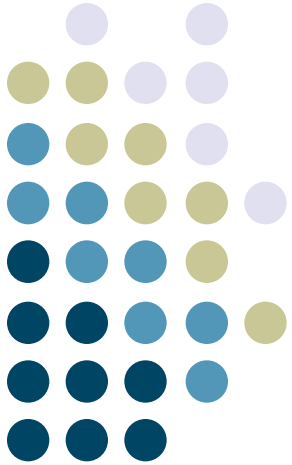
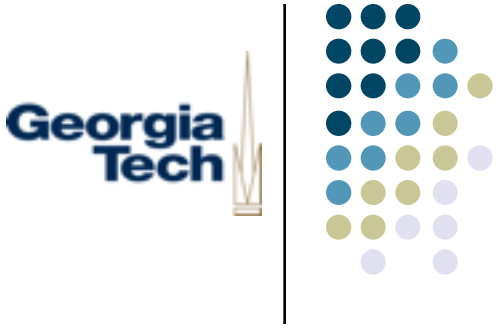
Extracting Information from the Web (Web Services and Friends)



Artificial Intelligence

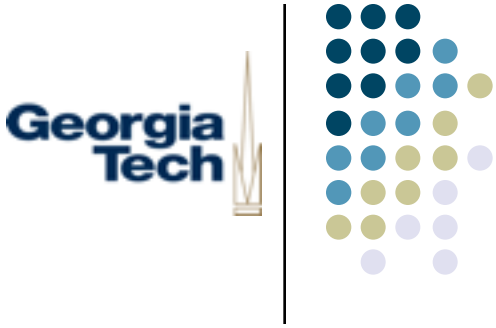
- Variety of techniques for making machines able to achieve goals in the world in ways that mimic human abilities
  - Techniques do not necessarily have to mimic the biological *ways* in which these abilities in humans work though
- Long a focus of study for CS
- Examples:
  - Chess playing
  - FPS games
  - Speech recognition
  - Image recognition
  - Recommender systems (machine learning)
  - Handwriting recognition systems (classifiers)
- One constant though: "As soon as you figure out how to do it, it's no longer AI"



How to Exploit AI Techniques

- Find the algorithms, understand them, implement them!
  - Many are math-intensive
  - Wide variety of algorithms to cover: learning how to write classifiers doesn't help you write game AI
- Find someone else's code and integrate it
  - All the usual problems of dealing with the idiosyncrasies of others' code, Jython integration issues (a la Swing's weirdness), etc.

Another Approach: Exploiting *Human* Intelligence



- There's a lot of knowledge out there already
- Some of it is encoded in a way that machines can make sense of it
- If you're really clever, you may be able to get people to help out directly
- Why? Humans are generally smarter than machines

"Computers are worthless. They can only give you answers."

- Pablo Picasso

"Computers are incredibly fast, accurate, and stupid. Human beings are incredibly slow, inaccurate, and brilliant."

- Albert Einstein

Exploiting Human Intelligence: Approach #1

- Clever UI design: make the people do your work for you without them knowing it
- Example: the ESP game (http://www.espgame.org/) [Luis von Ahn and Laura Dabbish]

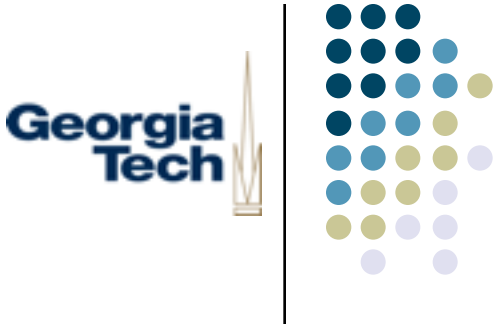
The ESP Game

- Two player web-based game
- You are randomly paired with an online partner
- Both see the same image
- Goal is to guess what your partner is typing about the image
- As soon as a guess of yours is equal to a guess that your partner has made you get a new image
- "Taboo words" can't be used
- You get points each time you agree with your partner; number of points depends on number of taboo words
  - More taboo words -> harder to guess -> more points

Behind the Scenes

- Guise: moderately entertaining game
- Real goal: label all images on the web
  - Provide a textual search engine for images
  - Provide meaningful alternative text to visually impaired users
- The ESP game is a front end to *image tagging*
  - Annotate images with terms that describe them
  - Human-provided information is more accurate, richer, more subtle than machine analysis of the image
  - Most approaches don't even do this: rely on text in <IMG> tags
- Taboo words are words tags that have already been found
  - System rewards refinement of tags with more points
- Already collected 14M labels for approximately 7M images
- Doing this is more an art than a science, but it's way cool...

Accessing Human Intelligence: Approach #2



- Find latent knowledge embedded in the world and *mine* it
- The web makes this easier than it's ever been before

- In all likelihood, this provides different sorts of knowledge than "traditional" AI (you probably couldn't build opponents in a first-person shooter game using this technique)

Example

- Through the act of buying, people express their preferences, tastes, opinions
- Amazon has mountains of this data
  - Not just who has bought what
  - Similarities between books
  - Confluences of interest among book buyers
- All of this encoded into the Amazon website, waiting to be used
- How might you use it?
  - Social networking applications?
  - Suggest dating opportunities based on overlap of Amazon recommendation lists?
  - Visualize degree-of-separation between people, based on similarities of their book tastes?

