Building event-driven user interfaces with Dojo

May 12, 2006

Dylan Schiemann, Renkoo, Inc.,

dylan.io/tae/eventui.pdf

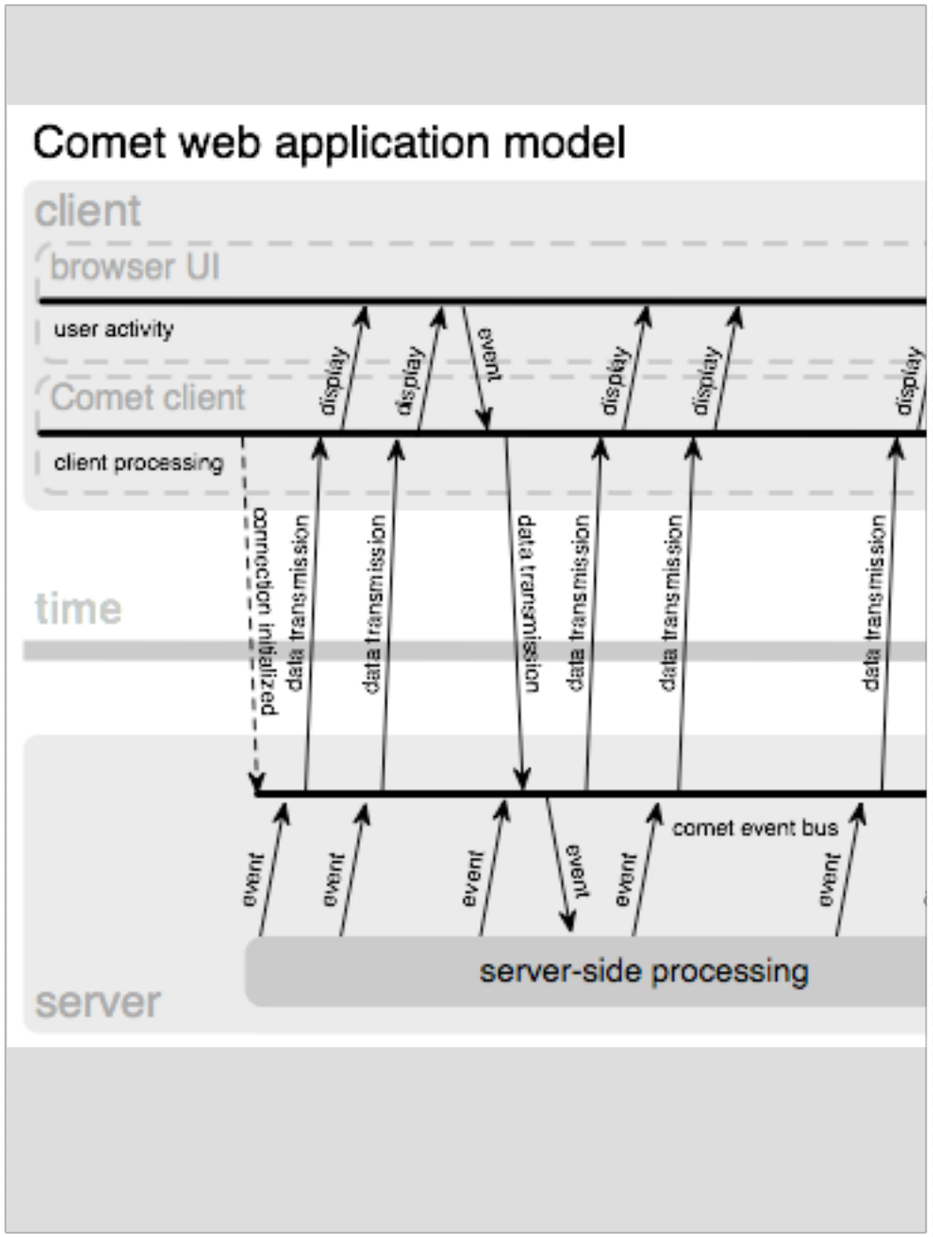
dojo the javascript toolkit

Examples

- Renkoo (demo)
- JotLive
- IM apps
  - Instaroom
  - Meebo
  - GMail Chat
  - 3Bubbles
- GreenPlum MPP Monitor (demo)

