, state management, clustering..

Enterprise Bean Parts



AccountHome Interface

```
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.FinderException;
import java.util.Collection;

public interface AccountHome extends javax.ejb.EJBHome {
//create methods
Account create (String lastName, String firstName) throws RemoteException,
                                CreateException, BadNameException;
Account create (String lastName) throws RemoteException, CreateException;

// find methods
Account findByPrimaryKey (AccountKey primaryKey) throws RemoteException,
                                FinderException;
Collection findInActive(Date sinceWhen)
      throws RemoteException, FinderException, BadDateException;
```


Account Interface

```
import java.rmi.RemoteException;

public interface Account extends javax.ejb.EJBObject {

BigDecimal getBalance() throws RemoteException;

void credit(BiGDecimal amount) throws RemoteException;

Void debit(BigDecimal amount) throws RemoteException,
                                InSufficientFundsException;
}
```


AccountBean class

```
public class AccountBean implements javax.ejb.EntityBean {
// life cycle methods from home interface
public AccountKey ejbCreate (String latName, String firstName) throws ... {
public AccountKey ejbCreate(String lastName) throws ...{...}
public AccountKey ejbFindByPrimaryKey(AccountKey primarykey)... {...}
Public Collection ejbFindInactive( Data sinecWhen).. {...}

// business methods from remote interface
public BigDecimal getBalance() {....}
public void credit(BigDecimal amt) {...}
Public void debit(BigDecimal amt) throws InsufficientFundException {....}

// container callbacks from EntityBean container
public ejbRemove( ) throws RemoveException{ ...}
public void setEntityContext(EntityContext ec) {...}
public unsetEntityContext(EntityContext ec) {...}
public void ejbActivate() {...}
public void ejbLoad() {....}
public void  ejbStore() {....}
}
```


Deployment Descriptor

```
...
<entity-bean>
   <ejb-name>AccountEJB</ejb-name>
   <home>com.wombat.AccopuntHome</home>
   <remote>com.wombat.Account</remote>
   <ejb-class>com.wombat.AccountBean</ejb-class>
   <persistence-type>Bean<\persistence-type>
   <primary-key-class>com.wombat.AccountKey</primary-key-class>
</entity-bean>

...
<container-transaction>
   <method>
     <ejb-name>AcoountEJB</ejb-name>
     <method-name>*</method-name>
   </method>
   <trans-attribute>required</trans-attribute>
</container-transaction>
```


Compilation and Deployment

- Compilation (building the executables) uses build tool such as Apache Ant.
- The components of the various tiers are packaged: .jar, .war, .ear
- Declarative resources are added.
- A deploy tool or management tool is used to deploy the packaged units into a J2EE server (container).

Summary

- J2EE environment provides a framework for bundling together the components into an application and provide the applications necessary common services such as persistence, security, mail, naming and directory service etc.
- Next class we will look a complete running example.
- Browse through:
  - http://java.sun.com/j2ee/faq.html
  - http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/index.html#chapters
  - http://java.sun.com/developer/onlineTraining/J2EE/Intro2/j2ee.html

Software Development using MacroMedia's JRun

B.Ramamurthy

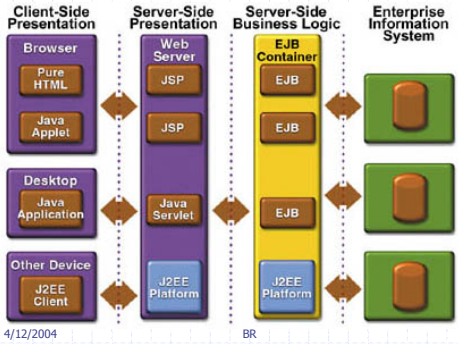
Objectives

- To study the components and working of an enterprise java bean (EJB).
- Understand the features offered by Jrun4 environment.
- To be able to deploy and execute application using JMC of Jrun4.
- Analyzing a problem and arriving at a component-based solution.

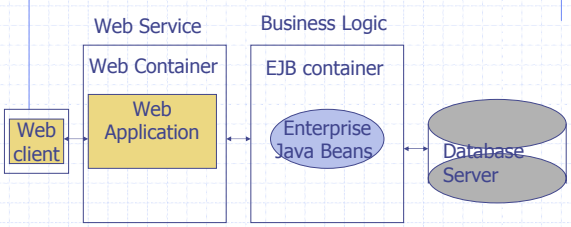
Topics for Discussion

- General introduction to Enterprise EJB
- JRUN 4 application server from Macromedia
- Demos on JRUN 4
- From problem statement to J2EE "components" via use case analysis

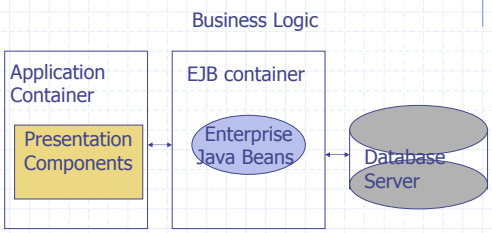
Enterprise Application Model



J2EE Application Programming Model for Web-based applications



J2EE Application Programming Model for Three-tier Applications



EJB Component Model

- **Business logic can be encapsulated in EJB components.**
- **The EJB component model simplifies the development of middleware applications by providing automatic support for services such as transactions, security, database connectivity, and more.**

What are EJBs?

- **Enterprise JavaBeans™** is the server-side component architecture for the J2EE™ platform. EJB™ enables rapid and simplified development of distributed, transactional, secure and portable Java applications.
- An EJB is a collection of Java classes, and a XML file (deployment descriptor) bundled into a single unit.
- Java classes in this bundle follow certain rules and provide specific callbacks for the containers.

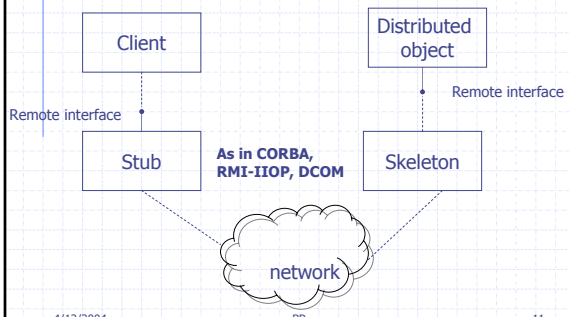
EJB Types

- There are three major types of EJBs:
  - Session: Represents **conversational**/transient state; stateless and stateful
  - Entity bean: Represents a **persistent** relation in the relational DB. Bean-managed persistence (BMP), container-managed persistence (CMP)
  - Message-driven: Alternative to remote method call: asynchronous and used for realizing loose coupling among systems. Uses **messaging** middleware.
- Lets look at Ed Roman's view of the EJB technology.

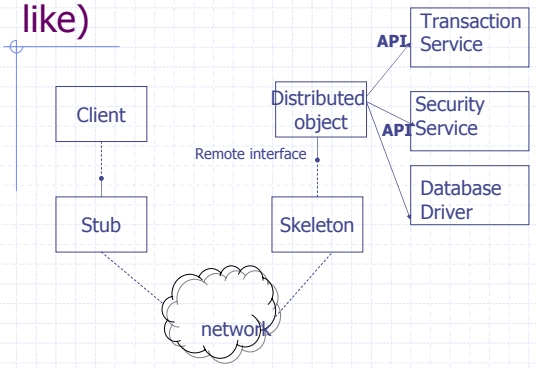
Examples of Session beans calling entity beans

Session bean	Entity bean
Bank teller	Bank account
Credit card authorizer	Credit card
Order entry form	Order, line item
Catalog engine	Product
Auction broker	Bid, item
Purchase order router	Purchase order

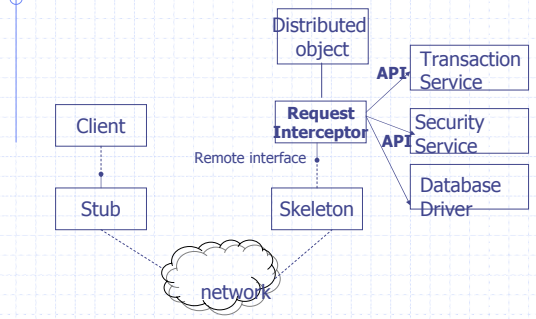
Simple Distributed Objects



Explicit Middleware (CORBA-like)



Implicit Middleware (Through declarations as in J2EE)



Implicit VS Explicit services

- We used to include the services such as transaction, security, data base drivers, etc. programmatically making every programmer learn the inner details all the possible services needed in an application.
- Now we can declare what we want and let the container take care of carrying it out.
- Container is the silent partner: container's glue code tools are responsible for transforming an enterprise into a fully managed, distributed server-side component.
- Declaration is done through a XML deployment descriptor.

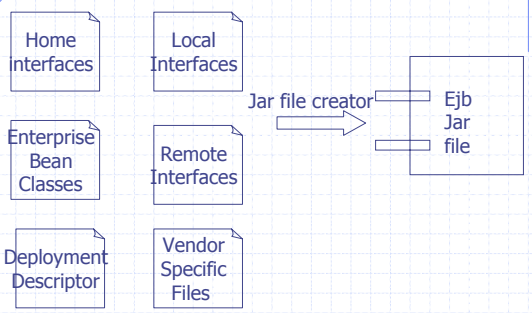
Parts of EJB

- **EJB class** that implements the business methods and life cycle methods; uses other helper classes and libraries to implement.
- **Client-view API**: consists of EJB home interface and remote interface.
  - Home interface: controls life cycle : invokes Home Object methods: create, remove, find methods
  - Remote interface: to invoke the EJB object methods

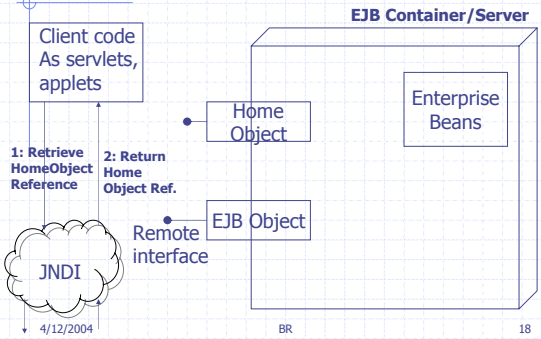
Parts of EJB (contd.)

- **Deployment Descriptor**: XML document for bean assembler and deploy tool;
  - A declaration about EJB environment needed for customizing the bean to the operating environment.
  - Container Runtime services that can be **declared** include: transactions, security,distribution,load balancing, multithreading, persistence, failure recovery, resource pooling, state management, clustering..

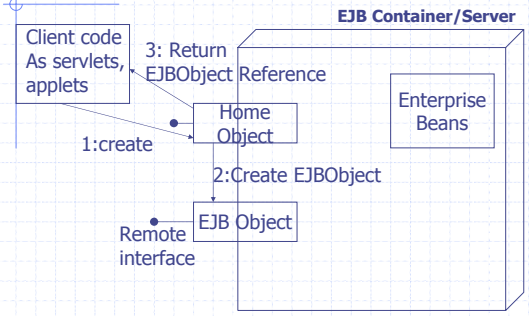
Creating a EJB-jar file



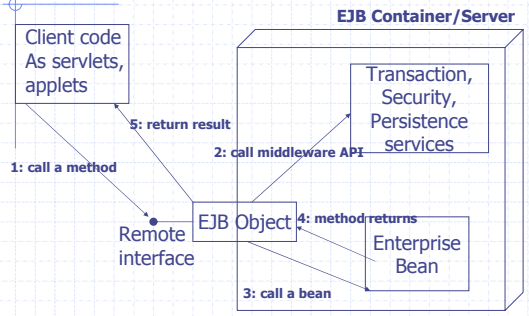
Step 1: Retrieve Home Object reference using JNDI



Step 2: Retrieve EJBObject using Home Interface and Objects



Step 3: Invoke Business Methods Using Remote Interface and EJB Objects



JRUN4

- JRun (J2EE) Server can be started, stopped, refreshed, and status checked three different ways,
  - From command line
  - Using a JLauncher
  - Using a web-based JRun Management Console (JMC)
- Demo1: Jrun4 Environment

JRun4 Development Version

- Comes with three servers: admin, default and sample
- Admin: is reserved for running administrative tools such as JMC. So you are advised not to do any application development on this. At port 8000.
- Samples: has many applications already deployed for you to study the working code for various J2EE technologies. At port 8200.
- Default: is where we will do most of our development and deployment. At port 8100.
- Demo2: Lets study the application "compass" served by the "samples" server.

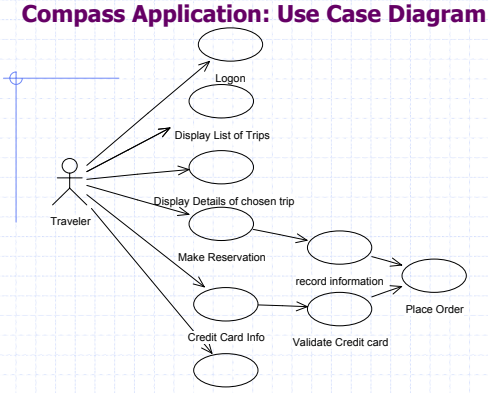
Demo 3: Add a server "tutorial" at port 8101

- Add a server tutorial. We can do hot deployment by copying over the compass application.
- Also see how the data access to the pointbase data base is declaratively added to the server using the JMC.
- Look around the other features offered by JMC.
- Observe how easy it is to delete, refresh, and stop a server using the various iconized buttons.
- Study the explorer window on the left pane of JMC to see the various declarative customization possible for your applications.

Compass Online Vacation Reservation System

1. User **logon** for authentication using a registered user id and password.
2. Application **home** provides a list of trips you can choose from. Click the name of a trip to get details about that trip
3. **Trip details** provides details about the selected trip. Click the book button to book the trip.
4. **Reservation** allows you to enter payment information. Select a credit card type, specify a credit card number and the expiration date.
5. **Confirmation** displays your confirmation number if the reservation was successful, and an error message if a problem occurred.

Compass Application: Use Case Diagram



Compass Application: From Use Cases to Component List

- Use JSP for all web user interface: Logon.jsp, home.jsp, Triplist.jsp, Atrip.jsp, Reservation.jsp (includes confirmation/denial use case), creditcard.jsp
- Data access for non-conversational JSP can be direct.
- For reservation and credit card we have a conversation with the user, so will have a stateless session bean ReservationBean and CreditCardBean. These two are **remote**ly accessible beans.
- Finally all the information gets conveyed to a entity bean OrderBean for storing the order information. This could be a "local" bean not (remotely) accessible to the client.

Business Entities, Processes and Rules

- EJB Applications organize business entities, processes and rules into components.
- Entity: is an object representing some information maintained in the enterprise. Has a "state" which may be persistent.
  - Example: Customer, Order, Employee
- Process: Is an object that typically encapsulates an interaction of a user with business entities. A process typically updated and changes the state of the entities.
  - Example: Sales, pricing, place order, reservation
- Rules: constraints on the state of the entities.
  - Example: overDraft, creditLow, validity

Choosing the type of Bean

- Entity (business entity) is typically implemented as entity bean or a dependent object of an entity bean.
- Conversational (business) process as a session bean.
- Collaborative bean as an entity bean.
- Any process that requires persistence is implemented as an entity bean.
- When exposure to other applications are not needed for an entity or process (local/private process) then they are implemented as bean dependent objects. You may use local EJBs for this purpose if container services are needed.

Review

- We studied the basics of Enterprise Java Beans. We will develop on these concepts further in the next lectures.
- We also looked JRUn4 environment: its JLauncher, JRun Management Console (JMC), and servers and deployment of applications.
- We looked at how to analyze a problem to arrive at a set of components (web components and different types of ejb components).

On To EJBs

- Understand the parts of the EJBs
- Package the EJBs and deploy them
- Design web application to access the EJBs
- Understand the various descriptors and directory structure
- Understand local naming conventions and JNDI naming conventions

Designing Components

- Designing components: esp. enterprise java beans: session beans: stateless and stateful.
- Connecting web component to an EJB.
- Enterprise application (ear) directory structure and naming conventions; hot deploy.
- XYZ-INF : META-INF, WEB-INF, SERVER-INF, web.xml, ejb-jar.xml, jrun.xml.
- Analyzing compass application of the samples server; JNDI and java naming.

Contents of an Enterprise Bean

- Interfaces: The remote and home interface for remote access. Local and local home accesses for local access.
- Enterprise bean class: Implements the methods defined in the above interfaces.
- Deployment descriptor: An XML file that specifies information about the bean such as its type, transaction attributes, etc.
- Helper classes: non-bean classes needed by the enterprise bean class such as utility and exception classes.

