ASYNCHRONOUS AJAX FOR REVOLUTIONARY WEB APPLICATIONS

Jeanfrancois Arcand
Ted Goddard, Ph.D.

TS-5250

Java

Sun microsystems

Join the Asynchronous Web Revolution!

Easily develop multi-user collaboration features in NetBeans with Ajax Push and Comet using Dojo, DWR, or ICEfaces.

Deploy and scale on Jetty, Tomcat, or GlassFish™ project.

Agenda

- Web2.0™
- Multi-user Ajax Demo
- Asynchronous HTTP on the Wire
- Asynchronous HTTP and the Server
- Developing Asynchronous Applications
- Conclusion

What sort of revolution?

"And yet it moves."

American Revolution

Dump everything in the bay

French Revolution

Storm the Bastille

Scientific Revolution

Experimentation and Reason

Web 2.0

A Web by the people, for the people.

- Documents on the web increasingly generated by users

- Out of the Information Age, into the Participation Age
- As a whole, the World Wide Web is a collaborative environment, but individual pages are only weakly so
- Are web user interfaces becoming more powerful?
- Is the user an HTTP client?

Ajax

Ajax is a state of mind.

- It was AJAX (Asynchronous JavaScript™ Technology with XML)
  - or Asynchronous JavaScript technology with XMLHttpRequest
  - now it's Ajax (not an acronym) because many different techniques satisfied the same goals
  - coined by Jesse James Garrett in 2005 to sell an insurance company on re-writing all their software
- Is the web defined by the W3C or by browser implementers? (Ajax does not exist in W3C universe yet.)
- Ajax decouples user interface from network protocol
- Ajax is the leading edge of the user interface possible with current popular browsers
- The user experience is important

The Asynchronous Web Revolution

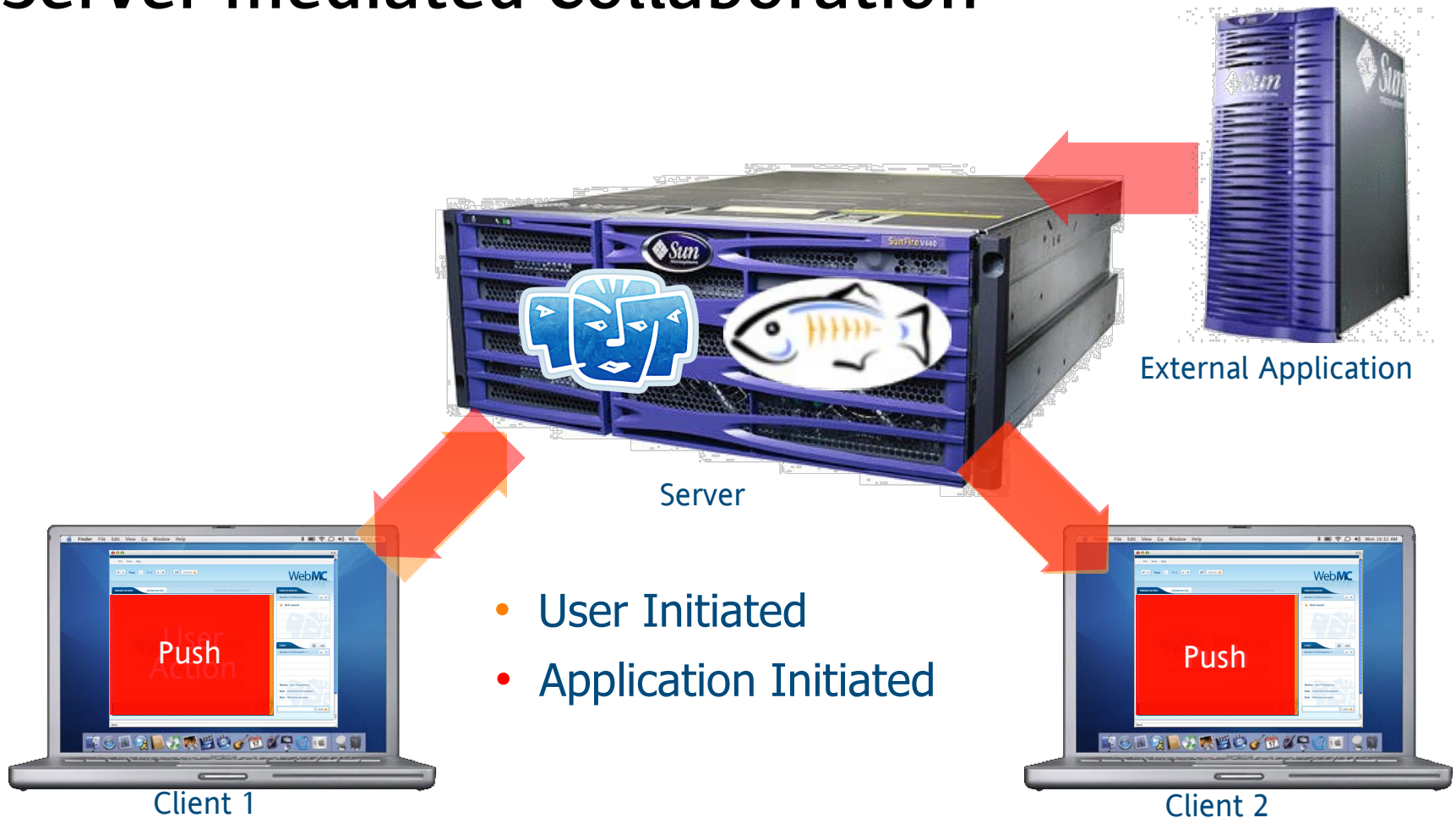
The Web enters the Participation Age.

