Web Services and Service Oriented Architecture

CS 4720 – Mobile Application Development
The traditional software model

- Organizations build it all from scratch
  - “We can't trust anyone! Our competitors will sabotage us!”
  - “We must own everything! Hardware and software!”

- Companies that build software need components that do X or Y...
  - ... so they buy COTS components
  - Service contracts, new releases, regression testing...
The traditional software model

• How well does this work?

• Some project data from the DOD:
  – 47% of software delivered could not be used
    • Usually didn't meet requirements
  – 29% of funded software never delivered
    • Usually canceled due to cost/schedule overruns
  – 19% of software useful after extensive rework
    • Costs 36 times more to fix problems after release
A push to distribution

• “We're really good at __________, how can we get our __________ out for people to use?”

• The idea of distributed computing
  – “We're good at X, but not so good at Y…”
  – “What if we got someone to help us with Y... but in a way that we didn’t have to include any fancy libraries…
  – … and we could use them in web apps! And mobile devices! And even desktop apps!”
A Web Service

• From W3C: “a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards”
A Web Service?

• Huh?

• A bit more simply: A program, accessible via the Internet, that can do some function (either for free or a fee) and can be dynamically discovered and used.

• So... I use web apps all the time... are those web services?
Web Service as RPC

• The earliest form of a true “web services” was an RPC – remote procedure call.

• Exactly what it sounds like – there is an exposed function/method that is accessed via the web where you pass the parameters and the method name and you get back a return value.

• Notice how this is very different from the RESTful model (verbs vs. nouns)
Web Service as SOA

- RPC was okay... but it turned out to be a bit more language specific than we'd like
- What if we just a structured message (like an XML document) that described what we wanted, as opposed to knowing the exact function call?
- This is the basis of the Service-Oriented Architecture
What is a service?

"A service is a discoverable resource that executes a repeatable task, and is described by an externalized service specification."

- Business alignment – business requirements
- Specifications – self-contained, well specified
- Reusability – general enough to be reused
- Agreements – based on function, not platform
- Hosting and discoverability – available
- Aggregation – can be combined to make bigger services
• What is an architecture?

• "A formal description of a system, or a detailed plan of the system at component level to guide its implementation."

• “The structure of components, their interrelationships, and the principles and guidelines governing their design and evolution over time."
Architecture

• Architecture is:
  – A high-enough level of abstraction that the system can be viewed as a whole and yet still provides enough information to make decisions.
  – Supports the functionality of the system.
  – All implementation details are hidden.

• Service orientation is a way of integrating a business as a set of linked services.
What are we actually trying to do?

- Receive PO
- Fork
  - Compute Export Tax
  - Compute International Shipping
- Merge
- Compute Subtotal
  - [ship within US]
  - [ship outside US]
- Compute Shipping Cost
- Compute Total
- Ship Order
- Join
- Update Customer Profile
- Get Items from Inventory
- Compute Subtotal
- Compute Export Tax
- Compute International Shipping
- Compute Total
- Ship Order
SOA – The Quick Version

• Right now, you probably think of a software system as being a collection of classes / objects

• But users don't think of systems like that... they think of systems as sets of functionality that help them do something

• So... why do we use objects?
  – Easier to model
  – Easier to program
  – Easier to explain to other programmers
Or is it actually easier?

- Turn our idea of a system 90 degrees
- Functionality objects (procedural abstraction) is the key idea, not world objects (data abstraction)
- This is the key in SOA
  - An SOA system has the functionalities as the main players, not the objects themselves
  - But more so, these services are provided by many different players
But what language do we speak?

- One early way to do web services was with SOAP
- SOAP - *Simple Object Access Protocol*
  - A communication protocol
  - A format for sending messages
  - XML based
- Not really much more than an HTTP request that follows XML/SOAP standards
SOAP Model

- Web service
- WSDL interface
- SOAP messages
- Transfer protocol (e.g., HTTP)
- TCP/IP stack
SOAP Model
SOAP Request

POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
SOAP Response

- HTTP/1.1 200 OK
  Content-Type: application/soap+xml; charset=utf-8
  Content-Length: nnn

```xml
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>
</soap:Envelope>
```
Parts of SOAP

• The Envelope – required root element defining the document as being a SOAP request
• The Header – not required, but contains authentication and/or payment info for the request
• The Body – the actual message being sent
Using a Web Service

- If you took the time to create a service, then you probably want people to use it.
- In order to use a service, users need to know what information they can send the service and what information is going to be sent back.
- What would be even better is if the software could do all this automatically...
Enter WSDL

- **WSDL**: Web Services Description Language
  - An XML document used to give the specifics of a service
    - Location
    - Methods
    - Allowed messages
    - Potential error messages
Enter WSDL

- `<definitions>`: Root WSDL Element
- `<types>`: What data types will be transmitted?
- `<message>`: What messages will be transmitted?
- `<portType>`: What operations (functions) will be supported?
- `<binding>`: How will the messages be transmitted on the wire? What SOAP-specific details are there?
- `<service>`: Where is the service located?
Quick Sidestep: UDDI

- UDDI: Universal Description, Discovery and Integration
- A solution in search of a problem
- The idea is okay: a language and schema for allowing people to publish their WSDL schemas so that others can discover their services
- IBM, Microsoft, and SAP announced they were closing their public UDDI nodes in January 2006
- No one used it!
Quick Sidestep: UDDI

• Where might it work decently?
  – Perhaps inside a single organization for internal code

• The public sides were either:
  – Empty
  – Overrun with junk

• Whichever it was UDDI, isn't really used

• Quote Marty Humphrey: “It was a good problem to solve, but a terrible solution.”
The Original Idea

Service provider

Publish

Service broker

Find

Service requester

Bind
Back to WSDL

- We DO use WSDL though
- It is how a web service is bound to an application
- It's more XML...
- But... the best part is...
The Best Part about WSDL

• No one ever writes the stuff themselves!
• It's auto-generated!
The Worst Part

• This was really complicated
• Even with auto-generating code, it was tough to build and maintain
• It made it easier for whoever wanted to consume the service (theoretically), but even then there was a lot of setup
• People were already starting to parse their own XML or JSON...
Modern REST APIs

• Most all modern REST web services now operate using JSON – JavaScript Object Notation
  – Easy to parse
  – Easy to create
  – Most web apps are already using JavaScript, so it works seamlessly
  – JSON parsing isn’t expensive for other platforms
Modern REST APIs

- Consider https://dev.twitter.com/rest/public
- Let’s examine these in the context of the five aspects of REST
- The purpose of a REST API web service is to:
  - Expose functionality for others to use
  - Allow that functionality to be built in to other apps
  - Make it easy to use for various platforms
  - Still allow the developer to make money some way on the service
Which brings us back to mobile!

• With mobile:
  – We like REST! It works well with network that comes and goes and how we use mobile devices!
  – We like MVC! It makes sense in how we build our apps!
  – We like web services! It allows us to utilize powerful features developed by others to make our own apps even better!
  – Now... how do we do it???