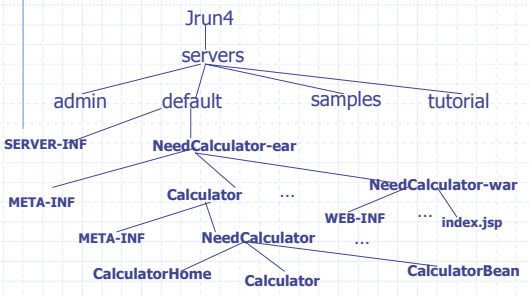
Naming Conventions

Item	Syntax	Example
Directory Name	**<name>-ear**	**Account-ear**
EJB JAR display name (DD)	**<name>**	**Account**
Enterprise bean class	**<name>Bean**	**AccountBean**
Home interface	**<name>Home**	**AccountHome**
Remote interface	**<name>**	**Account**
Local home interface	**Local<name>Home**	**LocalAccountHome**
Local interface	**Local<name>**	**LocalAccount**

Session Beans

- Tuition Need Calculator application.
  - It takes in many numbers and uses handful of formulae to come up a dollar amount for financial need for attending a given college.
  - We will implement this using a session bean.

Directory Structure for Need Calculator examples



INF Directories

- Contain the descriptor files
- Descriptor files are in XML
  - Can be auto-generated by tools.
- SERVER-INF has configuration of the server such as users, security.
- META-INF directory for ejb has ejb-jar.xml (ejb specific details) and jrun-ejb-jar.xml (ejb services specific details)
- WEB-INF directory for web applications has web.xml and jrun-web.xml.
- In general, xyz.xml and jrun-xyz.xml separate the application-server dependent and independent descriptors respectively.

Session Beans

- Session beans implement the "façade" design pattern, typically facilitating the data transfer between the user interface and the business logic beans (possible entity beans).
- These are **conversational** as opposed to entity beans being **transactional**.
- Stateless session beans don't remember anything about the user data so can be freely shared.
- Lets say we have 5000 users accessing your system, instead of 5000 sessions running, 50 stateless sessions can be shared among the users.

Home Interface: CalculatorHome.java

```
package NeedCalculator;
import javax.ejb.EJBHome;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import java.util.Collection;
public interface CalculatorHome extends EJBHome
{
        public Calculator create() throws
RemoteException,  CreateException;
}
```


Remote Interface: Calculator.java

```
package NeedCalculator;
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
public interface Calculator extends EJBObject
{
   public double calc (double cost, double avail) throws
   java.rmi.RemoteException;
}
```


Session Bean: CalculatorBean.java

```
package NeedCalculator;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import javax.ejb.CreateException;
public class CalculatorBean implements SessionBean
{  private SessionContext context;
  public double calc (double cost, double avail) {
    return (cost – avail); }
  public CalculatorBean() {}
```


CalculatorBean (contd.)

```
public void ejbCreate()  throws CreateException { }
public void setSessionContext(SessionContext context) {
                this.context = context; }
public void ejbRemove() { }
public void ejbActivate() {}
public void ejbPassivate() {}
public void ejbPostCreate() {}
}
```


Descriptor (ejb-jar.xml)

```
<ejb-jar>
  <enterprise-beans>
    <session>
      <display-name>Calculator</display-name>
      <ejb-name>Calculator</ejb-name>
      <home>NeedCalculator.CalculatorHome</home>
      <remote>NeedCalculator.Calculator</remote>
      <ejb-class>NeedCalculator.CalculatorBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
      <security-identity>
        <use-caller-identity />
      </security-identity>
    </session>
  </enterprise-beans>
</ejb-jar>
```

Creating the files

- Can do it manually using templates of previous applications.
- Can use JWizard that will automatically generate the XML descriptors.
- Can use XDocLet which will automatically generate files and regenerate to reflect any changes.
- Other methods from a integrated development environment such as Sun Studio, and IntelliJ.

Deployment

- Hot deploy: This is most convenient way to deploy the components. Lets try this with compass example.
- Create the standard directory structure either manually or using tools. Place the files in the appropriate directories.
- Start the server or restart the server.
- If there are errors, correct them, recompile and restart/redeploy.

Web Application to test the NeedCalculator

- We will write a very simple JSP file called index.jsp that:
  1. Resolves the JNDI name from the initial context to create the home directory.
  2. Narrows and casts the object reference obtained in the above steps to the home object of the NeedCalculator.
  3. Creates the EJbObject representing the remote interface of the Calculator.
  4. Invokes the calc method on the reference obtained in step3.

NeedCalculator-war/index.jsp

```
<%@ page import="NeedCalculator.*" %>
<html>
<head>
<title>Need Calculator</title>
</head>
<body>
<%
        try {
        javax.naming.InitialContext ctx = new javax.naming.InitialContext();
                Object obj = ctx.lookup("java:comp/env/ejb/Calculator");
```


Web Application (contd.)

```
CalculatorHome home =
(CalculatorHome)javax.rmi.PortableRemoteObject.narrow(ob
j, CalculatorHome.class);
                Calculator needCal = home.create();
                double d= needCal.calc(10000, 5000);
                out.println("Your Need is = $" + d);
%>
<br>Thank you.Your need has been calculated.<br><br>
<%
        } catch (Exception e) {
%>
        <br>Sorry, unable to calculate need.
```

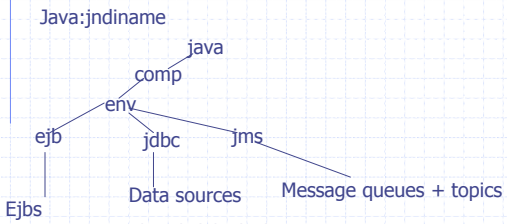

WEB-INF/web.xml

```
<welcome-file-list>
 <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
<ejb-ref>
 <description>Calculator session bean</description>
 <ejb-ref-name>ejb/Calculator</ejb-ref-name>
 <ejb-ref-type>Session</ejb-ref-type>
 <home>NeedCalculator.CalculatorHome</home>
 <remote>NeedCalculator.Calculator</remote>
</ejb-ref>
```

JNDI Names

- This application uses ejb-refs so that clients can always locat the ejb under the java:comp/env environment naming context (ENC).
- The jrun-web.xml file maps the ejb-ref-name to the actual JNDI location.
- Clients can then lookup the EJB using either the actual JNDI location or java:comp/env/*ejb-ref-name*
- If there is no tags corresponding to ejb-ref then lookup will be to the actual name "Calculator" of the java naming service.

JNDI Names (contd.)



java:comp/env/ejb/Calculator
java:comp/env/jdbc/compass
java:comp/env/jms/newsQueue

Understanding and Designing with EJB

B.Ramamurthy

Based on j2eetutorial documentation.
http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html

Review

- Request/Response Model
- Distributed Objects: stubs and skeleton providing location transparency
- Naming and Lookup: registry and binding
- Server-side technology: servlets
- Web applications: can be written entirely using Java Server Pages (static and dynamic content and data access can be provided); JSP is wrapper on servlet technology.
- Concept of initial context:The starting point for resolution of names for naming and directory operations.
- Data base access: using Java Data Base Connectivity

When to use EJB

- For large scale applications: where resources and data are distributed.
- When the application is run on servers at many locations.
- Where scalability is critical.
- Where transactions are required to ensure data integrity
- When a variety of clients need to handled.

Types of Enterprise Bean: Session

- **Session bean**: represents a single client inside the J2EE server. Session represents an interactive session. When a client terminates the session bean terminates/is no longer associated with the client.
- Stateful session bean: maintains a conversational state for the duration of a session. Ex: items reviewed in a session at some sites
- Stateless session bean: does not maintain a conversational state. Ex: computing a formula for a given value

Types of Enterprise Bean: Entity

- An entity bean represents a business object in a persistent storage mechanism. Ex: customers, orders, and products.
- Each entity bean typically has an underlying table in a relational database (business data), and each instance of the bean corresponds to a row in that table.
- Transactional and recoverable on a server crash.

Types of Enterprise Bean: Message-Driven

- A message driven bean is an enterprise bean that allows J2EE applications to process messages asynchronously.
- It acts as a JMS listener, which is similar to an event listener except that it receives messages instead of events.
- The messages can be sent by any J2EE component: an application client, another enterprise bean, or a web component, or a non-J2EE system using JMS.
- Retain no data or conversational state.

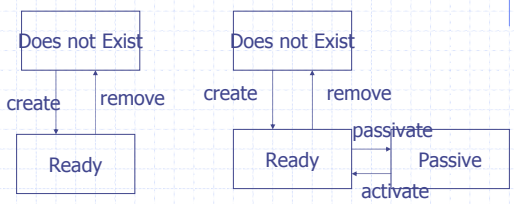
Contents of an Enterprise Bean

- Interfaces: The remote and home interface for remote access. Local and local home accesses for local access.
- Enterprise bean class: Implements the methods defined in the above interfaces.
- Deployment descriptor: An XML file that specifies information about the bean such as its type, transaction attributes, etc.
- Helper classes: non-bean classes needed by the enterprise bean class such as utility and exception classes.

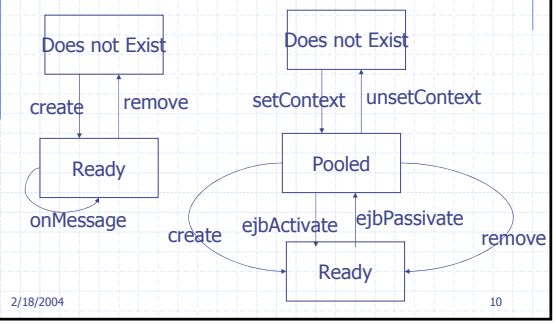
The life cycles of enterprise beans

- An enterprise bean goes through various stages during its lifetime. Each type has different life cycle.

Session bean



Entity and Message-driven Bean Lifecycle



Entity Bean

- Data is at the heart of most business applications.
- In J2EE applications, entity beans represent the business objects that need persistence (need to be stored in a database.)
- You have choice of bean-managed persistence (BMP) and container-managed persistence (CMP).
- In BMP you write the code for database access calls. This may be additional responsibility but it gives control to the bean developer.

Entity Bean class

- Implements EntityBean interface
- Zero or more ejbCreate and ejbPostCreate methods
- Finder methods
- Business methods
- Home methods

Entity Bean Methods

- ejbCreate inserts the entity state into the database; initializes the instance variables and returns the primary key.
- ejbRemove will delete the record corresponding to the bean from the database.
- ejbLoad and ejbStore methods synchronize instance variables of an entity bean with the corresponding values stored in a database. ejbLoad refreshes the instance variables from the db and ejbStore writes variables to the database. Container does this not the client.
- ejbFinder allows client to locate entity beans. Find the collection of records with "Smith" as author.
- Business methods and home methods.

SQL statements in Entity Bean

Method	SQL Statement
ejbCreate	INSERT
ejbFindPrimaryKey	SELECT
ejbFindByLastName	SELECT
ejbFindInRange	SELECT
ejbLoad	SELECT
ejbRemove	DELETE
ejbStore	UPDATE

Midterm Exam Review

- Web application design: n-tier design from word problem. Represent using block diagram, use case and class diagram. Stepwise explanation; project 1
- J2EE Application model: application model
- Enterprise beans: Session, entity and (message-driven beans): characteristics and life cycle
- Enterprise integration.

Exam format

- Open Book and Open Notes
- Questions are design-based (Be prepared with UML diagrams)
- Technology questions will be J2EE (EJB) based.

Grid Application Model and Design and Implementation of Grid Services

B.Ramamurthy

The Scientific Imperative

- Computation and data management infrastructure
- Data intensive science
  - Prospect of federating many archives from different globally distributed sources
  - Virtual observatory, lab
- Simulation-based science
  - Compute intensive
- Remote access to experimental/expensive apparatus
- Virtual community science

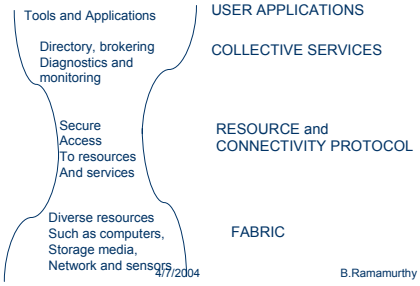
The Industrial Imperative

- Evolution of technology: Phase I: Development phase: How to build, how it works, feasibility, trial and error, … popularity of technology grows leading standardization and mass production.
- Phase II: Post technology. Adoption of well tested technology, general public simply assumes the technology. Its existence is transparent.

The Social Imperative

- Computing benefits delivered to the masses as a commodity or utility.
- People don't have to own a computer to access computing.
- Resources among participants will be trustfully shared.
- Virtual organization concept will lead to creative business models.

Grid Architecture



Applications

- Predictive maintenance: distributed aircraft engine diagnostics
  - distributed, data centric
  - Requires collaboration among a number of diverse actors within the stakeholder organizations, who may need to deploy a range of different engineering and computational tools to analyze a problem.
  - VO to support services, individuals and systems.

Distributed Telepresence

- The NEESGrid earthquake engineering collaboratory
  - Broad range of activities performed by a community of engineers and researchers engaged in improving performance of buildings and other structures when subjects to effects of earthquake.
  - Expensive experimental facilities
  - Simulation systems
  - Real earthquake prone areas
  - Teleobservation and telecontrol

Scientific Data Federation

- The world-wide telescope
- Astronomy community has a fairly unified taxonomy and metrics and units.
- Cross-comparison of data from various sources, media, and times.
- Making discoveries
- Virtual observatory enabled by the grid
- Statistics and computationally intensive operations

Medical Data Federation

- Biomedical informatics research network: National Institute of Health (NIH) is pioneering use of grid structure for medical research and patient care through Biomedical Informatics Research Network (BIRN) project.
- Scalable infrastructure consisting of advanced network, federated distributed data collections, computational resources and software technologies to handle evolving needs of users.
- Imaging, morphology, mouse models, information mediation.
- Sharing expensive research results.

Knowledge Integration

- In silico experiments in bioinformatics: is a procedure that uses computer-based information repositories and computational analysis to test a hypothesis, derive a summary, search for patterns, or demonstrate a known fact.
- Mygrid is one such experiment.
- More service orientation.
- Services for data-intensive integration.
- Semantic discovery and metadata management.
- Forming experiments.
- See Figure 9.2

Distributed Data Analysis

- CMS: Compact Muon Solenoid is a high-energy physics at European Center for Nuclear Research (CERN) near Geneva, Swiz.
- To be completed in 2007.
- Will record data from highest-energy proton-proton collision.
- Will shed light on fundamental scientific issues.
- Condor is one of the predominant software used.

Other Applications

- Desktop grids
- Enterprise integration
- Multiplayer gaming infrastructure
- Service virtualization
- Group oriented collaboration systems
- Astrophysics: black holes, novas, stars and galaxies

Reference

- The Grid2: Blueprint for a New Computing Infrastructure by I.Foster and C. Kessleman
- Globus Toolkit 3.0 Quick Start by IBM RedBook
- Globus Tutorial at http://www.casa-sotomayor.net/gt3-tutorial/
- Grid Physiology Paper

Design and Development of a Federated Information System

Bina Ramamurthy

CSE4/587 Information Structures **Due Date: 2/24/2004 by mid-night.**

Purpose:

1. Design and develop a multi-tier distributed system offering remotely accessible services.
2. Understand the components, the core technologies, the architecture and the protocols enabling a J2 Enterprise Edition (J2EE)-based distributed system.
3. Design and implement system processes using Enterprise Java Beans (EJB).
4. Understand the process of preparing and deploying an interoperable remote service.
5. Build a Federated Information SysTem (FIST) through interoperation of several autonomous distributed systems.

Preparation:

1. Get a clear understanding of multi-tier distributed systems. (See lecture notes).
2. Understand the technology underlying the J2EE: its architecture and application models. See http://java.sun.com/developer/onlineTraining/J2EE/Intro2/j2ee.html
3. Learn how to use the XML-based build tool Ant at http://ant.apache.org/
4. Understand the role of deployment descriptors. The deployment descriptors are XML files used to configure runtime properties of an application thus relieveing application to deal only with the programmatic details.
5. Learn to use the application interface to the Oracle database using embedded SQL and JDBC. Alternatively you may use a file-based simple database Cloudscape.
6. Download and install Macromedia JRun4 Devloper edition and then the Updater2 (service pack). JRun4 is a J2EE compliant software environment for developing distributed systems. This can be done either or both in the project space that will be allocated to you and at your home, if you have the facility.

