Building Fully Accessible Social Software and Rich Web Applications with WAI-ARIA

Damian Chojna

IBM Collaborations Software Engineer, IBM Software

Matt King

I/T Chief Accessibility Strategist, IBM BT/CIO Office

Rich Schwerdtfeger

CTO Accessibility IBM Software

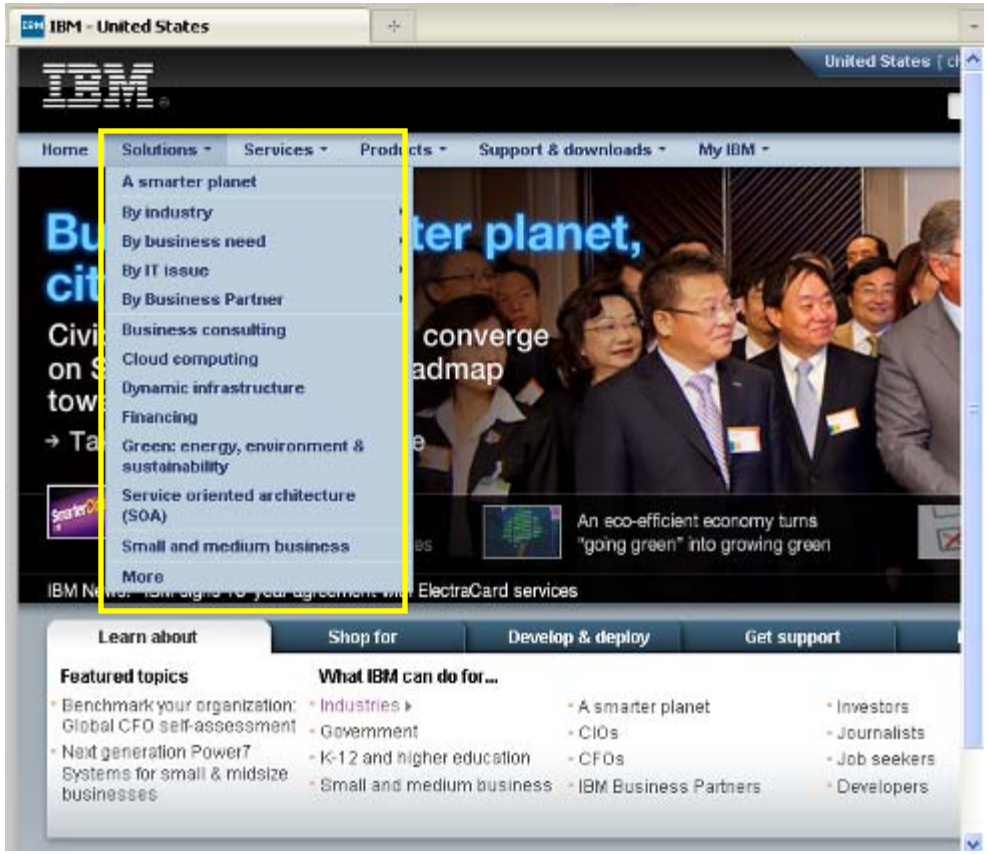
Agenda

- What is W3C WAI-ARIA?
- A look at IBM Connections
- Design considerations for accessible web applications
- Lessons learned

What is W3C ARIA (Accessible Rich Internet Applications)?

- A way for authors to apply rich accessibility semantics in Web 2.0 content to support OS platform accessibility
- A way to reproduce the keyboard functionality of desktop applications on a web page
- A vehicle to provide full interoperability with assistive technologies for Rich Internet Applications through the browser
- A vehicle to correct static (X)HTML accessibility deficiencies

WAI-ARIA an Open Standard



- 20% of the work needed for rich desktop
- A Cross platform accessibility API
- Full Keyboard navigation like desktop
- Ubiquitous adoption
- Included in over 170 IBM products
- Designed to support WCAG 2 and the U.S. 508 Refresh
- All major browsers providing support

Markup for menu:

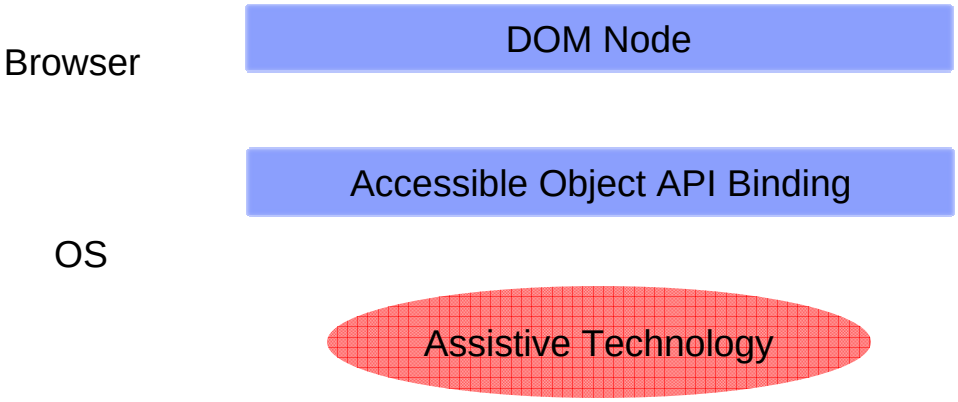
```
<div role="menu" aria-haspopp="true">
  <div role = "menuitem" aria-selected="true">
  A smarter planet
  </div>…
</div>
```

The Assistive Technology would read the menu as: Menu Item. A Smarter Planet. One of 13 menu items.

Browser converts ARIA to accessibility services

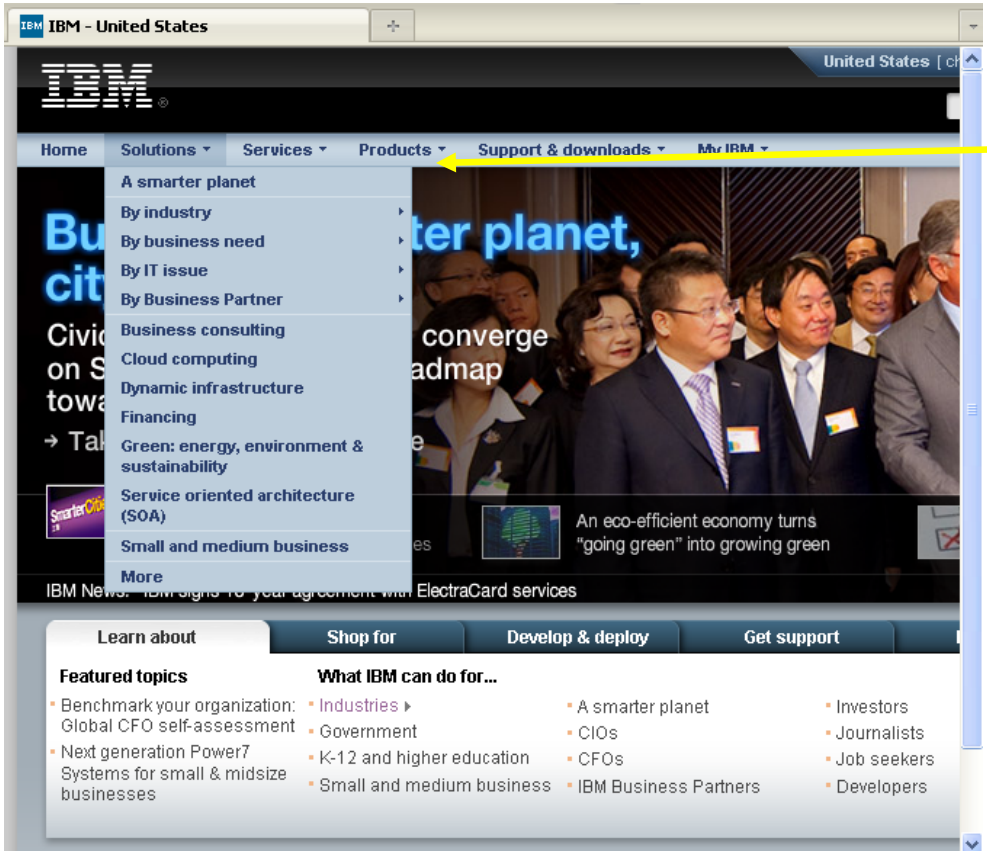
WAI-ARIA – Significant advancement in accessibility vs. desktop