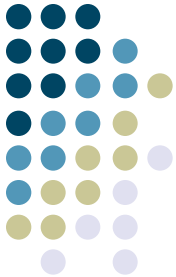
Example

- People are very good at separating the wheat from the chaff when it comes to browsing web pages
  - Some you consider authoritative, fun, etc., and may check on a day-by-day basis
  - Some you may link to yourself
- Google knows how people rate pages on the web, by calculating how many people link to certain pages: PageRank algorithm
- Hard-core algorithmic work running on their servers...but results are sitting around, waiting for you to reap
- How might you use it?
  - Build an app that provides easy access to authoritative information around you
  - Example: ubicomp application that, as I walk around the city, gives me top-ranked info on the business I'm nearest ("how good is this restaurant?")

Example

- People are very good at understanding relationships, subtle differences between words
- Thesaurus.com provides an expert-eye view of word similarity
- Massive lists of semantic relationships among words, waiting to be mined and extracted
- How might you use it?
  - Creative writing app that provides built-in synonym lookup on every word
  - Image tagger that uses synonyms of suggested words, to broaden search possibilities
  - del.icio.us social bookmarks manager that automatically uses synonyms to provide search based on word similarity

Extracting Information from the Web



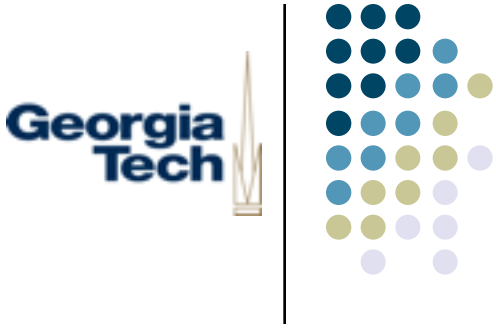
- Can be really *really* painful

Mining nice looking page:	Means parsing this:
	<style type="text/css"><!-- BODY { font-family: verdana,arial,helvetica,sans-serif; font-size: small; background-color: #FFFFFF; color: #000000; margin-top: 0px; } TD, TH { font-family: verdana,arial,helvetica,sans-serif; font-size: small; } a:link { font-family: verdana,arial,helvetica,sans-serif; color: #003399; } a:visited { font-family: verdana,arial,helvetica,sans-serif; color: #996633; } a:active { font-family: verdana,arial,helvetica,sans-serif; color: #FF9933; } .serif { font-family: times,serif; font-size: medium; } .sans { font-family: verdana,arial,helvetica,sans-serif; font-size: medium; } .small { font-family: verdana,arial,helvetica,sans-serif; font-size: small; } .h1 { font-family: verdana,arial,helvetica,sans-serif; color: #CC6600; font-size: medium; } .h3color { font-family: verdana,arial,helvetica,sans-serif; color: #CC6600; font-size: small; } .tiny { font-family: verdana,arial,helvetica,sans-serif; font-size: x-small; } .listprice { font-family: arial,verdana,helvetica,sans-serif; text-decoration: line-through; font-size: small; } .attention { background-color: #FFFFD5; } .price { font-family: arial,verdana,helvetica,sans-serif; color: #990000; font-size: small; } .tinyprice { font-family: verdana,arial,helvetica,sans-serif; color: #990000; font-size: x-small; } .highlight { font-family: verdana,arial,helvetica,sans-serif; color: #990000; font-size: small; } .alertgreen { color: #009900; font-weight: bold; } .topnav { font-family: verdana,arial,helvetica,sans-serif; font-size: 12px; text-decoration: none; } .topnav a:link, .topnav a:visited { text-decoration: none; color: #003399; } .topnav a:hover { text-decoration: none; color: #CC6600; } .topnav-active a:link, .topnav-active a:visited { font-family: verdana,arial,helvetica,sans-serif; font-size: 12px; color: #CC6600; text-decoration: none; } .eyebrow { font-family: verdana,arial,helvetica,sans-serif; font-size: 10px; font-weight: bold;text-transform: uppercase; text-decoration: none; color: #FFFFFF; } .eyebrow a:link { text-decoration: none; } .popover-tiny { font-size: x-small; font-family: verdana,arial,helvetica,sans-serif; } .popover-tiny a, .popover-tiny a:visited { text-decoration: none; color: #003399; } .popover-tiny a:hover { text-decoration: none; color: #CC6600; }

Strategies

- Some web sites try to make this easier for you

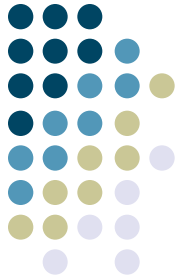
http://en.wikipedia.org/wiki/jython	http://en.wikipedia.org/wiki/jython?action=raw



Strategies

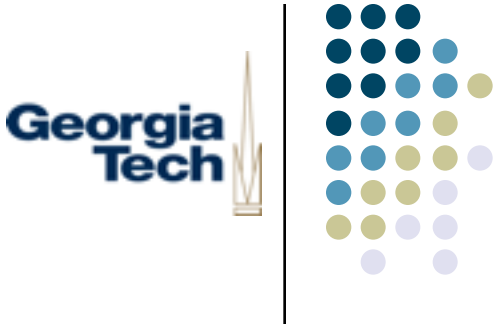
- There are tools to help with parsing
- *DOM - the Document Object Model (and related tools)*
- Makes HTML-formatted text (along with CSS, JavaScript, etc.) look like a tree data structure
  - (Relatively) easy programmatic tools for walking through the structure, extracting key bits, etc.
- Many APIs and programming models, some simple, some not
- Caveat: if the page's structure changes, you're hosed

