À propos de l'éditeur

Cet article est publié par le magazine Le train de 13h37.
Le train de 13h37 est un magazine en ligne francophone dédié à la conception Web qui publie régulièrement des articles exclusifs rédigés par des auteurs invités et rémunérés.

Nous avons fait le choix d'une publication gratuite sur notre site, sous licence Creative Commons BY-NC-SA et non financée par la publicité.

Nous éditons aussi des livres papiers et numériques, avec les Éditions En Voiture Simone.

Magazine édité par la société WAGON 42 SAS , au capital de 5 000€ - SIRET : 752 205 856 000 12
25 Impasse Ernest Feuillet - 84000 Avignon // 09.81.61.08.07 – contact@wagon42.fr

ARIA, il serait temps de s'y mettre !

Bien que la mode du web 2.0 soit passée, l'usage massif de javascript pour réaliser des sites web ou désormais de véritables applications en ligne est bel et bien plus que d'actualité.
Cette utilisation de javascript et ces nouveaux usages du web ne vont pas sans poser quelques difficultés aux utilisateurs de lecteurs d'écrans.
"Utilisateurs de lecteurs d'écrans" avez-vous dit, késako ? Il s'agit principalement d'utilisateurs mal- ou non voyants qui manipulent et accèdent au contenu du web et de leur ordinateur à l'aide d'un outil qui restitue vocalement le contenu et donne des informations sur celui-ci.
C'est dans le but de lever ces éventuelles difficultés que le W3C propose une spécification technique nommée ARIA *« Accessible Rich Internet Application »*.
Nous allons donc en étudier les grands principes et voir comment l'utiliser.

Qu'est ce qu'ARIA ? Explication

ARIA est en fait une surcouche sémantique que l'on vient mettre par-dessus un langage existant tel que HTML, SVG, XML, etc. Il se compose de rôles, d'états et de propriétés.
Ces ajouts sémantiques vont permettre au lecteur d'écrans de renvoyer à l'utilisateur des informations quant à la nature de ce qui est affiché à l'écran. En effet, dans les applications web ou les sites web modernes il n'est pas rare de trouver des composants d'interface proches de ceux présents dans les applications traditionnelles. Il s'agit par exemple d'onglets, d'arbres dépliants, de *cover flows*, d'info-bulles, de fenêtres modales. Autant d'éléments qui ne peuvent être décrits sémantiquement par le simple langage HTML, y compris HTML5.
Nous allons voir dans cet article le fonctionnement d'ARIA au travers de quelques exemples. Pour plus de simplicité, dans les exemples donnés, nous utiliserons ARIA sur du HTML.
L'ensemble de la spécification ARIA est quant à elle disponible à l'adresse suivante : http://www.w3.org/TR/wai-aria/.

