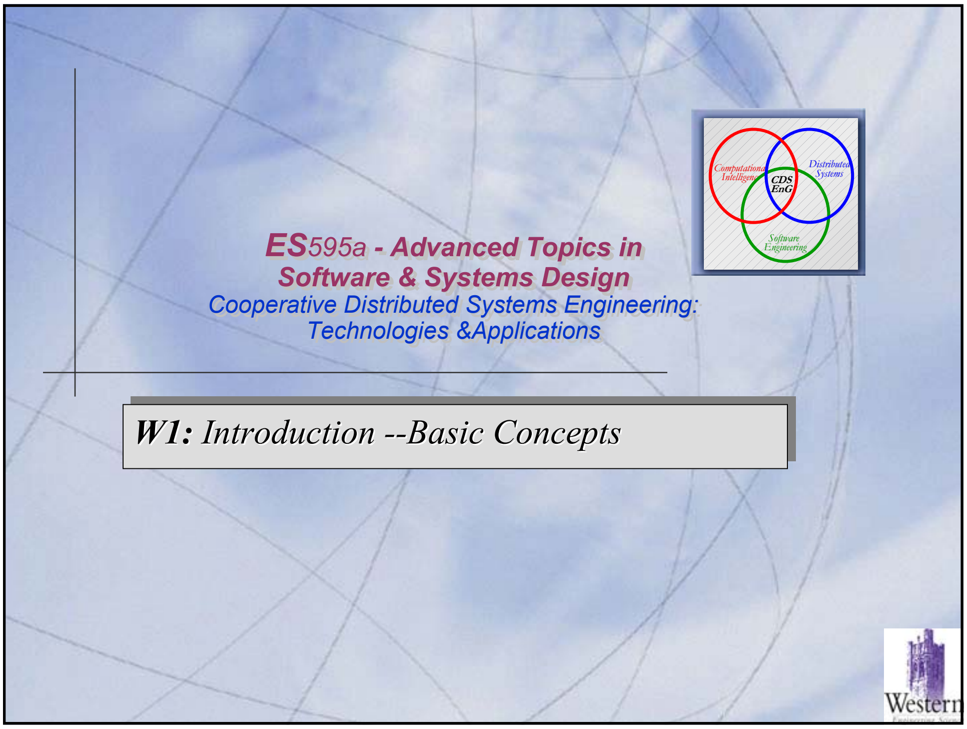


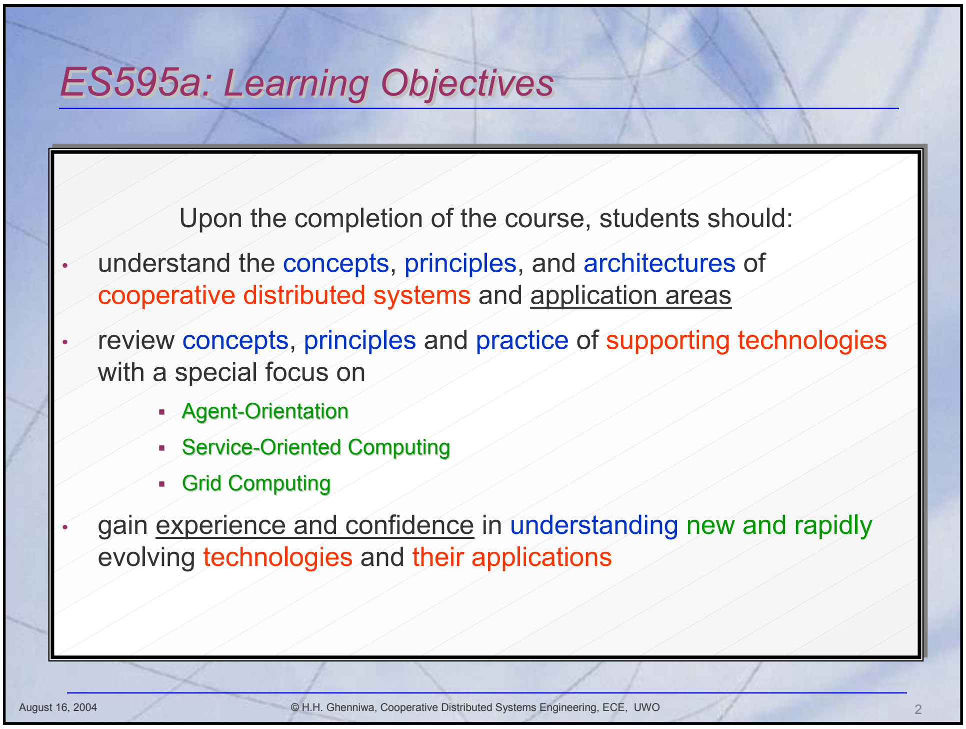
Computational Intelligence
Distributed Systems
CDS EnG
Software Engineering

ES595a - Advanced Topics in Software & Systems Design

Cooperative Distributed Systems Engineering: Technologies &Applications

W1: Introduction --Basic Concepts

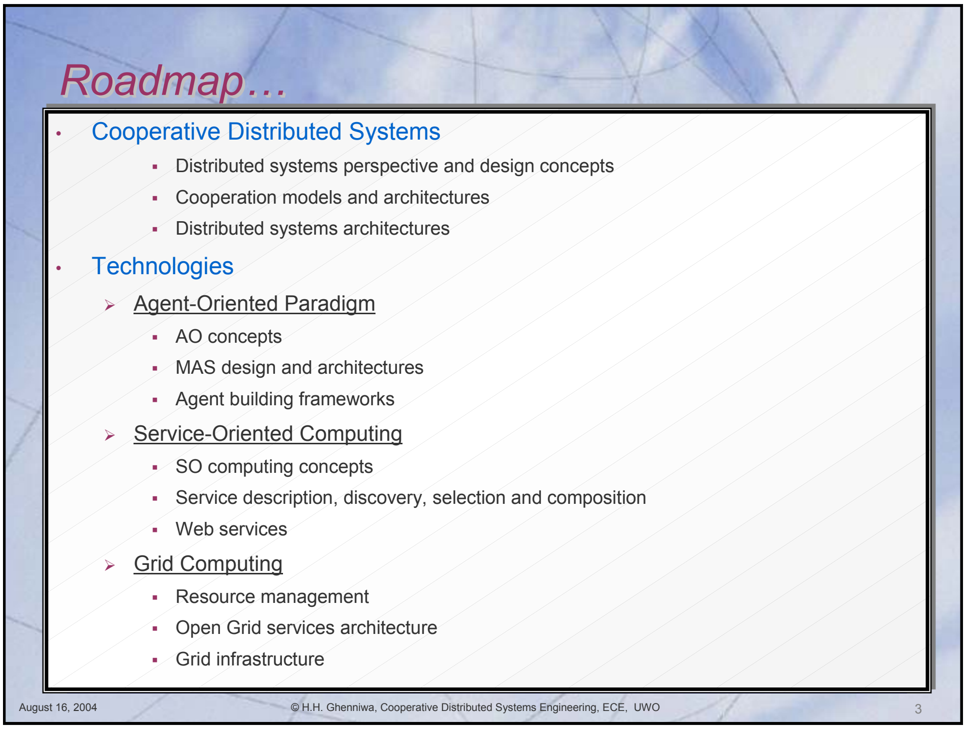
Western



ES595a: Learning Objectives

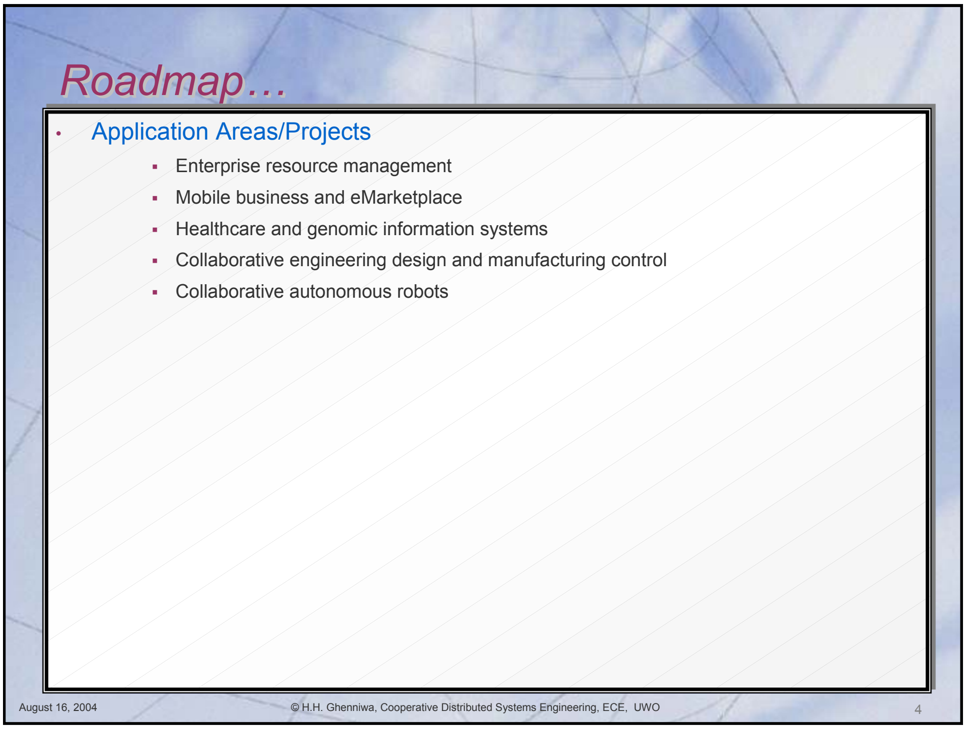
Upon the completion of the course, students should:

- understand the concepts, principles, and architectures of cooperative distributed systems and application areas
- review concepts, principles and practice of supporting technologies with a special focus on
  - Agent-Orientation
  - Service-Oriented Computing
  - Grid Computing
- gain experience and confidence in understanding new and rapidly evolving technologies and their applications



Roadmap…

- Cooperative Distributed Systems
  - Distributed systems perspective and design concepts
  - Cooperation models and architectures
  - Distributed systems architectures
- Technologies
  - Agent-Oriented Paradigm
    - AO concepts
    - MAS design and architectures
    - Agent building frameworks
  - Service-Oriented Computing
    - SO computing concepts
    - Service description, discovery, selection and composition
    - Web services
  - Grid Computing
    - Resource management
    - Open Grid services architecture
    - Grid infrastructure



Roadmap…

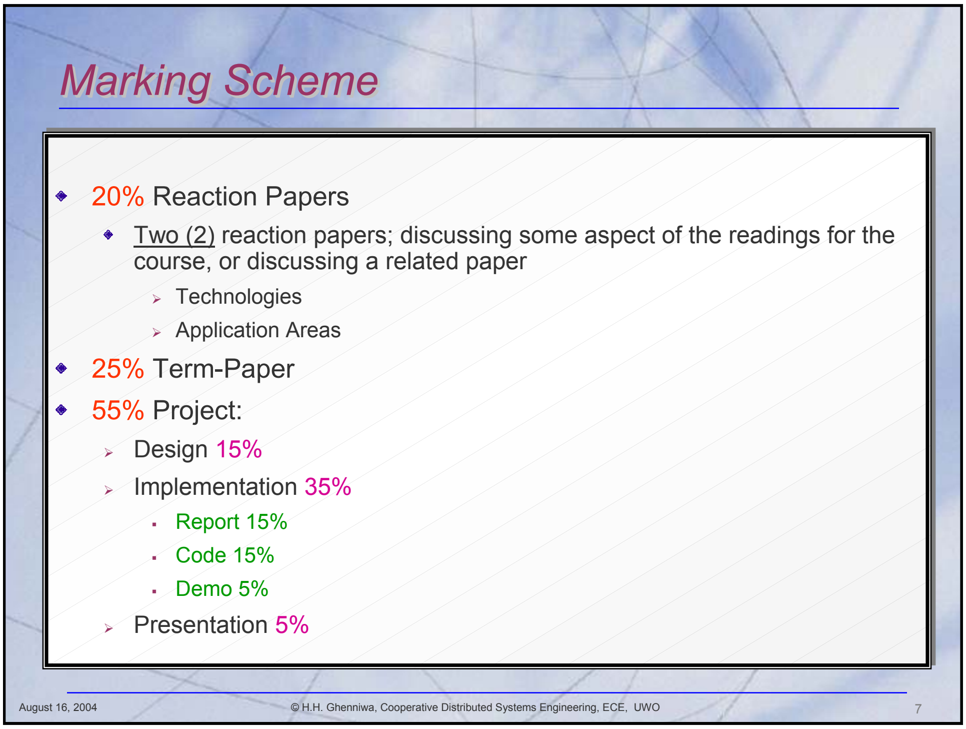
- Application Areas/Projects
  - Enterprise resource management
  - Mobile business and eMarketplace
  - Healthcare and genomic information systems
  - Collaborative engineering design and manufacturing control
  - Collaborative autonomous robots

References

- Course notes, papers and supplementary material will be available on the Class Web site http://instruct.uwo.ca/engin-sc/se595a/index.htm
- An Introduction to Multiagent Systems, Michael Wooldridge, John Wiley & Sons (Chichester, England). ISBN 0 47149691X, 2001
- “Readings in Agents”, Michael Huhns and Munindar Singh (Eds.),1997
- Multiagent Systems Gerhard Weiss (Ed.), MIT Press, 1999
- Service-Oriented Software System Engineering: Challenges And Practices, Z. Stojanovic and A. Dahanayake (Eds.), Idea Group, Inc., 2004
- Service-Oriented Computing, Munindar P. Singh and Michael N. Huhns, Wiley, 2004
- The Grid 2: Blueprint for a New Computing Infrastructure, 2nd Edition, Ian Foster and Carl Kesselmann (Eds), Morgan Kaufmann Publishers, 2004 ISBN: 1-55860-933-4
- Grid Computing: Software Environment and Tools, Omer F. Rana and Jose C. Cunha (Eds), Springer LNCS, 2004.

