TESINA DE LICENCIATURA

Titulo: Accesibilidad y Crowdsourcing: Uso de etiquetas semánticas para mejorar la accesibilidad de aplicaciones web.
Autores: Martín Sebastián Zanotti
Director: Alejandra Garrido
Codirector: Sergio Firmenich
Asesor profesional: —
Carrera: Licenciatura en Sistemas

Resumen

Hoy en día la tecnología se integra y relaciona a la mayoría de las actividades de nuestra vida cotidiana pero aun existen dificultades para que la misma sea accesible para todos. Internet constituye la mayor fuente actual del conocimiento y su inaccesibilidad marca una de las mayores desventajas en la brecha digital [Toledo et al. 2005]*. Aquí propondremos un enfoque para mejorar la accesibilidad en las aplicaciones web centrado en beneficiar a aquellas personas que poseen una discapacidad visual. En general, la información en las páginas web se fragmenta y organiza en grupos visualmente reconocibles, lo cual sumado a un conjunto de conocimientos previos adquiridos por los usuarios, podemos identificar y comprender dicha información y el rol que cumple. Gracias a esto, los usuarios logran procesar rápidamente la información no por su contenido sino por su diseño y disposición, ayudándolos a ubicarse en el contexto de la aplicación web. Sin embargo esto no resulta igual de simple para usuarios con dificultades visuales quienes no poseen acceso a dicha información visual. Aquí nos enfocaremos en el desarrollo de un sistema que ayude a capturar esta información de las páginas web para hacerla accesible (mediante la transcodificación) a usuarios con discapacidad visual y ofrezca una plataforma de crowdsourcing como método para escalar esta solución a la gran cantidad de páginas web presentes en la internet.*

Palabras Claves

Accesibilidad, Usabilidad, Transcodificación, Refactoring, Crowdsourcing, Etiquetas semánticas, Objetos semánticos, Información semántica, Javascript, Inyección de dependencias, HTML, páginas web, Extensión de Chrome, Lector de pantalla, Algoritmo de similitud, reconocimiento de elementos similares, Pattern matching,

Conclusiones

El enfoque presentado en este trabajo permite agregar información semántica adicional rápidamente a las interfaces web de manera de estandarizar el reconocimiento de objetos semánticamente idénticos dentro de distintos códigos HTML. De esta forma herramientas que brindan mayor accesibilidad (como los lectores de pantalla) pueden hacer uso de la información adicional provista para ofrecer mejores resultados. El crowdsourcing cumple un rol fundamental para la generación de esta información adicional y es así como pretendemos que usuarios voluntarios no propietarios de los sitios web cooperen para brindar una mayor accesibilidad a los mismos.

Trabajos Realizados

-Investigación sobre experiencias de usuarios en el uso de lectores de pantalla, reconocimiento de factores problemáticos y propuesta de una solución.
-Creación e implementación de un algoritmo para el reconocimiento de elementos web similares.
-Desarrollo de un plugin de Chrome para seleccionar elementos web y vincular nueva información semántica, haciendo uso del algoritmo de similitud mencionado.
-Desarrollo de un plugin de Chrome para acceder al contenido de la página con la nueva información semántica.
-Desarrollo de un sistema de Crowdsourcing para un mayor alcance de la solución.

Trabajos Futuros

-Autoría colaborativa
-Mejoras de integración con otros lectores de pantalla.
-Implementaciones propuestas no concluidas.
-Implementación de un lector de pantalla propio.
-Refinamiento del algoritmo de similitud de objetos semánticos.
-Mejoras en las herramientas desarrollas y pruebas adicionales.

Fecha de la presentación: Mayo 2015

Tabla de contenido

Capitulo I
Introducción

1.1 Introducción

La tecnología se integra y relaciona a la mayoría de las actividades de nuestra vida cotidiana, pero aun existen dificultades para que la misma sea accesible para todos. Muchos usuarios deben operar en contextos muy distintos a los habituales: pueden poseer problemas visuales, auditivos o motrices, pueden no ser capaces de procesar cierto tipo de información, contar con tecnología limitada o precisar de programas especiales como los lectores de pantalla o hasta de un sistema operativo distinto.

