GOBIERNO DE ESPAÑA
MINISTERIO DE INDUSTRIA, TURISMO Y COMERCIO
plan avanza2
INTECO
Instituto Nacional de Tecnologías de la Comunicación

GUÍA DE WAI ARIA

Centro de Referencia en Accesibilidad y Estándares Web

Agosto 2010

ÍNDICE

ÍNDICE 3

1. OBJETIVO DE LA GUÍA 4

2. WAI ARIA 5

2.1. ¿Qué es? 5

2.2. ¿Cómo funciona? 5

2.3. Ventajas y desventajas 5

3. INTRODUCCIÓN TÉCNICA 7

3.1. ¿Por qué WAI ARIA? 7

3.2. Partes de WAI ARIA 8

3.2.1. Roles 8

3.2.2. Estados y propiedades 10

3.2.2.1. Atributos Widget 11

3.2.2.2. Atributos de regiones activas 11

3.2.2.3. Atributos de Drag and Drop 12

3.2.2.4. Atributos de relaciones 12

3.2.3. Acceso mediante teclado 13

3.2.4. Pasos básicos para la implementación de WAI ARIA 14

3.3. Soporte en navegadores y productos de apoyo 20

3.4. Herramientas para desarrolladores 22

4. ANEXO 1 LISTADO DE ROLES, PROPIEDADES Y ESTADOS 24

4.1. Roles 24

4.2. Estados y propiedades 27

4.3. Listado de roles con las propiedades y estados admitidas 30

1. OBJETIVO DE LA GUÍA

A lo largo del tiempo las páginas web han sufrido cambios en cuanto a su objetivo y estructura. Mientras que el lenguaje HTML (Hyper Text Markup Language) fue concebido para la presentación de documentos con hiperenlaces, han surgido nuevas necesidades que han convertido estas páginas en documentos más complejos y más parecidos a las aplicaciones de escritorio.

Se ha incorporado una nueva tipología de contenido mucho más dinámico, caracterizado por la presentación y las actualizaciones de datos del lado del cliente con tecnologías como AJAX (Asyncronous JavaScript And XML). El W3C (World Wide Web Consortium), atendiendo a tales necesidades, ha propuesto una vía para conseguir que este tipo de tecnología sea accesible; su propuesta es WAI ARIA (Web Accessibility Initiate Accessible Rich Internet Application).

El objetivo de esta guía es demostrar las posibilidades de WAI ARIA en las aplicaciones web enriquecidas con contenido dinámico. Se explicarán sus diferentes partes y posibilidades de implementación y algunos casos de estudio que ejemplifiquen su funcionamiento.

2. WAI ARIA

2.1. ¿QUÉ ES?

WAI ARIA es una iniciativa del W3C que responde a la necesidad de proporcionar accesibilidad a las aplicaciones web enriquecidas. Pretende ser una ayuda en el contenido dinámico de los interfaces actuales de las aplicaciones web, que cada día son más parecidos a los entornos de escritorio y que su propio funcionamiento puede interferir en los productos de apoyo.

Esta tecnología tiene como principal objetivo aportar información acerca de las diferentes partes que constituyen los contenidos dinámicos generados, normalmente, por medio de *scripts*. Toda esta información será utilizada por los productos de apoyo para la interacción con el usuario final.

Los documentos técnicos de WAI ARIA están desarrollados por el grupo de trabajo del W3C PFWG (Protocols and Formats Working Group). El W3C pone a disposición varios documentos sobre WAI ARIA, entre ellos, destaca la especificación técnica que se encuentra alojada en la siguiente dirección: http://www.w3.org/TR/wai-aria/.

2.2. ¿CÓMO FUNCIONA?

WAI ARIA proporciona una serie de atributos que funcionan como identificadores de las diferentes partes de la aplicación que interactúa con el usuario. También se incluyen mapeo de controles y eventos para la accesibilidad de las APIs (Application Programming Interfaces).

WAI ARIA dispone de roles que describen tanto los *widgets* (componentes con funcionalidad propia de las interfaces de escritorio o web) de la aplicación como la estructura de la página web, como por ejemplo: los encabezados y las regiones. También dispone de varias propiedades como los estados de los *widgets*, las regiones activas de actualización de contenidos y sobre características *drag-and-drop*. A su vez, provee una manera de navegar mediante teclado dentro de los componentes.

2.3. VENTAJAS Y DESVENTAJAS

Al usar esta tecnología se encuentran una serie de ventajas:

- Al añadir valor semántico al contenido y a los *widgets* se puede situar al usuario exactamente en donde está. Además de su facilidad de implementación ya que son los atributos los que los define.

- Las actualizaciones de contenido dinámico, normalmente realizado con tecnologías de *script,* son notificadas al usuario por medio de su producto de apoyo.

- Accesibilidad de los *widgets* por medio del teclado.

- Se dota de información de cómo se utiliza un *widget* y qué tipo de datos proporciona.

En cambio, se puede identificar una desventaja manifiesta, ya que al aplicar esta tecnología, los documentos web que se desarrollen no validarán tanto en HTML 4 como en XHTML 1.0. El primero no soporta espacio de nombres, por lo que no se podrá incluir los atributos específicos de WAI ARIA. Con XHTML 1.0, su validación no sería posible porque no estaría incluido en el esquema que define al lenguaje.

Una posible solución sería incluir los atributos de WAI ARIA mediante DOM, es decir, incluir con JavaScript, dinámicamente, los atributos en los elementos destinados a ellos. Dicha solución haría dependiente de la tecnología *script* cualquier tipo de implementación.

Existe otra solución que podría resolver el problema: la utilización de XHTML 1.1. Este lenguaje es una especificación de lenguaje modular y extensible que permitiría personalizar el DTD incorporando el esquema de WAI ARIA.

3. INTRODUCCIÓN TÉCNICA

3.1. ¿POR QUÉ WAI ARIA?

HTML fue creado para la presentación de documentos textuales con formatos específicos para su función, como puedan ser los encabezados, marcadores de párrafos o cualquier otro tipo de elementos que tuvieran que ver con el formateo de texto. Esto es así desde que en las primeras versiones se incluyó el elemento `IMG` para la incorporación de elementos de imágenes dentro de los documentos.

El lenguaje fue desarrollado para un modelo de cliente-servidor, donde el usuario pedía el documento y desde un servidor se lo proporcionaba. Esta comunicación está orientada de forma secuencial, es decir, un usuario envía una petición a un servidor, normalmente de un documento HTML, y el servidor lo procesa para, finalmente, enviarlo al usuario final.

Con la evolución de la web, las páginas han ido evolucionando hacia las llamadas aplicaciones web, mucho más parecidas a los entornos de escritorio que tienen un funcionamiento más dinámico con respecto a los contenidos. Para ello, se han utilizado tecnologías web como JavaScript, y más concretamente AJAX, que de unos años hasta hoy ha tenido un crecimiento sustancial.