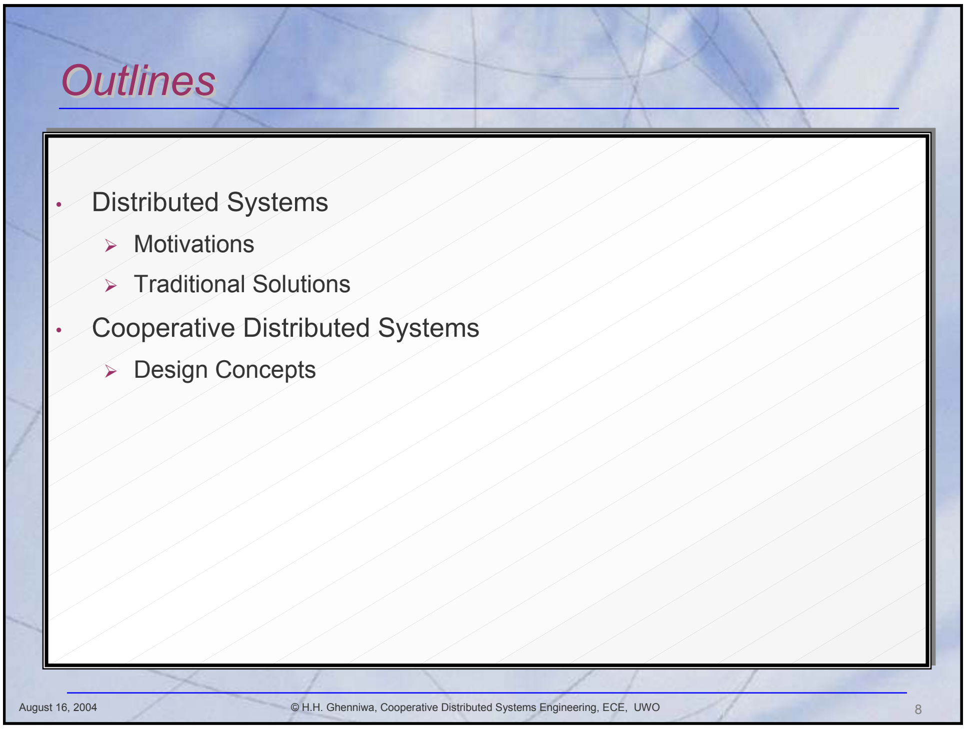
References…*(More)*

- "Agent-Oriented Software Engineering", Ciancarini, P. and Wooldridge, M. (Eds.) Springer-Verlag Lecture Notes in Computer Science Volume 1957, January 2001.
- “Multi-Agent Systems for Concurrent Intelligent Design and Manufacturing” Weiming Shen, Douglas H. Norrie and Jean-Paul Barthes Published by Taylor & Francis



Marking Scheme

- 20% Reaction Papers
  - Two (2) reaction papers; discussing some aspect of the readings for the course, or discussing a related paper
    - Technologies
    - Application Areas
- 25% Term-Paper
- 55% Project:
  - Design 15%
  - Implementation 35%
    - Report 15%
    - Code 15%
    - Demo 5%
  - Presentation 5%



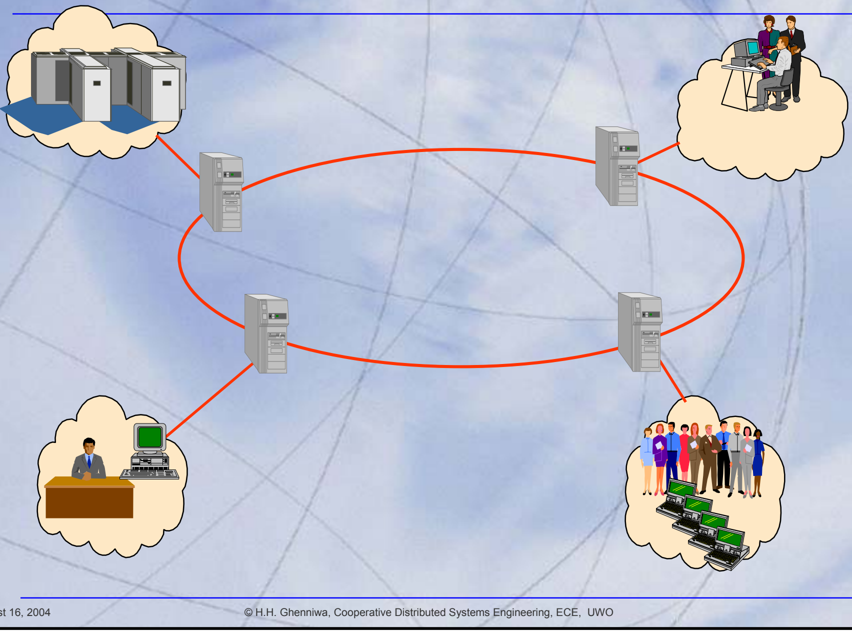
Outlines

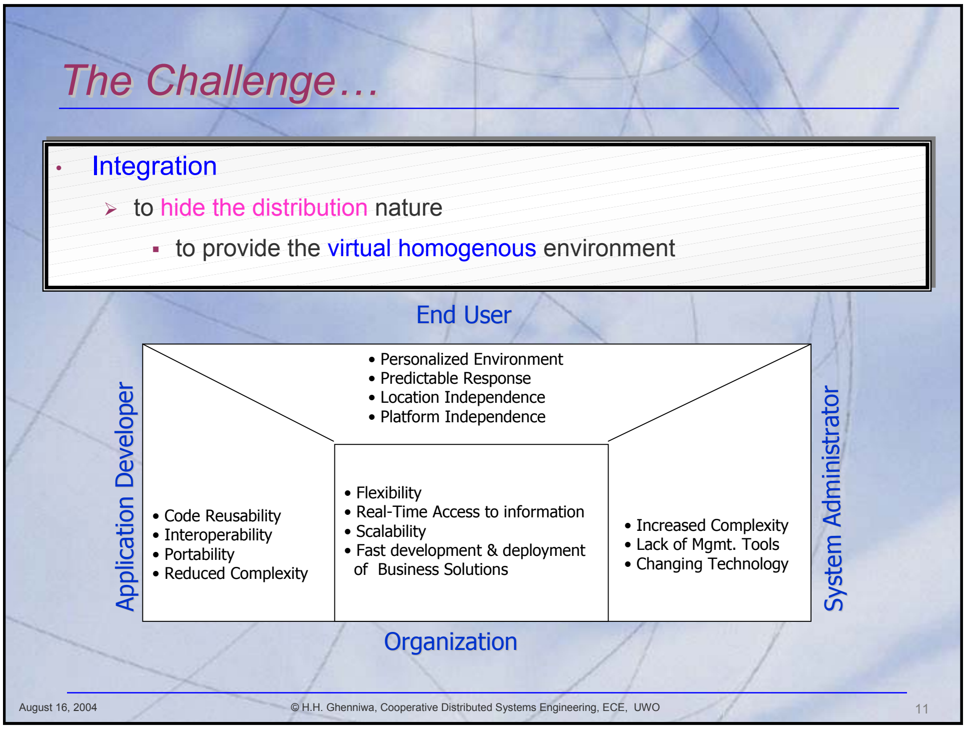
- Distributed Systems
  - Motivations
  - Traditional Solutions
- Cooperative Distributed Systems
  - Design Concepts

Why Distributed Computing?

- The way things should be!
  - customers, suppliers, and companies are at different sites
- Exploitation the power of distribution or special hardware
- Cost
- Globalization
- etc.

Distributed Systems





The Challenge…

- Integration
  - to hide the distribution nature
    - to provide the virtual homogenous environment

End User

- Personalized Environment
- Predictable Response
- Location Independence
- Platform Independence

Application Developer

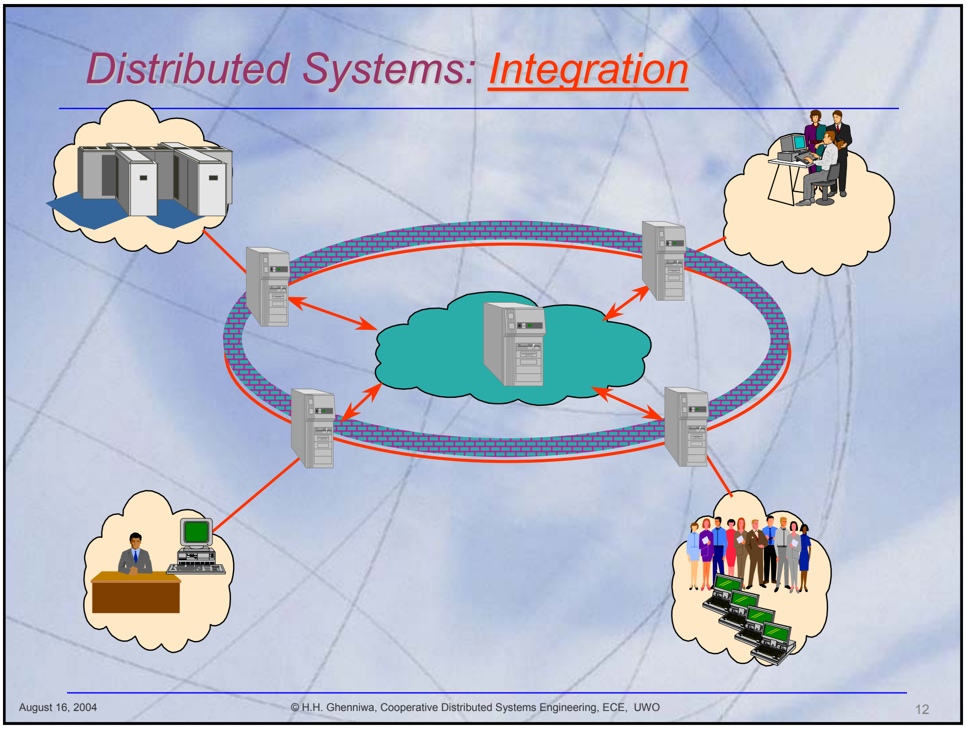
- Code Reusability
- Interoperability
- Portability
- Reduced Complexity

- Flexibility
- Real-Time Access to information
- Scalability
- Fast development & deployment of Business Solutions

System Administrator

- Increased Complexity
- Lack of Mgmt. Tools
- Changing Technology

Organization

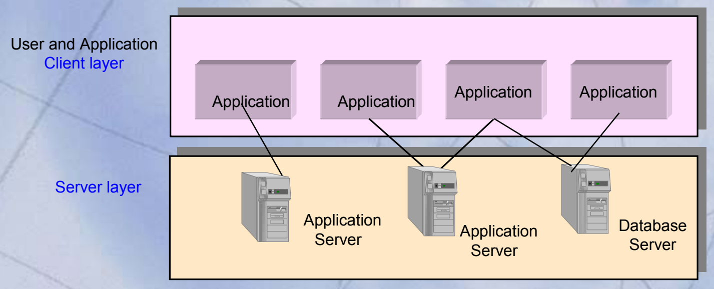


Distributed Systems: Integration

The Magic Bullet → Middleware!

- A layer between
  - the application and
  - the underlying complexities of
    - the network, the host operating system, and any resource servers
  - It makes different and possibly heterogeneous platforms appear the same to an application

