

Distributed Computing: Trends in Middleware Technologies



Original presentation
by Paul Sivilotti
Modified for CSE 794R/ECE 694R
by Paolo Bucci

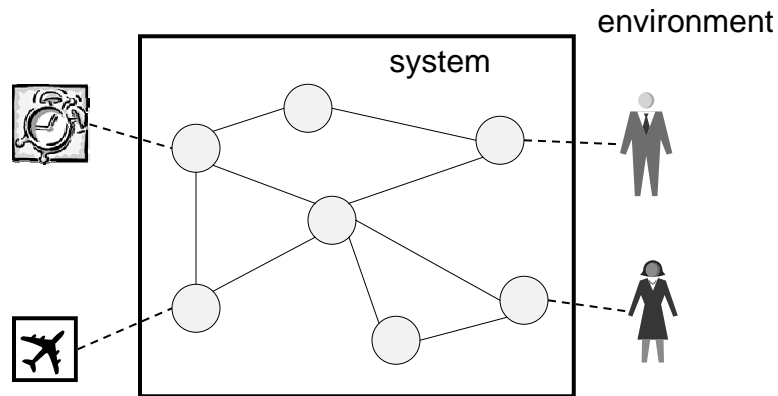


What Is Distributed Computing?

- "When a computer that you didn't even know existed can cause your machine to stop working"
(Leslie Lamport)



Distributed Autonomous Peer-to-peer Systems



Distributed Computing: Trends in Middleware Technologies

3

Why Is Distributed Computing?

- Primary reason: Problem is inherently distributed
 - Physical separation of people
 - Customers, employees, field agents, students, ...
 - Distributed command and control
 - Battlefield logistics, avionics, disaster response, ...
 - Geographic distribution of data
 - Akamai, Google, BitTorrent, ...
 - Integration of legacy systems
 - Access old software on old platforms with new clients

Distributed Computing: Trends in Middleware Technologies

4



Why Is Distributed Computing?

- Secondary reason: Harness unused computation cycles
 - Really the realm of *parallel computing*
 - Problem has a sequential specification
 - Weather prediction, scientific computation, complex simulation, graphics rendering, ...
 - Narrower problem domain helps
 - Technologies: Parallelizing compilers, SMPs, Cluster computing, Grid computing,...



Some Basic Challenges

1. Heterogeneity
 - Architectures, OS's, implementation languages
2. Partial Failures
 - Local failures, global resilience and recovery
3. Concurrency
 - Action interleaving and interference
4. Integration Compatibility & Correctness
 - When/how can a component be used?
5. Security
 - Intrusion prevention/detection/repair; privacy





Evolution of Middleware

- Over the last decade, we've seen four major "epochs"

1. DCE - *classical*



2. CORBA - *romantic*



3. Java RMI/EJB - *modern*



4. Web Services - *postmodern*



Distributed Computing: Trends in Middleware Technologies

7



Cautionary Remarks

1. DCE
 2. CORBA
 3. Java RMI/EJB
 4. Web Services
- Not ordered strictly by increasing power
 - None made obsolete by the next
 - Depends on situation
 - Combination often needed
 - Not always directly comparable
 - DCE is one integration technology for CORBA
 - Java RMI supports CORBA's IIOP



Distributed Computing: Trends in Middleware Technologies

8



Overall Trend

1. DCE
 2. CORBA
 3. Java RMI/EJB
 4. Web Services
- Increasing support for: "loose coupling"
 - Separation of:
 - Development
 - Deployment
 - Discovery
 - Integration
 - Across time, space, companies, platforms, ...



Classic: DCE



- "Distributed Computing Environment"
 - Open Software Foundation
- Basic RPC abstraction
 - Remote procedure calls are easier than sockets
 - Synchronous invocation semantics
- Handles marshalling/demarshalling
 - Allows for heterogeneity of platforms
- IDL for procedure signatures
- Has given security considerable attention





Evolutionary Pressure

- DCE is a descendant of “procedural school” of design
 - Decomposition along functional lines
- Object-orientation is a natural fit for distributed systems
 - Data + method encapsulation for information hiding
 - Data locality for performance



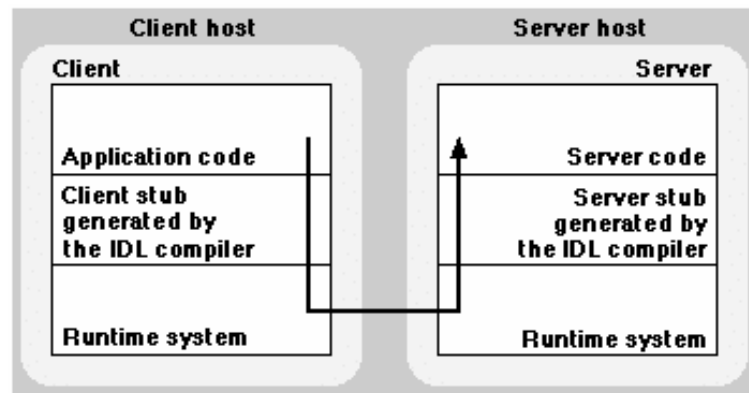
Romantic: CORBA



- “Common Object Request Broker Arch.”
 - The Object Management Group
- Object model for distributed computations
 - Interface inheritance, exceptions, object ref's
- IDL for object interfaces
 - Compile to stubs/skeletons for each platform
- Direct support for asynchrony
- Many “services” for common functionality
 - Transaction, Naming, Event, Security ...