Con estas técnicas, al hacer operaciones y procesamientos desde el lado del cliente, se afecta de forma notable a la accesibilidad de los sitios web, ya que los productos de apoyo no reconocen los componentes o *widgets* insertados.

Estas tecnologías pueden cargar datos sin tener que actualizar toda la página web, es decir, existen regiones activas que envían peticiones al servidor en un segundo plano. Muchos productos de apoyo no reconocen estas áreas activas o, simplemente, no pueden reconocer que había información nueva, ya que la propia página no había sido actualizada por completo desde el lado del servidor.

Existen en HTML componentes de interfaz, como los existentes para los formularios, que son insuficientes para la gran cantidad de funcionalidades aparecidas en los últimos años. Para ello se han recurrido a técnicas de *scripting* o de objetos para suplir esta carencia. También la presentación o el estilo de los mismos ha sido, en ocasiones, modificado para que fuese más atractivo. Es el caso de los *radiobuttons* en HTML que, en muchas ocasiones, son personalizados en *sliders* con varios estados o posiciones que apuntan a un determinado valor.

También está presente el problema de la independencia de dispositivo, ya que algunos componentes de las aplicaciones web enriquecidas no son accesibles desde el teclado, orientando su uso al ratón. Esta característica entraña un grave problema de accesibilidad, dado que muchas personas con discapacidad física son incapaces de utilizar el ratón.

3.2. PARTES DE WAI ARIA

WAI ARIA propone a los desarrolladores una serie de soluciones destinadas a hacer accesibles *widgets*, áreas activas y demás componentes enriquecidos que se encuentran en la mayoría de las aplicaciones web en la actualidad.

Para ello describen roles y propiedades con la finalidad de dotar de información a los productos de apoyo y para que interactúen adecuadamente con los componentes más normales de las aplicaciones web.

Los roles pueden ser de dos tipos.

- Los primeros describen el tipo de *widget* del que se trata. Los *widgets* pueden ser de varios estilos, desde menús desplegables en forma de árbol, hasta *sliders,* pasando por barras de medición de progreso.

- El segundo tipo de rol es aquel que define la estructura de la página web. Estos roles pueden definir elementos de encabezado o incluso tablas o *grid*, es muy común, también, definir diferentes regiones.

Las propiedades pueden describir el estado de los propios *widgets*: un *widget* que funcione como un *checkbox* puede tener un estado de *checked,* por ejemplo. También se utilizan para definir regiones activas donde se actualizarán contenidos proporcionando políticas de aviso de actualización. Otra funcionalidad de las propiedades de WAI ARIA es aquella destinada a describir los *drag-and-drop,* definiendo las fuentes *drag* y los objetivos *drop*.

3.2.1. Roles

Como se ha comentado antes, los roles dentro de WAI ARIA sirven para describir los *widgets,* así como para definir elementos propios de las páginas web. Estos atributos proporcionan información a los productos de apoyo sobre los componentes *widgets*.

Es evidente que, por definición, las etiquetas de HTML tienen un rol predefinido, el mismo que el propio elemento indica. Un rol de WAI ARIA ayuda a concretar la funcionalidad de la etiqueta en la que va definida y se podría afirmar que la sustituye. A continuación se ve un ejemplo en forma de código:

```
<ul role="toolbar" tabindex="0">
  <li id="boton1">Copiar</li>
  <li id="boton2">Pegar</li>
  <li id="boton3">Cortar</li>
</ul>
```

El código anterior representa un *widget* sencillo. En este caso el desarrollador ha querido representar una barra de herramientas con las tres funcionalidades más populares de la

edición de texto, como son Cortar, Pegar y Cortar. Se intuye que estas opciones están dotadas de una funcionalidad mediante *script*, como pueda ser JavaScript. Por defecto, el elemento `<UL>` tiene de por sí un significado propio como etiqueta de lista. En el contexto de la página web se sabe que tendrá una función muy determinada, como barra de herramientas, ahí entra en valor el atributo de WAI ARIA *role* que viene a renombrar su definición concretándola aún más, en este caso con el valor *toolbar*. Existen otros roles predefinidos como *log*, *progressbar* o *timer* (se puede encontrar la lista completa en la siguiente dirección: http://www.w3.org/TR/wai-aria/roles o en el Anexo 1).

Los roles también pueden definir regiones de un documento web. Es común poder identificar partes sustanciales de una página web, más allá de los predefinidos por las etiquetas de HTML dentro de un nivel de precisión más alto. Los roles *Landmark* son un tipo que permite dotar de un significado a ciertas regiones que tienen una funcionalidad o propósito bien marcado y relevante para el uso de cualquier usuario. Las regiones destinadas a la navegación o a la búsqueda de términos pueden ser consideradas como *Landmarks.* Algunos tipos de roles *Landmarks* son por ejemplo: *contentinfo*, *form*, *banner* o *search*.

Existen otros tipos de roles que definen estructuras dentro del documento o página web. No suelen representar elementos interactivos con los usuarios y un ejemplo pueden ser los valores como *group*, *region* o *row*.

También se pueden encontrar roles llamados abstractos, pero no se han de utilizar por parte de los desarrolladores web ya que sirven para completar la ontología en la especificación WAI ARIA.

Para cualquier desarrollador web puede resultar fácil implementar este atributo en una página web bien estructurada. Es común proporcionar nombres semánticos a ciertos elementos en los atributos *id* o *class*. Estos atributos dan pistas del propósito del contenido de dichos elementos. Se verá con el siguiente ejemplo:

```
<div id=”navegacion”>
      <! – contenido de navegacion -- >
</div>
<div id=”contenidoPrincipal”>
      <!—contenido principal -- >
</div>
<div id=”anuncios”>
<!—contenido anuncios-- >
</div>
```

En este ejemplo podemos ver tres bloques `<DIV>` cuyos atributos *id* poseen nombres semánticos. Este tipo de atributos pueden servir para guiarnos e incluir atributos *role* de WAI ARIA en el documento HTML. Se verá como quedaría el código:

```
<div id=”navegacion” role=”navigation”>
      <! – contenido de navegacion -- >
</div>
```

```
<div id=”contenidoPrincipal” role=”main”>
      <!—contenido principal -- >
</div>
<div id=”anuncios” role=”banner”>
<!—contenido anuncios-- >
</div>
```

En este ejemplo de código se han añadido los atributos de WAI ARIA *role* que aportan un valor semántico que se ha recogido de los atributos *id*. Se verá que el <DIV> con *id* navegación puede corresponder con un *role* de tipo *navigation*, y, como a su vez, el <DIV> con *id contenidoPrincipal* puede corresponder al role con valor *main*. Aunque ésta no es una técnica fidedigna, y siempre se ha de corresponder con el criterio de los desarrolladores, aporta pistas y muestra la visibilidad de la correspondencia semántica de algunos atributos y su aprovechamiento para las técnicas de WAI ARIA.

3.2.2. Estados y propiedades

WAI ARIA define diferentes tipos de propiedades. Las propiedades pretenden ser una guía a los productos de apoyo que proporciona información de cómo interactuar con ciertos *widgets* que se encuentran en una página web.