Example (Simple) DOM Usage



- httpunit: http://httpunit.sourceforge.net

```
// need js.jar, nekohtml.jar, xerces.jar and perhaps other in classpath!

import com.meterware.httpunit as httpunit
import sys

class Test:
	def __init__(self, url):
		httpunit.HttpUnitOptions.setExceptionsThrownOnScriptError(False);
		wc = httpunit.WebConversation()
		req = httpunit.GetMethodWebRequest(url)
		resp = wc.getResponse(req)
		page = wc.getCurrentPage()
		images = page.getImages()
		forms = page.getForms()
		links = page.getLinks()

		print "---- Images ----"
		for i in images:
			if i.link != None:
				print i.name, "(", i.link.getURLString(), ")"

		print "---- Forms ----"
		for f in forms:
			print f.action

		print "---- Links ----"
		for l in links:
			print l.text, "(", l.getURLString(), ")"

if __name__ == "__main__":
	t = Test(sys.argv[1])
```



A Strategy Recap

- So far we have two strategies:
  - Either get the site to return the *least* information possible (a la Wikipedia), and then parse it
  - Or, get ready to do some heavy-duty HTML parsing (perhaps with the assistance of one of the many DOM libraries)
- Why so hard?
- Largely a mismatch in goals:
  - The web is designed to provide information to *people*, not *programs*
  - Writing a program to extract information from web content (as opposed to web structure) is both hard and fragile
- Is there an equivalent of the web designed for *programs*, not *people?*

Web Services

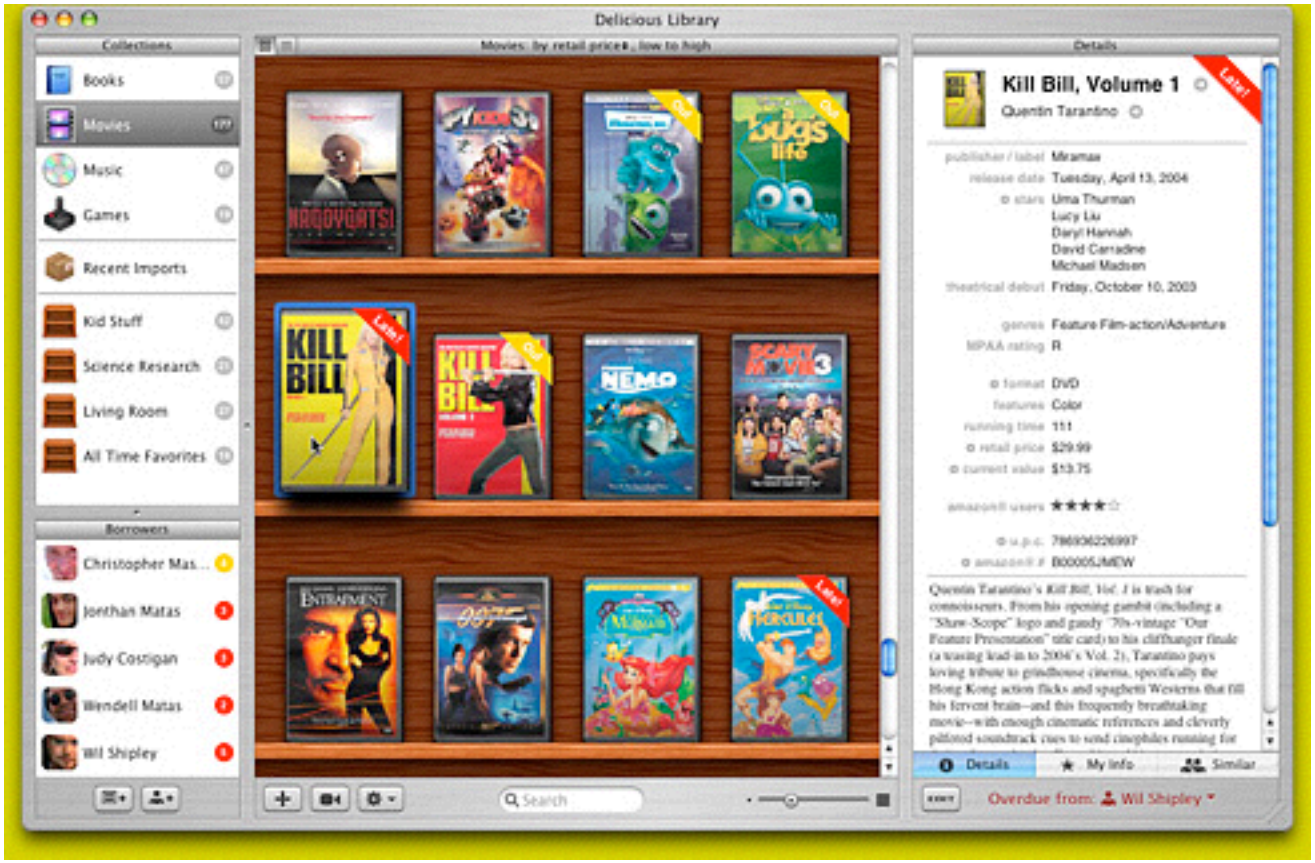
- The Big New Thing on the web (well, not so "new")
- *Web Services* are the web for programs
- Provide *network protocols* designed for accessing information in a structured and systematic way, amenable to machine processing
- Co-exists alongside the regular human-readable web
- Example: Amazon
  - Has a human-readable web site (the one you use)
  - Has a machine-readable *web services* site (an extra service that provides access to Amazon's data through a standard network protocol)
- Who has Web Services?
  - Amazon, Google, FedEx, eBay, PayPal, ...
  - Makes extracting the information from these sites *way easier* than trying to use the human-readable web