La Accesibilidad web es el grado en el que todas las personas pueden hacer uso de la web [W3C]. Es el esfuerzo adicional realizado para permitir que la información no solo se encuentre disponible en la web, sino que pueda ser accedida por el mayor número de personas posible, teniendo en cuenta las limitaciones y condiciones específicas en las que se pueden encontrar (como las mencionadas anteriormente). Si bien la accesibilidad trata de contemplar todos los contextos posibles en los que se puede encontrar un usuario, nosotros nos limitaremos a aquellos en donde existe una inaccesibilidad a la información debido a la falta de capacidad visual por el usuario. Es por eso que a lo largo de este trabajo, cuando hablemos de la falta de accesibilidad web estaremos haciendo referencia a la inaccesibilidad presentada por este caso en particular.

Según la Organización Mundial de la Salud (OMS) en el año 2014 se calcula que hay aproximadamente 285 millones de personas con discapacidad visual en el mundo, de las cuales un 13,6% son ciegas y el resto presentan baja visión [OMS 2014]. En la Argentina, según la Encuesta nacional de Personas con Discapacidad realizada en el 2003 un 7,1% de la población Argentina posee alguna discapacidad de las cuales un 22% de los casos se trata de una discapacidad visual (aproximadamente 314.423 personas) [INDEC 2003]. Dado que no representan un gran sector de la población no suelen ser considerados normalmente en los desarrollos tecnológicos pero esto no implica un menor grado de importancia.

En la Argentina, la accesibilidad web esta protegida bajo la Ley 26.653 (Ley de accesibilidad de la información en las paginas web, reglamentada el 4/4/2013) [Senado y Cámara de Diputados de la Nación 2010], la cual establece:

"El Estado nacional, entiéndanse los tres poderes que lo constituyen, sus organismos descentralizados o autárquicos, los entes públicos no estatales, las empresas del Estado y las empresas privadas concesionarias de servicios públicos, empresas prestadoras o contratistas de bienes y servicios, deberán respetar en los diseños de sus páginas Web las normas y requisitos sobre accesibilidad de la información que faciliten el acceso a sus contenidos, a todas las personas con discapacidad con el objeto de garantizarles la igualdad real de oportunidades y trato, evitando así todo tipo

de discriminación… Se entiende por accesibilidad a los efectos de esta ley a la posibilidad de que la información de la página Web, puede ser comprendida y consultada por personas con discapacidad y por usuarios que posean diversas configuraciones en su equipamiento o en sus programas…"

A nivel internacional el W3C [W3C] (un grupo independiente que define los protocolos y estándares para la web) lanzó una de sus iniciativas llamada WAI (*Web Accessibility Initiative*) [W3C 1997], la cual estableció diferentes pautas o guías que explican cómo se tienen que crear las páginas web para que sean accesibles. Estas guías son llamadas WCAG (*Web Content Accessibility Guidelines*) y actualmente se encuentran en la versión 2.0. Estas pautas son las que generalmente suelen ser tomadas en cuenta por la mayoría de los organismos que buscan mejorar los niveles de accesibilidad.

Foco del trabajo

Generalmente, los creadores de páginas web tratan de presentar la mayor cantidad de información posible dentro de una misma página valiéndose del uso de estructuras y diseños visuales que permiten a los usuarios distinguir y reconocer fácilmente los distintos grupos de información presentada y el rol que cumple cada uno. Por ejemplo, en la Figura 1.1 se muestra una captura de pantalla del sitio web "Wikipedia". Al verlo rápidamente podemos distinguir las diferentes secciones que posee, sin la necesidad de leer su contenido, sino tan solo basándose en la información visual que percibimos. Podemos detectar una serie de enlaces a la izquierda, el articulo principal en el centro, información detallada a la derecha, un buscador en la esquina superior derecha, etc.