Ajax and Comet

- not just for data!
- send events to and from the server
  - XMLHttpRequest (Ajax)
  - IFrame + HTTP 1.1 chunked encoding (comet)
  - Other remote scripting techniques
- JavaScript/JSON, XML, etc.
- dojo.io
- http://alex.dojotoolkit.org/?p=545



4

Event-driven UI

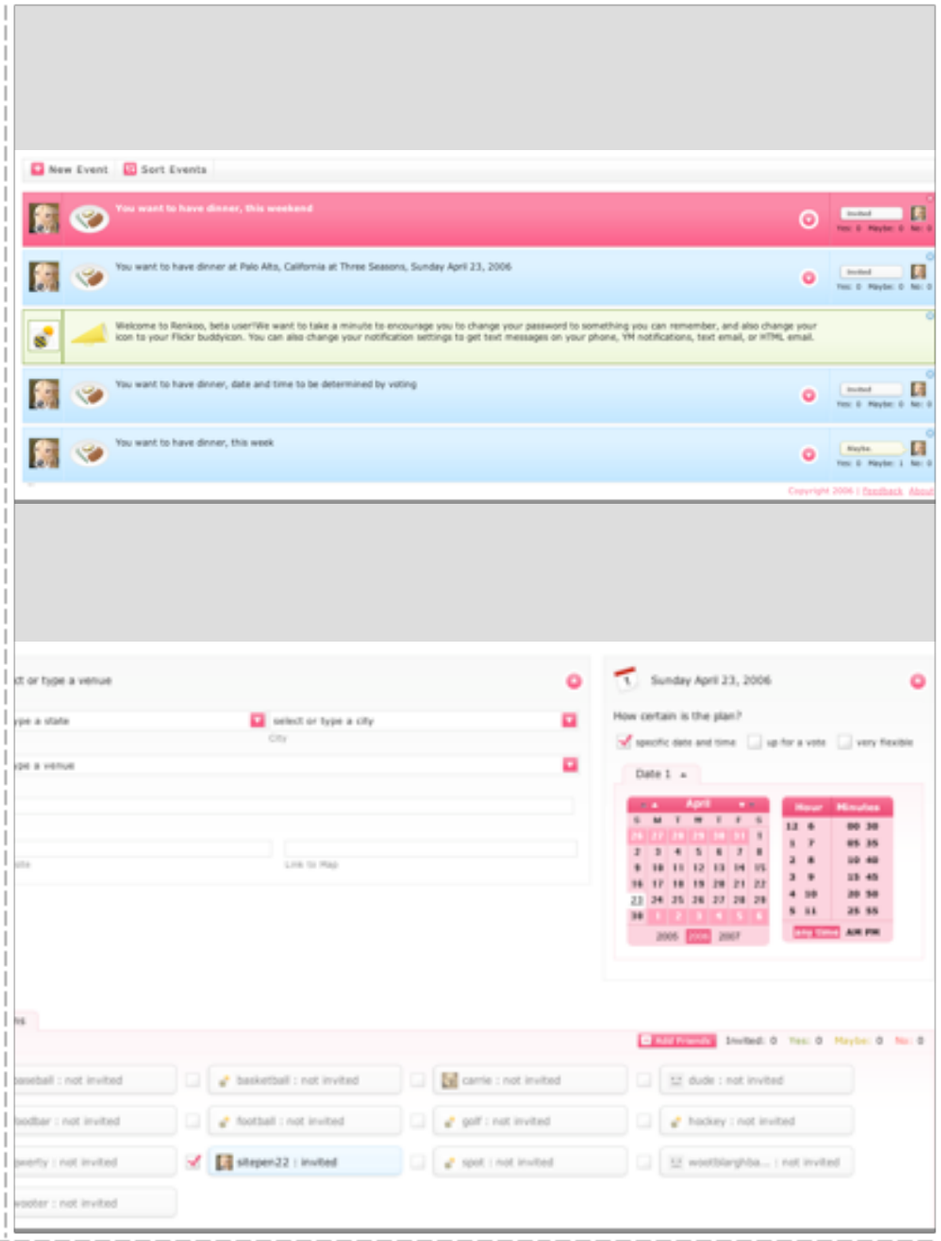
Implications

Possibilities

Best Practices

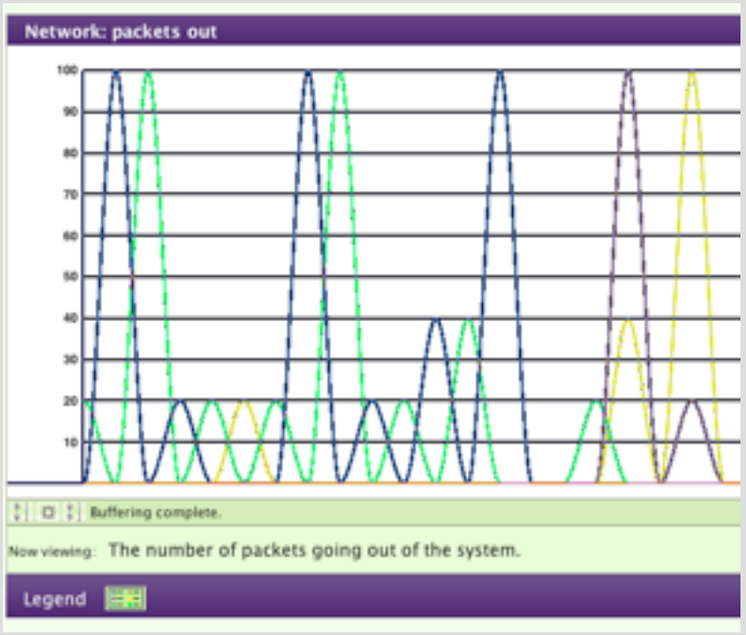
Implications

- Uniquely identifiable
- Self-assembly
- Synchronization
- Code dependencies



Problem: Uniquely identifiable

- HTML, SVG id must be unique?
  - Difficult to include two of same “type”
  - document.getElementById is efficient
- Interrelated nodes?
  - Nodes are often interdependent
- Code from different sources?
  - Potential for namespace collisions



Solution: Dojo widget system

- Create “widgets” from markup, css, JS
- Cleanly define relationships
- Instantiate from markup or script

```
<div dojoType=”datepicker”></div>

<div class=”datepicker”></div>

dojo.widget.createWidget("DatePicker", {foo:
"bar"}, node);
```

Date Picker Code Example

More than one way to create and extend widgets

```
dojo.provide("dojo.widget.html.DatePicker");
dojo.require("dojo.widget.*");
dojo.require("dojo.widget.HtmlWidget");
dojo.require("dojo.widget.DatePicker");
dojo.require("dojo.event.*");
dojo.require("dojo.html");
dojo.require("dojo.date");
dojo.widget.html.DatePicker = function(){
  dojo.widget.DatePicker.call(this);
  dojo.widget.HtmlWidget.call(this);
  var _this = this;
  this.today = "";
  this.date = "";
  this.storedDate = "";
  this.currentDate = {};
  this.firstSaturday = {};
  this.classNames = {