Other Uses for Web Services: Alternative UIs

- Piranha Pricecheck: customized Amazon UI for smart phones
  - Access reviews, deals, etc., faster than wireless web

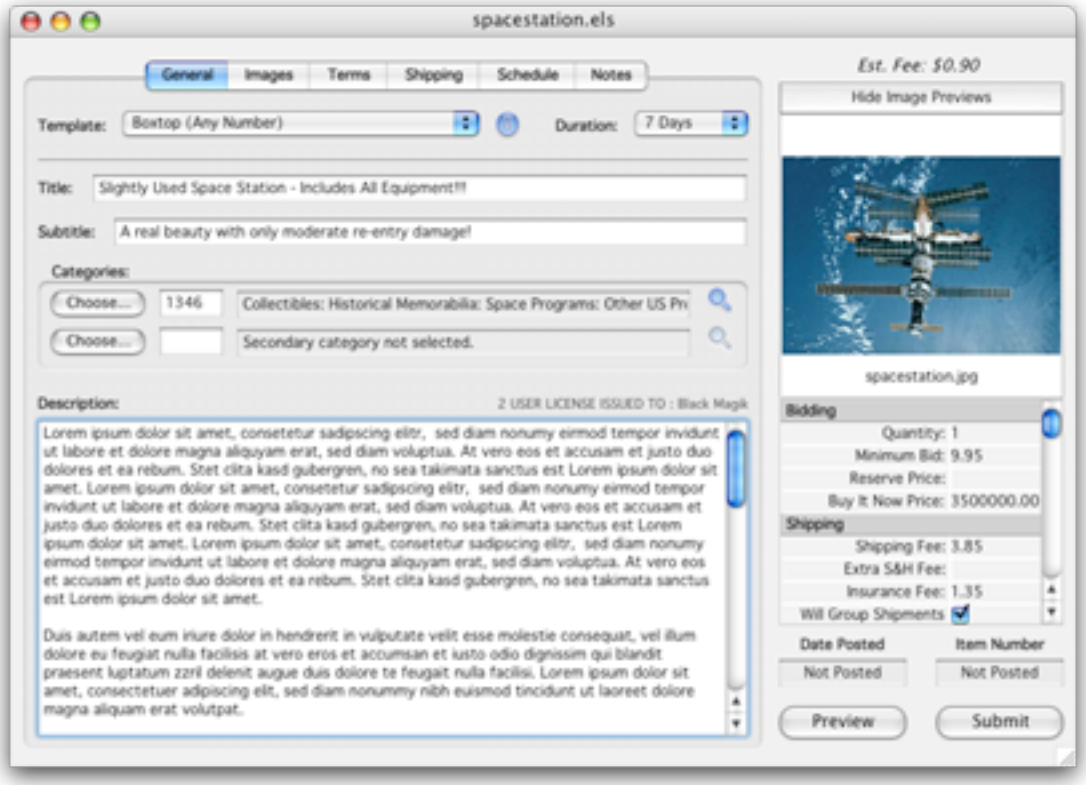
Other Uses for Web Services: Alternative UIs

- Delicious Library: uses Amazon as a back-end data source for personal book/CD/DVD catalogs

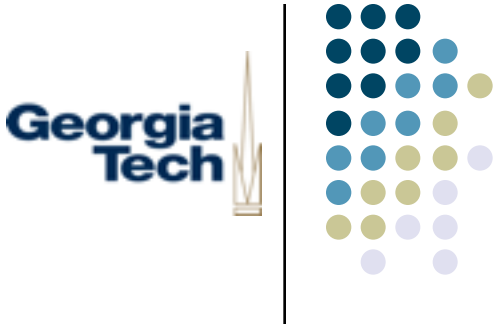


Other Uses for Web Services: Alternative UIs

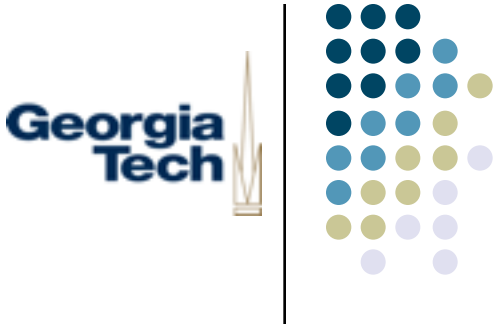
- eLister: Custom application for creating and managing eBay auctions



The Building Blocks of Web Services



- Move away from simple HTTP and HTML
  - Too unstructured for easy machine use
- New tools:
  - Ways to define the operations available on a service
    - Parseable specification of the network protocol for a service
    - Not as ad hoc and error prone as a human-readable specification
  - Ways to communicate with the service in a structured way
    - HTTP has basically GET and POST and crams everything into these
    - Need something that looks and works more like function calls in a program: pass parameters to service, get results from service
- Requirement: language/OS/platform independent
  - So, no crazy pickled Jython data structures
  - Too Jython dependent, not easy to write clients in other languages