A diferencia de los roles, en el que sólo existe un atributo *role* y varios valores (*navigation*, *menu*, *search*, etc…), los atributos de los estados y propiedades pueden ser varios, y cada uno puede aceptar varios valores (se pueden ver los estados y valores en la siguiente página web: http://www.w3.org/TR/wai-aria/states_and_properties o en el Anexo 1). Por ejemplo, la propiedad *aria-dropeffect* puede tener los siguientes valores:

- copy:
- move:
- link:
- execute:
- popup
- none:

Cada valor está relacionado con el tipo de funcionalidad que tenga el *widget* de efecto de tipo *drop*.

Existen estados y propiedades de carácter global, es decir, pueden ser utilizados independientemente del elemento o del tipo de rol que tenga. Pero en la mayoría de los casos la clasificación de los estados y propiedades viene distribuida de la siguiente forma:

- Atributos *Widget*
- Atributos de regiones activas
- Atributos de *Drag and Drop*
- Atributos de relaciones

3.2.2.1. Atributos Widget

Estos atributos son específicos para las interfaces de usuario más comunes. Están orientados a componentes en los que el usuario ha de introducir una entrada de datos para procesarla. Los atributos **Widget** también están pensados para apoyar los roles de los *widgets*. A continuación se verá un ejemplo de este tipo de atributos:

```
<label for="NombreYApellidos"> Nombre y Apellidos
<input       type="text"       id="NombreYApellidos"       name="
NombreYApellidos" aria-required="true" />
</label>
```

Es muy común encontrarse en los formularios campos que son obligatorios. Normalmente se indican con un carácter que implícitamente se encuentra al lado del campo. Para que los productos de apoyo reconozcan este tipo de campo, hay que explicitar la semántica. Con WAI ARIA y el atributo de *widget* `aria-required` podemos identificar claramente que el campo es obligatorio, aplicándole un valor *booleano* `true`.

3.2.2.2. Atributos de regiones activas

Los atributos destinados a las regiones activas, es decir, aquellas susceptibles de incorporar contenido dinámico actualizable, pueden ser aplicados a cualquier elemento. Su principal objetivo es informar de que estas áreas pueden actualizarse aún no teniendo el foco activo. Se verá a continuación en un ejemplo:

```
<p aria-live="polite">
<! - - contenido del párrafo - - >
</p>
```

En el código anterior se puede observar que al elemento <P> se le ha añadido un atributo de región activa `aria-live` con el valor `polite`. Un producto de apoyo que reconozca WAI ARIA sabrá que el anterior párrafo será un área activa que podrá ser actualizada con contenido. El valor del atributo `polite` le indicará que el usuario podrá responder al nuevo contenido una vez haya acabado las tareas que esté haciendo en ese momento. Si se diera el caso de que el valor del atributo `aria-live` fuera `assertive` indicaría que la región activa

es mucho más prioritaria que con *polite* pero que no sería necesario interrumpir las tareas actuales del usuario.

3.2.2.3. Atributos de Drag and Drop

Son atributos que proporcionan información sobre los efectos *Drag and Drop* que se encuentran en algunas aplicaciones, como por ejemplo, desplazar objetos por la página web. De esta forma el objetivo del elemento a mover *Drop* es reconocido por los productos de apoyo que utilicen las personas con discapacidad. Se verá un ejemplo a continuación:

```
<div role="menuitem" aria-dropeffect="copy move">
```

El elemento <DIV> tiene un atributo *role* con valor *menuitem*, este rol permite insertar la propiedad *aria-dropeffect* que indica el tipo de acción que acepta este elemento. El valor pueden ser varios, en el caso del ejemplo, puede aceptar tanto el efecto copiar como cortar.

3.2.2.4. Atributos de relaciones

Estos atributos tienen la función de relacionar elementos que no se puede establecer mediante la estructura del documento. A continuación se verá un ejemplo:

```
<h1 id="titulo1">Mi foto cuando tenía 3 años</h1>
<p id="descripcion">Cuando tenía tres años todavía tenía el pelo
de color castaño claro, media…</p>
<div><img src="yopequeño.jpg" alt="yo cuando tenía tres años"
aria-labelledby="titulo1"          aria-describedby="descripcion"
/></div>
```

En este ejemplo de código tenemos tres elementos HTML que tienen un significado bien definido. En primer lugar se encuentra un encabezado de primer nivel <H1> con una breve reseña de lo que se va a encontrar dentro del documento, es decir, una foto de un niño de 3 años. A continuación se tiene un párrafo <P> en el que se describe con más detalle el contenido de la imagen que se encuentra al final del documento <IMG>.

Como se puede observar, tanto el elemento de encabezado como el de párrafo están identificados con un atributo *id*. Se tiene claro que ambos elementos poseen una relación con el elemento imagen por lo que podemos aprovechar su *id* como valor a los atributos WAI ARIA: *aria-describedby* y *aria-labelledby* que se encuentran en <IMG>.

El atributo *aria-labelledby* tiene como valor el *id* del elemento que ejerza como función de etiqueta, en el presente caso, el encabezado puede tener perfectamente dicha función. Mientras que con el atributo *aria-describedby* se puede identificar una descripción más larga de el elemento. En el caso, el párrafo puede ocupar dicha funcionalidad.

3.2.3. Acceso mediante teclado

Para la accesibilidad en general, y la web en particular, es imprescindible poder acceder a la información independientemente del dispositivo que se utilice. Actualmente existen muchos complementos o *widgets* que no pueden ser accedidos por medio del teclado. Normalmente existen objetos que son diseñados exclusivamente para interactuar con el ratón, lo que supone un fallo grave de accesibilidad. Personas con grave discapacidad física, que sólo pueden manejar el teclado o un conmutador no podrán manejar este tipo objetos.

No todos los elementos de HTML obtienen el foco donde el usuario interactúa, pero estos elementos son utilizados para desarrollar *widgets*. Los elementos de bloque como `<DIV>` no pueden captar el foco, por lo que la funcionalidad establecida no podrá ser accedida mediante teclado, más concretamente por tabulación.

Los elementos que tienen foco pueden ser controlados a partir del atributo `tabindex`. Para estos elementos, si no se indica el valor del *`tabindex`* o tienen valor 0, el orden de tabulación corresponderá al orden lógico de las estructura del documento. Podemos variar el orden de tabulación asignando números al atributo mencionado.

WAI ARIA permite implementar el atributo `tabindex` a cualquier elemento que se encuentre y que se utilice para una funcionalidad que requiera el foco. Dependiendo del valor que tome cambiará su actuación. A continuación se expone los posibles valores de *`tabindex`* y su comportamiento en el ámbito de los widgets:

- Sin especificar: mediante ratón y *JavaScript* sigue el comportamiento normal del elemento sólo para elementos de formulario y anclas. Desde la navegación sigue el comportamiento del elemento por defecto.