Requesting Services with DCE or CORBA



CORBA: Loose Coupling

- Naming and Trading Service
 - Run-time discovery of remote objects
- DII: Dynamic Invocation Interface
 - Static client must be compiled with remote interface stubs
 - MyStack.push(100);
 - Dynamic invocation created at run-time
 - Request R = create_request(MyStack, ...);
 - R.invoke();





Evolutionary Pressure

- IDL compilation is platform-specific
 - To port server, must regenerate skeletons
 - To add clients in new languages, must generate new stubs
 - (Or use cumbersome DII)
- Solution: eliminate heterogeneity!
 - Use a standard virtual machine
 - Implement VM on many real platforms
 - "write once, run anywhere"



Modern: Java RMI/EJB



- Java interface "Remote"

```
public interface StackI extends Remote {
    void push (int x) throws RemoteException;
    int pop ( ) throws RemoteException;
}
```

```
class RemoteStack implements StackI {...}
```
- Client invocations look like local calls
`MyRemoteStack.push(100);`
- Rmiregistry for dynamic discovery of object references
- EJB introduce separation of concerns
 - Encapsulate business logic of an application
 - Instance behavior vs. session behavior





Java RMI: Loose Coupling

- Pass-by-reference for remote objects
 - Receiver obtains object reference
 - `MyRemoteStack.push(MyRemoteX);`
 - Remote stack can now make calls on remote X
- Pass-by-value for local objects
 - "Value" includes method implementation
 - Receiver dynamically obtains code!
 - `MyRemoteStack.push(MyX);`
 - Remote stack now has a local X



Evolutionary Pressure

- The world wide web
 - It has changed everything else, why not distributed computing?
- Didn't Java evolve from the web?
 - Yes, but not RMI
- Take-home lessons from the web:
 - Simplicity (get/post)
 - Ubiquity





Postmodern: Web Services



- Enabling technology: XML
 - Simple markup language
 - Data structured in nested tags

```
<talk>
  <outline> <topic> Intro </topic> ... </outline>
  <section name=Intro> ... </section>
</talk>
```
 - Extensible: tags are defined in a *schema*
 - All ASCII-based (portability)
 - Simple, ubiquitous data format
- Web Services based on 3 technologies:
 - SOAP, WSDL, UDDI



Distributed Computing: Trends in Middleware
Technologies

19



SOAP

- "Simple Object Access Protocol"
 - Now managed by W3C (submitted to IETF)
 - HTTP is the underlying transport protocol
 - Firewall issues minimized
- XML encoding of invocation request/reply
 - Endpoint information (where to send this)
 - Payload information (what to do there)
- Support led by Apache, IBM, MS...



Distributed Computing: Trends in Middleware
Technologies

20



WSDL

- "Web Service Description Language"
- The IDL of Web Services
- XML grammar for describing services
 - Interface information
 - End-point information
- Typical model turns old development cycle on its head:
 - IDL: write, parse, implement component
 - WSDL: implement component, extract, publish
- Led by Ariba, IBM, Microsoft



UDDI

- "Universal Description, Discovery and Integration"
- White pages for Web Services
 - Centralized list of globally available services
 - Metadata describing interface (WSDL)
 - Ubiquitous protocol (HTTP)
- Supports a variety of queries
 - Both human and programmatic interfaces
- IBM, Microsoft, HP, Ariba
 - Aims to be Google/Amazon/Ebay of services
 - Currently more like old <http://www.ncsa.edu/>





Challenges Solved?

- Dominant trend: Loose coupling
- Makes challenges harder, not easier!
 - ✓ Heterogeneity is directly addressed
 - × Partial failures are more common
 - × Concurrency is harder to control
 - × Integration is more unreliable
 - × Security is more difficult



Report Card

1. Heterogeneity A
 - IIOP, Java, XML, SOAP
2. Partial Failures C
 - TMR, leases, transactions
3. Concurrency C
 - Transaction services, synchronization
4. Compatibility & Correctness F
 - IDL, rmiregistry, WSDL
5. Security D
 - Firewalls, encryption, authentication





Take-home Messages

- Distributed computing is becoming more chaotic, not less
- Technology trend: loose coupling
 - Simple, ubiquitous protocols and access
- Makes some hard problems even harder
 - Semantic integration