WAI-ARIA



- Semantical Structure through tree hierarchy
- Attribute change notification
- Focus management
- Styling
- Role
- States and Properties
- Rich Text
- State and Property Event Notification
- Actions
- Structural access to other objects
- Advanced interfaces (Tables, relationships)
- OS Accessibility API Notification

New Information as seen by Assistive Technology



New accessibility Information

Role	Menu Item
State	Selected
Name	Top Stories
Actions	None
Parent	Menu

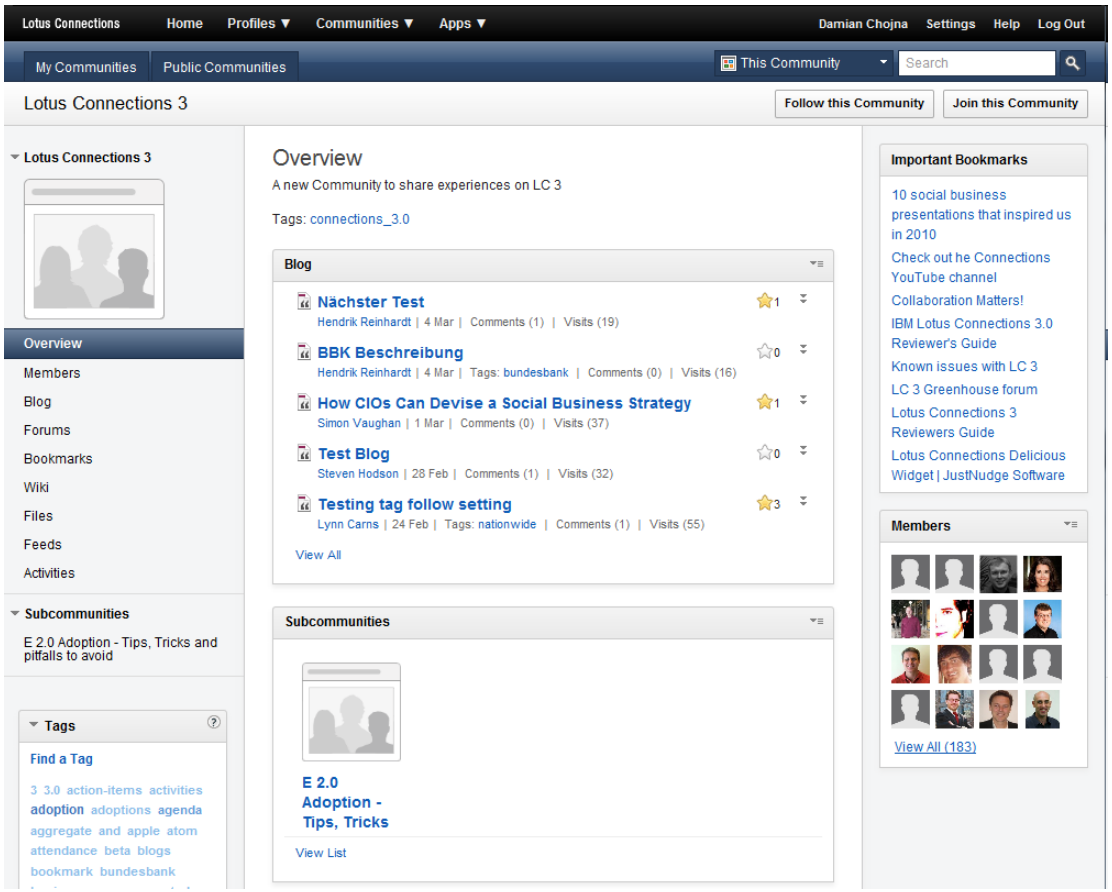
WAI-ARIA – delivers semantics and desktop keyboard functionality to provide full interoperability with ATs