Support navigateur et lecteur d'écrans

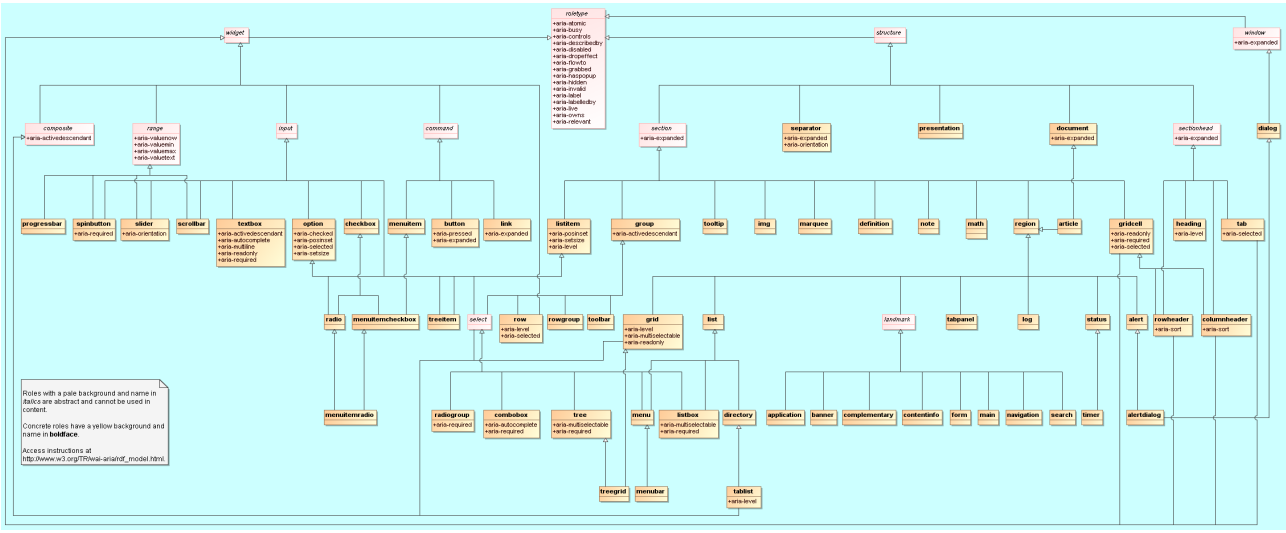
Afin d'être réellement utile aux utilisateurs, ARIA a besoin d'être supportée à la fois par les navigateurs et par les lecteurs d'écrans. Du côté des navigateurs, sur Internet Explorer, le support d'ARIA a été introduit depuis IE8, puis amélioré sur IE9. Sur Safari (OS X et iOs), le support a été introduit depuis la version 4, et largement amélioré sur la version 5 et 5.1. Sur Firefox, le support partiel a débuté dans la version 3, et est désormais complet. Enfin sur Chrome, un début de support existe depuis la version 9 et progresse régulièrement.
Du coté des lecteurs d'écrans, la plupart des versions disponibles sur le marché annoncent désormais supporter ARIA. C'est notamment le cas de Jaws, NVDA et VoiceOver. Néanmoins, malgré ce support plutôt bon, sur le terrain il est encore possible de rencontrer des soucis de compatibilité. Il reste donc impératif de tester son code régulièrement afin de vérifier son bon fonctionnement ou de faire appel à des utilisateurs réguliers de lecteur d'écrans pour le faire pour vous.

Les rôles ARIA

Il s'agit de l'élément de base d'ARIA, le plus simple à comprendre et à mettre en œuvre. Cela consiste simplement à affecter une valeur sémantique à un élément afin de décrire son contenu. On peut par exemple définir qu'un élément sans valeur sémantique comme l'élément `<span>` est en fait un bouton en utilisant le rôle « `button` »

```
<span role="button">Ceci est un bouton</span>
```

Il existe de très nombreux rôles et ceux-ci peuvent être de trois types différents.



Des rôles de structuration sémantique des documents (cf. spécifications)

directory

Pour désigner un sommaire ou une table des matières.

```
<ul role="directory">
   <li><a href="#partie1">Partie 1</a></li>
   <li><a href="#partie2">Partie 2</a></li>
   <li><a href="#partie3">Partie 3</a></li>
</ul>
```

note

Pour désigner une note.

presentation

Contrairement aux autres, permet de supprimer toute valeur sémantique d'un élément. Le contenu

d'une liste `<ul>` avec un rôle `presentation` ne sera pas restitué vocalement comme une liste. Par exemple, en terme de restitution à l'utilisateur :

```
<ul role="presentation">
   <li>Bonjour</li>
   <li>Comment vas tu ?</li>
</ul>
```

Sera équivalent à :

```
<span>
   <span>Bonjour</span>
   <span>Comment vas tu ?</span>
</span>
```

Si l'élément sur lequel vous appliquez le rôle `presentation` n'a pas l'obligation de contenir des éléments enfants, les enfants continueront à être restitués. Dans l'exemple suivant, un élément de liste `<ul>` contient obligatoirement des enfants `<li>`, le rôle `presentation` s'appliquera donc aux `<li>` enfants mais pas aux éventuelles balises contenues dans les `<li>` (ici un élément `<a>`) qui restera donc restitué.
Exemple :

```
<ul role="presentation">
   <li><a href="xxx">Bonjour</a></li>
   <li>Comment vas tu ?</li>
</ul>
```

sera équivalent à :

```
<span>
   <span><a href="xxx">Bonjour</a></span>
   <span>Comment vas tu ?</span>
</span>
```

Des rôles de « Landmarks » permettant de définir des zones de navigation (cf. spécifications)

Par exemple :

banner

Pour désigner l'en-tête d'un site web.

complementary

Pour désigner une zone de contenu complémentaire.

main

Pour désigner la zone de contenu principale.

navigation

Pour désigner la zone de navigation.

search

Pour désigner la zone de recherche.