    previous: "previousMonth",
    current: "currentMonth",
    next: "nextMonth",
    currentDate: "currentDate",
    selectedDate: "selectedItem"
  };
  this.templatePath =  dojo.uri.dojoUri(
  "src/widget/templates/HtmlDatePicker.html");
  this.templateCssPath = dojo.uri.dojoUri(
  "src/widget/templates/HtmlDatePicker.css");
  this.fillInTemplate = function(){
    this.initData();
    this.initUI();
  }
```

```
  this.initData = function() {
    this.today = new Date();
    if(this.storedDate && (this.storedDate.split("-").length > 2)) {
      this.date = dojo.widget.DatePicker.util.fromRfcDate(this.storedDate);
    } else {
      this.date = this.today;
    }
    this.today.setHours(0);
    this.date.setHours(0);
    var month = this.date.getMonth();
    var tempSaturday =
    dojo.widget.DatePicker.util.initFirstSaturday(this.date.getMonth().toString(),
    this.date.getFullYear());
    this.firstSaturday.year = tempSaturday.year;
    this.firstSaturday.month = tempSaturday.month;
    this.firstSaturday.date = tempSaturday.date;
  }
  this.setDate = function(rfcDate) {
    this.storedDate = rfcDate;
  }
  this.initUI = function() {
    this.selectedIsUsed = false;
    this.currentIsUsed = false;
    var currentClassName = "";
    var previousDate = new Date();
    var calendarNodes =
    this.calendarDatesContainerNode.getElementsByTagName("td");
    var currentCalendarNode;
    // set hours of date such that there is no chance of rounding error due to
    // time change in local time zones.
    previousDate.setHours(8);
    var nextDate = new Date(this.firstSaturday.year, this.firstSaturday.month,
    this.firstSaturday.date, 8);
```

