namics
a PubliGroupe Company

Accessible Rich Internet Applications (ARIA)

Best of Accessibility Symposium
22. September 2008

namics (deutschland) gmbh
Gutleutstraße 96
D–60329 Frankfurt am Main

t [+49] 69 365 059 0
f [+49] 69 365 059 100
info@namics.com

Büros in:
Bern, Frankfurt, Hamburg, München,
St. Gallen, Zug, Zürich

Inhaltsverzeichnis

1 Accessible Rich Internet Applications (ARIA) in der Praxis

von Martin Kliehm, Senior Frontend Engineer bei
namics (deutschland) gmbh, Frankfurt am Main

Web 2.0 Anwendungen sind aufgrund der beschränkten Möglichkeiten von (X)HTML oft nicht barrierefrei oder haben Usability-Probleme. Der Standard-Entwurf des W3C für *Accessible Rich Internet Applications* (ARIA) überbrückt diese Beschränkungen. Er schafft neue Wege zur Kommunikation von Bedeutung, Relevanz, Beziehungen, füllt die Lücken in den (X)HTML Spezifikationen und steigert die Usability für alle Nutzer, indem er vertraute Navigationsmodelle des Desktops übernimmt. Und das Beste: man kann ARIA sofort einsetzen, um die Zugänglichkeit von Webseiten zu verbessern.

Diese Erweiterung wurde zuerst von Richard Schwerdtfeger von IBM vorgeschlagen. Er ist Mitglied der WAI Protocols and Formats Working Group und der HTML Working Group. Ebenso beteiligt war Becky Gibson, die in der WAI WCAG Working Group aktiv ist. Zusammen mit Aaron Leventhal haben sie diese und andere barrierefreie Techniken in Firefox ab Version 1.5 und in verschiedenen assistiven Technologien implementiert.

„Natürlich ist das alles illegal“, erwiderte Joe Clark, und während ich voll zustimme, dass Standards nicht missachtet werden dürfen und dass es dringendere Themen gäbe, war es eine pragmatische Lösung zur rechten Zeit. Im August 2005 spendete IBM eine große Menge Quellcode, um die Barrierefreiheit des Mozilla Browsers zu verbessern. Kurz nach Clark’s Kritik wurden die ersten Standardentwürfe des W3C für Roles sowie States and Properties für Accessible Rich Internet Applications (ARIA) veröffentlicht. Fast ein Jahr später wurde das `role`-Attribut als XHTML 1.1 Modul eingeführt. Das „X“ in „XHTML“ steht schließlich für „extensible“ (erweiterbar), aber mehr dazu später. Die Vorschläge wurden mit ungewöhnlichem Tempo durch die W3C-Prozeduren gepuscht und sollen im zweiten Quartal 2007 Standard-Status erlangen. Phantastisch, wie effizient das W3C unter den richtigen Bedingungen arbeiten kann.

1.1 Semantischer Zuckerguss

Zeitgemäße Web-Programmierung verwendet Aufzählungslisten (`li`) für die Navigation oder den Brotkrumenpfad, und es gibt verschiedene Bereiche in einer Seite für den Textinhalt oder die Marginalspalte. Zudem kann in Web 2.0 Anwendungen *alles* ein Button oder ein Bedienelement sein (etwa ein Schieberegler). Das Problem mit der Barrierefreiheit ist, dass Screenreader bislang keine Möglichkeit haben festzustellen, welche Funktionalität diese Elemente besitzen. Das XHTML `role`-Attribut füllt diese Lücke, indem es „semantischen Zuckerguss“ hinzufügt:

```
<ul role="navigation"> […] </ul>
```

Das stellt ein semantisches, maschinenlesbares `role`-Attribut zur Verfügung, das ein Browser auf die zugrundeliegende Accessibility API des Betriebssystems mappen kann. Das Attribut verleiht dem Element semantische Informationen, die von Maschinen und Menschen im Quellcode gelesen werden können. Assistive Technologien wie Screenreader können aufgrund dieser Informationen das Element korrekt interpretieren.

Rollen gibt es in zwei Geschmacksrichtungen: XHTML und WAI-ARIA. Ein Basisset ist im XHTML 1.1 Role Attribute Modul definiert. Es wird erweitert durch die WAI ARIA-Role RDF Taxonomie.

Rollen werden darüber hinaus unterschieden in **Widgets** und **strukturelle Rollen**. Widget-Rollen beinhalten `progressbar`, `slider`, `button`, `tree`, `textfield`, `checkbox`, `alert` oder `dialog`. Wenn man also lieber einen Layer anstelle einer schnöden Dialogbox verwenden möchte, kann man das dem Screenreader über `role="dialog"` mitteilen. Weitere Beispiele für Widgets bietet mozilla.org.

Gängige strukturelle Rollen beinhalten `main`, `secondary`, `group`, `section`, `liveregion` (in der Inhalte durch AJAX oder vergleichbare Techniken verändert werden), `tab`, `navigation`, `menubar`, `toolbar`, `breadcrumbs`, `search`, oder `banner`.