Client Server Architecture



User and Application
Client layer

Application

Application

Application

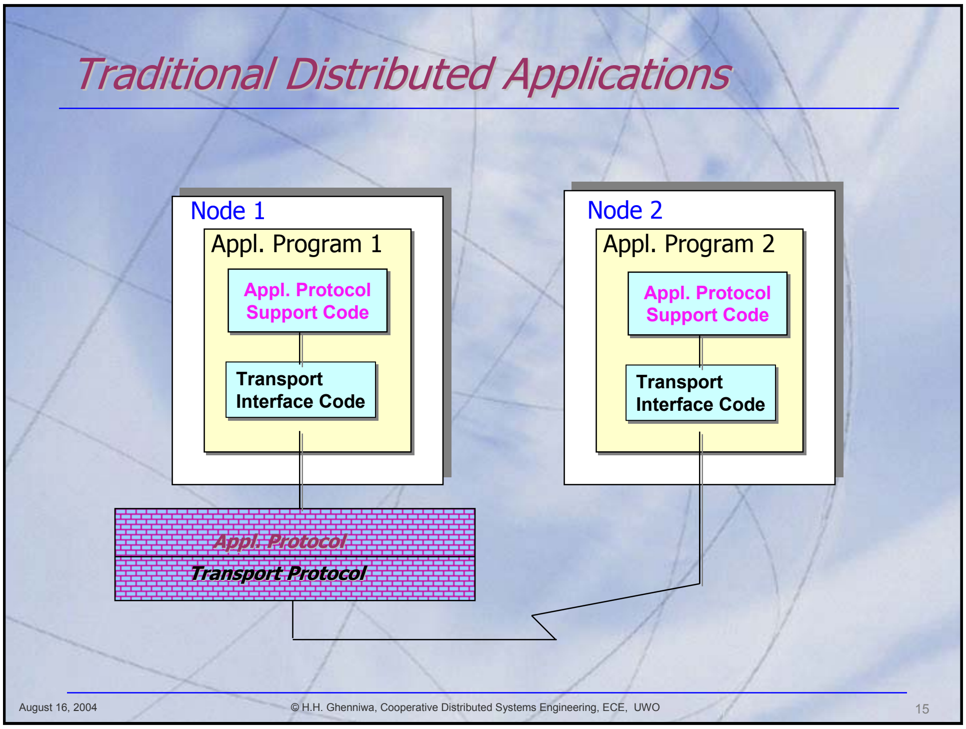
Application

Server layer

Application
Server

Application
Server

Database
Server



Traditional Distributed Applications

Node 1

Appl. Program 1

Appl. Protocol Support Code

Transport Interface Code

Node 2

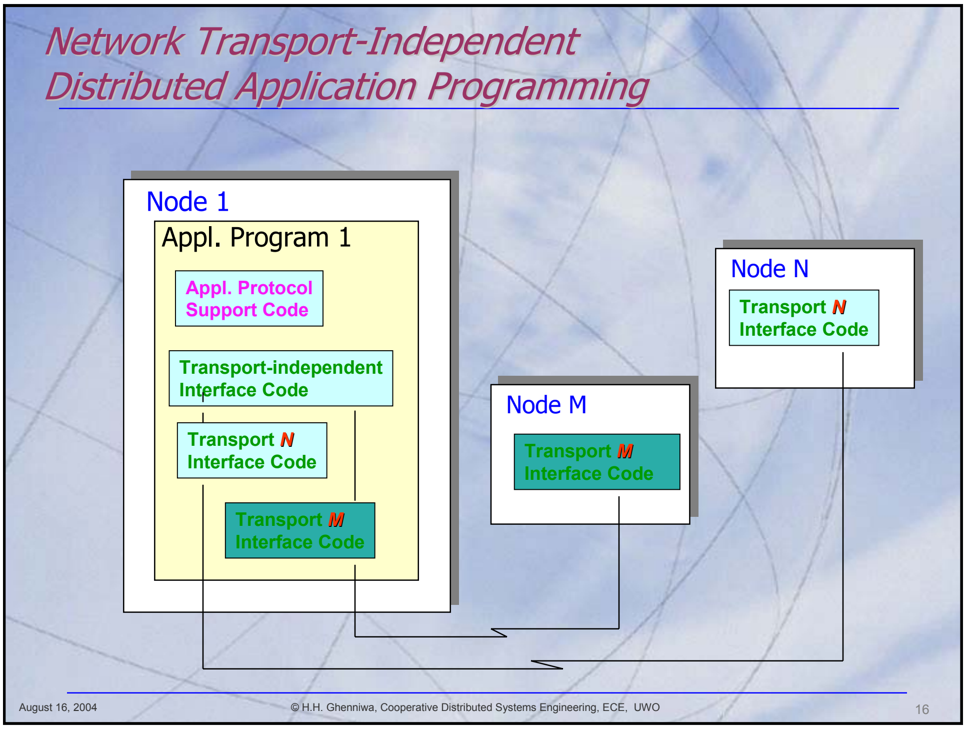
Appl. Program 2

Appl. Protocol Support Code

Transport Interface Code

Appl. Protocol

Transport Protocol



Network Transport-Independent Distributed Application Programming

Node 1

Appl. Program 1

Appl. Protocol Support Code

Transport-independent Interface Code

Transport *N* Interface Code

Transport *M* Interface Code

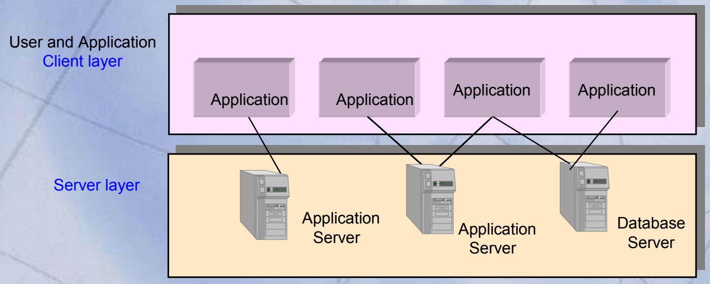
Node M

Transport *M* Interface Code

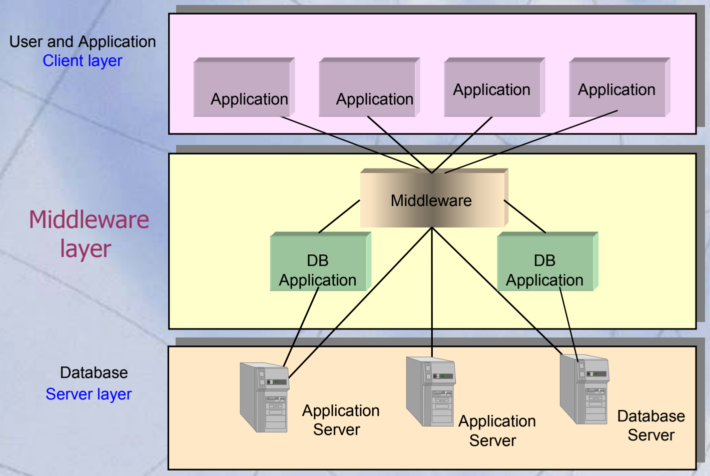
Node N

Transport *N* Interface Code

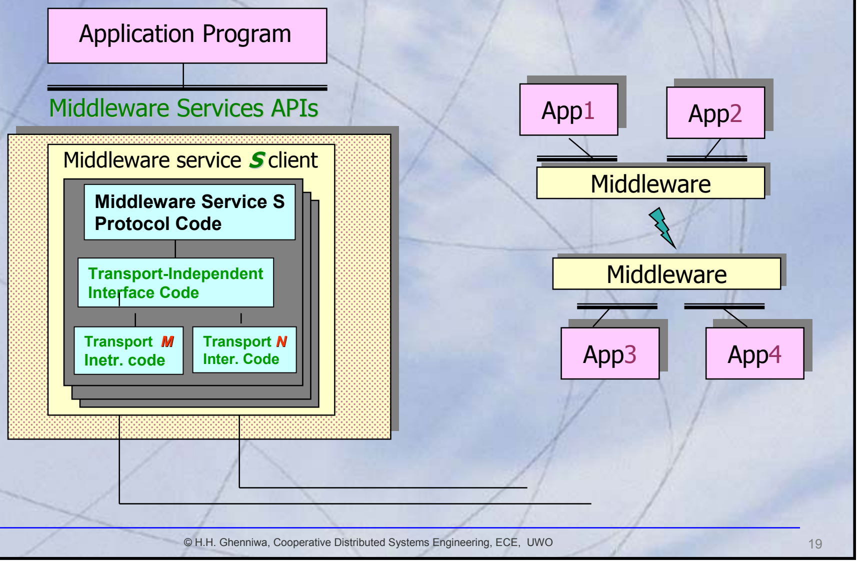
Client Server Architecture



Client Server Architecture



Service-Based Distributed Programming



Application Program

Middleware Services APIs

Middleware service *S* client

Middleware Service S
Protocol Code

Transport-Independent
Interface Code

Transport *M*
Inetr. code

Transport *N*
Inter. Code

App1

App2

Middleware

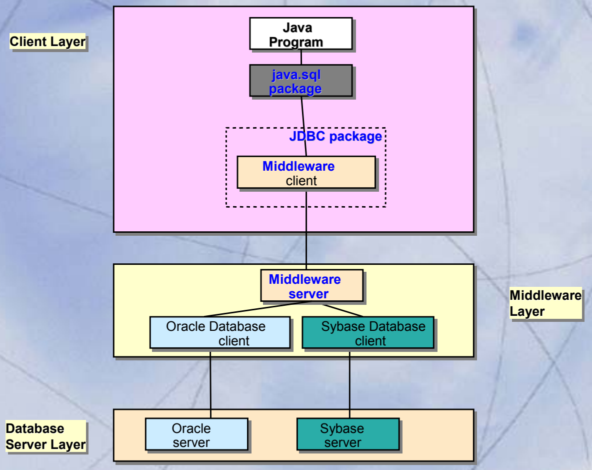
Middleware

App3

App4

Example DB Middleware:

Java Environment



Client Layer

Java
Program

java.sql
package

JDBC package

Middleware
client

Middleware
server

Oracle Database
client

Sybase Database
client

Middleware
Layer

Database
Server Layer

Oracle
server

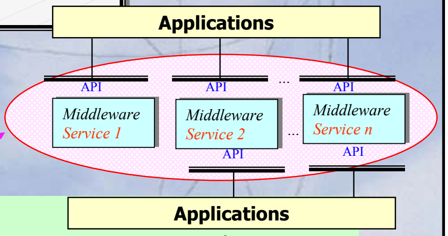
Sybase
server

Technically

Middleware is a set of services and interfaces (APIs)

allow an application to

- locate transparently across the network
- providing interaction with another application or service
- be independent from infrastructure services
- scale up in capacity without losing function



The main purpose of middleware
to help solve many application connectivity and interoperability problems

Types of Services…

- Application enabling services
  - give applications access to distributed services and the underlying network
- Distributed system services
  - e.g., communications, program-to-program, and data management services
- Middleware management services
  - enable applications and system functions to be continuously monitored
    - to ensure optimum performance of the distributed environment

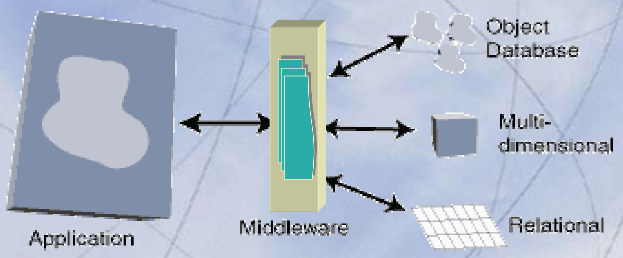
Types of Middleware

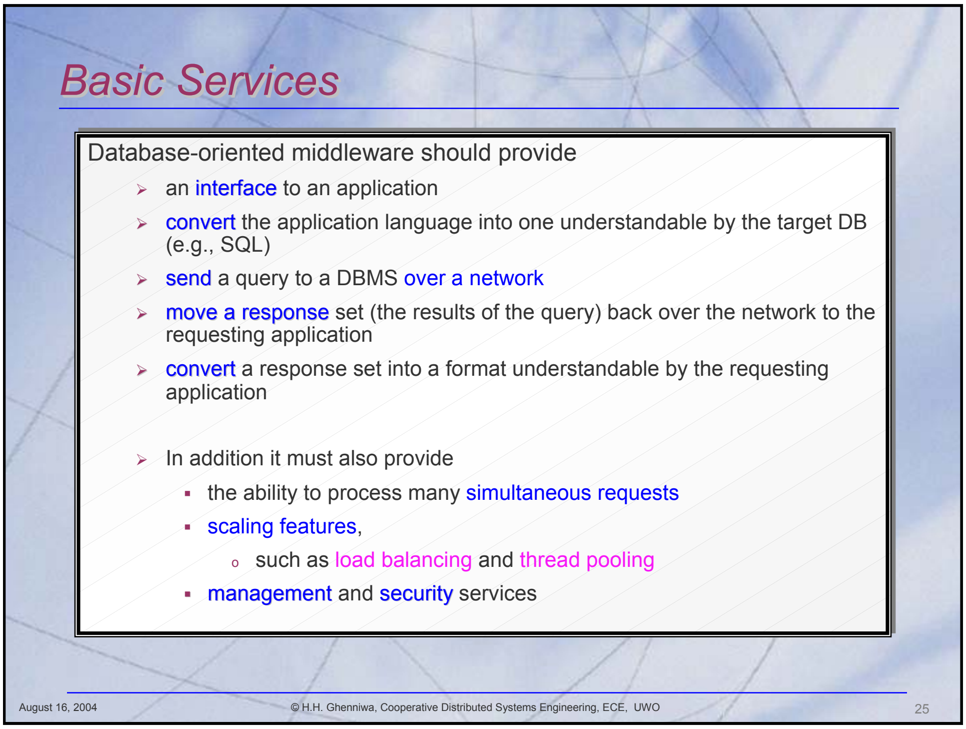
- Middleware can take different forms
  - ✓ Database-Oriented middleware
  - ✓ Virtual Systems
    - ✓ Object request brokers (ORB)
    - ▪ Remote procedure call based (RPC)
  - ➢ Transaction Processing (TP) monitors
  - ➢ Message-Oriented middleware (MOM)
  - ➢ ...

