IBM Software Group

Accessibility with Dojo

Becky Gibson
Web Accessibility Architect
IBM Emerging Technologies

ON DEMAND BUSINESS™

Agenda

- Web Accessibility Basics
  - Issues
  - Why Should Developers Care?
- Dojo Accessibility Issues
- Dojo Accessibility Strategy
  - Accessibility Detection
  - Accessible Widget subclass
  - Keyboard support
  - WAI – Accessible Rich Internet Application (ARIA) Techniques
- Demos
- AJAX Accessibility Techniques

Disabilities and the Web

- Cognitive
- Hearing
- Mobility
- Vision

Cognitive

Issues

- Learning disabilities
- Distractions
- Time limits
- Seizures

Technologies

- Specific color schemes
- Screen readers
- Close captioning
- Spell and grammar checkers

Hearing

Issues

- Deaf
- Hearing impairments

Technologies

- Captioning
- Signed captioning
- Increased volume

Mobility

Issues

- Limited mouse movement
- Keyboard only access
- No use of hands

Technologies

- Simple mouse navigation
- Keyboard only navigation
- Single click devices
- On screen keyboard
- Touch screen
- Speech recognition
- Screen magnification
- Large fonts

Vision

Issues

- Blind
- Low vision
- Color blindness

Technologies

- Keyboard only navigation
- Screen readers
- Braille displays
- Large fonts
- High contrast mode
- Screen magnifiers
- Specific color schemes