- `tabindex=0`: Obtiene el foco mediante el ratón o JavaScript y mediante la navegación por tabulación va obteniendo el foco en el orden del código del documento.

- *`tabindex=numero positivo hasta 32768`*: Recibe el foco mediante ratón o JavaScript y mediante teclado el número asigna el orden de tabulación.

- `tabindex=-1`: Alcanza el foco mediante ratón o *JavaScript*. Mediante tabulación no se puede alcanzar, consiguiéndolo mediante *element.focus()* como resultado de la pulsación de una tecla o flecha.

WAI ARIA también específica la forma en cómo obtienen el foco los hijos de los componentes *widgets*. Existe el atributo *`aria-activedescendant`* que permite la activación del foco en los elementos hijos de un *widget*.

3.2.4. Pasos básicos para la implementación de WAI ARIA

Uno de los objetivos de WAI ARIA es evitar modificar la forma en que los desarrolladores implementan las páginas o aplicaciones web. Más bien, intenta ser un complemento en sus desarrollos que añada semántica a los documentos para que puedan ser interpretados correctamente por los productos de apoyo.

Se pueden definir una serie pasos para implementar WAI ARIA en un documento web ya desarrollado. Se realizará un ejemplo para mostrar cual serían las etapas lógicas en la implementación de WAI ARIA en un documento web.

```
<div id="frontal">
      <h1 id="titulo1">Papelería González y Hermanos</h1>
      <p id="descripcion">Una papelería especialistas en
productos de ofimática e impresión digital</p>
      <div><img src="papeleria.jpg" alt="Escaparate de Papelería
González y Hermanos" /></div>
</div>
<div>
      <ul id="navegacion">
            <li><a href="index.php">Inicio</a></li>
            <li><a href="tarifas.php">Tarifas</a></li>
            <li><a href="contacto.php">Contacto</a></li>
      </ul>
</div>
<div>
      <form id="busqueda" method="get" action="busqueda.php">
            <label for="campoBusqueda">Búsqueda
                  <input type="text" id="campoBusqueda"
name="campoBusqueda"/>
                  </label>
                  <input type="submit" value="Buscar" />
            </form>
</div>
<div id="contenidoPrincipal">
<h2 id="titulo2">Tarifas</h2>
<!-- contenido de Tarifas -- >
<div id="ofertas"></div>

</div>
<div id="anuncios">
      <img src="anuncio1.jpg" alt=" " />
</div>
<div id="pie">
      <ul id="menuSecundario">
            <li>&copy;2010 Papelerías González y Hermanos
<acronym title="Sociedad Limitada">S.L.</acronym></li>
            <li><a href="privacidad.php">Privacidad</a></li>
            <li><a href="aviso.php">Aviso Legal</a></li>
      </ul>
</div>
```

En el ejemplo de código anterior tenemos una página web correspondiente a una sección del sitio de la empresa *Papelería González y Hermanos*. Después de una primera observación de la estructura del documento se verá que la página web tiene las regiones típicas:

- Un frontal con el nombre de la empresa, una imagen y una descripción de la misma.
- Una zona de navegación con enlaces a las diferentes partes del sitio web.
- Un formulario de búsqueda de términos.
- La región del contenido principal de la sección.
- Una zona de anuncios.
- Un pie de página web con enlaces sobre el copyright, privacidad y aviso legal.

Todas estas regiones tienen un significado concreto que puede sacarse del valor del `id` que ha elegido el desarrollador, o por el propio contenido. WAI ARIA dispone de roles que añaden valor semántico explícito. Como se ha visto anteriormente, existen roles que identifican regiones de la estructura del documento.

El elemento de lista desordenada cuyo `id` es `navegacion` puede corresponder semánticamente al valor del atributo `role, navigation`. Se podría incluir dicho atributo con ese valor en el elemento `<UL>` que contiene los enlaces correspondientes a las secciones del sitio web.

Otro tanto ocurre con el formulario de búsqueda, ya que en WAI ARIA existe un valor para atributo `role` que es `search`. Sería conveniente poner dicho atributo en la etiqueta `<FORM>` correspondiente a la búsqueda de texto en el sitio web, que le añadiera valor semántico explícito.

En casi todos los sitios web existe una región donde se incluye el contenido principal de una determinada sección, es lo que ocurre en el ejemplo. El elemento `<DIV>` cuyo `id` es `contenidoPrincipal` puede atribuirse dicha funcionalidad. En WAI ARIA existe el valor `main` del atributo role que identifica este tipo de región, por lo que sería acertado incluirlo en el elemento de bloque genérico.

Se puede identificar una zona de anuncios donde contiene imágenes de promoción de la empresa. Entre los posibles valores que pueda tener el atributo `role`, existe uno específico para los anuncios que es `banner`, por lo que sería correcto aplicarle dicho valor al elemento de bloque.

Con lo que respecta al elemento de bloque cuyo *id* es *pie*, podemos observar que contiene un listado donde el *id* tiene valor de *menuSecundario*. Aunque los ids no correspondan con un nombre que coincida semánticamente con el contenido, sí que se puede observar que los enlaces pertenecen al *copyright* o a las políticas de privacidad. En WAI ARIA estos conceptos pueden incluirse dentro del valor del atributo *role*, *contentinfo*. En el ejemplo se pondría este valor dentro del elemento <UL>.

Existe un valor genérico para atributo *role* llamado *region* destinado a aquellas regiones cuyo significado no puede corresponder a los especificados en WAI ARIA. Se recomienda que, de aplicarse, se incluyera el atributo *title* con un nombre apropiado. En el ejemplo anterior el elemento <DIV> de atributo *id* con valor *ofertas* podría contener un *role* cuyo valor fuera *region* y a su vez indicarle un *title* con el valor *ofertas de la papelería*.

También se pueden extraer relaciones entre diversos elementos en una página web. En el frontal se puede extraer un ejemplo de esta característica. Se podría considerar el encabezado <H1> la etiqueta de la imagen contenida dentro del frontal. A su vez el párrafo cuyo *id* es descripción podría ser una explicación breve de la Papelería. Aplicando la técnica de WAI ARIA se incluirían los dos atributos de propiedades *aria-labelledby* para la etiqueta y *aria-describedby* para la descripción. Sus valores corresponderían a los ids de los elementos que ejercen la función determinada.

En el ejemplo se encuentra un elemento <DIV> vacío cuyo atributo *id* tiene como valor *ofertas*. Este <DIV> carga dinámicamente contenido a través de tecnologías como AJAX. WAI ARIA propone una solución anteriormente explicada, se debería aplicar el atributo *aria-live*. El valor del atributo depende del tipo de actualización que se haga, es decir, el nivel de prioridad o de atención que debe prestarle el usuario. En el caso del ejemplo, el elemento actualiza ofertas de la empresa, por lo que el nivel de atención del usuario será aquel que el usuario precise, por lo que se podría incluir un valor *polite*.