Technology details:

J2EE offers a suite of software specification to design, develop, assemble and deploy enterprise applications. It provides a distributed, component-based, loosely coupled, reliable and secure, platform independent and responsive application environment. It encompasses technology solutions for all tiers of a distributed system: client tier, web tier, (business) logic tier, and enterprise information system (database) tier. Sun Microsystems Inc. provides a reference implementation of J2EE compliant environment and many businesses offer fine products such a Macromedia JRun4 and BEA Weblogic for J2EE-based development. For this project, we suggest you use JSP (Java Server Pages) for the web-tier, EJB (Enterprise Java Bean) for the logic tier, and any relational data base (Cloudscape or Oracle) for the data-tier. An *enterprise bean* is a server-side component that contains the business logic of an application. At runtime, the application clients execute the business logic by invoking the enterprise bean's methods. Enterprise Java Bean architecture frees the application developer from having to deal with the system level aspects of an application. Developer can deal with the programmatic aspects of the application while the systemic needs of the application such

as data base driver and message queue can be specified declaratively. Ultimate goal of introducing J2EE at this point is to encourage the students to compare it to the grid technology that will be discussed later in the semester.

Assignment (What to do?):

Consider a very common service sought by many people at this time of the year, the tax return filing. It is a yearly duty that many of us love to hate. If we can bring together the organizations that are involved in this tax filing process and allow interactions among them to perform the tax return filing in a trustworthy manner, it will be a great benefit to the society. Assuming that each organization can be modeled as a distributed information system, the above paradigm will allow free and secure exchange of information among the organizations, thus resulting in a Federated Information SysTem or FIST. We will consider four hypothetical organizations as shown in Figure 1: (i) Personal profile system, (ii) Employee information system, (iii) Banking information system and (iv) Internal Revenue System (IRS). We refer to an organizational system as a Virtual Organization (VO) following the terminology grid technology uses.

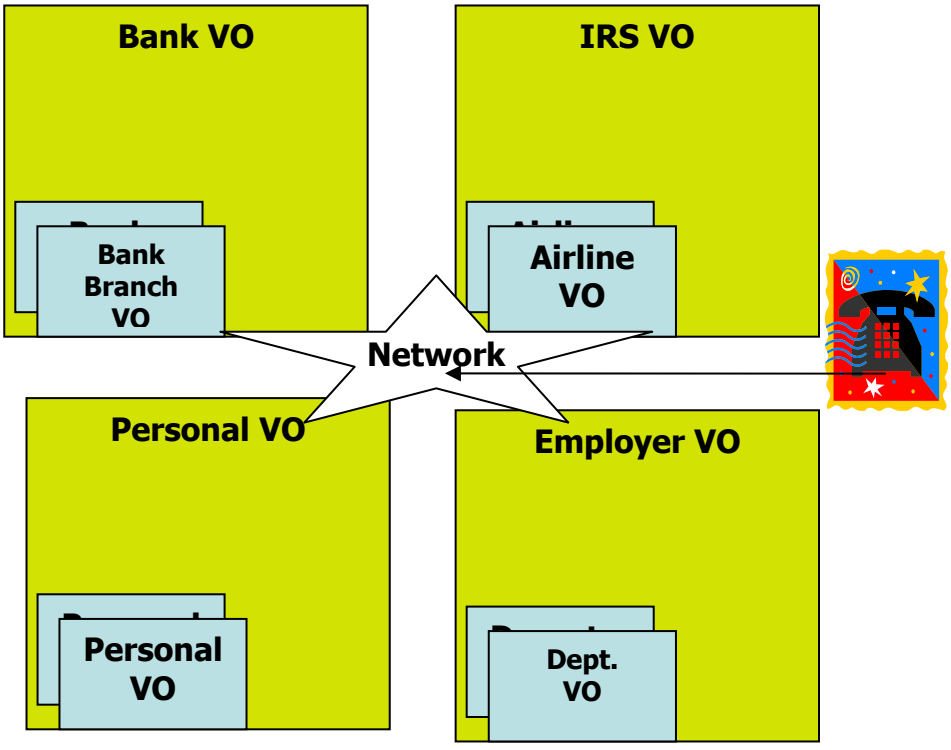


Figure 1: Application Model of a FIST

A person who wants file a tax-return calls up a number and authenticates himself/herself with appropriate personal information (say, last five digits of social security number and mother's maiden name) and authorizes a proxy to file his/her tax return and collect all the necessary information from the FIST shown in Figure 1. Typically there may not be any more interaction needed from the user. Any information needed by the tax filing process is automatically gathered from the organizations collaborating in the FIST. User interface can

be any device accessible to a user; however you will use a simulated interface. Determine the user profile using the authentication provided by the user without any explicit request. Gather user information based on this profile. Make decisions and selections to come up with best solution based on user profile and the data collected. Determine payment methods based on the user information and complete the transactions. Notify user if notification were requested. Log the status of the process and any anomalies.

Your assignment consists of two parts: (i) design and implement one of the VOs in the Figure 1 and (ii) write a FIST application that implements tax return filing. You will work in groups of not more than two people. Implementation of the individual VO will be completed by an earlier due date of 2/18/2004. You should submit a J2EE-based tested and operational VO by this date. Then each of the group will work their own FIST application that provides a user interface and interoperates among the VOs to provide the service of filing tax return. This will be submitted on the posted due date of 2/24/2004.

Analysis and Design:

Server side: Research and analyze the problem to understand the requirements. Represent the system requirements using UML (Unified Modeling Language) diagram. Choose one of the systems (VO) for your further design. Identify the entities, processes and rules. Discover classes needed to implement the processes and entities. The rules are typically represented by methods in the classes. Represent the classes and relationship among them using a UML class diagram. Decide which among the classes will have methods that will be exposed to the users. Typically these will be implemented as enterprise components (EJBs in our case). Design a relational database to store persistent entities. The design document at the end of this phase will have use case diagram(s), class diagram(s), and a diagram (Entity-Relationship diagram) representing the database design. These documents have enough information to start coding.

Client side: Design a simple interface with a client-tier and web-tier combined for the VOs (Reminder: Each group will design only one VO). However, design a creative user interface for the FIST (overall system).

Implementation steps and details:

1. **Getting used to building client-server systems**: When you implement a simple cient side application program there are just two steps involved: compile and execute the code. In a client-server system, you will have to take care the server side as well as the client side. On the server side, you will compile the code, generate stubs or proxies using special compilers, deploy the service, register and publicize the service for the clients to use. On the client side you will prepare the client code with appropriate stubs, and during excution lookup the service needed and use it. You will notice that besides simple compile and execute, configuration and deployment of a service are important issues to be reckoned with.

2. **Working with the relational database and embedded SQL:** In this project you will store the data in a relational table and access it using SQL statements embedded in Java lanaguage. Work on a simple java program to refresh your knowledge about accessing the Oracle database. See http://www.cse.buffalo.edu/local/Consulting/Orcale.html for examples and access details.

3. **Building systems using build tools such as Ant:** In order to tackle complexities in configuration and deploying server-side applications, you will need to use special build tools. Apache Ant is a XML-based build tool which similar to "make" utility that most of you are familiar with. This topic will be covered during the recitation this week. Work on simple simple files to familiiarize yourself with the Ant build tool.

4. **Study and understand Enterprise Java Bean building and deployment details:**

   a. You will user Macromedia JRun developer edition for the J2EE components. Download details will be discussed during the recitations. They are quite simple. You can work at home by downloading one into your personal computer and bring the deployable units to school for deployment.
   b. Study the examples in the documentation that comes with the JRun installation.
   c. For the database you may use the database that comes with JRun or Oracle database.
   d. For the client-tier we suggest that you use JSP. We will cover JSP and servlet during the recitation.
   e. While you have choice of technology in implementing data tier and client-tier, it is required that main exposed business logic should be implemented using EJBs. However, utilities supporting the business logic can be implemented using regular Java classes.
   f. It is very important that you understand the concept of remote method call, name resolution, registering and lookup. The concept of component programming using EJBs is also equally important. We will discuss these with examples during lecture.

5. **Design, implement and test your Virtual Organization:** Using the frame work given in the Step 4 above design the VO of your choice. This is expected to be the most time consuming part of the project due to the novelty of the topic.

6. **Deploy the integrated system:** The various components listed above were deployed and tested individually. Your final application will use VOs implemented by other groups. So we will need well defined interfaces.

   a. Test the individual modules before assembling into a VO application.
   b. The final application should single-click accessible from the web.

7. **Work in Groups:** You will collaborate in groups to implement a FIST for tax return filing. The protocol for interaction within and among groups will be clearly specified.

8. **Practice good programming style**: Finally, practice all the good programming styles that you learned in the lower-level courses.

Submission Details:

Create a project1 directory and use that as the working space. Let the code directory tree be located in this directory. Let the design be represented by an integrated class diagram and presented in a file project1.pdf. Provide internal documentation using javadoc style comments. You will create a README file containing the details of the package and processing. Zip the project1 directory and submit the resulting zip file, project1.zip. Making sure that you current directory contains your project1 directory, you can create this file as follows:

```
zip -r project1.zip project1
```

Use the electronic submission program that corresponds to your class (cse4/587). For instance students in cse587 will submit by typing

```
submit_cse587 proejct1.zip
```

at the Unix prompt.

Documentation and Report: See report details.

Project 1: Federated Information System

Phase 2: Developing the Tax Filer Portal

Vijayram Arthanari

CSE 487/587

February 24, 2004

Phase 1: Developing EJBs - Completed

- Develop Four Entity Beans each representing on of the four VOs – Personal Info, Employee Info, Banking Info and IRS Info.
- Test the entity beans individually using JSP based web clients and a relational database to persist the bean data.
- Use CMP 2.0 or BMP to implement the persistence for the entity beans

Phase 2: Developing the Tax Filer Portal

- Design the Tax Filer Portal with following functionality:
  - User login
  - File tax return
  - Query status of the returns filed
- Simple JSP-based application federates information from various VOs.
- Suggestions:
  - Use session bean as a facade for the entity beans.
  - Any additional functionalities can be implemented if needed.

Phase 2: Developing the Tax Filer Portal

- Typical sequence of events:
  - User logs on to portal and chooses to file a return
  - Portal looks up the EJBs and gathers all information required for filing the return from Banking, Personal, Employee VOs using the SSN of user.
  - Form1040NREZ is populated with the appropriate values and submitted to IRS VO.
  - IRS VO verifies the return and does a direct debit/credit on the Bank VO if there is any tax due/refund.
  - The status is reported to the user upon request.

Phase 2: Using JNDI lookup

- Get the required VOs (ears) from other groups or develop the VOs using JRun wizard and deploy on your server to test your application.
- Test the application in following scenarios:
  - Deploy each of the VOs on a JRun server. The portal would access the EJBs and perform the desired functions. (Default case)
  - Deploy each of the VOs on **different** JRun servers. The portal would use JNDI lookup to locate the EJBs distributed among various servers.
  - (optional) Use service data parameters of the VOs to choose the most cost-effective VO to perform the functionality.

Analyzing and Visualizing a Large Data Set Using Grid

Bina Ramamurthy

CSE4/587 Information Structures **Due Date: 4/18/2004 by mid-night.**

Purpose:

1. Design and develop a solution to analyze a large set of real data from a pharmaceutical experiment.
2. Understand the components and operation of a condor-based (High Throughput Computing) grid (CSECCR) built using recyclable Sparc 4 machines.
3. Design and implement a Java application and submit script to execute the solution developed in step 1.
4. Learn to use database and graphing tools with grid-based jobs.
5. Understand the process of utilizing CPU cycles offered by CSECCR grid.

Preparation:

1. Get a clear understanding of condor-based CSECCR grid you will be using for this project. (See notes given below).
2. Understand the technology underlying Condor: its architecture and application models. See http://www.cs.wisc.edu/condor/
3. Understand the role of submit scripts.
4. Learn to use various tools such as GnuPlot for drawing graphs of various relationships among the data.
5. Make sure you have an account on johnlee.ccr.buffalo.edu by logging into it using secure shell from any of the cse machines. Your username name is same as UBIT username and the password is your person number.

Technology details:

You will work with CSECCR grid shown in Figure 1. The grid is primarily composed of Sun Sparc4s which form the compute nodes. The 40 compute nodes form an internal private Class C network with a grid front end. All nodes run Solaris 8, and middleware is configured for a 'shared file system' oriented job execution. There are a total of around 40 CPU's and the total memory is around 2.5 GB. The front end has another external interface, through which jobs are submitted.

The grid also has various middleware solutions installed in it for educational research. The middleware is primarily composed of NMI (NSF Middleware Initiative) components. Middleware components installed range from Globus, Condor-G and PBS (Portable Batch System) and NWS (Network Weather Service). PBS is for job management and scheduling, Globus and Condor for resource management, NWS and Ganglia for distributed resource monitoring. You shall be using Condor as your primary grid middleware. This means that both the *job management* part and the *resource management* part of the grid is taken care by Condor daemons. Condor has various commands which let's you submit

jobs, monitor and manipulate the job queue, assign job preferences etc as described later in the Getting Started with CSECCR Grid.

Assignment (What to do?):

You are given a data from a gene expression experiment. The gene expressed by their DNA after treatment with a certain drug for a set of patients is recorded and provided in the database. This data is available at /projects/bina/PD_pt1_14.xls. This will be ported to a relational database connected to the CSECCR grid. (You don't have to do this. Vijay will do this.) The data contains 14 patient's information, different types of genes, their expression levels at various times (1 hours, 3hours, 1 day, 1 week etc.) after treatment with some experimental drug. This provides a "time series". You analysis and graphing can be (i) as simple as line drawings of gene expression over time, (ii) average of gene expression for a specific time over patients (iii) repeated measures ANOVA, (iv) application sophisticated algorithm such as that of Markovitz to choose best responding gene etc. It is our goal to provide all possible analysis. It is up to the expert to interpret the analysis. For this purpose you may provide a simple portal for visualizing the results of the analysis.

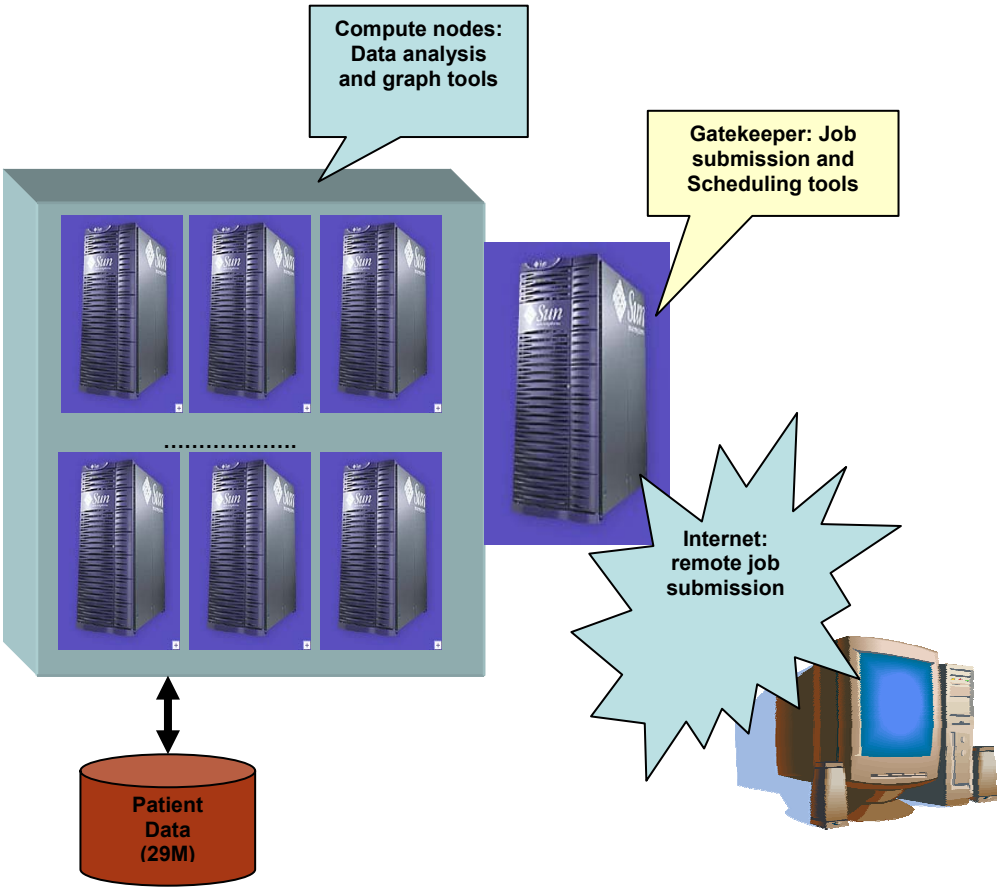


Figure 1: CSECCR Condor-based grid

Use Scenario: A scientist who has gene expression data or similar data in that domain will populate the database with the data. He/she will then choose a set of analysis to be carried

out on the data. This could be selected from a menu of available analysis tool. Once the scientist is satisfied with the selection he/she submit the job to the grid for processing. The grid software will orchestrate the analysis of the data but organizing needed resources and tools. On completion the scientist will be notified and he/she may review the results before initiating further action/analysis/repeat experimentation. A portal that facilitates all the operations/features discussed above will be an ideal solution. The requirement for the project within the scope of this course is to submit a Java or C-based program to analyze the data and to obtain outputs in the form of graphs (visuals).