Date Picker Template

- http://trac.dojotoolkit.org/browser/trunk/src/widget/templates/HtmlDatePicker.html
- http://trac.dojotoolkit.org/browser/trunk/src/widget/templates/HtmlDatePicker.css

```
<div class="datePickerContainer" dojoAttachPoint="datePickerContainerNode">.
  <table cellspacing="0" cellpadding="0" class="calendarContainer">.
    <thead>.
      <tr>.
        <td class="monthWrapper" valign="top">.
          <table class="monthContainer" cellspacing="0" cellpadding="0"
          border="0">.
            <tr>.
              <td class="monthCurve monthCurveTL" valign="top"></td>.
              <td class="monthLabelContainer" valign="top">.
                <span dojoAttachPoint="increaseWeekNode" .
                  dojoAttachEvent="onClick: onIncrementWeek;" .
                  class="incrementControl increase">.
                  <img
                  src="${dojoRoot}src/widget/templates/images/incrementMonth.png"
                  alt="&darr;" style="width:7px;height:5px;" />.
                </span>.
                <span .
                  dojoAttachPoint="increaseMonthNode" .
                  dojoAttachEvent="onClick: onIncrementMonth;"
                  class="incrementControl increase">.
                  <img
                  src="${dojoRoot}src/widget/templates/images/incrementMonth.png"
                    alt="&darr;"  dojoAttachPoint="incrementMonthImageNode">.
                </span>.
                <span .
                  dojoAttachPoint="decreaseWeekNode" .
                  dojoAttachEvent="onClick: onIncrementWeek;" .
                  class="incrementControl decrease">.
```

HTML, SVG id must be unique?

- Solution: let Dojo widgets manage ids for you
  - Each widget instance gets a unique id reference
  - The widget system has convenience methods for getting widgets
    - by widgetId, by reference, by type, or by filter object
    - by filter object allows you to key off of other attributes
  - <widgetInstanceReference>.domNode points to the containing node
  - Widget system won't allow adding two widgets with the same id

Inter-related nodes?

