

# Distributed Software Development Integration and Interoperation

Chris Brooks

Department of Computer Science  
University of San Francisco

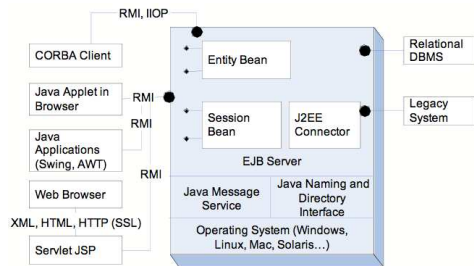
## 12-2: Integration and Interoperation

- A primary argument for service-oriented computing is to ease the burden of getting heterogeneous components to play nicely together.
- Integration is used to refer to pulling together several resources into a single logical resource.
- Interoperation is used to refer to a layer that allows existing components to work together.
- In practice, the concepts are often blurred.

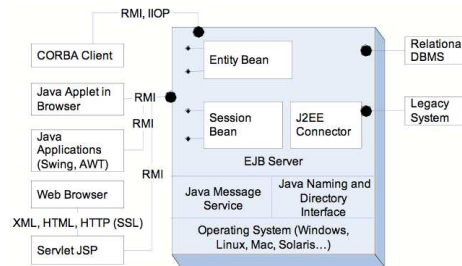
## 12-3: Enterprise Integration

- Enterprise Integration refers to the problems surrounding heterogeneous environments and data representations.
- Enterprise Application Integration refers to the problems involved in allowing heterogeneous environments and applications to communicate.
  - J2EE, .NET
- Enterprise Information Integration refers to the problem of resolving inconsistencies and semantic differences between data sources.

## 12-4: Enterprise Architectures - J2EE



## 12-5: Enterprise Architectures - .NET



## 12-6: Integrating Legacy Systems

- A challenge to integration is the presence of *legacy systems*.
- A pejorative term for computing systems that
  - run on obsolete hardware and nonstandard communication networks
  - run poorly documented, unmaintainable software
  - consist of poorly modeled databases on hierarchical or network DBMSs
  - support rigid user interfaces

### 12-7: How Legacy Systems Arise

- Proprietary software
  - not documented
  - not supporting industry standards (vendors who hope to lock in the market through incompatibility)
  - No standards developed.
- Semantics embedded procedurally in the code
- Ad hoc changes to software in response to
  - changing requirements, because of changes in laws, regulations, competition, or other business needs
  - bugs

### 12-8: Legacy Systems: Negative

- Difficulties in reuse and sharing of data and programs cause redundancy, wasted effort, and integrity violations
- Closed: typically, use a vendor's proprietary software, and cannot cooperate with other systems

### 12-9: Legacy Systems: Positive

- Fulfill crucial business functions
- Work, albeit suboptimally
- Run the world's airline reservation systems
- Run most air traffic control programs
- Have dedicated users
- Perform critical functions
- Represent huge investments in time and money

### 12-10: Legacy Systems - example

- USF's Data Management systems are all legacy systems
  - Development
  - Financial Aid/Bursar
  - Registrar
  - Payroll/HR
- Out-of-date, proprietary systems (they run on VMS!)
- Difficult to maintain
- Poor user interface.
- Why are they still in use?

### 12-11: Legacy Systems - example

- Why are they still in use?
- Continual usage. Many of these systems cannot be unavailable for extended periods.
  - Periods of heavy usage may not align
- Data transfer is a big, complex job
- Money
  - Expense devoted to current system
  - New systems are very expensive.
- Institutional inertia
  - New systems require retraining, change in a large number of places.
- USF is currently transitioning away from its legacy systems.

### 12-12: Accommodating Legacy Systems

- Introduce new technology as needed
- Integrate legacy systems with new components
- Integrate the legacy systems with each other
- But don't spoil existing applications
  - Is this even possible?
  - If not, why not?
  - If so, how might one achieve this?