Your assignment consists of three parts: (i) design and implement a Java/C/C++ program that reads in the data from the data source connected to the grid, (ii) interface Java/gnuplot API and and (iii) prepare graphs for visualization. It is enough if you generate a few representative set of graphs as a proof of concept. If you prepare the same graph for every single data set you may run of limited disk space that may be allocated to you. You will work in groups of not more than two people.

Analysis and Design:

Data Analysis: Study the excel file that contains the data. Some of the columns are easy to understand and many of them of domain-specific information which is quite common with such data collections. One of the sub tables you will be working with is the gene-expression over a period of time. Locate this on the excel sheet and study it. In the second worksheet of the excel workbook the averages for the same data are given. You will work with these sets of data for specific genes of your choice. You can draw line graphs, average line graphs, and other sophisticated graphs of your choice.

Grid side: You should design your application in Java/C/C++ and use appropriate external and built-in API for generating graphs. Alternatively you may generate graph data files and visualize these explicitly using appropriate tools such as gnuplot.

Client side: Ideally we would like to see a portal implemented to carry out the analysis and visualization. Due to lack of time this aspect is left as an optional exercise.

Implementation steps and details:

1. **Working with Condor-based grid**: You will be given accounts on the CSECCR grid described above. You will login and verify that you have an account. If not send mail to bina@cse.buffalo.edu Log into the account and prepare a simple Java program or C program and prepare job submission script for as described in the "Getting Started" section above. Submit the job and monitor its progress using command line operations or the web-based monitoring tool Ganglia (http://johnlee.ccr.buffalo.edu/ganglia)

2. **Working with the data source:** We will convert the data available in the excel file into a relational databse and make it accessible through jdbc/odbc interface. However

this may work for Java programs. For C++/C programs you may use a simple subset of the data in a file in your local disk space.

3. **Working with graphing tool:** Your java program will read the data from the data source, process it and generate data for graphing. You may use Java 2D API or free ware gnuplot API for Java at http://www.is.informatik.uni-duisburg.de/projects/java-unidu/api/de/unidu/is/gnuplot/package-summary.html

4. What are we interested in? We are interested in monitoring and benchmarking the power of the condor-grid we have assembled. We expect that you will computationally intensive analysis of the data. An example of is repeated measure ANOVA, the details of which can be found http://www.utexas.edu/cc/docs/stat40.html

5. What will you learn? Working with grid-based job preparation, submission, monitoring, and managing data for data and computationally intensive problems.

6. For Java-based job, you will need the class files (ex: helloWorld.java), shell script with java command (hello.sh), and a submit script (hello.submit) specifying resource requirements. You will condor_submit hello.submit to schedule the job on the grid. See "Getting Started with CSECCR Grid", document for details on how to prepare and submit a job.

Submission Details:

Create a project2 directory in your project space. Copy your source code and outputs from CSRCCR grid into this directory. Add an experience report that details how you accomplished tasks outlined in this project. Call it ExpReport2. This report should also outline how we can use your program to obtain your outputs.

zip -r project2.zip project2

Use the electronic submission program that corresponds to your class (cse4/587). For instance students in cse587 will submit by typing

submit_cse587 proejct2.zip

at the Unix prompt.

Getting Started with CSECCR grid
Prepared by Karthikram Ramamani
kv8@cse.buffalo.edu

A typical job for the CSECCR grid will be written a high level language, say, Java or C++, compiled and preprocessed, if necessary to prepare the executable. A submit script is prepared that declares all the requirements for scheduling and execution of the job. Then the job is submitted using Condor submit command. Condor provides commands for monitoring and controlling the executing job.

Preparing the executables:

For java program, compile the programs using javac. No special preprocessing is needed. Prepare a submit script and shell script using the sample scripts provided in your home directory.
For C/C++ programs, compile using
condor_compile gcc sourceName –lm –o executableName

Job* Submissions *via Condor:

The condor_submit command is used to submit jobs to the Condor scheduler. The argument to the command is a submit script file which specifies the job preferences. Matchmaking in Condor is done on the basis of this submit file or *ClassAds*. You will find example submit files in your home directory. Modify it as per your needs. Certain attributes are mandatory for proper job submissions, so go through the Condor manuals before you make any considerable changes to the submit files. The load on the cluster at any given time can be monitored using Ganglia Distributed Monitoring, by connecting to the apache server at http://johnlee.ccr.buffalo.edu/ganglia

Any error conditions that arise while executing your jobs are logged in the log file you specify. If your jobs are go into idle state for long periods, check your submit files for incompatibilities. 'condor_q –analyze' command gives the analysis of the submitted job.

Condor Quick Reference:

condor_compile

The condor_compile command assists in linking jobs with the Condor libraries so that Condor features like migrating and check-pointing are made use of. For your Java project you need not make use of condor_compile.

condor_q

The condor_q command displays the Condor job queue at the instant. The job id, running time, job status etc are displayed. The command has various options which make the output more descriptive. (Go through Condor manual for preferences)

condor_rm

This command along with the job id, removes the specified job out of the queue. Use this command to delete your jobs from the queue.

condor_status

This command displays the hosts running Condor, their status (Claimed/Unclaimed), host info. Command has options for displaying host specific information like *machineAds,* architecture etc.

condor_submit

This command is used to submit jobs to Condor. Takes as argument a submit file, which specifies the ClassAd (User Job Preferences).

condor_history

Lists the history of jobs submitted to Condor along with their exit status.

condor_hold & condor_release

These commands hold and release the job specified respectively.

Sample Condor Submission.

Any condor submission will require a submit script which specifies your job classAd (job preferences). Samples submit script is given below. Note that "kv8" has to be replaced by your username.

```
*Executable = /home/kv8/submit.sh
Output =        /home/kv8/submit.out
Log =           /home/kv8/submit.log
Error=          /home/kv8/submit.error
*Universe =    java
*Requirements= Arch == "SUN4x"
*Rank       =     Memory >= 30
*Arguments= helloworld
*Queue
```

*indicates the attribute is mandatory

Submit script above says the execution universe is a java universe, class name is helloworld, specifies the executable to move, the job requirements and the rank. You also request the job to be queued for submission. A sample *submit.sh* for above job would be:

```
#!/bin/sh
java helloworld
```

Note that all references to files are absolute paths. Please avoid using relative path names in your submit scripts. Go through the Condor manual at http://cs.wisc.edu/condor for detailed information on Condor.

CSECCR Grid Etiquettes:

The server you log into is johnlee.ccr.buffalo.edu. Your username will be your UB IT NAME and your initial password will be your UB person number. Make sure you change the password immediately after you log in.

Considering the infrastructure of the Grid, during times of heavy load, the grid is bound to be slow. Make sure you don't submit too many jobs at a time and flood the job queue. When your job finishes running, you will be notified about the exit status of the jobs via email. You can monitor the status of your jobs in the queue using Condor commands. Feel free to remove your jobs from the queue if you don't need them. We would certainly appreciate saving computational cycles.

PLEASE wait for your jobs submissions to finish, before you fire up other jobs. This would ensure smooth network traffic and optimal performance for all users. During times of high network traffic, redundant jobs from a user shall be paused or removed if necessary.

Use JohnLee strictly for job submissions only. Do not log into other servers from JohnLee or initiate Netscape connections. **Your disk quota is a hard limit of 25 MB only. Avoid using relative path names in any job submission scripts you write. Please specify files using absolute path names, wherever you use them.**

Design and Development of a Virtual Organization using Globus Toolkit 3.0

Bina Ramamurthy

CSE4/587 Information Structures **Due Date: 4/18/2004 by mid-night.**

Purpose:

1. Understand the components and functions defined by Open Grid Services Architecture (OGSA).
2. Get hands-on experience, working with an implementation of OGSA in Globus Toolkit 3.0 (GT3).
3. Understand the concepts of virtual organization (VO), service definition and service oriented architectures (SOA).
4. Design and implement a grid service for IRS tax filing (as discussed in project 1).
5. Write a Java application to test the service developed in step 4.
6. (optional) Enhance the features of the service by adding logging, notification, security and other persistence services offered by grid framework.

Preparation:

1. Download GT3 and install it project space. Work with the samples in the download. You should be able to run grid services in the samples directory by starting the GUI browser for Globus services.
2. Understand the technology underlying Globus: its architecture and application models.
3. Download the GT3 tutorial that explains how to write a real grid service.
4. All these can be done in your project space.
5. You are also given accounts on LinuxGlobusGrid put together by KenSmith at CSE department. Make sure you have accounts on this grid by logging into "cerf", "mills" or "vixen" from host machine. You will "ssh" into these machines.

Technology details:

Open Grid Services Architecture (OGSA) defines the components of a grid service and Open Grid Services Infrastructure (OGSI) specifies the functionality. Globus Toolkit 3.0.2 is an implementation of the OGSI. A virtual organization (VO) supports one or more grid services by sharing resources from various organizations.

A grid service is a web service with features as shown in Figure 1. Basic service is enhanced by standard functionality specified by OGSA. In other words, a grid service can provide in a standard way logging, notification, service data, routabilty, security etc. These standard functionalities enable the seamless interaction of grid services in a global large scale and high density distributed system. Basic application model is also enhanced by collaborative models, and competitive models with such higher level capabilities as negotiation and mediations. These are initial steps towards commoditization of services and their availability as transparent utilities similar to electricity and water utilities. Such a model will certainly impact the society in a very significant way. Benefits of computers will be experienced by masses without any need to explicitly learn about computers or computing.

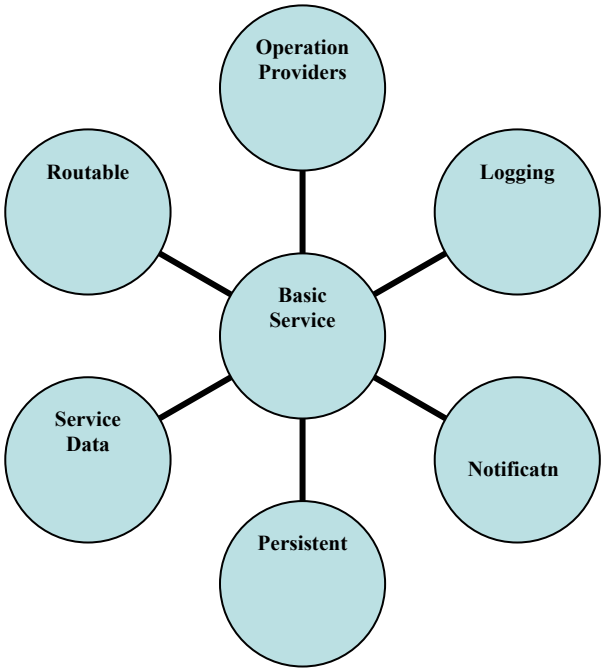


Figure 1 Features of a Grid Service

Assignment (What to do?):

You will implement virtual organization that will feature tax return filing service. Logical specification of the service is the same as given in Project 1.

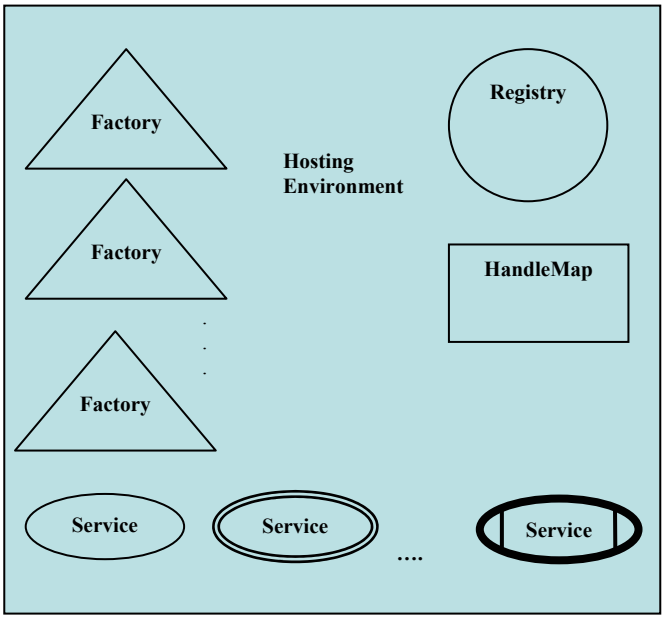


Figure 2: Virtual Organization with Distinct Symbols for Components

You will implement VOs for IRS, Employer, Bank, and Personal profile. IRS VO supports a grid service to perform the tax returns, and each of the other VOs support grid services to perform their respective operations. Let the IRS grid service be a logging grid service and bank service be adorned with notification feature. That is, bank will be a notification sink for messages from IRS. Other than that it is optional for you to add other OGSI features. Figure 2 shows a generic VO with newly defined symbols (by Bina Ramamurthy) for the various components: (service) Factory, Registry, HandleMap, three types of services (simple, complex, and end-to-end service) and the hosting environment. Figure 3 shows the IRS VO composed out of many such VOs shown in Figure 2.

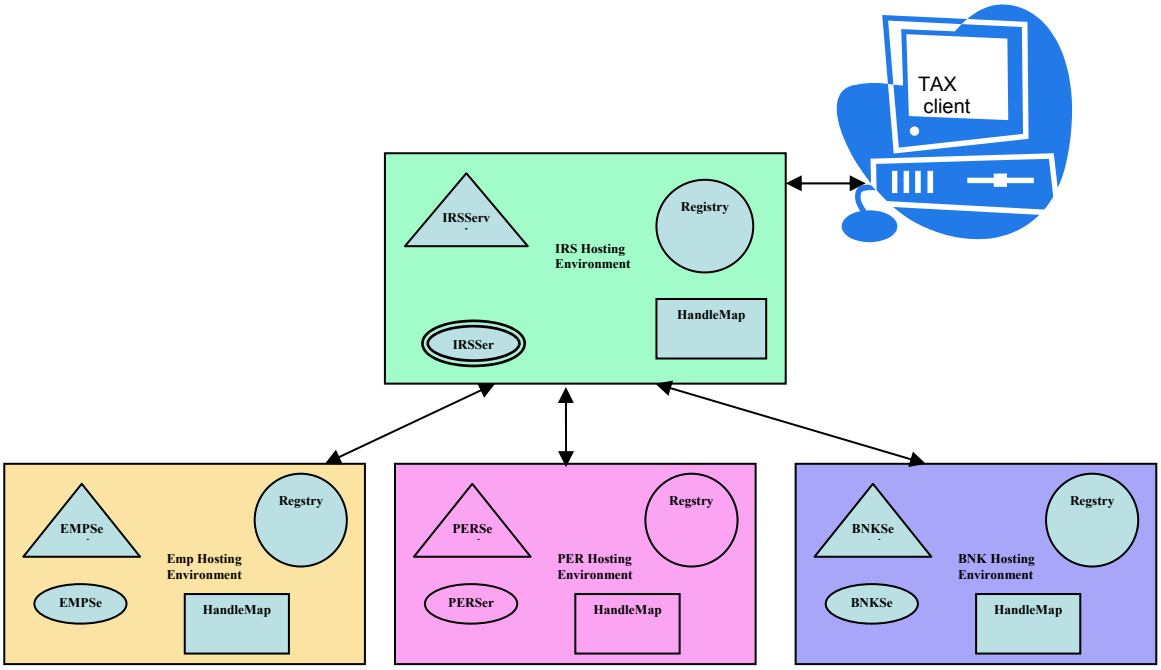


Figure 3: Tax Filing System Architecture

Use Scenario:
Any client who wants to file tax returns uses his/her communication device (a computer, cell phone, pager, telephone, PDA etc.) and authenticates himself/herself by sending appropriate information such as social security number. Then he/she authorizes filing of returns. Tax client then acts as a proxy for the user discovers and instantiates complex service IRSService which in turn invokes the EMPService, PERService and BNKService to accomplish the tax return filing.

Your assignment consists of these parts: You will implement the bottom three grid services independently in your project space and test them. Then write IRS Service that uses three services and accomplishes tax filing. You will work in groups of not more than two people.

Analysis and Design:

Server side: Research and analyze the problem to understand the requirements. Represent the system requirements using UML (Unified Modeling Language) diagram. For each of the VOs identify the entities, processes and rules. Discover classes needed to implement the processes and entities. The rules are typically represented by methods in the classes. Represent the classes and relationship among them using a UML class diagram. Implement very simple grid services with simple files for persistence.