Database-Oriented Middleware

enable applications to access local or remote DBs

regardless of the model employed or the platform upon which they exist

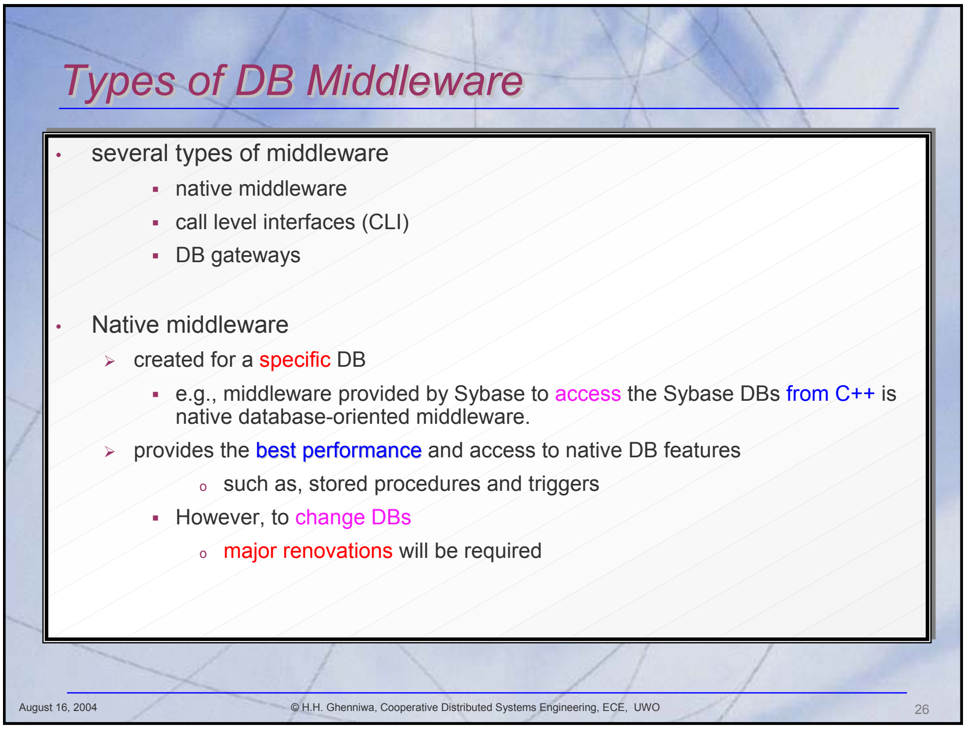




Basic Services

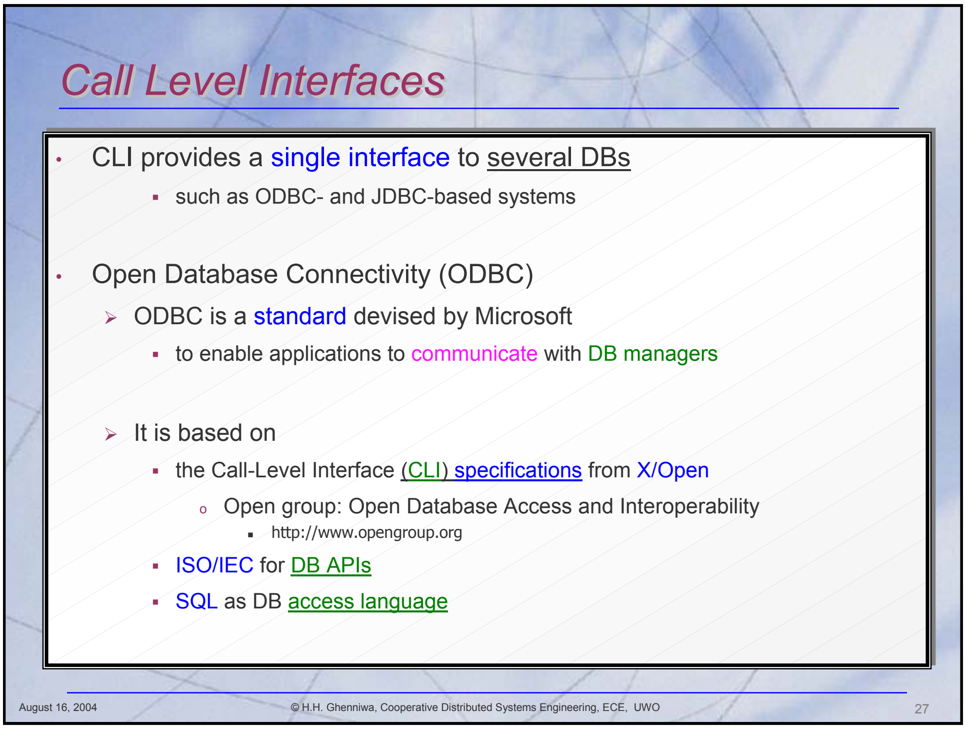
Database-oriented middleware should provide

- an interface to an application
- convert the application language into one understandable by the target DB (e.g., SQL)
- send a query to a DBMS over a network
- move a response set (the results of the query) back over the network to the requesting application
- convert a response set into a format understandable by the requesting application
- In addition it must also provide
  - the ability to process many simultaneous requests
  - scaling features,
    - such as load balancing and thread pooling
  - management and security services



Types of DB Middleware

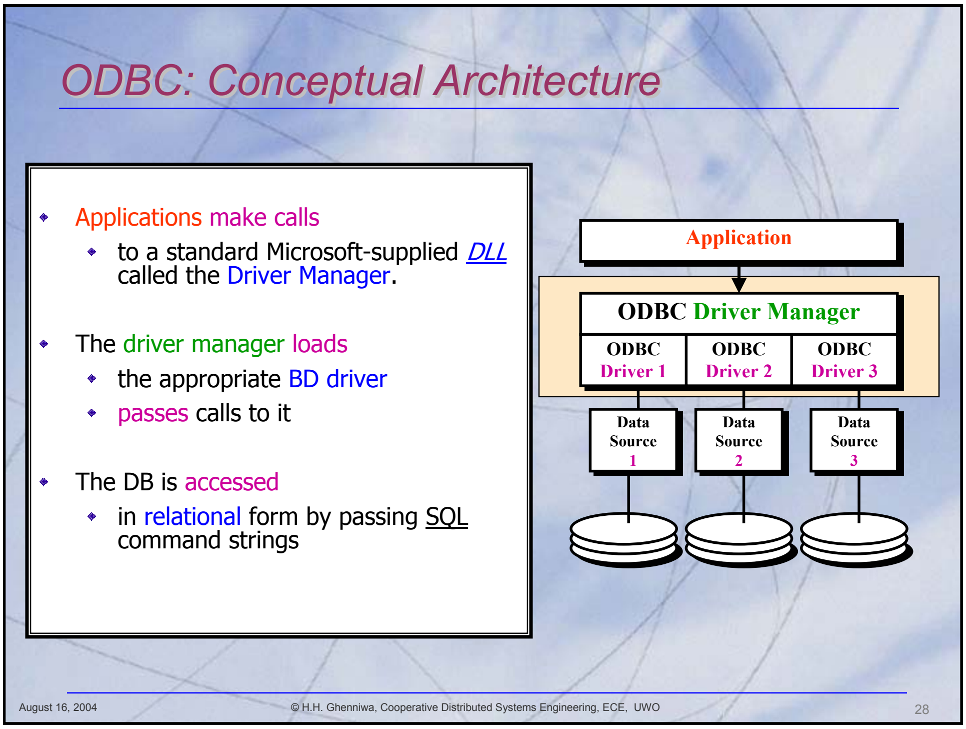
- several types of middleware
  - native middleware
  - call level interfaces (CLI)
  - DB gateways
- Native middleware
  - created for a specific DB
    - e.g., middleware provided by Sybase to access the Sybase DBs from C++ is native database-oriented middleware.
  - provides the best performance and access to native DB features
    - such as, stored procedures and triggers
    - However, to change DBs
      - major renovations will be required



Call Level Interfaces

- CLI provides a single interface to several DBs
    - such as ODBC- and JDBC-based systems

- Open Database Connectivity (ODBC)
    - ODBC is a standard devised by Microsoft
        - to enable applications to communicate with DB managers
    - It is based on
        - the Call-Level Interface (CLI) specifications from X/Open
            - Open group: Open Database Access and Interoperability
                - http://www.opengroup.org
        - ISO/IEC for DB APIs
        - SQL as DB access language



ODBC: Conceptual Architecture

- Applications make calls
    - to a standard Microsoft-supplied *DLL* called the Driver Manager.

- The driver manager loads
    - the appropriate BD driver
    - passes calls to it

- The DB is accessed
    - in relational form by passing SQL command strings

Roles (Services)

Driver Manager

- **Loads drivers on an as-needed basis for applications**
- **Maps a target database name to a specific driver**
- **Processes several ODBC initialization calls**
- **Validates the parameters and sequences of ODBC calls**

Driver

- **Processes the ODBC function calls**
- **Modifies them to the target database format**
- **submits the SQL statements to the target database**
- **receives the results, and presents the results to the application**

Architectural Types of the Driver

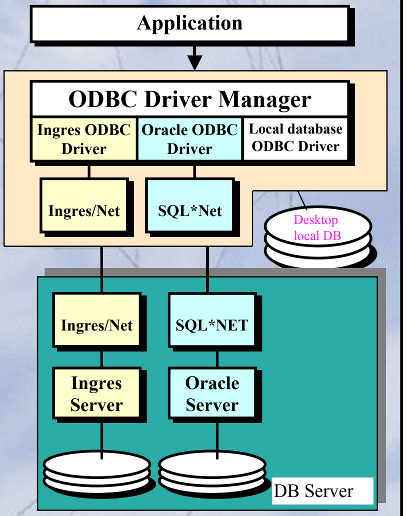
- There are two main driver architectures

  1. single-tier driver
     - manipulates DB files directly
       - typically these drivers are used for native PC DBs
  2. multi-tier drivers
     - generate SQL requests
       - which are passed to the DB server to be processed

Multi-tier Drivers:
Configuration Forms

Configuration 1

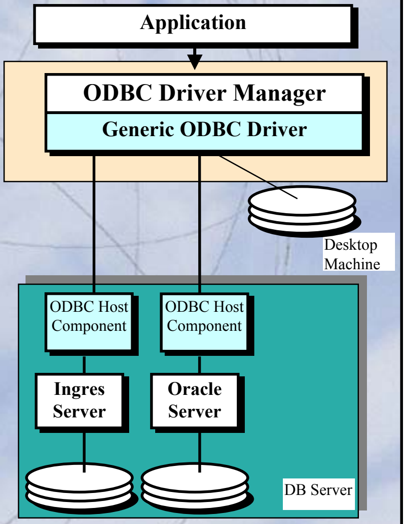
- require DBS vendor's networking software (e.g. Ingres/Net or SQL*Net)
  - The driver is a Windows DLL
    - loaded by the ODBC driver manager
    - then, passes the SQL request to the DB vendors network software
      - which communicates with the network software on the host



Multi-tier Drivers:
Configuration 2

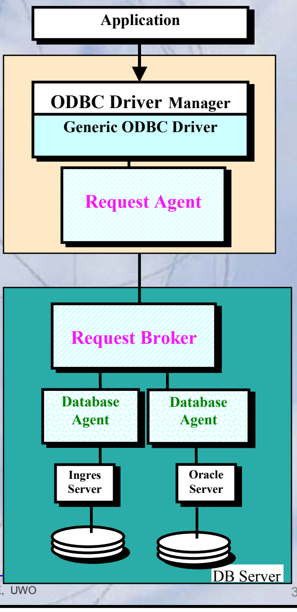
- do not require vendors dependency
- These drivers have components that run on
  - the client machine
  - the DB server
- One particular advantage
  - only one driver is used on the client

    regardless of the number of DB servers used



Multi-tier Drivers:
Configuration 3 --OpenLink

- applications establish a session with the generic ODBC driver
- The request agent on the client then establishes a DB session via the request broker on the host
- The request broker spawns or replicates one or more DB agents
  - associates them with the request agent
  - The DB agents are the only DB specific components
  - They act as clients to the DB server processes of the relevant DB
- The Openlink configuration supports
  - multiple concurrent client applications through a single driver instance



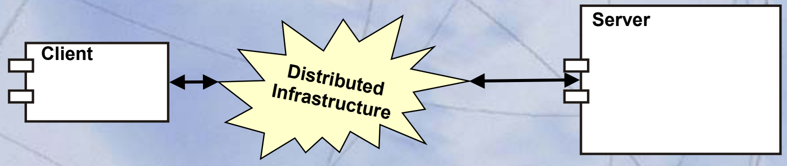
Types of Middleware

- Middleware can take different forms
  - ✓ Database-Oriented middleware
  - ✓ Virtual Systems
    - Object request brokers (ORB)
    - Remote procedure call based (RPC)
  - Transaction Processing (TP) monitors
  - Message-Oriented middleware (MOM)
  - …

Object Request Broker

- ORBs promote interoperability of distributed object systems
  - enable users to build systems
    - by piecing together **objects**
    - that **interoperate** with each other **via ORB**
  - interoperability
    - gives the illusion of locality
      - to make it appear as if the object is local to the client,
        - while it may reside in a different process or machine
    - enables communication of objects across platforms
      - allowing objects to hide their implementation details from clients
        - including programming language, operating system, host hardware, and object location
- developers are only concerned with the object interface details
  - This form of information-hiding enhances system maintainability

Broker Framework

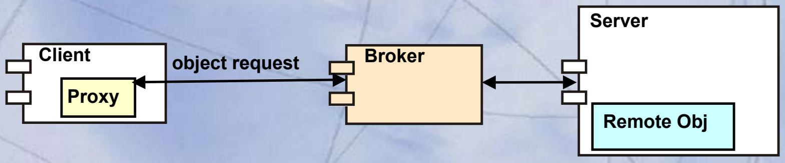


Client

Distributed Infrastructure

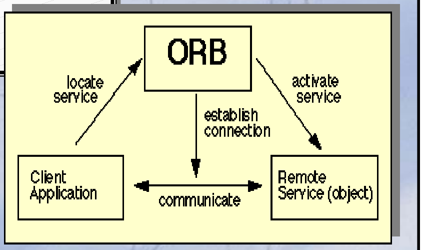
Server

Broker Framework



ORB: Principles

- The relevant functions of an ORB technology are
  - **interface definition**
  - **locating** and possible **activation** of remote objects
  - establish **communication** between clients and "service object"



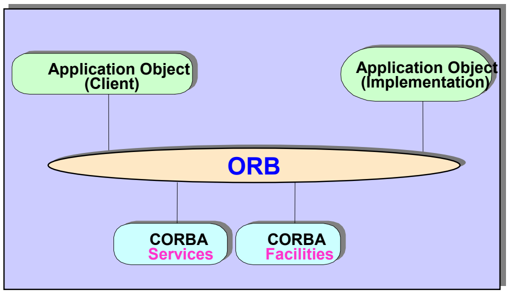
ORB Technology

- There are two major ORB technologies
  - OMG CORBA specification
    - OMG: Object Management Group
    - CORBA: Common Object Request Broker Architecture
  - Microsoft's COM-based (DCOM)
    - COM: Component Object Model