Exemple de structure de page utilisant les *landmarks*

```
<div role="banner">entête
      <form role="search">…</form>
</div>
<ul role="navigation">
   <li>…</li>
   <li>…</li>
</ul>
<div role="main">…</div>
<div role="complementary">…</div>
```

Enfin, des rôles de composants d'interface simples (cf. spécifications)

Par exemple :

button

Pour désigner un bouton.

dialog

Pour désigner une fenêtre modale.

progressbar

Pour désigner une barre de progression.

slider

Pour désigner une glissière.

tooltip

Pour désigner une info-bulle.

ou plus complexes car pouvant eux même contenir d'autres composants d'interface

combobox

Pour désigner une liste s'affichant en dessous d'un champ texte (pour des suggestions de saisie par exemple) et qui sera reliée à une zone ayant le rôle `listbox`.

tablist

Pour désigner une barre d'onglets qui contiendra des `tabs` eux-mêmes reliés à un `tabpanel`.
Exemple de système d'onglets :

```
<div id="tabs">
	<h2 id="tabsDemoLbl">Exemple d'onglets</h2>
	<ul role="tablist">
		<li role="presentation">
			<a href="#tabsdemo-1" aria-controls="tabsdemo-1" role="tab" aria-selected="true"
tabindex="0" id="tabsdemo-1-tab">Onglet 1</a>
		</li>
		<li role="presentation">
			<a href="#tabsdemo-2" aria-controls="tabsdemo-2" role="tab" aria-selected="false"
tabindex="-1" id="tabsdemo-2-tab">Onglet 2</a>
		</li>
		<li role="presentation">
			<a href="#tabsdemo-3" aria-controls="tabsdemo-3" role="tab" aria-selected="false"
tabindex="-1" id="tabsdemo-3-tab">Onglet 3</a>
		</li>
	</ul>
	<div id="tabsdemo-1" role="tabpanel" aria-hidden="false" aria-expanded="true" aria-
labelledby="tabsdemo-1-tab">
		<h2>Onglet 1</h2>
		<p>contenu de l'onglet 1</p>
	</div>
	<div id="tabsdemo-2" role="tabpanel" aria-hidden="true" aria-expanded="false" aria-
labelledby="tabsdemo-2-tab">
		<h2>Onglet 2</h2>
		<p>contenu de l'onglet 2</p>
	</div>
	<div id="tabsdemo-3" role="tabpanel" aria-hidden="true" aria-expanded="false" aria-
labelledby="tabsdemo-3-tab">
		<h2>Onglet 3</h2>
		<p>contenu de l'onglet 3</p>
	</div>
</div>
```

tree

Pour désigner un arbre dépliable qui contiendra des `treeitem`.
Exemple avec le menu "minéral" replié et le menu "végétal" déplié :

```
<h1 id="treelabel">Arbre dépliable</h1>
<ul role="tree" aria-labelledby="treelabel" aria-activedescendant="zoneayantlefocus"
tabindex="0">
	<li role="treeitem">Animal</li>
	<li role="treeitem" aria-expanded="false">Minéral
		<ul role="group">
			<li role="treeitem">Diamant</li>
			<li role="treeitem">Charbon</li>
		</ul>
```

```
    </li>
     <li role="treeitem" aria-expanded="true">Végétale
            <ul role="group">
                <li role="treeitem" id="zoneayantlefocus">Tomate</li>
                <li role="treeitem">Laitue</li>
            </ul>
     </li>
</ul>
```

Les propriétés et les états

Les propriétés et les états correspondent à des attributs comme l'attribut `id` ou `class` sur un élément HTML. Ce sont des valeurs que l'on vient appliquer à un élément ayant ou non un rôle ARIA pour lui permettre de renvoyer des informations spécifiques.
Par exemple, on peut établir une relation entre deux éléments en utilisant `aria-describedby`.

```
<img alt="chien" aria-describedby="mydog" src="monchien.jpg" />
<div id="mydog">
    ici la description complète de mon chien
</div>
```