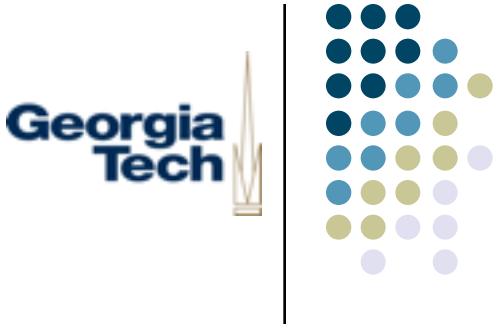
WSDL

- The *Web Services Description Language*
- Pronounced *whizz-dull*
- A markup language for describing network protocols for services
  - What operations
  - What parameters
  - What return values
- You create a document that describes the protocol that your service will speak
  - Or, if you're writing a client, you find and use the document describing the protocol of the service you want to communicate with

WSDL Pros and Cons

- Pros:
  - Allows you to write specs for protocols that can be *parsed by machines*
  - Why is this important?
    - Can have programs that *generate* the necessary networking boilerplate
    - More robust, less errorprone, than hand-coded protocols
    - The *spec is the spec*, rather than the *implementation is the spec*
  - Platform independent: can have programs that generate WSDL protocol implementations in any language
- Cons:
  - Verbose, verbose, verbose...
  - Still only specifies the syntax of a network protocol--you have to specify the semantics

WSDL Concepts



- *Ports:* endpoints for communication. Think of them as analogous to a socket bound to an address (IP address + port) on which a program is listening
- Ports are *typed*: meaning that they are defined to accept a certain set of operations (analogous to the message types in our IM protocol)
- *Services:* collections of one or more ports
- *Messages:* data exchanged over ports
- Messages are also typed: meaning that the data they contain is in a fixed, agreed-upon format

Using WSDL

- WSDL is an XML-based language (read: monstrously verbose)
- Example: StockQuote service
  - Defines the operations available on the service
  - GetTradePrice request (client-to-server)
    - Takes ticker symbol of a stock, and a time
    - Returns price in a response

WSDL StockQuote Service

```
<?xml version="1.0"?>
<definitions name="StockQuote"
          targetNamespace="example.com–stockquote.wsdl"
          xmlns:tns="example.com–stockquote.wsdl"
          xmlns:xsd="www.w3.org–XMLSchema"
          xmlns:xsd1="example.com–stockquote.xsd"
          xmlns:soap="schemas.xmlsoap.org–soap"
          xmlns="schemas.xmlsoap.org–wsdl">

    <message name="GetTradePriceInput">
        <part name="tickerSymbol" element="xsd:string"/>
        <part name="time" element="xsd:timeInstant"/>
    </message>

    <message name="GetTradePriceOutput">
        <part name="result" type="xsd:float"/>
    </message>

    <portType name="StockQuotePortType">
        <operation name="GetTradePrice">
           <input message="tns:GetTradePriceInput"/>
           <output message="tns:GetTradePriceOutput"/>
        </operation>
    </portType>

    <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
        <soap:binding style="rpc" transport="schemas.xmlsoap.org–http"/>
        <operation name="GetTradePrice">
           <soap:operation soapAction="example.com–GetTradePrice"/>
           <input>
               <soap:body use="encoded" namespace="example.com–stockquote"
                          encodingStyle="schemas.xmlsoap.org–encoding"/>
           </input>
           <output>
               <soap:body use="encoded" namespace="example.com–stockquote"
                          encodingStyle="schemas.xmlsoap.org–encoding"/>
           </output>
        </operation>>
    </binding>

    <service name="StockQuoteService">
        <documentation>My first service</documentation>
        <port name="StockQuotePort" binding="tns:StockQuoteBinding">
           <soap:address location="example.com–stockquote"/>
        </port>
    </service>
</definitions>
```

Boilerplate, pointers to related WSDL files, format version numbers, etc.

Defines the input argument type of the request

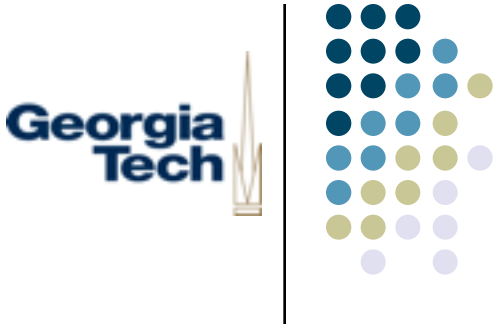
Defines the output result type

Sets up a *port* containing one operation: GetTradePrice, saying what the input and return parameter types are

Boilerplate, describing how messages will be encoded.

Finally, define the *service* itself, with a documentation string and the port description

SOAP

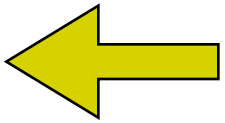


- WSDL defines the operations defined by a service
- *SOAP* is the mechanism for encoding the actual messages and transmitting them to and from the server
- SOAP stands for *Simple Object Access Protocol*
- Wins the award for the **worst name ever**, because SOAP is many things, but not simple
- SOAP messages are encoding using XML
  - So the data that gets sent "on the wire" is an XML document that structures the message, its parameters, etc.
  - At least as verbose as WSDL
  - As you might expect, not the fastest protocol in the world...
- Again, SOAP is the language that defines the messages that are actually sent and received
  - This is what you would have to parse if you implemented a web service or client "by hand"