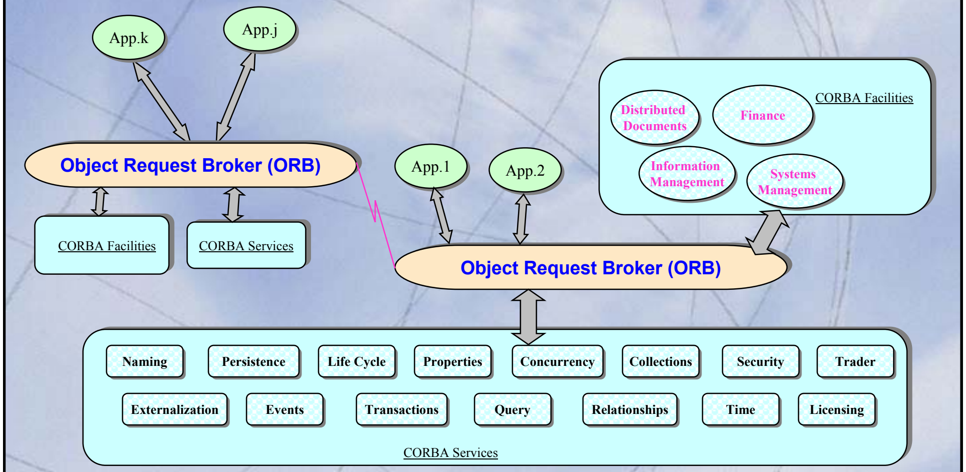
- Additionally, RMI
    - RMI: Remote Method Invocation
  - this is specified as part of the Java language/virtual machine
    - RMI allows Java objects to be executed remotely
      - This provides ORB-like capabilities as a native extension of Java

OMG: Object Management Architecture

- OMG-OMA provides the conceptual infrastructure upon which all OMG specifications are based
- The reference model has the following elements
  - **Application Objects**
  - **ORB**
  - **Object Services**
  - **Common facilities**


Application Object (Client)
Application Object (Implementation)
ORB
CORBA Services
CORBA Facilities

OMG-OMA



App.k

App.j

Object Request Broker (ORB)

CORBA Facilities

CORBA Services

App.1

App.2

CORBA Facilities

Distributed Documents

Finance

Information Management

Systems Management

Object Request Broker (ORB)

Naming

Persistence

Life Cycle

Properties

Concurrency

Collections

Security

Trader

Externalization

Events

Transactions

Query

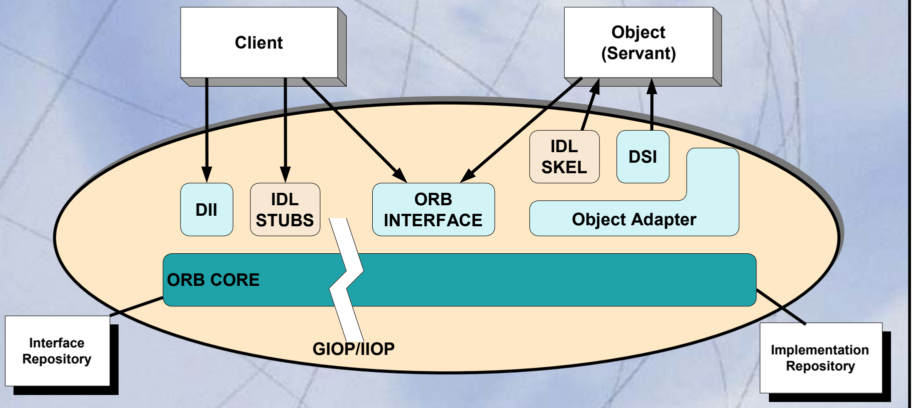
Relationships

Time

Licensing

CORBA Services

OMG-CORBA



Client

Object (Servant)

IDL SKEL

DSI

DII

IDL STUBS

ORB INTERFACE

Object Adapter

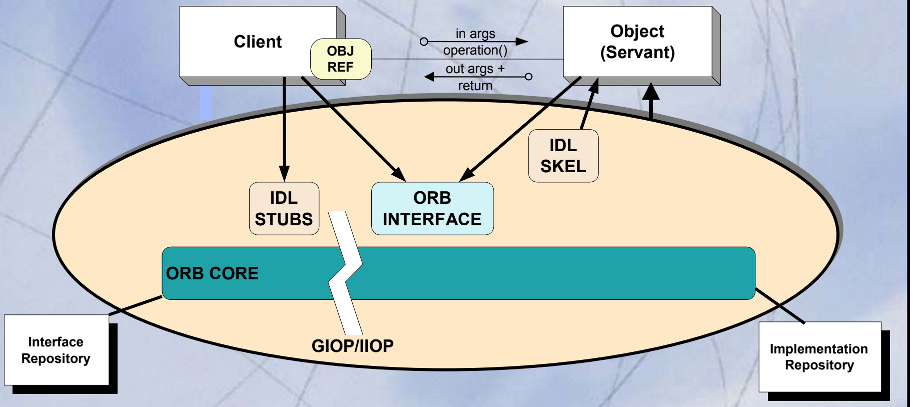
ORB CORE

GIOP/IIOP

Interface Repository

Implementation Repository

OMG-CORBA



OMG Interface Definition Language

IDL Compiler

- IDL is an object-oriented declarative language
  - for **specifying interfaces**
    - It is *not* a programming language
      - implementations are not written in IDL
      - rather, IDL interface descriptions are **mapped** to a programming language for implementing the object
      - Standardized **language mappings**
        - C, C++, Ada95, COBOL, Smalltalk, Java
  - In IDL
    - an **interface** corresponds to a class
    - A **property** is defined as an attribute
      - mapped by the IDL compiler to get and set methods; read-only attribute maps to a get method only
    - An **operation** corresponds to a method
      - parameters must be specified as in, out, or inout

IDL Interface: Example

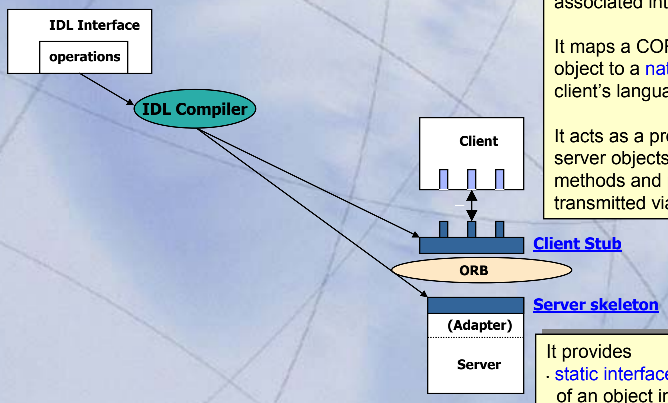
```
// IDL
interface Account
{
        //Attributes
        attribute float balance;
        readonly attribute string owner;

        //Operations
        void makeDeposit(in float amount, out float newBalance);
        void makeWithdrawal(in float amount, out float newBalance);
        };
```

Stubs and Skeletons

- Used in CORBA's static invocation
- PL-specific <u>counterparts</u> to IDL definitions
  - Stubs and skeletons are generated using IDL definitions
- Stubs
  - **create and issue <u>requests</u> on the client side**
    - a. k. a. surrogates or proxies
    - perform marshalling of requests
- Skeletons
  - **receive and forward <u>requests</u> to objects on the server side**
    - perform un-marshalling of requests
    - returns results via the server and client ORBs to the stub

CORBA Process



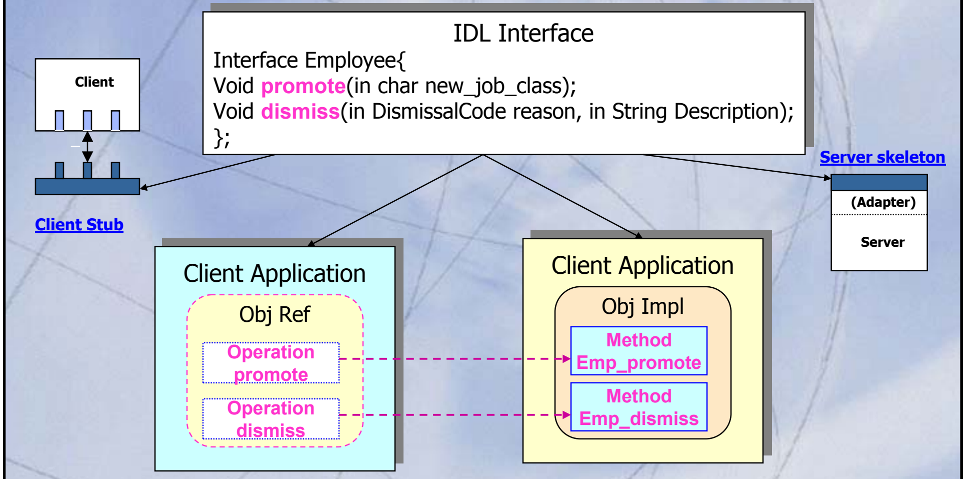
It gets linked to a program wishing to statically invoke a server method through the associated interface

It maps a CORBA server side object to a native object in the client's language.

It acts as a proxy for remote server objects, marshaling methods and parameters to be transmitted via the ORB

It provides

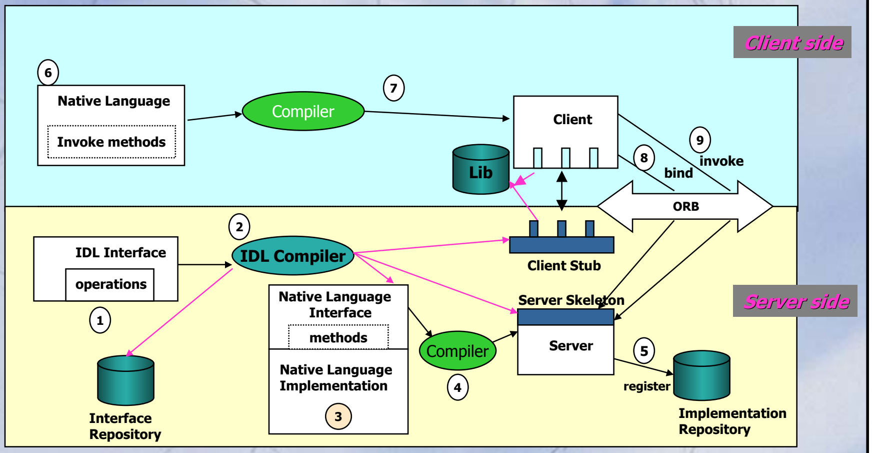
- static interfaces to call methods of an object implementation
- unmarshals methods and parameters that come from the client via the ORB.



IDL Interface

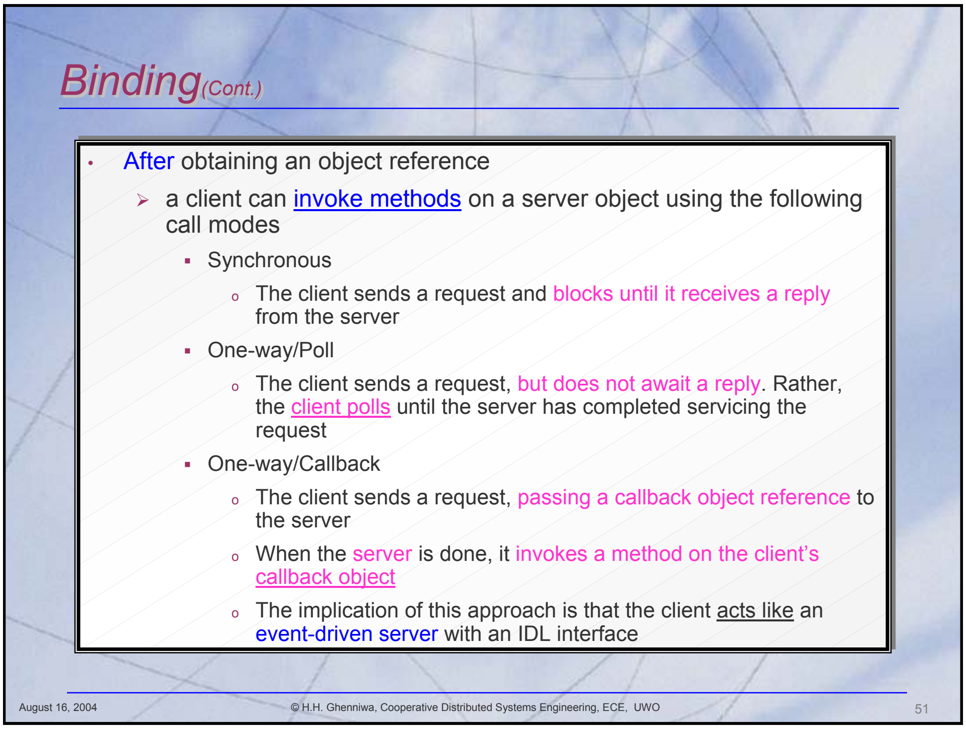
```
Interface Employee{
Void promote(in char new_job_class);
Void dismiss(in DismissalCode reason, in String Description);
};
```