El ejemplo anterior quedaría codificado con WAI ARIA de la siguiente forma:

```
<div id="frontal">
      <h1 id="titulo1">Papelería González y Hermanos</h1>
      <p id="descripcion">Una papelería especialistas en
productos de ofimática e impresión digital</p>
      <div><img src="papeleria.jpg" alt="Escaparate de Papelería
González y Hermanos" id="imagen" aria-labelledby="titulo1" aria-
describedby="descripcion"/></div>
</div>
<div>
      <ul id="navegacion" role="navigation">
            <li><a href="index.php">Inicio</a></li>
            <li><a href="tarifas.php">Tarifas</a></li>
            <li><a href="contacto.php">Contacto</a></li>
      </ul>
</div>
<div>
```

```
      <form id="busqueda" method="get" action="busqueda.php"
role="search">

            <label for="campoBusqueda">Búsqueda
                  <input type="text" id="campoBusqueda"
name="campoBusqueda"/>
                  </label>
                  <input type="submit" value="Buscar" />

</form>
</div>
<div id="contenidoPrincipal" role="main">
<h2 id="titulo2">Tarifas</h2>
<!- - contenido de Tarifas - - >
<div id="ofertas" role="region" aria-live="polite"></div>

</div>
<div id="anuncios" role="banner">
      <img src="anuncio1.jpg" alt=" " />
</div>
<div id="pie">
      <ul id="menuSecundario" role="contentinfo">
            <li>&copy;2010 Papelerías González y Hermanos
<acronym title="Sociedad Limitada">S.L.</acronym></li>
            <li><a href="privacidad.php">Privacidad</a></li>
            <li><a href="aviso.php">Aviso Legal</a></li>
      </ul>
</div>
```

En resumen, al documento se le ha añadido información semántica muy concreta para el uso de productos de apoyo. Es muy importante identificar los *widgets* de las aplicaciones web, así como, establecer la estructura adecuadamente. Se muestra el ejemplo anterior de la barra de herramientas:

```
<ul role="toolbar" tabindex="0">
  <li id="boton1">Copiar</li>
  <li id="boton2">Pegar</li>
  <li id="boton3">Cortar</li>
</ul>
```

En el ejemplo ya se había identificado el atributo role del *widget* mediante el valor *`toolbar`*. El elemento `<UL>` puede adquirir el foco ya que hemos incluido un atributo *`tabindex`* como se explicó para el acceso mediante teclado. Mediante el DOM se puede especificar los elementos que son contenidos dentro del *widget*. En el caso, tres elementos de lista cuyas funciones principales son representar los botones para copiar, pegar y cortar.

Como los elementos tienen funcionalidad de botones, se podría incluir el atributo *`role`* con valor *`button`*. De esta forma los productos de apoyo lo identificarían como tales.

Para que este *widget* fuera accesible mediante teclado sería conveniente incluir el atributo *aria-activedescendant* en el elemento <UL>. El valor del atributo corresponde al valor del *id* del elemento hijo al que se quiera centrar el foco. De esta forma se podría acceder mediante teclado a los diferentes botones de la barra de herramientas. En el caso del ejemplo *aria-activedescendant* tomaría el valor del botón de copiar, *boton1*. Mediante JavaScript se podrían asignar eventos de teclado que cambiarían el valor del atributo por los ids subsiguientes. El ejemplo anterior quedaría de la siguiente forma:

```
<ul role="toolbar" tabindex="0" aria-activedescendant=”boton1”>
  <li id="boton1" role=”button”>Copiar</li>
  <li id="boton2" role=”button”>Pegar</li>
  <li id="boton3" role=”button”>Cortar</li>
</ul>
```

También se considera importante sincronizar los estados o propiedades de WAI ARIA con las características de presentación de las aplicaciones web. Los efectos visuales, normalmente, se considerarán mediante CSS. Se pueden encontrar *widgets* como los elementos de árbol que puedan variar su color de fondo dependiendo si el elemento está seleccionado o no. Por JavaScript se podría variar el CSS de un elemento si su *id* coincide con el valor *aria-activedescendant* del elemento padre del *widget*.

Otro ejemplo claro sucede con los elementos que se ocultan. Existe un atributo *aria-hidden* que indica si los elementos están ocultos o no. Se debería corresponder el valor de los atributos con el CSS correspondiente. Así, si el valor es *true*, se debería aplicar el CSS indicado para elementos ocultos, y al contrario, si fuera *false*, tomaría el valor de CSS correspondiente a cuando está visible.

Con respecto a los formularios que podemos encontrar en una aplicación Web, son varios los factores a tener en cuenta al incluir WAI ARIA. Se destaca la existencia de dos propiedades en WAI ARIA que son *aria-required* y *aria-invalid*. Con estas propiedades se puede indicar si un elemento del formulario es obligatorio o si es inválido respectivamente. A continuación se muestra un ejemplo:

```
<form name=”form1” id=”form1” action=”datos.php” method=”post”>

            <label for=”nombre”>Nombre:
            <input type=”text” name=”nombre” id=”nombre” />
            </label>
            <label for=”apellidos”>Apellidos:
            <input  type=”text”  name=”apellidos”  id=”apellidos”
/>
            </label>
            <label for=”telefono”>Teléfono:
            <input type=”text” name=”telefono” id=”telefono” />
            </label>
            <label for=”correo”>Correo:
            <input type=”text” name=”correo” id=”correo” />
            </label>
```

```
   <input type="submit" value="Enviar" />
</form>
```

El anterior formulario corresponde a la introducción de datos personales de una persona. Muchos de estos datos serán obligatorios pero, a veces, en los formularios se incluyen campos opcionales como el del correo electrónico. En este caso se podrían introducir atributos de WAI ARIA como *aria-required* para identificar los campos obligatorios. De esta forma los productos de apoyo podrían identificar que campos son obligatorios y avisar al usuario. El formulario quedaría de la siguiente forma:

```
<form name="form1" id="form1" action="datos.php" method="post">

            <label for="nombre">Nombre:
            <input type="text" name="nombre" id="nombre" aria-
required="true"/>
            </label>
            <label for="apellidos">Apellidos:
            <input type="text" name="apellidos" id="apellidos"
aria-required="true"/>
            </label>
            <label for="telefono">Teléfono:
            <input type="text" name="telefono" id="telefono"
aria-required="true" />
            </label>
            <label for="correo">Correo:
            <input type="text" name="correo" id="correo" />
            </label>
      <input type="submit" value="Enviar"/>
</form>
```

Un aspecto muy importante es el de dotar de semántica a los elementos *Drag and Drop*. Primeramente, se debería identificar con roles aquellos componentes que sean *Drag and Drop* ya que podemos tener elementos genéricos en HTML que se comporten de esta manera. Si un elemento es *Drag and Drop*, existe un estado, *aria-grabbed,* en WAI ARIA que identifica si está "agarrado" de forma *booleana.*

Cuando tengamos los elementos identificados debemos marcar el objetivo de arrastrar dichos elementos. Se debe indicar, en el elemento adecuado, el atributo *aria-dropeffect* encargado de definir el tipo de operación que se realizará, y que se ha comentado anteriormente, como son copiar, mover, enlazar, ejecutar o lanzar un menú *popup*.