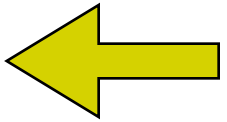
SOAP Example

- This is the "on the wire" data transmitted in the WSDL example
- Curious fact: SOAP messages are often transmitted *using* HTTP
  - They are the payload in HTTP GET and POST messages
- Request:

```
POST /StockQuote HTTP/1.1
 Host: www.stockquoteserver.com
 Content-Type: text/xml; charset="utf-8"
 Content-Length: nnnn
 SOAPAction: "Some-URI"

 <SOAP-ENV:Envelope
   xmlns:SOAP-ENV="schemas.xmlsoap.org—envelope"
   SOAP-ENV:encodingStyle="schemas.xmlsoap.org—encoding">
    <SOAP-ENV:Body>
        <m:GetLastTradePrice xmlns:m="Some-URI">
            <symbol>DIS</symbol>
        </m:GetLastTradePrice>
    </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
```



HTTP headers

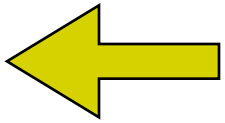


Actual message contents and parameters

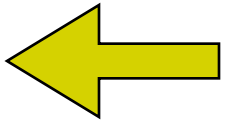
SOAP Example (cont'd)

- Here's the response:

```
HTTP/1.1 200 OK
 Content-Type: text/xml; charset="utf-8"
 Content-Length: nnnn

 <SOAP-ENV:Envelope
   xmlns:SOAP-ENV="schemas.xmlsoap.org—envelope"
   SOAP-ENV:encodingStyle="schemas.xmlsoap.org—encoding"/>
    <SOAP-ENV:Body>
        <m:GetLastTradePriceResponse xmlns:m="Some-URI">
            <Price>34.5</Price>
        </m:GetLastTradePriceResponse>
    </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
```



HTTP headers

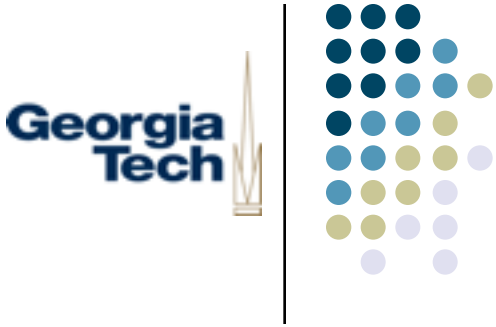


Actual message contents and parameters

Implementation Notes

- At the lowest layer, all of this still uses sockets
  - Many, many layers on top of sockets:
    - SOAP is often sent *using* HTTP (i.e., embedded inside HTTP GET and POST messages)
    - HTTP is generally layered atop raw IP sockets
- So, if you wanted to, you could do all of this just using send() and recv() on top of sockets
- (But you wouldn't want to)

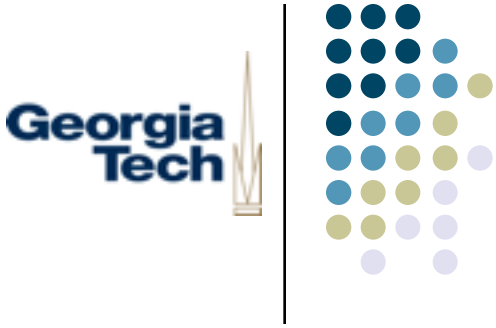
Why is Any of This a Good Thing?



- Because you generally don't do any SOAP by hand
  - You'd only do WSDL by hand if you were creating your own web service
- Designed to let tools manipulate the protocols
  - We'll see how these work next
- Inertia: even though web services might not be the ideal approach, they've been implemented widely enough that they've got momentum
  - Widespread enough that it's good to know how to work with them
- Standards: efforts are underway to standardize some service interfaces
  - Means you could write one client that could work with multiple web services
  - *Semantic Web:* defining web interfaces for basically everything

Using Web Services

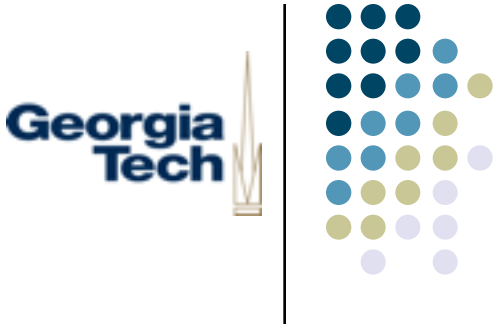
- Using tools to create web services code
- Feed WSDL file into tool that emits *stub* code
  - Contains all of the networking code, exposes function call interfaces
  - Stubs take care of mechanics--you have to write the semantic code
    - Eg: stub would handle all of the parameter passing of ONLINE_USERS; you'd still have to use those parameters to update your GUI
  - Call into the generated stubs whenever you need to communicate with the server
- *Remote Procedure Calls*: make networked message passing look like function calls (sort of)
- Very often, you'll use web services without actually seeing any XML



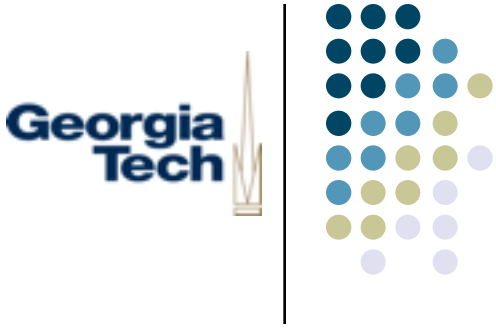
Web Services Tools

- Many, many flavors
- I couldn't find any Python/Jython ones that didn't suck
- Reasonable Java tool: Apache Axis
  - http://ws.apache.org/axis/
  - Provides command line tool for generating stubs
  - Generated stub code requires Axis library (and a bunch of other stuff)
  - More complexity than you need for most things
- Outline:
  - Feed WSDL file into tool, get out Java source code for stubs
  - Compile Java source code
  - Use compiled Java code from Jython to communicate with server

Example: Using the Google API (the hard way...)