- Ajax is still typically synchronous with user events
- Full asynchrony has updates pushed from server any time

- Update pages after they load
- Send users notifications
- Allow users to communicate and collaborate within the web application

- Called “Ajax Push”, “Comet”, or “Reverse Ajax”
  - This is the full realization of Ajax, now fully asynchronous

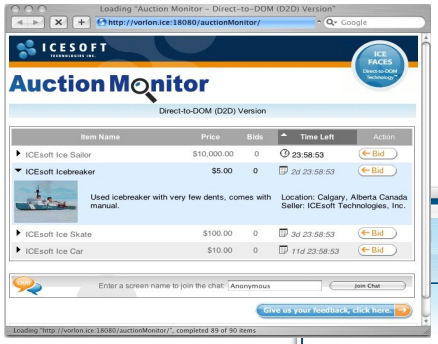
Server-mediated Collaboration



- User Initiated
- Application Initiated

Applications in the Participation Age

Application-mediated communication.



- Distance learning
- Collaborative authoring
- Auctions
- Shared WebDAV filesystem
- Blogging and reader comments
- SIP-coordinated mobile applications
- Hybrid chat/email/discussion forums
- Customer assistance on sales/support pages
- Multi-step business process made collaborative
- Shared trip planner or restaurant selector with maps
- Shared calendar, “to do” list, project plan
- Games

Agenda

- Web 2.0
- Multi-user Ajax Demo
- Asynchronous HTTP on the Wire
- Asynchronous HTTP and the Server
- Developing Asynchronous Applications
- Conclusion

Asynchronous Ajax Demo

GlassFish project/Project Grizzly with ICEfaces WebMC

DEMO

http://webmc.icefaces.org/javaone

Agenda

- Web 2.0
- Multi-user Ajax Demo
- Asynchronous HTTP on the Wire
- Asynchronous HTTP and the Server
- Developing Asynchronous Applications
- Conclusion