Client side: Write a simple command line client. You may discovery services to locate your IRS service.

Implementation steps and details:

1. **Getting used to building grid services:** Work with Globus tutorial and understqand building grid services. You may use directory structure used by the tutorial or Globus core.
2. **Building systems using build tools such as Ant:** In order to tackle complexities in configuration and deploying server-side applications, you will need to use special build tools. Apache Ant is a XML-based build tool which similar to "make" utility that most of you are familiar with. This topic will be covered during the recitation this week. Work on simple simple files to familiiarize yourself with the Ant build tool.

3. **Study and understand grid services building and deployment.**

4. **Design, implement and test your Virtual Organizations and test them.**

5. **Deploy the integrated system:** The various components listed above were deployed and tested individually. Your final application will use VOs implemented by other groups. So we will need well defined interfaces. Test the individual modules before assembling into a VO application.

6. Create .gar (grid archive) for each VO. Please follow strict naming conventions: usernamePerService.gar, usernameEmpService.gar, usernameBNKService.gar, usernameIRSService.gar. Name other files based on this naming convention.

7. **Work in Groups:** You will collaborate in groups to implement a VO tax return filing. You can assume any interface and helpers required.

8. **Practice good programming style**: Finally, practice all the good programming styles that you learned in the lower-level courses.

Submission Details: Use the electronic submission program that corresponds to your class (cse4/587). Submit all gar files.
submit_cse587 xyz.gar at the Unix prompt.
Documentation and Report: See **report details**.

CSE487/587: Information Structures - Spring 2004
Macromedia JRun 4
Installation, Configuration and Verification Instructions (on CSE Machines)

Prepared by Vijayram Arthanari (va8@cse.buffalo.edu)

Installation:

1. Download into your project space /**projects/Spring_2004/cse487/*username*** on yeager/pollux server the packages:
   i. **JRun 4 Developer Edition** (English | Solaris) from http://www.macromedia.com/software/jrun/trial/
   ii. **JRun 4 Updater 2** for Unix (Solaris) from http://www.macromedia.com/go/jrun_updater
2. Edit the **.cshrc** file in your home directory. Add the following lines:
   setenv JRUN /projects/Spring_2004/cse487/*username*/jrun4
   setenv JAVA_HOME /usr/j2se
   set path = ($path $JAVA_HOME/bin $JRUN/bin)
   Save the .cshrc file and do **'source .cshrc'** at the command prompt. Now your path variables are updated with JRun and Java directory details
3. Set execute permission for the JRun installation shell scripts, using the following command:
   cd /projects/Spring_2004/cse487/*username*
   chmod 700 jrun*.bin
4. Run the JRun installation script using the following command:
   sh jrun-40-solaris-en.bin
5. Installation scripts starts. Press Enter to view each page of the license agreement and enter **y** to accept the agreement at the prompt.
6. Enter zero (for the Developer Version) when prompted for JRun product serial number. (A serial number is not required to install the JRun Developer Version)
7. JRun prompts you to choose an install folder. Enter the absolute path of your JRUN directory as:
   /projects/Spring_2004/cse487/*username*/jrun4
8. JRun prompts you to choose the product features to install. Select the **Complete** feature set.
9. Enter a user name for the JRun Management Console (JMC), and press Enter. Enter a password for the JMC, and press Enter. (*Note*: Make a note of your user name and password. You would need them to log in to the JMC)
10. JRun prompts you to choose a Java Virtual Machine (JVM) version 1.3.0 or later giving a default choice. Enter the following JVM at the prompt:
    /usr/j2se/bin/java
11. Review the Pre-Installation Summary. Verify that the information is acceptable, and press Enter. The JRun installer creates the appropriate directories and extracts the system files. The installation completes and the port numbers for the servers are listed on the screen.
12. Run the JRun 4 Updater 2 installation script using the following command:
    sh jrun4-unix-en-updater.bin
    (*Note*: If updater is installed after configuring and verifying the JRun 4 installation, then make sure that the JRun servers are **stopped** using stop command – see instruction 21 – for the updater to install successfully)
13. Follow the procedure as described in the previous steps and enter the installation directory details for the updater to install the service packs. (The update could be verified later by checking the build number in the JMC which is discussed in Verification section)

Configuration:

14. Web Port configuration: You would be choosing unique port numbers for your servers so that they would not compete for ports with others' servers running on same CSE machines.
    i. Go to **$JRUN/servers/admin/SERVER-INF** directory. Edit **jrun.xml.**
    ii. Search for port number '8000'. It would typically be listed as:
    <service class="jrun.servlet.http.WebService" name="WebService">
    *<attribute name="port">**8000**</attribute>*
    <attribute name="interface"></attribute>*
    </service>
    Choose a 5-digit port number **not greater than 65535** (*Hint*: last five digits of your person number would give you a unique port number) and replace 8000 with your port number (say port="12345"). Save and close the file. Remember admin server's web port for starting JMC.
    iii. Similarly, choose unique port numbers for default (replacing 8100) and samples servers (replacing 8200) in **jrun.xml** files located in **$JRUN/servers/default/SERVER-INF, $JRUN/servers/samples/SERVER-INF** directories respectively. Save and close the files when done.(Choose three different port numbers for the three servers)
15. JNDI Port Configuration:
    i. Go to **$JRUN/servers/admin/SERVER-INF** directory. Edit **jndi.properties**
    ii. Search for JNDI port number '2910'. It would typically be listed as:
    java.naming.provider.url=localhost\:2910
    Choose a 5-digit unique port number not greater than 65535 (different from all the ports you had chosen in previous step) and replace 2910 with your port number. Save and close the file.
    iii. Similarly, choose unique port numbers for default (replacing 2908) and samples servers (replacing 2918) in **jndi.properties** files located in **$JRUN/servers/default/SERVER-INF, $JRUN/servers/samples/SERVER-INF** directories respectively. Save and close the files when done. (Choose three different JNDI port numbers for the three servers)

Verification:

16. Start your JRun admin server as follows:
    jrun –start admin &
    (Refer to the JRun Documentation at $JRUN/docs directory for more options to start the jrun servers). Wait for the admin server to come up. A message saying "**Server admin ready (startup time: xx seconds)**" would be displayed.
17. Open a web browser and go to "**http://machine.cse.buffalo.edu:12345**" (Replace '12345' with your port number and 'machine' with the machine name (pollux or yeager) on which your JRun server is running, if different).
18. The JMC login page would be displayed. To login, enter your JMC username and password selected during JRun 4 installation.
19. Go to admin > Settings > Version link. JRun Version Information is displayed. Check if the **Build Number** is **61650.** If yes, then JRun4 with Updater 2 has been successfully installed.
20. Start the Samples server from the JMC. Once server is up open the address: "**http://machine.cse.buffalo.edu:23456**" where machine is pollux/yeager and 23456 is the web port of your samples server. Try running the sample applications to verify if the JRun installation is successful.
21. Shutdown the JRun servers as follows:
    jrun –stop
22. If the verification works fine, then you are all done with the installation.

CSE487/587: Information Structures - Spring 2004
Usage Notes: Gene Expression Database on MySQL
(for johnlee.ccr.buffalo.edu)

Prepared by Vijayram Arthanari (va8@cse.buffalo.edu)

Configuration:

Edit the **.cshrc** file present in the home directory on johnlee.ccr.buffalo.edu. Add the following lines:

```
setenv JAVA_HOME        /usr/j2sdk1.4.2
setenv MYSQL_DIR        /opt/mysql/mysql-standard-4.0.18-sun-solaris2.8-sparc
set path = ($path  $JAVA_HOME/bin  $MYSQL_DIR/bin)
```

Save the .cshrc file and do '**source .cshrc**' at the command prompt. Now the path variables are updated with the directories of MySQL and Java.

Using MySQL:

To log on to the MySQL server, Run:

```
mysql --user=tux --password=cse587
```

This would connect you to the MySQL database server and open the "mysql>" prompt where SQL queries to access the tables could be typed.

To logout from MySQL server session, type:

```
exit
```

For other commands, refer to the tutorial at: http://www.mysql.com/doc/en/Tutorial.html

Gene Expression Database:

The Gene Expression data is stored in two tables – **averages, cluster_data** – both placed in **GeneExprData** database. This database is read-only. To view the records in the averages table, start mysql server as described above. At the mysql> prompt, run:

```
SELECT address, cluster_id, clone_number, title, location FROM GeneExprData.averages
WHERE clone_number <= 20;
```

(Or)

```
USE GeneExprData;
SELECT address, cluster_id, clone_number, title, location FROM averages WHERE
clone_number <= 20;
```

Note: In the tables, empty string fields are filled with "**NULL**" strings and empty numeric fields are filled with a value **-1** which is an invalid value for this application.

Using JDBC:

The jdbc driver for MySQL is "**com.mysql.jdbc.Driver**" which already has been copied to $JAVA_HOME/jre/lib/ext/ to be used by java programs with jdbc.

The DriverManager.getConnection method would take the following parameters:

Database URL:	**"jdbc:mysql://localhost/mysql"**
Username:	**tux**
Password:	**cse587**

An example java program using jdbc to access the gene expression database can be found at: http://www.cse.buffalo.edu/~va8/cse4587/samples/DBConnector.java

Table Structure: GeneExprData.averages

Field	Type	Null	Key	Default
address	varchar(255)	YES		NULL
clone_number	int(11)		PRI	0
acc	varchar(255)	YES		NULL
cluster_id	varchar(255)	YES		NULL
location	varchar(255)	YES		NULL
title	varchar(255)	YES		NULL
pre	double	YES		NULL
1hr	double	YES		NULL
2hr	double	YES		NULL
4hr	double	YES		NULL
8hr	double	YES		NULL
24hr	double	YES		NULL
48hr	double	YES		NULL
120hr	double	YES		NULL
168hr	double	YES		NULL
3mos	double	YES		NULL

GeneExprData.averages table contains 4234 records

Table Structure: GeneExprData.cluster_data

Field	Type	Null	Key	Default
address	varchar(255)	YES		NULL
clone_number	int(11)		PRI	0
acc	varchar(255)	YES		NULL
cluster_id	varchar(255)	YES		NULL
location	varchar(255)	YES		NULL
title	varchar(255)	YES		NULL
PD_PtA_pre	double	YES		NULL
PD_PtA_1hr	double	YES		NULL
PD_PtA_2hr	double	YES		NULL
PD_PtA_4hr	double	YES		NULL
PD_PtA_8hr	double	YES		NULL
PD_PtA_24hr	double	YES		NULL
PD_PtA_48hr	double	YES		NULL
PD_PtA_5day	double	YES		NULL
PD_PtA_7day	double	YES		NULL
PtA_3_mos	double	YES		NULL
PD_PtB_pre	double	YES		NULL
PD_PtB_1hr	double	YES		NULL
PD_PtB_2hr	double	YES		NULL
PD_PtB_4hr	double	YES		NULL
PD_PtB_8hr	double	YES		NULL

PD_PtB_24hr	double	YES		NULL
PD_PtB_48hr	double	YES		NULL
PD_PtB_5day	double	YES		NULL
PD_PtB_7day	double	YES		NULL
PtB_3_mos	double	YES		NULL
PD_PtC_pre	double	YES		NULL
PD_PtC_1hr	double	YES		NULL
PD_PtC_2hr	double	YES		NULL
PD_PtC_4hr	double	YES		NULL
PD_PtC_8hr	double	YES		NULL
PD_PtC_24hr	double	YES		NULL
PD_PtC_48hr	double	YES		NULL
PD_PtC_5day	double	YES		NULL
PD_PtC_7day	double	YES		NULL
PtC_3_mos	double	YES		NULL
PD_PtD_pre	double	YES		NULL
PD_PtD_1hr	double	YES		NULL
PD_PtD_2hr	double	YES		NULL
PD_PtD_4hr	double	YES		NULL
PD_PtD_8hr	double	YES		NULL
PD_PtD_24hr	double	YES		NULL
PD_PtD_48hr	double	YES		NULL
PD_PtD_5day	double	YES		NULL
PD_PtD_7day	double	YES		NULL
PtD_3_mos	double	YES		NULL
PD_PtE_pre	double	YES		NULL
PD_PtE_1hr	double	YES		NULL
PD_PtE_2hr	double	YES		NULL
PD_PtE_4hr	double	YES		NULL
PD_PtE_8hr	double	YES		NULL
PD_PtE_24hr	double	YES		NULL
PD_PtE_48hr	double	YES		NULL
PD_PtE_5day	double	YES		NULL
PD_PtE_7day	double	YES		NULL
PtE_3_mos	double	YES		NULL
PD_PtF_pre	double	YES		NULL
PD_PtF_1hr	double	YES		NULL
PD_PtF_2hr	double	YES		NULL
PD_PtF_4hr	double	YES		NULL
PD_PtF_8hr	double	YES		NULL
PD_PtF_24hr	double	YES		NULL
PD_PtF_48hr	double	YES		NULL
PD_PtF_5day	double	YES		NULL
PD_PtF_7day	double	YES		NULL
PtF_3_mos	double	YES		NULL

PD_PtG_pre	double	YES		NULL
PD_PtG_1hr	double	YES		NULL
PD_PtG_2hr	double	YES		NULL
PD_PtG_4hr	double	YES		NULL
PD_PtG_8hr	double	YES		NULL
PD_PtG_24hr	double	YES		NULL
PD_PtG_48hr	double	YES		NULL
PD_PtG_5day	double	YES		NULL
PD_PtG_7day	double	YES		NULL
PtG_3_mos	double	YES		NULL
PD_PtH_pre	double	YES		NULL
PD_PtH_1hr	double	YES		NULL
PD_PtH_2hr	double	YES		NULL
PD_PtH_4hr	double	YES		NULL
PD_PtH_8hr	double	YES		NULL
PD_PtH_24hr	double	YES		NULL
PD_PtH_48hr	double	YES		NULL
PD_PtH_5day	double	YES		NULL
PD_PtH_7day	double	YES		NULL
PtH_3_mos	double	YES		NULL
PD_PtI_pre	double	YES		NULL
PD_PtI_1hr	double	YES		NULL
PD_PtI_2hr	double	YES		NULL
PD_PtI_4hr	double	YES		NULL
PD_PtI_8hr	double	YES		NULL
PD_PtI_24hr	double	YES		NULL
PD_PtI_48hr	double	YES		NULL
PD_PtI_5day	double	YES		NULL
PD_PtI_7day	double	YES		NULL
PtI_3_mos	double	YES		NULL
PD_PtJ_pre	double	YES		NULL
PD_PtJ_1hr	double	YES		NULL
PD_PtJ_2hr	double	YES		NULL
PD_PtJ_4hr	double	YES		NULL
PD_PtJ_8hr	double	YES		NULL
PD_PtJ_24hr	double	YES		NULL
PD_PtJ_48hr	double	YES		NULL
PD_PtJ_5day	double	YES		NULL
PD_PtJ_7day	double	YES		NULL
PtJ_3_mos	double	YES		NULL
PD_PtK_pre	double	YES		NULL
PD_PtK_1hr	double	YES		NULL
PD_PtK_2hr	double	YES		NULL
PD_PtK_4hr	double	YES		NULL
PD_PtK_8hr	double	YES		NULL

PD_PtK_24hr	double	YES		NULL
PD_PtK_48hr	double	YES		NULL
PD_PtK_5day	double	YES		NULL
PD_PtK_7day	double	YES		NULL
PtK_3_mos	double	YES		NULL
PD_PtL_pre	double	YES		NULL
PD_PtL_1hr	double	YES		NULL
PD_PtL_2hr	double	YES		NULL
PD_PtL_4hr	double	YES		NULL
PD_PtL_8hr	double	YES		NULL
PD_PtL_24hr	double	YES		NULL
PD_PtL_48hr	double	YES		NULL
PD_PtL_5day	double	YES		NULL
PD_PtL_7day	double	YES		NULL
PtL_3_mos	double	YES		NULL
PD_PtM_pre	double	YES		NULL
PD_PtM_1hr	double	YES		NULL
PD_PtM_2hr	double	YES		NULL
PD_PtM_4hr	double	YES		NULL
PD_PtM_8hr	double	YES		NULL
PD_PtM_24hr	double	YES		NULL
PD_PtM_48hr	double	YES		NULL
PD_PtM_5day	double	YES		NULL
PD_PtM_7day	double	YES		NULL
PtM_3_mos	double	YES		NULL
PD_PtN_pre	double	YES		NULL
PD_PtN_1hr	double	YES		NULL
PD_PtN_2hr	double	YES		NULL
PD_PtN_4hr	double	YES		NULL
PD_PtN_8hr	double	YES		NULL
PD_PtN_24hr	double	YES		NULL
PD_PtN_48hr	double	YES		NULL
PD_PtN_5day	double	YES		NULL
PD_PtN_7day	double	YES		NULL
PtN_3_mos	double	YES		NULL

GeneExprData.cluster_data table contains 4234 records

CSE487/587: Information Structures - Spring 2004
Globus Toolkit 3.0
Installation, Configuration and Verification Instructions (on CSE Machines)

Installation:

1. Download the GT3 Core Binary bundle, into /**projects/Spring_2004/cse487/*username*** on yeager/pollux server, from http://www-unix.globus.org/ftppub/gt3/3.0/3.0.2/gt3.0.2-core-bin.tar.gz
2. Go to your project space and unpack the GT3 core bundle:
   cd /projects/Spring_2004/cse487/*username*
   gunzip gt3.0.2-core-bin.tar.gz
   tar xvf gt3.0.2-core-bin.tar
   (Run **gtar xvf gt3.0.2-core-bin.tar** in case of checksum error occurs with tar command)
   This creates a directory named ogsa-3.0.2 containing the files of gt3 core.