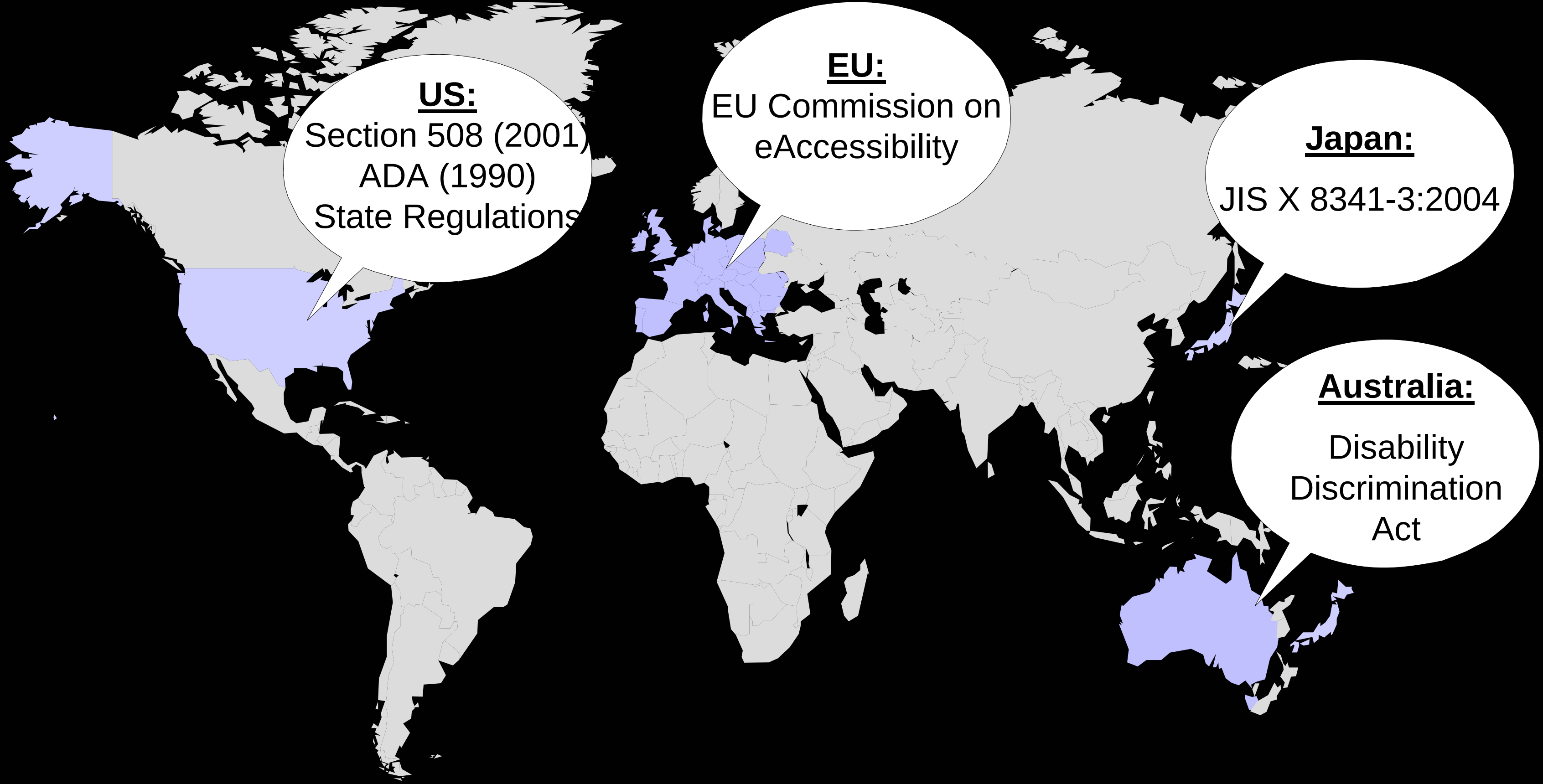
Basic Accessibility

Screen Reader Demo

Web Accessibility – Why Do I Care?

- Legal
- Financial
- Equal Access

Worldwide Accessibility Legislation



Financial

- Sell to US government
- Purchasing power

  "The large and growing market of people with disabilities has $175 billion in discretionary spending, according to the U.S. Department of Labor. $175 billion is almost two times the spending power of teens and more than 17 times the spending power of tweens (8-12 year-olds), two demographics sought after by businesses. " U.S.Department of Justice, Civil Rights Division, *Disability Rights Section*, April 2005. (**http://www.usdoj.gov/crt/ada/busstat.htm**)
- Good reputation for your company

Equal Access

- Improvements help all
- All know someone with disability
- Aging population

Dojo Accessibility Issues

- 0.4 Release
- Use of CSS background images
  - Better performance than <img> elements
  - Easier to modify the theme via CSS
  - BUT don’t work in high contrast or images off mode
- (Many) Widgets work only with Mouse
  - No Keyboard Access
- Limited information about behavior of Widget

Demo of Issues

Dojo Accessibility Strategy

- Detect high contrast mode / images off
- Render widget using <img> elements or fall back to HTML control
  - <img> elements are visible in high contrast mode
  - Theme is not applicable in high contrast mode
  - In images off mode <img> elements have a text alternative via the alt attribute
- Add full keyboard support
- Implement W3C Web Accessibility Initiative Accessible Rich Internet Application (WAI-ARIA) Techniques to provide full accessibility

Dojo A11y Strategy – Accessibility Detection

- Detect High Contrast / Images Off Mode
  - Before the widgets are rendered
  - Only once per page
- Perform the A11y check by default
- Created A11y object

Dojo A11y Strategy – A11y.js

- **`checkAccessible()`**
  - One time check for high contrast / images off
- **`testAccessible()`**
  - Performs check each time it is called
- **`setCheckAccessible(/* Boolean */ bTest)`**
  - Turns automatic checking on/off
- **`setAccessibleMode()`**
  - Perform check and set mode to load accessible widgets if necessary
  - Called from **`Widget.buildWidgetFromParseTree()`**

Check for High Contrast / Images Off

```
this.accessible = false; //default
if (dojo.render.html.ie || dojo.render.html.mozilla){
        var div = document.createElement("div");
        div.style.backgroundImage = "url(\"" + this.imgPath + "/tab_close.gif\")";
        // must add to hierarchy before can view currentStyle below
        dojo.body().appendChild(div);
        // in FF and IE the value for the current background style of the added div
        // will be "none" in high contrast mode
        // in FF the return value will be url(invalid-url:) when running over http
        var bkImg = null;
        if (window.getComputedStyle  ) {
                var cStyle = getComputedStyle(div, "");
                bkImg = cStyle.getPropertyValue("background-image");
        }else{
                bkImg = div.currentStyle.backgroundImage;
        }
        var bUseImgElem = false;
        if (bkImg != null && (bkImg == "none" || bkImg == "url(invalid-url:)" )) {
                this.accessible = true;
        }
        dojo.body().removeChild(div);
}
return this.accessible; /* Boolean */
```

Dojo A11y Strategy – Create Vision Accessible Widget

- Use Dojo’s ability to load appropriate widget class based on renderer
- Create new subclass for accessible version of widget
  - Override functions as needed to:
    - Use real <img> elements rather than CSS images
    - Render native HTML controls where appropriate
  - Create accessible template file if needed

Dojo A11y Strategy – Vision Accessible Checkbox

```
dojo.widget.defineWidget(
        "dojo.widget.a11y.Checkbox",
        dojo.widget.Checkbox,
        {
        templatePath: Dojo.uri.dojoUri('src/widget/templates/CheckboxA11y.html'),
        postCreate: function(args, frag){
                this.inputNode.checked=this.checked;
                if (this.disabled){
                        this.inputNode.setAttribute("disabled",true);
                }
        },
        fillInTemplate: function(){
        },
        _onClick: function(){
                this.onClick();
        }
        }
);
```

