Web Service Architecture for Composable, Interdisciplinary Applications

Elizabeth Latronico
beth@berkeley.edu
Outline

- Ptolemy has a web server!
- Motivation
- Web service building blocks
- Example
- Limitations (“Future Work”)

Ptolemy Miniconference, November 7, 2013

Latronico, Berkeley 2
The research accessibility challenge

- Great Ph.D. student with excellent results
  - Then, the inevitable: Graduation

- Artifacts (in addition to publications)
  - Pile of code for a highly specialized purpose, with a lifespan equal to Ph.D. student’s enrollment
  - Extension points? Maintenance? Install help?

- Disadvantages
  - Good results might have low impact due to low accessibility
  - Interesting research at intersection of fields passed up
Can a web service paradigm help?

- Frame results as web services for composability
  - Use web API for accessibility with low coordination overhead
- Tap into data sources
- Wrap web API around software
- Snap together new applications
Anatomy of a web service API

- RESTful approach – REpresentational State Transfer
  - Organize system into a set of resources (can be objects or services)
  - Client-server; Server prohibited from storing client state

- Offer URL for each resource (Scaife Hall example)

- Uniform set of operations (“verbs”)
  - GET, POST, PUT, DELETE, more…
  - An individual resource may allow only some operations
  - Info may be appended to a request (e.g. form input, cookies)
Ptolemy building blocks

- Documentation! [http://ptolemy.eecs.berkeley.edu/books/Systems/](http://ptolemy.eecs.berkeley.edu/books/Systems/)
- Director – Discrete Event Director
- Attributes – WebServer, XMPPGateway (Sensor Andrew)
- Request handling – HttpActor
- Data sources – HttpGet, XMPPSource
- Software wrappers – Simulator, ModelReference
- UI – FileReader, HTMLPageAssembler
- Testing – HttpGet, HttpPost (Can test non-Ptolemy services)
Demos

- Three sample models, checked in to repository:
  - Sensor Andrew live temperature map
  - TuLiP controller synthesis
  - Building Controls Virtual Testbed / EnergyPlus building simulation

- Can imagine interesting interactions in the future!
BCVTB/EnergyPlus example

- Building temperature simulation
- User specifies cooling setpoint
- 3 requests:
  - GET main page
  - POST setpoint
  - GET chart data
Ptolemy model

This model launches a web server and provides a service available at http://localhost:8078/eplus

GET Request

GET the web page showing a temperature chart using baseline data

POST Request

Run EnergyPlus simulation with the POSTed setpoint

Output is written to pages/temperatures.csv
Execution

- **Discrete Event Director**
  - Timed model of computation
  - Run indefinitely until manual stop

- **WebServer**
  - Starts a Jetty web server when the model is run
  - Specify locations of any files to host (images, scripts…)

---

This model launches a web server and provides a service available at http://localhost:8078/eplus

**stopWhenQueueIsEmpty**: false

**resourceLocation**: $PTII/org/pptolemy/ptango/demo/TemperatureSimulation/pages
Handling an Http GET request

1) Http GET request arrives, e.g.
   
   GET http://server:8078/eplus  
   
   Matching HttpActor fires.

2) Token is produced on “getRequestURI” port. FileReader fires.

3) FileReader outputs file contents (here, a web page).
   
   MicrostepDelay advances time, so response occurs after request.

4) HttpActor fires again, consuming token on “response” input port.

---

GET the web page showing a temperature chart using baseline data
Handling an Http POST request

1) Http POST request arrives, e.g.
   \texttt{POST http://server:8078/eplus?setpoint=24}  \text{Matching HttpActor fires.}

2) Record token with setpoint is produced on “postParameters” port.

   Saves results to file. Produces token on “done” output port.

4) HttpActor fires, consuming “response” input port token.

(Not shown) Client issues GET request for data
   \texttt{GET http://server:8078/eplus/pages/temperatures.csv}
Properties

- Modular:
  Can divide problem into a set of independent model loops

- Separation of concerns:
  Can separate execution control and data retrieval

- Quick assembly:
  Relatively fast to put together (not counting custom UI 😊)

- Low coordination overhead:
  Usually, integrated resource not modified much (first instance setup can take effort on Ptolemy side)
Limitations (i.e. “Future Work”)

- **Server**
  - Currently: Single machine
  - Want: Something easy for everyone to share (Cloud?)

- **Security**
  - Currently: Supports some basic access control
  - Want: Everything from “Attack Modeling in Ptolemy” (thanks Armin!)

- **Graceful fault handling**
  - Currently: A Ptolemy exception will crash whole server
  - Want: Contain crashed services; retry; restart
Limitations (2)

- **App management**
  - Currently: Stop, start apps through GUI/command line
  - Want: App manager with web interface

- **Many additional topics**
  - Multiple client support for publish-subscribe
  - Support for other REST operations and content types
  - Widget library for web page construction
  - ...

- Your request here!
Ideas?

- Nifty applications? Composing services?
- What are most important infrastructure features to develop next?