- Typical widget states
  - aria-checked, aria-selected, aria-disabled, aria-currentvalue, aria-expanded, etc.
- Relationships
  - aria-describeby, aria-controls, aria-flowto, aria-labelledby, aria-owns
- New AJAX Live Region properties
  - aria-live (off, polite, assertive)
  - aria-relevant (additions, deletions, text, all)
  - aria-atomic
- Drag/Drop
  - aria-grabbed
  - aria-dropeffect
- Miscellaneous
  - aria-sort (ascending, descending)
  - aria-setsize, aria-posinset, aria-level
- Role (widgets and navigational landmarks)
  - Widgets: (tree, grid, button, checkbox, menu, dialog, etc.)
  - Structural: (directory, list, header, etc.)
  - Landmarks: (main, navigation, complementary, banner, contentinfo, form, search, etc.)
- Tabindex

```
<div tabindex="-1" role = "menuitem" aria-disabled="true">
```

Overview of Lotus Connections 3.0

- Social Software for business
- Build and maintain social networks and communities
- Manage your profile
- Share
  - Files
  - Blogs
  - Wikis
  - Forums
  - Bookmarks
- Customize how you see your social network in Home page
- Extensive search across components



Design Considerations Overview

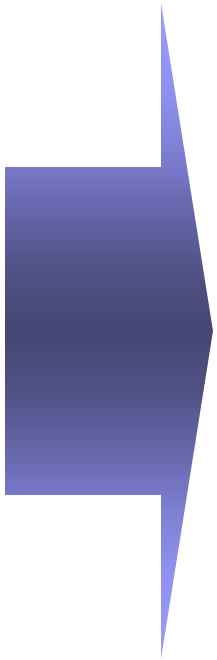
- Choosing a Toolkit or Defining a Custom Control
- Workflow & Navigation
  - Landmarks
  - Keyboard
  - Focus
- Content Editable Sections
- High Contrast

WAI-ARIA Design Consideration – Choosing a Widget Library

- Why IBM chose Dojo
  - Rich widget library
  - WAI-ARIA enabled
  - Keyboard enabled
  - Supports High Contrast

Web 2.0/ARIA Navigation Paradigm Shift Navigation

- Navigation Tab and Click
- Everything but forms and links are browsed by AT
- Page reloads for new content



- Tab to
  - Links
  - Form elements
  - Widgets *
  - Read-only documents *
- Arrow key navigation within Widgets *
- Keyboard accelerators for Widgets *
- In page navigation based on ARIA regions *
- Mix: Web Applications and documents

* Provided by author

WAI-ARIA Design Considerations – Workflow Comprehension

- How will the user understand what content and function is available?
- Which page regions should have landmark roles?
  - banner, navigation, search, main, form, application, contentinfo, complementary
  - Pay careful attention to labeling and nesting
- What other portions of the page would benefit from declaring ARIA structural roles?
  - e.g., region, article, document

WCAG2A 1.3.1: Information, structure, and relationships conveyed through presentation can be programmatically determined or are available in text.

WCAG2A 1.3.2: When the sequence in which content is presented affects its meaning, a correct reading sequence can be programmatically determined.

WCAG2A 2.4.1: A mechanism is available to bypass blocks of content that are repeated on multiple Web pages.

WAI-ARIA Design Considerations – High-level Site navigation

- How will users navigate among application components?
  - Is navigation layered, e.g., hierarchical tree, nested tabs?
  - Is the interface tabbed?
  - If tabbed, does it fit the conventional tabbed interface pattern? Or, are the tabs and tab panels dispersed among and visually separated by other page elements?
- Options to consider for dispersed tabbed interfaces:
  - navigation toolbars with toggle buttons
  - menubars with menuitemradio elements
  - listbox of links

WCAG2A 2.4.3: If a Web page can be navigated sequentially and the navigation sequences affect meaning or operation, focusable components receive focus in an order that
preserves meaning and operability.