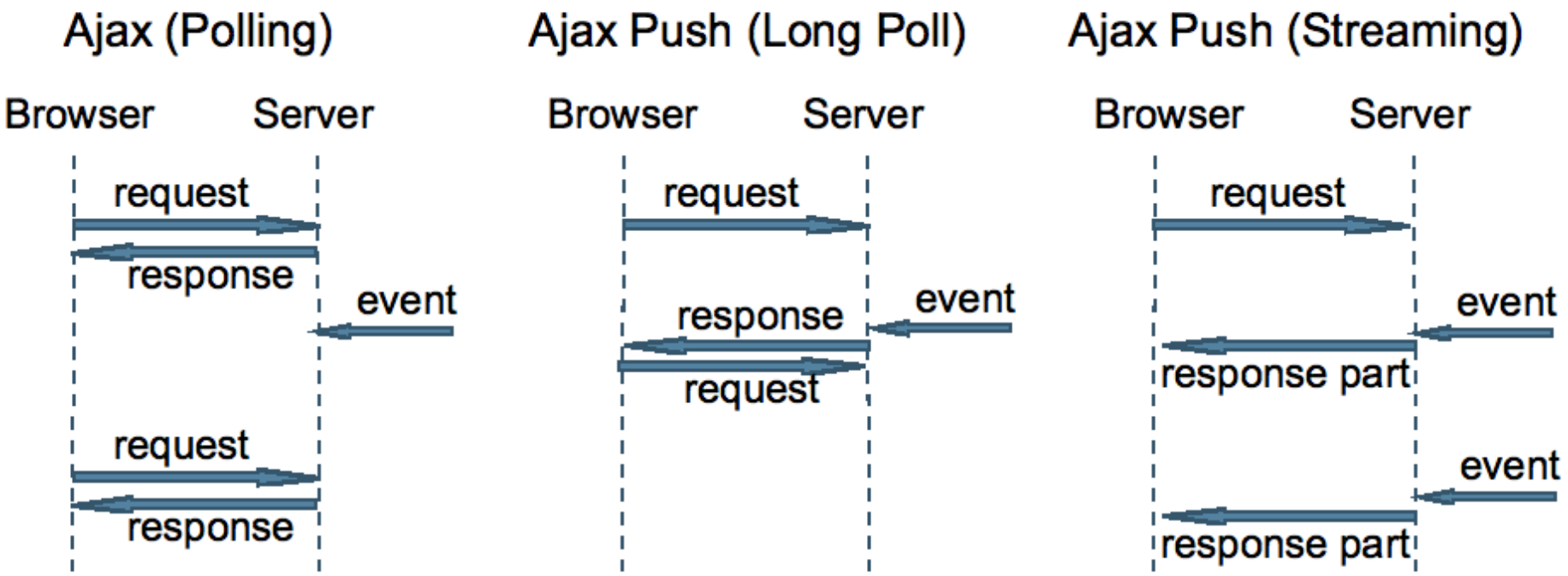
What is Ajax Push, exactly?

Responsive, low-latency interaction for the web.

- highly responsive, event driven browser applications
  - Keep clients up-to-date with data arriving or changing on the server, without frequent polling
- Pros
  - Lower latency, not dependent on polling frequency
  - Server and network do not have to deal with frequent polling requests to check for updates
- Example Applications
  - GMail and GTalk
  - Meebo
  - Many more ...
  - 4homemedia.com (using GlassFish project's Comet)
  - JotLive
  - KnowNow

Ajax Poll vs Ajax Push

Bending the rules of HTTP.



Ajax Poll vs Ajax Push

Bending the rules of HTTP.

- Poll:
  - Send a request to the server every X seconds.
  - The response is “empty” if there is no update.
- Long Poll:
  - Send a request to the server, wait for an event to happen, then send the response.
  - The response is never empty.
  - HTTP specification satisfied: indistinguishable from “slow” server
- Http Streaming:
  - Send a request, wait for events, stream multi-part/chunked response, and then wait for the events.
  - The response is continually appended to.

How Push works

Keep an open connection.