- DojoAttachPoint
  - Attribute in template markup

    ... foo text ...
  - Creates reference in scope of the widget instance

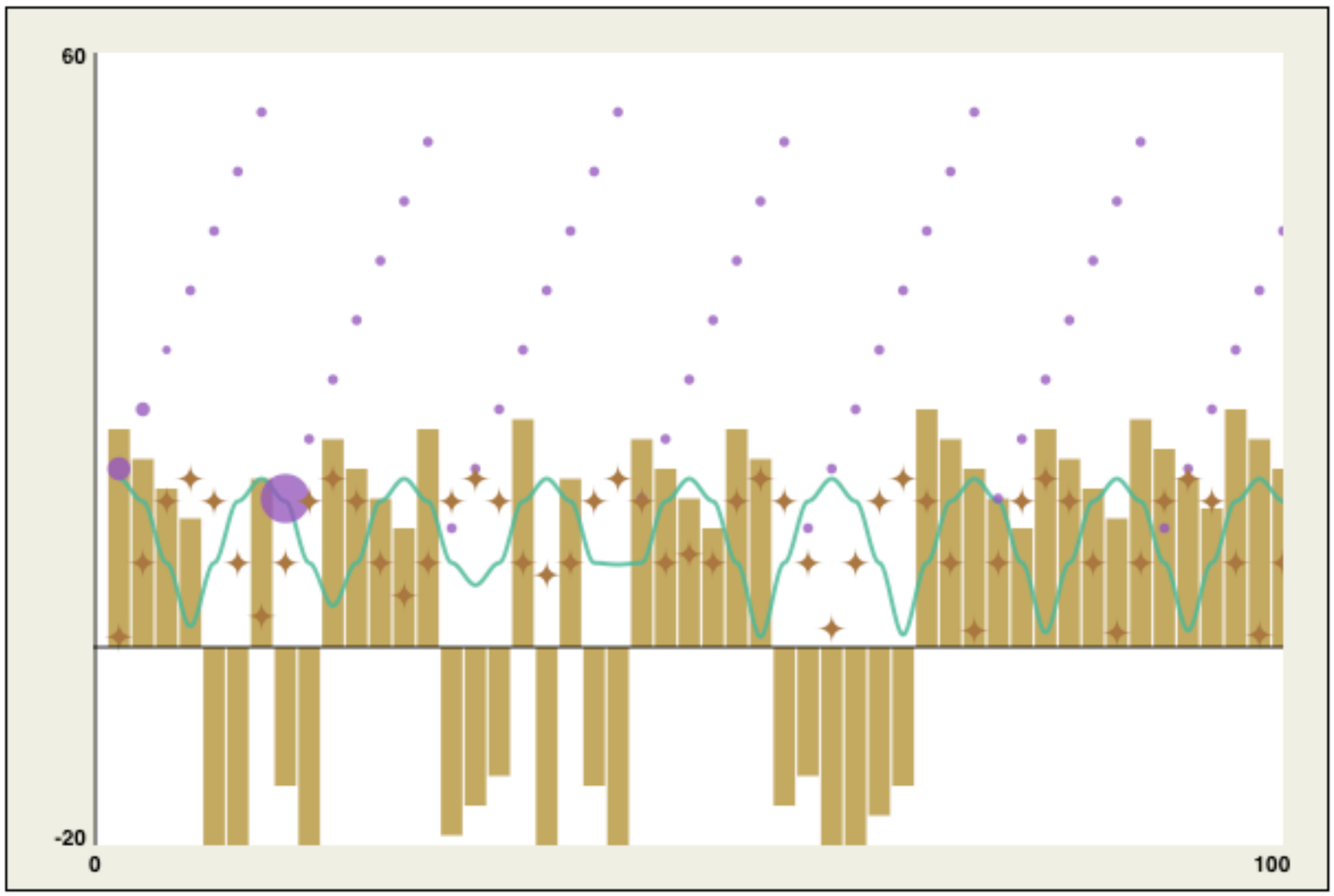
    this.fooNode.innerHTML = “bar text”;
- Decoupled markup and logic
  - Replace template html and css, but preserve the dojoAttachPoints
  - Allows radical changes to the widget appearance with little JS changes

Code from different sources?

- Don’t pollute the global namespace, which is way too easy to do
- Don’t pollute the global objects by extending object, array, etc.
- Dojo namespace, package systems allow naming, dependencies
  - dojo.provide("dojo.widget.html.DatePicker");
  - dojo.require("dojo.date");
- MochiKit supports this style of namespacing (as could anyone else)
- Fairly easy to create your own namespaces

What about SVG, canvas?

- SVG, VML, Canvas
- Mixed-mode only
- Charts
- Other widgets soon
- Drawing API



Problem: Self-assembly

- Adding/removing interface elements
- Wiring up interactions
- Populating with preliminary data

Solution: Dojo event system

- DOM Events are just the first step
- Connect any two methods together
- AOP style advice
- Define handlers in template markup

```
<div dojoAttachEvent=”onClick: onFoo;”>
```

- ... or in script

```
dojo.event.connect(srcObj, “srcFuncName”,
tgtObj, “tgtFuncName”);
```

Adding/Removing elements

- Solution: instantiate or remove widget instances from script

```
dojo.event.connect(this, “onEvent”, this, “onCreate”);

    // in the scope of an event queue

this.onCreate = function(evt) {

    // do some condition checking

    dojo.widget.createWidget("DatePicker", {foo: evt.prop1}, this.insertNode);

    // do more stuff

}
```

Wiring up interactions

- Solution: dojoAttachEvent in template markup

```
<div dojoAttachPoint=”fooNode”
    dojoAttachEvent=”onClick: onFoo;”>foo</div>
this.onFoo = function(evt) {
    dojo.debug(evt.target.innerHTML);
}
```