WCAG2A 4.1.2: For all user interface components ... the name and role can be programmatically determined; states, properties, and values ... can be programmatically set; and notification of changes to these items is available

Demo Profiles

- Landmarks
- Navigation Toolbar

Navigation within Connection components

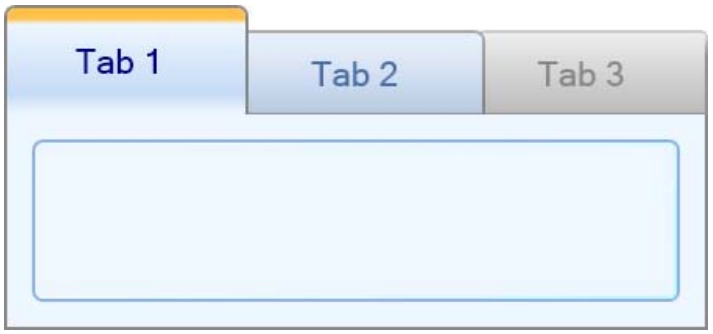
- Two levels of navigation within a Connections component
  - Top level for navigating between different sections
  - Secondary level for navigating and/or filtering within the same application section

Tab panel or toolbar?

- Both tab panel and toolbar have similar container-child structure
- Different interaction patterns
- Primary motivation on which interaction pattern to apply is based on the user's interaction with the widget
- Visual representation is not always the deciding factor

Tab panel

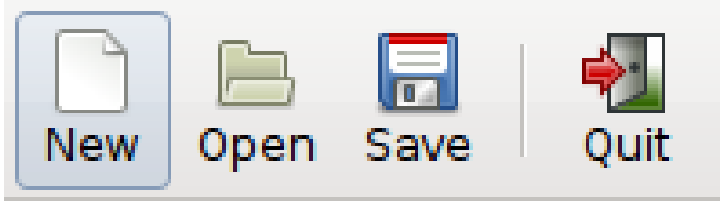
– Container for resources associated with a tab: set of layered pages, only one page is displayed at a time



– When the user activates a tab, the contents of the corresponding tab panel are made visible; the tab is considered "active" and remains such until another tab is activated
– The active tab is placed into the tab order; only the active tab should be in the tab order
– A default tab is specified that is active when the tabbed interface component is initialized

Toolbar

- Flat non-hierarchical collection of controls that provides quick access to a subset of functions



- Tab moves focus to the first enabled toolbar button
- A subsequent Tab moves focus out of the toolbar
- Left Arrow and Right Arrow keys navigate to the enabled buttons in the toolbar
- Direction needs to be adjusted for Right to Left languages