Configuration:

3. Go to your home directory and edit the **.cshrc** file. Add the following lines
   setenv JAVA_HOME /usr/j2se
   setenv ANT_HOME /projects/bina/ant-1.6
   setenv GLOBUS_LOCATION /projects/Spring_2004/cse487/*username*/ogsa-3.0.2
   set path = ($path $ANT_HOME/bin $JAVA_HOME/bin $GLOBUS_LOCATION/bin)
   Save the .cshrc file and do '**source .cshrc**' at the command prompt. Now the path variables are updated with the directories of Globus Toolkit, Java and Ant installations. Now, go to **$GLOBUS_LOCATION** directory to run the remaining configuration and verification instructions.
4. Port configuration:
   i. Edit **ogsa.properties**
   ii. Choose a 5 digit port number and assign it to **service.port** property replacing the value 8080.
   iii. Save and close the file
5. To generate the command-line scripts, run:
   ant setup
   The scripts are generated in the $GLOBUS_LOCATION/bin directory. Additional scripts for compiling and running grid service clients can be downloaded from:
   http://www.cse.buffalo.edu/~va8/cse4587/utils/globus-java-util.zip
   and extracted into $GLOBUS_LOCATION/bin directory.
6. The setenv scripts can be used to set the proper classpath environment variable in order to launch a Java class that uses gt3 core packages, from the command-line. To run the setenv script, execute
   source $GLOBUS_LOCATION/setenv.csh
   (If you get "Word too long" error, please use globus-java scripts downloaded in step 5.
   Details at: http://www.cse.buffalo.edu/~va8/cse4587/utils/gt3-util-readme.htm)

Verification:

7. Build and deploy the samples in the core package.
   ant samples
   ant deployGuide

8. Run the standalone service container by typing:
   ant startContainer (or)
   globus-start-container
   The container starts up listening to the port specified as service.port in ogsa.properties and lists all the services that are currently deployed on it.

GUI Client:

9. Start the service browser GUI by typing:
   ant gui (or)
   globus-service-browser
   The grid service browser displays the list of the services that are currently displayed on the container
10. Select and double click on Basic Counter Factory Service. (Scroll down on the new window and) click on Create Instance button to create a service instance for testing.
11. A new window with the created instance would show up. Enter a number in the text box and click on Add/Subtract. The result of counter would be shown (as in a calculator). Once tested, click on 'Close' to close the gui client working on the created instance. Or, click on 'Destroy' to destroy the created instance and close.
12. Repeat steps 10 & 11 for other samples (like Weather, Google etc.,) and click on 'Close' to exit the service browser gui.

Command Line Client:

13. Make sure that grid service container is up and running. (If not, refer to step 8 for starting the container)
14. Create service instance using "ogsi-create-service <server url>/<sample factory service name> |id|" The <id> is used to distinguish between instances you create under the same factory, and may be omitted in which case the server generates this id. The <server url> is typically http://<host>:<port>/ogsa/services. The <sample factory service name> must be the same name as defined in server-config.wsdd for the service. Example:
    ogsi-create-service http://*host:port*/ogsa/services/guide/counter/CounterFactoryService cal
    (host = service.host , port = service.port as in ogsa.properties)
15. Run command line client, giving it the URL of the endpoint returned by the ogsi-create-service call in step 14. Example:
    If environment variables are set properly using the setenv scripts in step 6:
    **java org.globus.ogsa.guide.impl.CounterClient **
    http://*host:port* /ogsa/services/guide/counter/CounterFactoryService/calc add 10
    If environment variables are NOT set:
    **globus-java org.globus.ogsa.guide.impl.CounterClient **
    http://*host:port* /ogsa/services/guide/counter/CounterFactoryService/calc add 10
16. Stop the grid container:
    ant stopContainer (or)
    globus-stop-container

For more details about configuring the grid container, running the samples and writing a grid service, refer to the Globus Toolkit User's guide and Programmer's guide at:
http://www-unix.globus.org/toolkit/documentation.html

CSE487/587: Information Structures - Spring 2004
Globus Toolkit 3.0
Installation, Configuration and Verification Instructions (on Windows)

Prepared by Vijayram Arthanari (va8@cse.buffalo.edu)

1. Download:
   i. Java 1.4.2 SDK (installer) from http://java.sun.com/j2se/1.4.2/download.html
   ii. Apache Ant 1.6.1 (zip file) from http://ant.apache.org/bindownload.cgi
   iii. GT3 Core Binary from http://www-unix.globus.org/ftppub/gt3/3.0/3.0.2/gt3.0.2-core-bin.tar.gz
2. Setting Environment Variables
   i. Go to Start->Settings->Control Panel
   ii. Click on System Icon
   iii. Select Advanced tab
   iv. Click on Environment Variables
   v. In System Variables section, Click 'New' button to create a new Environment variable. Click 'Edit' button to edit an existing Environment variable
   vi. Create and set the Environment variables
   GLOBUS_ROOT = c:\grid
   ANT_HOME = %GLOBUS_ROOT%\ant
   JAVA_HOME =%GLOBUS_ROOT%\java
   GLOBUS_LOCATION =%GLOBUS_ROOT%\ogsa-3.0.2
   vii. Add to PATH Environment Variable
   %JAVA_HOME%\bin; %ANT_HOME%\bin; %GLOBUS_LOCATION%\bin
3. Create GLOBUS_ROOT directory (c:\grid) and install Java 1.4.2 into c:\grid\java.
4. Extract the Ant zip file into c:\grid. Rename the apache-ant-1.6.1 folder (created in c:\grid) to ant.
5. Extract the GT3 archive into c:\grid using WinZip. This would create a directory named ogsa-3.0.2 containing the files of gt3 core.
6. Port configuration:
   i. Edit the file **ogsa.properties** in %GLOBUS_LOCATION%.
   ii. Choose a 5 digit port number and assign it to **service.port** property replacing the value 8080.
   iii. Save and close the file
7. To generate the command-line batch files, open Command Prompt and run:
   cd %GLOBUS_LOCATION%
   ant setup
8. The setenv batch scripts can be used to set the proper classpath environment variable in order to launch a Java class that uses gt3 core packages, from the command-line. To run the setenv script, execute (continue using the command prompt started in step 7):
   setenv.bat
   Note: Run setenv.bat every time a new command prompt window is opened to set the environment
9. Build and deploy the samples in the core package.
   ant samples
   ant deployGuide
10. Run the standalone service container by typing:
   ant startContainer (or)
   globus-start-container

The container starts up listening to the port specified as service.port in ogsa.properties and lists all the services that are currently deployed on it.

GUI Client:

11. Start the service browser GUI by open Command Prompt and typing:
    cd %GLOBUS_LOCATION%
    ant gui (or)
    globus-service-browser
    The grid service browser displays the list of the services that are currently displayed on the container
12. Select and double click on Basic Counter Factory Service. (Scroll down on the new window and) click on Create Instance button to create a service instance for testing.
13. A new window with the created instance would show up. Enter a number in the text box and click on Add/Subtract. The result of counter would be shown (as in a calculator). Once tested, click on 'Close' to close the gui client working on the created instance. Or, click on 'Destroy' to destroy the created instance and close.
14. Repeat steps 12 & 13 for other samples (like Weather, Google etc.,) and click on 'Close' to exit the service browser gui.

Command Line Client:

15. Make sure that grid service container is up and running. (If not, refer to step 10 for starting the container)
16. Create service instance using "ogsi-create-service <server url>/<sample factory service name> |id|" The <id> is used to distinguish between instances you create under the same factory, and may be omitted in which case the server generates this id. The <server url> is typically http://<host>:<port>/ogsa/services. The <sample factory service name> must be the same name as defined in server-config.wsdd for the service. Example (run in command prompt):
    ogsi-create-service http://*host:port*/ogsa/services/guide/counter/CounterFactoryService cal
    (host = service.host , port = service.port as in ogsa.properties)
17. Run command line client, giving it the URL of the endpoint returned by the ogsi-create-service call in step 16. Example (type whole command on single line):
    java org.globus.ogsa.guide.impl.CounterClient
    http://*host:port* /ogsa/services/guide/counter/CounterFactoryService/calc add 10
    Note: Run **%GLOBUS_LOCATION%\setenv.bat** before running the client to set the environment variables appropriately.
18. Stop the grid container:
    ant stopContainer (or)
    globus-stop-container

For more details about configuring the grid container, running the samples and writing a grid service, refer to the Globus Toolkit User's guide and Programmer's guide at:
http://www-unix.globus.org/toolkit/documentation.html

GridForce: A Comprehensive Model for Improving the Technical Preparedness of our Workforce for the Grid

B. Ramamurthy
CSE Department
University at Buffalo, SUNY
Amherst, NY 14260
716-645-3180 (108)
bina@cse.buffalo.edu
http://www.cse.buffalo.edu/~bina

Abstract

An enormous challenge when the Internet matured into a mainstream technology was meeting the information technology workforce needs in a competitive business environment. In anticipation of a similar scenario for upcoming grid technology we are in the process of implementing a comprehensive multi-tier NSF-supported adaptation of grid technology in education. The project addresses the above mentioned challenge at three important levels of our educational system: the undergraduate, the graduate and the industrial training. Our grid technology-based curriculum has been developed for a sequence of two new courses for senior level undergraduates. The same courses would be taught at the graduate level with emphasis on research. Additionally, seminars are planned for spreading grid awareness to the local businesses and industries by using domain-dependent grid applications. This paper presents the details of the model we call ***GridForce*** *(**Grid For Research, Collaboration** and **Education**) and our experiences with its implementation, with the objective of improving the technical preparedness of the workforce for the grid.*

Categories and Subject Descriptors
C.2.4 [Distributed Systems]: Distributed Applications, K.3.2 [Computer and Information Science Education] Computer Science Education, Curriculum.

General Terms

Distributed programming.

Keywords

CS education, design, laboratory experiments.

1. Introduction

The primary goal of *GridForce* is to promote grid awareness and technical readiness among all levels of our workforce. We accomplish this by the following strategies:

(i) Introducing grid computing in the CSE (Computer Science and Engineering) curriculum in the senior-level undergraduate and graduate courses. The courses CSE4/586 Distributed Systems and CSE 4/587 Information Structures are currently being offered as a two-semester course sequence in the CSE department of University at Buffalo.

(ii) Building laboratory prototypes that will support grid application development in the courses noted above. We are currently developing two grid prototypes, one with newer Dell Blades and another with old Sparc4 machines.

(iii) Conducting workshops for strengthening local industry workforce. We plan to offer our first seminar through the Center for Industrial Effectiveness (TCIE) at the University at Buffalo in March 2004.

(iv) Assessing the outcome and making ongoing adjustments. An external consultant is currently assisting in assembling a model of the courses and in carrying out a formal assessment of the effectiveness of the model.

This paper describes the educational model defined and implemented by GridForce. Related grid information and the current status of grid education are discussed in Section 2. Details of the various tiers of the multi-tier GridForce model, its implementation, and our experiences are detailed in Section 3. Available resources for adoption of various GridForce components and outcome assessment details are in Section 4. A summary of significant GridForce

contributions and acknowledgements are found at the end.

2. Project Background

A grid is a network of computational units cooperating to share compute cycles, data and other resources across multiple administrative domains, using an open and standardized service-based framework [8, 24]. Under NSF's Partnership for Advanced Computational Infrastructure (PACI) program [15], the scientific community is in the process of developing a national tera-scale infrastructure for high-performance computing. Prominent industries have eagerly embraced grid computing and are promoting it under different names such as utility computing and on-demand computing [11, 12, 13, 20]. Scientists as well as the practitioners believe that the grid developed for scientific computing is on the brink of making computing freely available as yet another "utility," similar in ease of accessibility to the power grid that supplies electricity and the telephone grid that enables voice communication.

We examined three universities that play a prominent role in shaping the grid technology: the University of Chicago, the University of California at San Diego (UCSD), and the University of Tennessee at Knoxville. The grid is presented as one of the topics in parallel computing courses [1, 2, 7], or as a seminar course [3]. The courses at UCSD taught by Dr. Fran Berman's research group, CSC160 (Parallel Computation) [2] and CSE225 (High Performance Computing and Computational Grids) [1] focused on high performance computing. We also studied the past offerings of cluster computing courses at the University at Melbourne which focused on parallel implementation of mathematical problems [4]. At the University of Wisconsin, the home of the Condor cluster/grid computing, the list of course offerings did not reflect any undergraduate or graduate courses in grid computing. We also observed that many schools have recently added programs and courses devoted to the emerging field of bioinformatics while they do not even have a single course devoted to grid computing. It is possible that many other schools may offer courses related to the grid but these have not been accessible due to the lack of a publicizing forum. Our GridForce is a comprehensive suite of courses and short courses rather than an entire program. Our experience is that this model is easier to sell to university administrators and can be implemented as a whole or in parts.

3. The GridForce Project

The GridForce project comprises three major components as depicted in the Figure 1. The **courses** (Section 3.1) play a central role with **laboratory infrastructure** (Section 3.2) and **research** (Section 3.3) components providing practical support. We will discuss the courses, laboratory projects and educational aspects in detail. Ongoing research projects will serve as topics for additional paper and are beyond the scope of this paper, the reason for indicating research sub-tree with no branches in the Figure 1. The project duration is for 2 years starting from the Fall 2003 that will allow for scheduling two offerings of each course.

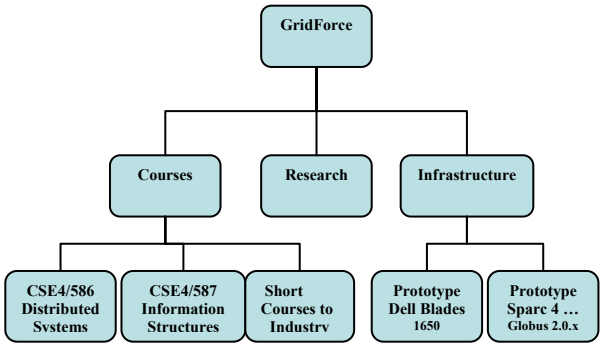


Figure 1: An Organizational Model of GridForce

3.1 Courses

The courses offered under this project form a virtual tiered structure with at least two fundamental tiers as described below. CSE4/586 is a course (undergraduate/graduate pair) in Distributed Systems. CSE4/587 (Information Structures) is a sequel to the Distributed Systems course. Focus of CSE4/586 is on fundamentals of grid computing whereas CSE4/587 deals with application development. Both courses include hands-on laboratory projects. Together these two courses form the Grid Services Developer Tier and are meant for the designers and developers of grid services and components. The next tier is an Industrial Training Tier and is meant for training industrial workforce in grid technology. It combines the salient features of academic courses with customized exercises and domain-dependent applications of interest to the workforce getting trained. Educators can extend this model by adding other tiers as seen appropriate.

3.1.1 Grid Services Developer Tier

Each course has a prescribed text book supplemented by recommended online literature. For

the CSE4/586 we used a traditional distributed systems text by Coulouris et al. [6] as the main text. The first half of the course covered the fundamentals of distributed systems and the second half explored grid fundamentals, grid programming, and grid application model. The highlight of the course is a set of grid programming labs designed to provide hands-on experience for the students. The objectives, course outline, weekly schedule, project descriptions and the all lecture material by date for the current semester can be found at [16]. We briefly explain the lab projects below.