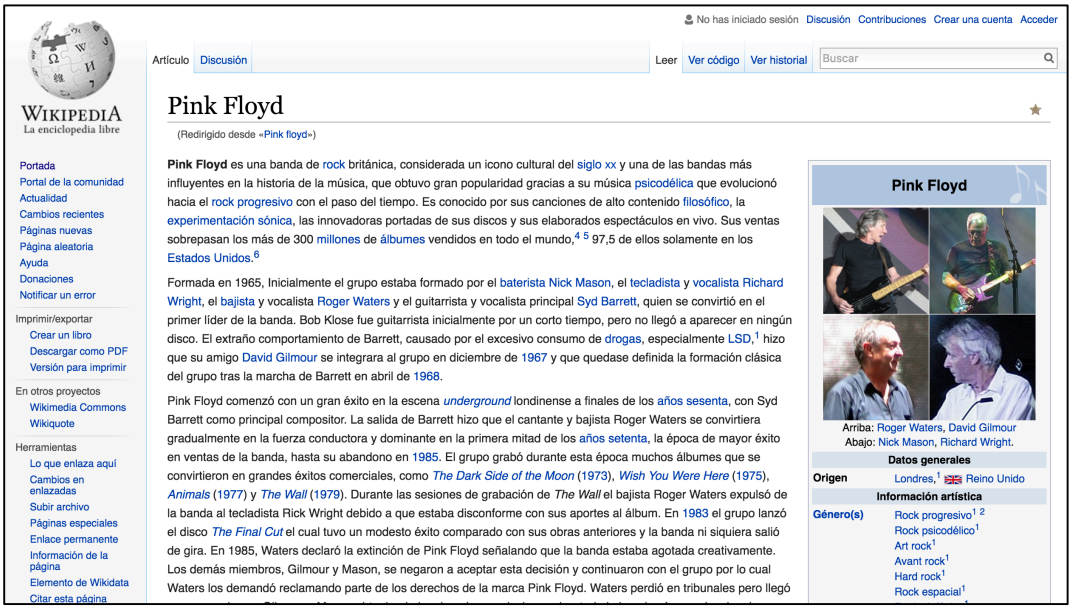


Figura 1.1: Captura de pantalla del sitio web "Wikipedia"

Esta información visual que recibimos anteriormente no se encuentra disponible para aquellos usuarios con dificultades visuales. Entonces, ¿Cómo hacer disponible esta información a usuarios con discapacidad visual?, esta será una de las principales problemáticas a resolver por el sistema a desarrollar dentro de este trabajo.

La accesibilidad web puede ser validada a través de herramientas de evaluación automáticas que ayudan a los programadores a identificar violaciones a las pautas detalladas en las WCAG (de las cuales hablaremos más adelante). Sin embargo, no todas las pautas pueden ser validadas automáticamente, varias de ellas requieren de un factor humano (como el sentido común sobre el orden en que los elementos son dispuestos en la web, o las descripciones de las imágenes), es por eso que no podemos confiar simplemente en una herramienta automática para considerar que una página web sea accesible. En la práctica, la mejor forma de garantizar que una página sea accesible es que realmente pueda ser utilizada por un usuario sin dificultad, si bien esto suena trivial, lo que queremos decir es que la usabilidad hace a la accesibilidad y viceversa, por ende más allá de las buenas prácticas existentes sobre accesibilidad, esta será realmente verdadera cuando verdaderos usuarios puedan utilizarla para afirmarlo.

Finalmente, la gran cantidad de páginas web existentes nos hace querer diseñar un sistema que permita escalar fácilmente a la mayor cantidad de páginas posibles, es por eso que investigamos en que consiste el modelo de Crowdsourcing para poder aplicarlo en nuestro contexto.

1.2 Limites del trabajo

Si bien la accesibilidad web abarca un gran conjunto de casos en los que los usuarios encuentran complicaciones a la hora de acceder a la web, este trabajo se enfoca a aquellos en donde los límites se encuentran establecidos por la carencia de la información visual. Los disminuidos visuales poseen una discapacidad para percibir toda la información gráfica provista por la web la cual existe en abundancia y es plenamente necesaria a la hora de la navegación.

Nos enfocaremos en investigar las alternativas existentes para los usuarios en estas condiciones, analizaremos sus defectos y contribuiremos con un aporte que mejore la experiencia de estos usuarios en la navegación web.