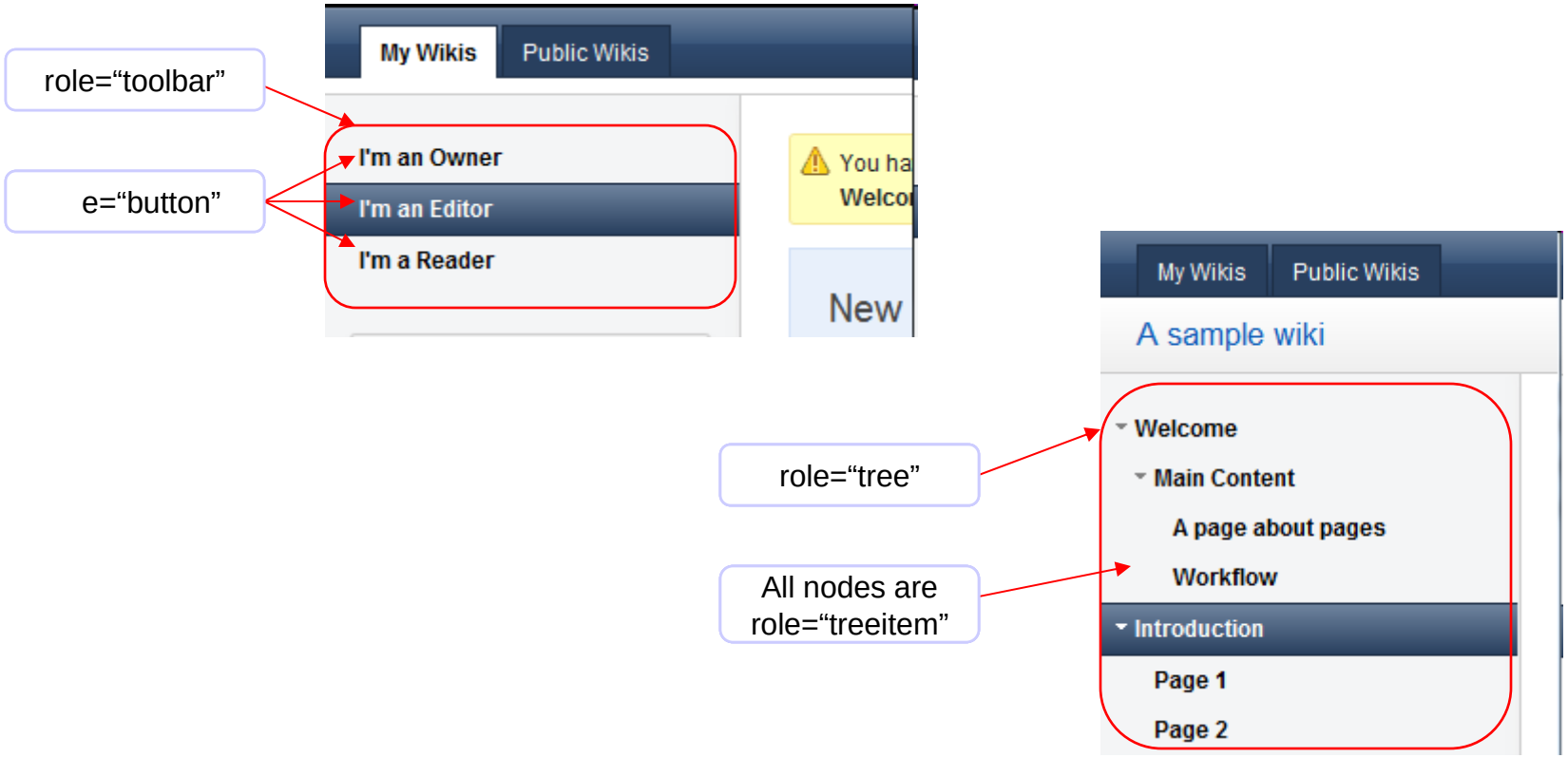
Top navigation

- Implemented as a toolbar with toggle buttons
- Use aria-controls to indicate which part of the UI the widget controls


```
<ul role="toolbar" aria-label="Wikis navigation tab" aria-controls="lotusMain">
        <li>
            <a href="" role="button" aria-pressed="true" aria-label="My Wikis"></a>
        </li>
        <li>
            <a href="" role="button" aria-pressed="false" aria-label="Public Wikis"></a>
        </li>
</ul>
```

Secondary navigation

– Use toolbar, tree or other structure depending on interaction and complexity of choices



Tabbed navigation example


```
<ul role="tablist" aria-label="...">
  <li>
    <a aria-selected="true" aria-controls="comments“ href="..." role="tab">Comments (1)</a>
  </li>
  <li>
    <a aria-selected="false" aria-controls="versionHistory“ href="..." role="tab" tabindex="-1">Versions (1)</a>
  </li>
  <li>
    <a aria-selected="false" aria-controls="attachments“ ref="..." role="tab" tabindex="-1">Attachments (0)</a>
  </li>
  <li>
    <a aria-selected="false" aria-controls="about“ href="..." role="tab" tabindex="-1">About</a>
  </li>
</ul>
```