Courses are assumed to span a 14-week semester. The course format involves a lecture that meets 3 X 50 minutes every week and a formal lab session that meets 50 minutes weekly. Prerequisites for the grid courses are data structures, algorithms and programming language courses. The fundamental concepts relevant to the lab and the lab description will be introduced and discussed during the lecture session and will be followed up during the lab session. Lab sessions will also cover details of configuration, packaging and deployment of the applications. Students will have open lab hours and help sessions where technical and debugging help for the projects will be available. Table 1 displays the lab exercises for the first (fundamental) course. The title of the labs and the learning objective are shown in this table. Students will be shown demos on the topic during the regular lecture sessions. Each student will complete an evaluation questionnaire that has both topic-related and learning-related questions. Students will submit online their solutions by a specified date.

Lab1 deals with Webservices [22] which moved the Internet from an information-delivery instrument to a computation-delivering channel. Introduction of the simple request and response model and standardization through SOAP (Simple Object Access Protocol) [22] help students understand (i) the "service" concept in its simplest form and (ii) at the same time realize the need for more sophisticated features such as lifecycle management and notification [19]. We use a simplified version of the Webservices tutorial provided at the Java Webservices site [25] to implement a version of a weather services application. The projects also use the Oracle 9i relational database for persistence, and Ant tool from Apache for building the code. Content-wise the lab also deals with the fundamental distributed systems issues of service registry, discovery and lookup.

Exercise	Topic	Learning Objective
Lab1	Webservices	To understand the alignment of the grid technology to Web Services Definition Language (WSDL) and service description using WSDL.
Lab2	Grid Infrastructure	A simple Java-based grid framework based on [14].
Lab3	Grid Programming	Design and implement a grid-based service using Globus 3.0.2

Table 1 Suggested Lab Projects for CSE4/586 Distributed Systems Course

Lab2 Currently many toolkits such as Globus Toolkit 3.0.2 [10] and Condor 6.5.5 [5] are available to implement the grid framework. However these frameworks are production-quality and are quite complex for the students to understand, deploy and take apart to study and experiment with the code. Lab 2 deals with building a minimal grid framework based on the article "A do-it-yourself framework for grid computing" by Anthony Karre in Java World [14]. Our focus in this project is on the client-side of the grid computing. The framework given in the paper offers these features: (i) machine independence through Java, Apache Tomcat servlet container and Apache Axis SOAP implementation, (ii) Security and scalability achieved through the use of SOAP-based Web services for client-server communication and (iii) task abstraction achieved through the use of jar files, and Java classloader.

In the current assignment of Lab2 we use *SOAP with Attachments API for Java* (SAAJ) [22] instead of the Apache Axis specified in the Karre's paper so as to work at a lower level of abstraction. Students also build a custom classloader and a simple user interface to suit the service (weather service) that has been implemented.

Lab3 deals with implementation of a grid service and an application that uses the grid service. The grid software used for this lab is Globus 3.0.2 core [19]. Students study in detail the Open Grid Services Architecture (OGSA) [9, 23] and Open Grid Services Infrastructure (OGSI) [24]. A comprehensive tutorial on Globus Toolkit (GT3) at [21] and the GT3 core are discussed during the lecture sessions. The students are required to build a many versions of the weather (grid) service from a basic version to a sophisticated one with

features listed in the Globus core distribution. The software that we will be using is the core of the Globus Toolkit 3.0.2. The core of Globus can be downloaded from [19]. Details of the core are available in a white paper on the core services at [19]. This white paper also contains a javadoc-style Grid Services API description, User's Manual and a Programmer's Manual. The user's manual provides the instructions to compile, build, convert, deploy and test a grid service. The programmer's manual provides the details of writing a grid service, the various programming choices available, and deployment description. A samples directory in the core package provides a numerous examples illustrating the various grid services features. Each student installs the Globus core in a special project space allocated for the course and develops and deploys a service on the server. The service is tested using a simple user application.

Projects in the second course CSE4/587 involve applying grid technology to solve problems in specific application domains as shown in Table 2. We have chosen two specific areas of topical interest to grid technologists. The first lab deals with a scientific application in bioinformatics. Lab2 is based on the commercial domain. In this lab we plan to study something topical such as volatility in the stock market and the models for it. Lab3 will be designing and/or modifying a grid-level service such as security and QoS and also defining a business process using a complex grid service. Students will have to come up with original ideas in this lab. There are plans to introduce the Java 2 Enterprise Edition (J2EE) framework for the early projects to allow the students to do a comparative study of the technologies. The author strongly believes the two technologies J2EE and Grid will have to unify to complement and strengthen each other as standard computational framework of the future.

3.1.2 Industrial Training Tier

For this tier, two approaches are possible: (i) an executive summary or a business overview for strategic decision makers or business people and (ii) a hardcore developer point of view. For business people we plan to offer a two-hour breakfast seminar (jointly sponsored by NSF, CSE Department and our University's Industrial Liaison office). Presentations for the developers will be characterized by deeper coverage in a selected area, faster pace and customized mode of delivery. The author of this paper conducts regular training sessions in object-oriented design, programming, and similar topics to the local industry. She has also conducted a how-to of industrial training to educators at national conferences [17, 18]. The author feels that industrial training (say, a six week, 3 hours per week, lab included) focused on a specific topic is one of the best ways to retrain the existing workforce. The industrial training will serve a dual purpose: (i) retraining the IT workforce to be ready for the compute grid, and (ii) field-test the grid technologies for practicality and usability in their respective application environment.

Exercise	Topic	Learning Objective
Lab1	High performance Scientific Application in bioinformatics.	Study requirements of scientific domain and implement. Ex: micro-array analysis
Lab2	Commercial Application	Study requirements of commercial domain and implement. Ex: Stock Market
Lab3	Defining a high-level grid service	Ex: Workflow service, a business process, improvements to QoS

Table 2 Suggested Lab Projects for CSE4/587 Information Structures Course

3.2 Infrastructure

Another important component of the educational model is the laboratory infrastructure. We are currently building two different experimental research and development grids: (i) 40 Sun Microsystem's Sparc 4 discarded computers (originally used for graduate students desktops) with an Ultra Sparc 5 as front-end gatekeeper all running Solaris 8.0 operating system and the Ultra Sparc running Globus 2.0 grid software, and (ii) a grid with four newer Dell blades 1650 hardware, a combination of FreeBSD and Red Hat Linux 9.0 operating systems, and all running Globus Toolkit 3.0.2. Only the computational resources are identified here. We are at a very early stage of addressing storage needs. Additionally both infrastructures are currently undergoing tests for full-scale deployment for student use in the courses for the Spring 2004.

3.2.2 Prototype 1: Using old Sparc 4 Machines

The goal of the infrastructure is to run remote job submissions in a distributed manner on a Sun Microsystems computational cluster running Globus. The grid is primarily composed of 40 Sun Sparc

machines, which form computational nodes, headed by a front-end Sun server running Globus. The internal Class C network is set up using custom NFS, NIS and jumpstart servers. The jumpstart server is an operating system server providing remote Solaris 8 installation for clients over the network. The installation scripts are custom-written facilitating running of jobs in a distributed manner.

The network also has an NFS service running that provides remote file mounts and access. The server exports its *home* and *util* directories for NFS clients in the network. Name service and network information is provided using an NIS+ service running along with the NFS server. The central server runs a custom version of Globus binaries handling remote Job submissions. The custom Globus binary package installed is comprised of binaries of 2.x versions optimized for specific performance issues. The grid certification mechanism is a DOE certification process providing host and user certificates.

3.2.2 Prototype 2: Using Dell blades 1650

This experimental grid is set up as one utility server and three compute nodes. FreeBSD was chosen for the utility server. This server is designed to provide network gateway/firewall services as well as basic UNIX-level account authentication (NIS) and file (NFS) services. Some of the support services for Globus also run on this server. The three compute nodes are running RedHat-9 and Globus Toolkit version 3 (GT3).

3.3 Research

The main focus of our group is applied research that can potentially expand the grid technology to support mainstream applications. Students in the courses described above each worked on a poster that explained a possible application of the grid. Some of the ideas include: (i) agent-based grid security application, (ii) grid application to pick stocks, (iii) income tax return filer grid service based on high-trust of the grid technology and (iv) grid application development environment. We plan to implement these projects as labs for the second course CSE4/587. The details of these projects are topics for future publications from our group.

4. Resources for Adopters

All the material needed for adoption and adaptation of GridForce courses is available on our webpage: www.cse.buffalo.edu/gridforce. An important component of the course model is the outcome assessment process. We have an extensive outcome assessment questionnaire prepared with the help of professional evaluators. For example, an assessment questionnaire for the course CSE4/586 has about 42 multiple choice questions and 4 short answer questions comprehensively covering all the elements of the course model. We will make this and similar forms available for educators to reuse. The details of the infrastructure will be made available as soon the testing is completed.

5. Summary

We have presented a comprehensive model addressing the need to improve awareness and technical preparedness of our workforce. The outcome of this model will impact a wide variety of audiences from undergraduate students to business strategists. All the information related to the model is web-accessible to potential adopters. We have plans to apply our experience to teaching fundamental concepts related to grid and to develop grid-based curriculum for computer architecture courses.

6. Acknowledgements

This work is partially supported by NSF grant CCLI A&I DUE 0311473.
The author acknowledges Ken Smith and Karthikram Venkataramani for help with the Dell and Sparc infrastructures respectively. We thank Center for Computational Research (CCR) at University at Buffalo for the grid expertise they have provided and for hosting the Sparc infrastructure. The author would like to acknowledge the anonymous reviewers for their feedback on the content of paper, and Dr. S. Goldberg for proof reading the paper.

7. References

[1] F. Berman, CSC225 High Performance Distributed Computing and Computational Grids, http://www-cse.ucsd.edu/classes/sp00/cse225/, Spring 2000

[2] F. Berman, CSC160 Parallel Computation, http://juggler.ucsd.edu/~nadya/160/ Fall 2000.

[3] H. Bos, Seminar on Grid Computing, University of Amsterdam, Netherlands, http://www.liacs.nl/~herbertb/courses/grid/, 2001.

[4] R. Buyya, 433-498: Cluster and Grid Computing, http://www.cs.mu.oz.au/~raj/grids/course, 2002.

[5] Condor High Throughput Computing. http://www.cs.wisc.edu/condor/

[6] G. Coulouris, J. Dilmore, and T. Kindberg. Distributed Syste: oncepts and Design, Thirds Edition, Addison-Wesley Publication, 2001.

[7] J. Dongarra, CSE594: Understanding Parallel Architectures: From Theory to Practice, University of Tennessee at Knoxville, Spring 2002.

[8] I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations**.** *International J. Supercomputer Applications*, 15(3), 2001, See http://www.globus.org/research/papers/anatomy.pdf

[9] I. Foster, C. Kesselman, J. Nick, S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002. See http://www.globus.org/research/papers/ogsa.pdf

[10] Globus Toolkit 3.0: Open Grid Services Architecture, http://www.globus.org/toolkit/

[11] Grid Computing at IBM, http://www-1.ibm.com/grid/

[12] Grid Computing: Making the Global Infrastructure a Reality. Edited by F. Berman, G.C. Fox, A.J.G. Hey, Wiley and Sons, 2003. ISBN: 0-470-85319-0

[13] Information Power Grid (IPG) is NASA's high performance computational grid. http://www.ipg.nasa.gov/, Nov. 2002

[14] A. Karre. A Do-it Yourself Framework for Grid Computing. http://www.javaworld.com/javaworld/jw-04-2003/jw-0425-grid.html, Java World, April 2003.

[15] NSF: NSF Announces Continuing Steps to Enhance Cyberinfrastrucure, http://www.cise.nsf.gov/news/cybr/cybr2.cfm October 2003.

[16] B. Ramamurthy. GridForce: Grid for Research, Collaboration and Education. http://www.cse.buffalo.edu/gridforce/index.htm, 2003.

[17] B. Ramamurthy. A Multi-tier Adaptation of Grid Computing in Computer Science Curriculum, to be presented at SIGCSE 2004 poster session, Norfolk, VA, 2004.

[18] B. Ramamurthy. Industrial Training Know-how for Educators. Workshop presented at ACM SIGCSE 2000, Austin, TX.

[19] T. Sandholm and J.Gawor. Globus Toolkit 3 Core – A grid Service Container Framework, Globus Project, July 2003, http://www-unix.globus.org/toolkit/3.0/ogsa/docs/gt3_core.pdf

[20] P. Shread, Sun Sees Campus Grids as next Stage, Grid Computing Planet on-line magazine, January 2002. See http://www.gridcomputingplanet.com/news/article/0,,3281_952091,00.html.

[21] B. Sotomayor. The Globus Toolkit3 Programmer's Tutorial, 2003. http://www.casa-sotomayor.net/gt3-tutorial/

[22] K. Topley. Java Web Services in a Nutshell, O'Reilly Publishers, June 2003.

[23] Towards Globus Toolkit 3.0: Open Grid Services Architecture, http://www.globus.org/ogsa/.

[24] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, D. Snelling, "Open Grid Services Infrastructure (OGSI) Version 1.0."; Global Grid Forum Draft Recommendation, 6/27/2003.

[25] Web Services Tutorial, Sun Micro Systems Inc., http://java.sun.com/webservices/tutorial.html, 2003.

GridForce:

A Multi-tier Approach to Prepare our Workforce for Grid Technology

Bina Ramamurthy
CSE Department
University at Buffalo (SUNY)
201 Bell Hall, Buffalo, NY 14260
716-645-3180 (108)
bina@cse.buffalo.edu
http://www.cse.buffalo.edu/gridforce
Partially Supported by NSF DUE CCLI A&I Grant 0311473

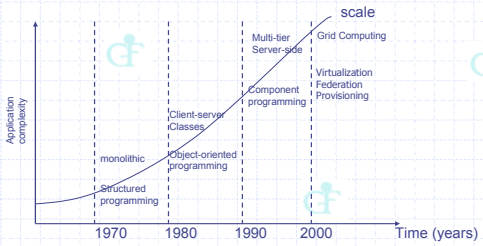
Introduction

- We present an adaptation of the upcoming grid technology in CS-based curriculum.
- Courses span multiple tiers:
  - CS undergraduate senior level (CSE486, CSE487)
  - CS graduate entry level (CSE586, CSE587)
  - Entry level scientists and engineers
  - Seminars to industry
- Goal is to improve technical preparedness of our workforce for grid technology.

Topics for Discussion

- What is grid technology? (General, Technical)
- Why grid technology?
- Adaptation of Grid Technology to CS Curriculum
- GridForce Project
- Courses: Curriculum CSE4/586, CSE4/587
- Lab Exercises: problem, approaches to solution, code base for solution
- Fundamental concepts covered
- Technologies and tools covered
- Preliminary Assessment of Effectiveness of Adaptation
- Grid infrastructure
  - Reusing old hardware (SparcGrid)
  - Grid with newer hardware (LinuxGrid)
- Industrial outreach
- Challenges in Adaptation
- Significant contributions of GridForce

Software Trends



Grid Technology

- Emerging enabling technology.
- Natural evolution of distributed systems and the Internet.
- Middleware supporting network of systems to facilitate sharing, standardization and openness.
- Infrastructure and application model dealing with sharing of compute cycles, data, storage and other resources.
- Promoted by NSF through its Network Middleware Initiative (NMI version 4).
- Publicized by prominent industries as on-demand computing, utility computing, etc.
- Move towards delivering "computing" to masses similar to other utilities (electricity and voice communication).

Adaptation of Grid Technology to CS-Curriculum

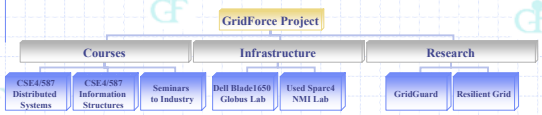
- Introduce grid technology into the CS undergraduate curriculum.
  - Goal: Design and deploy grid services and applications. Study grid application models.
  - Focus on lab exercises to illustrate fundamental grid concepts, and development of grid services and applications.
- Conduct seminars to industry.
  - Goal: Overview of grid technology landscape and its alignment to common technologies and application models.
  - Examine case-studies to expose potential uses of grid.
- Introduce grid to potential users of grid
  - Goal: Publicize the usage models of grid.
  - Use grid infrastructure for entry level courses in Sciences and Engineering.

GridForce

- Our adaptation is evolving into a comprehensive framework we call GridForce (**Grid For Collaboration and Education**):
  - Course curriculum,
  - Laboratory exercises (labs),
  - Infrastructure to support labs,
  - Research projects,
  - Industrial outreach.

GridForce Project Framework

GridForce is a comprehensive framework to adapt grid Computing into undergraduate curriculum.