1.3 Problemas de la Accesibilidad Web

La interacción con las computadoras es realizada a través de una **interfaz**, esta comprende el **punto de contacto entre el humano y la información** dispuesta en los medios tecnológicos [María and Paz 2012]. Es por esto que las interacciones con dichas interfaces comprenden el principal objeto de estudio para la mejora de la accesibilidad.

Normalmente, durante la navegación web, los usuarios videntes no suelen leer todas las palabra presentes en las páginas para comprenderlas, sino que realizan saltos continuos, escaneando las distintas partes de las páginas para concluir si es importante o no, si se trata de la información que buscaban o si precisan dirigirse hacia otro sitio. Esto no es muy distinto en el caso de los usuarios con disminución visual, si bien ellos utilizan lectores de pantalla que leen el texto presente en las interfaces, son igual de impacientes a la hora de obtener lo que buscan lo más rápido posible. Al utilizar los lectores de pantalla no escuchan cada palabra de una página, escuchan solo las primeras palabras de un enlace o una línea de texto y si no les parece relevante se desplazan al siguiente [Peter Krantz 2005]. A pesar de esto, interactuar mediante el uso de un lector de pantalla trae consigo varios problemas en la navegación:

Falta de contexto: Al utilizar un lector de pantalla, el usuario puede perder la noción del contexto general en donde el puntero del lector se encuentra ubicado. El mismo continuará pronunciando lecturas de texto de las cuales el usuario ya no comprende de donde provienen. (Ejemplo: Wikipedia suele utilizar varios links dentro del contenido principal de los documentos. Si navegamos link a link usando lectores de pantalla, escucharemos palabras aisladas las cuales resultarán confusas y fuera de contexto. Si bien es cierto que los lectores suelen avanzar línea por línea o de a párrafos, aun así es muy fácil desorientarse y perder conocimiento de la ubicación del lector en la página web).

Lectura secuencial: Los comandos para la navegación pueden obligar al usuario a recorrer el contenido de la aplicación de manera secuencial, por lo tanto, es muy importante introducir mecanismos que faciliten la identificación precisa de las partes que comprender a la página y de dar mayor importancia a aquellas que contengan el contenido por el cual se esta ingresando. (Ejemplo: en los resultados de búsquedas, la lectura secuencial puede trazar un camino por publicidades y links innecesarios antes de llegar a los verdaderos resultados buscados).

Estos problemas representan tanto falta de accesibilidad como de usabilidad: El usuario posee acceso a gran parte de la información textual y/o sonora, pero carece de acceso a la información visual, la cual atenta en contra de la orientación y dificulta la navegación y el uso de la aplicación. Cuando la aplicación posee una baja usabilidad gran parte de la información no es accedida por falta de interpretación. Es decir, que la información tal vez pueda ser leída por un lector de pantalla, pero la falta de contexto y la dificultad para manejarlo atenta contra la interpretación de dicha información haciéndola prácticamente inaccesible.

Aunque la accesibilidad y la usabilidad se relacionan en gran medida, muchas veces son tratadas por separado. La accesibilidad tiene como objetivo hacer los sitios web disponibles para la mayor cantidad de personas posibles, mientras que la usabilidad se centra en mejorar la experiencia del usuario en el sitio, a modo de que esta sea más eficiente y satisfactoria. Como mencionamos anteriormente, problemas en la accesibilidad concluyen en problemas de usabilidad y viceversa. Por lo tanto, si bien nuestro objetivo es mejorar la accesibilidad de los usuarios a las páginas web, gran parte de ello lo lograremos mejorando la usabilidad de las mismas.

Por último, los programadores web a menudo no tienen conciencia sobre los estándares de accesibilidad, deciden ignorarlos o aplican conceptos vagos [Ball 2013], ya que consideran innecesaria la inversión de su tiempo y esfuerzo en lo que puede significar un público pequeño con necesidades particulares. Por lo tanto, existen muchas páginas web que no brindan una alternativa para la navegación de quienes deben utilizar lectores de pantalla.

1.4 Motivación

Este trabajo esta motivado por lo dificultoso que puede resultar el uso de lectores de pantalla en los diversos diseños de las millones de páginas web presentes en internet, proponiéndose buscar una solución que simplifique el uso de estas herramientas.