El W3C conserva un documento en esta dirección: http://www.w3.org/TR/wai-aria-practices/ donde se explica con más detalle este tipo de prácticas así como se exponen ejemplos aclaratorios.

Como se ha explicado, una forma de incluir los atributos WAI ARIA es mediante JavaScript. Una técnica muy utilizada en lenguajes de *script* es acceder al DOM de la página e incluir los atributos dinámicamente. A continuación se muestra un ejemplo:

```
$(document).ready(function() {
$('#imagen').attr('aria-describedby', 'descripcion');
$('#imagen').attr('aria-labelledby', 'titulo1');
$('#navegacion').attr('role', 'navigation');
$('#busqueda').attr('role', 'search');
$('#contenidoPrincipal').attr('role', 'main');
$('#ofertas').attr('role', 'region');
$('#ofertas').attr('aria-live', 'polite');
$('#anuncios').attr('role', 'banner');
$('#menuSecundario').attr('role', 'contentinfo');
});
```

En el ejemplo se muestra la utilización de la librería JQuery para incluir en determinados elementos atributos WAI ARIA, en concreto corresponden al primer ejemplo de esta sección. De esta forma el documento web validaría, ya que los atributos son incluidos de forma dinámica.

Es muy común también utilizar librerías JavaScript para incluir *widgets* ya desarrollados. Existen librerías que implementan los atributos de WAI ARIA en muchos de sus componentes, a continuación se ofrece un listado con distintas librerías JavaScript que incluyen WAI ARIA en todas o en algunas de sus funcionalidades:

- *BBC Glow Widgets*
- *Dojo*
- *EXTJS*
- *Fluid Infusion*
- *Google Web Toolkit*
- *JQuery*
- *YUI*

3.3. SOPORTE EN NAVEGADORES Y PRODUCTOS DE APOYO

Normalmente las aplicaciones o productos de apoyo se comunican con las páginas web tradicionales mediante el DOM (*Document Object Model*). Del DOM obtienen la estructura jerárquica de la página así como la semántica de la misma, de ésta forma los productos de apoyo saben donde existen párrafos, imágenes o acrónimos.

El DOM también es utilizado para procesar objetos del documento mediante API's (*application programming interface*) para generar elementos propios de las aplicaciones enriquecidas.

Cuando las páginas web no están enriquecidas, los productos de apoyo solamente necesitan utilizar el DOM de los documentos para identificar cada objeto, si éstos están debidamente etiquetados. Cuando se añaden características enriquecidas se deben hacer corresponder la semántica de los objetos con el Accessibility API de las plataformas. El Accessibility API indica la forma que debe actuar el producto de apoyo en la plataforma donde está ejecutándose. Cuando se utiliza WAI ARIA, la semántica proporcionada debe ser asignada por la correspondiente Accessibility API para realizar la acción adecuada. Existen varios Accessibility API's:

- *Microsoft Active Accessibility (MSAA)*
- *IAccessible2*
- *User Interface Automation* (*UIA*)
- *Linux Accessibility Toolkit (ATK) and Assistive Technology - Service Provider Interface (AT-SPI)*
- *Mac OS X Accessibility Protocol*

Entre las características a tener en cuenta para que los productos de apoyo identifiquen bien las opciones relativas a la accesibilidad en las aplicaciones enriquecidas, se destaca la de activar la navegación por teclado de los componentes. Es importante incluir `tabindex` o `aria-activedescendant` para indicar el foco de los elementos, o cuáles de los elementos hijo obtiene el foco en un cambio del mismo.

Es clave que la semántica de WAI ARIA corresponda correctamente a un modo de comportamiento definido en el Accessibility API de la plataforma. De esta forma la asignación correcta entre la semántica de WAI ARIA y los comportamientos del Accessibility API corresponderá en un funcionamiento óptimo de los productos de apoyo. Las capacidades de cada *widget* serán definidas en los roles, estados o propiedades que se hayan especificado.

El W3C dispone de un documento técnico donde explica con detalle la asignación de las diferentes características de WAI ARIA con las Accessibility API más comunes. http://www.w3.org/TR/wai-aria-implementation/.

Entre los productos que soportan la implementación de WAI ARIA se encuentran varios navegadores y productos de apoyo, así como herramientas de desarrollo:

- A partir de Firefox 2.0

- Internet Explorer desde el DOM API (A partir de Internet Explorer 8)
- Opera
- GW Windows Eyes 5.5 con soporte para Firefox
- Jaws 7.0 con soporte de Firefox
- Windows Magnifier con soporte tanto para IE como para Firefox
- NVDA Reader
- ZoomText
- Fire Vox
- Orca Screen Reader

3.4. HERRAMIENTAS PARA DESARROLLADORES

Existe una herramienta útil para desarrolladores que examina áreas activas, roles y propiedades de WAI ARIA llamada *Juicy Studio Accesibility ToolBar*, una extensión para el navegador Firefox.

Esta extensión se divide en 4 herramientas:

- *Live Regions*
- *Aria*
- *Table Inspector*
- *Colour Contrast Analyzer*

Las dos herramientas orientadas a WAI ARIA son *Live Regions* y *Aria*. Con *Live Regions* se podrá identificar las áreas activas que son incluidas dinámicamente y que cambian el estado del documento. Con la herramienta *Aria* se tendrá tres opciones:

- *Document Landmarks*
- *Roles*
- *Roles and Properties*

Con la herramienta *Document Landmarks* se podrá identificar visualmente en la página los roles *Landmark* que dotan de significado a ciertas regiones dentro de un documento.

Con la opción de *Roles* se podrá generar un informe en forma de tabla con el listado de roles utilizados en la página, los elementos en donde están incluidos, los elementos padre y el código utilizado. Con *Roles and Properties* se tendrá un mismo informe que el anterior incluyendo las propiedades WAI ARIA contenidas en el documento.

4. ANEXO 1 LISTADO DE ROLES, PROPIEDADES Y ESTADOS

4.1. ROLES

Se definirán las funciones de los roles existentes en el siguiente listado (http://www.w3.org/TR/wai-aria/roles):

- `alert`: Se trata de un mensaje con algún tipo de importancia que es dependiente del tiempo.
- `alertdialog`: Es un tipo de diálogo que advierte sobre un asunto.
- `application`: Es una región considerada aplicación en contra de otras partes que son de tipo documento web.
- `article`: Es una sección del documento cuya composición forma una parte independiente del documento.
- `banner`: Región en la que se incluye algún tipo de publicidad o anuncio.
- `button`: Es una entrada de datos que mediante la pulsación o la activación genera una acción o evento.
- `checkbox`: es una entrada con tres tipos de estados: true, false o mixed.
- `columnheader`: Es una celda que contiene información de la columna.
- `combobox`: Tipo de selección de texto en forma de lista.
- `complementary`: Es una sección del documento que complementa a una principal del mismo.
- `contentinfo`: Es una región que contiene información sobre el documento padre.
- `definition`: Definición de un concepto.
- `dialog`: Es una ventana de aplicación que interrumpe el proceso actual para proporcionar información o atender a una respuesta necesaria.
- `directory`: Es una lista de miembros de un grupo.