1.2 Bedeutung schaffen

Das `role` Attribut vermittelt Informationen, *was das Objekt ist*. Das States and Properties Modul (ARIA-State) fügt *Bedeutung* hinzu, schafft Beziehungen zwischen Objekten und vermittelt ihren aktuellen Status:

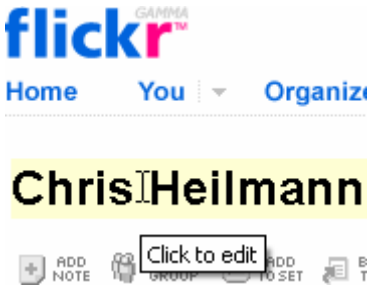
```
<input type="text" name="email" aria-required="true"/>
<div role="button" aria-controls="price">Sortierung
ändern</div>
<h2 id="price" aria-sort="descending">Preis</h2>
```

Andere Status umfassen `aria-checked`, `aria-disabled`, `aria-expanded`, `aria-haspopup`, `aria-invalid`, `aria-readonly`, `aria-describedby` und `aria-labelledby`, den `aria-level` eines Elements in einer Baumstruktur, `aria-valuenow`, `aria-valuemin` und `aria-valuemax` eines Schiebereglers, oder ob ein Menüpunkt gerade aktiv ist.

1.3 Wo bin ich?

Assistive Technologien müssen wissen, mit welchem Objekt sie gerade arbeiten und welches Element den Fokus hat. Derzeit können nur Links und Formular-Elemente den Fokus bekommen. ARIA-State erweitert gängige Textcontainer um das `tabindex` Attribut, so dass der Fokus entweder durch Scripting oder durch die Tabulatortaste gesetzt werden kann.

Derzeit kann beispielsweise eine editierbare Überschrift bei flickr nur den Fokus bekommen, wenn man mit der Maus darauf klickt. Mit `tabindex="0"` würde sie zugänglich per Tastatur.



Außerdem kann der Wert nun *negativ* sein. Elemente mit einem negativen Tabindex können den Fokus per JavaScript erhalten, werden aber in der Tabulator-Reihenfolge ausgelassen. Dieses Feature wurde von Microsoft im Internet Explorer 5.01 eingeführt und ist in Firefox ab Version 1.5 implementiert.

Zur Auffrischung eine Tabelle mit den Werten und dem Browserverhalten von `tabindex`:

	fokusierbar	**Tab navigierbar**
kein tabindex	Grundeinstellung (nur Formular-Elemente und Links)	Grundeinstellung
tabindex="-1"	Ja	Nein, Autoren müssen `element.focus()` für den `onkeydown`-Event von Pfeil- und anderen Tasten programmieren
tabindex="0"	Ja	Ja, in der Reihenfolge, in der die Elemente im Quellcode auftauchen
positiv, z.B. tabindex="100"	Ja	Ja, der `tabindex` bestimmt die Tabulator-Reihenfolge der Elemente. Diese Elemente erhalten den Fokus vor allen anderen Elementen mit `tabindex="0"` oder ohne `tabindex`.

1.4 Anwendung

Das ist ja schön und gut, aber müssen wir jetzt noch mal zehn Jahre warten, bis andere Browser diese Techniken unterstützen? Nicht wirklich. Man kann Roles, States und Properties sofort einsetzen. ARIA wird unterstützt von Firefox ab Version 1.5, Opera 9.5, Safari 3.0 und Internet Explorer 8, sowie einigen großen Screenreadern (Window Eyes 5.5+, Jaws 7.0+, NVDA, ZoomText).

Einige der neuen Attribute sind schon in XHTML 2 enthalten, XHTML 1.x müssten wir aber erweitern. Es gibt verschiedene Wege, ARIA dranzudübeln, aber mittelfristig kann auch die neue DTD des W3C verwendet werden, die derzeit allerdings noch nicht validiert:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML+ARIA 1.0//EN"
"http://www.w3.org/MarkUp/DTD/xhtml-aria-1.dtd">
```

1.5 Schlussbemerkung

Mit ARIA lassen sich Barrieren in dynamischen Web-Anwendungen ausräumen. Zwar validiert der Code derzeit noch nicht und widerspricht damit eigentlich einem Kriterium der Web Content Accessibility Guidelines (WCAG), aber Sinn dieses Kriteriums ist es ja, Websites zugänglicher zu machen. ARIA erfüllt das. Alle großen Browser und wichtige Screenrader unterstützen das Format, ältere ignorieren es einfach. Es gibt keinen Grund mehr, mit dem Einsatz in Projekten zu warten.

1.6 Weiterführende Literatur

» Introduction to WAI ARIA (Opera Software)
» Roadmap for Accessible Rich Internet Applications (WAI-ARIA Roadmap)
» XHTML Role Attribute Module
» Roles for Accessible Rich Internet Applications (WAI-ARIA Roles)
» States and Properties Module for Accessible Rich Internet Applications (WAI-ARIA States and Properties)
» Accessible DHTML
» Authoring Tips for Key-Navigable Custom DHTML Widgets
» Embedding Accessibility Role and State Metadata in HTML Documents

1.7 Über den Autoren

Martin Kliehm ist Senior Frontend Engineer bei namics. Er befasst sich mit Web Standards und Barrierefreiheit seit Ende 1999, schreibt in seinem Blog über diese Themen und spricht auf verschiedenen (Un)Konferenzen. Er ist Mitglied im Web Standards Project und bei den Webkrauts.