Keyboard Navigation (managing focus) basics

- Tabindex can be used to control how and if an element is in the tab order
- Tabindex= “-1” Can set focus on an element without adding to tab order
  - Ideal for Widgets
- Tabindex=“0” Place focusable elements in the tab order in document order
- Tabindex = “> 0” Same as today’s tabindex

Author-managed arrow key navigation within widgets

- Capture key strokes at the widget managing focus
- Move focus to the child using tabindex:
  - Set the tabindex="-1" on the child element to allow it to be focusable
  - Childelement.focus();
  - Draw visible focus to the child with focus using styling
  - Browser will fire a focus change event to the AT
- Managing parent indicate which child has focus using active descendant
  - Set aria-activedescendant="active childID" on the parent (like listbox or menu)
  - Draw visible focus to the child with focus using styling
  - Browser fires focus event on behalf of the child

Providing keyboard navigation support in Connections

- Use Dojo widgets that have been enabled with WAI-ARIA supported keyboard navigation
  - e.g., Tree navigation, menus etc.

- For custom widgets a reusable library was created to enable quick integration of standard keyboard interaction patterns
  - e.g., Toolbars and tab lists

ARIA Helper

- Connections components (Wikis, Profiles etc.) are developed by different teams each with their own implementation of the user interface
  - Some of the user interface existed before the WAI-ARIA support was added
  - Each team needs to provide keyboard navigation for their navigation widgets
- The ARIA Helper component was developed to help each developer to apply the correct keyboard interaction pattern for either Tabbed or Toolbar based navigation
- ARIA Helper takes a HTML DOM structure that has correct ARIA roles hierarchy for Tab panel or Toolbar and hooks in the correct keyboard navigation and focus control

