Course Recap

21 lectures
- Background (10)
  - Networking fundamentals
  - Sockets
  - RPC
  - Distributed Objects
- Web Services (11)
  - Architecture
  - SOAP
  - WSDL
  - UDDI
  - State Management
  - GXA Architecture
  - WS-Routing
  - WS-Referral
- Several code walkthroughs
- 6 programming labs
  - Sockets
  - XML-RPC
  - .NET Remoting
  - Web Services – WSDL, SOAP
  - Web Services – UDDI
  - Web Services – WS-Routing
- 2 exams

(From Lecture 1) What are Web Services?

- Depends on whom you ask …
  - A revolutionary new way of building distributed applications
  - The natural evolution of distributed programming APIs
    - Sockets → RPC → Distributed Objects → Web Services
    - To simplify application integration and interoperability
- Main ideas
  - Applications structured as lightweight components, which expose services
    - Example: A Weather component, which offers a GetTemperature service
      - Input parameter: Zip code
      - Output response: An integer that represents the temperature
  - Services discovered, described, and interacted with using standard protocols
    - UDDI, WSDL, SOAP, all of which make heavy use of XML
- Goal: Provide a simple application-to-application interface just like the web has provided a simple human-to-application interface
  - Specifications such as HTML and HTTP, servers and browsers

Announcements

- All labs must be submitted by December 10th, 9:00AM
  - Before Monday morning (December 8th), 9:00AM
    - Use web-based form to submit Lab 5
    - Send Labs 1 – 4 by e-mail to me and TAs
  - After Monday morning (December 8th), 10:00AM
    - Use web-based form to submit Lab 6
    - Send Labs 1 – 5 by e-mail
- Account access will be turned off immediately after December 18th
  - Please move any programs/data you want to before then
- Final exam: December 18th, 10:00am – 11:30am, 109 CIWW
  - Sample questions available in D:\VSDev\Public\vijayk\final-sample.pdf
  - Review on December 9th
  - Contact me ASAP if you cannot make it to the scheduled exam time
(From Lecture 1) What This Course is About

- Understanding the general issues that must be addressed while constructing distributed applications from component pieces
  - Discovery: How do components learn about each other? [UDDI]
  - Description: How does a component learn about another’s interface? [WSDL]
  - Interaction: How are these services invoked? [SOAP]
  - How are messages transferred from one component to another? [HTTP, TCP]
  - How do we encode service parameters, return values in these messages? [XML]
  - How do the components at either end know which encoding to use?

- Understanding how different approaches deal with these issues
  - Sockets, Remote Procedure Calls, Distributed Objects, Web Services

- Getting some practical experience in using these approaches
  - C# language, .NET framework class libraries, Visual Studio.NET IDE (tools)

Sockets, RPC, Distributed Objects … Web Services

Remote Procedure Calls

- Discovery
  - Server program identified by “name”, name server maps to location
  - Client must still know about the host

- Description
  - Common interface defining the RPC protocol
    - Parameters and return structures
    - Function signatures

- Interaction
  - Clients and server stubs facilitate procedure-call like interactions
    - All message passing is hidden from the application writer
  - Remote calls communicate structured data
  - Stateless interaction
    - Remote calls may be handled by one or more server-side instances

Distributed Objects (.NET Remoting)

- Discovery
  - Explicit: RPC-like name-to-instance mapping
  - Implicit: Portable object references permit clients to interact with server programs without knowing their location

- Description
  - RPC-like common interface, augmented with …
  - Run-time type inspection of object reference
    - However, comes at the cost of language/CLR dependence

- Interaction
  - RPC-like procedure calls
    - Implementation favors local network interaction
  - Rich support for state management
    - Singleton, single call, explicitly marshal-ed, client-activated objects … leases
Web Services architecture provides XML-based, language-neutral standards for

- Discovery [UDDI, WS-Inspection]
  - Approximate location-independent nature of object references in distributed object systems by relying on intermediate brokers, who store/categorize/provide information about services
- Description [WSDL]
  - Approximate run-type type inspection by encoding the service types/interface into an XML document that can be interpreted by clients
- Interaction [SOAP]
  - RPC-like procedure calls + asynchronous invocations
    - Implementation uses standard, interoperable protocols (HTTP)
  - Security tokens for authentication/encryption,
  - Message route information, …

Core Specifications: SOAP, WSDL, and UDDI

- SOAP: Simple Object Access Protocol
  - XML-RPC-like request/response protocol +
  - Support for asynchronous invocations
  - Encoding of additional information in the message
    - Security tokens for authentication/encryption,
    - Message route information, …
- WSDL: Web Services Description Language
  - RPC, Distributed Objects-like common structs/interface +
  - Support for asynchronous invocations
  - Possibility of language-neutral (and automatic) interpretation
    - Web-services tools use WSDL description of a service to automatically generate a SOAP-capable proxy
- UDDI: Universal Description, Discovery, and Integration
  - Defines ways of mapping service “characteristics” to service providers
  - “characteristics” generalize “names”