CORBA Process*(Cont.)*



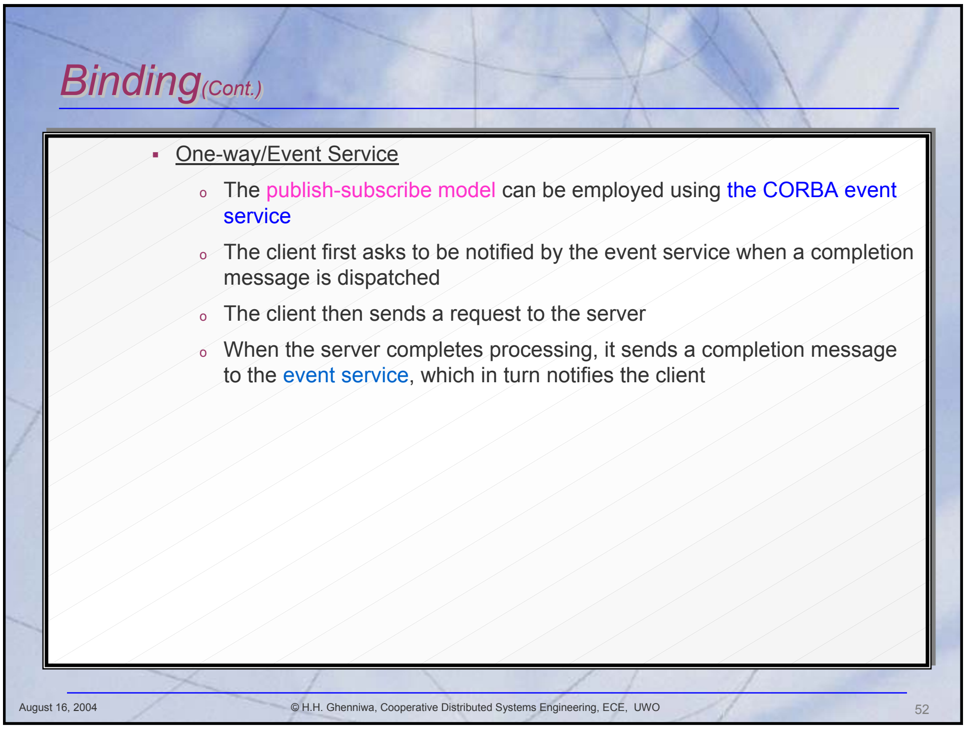
Binding (Integration) in CORBA

- A client can locate a server object by obtaining an object reference directly from a server
  - **by requesting an object by name via the <u>CORBA naming service</u>**
  - **<u>or</u>, by asking the <u>CORBA trader service</u>**
    - to return references to objects that <u>match some criteria</u>
  - **The CORBA standard specifies**
    - a format for an interoperable object reference (IOR) which can be passed across <u>different ORB implementations</u>



Binding(Cont.)

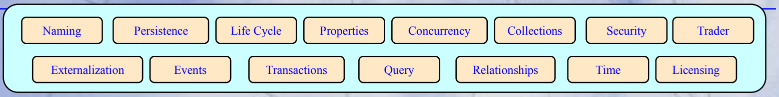
- After obtaining an object reference
  - ➢ a client can invoke methods on a server object using the following call modes
    - Synchronous
      - The client sends a request and blocks until it receives a reply from the server
    - One-way/Poll
      - The client sends a request, but does not await a reply. Rather, the client polls until the server has completed servicing the request
    - One-way/Callback
      - The client sends a request, passing a callback object reference to the server
      - When the server is done, it invokes a method on the client's callback object
      - The implication of this approach is that the client acts like an event-driven server with an IDL interface



Binding(Cont.)

- One-way/Event Service
  - The publish-subscribe model can be employed using the CORBA event service
  - The client first asks to be notified by the event service when a completion message is dispatched
  - The client then sends a request to the server
  - When the server completes processing, it sends a completion message to the event service, which in turn notifies the client

CORBA Services



Naming · Persistence · Life Cycle · Properties · Concurrency · Collections · Security · Trader

Externalization · Events · Transactions · Query · Relationships · Time · Licensing

- **Naming**:
location, uniqueness across contexts, directory interface
- **Persistence**:
storing on object, relational database and OS files
- **Lifecycle**:
creating,copying, moving and deleting components
- **Trader:**
"Yellow Pages" mechanism
- **Concurrency Control**:
lock manger
- **Transaction Service**:
Two phase commit coordination among recoverable components. Allows nested transactions.
- **Relationship Service**:
associations between components, also across naming context

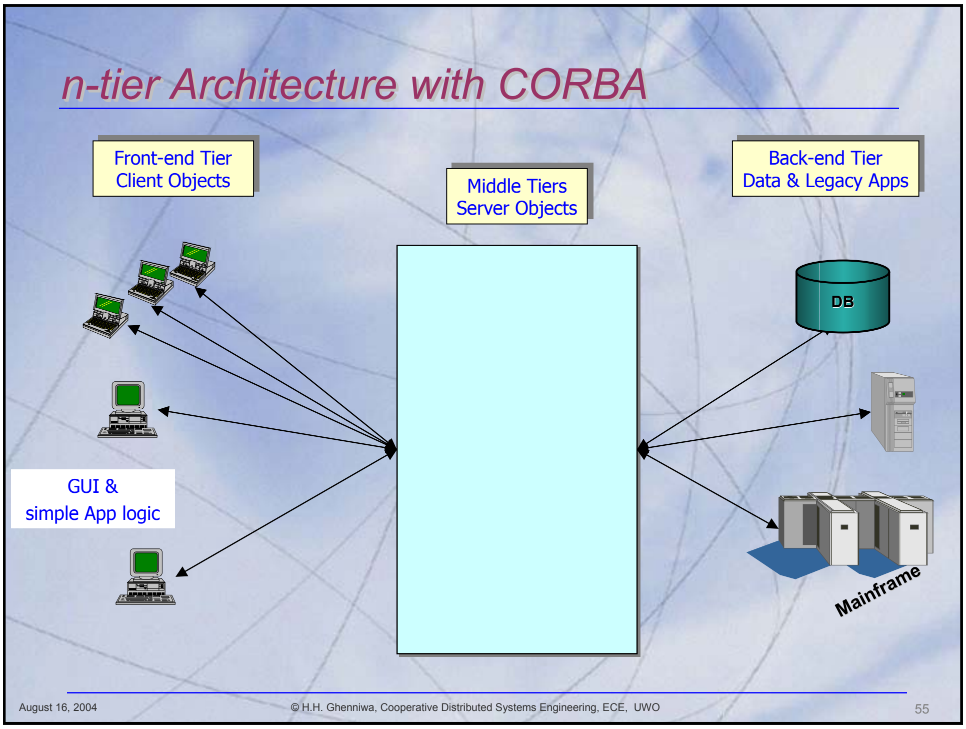
- **Query**:
query for objects. SQL superset. (SQL3 and OQL)
- **Licensing**:
metering use of components for royalty gathering
- **Properties**:
associate named values with a component(e.g. current date)
- **Time**:
synchronizing times. Triggering events
- **Security**:
authentication, access, confidentiality and non-repudiation
- **Externalization**:
streams for import\export of component data
- **Collections**:
basic collection classes

eBusiness Applications

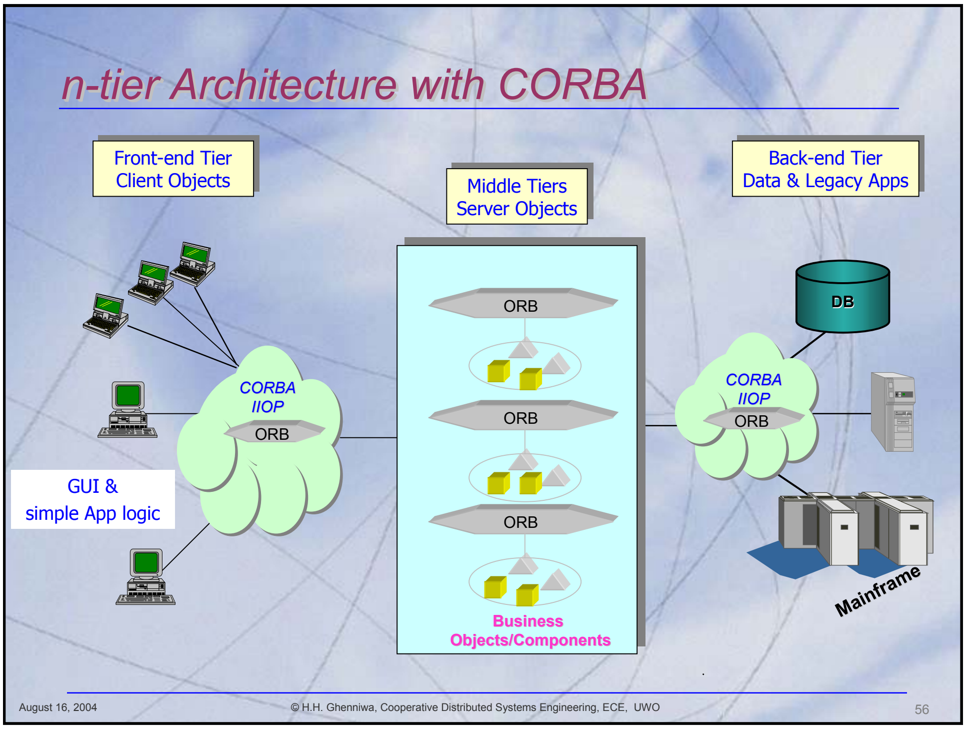
eBusiness applications are typically implemented in n-tier Architecture

- The first tier
  - the **presentation and interaction layer**; for example, a web browser.
- The middle tier
  - the **application logic**, which can be constructed from multiple components, such as **web** and **application servers**
  - Middle tier logic can be partitioned and dynamically distributed
    - thereby, moving processing closer to data sources and supporting load balancing
- The back tier
  - data repositories, such as relational or object-oriented databases

Objects can be new application components or legacy applications

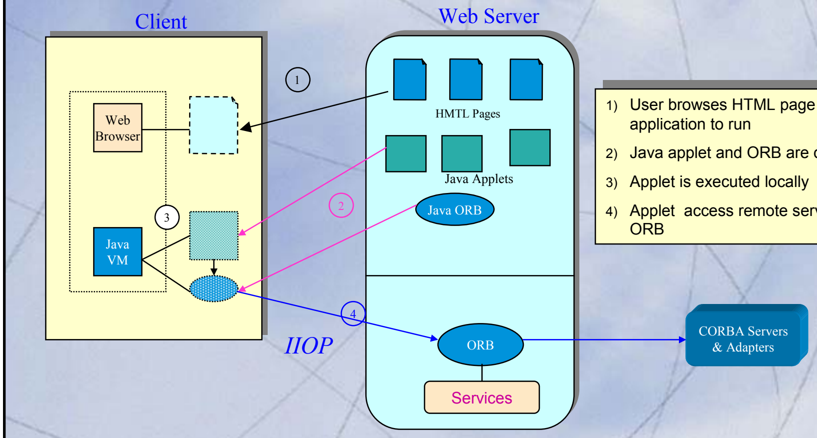


n-tier Architecture with CORBA



n-tier Architecture with CORBA

eBusiness Applications

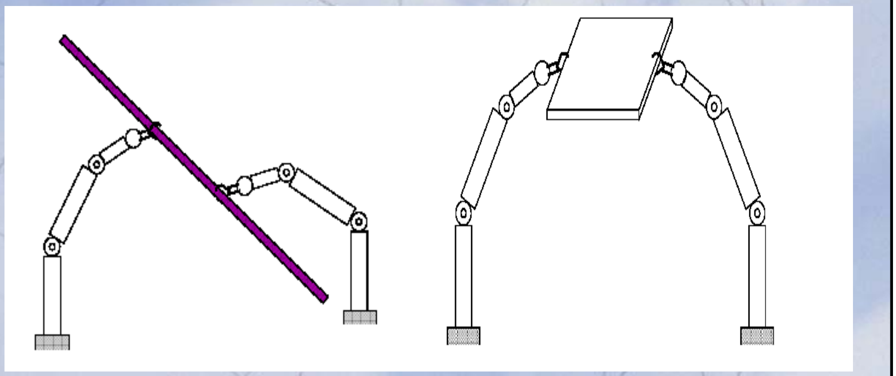


1) User browses HTML page and selects application to run
2) Java applet and ORB are downloaded
3) Applet is executed locally
4) Applet access remote servers through ORB