### 12-13: Accomodating Legacy Systems

- Consider the effort per legacy system one is willing to invest in
  - modifying existing applications
  - Acquiring knowledge about, i.e., models of, the existing applications
- The limits on the ranges of the new applications
- Whether improvements to legacy applications are sought

### 12-14: Migration

- Updating technology is
  - Essential
  - A continual process
- All at once?
  - Expensive
  - Risky
  - Brittle
  - Frustrating for users
- Gradual change: dismantle legacy and build desired system hand-in-hand
  - Install and test piecemeal
  - Need to ensure that compositional functionality remains correct.

### 12-15: Migrating - Converters

- A converter is a wrapper that translates data or interfaces between applications.
- An old-to-new converter maps legacy data or interfaces into a more modern representation.
- Example: hierarchical to relational converters, which generate SQL from hierarchical databases
- A new-to-old converter maps modern representations into legacy versions.
- Example: Mapping SQL inserts into the syntax needed to add records into a hierarchical DB.

### 12-16: Using Converters for Interoperation

- Converters work well where there are only a small number of applications
- Converters
  - can be applied, but expensively
  - need a converter between every pair of applications, user interfaces, and database systems
- Dream: develop tools that can act as generic converters.
- Singh and Huhns argue that services can potentially help with this.

### 12-17: Use cases for SOC

- Use cases help to identify:
  - Situations in which a service-oriented approach can be useful
  - Potential roadblocks and issues to consider
  - Data that must be exchanged
  - Dependencies among workflows and processes.

### 12-18: Interoperation within an Enterprise

- One use case is the interoperation of applications within an enterprise.
  - Make different software components work together.
- Need:
  - Connectivity between components
  - Ability to understand each other's communications
  - Not just syntax, but semantics and pragmatics.
- Since all components are within the same enterprise, authentication and trust is not as great an issue.

### 12-19: Interoperation between enterprises

- Communication between enterprises is an increasingly important use case.
  - Supply chain configuration, dynamic reconfiguration of manufacturing, value-added services.
- Previously, enterprises either used ad hoc methods, or rigid standards such as EDI.
- Challenges: balancing standardization against flexibility, providing autonomy to parts of an enterprise.

### 12-20: Dynamic software configuration

- Singh and Huhns provide a number of cases where services can be used to allow components or systems to dynamically interoperate.
  - Introduction of new applications
  - Dynamic selection of vendors/customers/partners
  - Grid computing
  - Utility computing
- One of the big advantages of the service-oriented approach is the ability to add, remove and replace components.

### 12-21: Service Composition

- Single services can clearly be very useful, but the real value in a service-oriented approach is the ability to compose simple services to get more complex functionality.
  - Example: constructing a travel package.
- Composition can allow a vendor to add value to an offering, and thereby avoid competing solely on price.
- Useful in:
  - Portals/information aggregation
  - Electronic Commerce
  - Virtual enterprises, supply chain management

### 12-22: Service Composition

- Simple service composition has been around for a while
  - Airline/car rental, metasearch engines
- More complex composition is still primarily a research topic.
- Challenges:
  - Need for maturity in lower-level system
  - Need for standardized representation and protocol for describing services.
  - Mechanisms for reputation and trust are needed.
  - Ability to describe processes and workflows.

### 12-23: Example

- Imagine a B2C scenario - a user wants to purchase an item from a website.
- the server must:
  - Record the sale in a DB
  - Verify and Process credit card
  - Send order to shipping department
  - Receive confirmation
  - Update inventory.
- What if problems arise?
  - Credit card payment fails after order shipped? Payment processed, but order lost?
- This can all be easily handled in a closed environment, but what about an open environment?
- Standard Web services don't have direct support for transactional models.
  - **This is an active area of development**

### 12-24: Challenges of composition

- How to ensure QoS?
- How to handle trust in an interaction amongst multiple enterprises?
- How to describe complex, long-lived processes?
- Solving composition problems may be computationally demanding - how to scale?
- How to resolve semantic and pragmatic differences among components?

## 12-25: Summary

- A Service-oriented approach can potentially make integration and interoperation easier, both for new systems and as wrappers for legacy architectures.
- Dealing with legacy systems can be a huge challenge in general.
- Services are designed to address the heterogeneity, multiple interests, and dynamism that characterize open environment.
- Today's challenges involve designing protocols on top of existing systems that provide additional semantics.