La différence entre les états et les propriétés tient au fait que la valeur d'un état va être modifiée dans le temps en fonction des actions de l'utilisateur. Par exemple, une case à cocher pourra avoir un état non coché (`aria-checked="false"`) qui sera modifié ultérieurement via javascript pour être passé à coché (`aria-checked="true"`).

```
<span id="check" role="checkbox" aria-checked="false">Choix 1</span>
```

Les propriétés et les états peuvent êtres utilisés dans quatre contextes différents.

Sur des composants d'interfaces (cf. spécifications)

aria-autocomplete

Pour indiquer si un input type text dispose d'un mécanisme d'autocomplétion.

aria-checked (état)

Pour désigner si un élément est coché.

aria-haspopup

Pour indiquer si un élément a un menu contextuel.

aria-required

Pour indiquer si un champ est un champ obligatoire.

aria-invalid (état)

Pour indiquer si un champ est en erreur.

Exemple

Champ de formulaire avant validation javascript :

<label for="codepostale">Code postal *</label><input type="text" value="" id="codepostale" aria-required="true" aria-invalid="false"/>

Le même champ après validation javascript :

<label for="codepostale">Code postal *</label><input type="text" value="" id="codepostale" aria-required="true" aria-invalid="true"/>

Sur des zones « live » (cf. spécifications)

Les zones « lives » sont des parties de page dont le contenu est mis à jour via javascript alors que le focus de l'utilisateur n'est pas dans la zone en question. Par exemple, un cours de bourse mis à jour en direct, les résultats d'un match de foot en direct.
Dans ce contexte, les propriétés et états ARIA peuvent être les suivants.

aria-atomic

Pour indiquer si lors d'une mise à jour l'ensemble de la zone doit être relu ou juste la partie mise à jour.

aria-busy (état)

Pour indiquer si la zone est en cours de mise à jour.

aria-live

Pour indiquer qu'une zone sera mise à jour toute seule.

aria-relevant

Pour indiquer le type de mise à jour que le lecteur d'écran doit annoncer (les ajouts, les suppressions, les modifications de texte ou tout).

Exemple pour un cours de bourse live

Cours de la bourse : +3 %

Lors de la mise à jour via javascript :

Cours de la bourse :

Après la mise à jour :

Cours de la bourse : + 3.5 %

Sur des éléments pouvant être glissé/déposé (cf. spécifications)

aria-dropeffect

Pour indiquer le type d'opération qui sera effectué quand un objet sera relâché sur sa destination.

aria-grabbed (état)

Pour indiquer si un élément est en train d'être glissé.

Exemple

Image qui est indiquée comme étant glissable et dont le nœud DOM sera copié dans la div lors du dépôt :

Image en train d'être glissée et dont le nœud DOM sera copié dans la div lors du dépot :

Image déposée sur la zone :

Sur des éléments à mettre en relation (cf. spécifications)

aria-controls

Pour indiquer qu'un élément est la zone permettant le contrôle d'une autre.

aria-describedby

Pour indiquer qu'un élément est décrit par le contenu d'une autre zone.

aria-labelledby

Pour indiquer qu'un élément est titré par le contenu d'une autre zone.

Exemple d'une barre d'outil

description du fonctionnement du menu

```
<ul aria-describedby="xxx" role="menubar" aria-controls="yyy">
    <li role="menuitem">ouvrir</li>
    <li role="menuitem">édition</li>
</ul>
<div id="yyy">partie du contenu sur lequel s'applique les actions du menu</div>
```

Quand et comment utiliser ARIA ?