Hasta ahora hemos descripto una serie de problemas relacionados con la accesibilidad web en el uso de lectores de pantalla. ¿Cómo podemos relacionar todo esto?

Podemos llegar a la conclusión de que para mejorar la accesibilidad de los usuarios que utilizan lectores de pantalla, debemos mejorar no solo la información de las páginas web para hacerlas más accesibles, sino también mejorar su usabilidad con respecto a los lectores de pantalla [Garrido et al. 2014]. La usabilidad en los lectores podría mejorarse si logramos presentar la información al usuario de forma simplificada por medio del lector de pantalla, otorgando un panorama de lo que el sitio ofrece (tal como cuando escaneamos visualmente la página, sin entrar en detalle, sino para resolver nuestra "inquietud" de entender el contenido presente), luego dar la posibilidad al usuario de ingresar a cada sección simplificada para obtener el resto de la información más detallada. Por lo tanto podríamos ofrecer distintos niveles de abstracción sobre el contenido, de modo que el usuario obtenga mayor detalle solo si lo solicita. Por último, debemos tener en cuenta que esta accesibilidad debe otorgarse por un medio externo a lo ofrecido por las propias páginas web, dado que puede que sus propietarios tal vez nunca decidan mejorar la accesibilidad de la misma pero puede haber voluntarios que decidan hacerlo.

Para comprender un poco más esta idea daremos un ejemplo. Supongamos que un usuario no vidente que utiliza un lector de pantalla ingresa al sitio web Wikipedia y su lector de pantalla solo distingue las secciones tal como se muestran en la Figura 1.2.

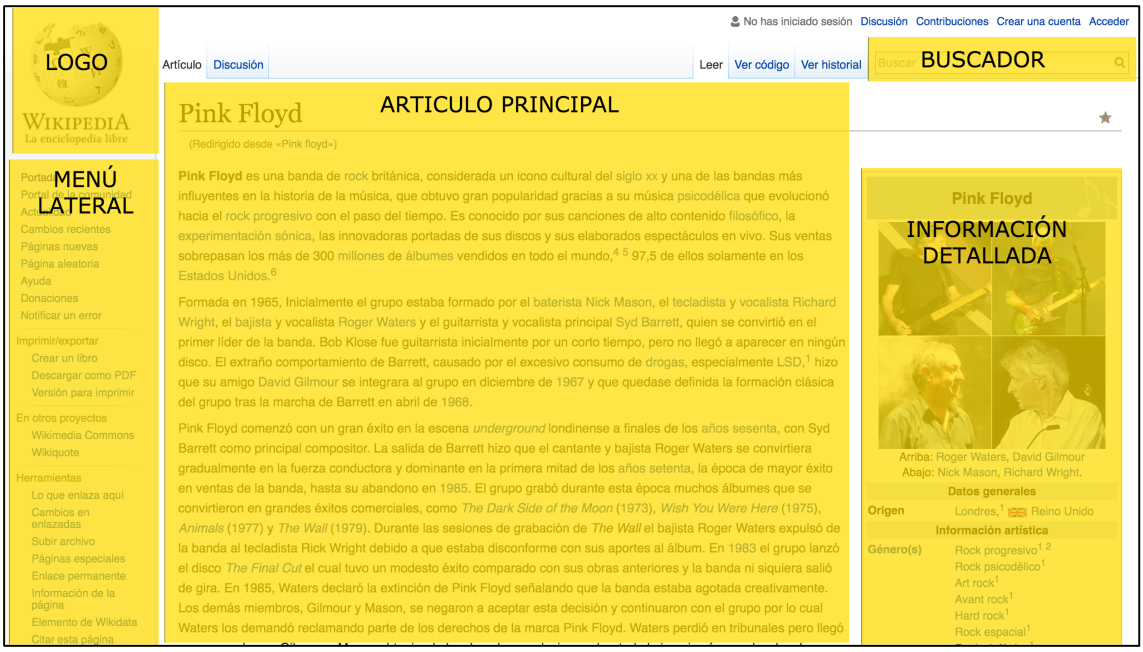


Figura 1.2: Captura de pantalla del sitio web "Wikipedia" con encapsulación del contenido en secciones. Los recuadros amarillos representan secciones "colapsadas" es decir, que sus elementos internos no pueden ser accedidos por los lectores de pantalla hasta que la misma no se abra para permitirlo.