- Solution:

```
dojo.event.connect(this.fooNode, “onclick”, this, “onFoo”);
```

Populating with data

- Solution: dojo.io.bind call to get data as needed

```
// after widget is created, in scope of the widget instance
this.fillInTemplate = function() {
    this.getData();}
this.getData = function() {
    dojo.io.bind({url: “http://dylan.io/getData.php”,
        mimetype: “text/javascript”, content{param1: “param1Value”},
        load: function(type, evaldObj) {this.updateData(evaldObj);}});}
```

AOP style advice

- Interject and interrupt the call order and arguments
  - after advice (default for connect)
  - before advice (target function called before src function)
  - before-around and after-around advice
    - manipulate the inputs and outputs of the src function
      - useful for matching call structures, etc.)
- dojo.event.connect(”before”, this, “onFoo”, this, “onBar”);
- More details at http://dojotoolkit.org/docs/dojo_event_system.html

Problem: Synchronization

- Keep UI elements in sync?
- Keep data up to date?
- Satisfy dependency chains?

Solution: Dojo event topics

- Simple API to provide a topic paradigm
- publish

```
dojo.event.topic.publish(”/updateDate”,
    filterObject);
```

- subscribe

```
dojo.event.topic.subscribe(”/updateDate”,
    listeningObj, “onUpdateDate”);
```

Solution: Dojo event topics

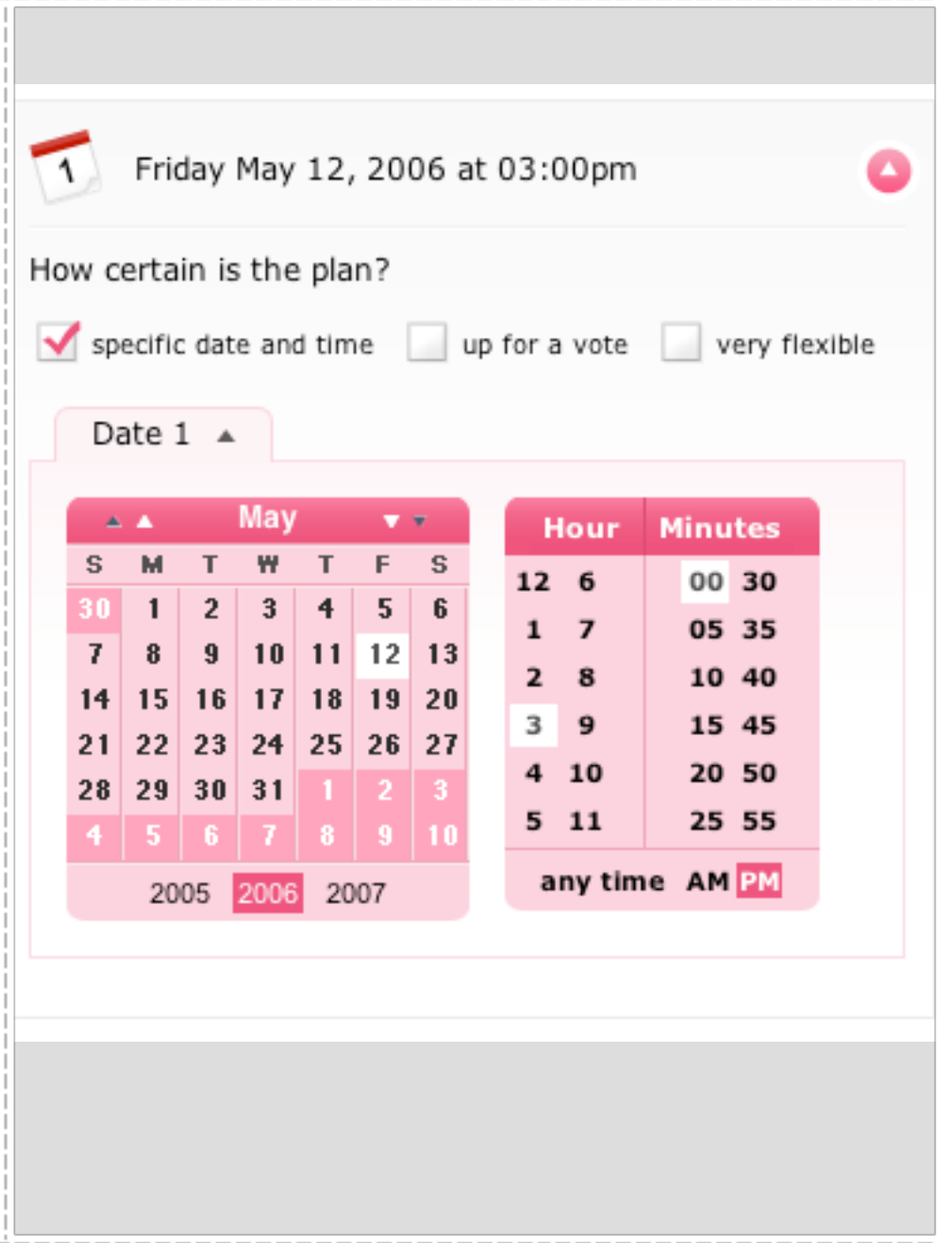
- Less dojo.event.connect calls
- Optional publisher registration
- Easy to bolt on to existing code...
  - (also true of dojo.event.connect)
- Ideal for keeping UI pieces in sync
  - a broadcaster/listener model