Si vos composants d'interfaces peuvent être décrit uniquement avec du HTML natif, préférez le HTML natif. Dès lors que ce n'est pas le cas, notamment parce qu'il n'existe pas de balise pour décrire sémantiquement votre composant ou que la balise existante est mal ou non supportée par les navigateurs, alors ARIA a sa place.
Ne modifiez pas la sémantique native des éléments (sauf à de très rares exceptions).
Il sera par exemple préférable de faire :

```
<a href="xxx">lien</a>
```

à :

```
<span role="link">lien</span>
```

De plus, il ne faut pas oublier que l'ajout de rôle ARIA ne supprime pas les comportements classiques des navigateurs.
Ainsi, un `button` que l'on modifie pour en faire un titre :

```
<button role="heading">titre</button>
```

… reste cliquable, atteignable au clavier et ressemble toujours à un bouton alors qu'il sera restitué comme un titre et donc non signalé comme cliquable.

Comment l'utiliser sur votre site web ?

Comme vous avez pu le voir dans les exemples illustrant les parties précédentes, ARIA est relativement simple à implémenter dans du HTML. Il s'emploie comme des attributs html classique. Sur le pur plan formel, il est nécessaire de savoir que pour l'heure, lors de l'ajout d'ARIA dans du code HTML 4.01 ou XHTML, cela génère des erreurs de validation.
Ce n'est pas le cas lorsque vous utilisez le doctype HTML5 :

```
<!DOCTYPE html>
```

Il faut toutefois noter qu'il est préférable de générer via javascript les rôles et les états amenés à être modifiés lors de l'utilisation de la page car sinon, en l'absence de javascript, cela pourrait générer des soucis pour l'utilisateur de lecteur d'écrans.
De plus, il est important de comprendre deux choses :

1. Aucune présentation visuelle ou comportement javascript n'est automatiquement ajouté par l'ajout d'ARIA. Par exemple, si vous voulez avoir une zone « *live* » avec le cours de la bourse, vous pourrez utiliser `role="region" aria-live="polite"` mais cela ne va pas pour autant mettre à jour la zone automatiquement. Il restera nécessaire de

développer les scripts javascript pour le faire.

2. ARIA ne gère pas non plus le comportement au clavier ; il sera toujours nécessaire de coder via javascript l'accès et l'utilisation au clavier des composants d'interface. Ainsi lors de l'ouverture d'une fenêtre modale (`role="dialog"`), il sera toujours nécessaire de coder en javascript le fait que la modale reçoive le *focus* au clavier. Pour cela ARIA précise que vous pouvez utiliser l'attribut `tabindex` : lorsqu'il a une valeur à 0 il permet à l'élément de recevoir le *focus* ; lorsqu'il a une valeur à -1 l'élément peut recevoir le *focus* au clavier uniquement via javascript.

En fonction du type de composants d'interface que vous utilisez sur votre site, il est par ailleurs recommandé de suivre les recommandations du W3C sur l'ergonomie de ceux-ci. En effet, si chaque développeur choisi de proposer des raccourcis ou un mode de navigation au clavier différents, cela risque d'être rapidement très dommageable pour l'utilisateur.
Par exemple, pour un système d'onglets il est recommandé que lors du parcours clavier, à l'aide de la touche tabulation, seul l'onglet actif sur la barre d'onglet puisse recevoir le focus alors que les autres onglets seront atteignables via les flèches gauche et droite. De plus, Ctrl+PageDown ou Ctrl+PageUp doivent permettre, lorsque le *focus* clavier est dans le contenu de l'onglet actif, d'aller à l'onglet précédent ou à l'onglet suivant.

Et mon application web dans tout ça ?

Du point de vue ARIA, il existe un rôle spécifique, `role="application"`, qui permet d'avertir l'utilisateur qu'il n'est pas dans un document mais dans une application. Une application sera généralement constituée d'un ensemble de composants d'interface comme par exemple un gestionnaire de mail. Dès lors, le lecteur d'écran et le navigateur essaieront de se comporter comme avec une application traditionnelle. Essaieront ? Oui, car c'est alors à vous de prévoir l'ensemble des comportements associés à ce fonctionnement.
Par exemple, si vous utilisez le `role="application"` et que vous avez des titres `h1`, `h2`, `h3`, etc. dans vos pages. Le mécanisme, existant pour un document web, permettant de naviguer de titre en titre ou d'avoir la liste des titres de la page, ne fonctionnera plus. Il sera donc nécessaire de coder via javascript un moyen de naviguer simplement de composant d'interface en composant d'interface.
Autant dire que l'usage de ce rôle est à manier avec prudence.