Dojo Accessibility Strategy – Keyboard Support

- Allow keyboard only usage
- Treat each widget as a unique component
- Use tab key to navigate between major components
  - Allows fast, direct keyboard access to specific functionality without excessive tabbing
- Within components use arrow keys to navigate
- Mimic the keyboard behavior of the desktop
- Support multi-selection where appropriate
- Set focus within the component
  - Allows screen reader to speak information about element

Setting Focus - Use of tabindex

tabindex Attribute	Focusable with Mouse or JavaScript via element.focus()	Tab Key Navigable
not present	Follows default behavior of element (yes for form controls, links, etc.)	Follows default behavior of element
<0	Yes	No, author must focus it with element.focus() as a result of arrow or other key press
0	Yes	In tab order relative to element's position in document
>0	Yes	Tabindex value manually changes where this element is positioned in the tab order. These elements will be positioned in the tab order before elements that have tabindex="0" or that are naturally in the tab order.

Assigning tabindex

- The element in component to receive initial focus gets tabindex=0
- Generally only one element within the component has tabindex=0 at any one time
- When an element receives focus
  - set tabindex=0
  - Element is added to the tab order and is tab navigable
- When an element loses focus
  - set tabindex=-1
  - Element is removed from the tab order but can receive focus via the mouse or programmatically

Setting Focus

- Keyboard and mouse interaction must be kept in-sync
- Set focus to elements, do NOT simulate focus via CSS
  - Screen reader will speak the element when it receives focus
- Element should implement onfocus handler to respond to focus via
  - Keyboard
  - Mouse
  - Programmatically
  - Speech or other input

Determining Keyboard Behavior

- Where possible follow the conventions of the Operating System
- DHTML Style Guide is in-process
  - Representatives from several companies working to define behavior of common Web components
  - Just getting started
  - Public Wiki to record results:

  http://www.weba11y.com/styleguide

So far we have support for high contrast / low vision and keyboard access,

Are we done?

NO!

We need a strategy to mimic the accessibility of Desktop Graphical User Interface (GUI) applications!

Enter
WAI – ARIA
W3C Web Accessibility Initiative
Accessible Rich Internet Applications

Why WAI-ARIA?

- Assistive Technology (AT) supports platform accessibility APIs
- Scripted Web content doesn’t (quite) support these APIs
- Assistive Technology needs to
  - Know active roles of HTML elements acting as interaction widgets
  - Monitor states of active HTML elements
- Accessibility API sees static structure of HTML (today)
- Script gives arbitrary HTML elements active roles and states
  - This information is unknown to the accessibility API and ATs
- WAI-ARIA defines these roles and states for use on the Web

WAI – ARIA Supported Formats

- Works in HTML or XHTML
  - XHTML
    - Uses XHTML 1.x role attribute
    - Uses namespaces to add state attributes
  - HTML
    - Embeds role and state information into class attribute
    - Accessibility Script library sets the role and state using DOM apis
    - Dojo WAI object to set role and state using DOM APIs
  - Implemented in Firefox 1.5
  - Supported by Window-Eyes 5.5 screen reader

WAI-ARIA Support in Dojo

- Add role via the Dojo Template
  or
- Can Add role programmatically in the widget code
- Update state programmatically as it changes
- Keyboard support and role and state are added to the base widget class
  - Accessibility “derived class” is only for visibility issues!

WAI Additions to DomWidget.js

```
dojo.widget.waiNames  = ["waiRole", "waiState"];
dojo.widget.wai = {
waiRole: { name: "waiRole",
 "namespace":"http://www.w3.org/TR/xhtml2",
 alias: "x2",
 prefix: "wairole:"},
waiState: { name: "waiState",
 "namespace": "http://www.w3.org/2005/07/aaa",
 alias: "aaa",
 prefix: "“},
setAttr: function(node, ns, attr, value){
if(dojo.render.html.ie){
 node.setAttribute(this[ns].alias+":"+ attr,
      this[ns].prefix+value);
}else{
  node.setAttributeNS(this[ns]["namespace"],
 attr,      this[ns].prefix+value);
}},
getAttr: function(node, ns, attr, value){…}
};
```

Assigning the Role Value in the Template

- TabContainer.html

```
<div id="${this.widgetId}“
     class="dojoTabContainer">
 <div dojoAttachPoint="tablistNode"></div>
 <div class="dojoTabPaneWrapper"
     dojoAttachPoint="containerNode"
     dojoAttachEvent="onKey"  waiRole="tabpanel">
 </div>
</div>
```