Keep UI elements in sync

- Each UI element subscribes to topics
- Tip: pair foo and onFoo methods
  - use one to listen, one to call in scope
  - useful for preventing circular pub/sub
  - alternatively, use a param in the filterObj

Keep data in sync

- Again, topics work great for notifying
- Data providers
  - provide a single interface to get data
  - read/write to a local object
  - use that object to pub/sub to the server
  - global, and local scope objects
    - useful when you have “related” widgets
- Overriding
  - prevent stored data from overriding user data



Satisfy Dependency Chains

- App state results in pieces being there or not
- dojo.event.topic solves some of this
- Significant # of if(condition) statements
- State checking objects

```
this.isFooValid = function() {

    // do some condition checks

    return bool;

}
```

Problem: Code Dependencies

- Namespace collisions
- Pulling code from multiple sources
- Timing issues

Solution: Dojo modules

- Package system
  - modular
  - namespace-aware
  - define dependencies
- Build system
  - Ant + package system
  - compressed, optimized, single .js file

Namespacing

- dojo.provide(“dojo.foo.bar”);
- Allows you to easily define dojo.foo.bar
  - Useful for classes, closures, & singletons
- Prevents collision of same global names

dojo.require

- dojo.require(“dojo.foo.bar”);
- Specify required dojo modules or your code
- dojo loads all files needed at run time
- Optional build system
  - determines all dependencies for you
  - compresses, strips comments
  - include your own custom packages

http://blog.dojotoolkit.org/2006/01/11/putting-your-code-outside-of-the-dojo-source-tree

Timing issues

- Time dependency issues have been an issue
  - since frames and JavaScript in Netscape
- Multiple libraries, and load order can be painful
- Dojo has several mechanisms to avoid this
  - different load times are captured
  - execution is halted on some things
  - build system helps by creating a single .js
- We try to avoid cross frame UI communication
  - use iframes for events and data transit

Possibilities

- Best of both worlds: web and desktop
  - web: live data, always-on, scripting
  - desktop: responsive, app-like “controls”
- We’ve only just started

Best Practices - review, part 1

- Use Dojo, or some toolkit that helps
- Ajax and comet for data AND events
- Add/update/remove for all
- Avoid hard-coded ids
- Loosely coupled nodes or widgets
- Namespacing
- Separate DOM from logic when possible

Best Practices - review, part 2

- Widgets
- Events, events, events, and more events
- AOP
- Topics and pubsub - notify
- Pair of methods for topics
- Data Providers - make them your friend
- State checking objects

Best Practices - review, part 3

- Timing conditions and performance
- Compression and packaging
- Take inspiration from the web
- Take inspiration from the desktop

Thanks

- Dojos: http://dojotoolkit.org/community/contributors.shtml
- The “Renkooks”
- The Ajaxians
- You

- dylan.io/tae/eventui.pdf

Questions?

Answers!