Sans rien faire, c'est magique avec les librairies javascript

Si vous êtes un développeur javascript débutant (ou simplement par commodité), il est probable que vous utilisiez ce qu'on appelle des librairies javascript comme Jquery, Dojo, YUI ou autre. A l'heure actuelle, Dojo dispose du meilleur support d'ARIA, les composants de Jquery UI ou de YUI intègrent partiellement ARIA sur certains d'entre eux. Vous n'auriez donc rien à faire pour ceux-là. Néanmoins, la documentation à ce sujet est très disparate voire inexistante, il est donc très difficile de savoir précisément ce qui est fait et ce qui ne l'est pas sans aller analyser le code.
Il est tout de même utile de citer certains travaux dans le domaine, notamment sur Jquery avec http://hanshillen.github.com/jqtest/.

Aller plus loin

ARIA et HTML5

Comme vous avez pu le voir, ARIA dispose de nombreux points commun avec HTML5. Ils permettent tous deux d'ajouter de la sémantique par rapport au XHTML ou à HTML4.01.
HTML5 propose pour cela l'ajout de nouvelle balise comme `<article>`, `<header>`, `<nav>`, `<section>`, etc.
Pour l'heure, bien qu'implémentant certaines de ces nouvelles balises, les navigateurs ne restituent pas pour autant leur valeur sémantique aux lecteurs d'écrans. Dès lors, et en attendant que cela soit le cas, il est tout à fait possible d'utiliser ARIA pour pallier à cette déficience.

Exemple de structure HTML5

```
<header role="banner">…</header>
<nav role="navigation">…</nav>
<section role="region">
   <article role="article">…</article>
   <aside role="complementary">
      <form>
         <label for="fname">Prénom *</label>
         <input type="text" aria-required="true" required id="fname" name="fname" />
      </form>
   </aside>
</section>
<footer role="contentinfo">…</footer>
```

ARIA et SVG

Comme nous l'indiquions au début de l'article, ARIA peut tout à fait être utilisée en dehors de HTML. Afin d'illustrer cela, voici deux exemples d'utilisation dans du svg.

Exemple de case à cocher via svg

À noter, l'exemple suivant ne gère pas l'accès au clavier.

```
<svg version="1.1" xmlns="http://www.w3.org/2000/svg"
   width="600px" height="400px">
   <title>checkbox svg avec ARIA</title>
   <g onclick="toggle('cb1')">
      <circle id="cb1" r="20" cy="100" cx="100"
            role="checkbox"
            aria-checked="true"
            aria-labelledby="text1" />
      <text id="text1" y="105" x="140">svg circle checkbox 1</text>
   </g>
   <g onclick="toggle('cb2')">
      <circle id="cb2" r="20" cy="200" cx="100"
```

```
            role="checkbox"
            aria-checked="false"
            aria-labelledby="text2" />
      <text id="'text2" y="205" x="140">svg circle checkbox 2</text>
   </g>
   <g onclick="toggle('cb3')">
      <circle id="cb3" r="20" cy="300" cx="100"
            role="checkbox"
            aria-checked="false"
            aria-labelledby="text3" />
      <text id="text3" y="305" x="140">svg circle checkbox 3</text>
   </g>
   <style type="text/css">
      circle {
         fill: white;
         stroke: black;
         stroke-width: 5px;
      }
      circle[aria-checked=true] {
         fill: red;
      }
      text {
         font: 20px sans-serif;
      }
   </style>
   <script type="text/javascript">//<![CDATA[
      function toggle(id) {
         var checkbox = document.getElementById(id),
            checked = checkbox.getAttribute('aria-checked') == "true";
         checkbox.setAttribute('aria-checked', checked ? "false" : "true");
      }
      //]]>
   </script>
</svg>
```