Demo

- Mega menu
- Maintaining focus

Integrating Content Editable Sections

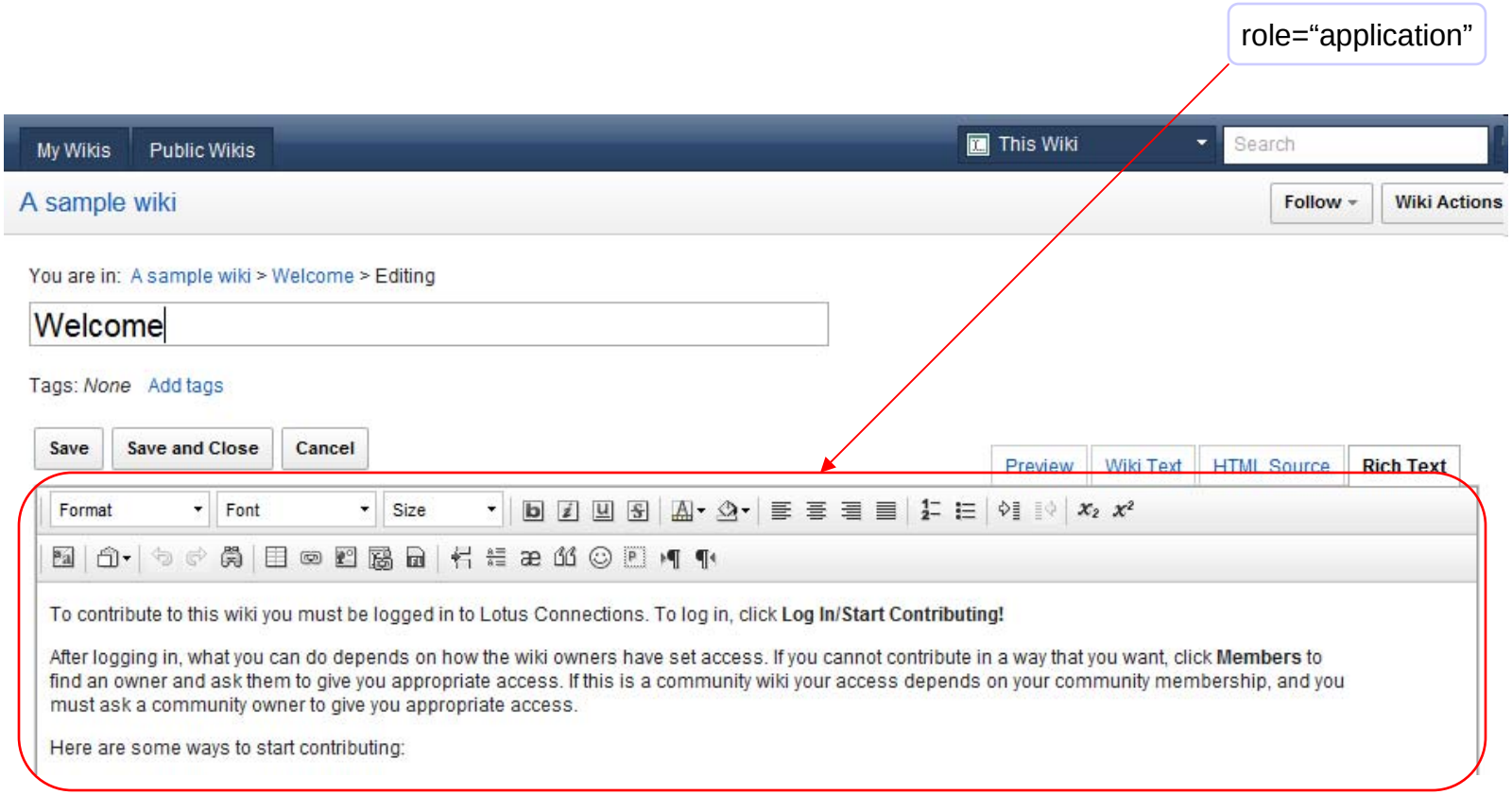
- What is a “contenteditable” section?
- Navigation between the toolbar and rich text editor
- Supporting dialogs
- Working with the browser and screen reader manufacturer
  - Embedded tables
  - Embedded objects

Demo

- Activities – create section
- CKEditor help

Rich Text Editor

- Use role=“application” for top level
- Self contained widget
- Keyboard navigation completely controlled by the widget



WAI-ARIA Design Considerations - Workflow

- HTML Body Role

- Is it a Document that contains application widgets? (<body role="document">)
  - This is the default approach familiar to all

- Is it an Application that contains document components?
  (<body role="application">)

- Screen reader considerations

- role=document (default) enables screen reader document reading mode on page load
  - What everyone is used to
  - Screen reader quick nav keys to move by landmark, heading, paragraph, form element, etc., are available
  - Screen reader goes in/out forms/app mode when app widgets are encountered

- role=application forces app mode on page load
  - With JAWS insert+z is required to leave app mode
  - Expect user to access all content via app mode (no screen reader quick nav keys)
  - All static text must be in keyboard nav order (tab index 0 and role document)
  - Author responsible for providing efficient keyboard method to navigate page

Design Considerations High Contrast

- High contrast is handled differently by different OS Platforms
  - Windows – High contrast color scheme, large gonts
  - Mac – Changes system palette
- Avoid hard coding font sizes, shapes, etc. in pixels
- Widget libraries that support high contrast
  - Dojo Dijit
  - Some JQuery

Detecting high contrast mode

- No simple way to detect high contrast mode
- How to detect high contrast?
  - **Step 12 in http://www.w3.org/WAI/PF/aria-practices/#accessiblewidget demonstrates how to detect high contrast mode**
  - **Dojo toolkit provides a mechanism to detect high contrast mode**

Developing & Testing

- High level design
- Blind user testing, walk through and reviews
- Developer tools
  - WCAG 2.0 and Section 508 verification with Webking
  - JAWS 12
  - Custom WAI-ARIA validator
- IBM will soon be deploying Rational Policy Tester…