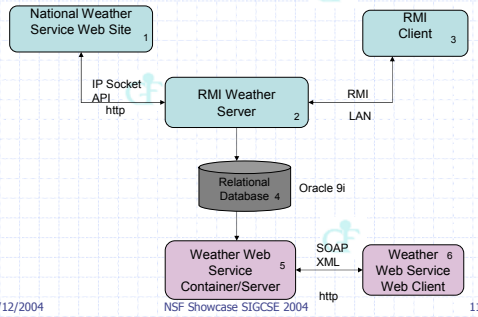
Courses: CSE4/586 Distributed Systems

- Learning outcome: fundamental concepts of distributed systems and grid.
- Lab exercises to support concepts:
  - Three-tier client server system using Web Services.
  - A simple grid framework.
  - Design and implementation of a grid service.
- Text: Distributed Systems: Concepts and Design (3rd Edition) by George Coulouris, Jean Dollimore, Tim Kindberg, Addison-Wesley Inc., 2000.
- Prerequisites: Data structures and algorithms, object-oriented design and development, working knowledge of Java.

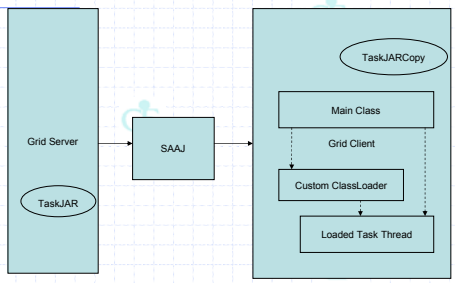
Courses: CSE 4/586: Lab Exercises

Exercise	Topic	Learning Objective
Lab1	Webservices	To understand the alignment of the grid technology to Web Services, WS Definition Language (WSDL) and service description using WSDL.
Lab2	Grid Infrastructure	A Webservices based grid.
Lab3	Grid Programming	Design and implement a grid-base service using Globus 3.0.2

Lab1: Web Services

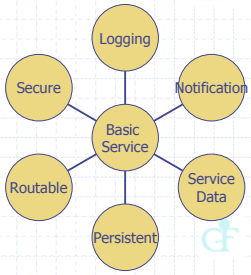


Lab2: Web Services-based Grid



Based on A simple do-it-yourself framework for grid computing by Anthony Karre

Lab3: Grid Service using Globus



Courses: CSE4/587:Information Structures

- Learning outcome: understand grid infrastructure and grid architecture, design and deploy grid services and grid applications.
- Lab exercises support:
  - Enterprise application using Java 2 Enterprise Edition.
  - Grid application in high performance area.
  - Service-oriented grid application.
- Text: "The Grid 2: Blueprint for a New Computing Infrastructure" by Ian Foster , Carl Kesselman, Morgan-Kauffmann, 2004.
- Prerequisites: Data structures and algorithms, object-oriented design and development, working knowledge of Java, fundamentals of client/server architectures.

Courses: CSE4/587: Lab Exercises

Exercise	Topic	Learning Objective
Lab1	Commercial Application	Study requirements of a commercial domain and implement an application.
Lab2	High performance Application.	Study requirements of scientific/business domain and implement compute intensive application.
Lab3	Defining a high-level grid service	Workflow service, a business process, improvements to QoS

Fundamental Grid Concepts Covered

- N-tier client server system
- Web applications
- Component programming
- Grid service
- Open grid services architecture (OGSA)
- Open grid services infrastructure (OGSI)
- Logging, notification and service data
- Virtualization, federation, provisioning.

Fundamental Knowledge Areas

- From ACM Curricula 2001:
- NC1: Net-centric computing: Distributed Systems
- NC5: Building web applications
- SE2: Using APIs
- SE3: Software tools and environments
- SE9: Component-based Programming
- SE12: Specialized system development
- CN4: High Performance computing

Tools and Technologies covered

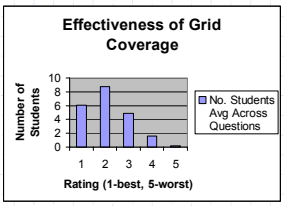
- Technologies include:
  - XML and SOAP
  - Web services (service definition, implementation and deployment)
  - Java 2 Enterprise Edition (Enterprise Java Beans)
  - Globus Toolkit 3.0.2 (GT3)
- Tools include:
  - UML (Unified Modeling Language) for design representation
  - Apache Ant: XML-based build tool

Outcome Assessment

- End of the course questionnaire is used to assess the effectiveness of the courses.
  - prepared by an external consultant (Dr. Neal of Erie Community College)
- Mainly multiple choice questions with a few short answer questions.
- The overall effectiveness of the CSE4/586 course as measured by the average of ratings for the 42 questions. (1- best to 5-worst) is shown.

Effectiveness of Adaptation (CSE486/586)

- Survey with 42 multiple choice questions pertaining to coverage of grid in CSE4/586.
- Average rating among 20 students who took the survey is shown.

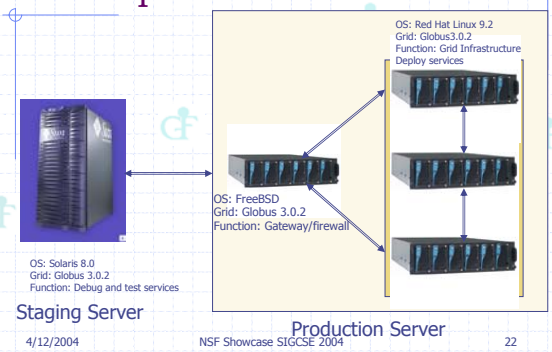


- External evaluator identified 7 areas for improvement.
  - Two of these pointed to unavailability of grid programming infrastructure for students to use.
  - We have remedied this situation with more than one grid lab infrastructure.

Infrastructure: LinuxGrid

- Goal: To facilitate development of service-oriented applications for the grid.
- Two major components: Staging server and Production grid Server.
- Grid application are developed and tested on staging server and deployed on a production server.
- Production grid server:
  - Three compute nodes with Red Hat Linux and Globus 3.0.2 instance.
  - One utility gateway node with Free BSD and Globus 3.0.2.
- Lab 1 will be deployed on the staging server, Lab 3 on the production grid.

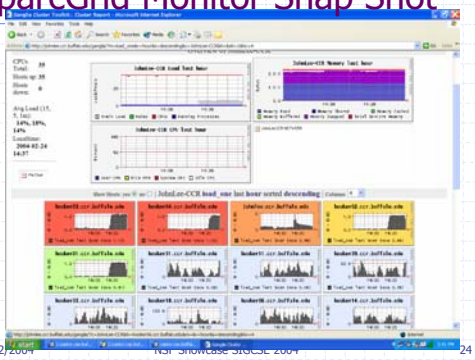
Development Environment



Infrastructure: SparcGrid

- Goal: To run jobs submitted in a distributed manner on a Condor-based computational cluster Condor.
- Composed of 50 Sun recyclable used Sparc4 machines, which form computational nodes, headed by a front-end Sun server.
- The installation scripts are custom-written facilitating running of jobs in a distributed manner.
- Partially supported by Center for Computational Research (CCR).
- Lab2 will be developed, deployed and tested on this infrastructure.

SparcGrid Monitor Snap Shot



Industrial Training Tier

- In collaboration with The Center for Industrial Effectiveness (TCIE) of University at Buffalo (UB).
- A Two-hour breakfast seminar introducing grid technology to business decision makers and potential adopters.
- Topics include:
  - Grid application domains
  - Grid application models
  - How relate grid to currently used technologies and
  - Details of the grid infrastructures currently operational at UB.

Resources for Adoption

- GridForce is modular that all or parts of it can be adopted by educators and practitioners.
- Course curriculum, project descriptions, solutions and lecture material are available online at www.cse.buffalo.edu/gridforce

Challenges in Adapting Grid Technology

- Adding to existing curriculum.
  - Solution: Addressed through labs.
- Adapting to versions of software and toolkits.
  - Solution: Discusses differences; however work with the latest version.
- Managing students with deficiencies in their technical background.
  - Solution: Special coverage during recitations.
- Maintaining grid infrastructure for hands-on labs.
  - Solution: Include a system administration support person in the implementation of curriculum.
- Lack of appropriate text books:
  - Solution: Good area for anybody with expertise to write a book.

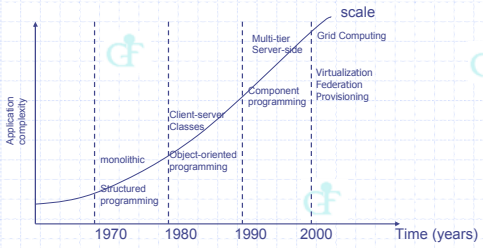
Contributions

- Expected number of students *directly* impacted: 200+ per year. With proper dissemination this will be much higher.
- Comprehensive framework covering grid technology in course curriculum, lab exercises, infrastructure to support labs, and applied research.
- Coverage addressing needs at various levels: undergraduate, graduate to industrial workforce and decision makers.
- Offers a model for adaptation of ever changing technology landscape.

A Multi-tier Adaptation of Grid Computing in Computer Science Curriculum

Bina Ramamurthy
CSE Department
University at Buffalo (SUNY)
201 Bell Hall, Buffalo, NY 14260
716-645-3180 (108)
bina@cse.buffalo.edu
http://www.cse.buffalo.edu/gridforce
Partially Supported by NSF DUE CCLI A&I Grant 0311473

Software Trends



Grid Technology

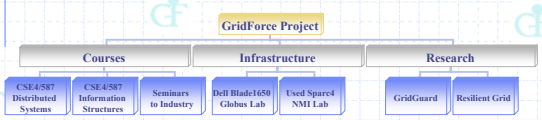
- Emerging enabling technology.
- Natural evolution of distributed systems and the Internet.
- Middleware supporting network of systems to facilitate sharing, standardization and openness.
- Infrastructure and application model dealing with sharing of compute cycles, data, storage and other resources.
- Promoted by NSF through its Network Middleware Initiative (NMI version 4).
- Publicized by prominent industries as on-demand computing, utility computing, etc.
- Move towards delivering "computing" to masses similar to other utilities (electricity and voice communication).

Adaptation of Grid Technology to CS-Curriculum

- Introduce grid technology into the CS undergraduate curriculum.
  - Goal: Design and deploy grid services and applications. Study grid application models.
  - Focus on lab exercises to illustrate fundamental grid concepts, and development of grid services and applications.
- Conduct seminars to industry.
  - Goal: Overview of grid technology landscape and its alignment to common technologies and application models.
  - Examine case-studies to expose potential uses of grid.
- Introduce grid to potential users of grid
  - Goal: Publicize the usage models of grid.
  - Use grid infrastructure for entry level courses in Sciences and Engineering.

GridForce Project Framework

GridForce is a comprehensive framework to adapt grid Computing into undergraduate curriculum.



Courses: CSE4/586 Distributed Systems

- Learning outcome: fundamental concepts of distributed systems and grid.
- Lab exercises to support concepts:
  - Three-tier client server system using Web Services.
  - A simple grid framework.
  - Design and implementation of a grid service.
- Text: Distributed Systems: Concepts and Design (3rd Edition) by George Coulouris, Jean Dollimore, Tim Kindberg, Addison-Wesley Inc., 2000.
- Prerequisites: Data structures and algorithms, object-oriented design and development, working knowledge of Java.

Courses: CSE 4/586: Lab Exercises

Exercise	Topic	Learning Objective
Lab1	Webservices	To understand the alignment of the grid technology to Web Services, WS Definition Language (WSDL) and service description using WSDL.
Lab2	Grid Infrastructure	A simple Java-based grid framework using custom ClassLoader.
Lab3	Grid Programming	Design and implement a grid-base service using Globus 3.0.2

Courses: CSE4/587:Information Structures

- Learning outcome: understand grid infrastructure and grid architecture, design and deploy grid services and grid applications.
- Lab exercises support:
  - Enterprise application using Java 2 Enterprise Edition.
  - Grid application in high performance area.
  - Service-oriented grid application.
- Text: "The Grid 2: Blueprint for a New Computing Infrastructure" by Ian Foster , Carl Kesselman, Morgan-Kauffmann, 2004.
- Prerequisites: Data structures and algorithms, object-oriented design and development, working knowledge of Java, fundamentals of client/server architectures.

Courses: CSE4/587: Lab Exercises

Exercise	Topic	Learning Objective
Lab1	Commercial Application	Study requirements of a commercial domain and implement an application.
Lab2	High performance Application.	Study requirements of scientific/business domain and implement compute intensive application.
Lab3	Defining a high-level grid service	Workflow service, a business process, improvements to QoS

Fundamental Knowledge Areas

- From ACM Curricula 2001:
- NC1: Net-centric computing: Distributed Systems
- NC5: Building web applications
- SE2: Using APIs
- SE3: Software tools and environments
- SE9: Component-based Programming
- SE12: Specialized system development
- CN4: High Performance computing

Tools and Technologies covered

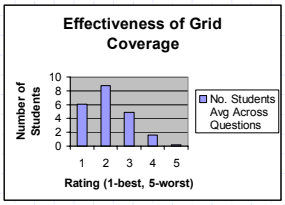
- Technologies include:
  - XML and SOAP
  - Web services (service definition, implementation and deployment)
  - Java 2 Enterprise Edition (Enterprise Java Beans)
  - Globus Toolkit 3.0.2 (GT3)
- Tools include:
  - UML (Unified Modeling Language) for design representation
  - Apache Ant: XML-based build tool

Outcome Assessment

- End of the course questionnaire is used to assess the effectiveness of the courses.
  - prepared by an external consultant (Dr. Neal of Erie Community College)
- Mainly multiple choice questions with a few short answer questions.
- The overall effectiveness of the CSE4/586 course as measured by the average of ratings for the 42 questions. (1- best to 5-worst) is shown.

Effectiveness of Adaptation (CSE486/586)

- Survey with 42 multiple choice questions pertaining to coverage of grid in CSE4/586.
- Average rating among 20 students who took the survey is shown.

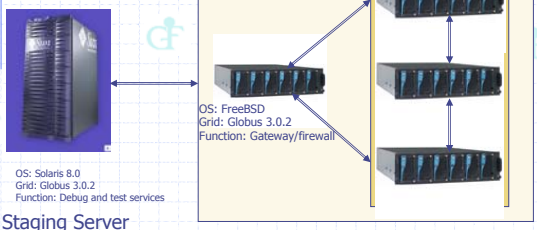


- External evaluator identified 7 areas for improvement.
  - Two of these pointed to unavailability of grid programming infrastructure for students to use.
  - We have remedied this situation with more than one grid lab infrastructure.

Infrastructure: LinuxGrid

- Goal: To facilitate development of service-oriented applications for the grid.
- Two major components: Staging server and Production grid Server.
- Grid application are developed and tested on staging server and deployed on a production server.
- Production grid server:
  - Three compute nodes with Red Hat Linux and Globus 3.0.2 instance.
  - One utility gateway node with Free BSD and Globus 3.0.2.
- Lab 1 will be deployed on the staging server, Lab 3 on the production grid.

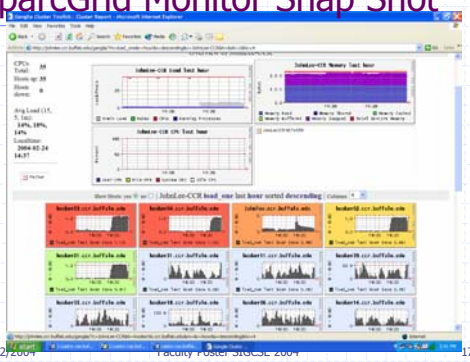
Development Environment



Infrastructure: SparcGrid

- Goal: To run jobs submitted in a distributed manner on a Condor-based computational cluster Condor.
- Composed of 50 Sun recyclable used Sparc4 machines, which form computational nodes, headed by a front-end Sun server.
- The installation scripts are custom-written facilitating running of jobs in a distributed manner.
- Partially supported by Center for Computational Research (CCR).
- Lab2 will be developed, deployed and tested on this infrastructure.

SparcGrid Monitor Snap Shot



Industrial Training Tier

- In collaboration with The Center for Industrial Effectiveness (TCIE) of University at Buffalo (UB).
- A Two-hour breakfast seminar introducing grid technology to business decision makers and potential adopters.
- Topics include:
  - Grid application domains
  - Grid application models
  - How relate grid to currently used technologies and
  - Details of the grid infrastructures currently operational at UB.

Resources for Adoption

- GridForce is modular that all or parts of it can be adopted by educators and practitioners.
- Course curriculum, project descriptions, solutions and lecture material are available online at www.cse.buffalo.edu/gridforce

Contributions

- Expected number of students *directly* impacted: 200+ per year. With proper dissemination this will be much higher.
- Comprehensive framework covering grid technology in course curriculum, lab exercises, infrastructure to support labs, and applied research.
- Coverage addressing needs at various levels: undergraduate, graduate to industrial workforce and decision makers.
- Offers a model for adaptation of ever changing technology landscape.