- Deliver data over a previously opened connection
- Always “keep a connection open”
  - do not respond to the initiating request until event occurs
- Streaming is an option
  - send response in multiple parts without closing the connection in between

HTTP Polling

Regularly checking for updates.

```
GET /chatLog HTTP/1.1
Accept: */*
Connection: keep-alive

<message>One</message>
```

- Uses the HTTP protocol in a standard way, but requests are regularly invoked

```
setTimeout('poll()', 10000);
```

Asynchronous HTTP Streaming

The long response.

```
GET /chatLog HTTP/1.1
Accept: */*
Connection: keep-alive

<message>One</message>
<message>Two</message>
<message>Three</message>
<message>Four</message>
```

- Parse most recent message in JavaScript programming language (not shown here)
- The original 1999 “Push” technique (Netscape 1.1)

Bayeux/Cometd

JSON Pub/Sub.

```
[
  {
    "channel": "/some/name",
    "clientId": "83js73jsh29sjd92",
    "data": { "myapp" : "specific data", value: 100 }
  }
]
```

- JSON Messages are published on specified channels
- Channel operations: connect, subscribe, unsubscribe, etc.
- Multiple transports: polling, long-polling, iframe, flash
- Server implementations: Perl, Python, Java™ programming language
- Server-side reflector with no server-side application possible

Ajax Push

HTTP message flow inversion.

```
GET /auctionMonitor/block/receive-updates?icefacesID=1209765435 HTTP/1.1
Accept: */*
Cookie: JSESSIONID=75CF2BF3E03F0F9C6D2E8EFE1A6884F4
Connection: keep-alive
Host: vorlon.ice:18080
```


```
HTTP/1.1 200 OK
Content-Type: text/xml;charset=UTF-8
Content-Length: 180
Date: Thu, 27 Apr 2006 16:45:25 GMT
Server: Apache-Coyote/1.1

<updates>
 <update address="_id0:_id5:0:chatText">
  <span id="_id0:_id5:0:chatText">Howdy</span>
 </update>
</updates>
```

Agenda

- Web 2.0
- Multi-user Ajax Demo
- Asynchronous HTTP on the Wire
- Asynchronous HTTP and the Server
- Developing Asynchronous Applications
- Conclusion

Architecture Challenge

Can Push scale?

- A blocking, synchronous technology will result in a blocked thread for each open connection that is “waiting”
- Every blocked thread will consume memory
- This lowers scalability and can affect performance
- To get the Java Virtual Machine (JVM™) to scale to 10,000 threads and up needs specific tuning and is not an efficient way of solving this

- Servlets 2.5 are an example of blocking, synchronous technology

Servlet Thread Catastrophe

Strangled by a thread for every client.