- Download and install Apache Axis
  - ZIP file, will create a directory wherever you unpack it
- Generate stubs from the WSDL file
  - set CLASSPATH to include *full path* to necessary libraries that come with Axis
    - axis.jar, commons-logging.jar, log4j-1.2.8.jar, wsdl4j.jar, commons-discovery.jar, jaxrpc.jar, saaj.jar
  - java org.apache.axis.wsdl.WSDL2Java GoogleSearch.wsdl
  - This creates a directory GoogleSearch with the Java source code in it
- Compile the Java source code
  - cd GoogleSearch
  - javac *.java
- Bundle the compiled code (".class" files) into a JAR ("Java Archive")
  - cd ..
  - jar cvf GoogleSearch.jar ./GoogleSearch
  - This creates a new file, GoogleSearch.jar, containing all the stubs
- Add GoogleSearch.jar to your CLASSPATH, and you can now use it from Jython!
  - import GoogleSearch as google



Or.... the easy way!

- Let Google do the work for you:
  - Google used to provide a "pre built" SOAP API
- This has gone away, but we're using a replacement I created for this class--typical of this style of web service API
  - Download googlehack.jar from the class website
  - (Optional, but recommended) Sign up with Google to get an "API Key"
    - This is a special code that lets Google know that it's you using their service
    - Less likely to "have the plug pulled" on you
    - Can reuse after this class
    - http://code.google.com/apis/loader/signup.html

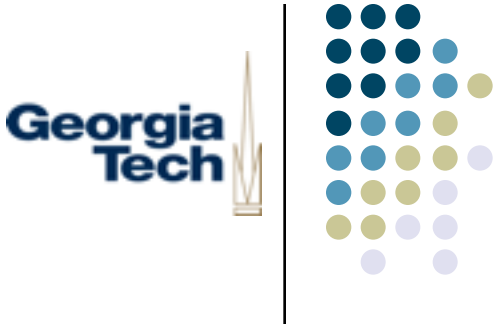
The Next Assignment: IM

- The *Know-it-All* interface
  - Extend the IM client to make you look like you are the master of all human knowledge
- Two parts for this one (actual window names may vary):
  - The Wiki Ferret window
    - Access Wikipedia to bring up encyclopedia entries for words in the chat transcript
    - "Old school" part of the assignment
    - Not a web service; just a regular old web server
    - I've provided a wrapper for talking to Wikipedia
    - You need to pull out text and display it appropriately formatted
  - The Googlemeister window
    - Find top-ranked hits for each exchange in the transcript; show the last several
    - Uses the Google Web Services API, via googlehack.jar
    - Display hits in a side panel or something, link to external Swing web view

Using Wikipedia

- wiki.py class (provided on web site)
  - w = Wiki()
  - w.fetchTopic("Jython")
- Returns a tuple: (HTTP_code, result)
- *Result* is the contents of the page, including HTTP headers
- A blank line separates the headers from the content
- Content has Wiki-style markup (square brackets) around linked items

- Word of warning: you don't want to hammer the Wikipedia server!
  - They're cautious about attacks, and will pull the plug on you
  - Try to take care to look up words once per session, rather than zillions of times per session



Example Results

HTTP/1.0 200 OK
Date: Thu, 10 Mar 2005 22:12:35 GMT
Server: Apache
X-Powered-By: PHP/4.3.10
Cache-Control: s-maxage=2678400, max-age=2678400
Last-Modified: Tue, 28 Dec 2004 20:47:50 GMT
Content-Type: text/x-wiki; charset=iso-8859-1
X-Cache: MISS from srv10.wikimedia.org
Connection: close

'''Jython''' is a version of [[Python programming language|Python]] that's written in [[Java (programming language)|Java]] and that runs in the Java environment.

Jython programs can seamlessly import and use any Java class. Except for some standard modules, Jython programs use Java classes instead of Python modules. For example, a user interface in Jython would be written with [[Swing (Java)|Swing]] or [[AWT]], rather than with Tkinter.
Jython has both an [[interpreter (computing)|interpreter]] and a [[compiler]].

Jython was formerly known as "JPython".

== External links ==
* [http://www.jython.org/ Jython Home Page]
* [http://www-106.ibm.com/developerworks/java/library/j-jython.html Charming Jython: Learn how the Java implementation of Python can aid your development efforts]
* [http://www-106.ibm.com/developerworks/library/j-alj07064/ Get to know Jython]
* [http://www-106.ibm.com/developerworks/db2/library/techarticle/dm-0404yang/index.html Learn how to write DB2 JDBC tools in Jython]

[[Category:Programming languages]]

[[de:Jython]]
[[fr:Jython]]
[[ru:Jython]]
[[zh:Jython]]

Displaying Results

- Provide some UI for looking up Wikipedia entries for words
  - Easy: double click on words in transcript area
  - Fancy: look up (long?) words as they are typed and indicate which have an entry (coloring, or other highlighting)
- Be sure to detect words with no entry
  - Returns headers only; no content
- Display results
  - Strip headers
  - Don't display internal formatting (double brackets, junk at top and bottom, etc.)
  - Several options for display: separate window, side bar, tool tip, etc.
- Up to you whether you want to display links in some special way
- Up to you whether you want to be fancy and support linking