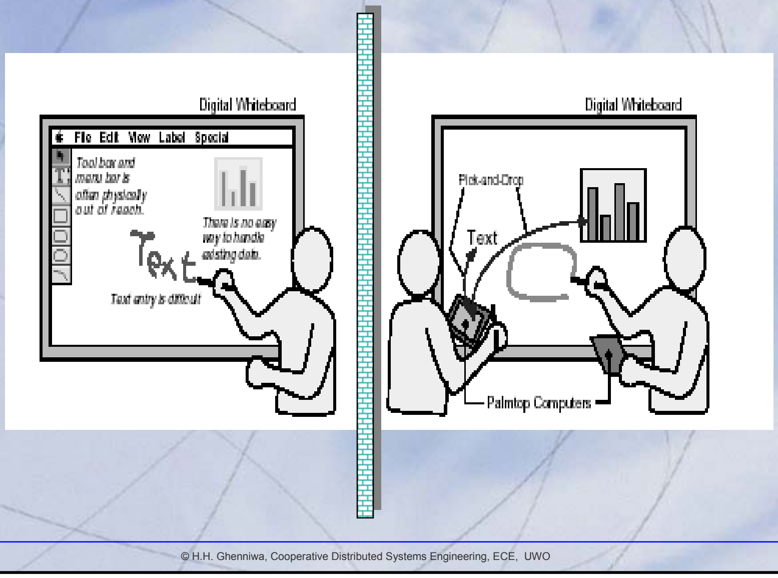
At Square One…

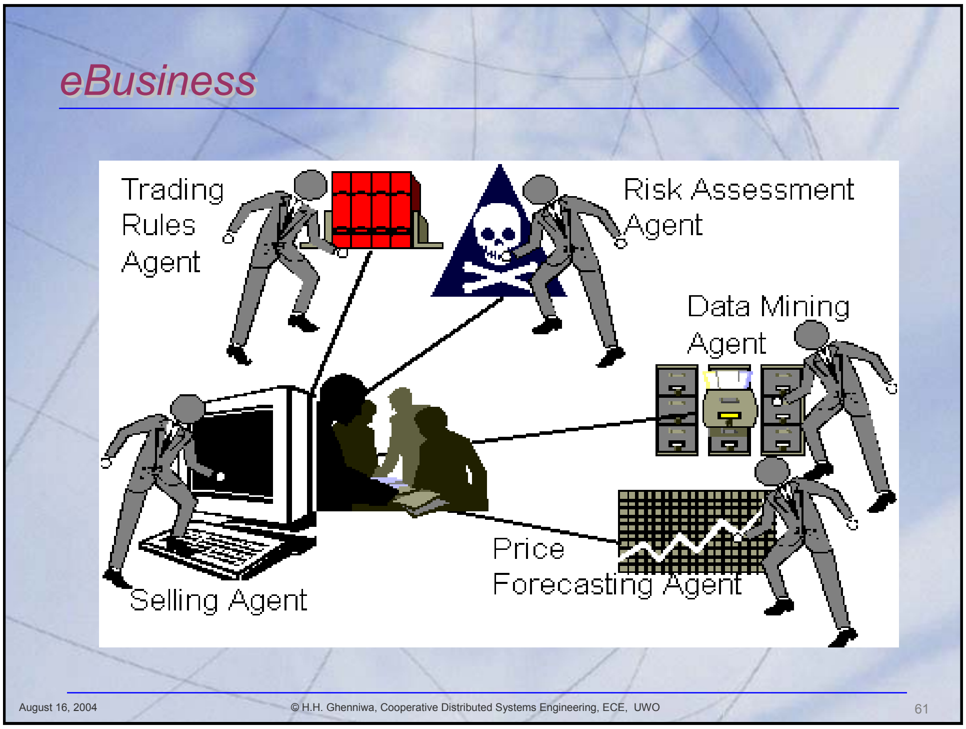
- Understand the evolving needs!
- The new technology expectations!
- Identify design issues

Multi-Robot Arm



Collaborative Design…


Digital Whiteboard
File Edit View Label Special
Tool box and menu bar is often physically out of reach.
Text
There is no easy way to handle existing data.
Text entry is difficult
Digital Whiteboard
Pick-and-Drop
Text
Palmtop Computers



eBusiness

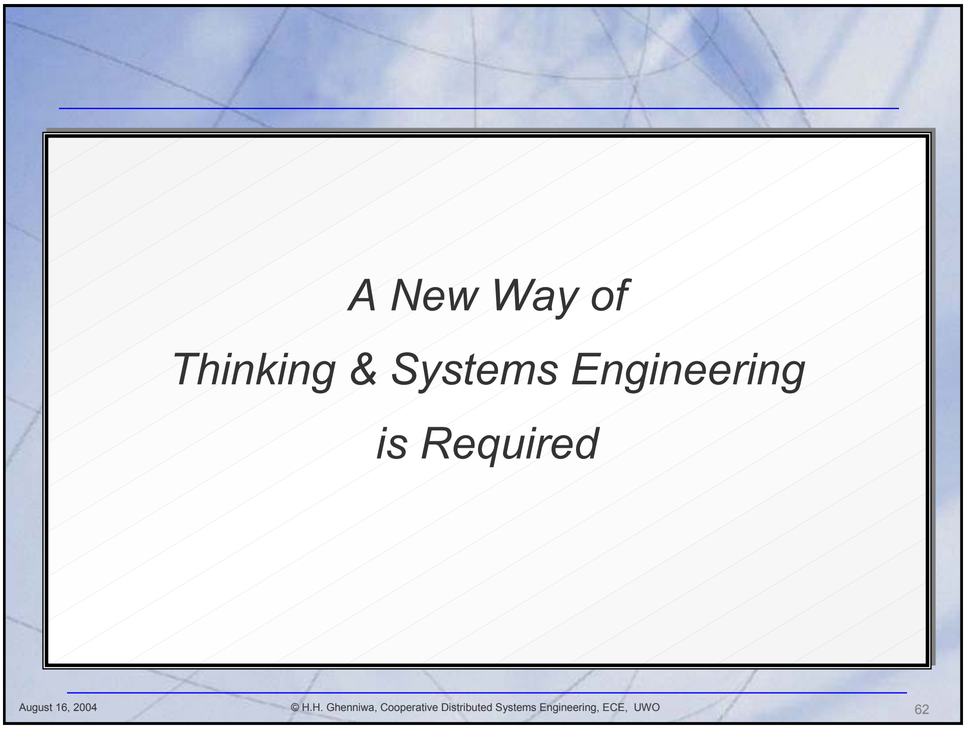
Trading Rules Agent

Risk Assessment Agent

Data Mining Agent

Price Forecasting Agent

Selling Agent



A New Way of

Thinking & Systems Engineering

is Required

Vision…

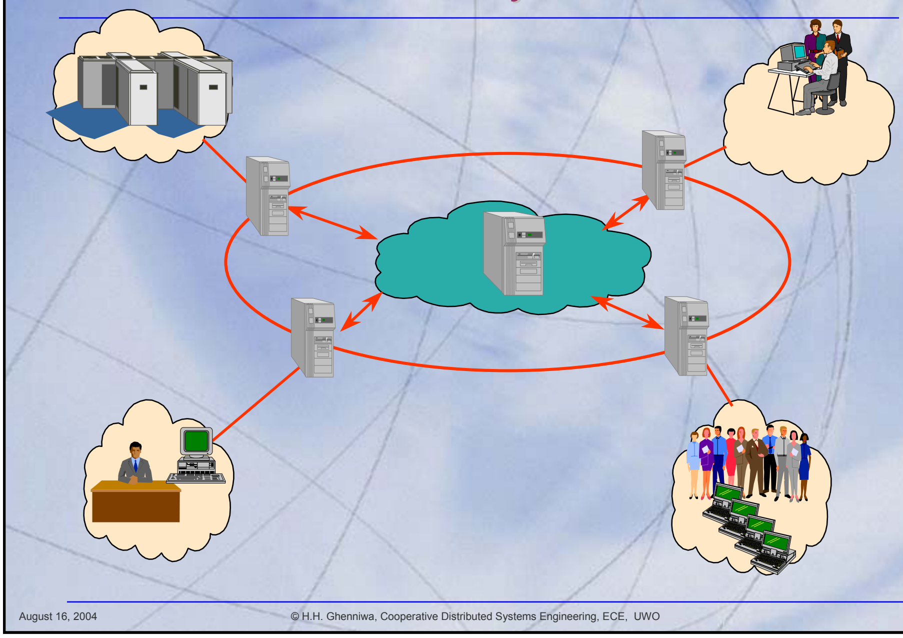
- the cyberspace is a large society of

  software artifacts, information sources, and devices that

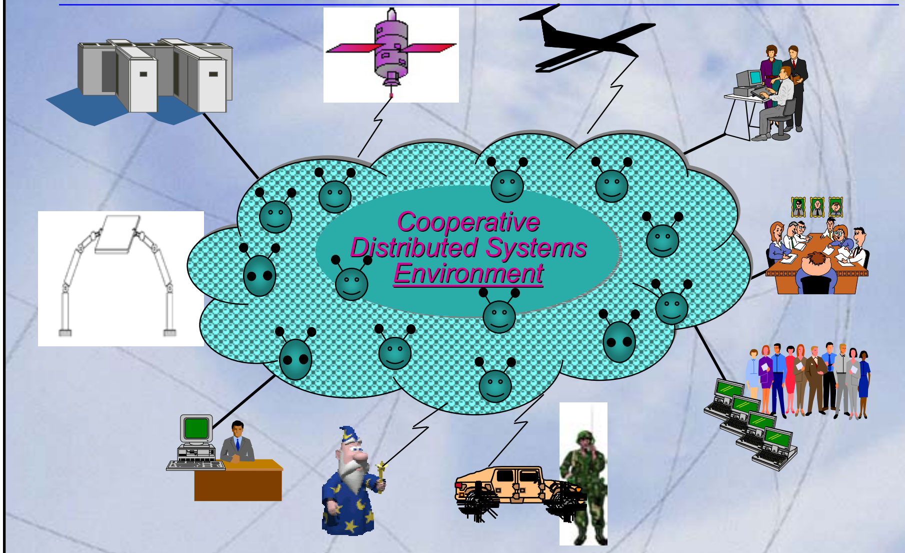
  - ubiquitously interact with each others
  - self-organize at run-time
  - with increasing degree of autonomy and responsibility over time

  yet, scaleable, reliable, secure, ...

Classical Distributed Systems

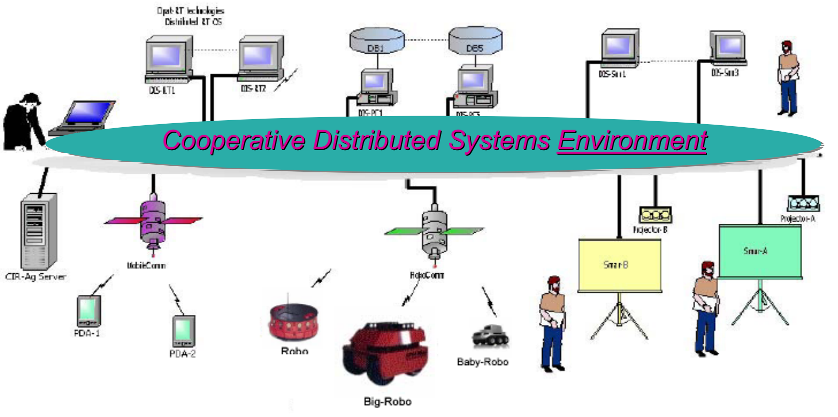


Cooperative Distributed Systems



CDS-EnG Facilities

Distributed Intelligent Systems
(DIS) Lab
InfraStructure



Cooperative Distributed Systems (CDS)

- CDS is a <u>class of systems</u> with entities that are able to
  - perform some functions independently and
  - exercise some degree of authority in
  - sharing their capabilities

CDS: Design Concepts ++

- Abstraction
- Refinement
- Modularity
- Information Hiding
- Interface
- Functional Independence
- Architecture
- Reusability

- Autonomy
- Coordination
  - Communication
  - Interaction
- Cooperation
- Adaptability

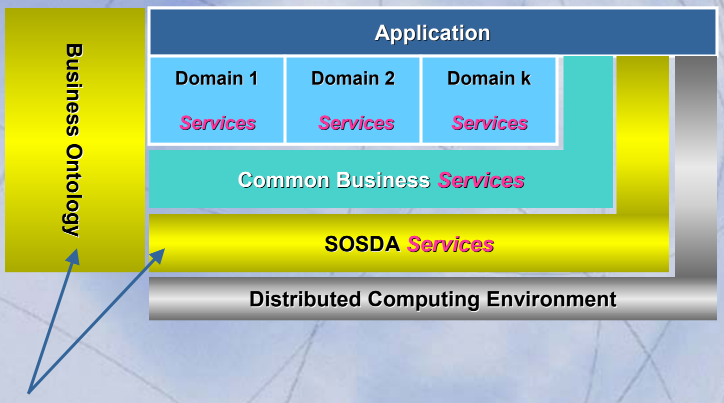
Service-Oriented Semantic-Driven architecture (SOSDA)

SOSDA is an integration architecture for cooperative distributed systems.

The SOSDA specifications and framework provide the abstraction to support the domain entities and applications independent of any specific technology. A key to SOSDA is a **service-oriented model** for CDS environment, in which the overall connectivity of the system supports a “virtual” point-to-point integration mechanism. The main elements of SOSDA include domain services, integration services, and domain ontology.

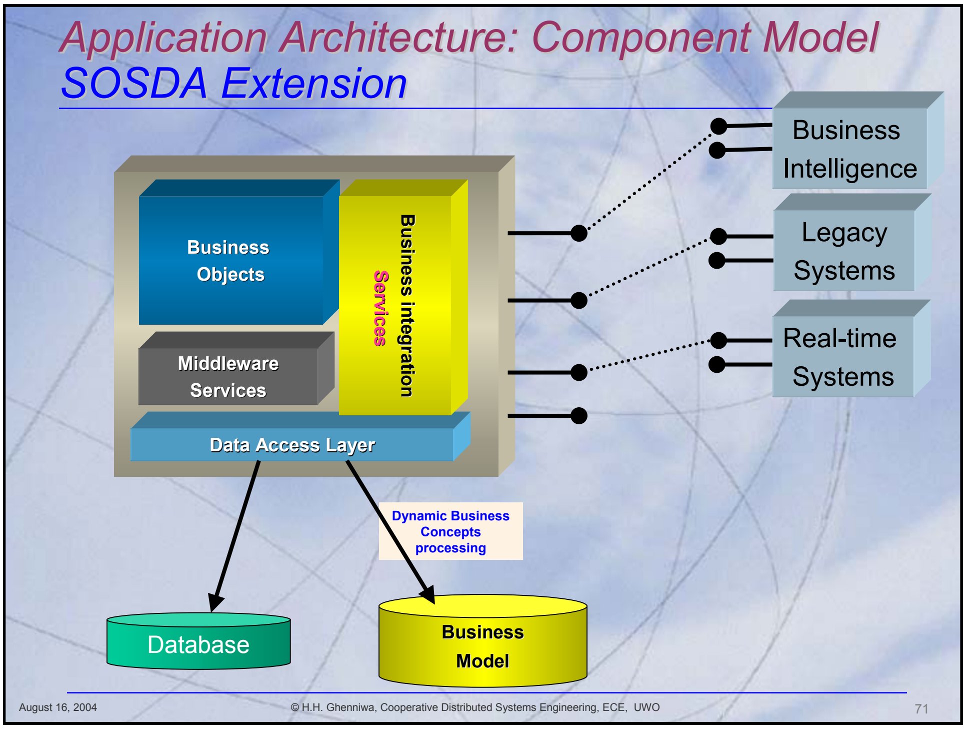
SOSDA provides three families of **integration services**. **Ontology and semantic integration** services support the semantic manipulations needed when integrating and transforming information or knowledge to satisfy a SOSDA task. **Coordination and cooperation** services support ad hoc and automated SOSDA configurations. This includes locating and discovering relevant applications and services. **Wrapping** services make different applications or modules comply with internal or external standards.