Es decir, al recorrer el sitio con el lector de pantalla, el mismo dictará las palabras "Logo", "Buscador", "Menú lateral", "Articulo Principal" e "Información detallada", luego de este recorrido secuencial el usuario notara que la página no dispone de más secciones.

Dado que el usuario ahora posee conocimiento de todas las secciones disponibles, el mismo puede decidir obtener mayor información sobre alguna de ellas, como por ejemplo la sección "Articulo Principal", para lograr esto, el usuario será capaz de solicitar la misma mediante el uso de algún comando especial del teclado, el cual provocará que dicha sección se expanda ingresando el foco del lector de pantalla dentro de ella y haciendo su contenido accesible. En esta instancia el usuario no vidente puede comprender fácilmente que se encuentra en la sección de "Articulo Principal" dejando de lado la información de las otras secciones por un momento. La Figura 1.3 muestra un ejemplo de cómo dicha sección es habilitada para su inspección.

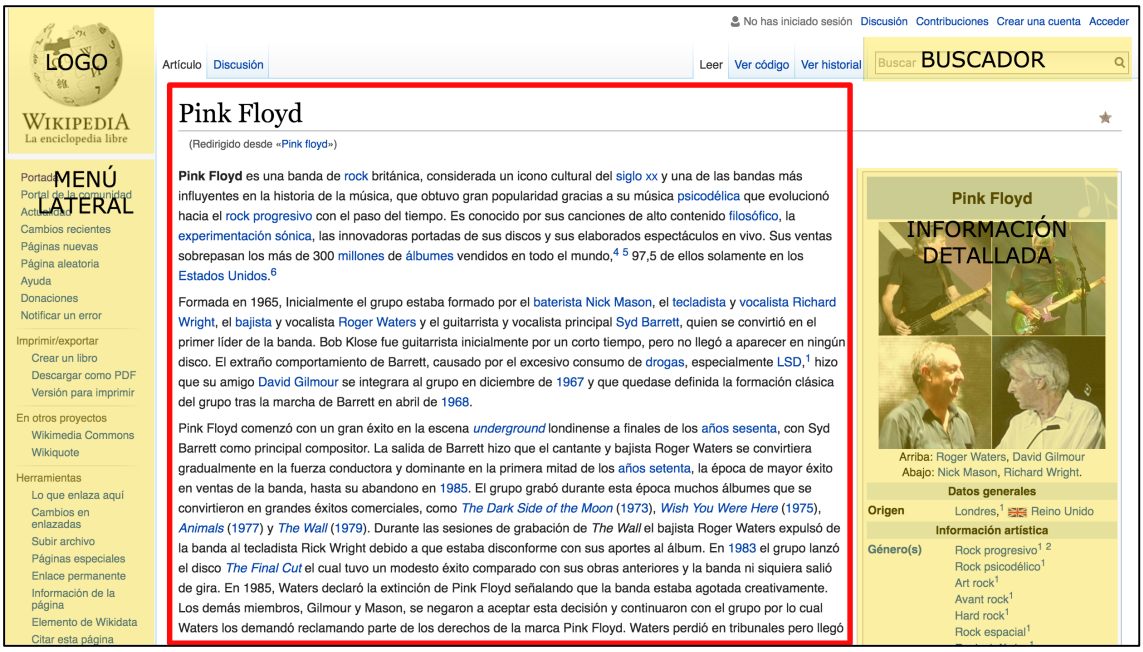


Figura 1.3: Captura de pantalla del sitio web "Wikipedia", cuya sección "Articulo Principal" es expandida para navegar dentro de él. Los recuadros amarillos representan secciones colapsadas (cerradas), mientras que el recuadro rojo representa aquella sección que se ha abierto (expandido) para permitir el ingreso a sus elementos internos.

Estas agrupaciones o "etiquetado" sobre el contenido no solo permiten simplificar el contexto de la página sino que a la vez pueden ser utilizadas para agregar información semántica a las mismas y otras funcionalidades de accesibilidad. La identificación y creación de dichas secciones será tema de desarrollo a lo largo de este trabajo.