Getting Ready to Use the Google API

- Easier: actual Web Service!
- Step 1: Download the googlehack.jar library from the website
  - Download googlehack.jar from the website
- Step 2: Optional, but recommended: Get a unique *key* from Google
  - Google's API uses a *key* that uniquely identifies you
  - http://code.google.com/apis/loader/signup.html
  - Used by Google to limit improper use (hacks)
- Step 3: Update your CLASSPATH to include googlehack.jar. NOTE: if you're using Eclipse or another tool, you may need to update CLASSPATH there too.
  - For Windows command line:
    - Control Panel -> System Properties -> Advanced -> Environment Variables
    - Edit CLASSPATH
    - Add *full path* to googlehack.jar, separated from existing stuff by a ";" (only)
  - For Macintosh command line:
    - Edit /Users/*yourname*/.profile
    - Add the following line:
    - export CLASSPATH=${CLASSPATH}:/*full*/*path*/*to*/googlehack.jar
    - Restart Terminal.app, or type "source /Users/*yourname*/.profile"

Using the Google API

- googlehack.jar is a pre-built library for talking to Google
  - No need to do crazy WSDL stuff!
- Import googlehack
- Create a GoogleHack object
- Call search("search term")
- Returned is a SearchResponse
  - Contains list of SearchResults
  - Each SearchResult contains title, URL, content, etc.
- To use your key:
  - Set the the value of the GOOGLE_KEY environment variable to your key and the library will use it
  - Windows: Control Panel > System; Mac: setenv GOOGLE_KEY *your_key_value*

```
import googlehack
import sys

class Demo:
        def __init__(self, searchterm):
                g = googlehack.GoogleHack()
                response = g.search("ferret")
                print "Server's response: ", response
                results = response.results
                for i in results:
                        print i.title + ":" + str(i.URL)

if __name__ == "__main__":
        g = Demo(sys.argv[1])
```

More Details

- Sometimes Google will report an error (if the server is busy, for example)
- You want your program to be robust to this, so catch exceptions and handle them correctly:

```
import googlehack
import sys

class Demo:
        def __init__(self, key):
                try:
                        g = googlehack.GoogleHack()
                        r = g.search("ferret")
                        results = r.results
                        for i in results:
                                print i.title + ":" + str(i.URL)
                except googlehack.SearchException, ex:
                        print "Problem contacting google: " + str(ex)

if __name__ == "__main__":
        g = Demo(sys.argv[1], sys.argv[2])
```

- Better: retry a couple of times (but not forever!) if it fails

Search API

- The *SearchResponse* object returned by search() has the following variables defined on it:
  - *responseStatus* -- this is an integer response code that indicates whether the request was successful. The number 200 will be returned if everything worked ok.
  - *responseDetails* -- if there was a problem with the request, this variable will be set to a string that contains details about what went wrong.
  - *results* -- this is a list that contains *SearchResult* objects (currently limited to 4 max)
- Each *SearchResult* contains the following variables:
  - *URL* -- The URL of this search hit
  - *title* -- The string title of the page for this hit; may contain HTML formatting
  - *titleNoFormatting* -- Page title stripped of HTML formatting
  - *content* -- A snippet of content of the page that matches the query; may contain HTML formatting.

More Details

- It may take a while for Google to return the results
  - Not a *long* time, but long enough for you to notice if your GUI is *blocking* waiting on Google to return
- Why would this happen?
  - You look up words in google in response to some action (user clicks "Send" etc.)
  - So your lookup happens in the *same thread* as your program, meaning that the program can't continue until the results are returned
- How to fix it?
  - Use another thread for google lookups
- **This is purely optional; no need to do this for the assignment**

Displaying the Results

- Take text that appears in the transcript
- Feed it into a the google search API
- Get several (3-4?) results (actual number depends on your UI)
- Display in an organized way
  - Example: time-ordered, like the transcript
- Show query, followed by results, visually distinct from one another
  - Results should show at least title and some way to click the URL
- Should display several (3-4?) sets of these, so that users can access queries for past text entries
- Options for display: side window, tool tips, hovering translucent do-dads, etc.
- Clicking URL should launch (or reuse) a Swing HTML viewer

Using Swing HTML Viewers

```
import javax.swing as swing
import java.awt as awt

content="""<HTML><BODY>
Hello there!<P>
<A HREF="http://www.cc.gatech.edu">CoC Home Page</A><P>
<A HREF="http://www.cc.gatech.edu/classes/AY2005/cs6452_spring">CS6452 Spring 05</A><P>
</BODY>
</HTML>"""

class HTMLExample:
    def __init__(self):
        self.frame = swing.JFrame("HTML Example")
        self.html = swing.JEditorPane()
        self.html.contentType = "text/html"
        self.html.editable = 0
        self.html.hyperlinkUpdate=self.followHyperlink
        self.html.preferredSize=(400, 400)
        self.html.text = content
        self.frame.contentPane.layout = awt.BorderLayout()
        self.frame.contentPane.add("Center", swing.JScrollPane(self.html))
        self.frame.pack()
        self.frame.show()

    # This callback is invoked whenever a link is clicked or moused
    # over.  The event is a HyperlinkEvent, which means that it
    # defines certain fields and operations, which we can use below
    def followHyperlink(self, event):
        if event.eventType == swing.event.HyperlinkEvent.EventType.ACTIVATED:
            self.html.setPage(event.URL)

if __name__ == "__main__":
    h = HTMLExample()
```