Assigning the role Value in Code

```
dojo.widget.wai.setAttr(this.domNode,
     "waiRole", “role", “slider”);
```

Assigning the State Value in Code

- Checkbox.js

```
_setInfo: function(){
// summary: set CSS class string according to checked/unchecked and
  disabled/enabled state
var state = "dojoHtmlCheckbox" + (this.disabled ? "Disabled" :     "") +
  (this.checked ? "On" : "Off");
dojo.html.setClass(this.imageNode, "dojoHtmlCheckbox " + state);
this.inputNode.checked = this.checked;
if (this.disabled){
  this.inputNode.disabled = true;
 dojo.widget.wai.setAttr(this.domNode, "waiState", “disabled", true);
}
else if(dojo.widget.wai.getAttr(this.domNode, “waiState”) == true){
  dojo.widget.wai.removeAttr(this.domNOde, “waiState”, “disabled”);
}
dojo.widget.wai.setAttr(this.domNode, "waiState", "checked",
  this.checked);
}
```

ARIA Support in Firefox

- Roles and states are assigned in Dojo template or code
- Firefox obtains the information from the DOM converts it into MSAA (Microsoft Active Accessibility API)
- Assistive Technology responds to MSAA events
- Window-Eyes 5.5 screen reader and Firefox 1.5 (or later versions) support ARIA techniques

Role Examples

- link
- combobox, option
- checkbox
- radio, radiogroup
- button
- progressbar
- slider
- spinbutton
- tree, treeitem
- alert
- application
- presentation
- group
- grid, gridcell
- tab, tabcontainer, tablist, tabpanel
- list, listitem
- menubar, menu
- toolbar
- more……

State Examples

State	Values
checked	true | false | mixed
disabled	true | false
readonly	true | false
expanded	true | false
valuemin, valuemax, valuenow	CDATA
hasparent, haspopup	IDREF
describedby. labeledby	IDREF

Demo

AJAX Accessibility Issues

- User does not know updates will occur.
- User does not notice an update.
- User can not find the updated information.
- Unexpected changes in focus.
- Loss of Back button functionality.
- URIs can not be bookmarked.

Specific Accessibility Issues

- Assistive Technology users are not aware of updates
  - Updates occur on a different section of the page than the user is currently interacting with.
  - Clicking a link updates a different section of the page.
  - Auto-complete fields or generated options not available to assistive technology.
  - User has no idea how to find new or updated content
  - Changes in focus prohibit complete review of the page
  - Changes in focus cause disorientation and additional navigation.

Informing the User

- Explain the interaction to the user
  - Before accessing the AJAX enabled page
  - Within the AJAX enabled page
- Where possible, provide a manual update option
  - Necessary for WCAG 2.0 Success Criterion (SC) 2.2.5 - Interruptions, such as updated content, can be postponed or suppressed by the user, except interruptions involving an emergency.
- Save the user’s update preference

Make Updates Noticeable

- Change the background color of updated data
  - Use a subtle color
  - Change only for a few seconds
  - Best for small areas of the page
- Briefly blink the updated data
  - Blink for 3 seconds or less
    - WCAG 2.0 SC 2.2.2: Content does not blink for more than three seconds, or a method is available to stop all blinking content in the Web page.
  - Avoid the flash threshold
    - WCAG 2.0 SC 2.3.1: Content does not violate the general flash threshold or the red flash threshold
    - WCAG 2.0 SC 2.3.2: Content does not contain any components that flash more than three times in any 1-second period.

Help Users Find Updated Information

- Provide option for updates via an Alert
- Provide option to set focus to new data.
- Use HTML header tags to mark sections with updated content.
- Use WAI-ARIA Alert role in conjunction with a floating pop-up box.
- Use WAI-ARIA describedby property to describe new content.
- Use WAI-ARIA liveregion role to describe regions which update (future)

Summary

- Accessibility is important!
- Dojo becoming Accessible
- Make Accessible Widgets
- Build Accessible AJAX Applications

References

- Dojo Manual (http://manual.dojotoolkit.org/WikiHome/DojoDotBook)
- 40 AJAX Accessibility Links (http://ajaxian.com/archives/40-ajax-accessibility-links)
- Roadmap for Accessible Rich Internet Applications (ARIA Roadmap) (http://www.w3.org/TR/aria-roadmap/)
- Roles for Accessible Rich Internet Applications (ARIA Roles) (http://www.w3.org/TR/aria-role/)
- States and Properties Module for Accessible Rich Internet Applications (ARIA States and Properties) (http://www.w3.org/TR/aria-state/)
- Web Content Accessibility Guidelines 2.0 (http://www.w3.org/WAI/GL/WCAG20/)