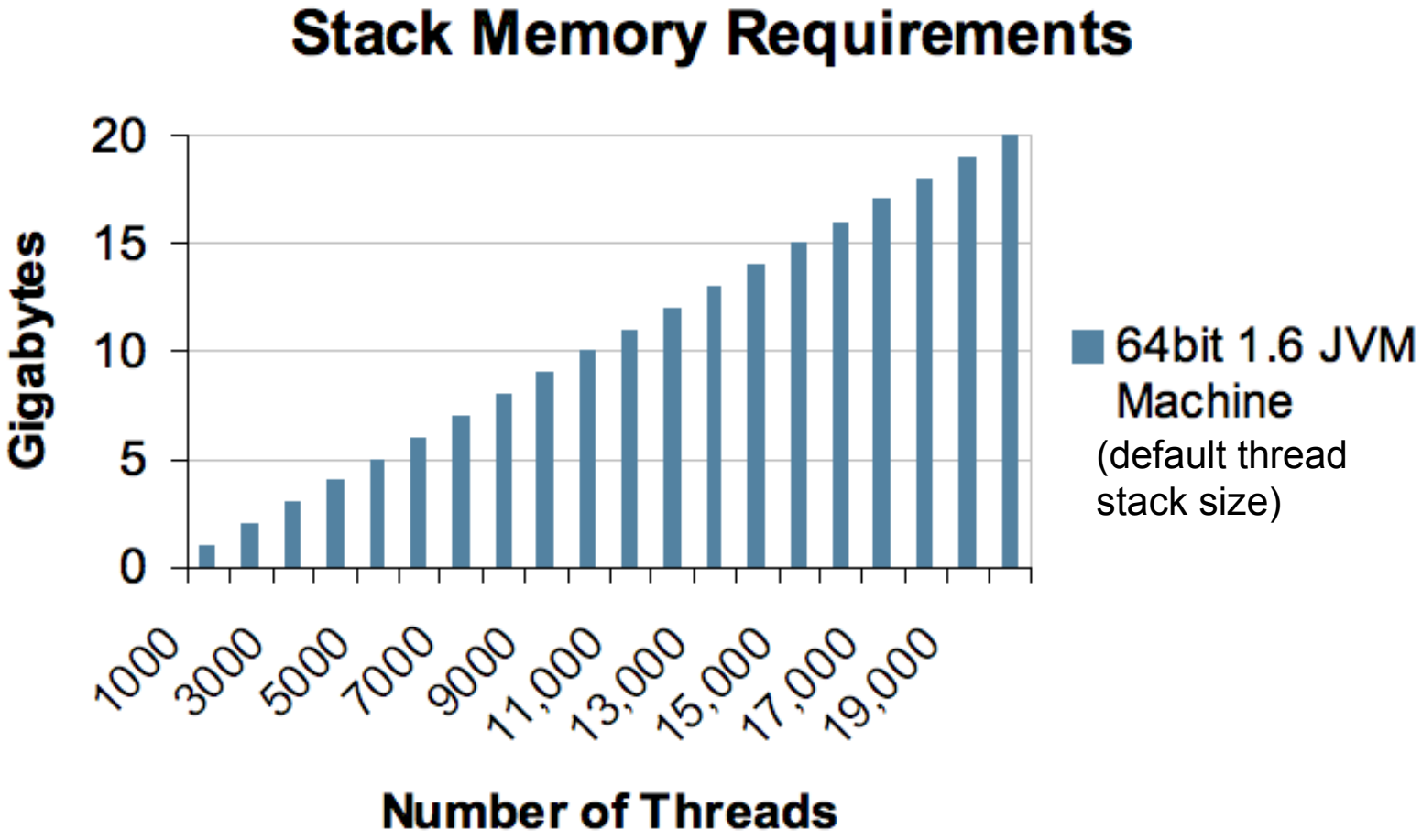
GET /updates HTTP/1.1
Connection: keep-alive

GET /updates HTTP/1.1
Connection: keep-alive

GET /updates HTTP/1.1
Connection: keep-alive

Architecture Challenges

The serious effect of blocking threads.



Technology Solution

NIO avoids blocked threads.

- Use new I/O (NIO) non-blocking sockets to avoid blocking a thread per connection
- Use technology that supports asynchronous request processing
  - Release the original request thread while waiting for an event
  - May process the event/response on another thread than the original request
- Advantages
  - Number of clients is primarily limited by the number of open sockets a platform can support
  - Could have all clients (e.g. 10'000) "waiting" without any threads processing or blocked

Server-side Ajax Push: Server considerations

Not all servers are non-blocking.

- To handle the “wait” for an event in Ajax Push, choose a Web Server / language that does not have to block
- Some inherently do not block
  - Traditional Web Servers (like Apache, Tomcat, and GlassFish v1 project) are blocking.
- Continuations are another option
  - E.g. JavaScript™ technology “continuation” in the “Rhino” open source implementation

Server-side Ajax Push: Required functionality

A spectrum of asynchronicity.

- Asynchronous Content Handlers
  - Asynchronous read and write
- Suspendable Requests
  - Suspend/resume requests/responses
- Support Delivery Guarantee mechanism
  - Push data from one connection to another
  - Ability to aggregate/filter/transform data before the push operation

Server-side Ajax Push: Who supports what

The asynchronicity matrix.