- `document`: Es una región que contiene información del documento en contra de lo definido en `application`.
- `form`: Es una región que contiene elementos propios de un formulario.
- `grid`: Región que contiene celdas de datos tabulares.
- `gridcell`: Es una celda de una región `grid`.
- `group`: Serie de objetos correspondientes a una interfaz de usuario.
- `heading`: Región que especifica la cabecera de un documento.
- `img`: Región que contiene una serie de elementos que componen una imagen.
- `link`: Es una referencia interactiva hacia un recurso interno o externo que hace que el producto de apoyo que lo active se redirija a el mismo.
- `list`: Es un grupo de elementos no interactivos.
- `listbox`: Es un componente que permite elegir entre varias opciones.
- `listitem`: Es un elemento perteneciente a una lista o a un directorio
- `log`: Es un tipo de región activa que va proporcionando información actualizada desde la más nueva a la más antigua.
- `main`: Región donde se encuentra el contenido principal del documento.
- `marquee`: Es un tipo de región activa que actualiza información relativamente importante.
- `math`: Se trata de un elemento que representa una expresión matemática.
- `menu`: Es un tipo de componente que ofrece una serie de opciones.
- `menubar`: Es una región normalmente horizontal que representa a un menú.
- `menuitemcheckbox`: Es un elemento de menú que puede tener tres valores: true, false o mixed.

- *menuitemradio*: Es un elemento de un menú de los cuáles sólo uno puede ser activado.
- *navigation*: Es un grupo de elementos para la navegación del documento.
- *note*: Se trata de una sección que corresponde a una nota o un anexo con respecto a una parte del documento principal.
- *option*: Es un elemento que puede ser elegido entre varios.
- *presentation*: Es un elemento cuyo significado nativo no corresponde al accessibility API.
- *progressbar*: Se trata de un elemento que tiene una barra de progreso que muestra el tiempo de proceso de una tarea.
- *radio*: Es un elemento perteneciente a un grupo cuyos elementos tienen su mismo rol y que sólo uno de ellos puede ser activo.
- *radiogroup*: Es un grupo de elementos de tipo *radio*.
- *region*: Sección del documento con la suficiente importancia para reconocerse como una región específica.
- *row*: columna de un elemento *grid*.
- *rowgroup*: Se trata de un grupo que contiene una o varias columnas.
- *rowheader*: Celda que contiene la información sobre una columna.
- *scrollbar*: Es un objeto que controla la vista de un área mayor que la visible.
- *search*: Es una zona específica para ejecutar búsquedas.
- *separator*: Es un objeto que divide contenido dentro de una región.
- *slider*: Es una entrada de datos para la elección de un rango determinado de valores.
- *spinbutton*: Es un campo para elegir entre varios opciones, en concreto. dentro de un rango.

- `status`: Es una región con contenido de aviso al usuario sin que sea tan intrusivo como una alerta.
- `tab`: Un grupo de etiquetas con un mecanismo para acceder al contenido representado en cada `tab`.
- `tablist`: Es una lista de elementos `tab` que hacen referencia a los `tabpanel`.
- `tabpanel`: Es un contenedor de recursos asociado a un `tab`.
- `textbox`: Es una entrada de datos que permite cualquier texto como su valor.
- `timer`: Se trata de un contador numérico que indica la cantidad de tiempo desde un punto de inicio y el tiempo restante hasta un punto final.
- `toolbar`: Es un grupo de funciones de uso frecuente contenidas dentro de una región específica.
- `tooltip`: Es un *popup* descriptivo de un elemento.
- `tree`: Es un tipo de lista con listas enlazadas que pueden desplegarse u ocultarse.
- `treegrid`: Se trata de una celda cuya columna puede ser expandida u ocultada.
- `treeitem`: Es un elemento de un `tree` que puede ser expandido u ocultado si éste pertenece a un subnivel.

4.2. ESTADOS Y PROPIEDADES

Se definirán ahora los estados y propiedades que existen en WAI ARIA (http://www.w3.org/TR/wai-aria/states_and_properties):

- `aria-activedescendant`: Identifica el elemento activo descendiente de un *widget* complejo.
- `aria-atomic`: Indica si el producto de apoyo se requerirá de forma total o en parte dentro del componente.
- `aria-autocomplete`: Indica si un elemento de entrada es de tipo autocompletar.
- `aria-busy` (estado): Indica si una determinada región está actualizándose.

- `aria-checked` (estado): Indica si un elemento como un checkbox, radiobutton o cualquier componente que tenga estado de *checked* lo está o no.
- `aria-controls`: Identifica un *elemento* o varios cuyo contenido es controlado por el elemento actual.
- `aria-describedby`: Identifica el elemento o elementos que describen un objeto.
- `aria-disabled` (estado): Indica si un elemento está desactivado, no se puede editar o no es operable.
- `aria-dropeffect`: Indica que tipo de función es lanzada cuando el objeto seleccionado llega a su objetivo.
- `aria-expanded` (estado): Indica si un elemento o grupo de elementos están expandidos u ocultos.
- `aria-flowto`: Indica el próximo elemento del orden de lectura sobrescribiendo el que tendría el documento por defecto.
- `aria-grabbed` (estado): Indica si un elemento está "agarrado" en la operación *drag and drop*.
- `aria-haspopup` : Indica si un elemento tiene un *popup* o elementos de subnivel.
- `aria-hidden` (estado): Indica si el elemento es visible o no para el usuario.
- `aria-invalid` (estado): Indica si el valor de la entrada tiene un formato correcto o no al requerido.
- `aria-label` : Define el valor de una cadena que etiqueta a un elemento.
- `aria-labelledby`: Identifica a un elemento o varios que actúa de etiqueta sobre otro elemento.
- `aria-level`: Define el nivel jerárquico de un elemento dentro de una estructura.
- `aria-live`: Indica si un elemento tiene actualizaciones dinámicas, así como describe que tipos de actualizaciones.
- `aria-multiline`: Indica si una caja de texto admite una o varias líneas.