GXA Specifications: WS-*, BPEL4WS, …

- Enable construction of applications involving complex orchestration among multiple web services
- WS-Routing
  - Permits routing of SOAP messages via intermediaries
- WS-Referral
  - Permits dynamic control of SOAP-level routing information
- WS-Inspection
- WS-Security
- BPEL4WS
- WS-Transaction (old), WS-Coordination (September 2003)
  - Atomic actions involving multiple services
- More coming ….
Distributed Objects versus Web Services

- Distributed Objects (e.g., .NET Remoting)
  - Tightly coupled: share a common language/IDL basis
    - Permits passing of object references across process boundaries
  - Better support for programming transparency
    - Little distinction between local and remote object reference
  - Typically used where performance is a concern
    - .NET remoting infrastructure provides general, extensible mechanisms

- Web Services
  - Loosely coupled: can be written in different languages
    - Support call-by-value semantics, no notion of opaque object references
  - Remote calls are identified as such (explicit proxy)
  - Primary focus is interoperability, performance less so
    - SOAP intermediaries are a step in the latter direction

Web Services – Future Directions

[Near term]
A lot of work needs to be done before web services see widespread use
- GXA-like specifications to encode common “patterns” of usage
  - Security, transactions, recovery, …
- Performance considerations
  - Connecting to a local instance of the service
  - On-demand caching/replication of web service functionality
  - …
- Composition mechanisms
  - Buying a book involves
    - book service, shipping service, credit-card service, …
  - Is it possible to create these compositions on the fly?
    - Say to choose the combination that realizes the cheapest, fastest purchase?

Web Services – Future Directions (cont’d)

[More longer term]
Enabling truly dynamic discovery and integration of web services
- UDDI enables dynamic discovery …
- … but only if you know what you are looking for
  - E.g., the tModel for the service, the categories a service belongs to
- WSDL enables dynamic proxy creation and invocation …
- … but only if you know what an operation corresponds to
  - “Compute” in “StringReverser” service “reverses strings”
  - Requires making a semantic connection

“Semantic” Web Services

- An effort to encode the semantics of a service as part of its description
  - Would ideally allow programmatic discovery of services that provide the right functionality
    - Beyond just type- or category-based
- Origins of the effort in the now-classic Semantic Web paper
  - A proposal to add semantic annotations to static web content to make the content machine-understandable
  - Example
    - Consider a program reading the course web page that wants to understand who the instructor is (to send spam e-mail about appropriate textbooks)?
    - One can imagine annotating the relevant information within an <instructor> … </instructor> tag
    - But how does the program know that the “instructor” tag does in fact refer to the instructor?
“Semantic” Web Services: The Problem

Need to add “semantics” (meaning) to the tags

Two common approaches

• External agreement on meaning of annotations
  – E.g., via XML namespaces
    • Agree on the meaning of a set of annotation tags
  – Problems with this approach
    • Inflexible
    • Limited number of things can be expressed

• Use ontologies to specify meaning of annotations
  – Ontologies provide a vocabulary of terms
  – New terms can be formed by combining existing ones
  – Meaning (semantics) of such terms is formally specified
  – Can also specify relationships between terms in multiple ontologies

Semantic Web Services: Ontologies

• An ontology describes a formal specification of a certain domain
  – Shared understanding of a domain of interest
  – Formal and machine manipulable model of a domain of interest

• Ontologies capture two aspects
  – Names for important concepts
    • “String” refers to an object whose members are a kind of data type
  – Background knowledge/constraints on the domain
    • “Reversing” a string produces another string, with the characters in reverse
    • Typically expressed using relations between classes

• Several researchers working on coming up with
  – appropriate ontology languages (the “how”)
  – ontologies (the “what”), and
  – query techniques for matching concepts in one ontology to that in another

1-Slide Overview of Ontology Languages

• Graphical
  – E.g., RDF (Resource Description Framework)
  – Allows statements about properties of resources
    
    \[
    \text{<description about="some.uri/person/vijayk">}
    \text{<hasHomePage resource=http://www.cs.nyu.edu/vijayk/>}
    \text{</description>}
    \]
  – Both properties and resources can be general URIs
  – Meanings of properties specified using RDF Schema language
    • Defines concepts such as “type”, “range”, “domain”, “subClassOf”, …
  – Turns out to be hard to automatically reason with

• Logic-based
  – OIL, DAML+OIL, OWL (Web Ontology Language)
    • Based on a formal model called Description Logic
    • Better understanding of what can be expressed, what can be computed, how fast, …