Container	Asynchronous IO	Suspendible Request/ Response	Delivery Guarantee
Jetty		X	
Tomcat	X	X	
GlassFish	X	X	X
Resin		X	
WebLogic		X	

Resin

Suspend, Wake, and Resume with Resin

```
public class CometServlet extends GenericCometServlet {
    public boolean service(ServletRequest request,
                           ServletResponse response,
                           CometController cometController)

      ...
      return true;
    }
    public boolean resume(ServletRequest request,
                          ServletResponse response,
                          CometController cometController)
      PrintWriter out = res.getWriter();
      out.write(message);
      return false;
}
```

Suspend

Resume

Asynchronously and elsewhere in the application …

```
message.setValue("Howdy");
cometController.wake();
```

Jetty

service() will resume shortly.

```
import org.mortbay.util.ajax.Continuation;

service(request, response) {
    Continuation continuation = ContinuationSupport
        .getContinuation(request, this);
    ...
    continuation.suspend();
    response.getWriter().write(message);
}
```

Asynchronously and elsewhere in the application ...

```
message.setValue("Howdy");
continuation.resume();
```

WebLogic

doRequest() and doResponse() separated by notify().

```
import weblogic.servlet.http.AbstractAsyncServlet;
import weblogic.servlet.http.RequestResponseKey;

class Async extends AbstractAsyncServlet {

boolean doRequest(RequestResponseKey rrk) {
    ... = rrk;
    return false;
}

void doResponse(RequestResponseKey rrk, Object message) {
    rrk.getResponse().getWriter.write(message);
}
```

Asynchronously and elsewhere in the application …

```
message.setValue("Howdy");
AbstractAsyncServlet.notify(rrk, message);
```

Tomcat 6

Eventful Comet.

```
import org.apache.catalina.CometProcessor;

public class Processor implements CometProcessor {

public void event(CometEvent event)  {
    request = event.getHttpServletRequest();
    response = event.getHttpServletResponse();

    if (event.getEventType() == EventType.BEGIN) { ...
    if (event.getEventType() == EventType.READ)  { ...
    if (event.getEventType() == EventType.END)   { ...
    if (event.getEventType() == EventType.ERROR) { ...
}
```

Asynchronously and elsewhere in the application …

```
message.setValue("Howdy");
response.getWriter().write(message);
event.close();
```

GlassFish Project
Suspend with Project Grizzly.

```
CometContext context =
    CometEngine.getEngine().register(contextPath);
context.setExpirationDelay(20 * 1000);

SuspendableHandler handler = new SuspendableHandler();
handler.attach(response);
cometContext.addCometHandler(handler);

class SuspendableHandler implements CometHandler  {

    public void onEvent(CometEvent event) {
       response.getWriter().println(event.attachment());
       cometContext.resumeCometHandler(this);
   }
```

Asynchronously and elsewhere in the application …

```
message.setValue("Howdy");
cometContext.notify(message);
```

Grizzlets
Bite-sized Project Grizzly

```
public void onPush(AsyncConnection asyncConnection) {
    GrizzletRequest req = ac.getRequest();
    GrizzletResponse res = ac.getResponse();
    if (asyncConnection.isResuming()) {
        res.write("Why Servlet? POJO much better!<br/>");
        res.write("</body></html>");
        res.flush();
        res.finish();
    } else if (asyncConnection.hasPushEvent()) {
        res.write(ac.getPushEvent().toString());
        res.flush();
    }
}
```

Asynchronously and elsewhere in the application …

```
message.setValue(“Howdy”);
asyncConnection.push(message.toString());
```

Asynchronous Ajax Demo
Grizzlet with Project jMaki

http://grizzly.dev.java.net

Servlet 3.0

Future Asynchronous Standard.

- Defined by JSR-315 Expert Group
- DWR, Jetty, Tomcat, GlassFish project, and ICEfaces participants
- Standard asynchronous processing API being defined
  - Asynchronous I/O
  - Suspendible requests
  - Delivery guarantee not included

- Will improve portability of DWR, Cometd, and ICEfaces
- (But unless you write Servlets today, this API will be hidden by your chosen Ajax framework.)