Foundation Architecture: *SOSDA –Service Oriented*

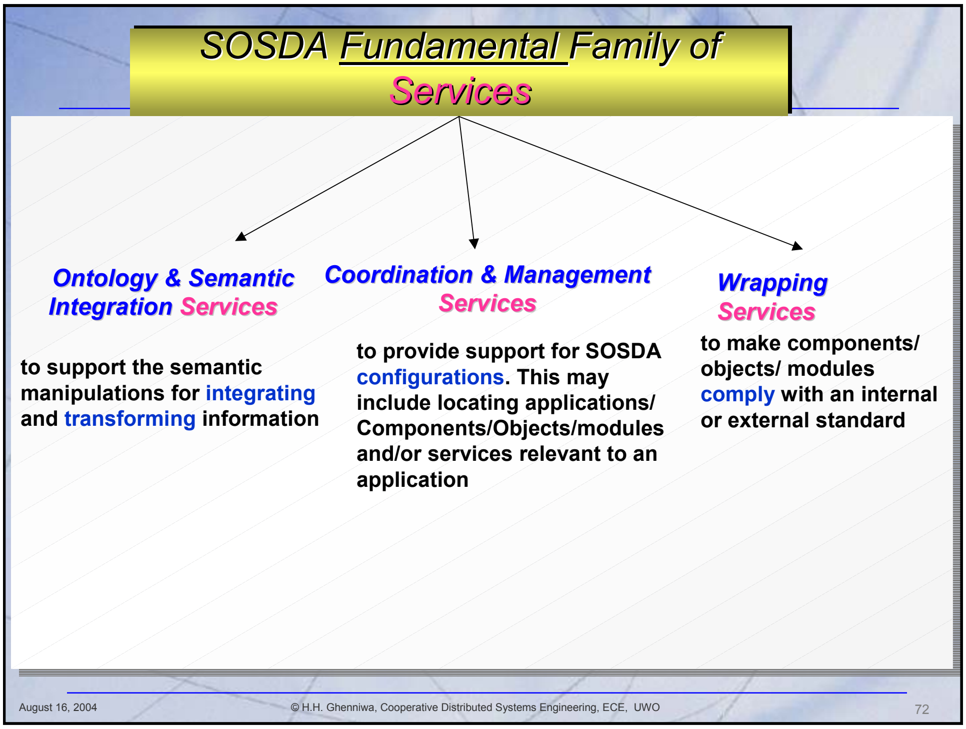


Business Integration

Business model (ontology) is a tool that captures the conceptualization of a business domain at the knowledge level



Application Architecture: Component Model
SOSDA Extension



SOSDA Fundamental Family of Services

Ontology & Semantic Integration Services

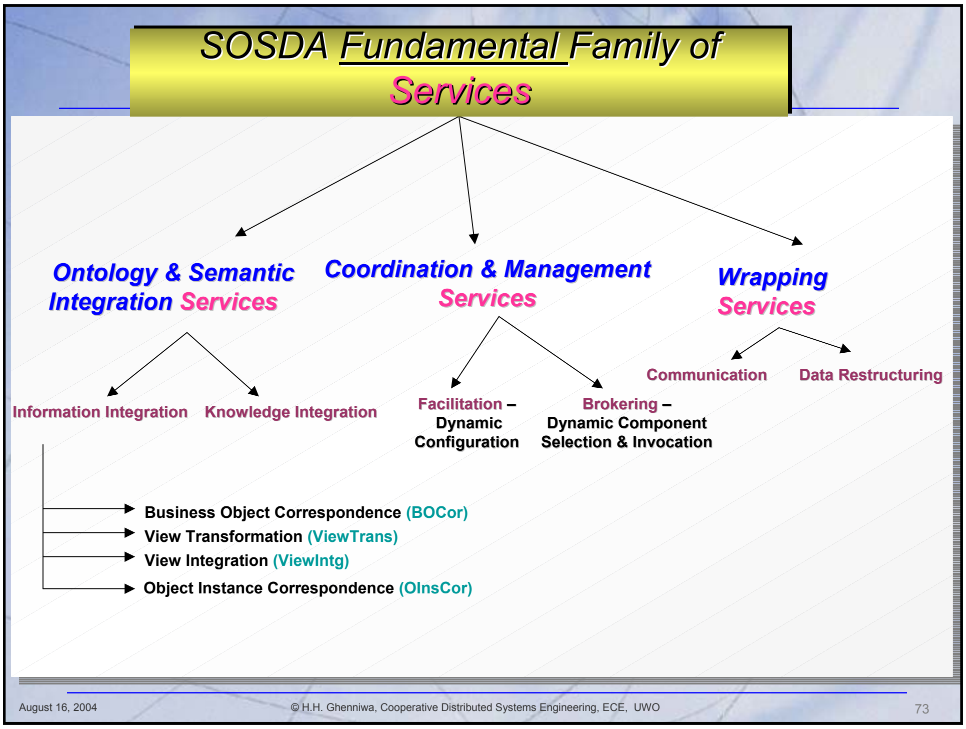
to support the semantic manipulations for integrating and transforming information

Coordination & Management Services

to provide support for SOSDA configurations. This may include locating applications/ Components/Objects/modules and/or services relevant to an application

Wrapping Services

to make components/ objects/ modules comply with an internal or external standard



SOSDA Fundamental Family of Services

Ontology & Semantic Integration Services

Coordination & Management Services

Wrapping Services

Information Integration

Knowledge Integration

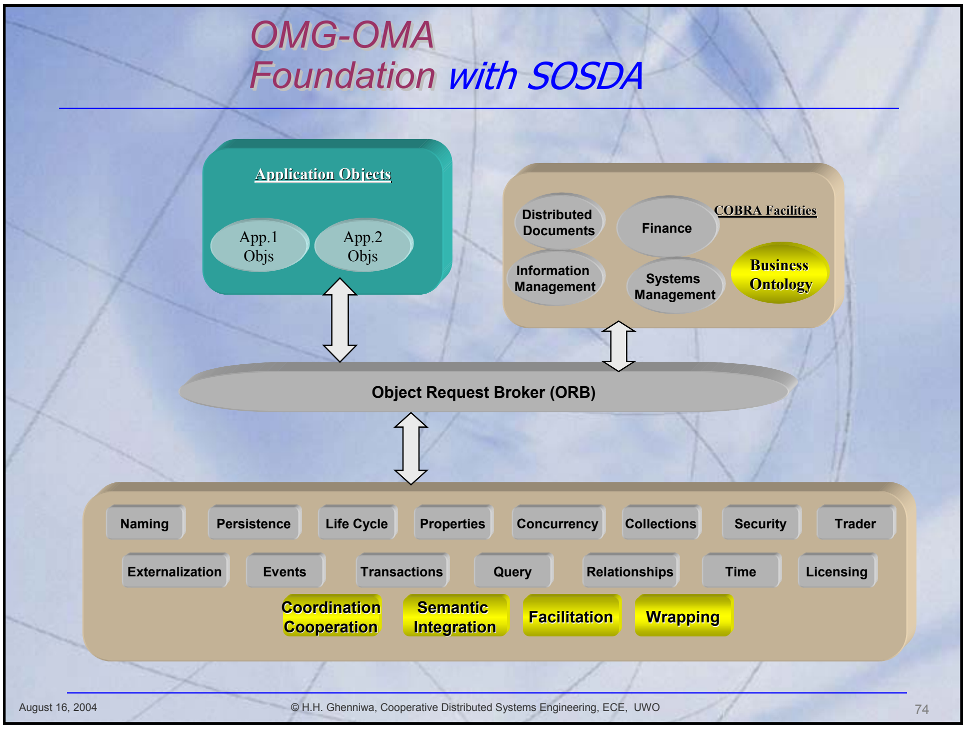
Facilitation – Dynamic Configuration

Brokering – Dynamic Component Selection & Invocation

Communication

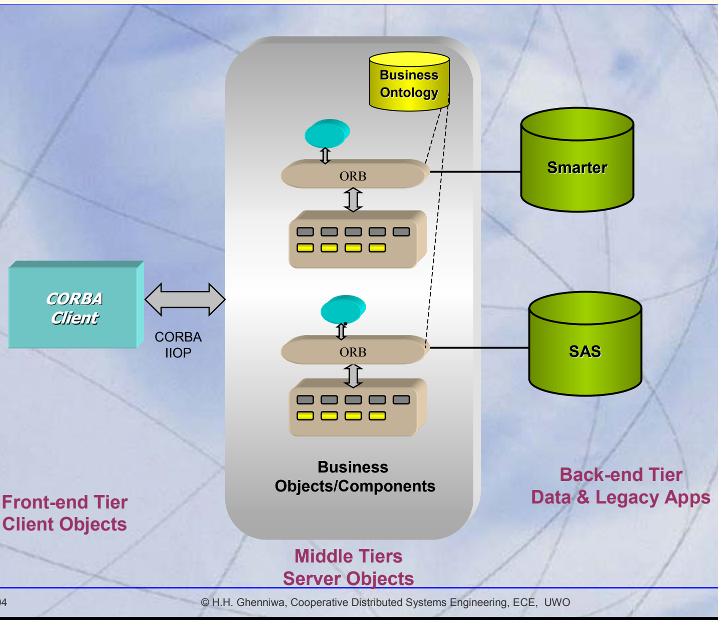
Data Restructuring

- Business Object Correspondence (BOCor)
- View Transformation (ViewTrans)
- View Integration (ViewIntg)
- Object Instance Correspondence (OInsCor)

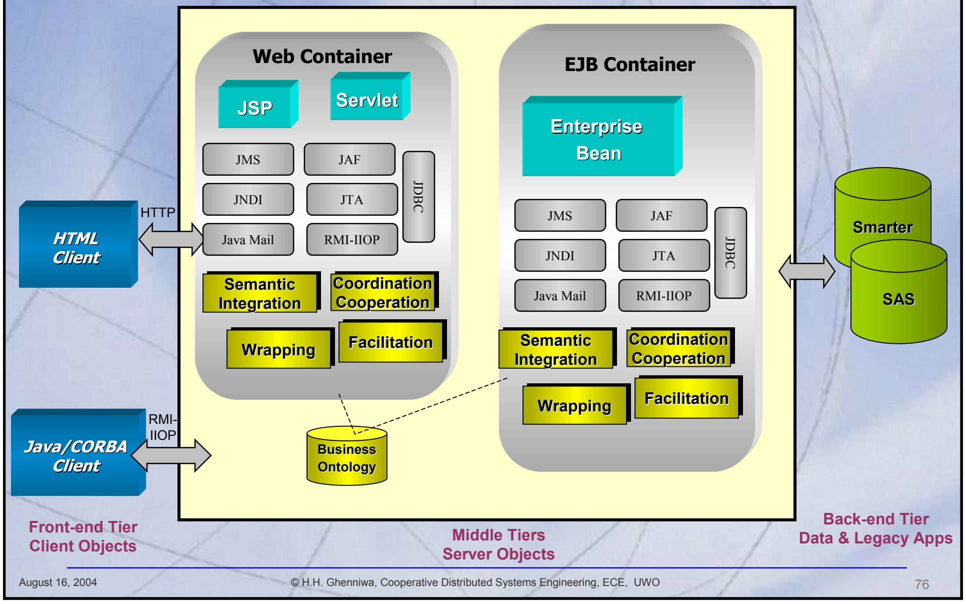


OMG-OMA Foundation with SOSDA

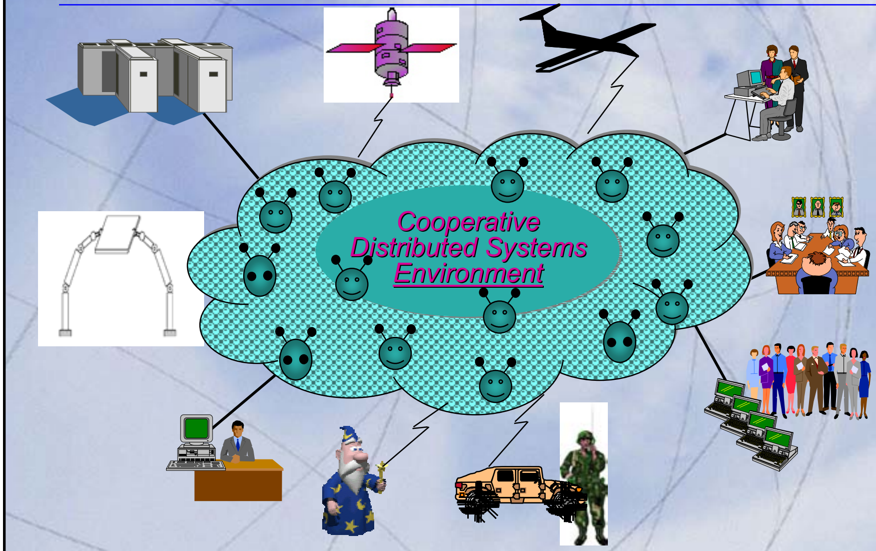
N-Tier Application Architecture with CORBA-Based SOSDA



N-Tier Application Architecture with EJB-Based SOSDA



Cooperative Distributed Systems


Cooperative
Distributed Systems
Environment