Exemple de tableau de données en svg avec ARIA

```
<svg version="1.1" xmlns="http://www.w3.org/2000/svg"
    width="420" height="110">
   <title>Tableau de données en SVG + ARIA</title>
   <rect x="10" y="10" width="400" height="30" fill="#FFFFCC" />
   <rect x="10" y="10" width="100" height="90" fill="#FFFFCC" />
   <path d="M10,10 l0,90 l400,0 l0,-90 l-400,0
         m100,0 l0,90 m100,0 l0,-90 m100,0 l0,90
         m100,-60 l-400,0 m0,30 l400,0
         M10" fill="none" stroke="#999999" />
   <g role="grid">
```

```
      <text role="row" x="0" y="32">
         <tspan role="columnheader" x="65">Trimestre</tspan>
         <tspan role="columnheader" x="160">Ventes</tspan>
         <tspan role="columnheader" x="262">Dépenses</tspan>
         <tspan role="columnheader" x="360">Résultat</tspan>
      </text>
      <text role="row" x="30" y="62">
         <tspan role="rowheader" x="20">T1</tspan>
         <tspan role="gridcell" x="204">223,00 €</tspan>
         <tspan role="gridcell" x="304">195,00 €</tspan>
         <tspan role="gridcell" x="400">28,00 €</tspan>
      </text>
      <text role="row" x="30" y="92">
         <tspan role="rowheader" x="20">T2</tspan>
         <tspan role="gridcell" x="204">183,00 €</tspan>
         <tspan role="gridcell" x="304">70,00 €</tspan>
         <tspan role="gridcell" x="400">113,00 €</tspan>
      </text>
   </g>
   <style type="text/css">
      text {
         font: 16px sans-serif;
         text-anchor: end;
      }
      tspan[role=columnheader],
      tspan[role=rowheader] {
         font-size: 18px;
         font-weight: bold;
         fill: #900;
         text-anchor: middle;
      }
      tspan[role=rowheader] {
         text-anchor: start;
      }
   </style>
</svg>
```

Conclusion

Comme vous avez pu le constater, si certains aspects sont simples au premier abord, cela se complique légèrement dès que allez devoir faire appel à javascript et gérer l'usage du clavier. Il n'en reste pas moins que c'est une solution d'ores et déjà utilisable (en prenant soin de tester le résultat évidemment) et qui permet une amélioration progressive non négligeable notamment sur HTML5.
En bref, ARIA est - comme toute nouvelle spécification - à utiliser, mais à utiliser correctement en sachant ce que l'on fait ;) et j'espère qu'en cela cet article vous aura été utile.

Ressources

Pour finir voici quelques liens qui pourraient vous être utiles si vous cherchez à en savoir plus sur ARIA et sur son utilisation :

Les spécifications

- http://www.w3.org/TR/wai-aria/
- http://www.w3.org/TR/wai-aria-practices/

Les démos ou exemples

- http://hanshillen.github.com/jqtest/
- http://www.oaa-accessibility.org/examples/
- http://www.paciellogroup.com/blog/misc/ddfrench/WAI-ARIA_DragAndDrop_French.html

Les ressources indispensables sur le sujet

- http://www.paciellogroup.com/blog/?s=%22HTML5+accessibility+chops%22
- http://html5accessibility.com/
- http://caniuse.com/wai-aria
- http://yaccessibilityblog.com/library/tag/aria
- http://www.marcozehe.de/category/aria/

À propos de l'auteur

Après une formation initiale de graphiste, Aurélien a complété ses connaissances par une formation de concepteur réalisateur multimédia à l'Institut Supérieur des Arts. En janvier 2004, il devient l'un des premiers "experts Accessiweb en évaluation". Il a depuis rejoint Temesis en 2008 en tant qu'expert accessibilité puis directeur général. Il a notamment co-rédigé le nouveau Référentiel Général d'Accessibilité pour les Administrations de la DGME dans le cadre du décret de la loi Handicap (février 2005) et animé plusieurs conférences dans le domaine de l'accessibilité (INOP, RMLL, réunions des experts du Groupe de Travail Accessiweb, Paris-web, TechDays). Il a contribué à la prise en compte de l'accessibilité dans des groupes comme la DILA, La Poste, LCL, Sciences Po. A titre personnel, il est également un des correspondants en France du WaSP-ILG ce qui l'amène à organiser régulièrement des W3Cafés.

- @goetsu
- http://www.fairytells.net/