Agenda

- Web 2.0
- Multi-user Ajax Demo
- Asynchronous HTTP on the Wire
- Asynchronous HTTP and the Server
- Developing Asynchronous Applications
- Conclusion

JavaScript Programming Language Polling

Are we there yet? Are we there yet? Are we there yet? ...

```
function poll()  {
  setTimeout('poll()', 10000);
  req = new XMLHttpRequest();
  req.onreadystatechange = update();
  req.open("POST", "http://server/getMessage.jsp");
}

function update() {
  chatLog.innerHTML = req.responseText;
}

poll();
```

Cometd

Distributed, loosely coupled, scripting

JavaScript programming language

```
function update(message) {
  chatLog.innerHTML = message.data.value;
}
...
  cometd.subscribe("chat", remoteTopics, "update")
  cometd.publish("chat", message)
```

Java programming language

```
import dojox.cometd.*;

Channel channel = Bayeux.getChannel("chat", create);
channel.subscribe(client);
```

Asynchronously and elsewhere in the application …

```
message.setValue("Howdy");
channel.publish(client, message, "chat text");
```

DWR

JavaScript programming language RPC

```
import org.directwebremoting.proxy.dwr.Util;

scriptSessions =
    webContext.getScriptSessionsByPage(currentPage);
    util = new Util(scriptSessions);
```

To “Reverse Ajax” and invoke arbitrary JavaScript technology:

```
util.addScript(ScriptBuffer script);
```

Asynchronously and elsewhere in the application …

```
util.setValue(“form:chat:_id3”, “Howdy”);
```

ICEfaces

Preserve MVC with Transparent Ajax.

PageBean.java

```
public class PageBean  {
    String text;

    public String getText()  {
        return text;
    }

    public void setText(String text)  {
                this.text = text;
    }

}
```

Presentation Model

Page.xhtml

```
<f:view
        xmlns:f="http://java.sun.com/jsf/core"
        xmlns:h="http://java.sun.com/jsf/html" >
  <html>
    <body>
      <h:form>
        <h:inputText value="#{pageBean.text}" />
      </h:form>
    </body>
  </html>
</f:view>
```

Declarative User Interface

A language for Ajax Push that preserves Designer and Developer roles

ICEfaces

High level push.

```
import org.icefaces.application.SessionRenderer;

SessionRenderer.render(SessionRenderer.ALL_SESSIONS);
```

Or to keep track of groups of users:

```
SessionRenderer.addCurrentSession("chat");
```

Asynchronously and elsewhere in the application …

```
message.setValue("Howdy");
SessionRenderer.render("chat");
```

The JavaServer™ Faces (JSF) platform lifecycle runs and each user's page is updated from the component tree.

Agenda

- Web 2.0
- Multi-user Ajax Demo
- Asynchronous HTTP on the Wire
- Asynchronous HTTP and the Server
- Developing Asynchronous Applications
- Conclusion

Summary

The Asynchronous Web Revolution is Now

- The Asynchronous Web will revolutionize human interaction
- Push can scale with Asynchronous Request Processing
- With ICEfaces, GlassFish project, and Project Grizzly, the revolution begins with your applications today
- Get ready for Servlet 3.0

For More Information

- TS-6482: Ajax and JavaServer™ Faces Technology: Wed 2:50
- BOF-5661: Comet: The Rise of Interactive Web: Wed 6:30
- BOF-4922: Using Google Web Tookit and Comet: Wed 7:30
- BOF-6584: Using Comet to Create a Web Game: Wed 8:30

- TS-5415: Java™ Servlet 3.0 API: Thu 10:50
- BOF-5495: Untangling the Asynchronous Web: Thu 8:30

- TS-4883: Java™ NIO Technology w/ Grizzly Framework: Fri 1:30

THANK YOU

Asynchronous Ajax for Revolutionary Web Applications

Jeanfrancois Arcand
Ted Goddard, Ph.D.
TS-5250