- `aria-multiselectable`: Indica si el usuario debe elegir uno o varios elementos de los descendientes de una opción.
- `aria-orientation`: Indica si el *scroll* de un elemento es orientado hacia la izquierda o derecha.
- `aria-owns`: Identifica un elemento o varios en orden a definir de forma visual, funcional o contextual la relación padre e hijo cuando la jerarquía del DOM no puede hacerlo.
- `aria-posinset`: Define el numero de un elemento en orden a una posición dentro de una estructura de listado o árbol.
- `aria-pressed` (estado): Indica si el estado de un botón está presionado o no.
- `aria-readonly` : Indica si un elemento es editable u operable.
- `aria-revelant` (estado): Indica que notificaciones de cambio monitorizará el producto de apoyo dentro de una región activa.
- `aria-required`: Indica si la entrada de un campo de un formulario es obligatoria o no.
- `aria-selected` (estado): Indica los actuales elementos seleccionados dentro de un *widget*.
- `aria-setsize` : Indica el número de elementos de un listado o un árbol.
- `aria-sort` : Indica si un elemento de una tabla está ordenado de forma ascendente o descendente.
- `aria-valuemax` : Define el valor máximo permitido en un rango de un *widget*.
- `aria-valuemin` : Define el valor mínimo permitido en un rango de un *widget*.
- `aria-valuenow` : Define el valor actual en un rango de un *widget*.
- `aria-valuetext` : Define el valor textual entendible por un hombre del valor actual en un rango de un *widget*.

4.3. LISTADO DE ROLES CON LAS PROPIEDADES Y ESTADOS ADMITIDAS

Existen propiedades y estados globales que admiten cualquier rol, se muestran en el listado siguiente:

- *aria-atomic*
- *aria-busy* (estado)
- *aria-controls*
- *aria-describedby*
- *aria-disabled* (estado)
- *aria-dropeffec*t
- *aria-flowto*
- *aria-grabbed* (estado)
- *aria-haspopup*
- *aria-hidden* (estado)
- *aria-invalid* (estado)
- *aria-label*
- *aria-labelledby*
- *aria-live*
- *aria-owns*
- *aria-relevant*

Seguidamente se especificarán los roles y las propiedades que admiten cada uno:

- *alert*
    - *aria-expanded* (estado)

- *alertdialog*
  - *aria-expanded* (estado)
- *application*
  - *aria-expanded* (estado)
- *article*
  - *aria-expanded* (estado)
- *banner*
  - *aria-expanded* (estado)
- *button*
  - *aria-pressed*
- *checkbox*
  - *aria-checked* (estado) obligatorio
- *columnheader*
  - *aria-short*
  - *aria-readonly*
  - *aria-required*
  - *aria-selected* (estado)
  - *aria-expanded* (estado)
- *combobox*
  - *aria-expanded* (estado) obligatorio
  - *aria-required*

    - *aria-activedescendant*

- *complementary*
    - *aria-expanded* (estado)
- *contentinfo*
    - *aria-expanded* (estado)
- *definition*
    - *aria-expanded* (estado)
- *dialog*
    - *aria-expanded* (estado)
- *directory*
    - *aria-expanded* (estado)
- *document*
    - *aria-expanded* (estado)
- *form*
    - *aria-expanded* (estado)
- *grid*
    - *aria-level*
    - *aria-multiselectable*
    - *aria-readonly*
    - *aria-activedescendant*
    - *aria-expanded* (estado)

- *gridcell*
  - *aria-readonly*
  - *aria-required*
  - *aria-selected* (estado)
  - *aria-expanded* (estado)
- *group*
  - *aria-activedescendant*
  - *aria-expanded* (estado)
- *heading*
  - *aria-level*
  - *aria-expanded* (estado)
- *img*
  - *aria-expanded* (estado)
- *list*
  - *aria-expanded* (estado)
- *listbox*
  - *aria-multiselectable*
  - *aria-required*
  - *aria-expanded* (estado)
  - *aria-activedescendant*
- *listitem*

  - *aria-level*
  - *aria-posinset*
  - *aria-setsize*
  - *aria-expanded* (estado)
- *log*
  - *aria-expanded* (estado)
- *main*
  - *aria-expanded* (estado)
- *marquee*
  - *aria-expanded* (estado)
- *math*
  - *aria-expanded* (estado)
- *menu*
  - *aria-expanded* (estado)
  - *aria-activedescendant*
- *menubar*
  - *aria-expanded* (estado)
  - *aria-activedescendant*
- *menuitemcheckbox*
  - *aria-checked* (estado) obligatorio
- *menuitemradio*

- *aria-checked* (estado) obligatorio
- *aria-posinset*
- *aria-selected* (estado)
- *aria-setsize*

- *navigation*
  - *aria-expanded* (estado)
- *note*
  - *aria-expanded* (estado)
- *option*
  - *aria-checked* (estado)
  - *aria-posinset*
  - *aria-selected* (estado)
  - *aria-setsize*
- *progressbar*
  - *aria-valuemax*
  - *aria-valuemin*
  - *aria-valuenow*
  - *aria-valuetext*
- *radio*
  - *aria-checked* (estado) obligatorio
  - *aria-posinset*

  - *aria-selected* (estado)
  - *aria-setsize*
- *radiogroup*
  - *aria-required*
  - *aria-activedescendant*
  - *aria-expanded* (estado)
- *region*
  - *aria-expanded* (estado)
- *row*
  - *aria-level*
  - *aria-selected* (estado)
  - *aria-activedescendant*
  - *aria-expanded* (estado)
- *rowgroup*
  - *aria-activedescendant*
  - *aria-expanded* (estado)
- *rowheader*
  - *aria-sort*
  - *aria-readonly*
  - *aria-required*
  - *aria-selected* (estado)

  - `aria-expanded` (estado)

- `search`
  - `aria-expanded` (estado)
- `separator`
  - `aria-expanded` (estado)
- `scrollbar`
  - `aria-controls` obligatorio
  - `aria-orientation` obligatorio
  - `aria-valuemax` obligatorio
  - `aria-valuemin` obligatorio
  - `aria-valuenow` obligatorio
  - `aria-valuetext`
- `slider`
  - `aria-valuemax` obligatorio
  - `aria-valuemin` obligatorio
  - `aria-valuenow` obligatorio
  - `aria-valuetext`
- `spinbutton`
  - `aria-valuemin` obligatorio
  - `aria-valuemax` obligatorio
  - `aria-valuenow` obligatorio

  - *aria-required*
  - *aria-activedescendant*
  - *aria-valuetext*
- *status*
- *aria-activedescendant*
  - *aria-expanded* (estado)
- *tab*
  - *aria-selected* (estado)
  - *aria-expanded* (estado)
- *tablist*
  - *aria-activedescendant*
  - *aria-expanded* (estado)
- *tabpanel*
  - *aria-expanded* (estado)
- *textbox*
  - *aria-autocomplete*
  - *aria-multiline*
  - *aria-readonly*
  - *aria-required*
- *timer*
  - *aria-activedescendant*

    - *aria-expanded* (estado)
- *toolbar*
    - *aria-activedescendant*
    - *aria-expanded* (estado)
- *tooltip*
    - *aria-expanded* (estado)
- *tree*
    - *aria-multiselectable*
    - *aria-required*
    - *aria-activedescendant*
    - *aria-expanded* (estado)
- *treegrid*
    - *aria-level*
    - *aria-multiselectable*
    - *aria-readonly*
    - *aria-activedescendant*
    - *aria-expanded*
- *treeitem*
    - *aria-level*
    - *aria-posinset*
    - *aria-setsize*

- `aria-expanded` (estado)
- `aria-checked` (estado)
- `aria-selected` (estado)