En [Takagi et al. 2008] se describe el modelo de una herramienta para mejorar la Accesibilidad mediante el uso de "Crowdsourcing"[1] (colaboración de voluntarios) y "Annotation-Based Transcoding"[2] (aplicación de anotaciones sobre los elementos web). Ellos ofrecen un sistema en donde los usuarios crean anotaciones sobre los elementos de las paginas web que carecen de accesibilidad. Dichas anotaciones permiten reconocer cuales deben ser los cambios a aplicar en el sitio tras ser accedido por medio de un proxy. Sin embargo, reconocen que el sistema carece de simplicidad para la creación de estas anotaciones, no solo su creación resulta engorrosa sino también confusas para un usuario común o sin conocimientos técnicos. Gracias a esta

[1] "*Crowdsourcing*" es el comportamiento de naturaleza colaborativa e interactiva de la red. En el sentido más estricto, tiene que ver con distribuir "cálculos" que son llevados a cabo por la colaboración de grupos de personas.

[2] En [Asakawa and Takagi 2000] se describe el termino de "*Annotation-Based Transcoding*" como una herramienta para la mejora en la accesibilidad de sitios web mediante el uso de anotaciones de los elementos de cada pagina con información relevante sobre accesibilidad.

referencia hemos reforzado algunos de los objetivos a lograr durante nuestro desarrollo:

1. El uso de Crowdsourcing para la generación de las agrupaciones o mejor dicho "etiquetado" del contenido web.
2. La búsqueda de un sistema de etiquetado simple.
3. La aplicación de dichas modificaciones del lado del cliente evitando la utilización de proxies.

1.5 Objetivos

Objetivos técnicos

- Estudiar qué tipo de información agregada a los distintos elementos de una pagina web permiten mejorar la accesibilidad de los mismos, considerando algunas de las pautas definidas en la WCAG 2.0.
- Modelar esta información como etiquetas semánticas de alto nivel.
- Crear herramientas que permitan automatizar el proceso de etiquetado y posterior modificación de las páginas según estas etiquetas.
- Desarrollar una interfaz simple que permita a usuarios contribuir fácilmente en la creación de etiquetas y la posterior utilización de las mismas.
- Diseñar la herramienta teniendo en cuenta el modelo de *Crowdsourcing* y sus obstáculos.

Objetivos sociales

- Enfrentar el problema de la accesibilidad web.
- Mejorar la experiencia de navegación para aquellas personas que requieren del uso de lectores de pantalla.

1.6 Contribuciones

A continuación enumeraremos las contribuciones de este trabajo:

1. Estudio y clasificación de diferentes métodos y herramientas actuales que buscan mejorar la accesibilidad de las páginas web.
2. Análisis comparativo de las técnicas de refactoring y transcoding.
3. Enumeración de problemáticas halladas en el uso de lectores de pantalla.
4. Definición de un método para mejorar la accesibilidad web mitigando las problemáticas definidas anteriormente.
5. Desarrollo de una framework extensible que implementa dicho método.
6. Desarrollo e implementación de un algoritmo de similitud de elementos HTML para la definición de objetos semánticos.
7. Desarrollo de una plataforma de crowdsourcing.
8. Manual de usuario para los distintos roles de la aplicación.
9. Documentación detallada del código para programadores.

1.7 Organización de la Tesis

Capitulo II

Se introducirá al lector dentro del contexto de trabajo, definiendo las terminologías relacionadas, detallando antecedentes existente y analizando trabajos y herramientas relacionadas en cuanto al tema, problema y/o metodología aplicada.

Capitulo III

Introducción al desarrollo propuesto. Se hablará en detalle de los obstáculos y conceptos que integran al desarrollo. Finalmente hablaremos de la arquitectura del mismo.

Capitulo IV

Servirá a modo de manual de usuario de cada una de las aplicaciones mencionadas en el capitulo III, con captura de pantalla de todas las funcionalidades.

Capitulo V

Servirá a modo de manual técnico, donde podrá encontrarse la organización del código de la aplicación y otros detalles para su desarrollo.

Capitulo VI

Conclusiones, descripción de las contribuciones y trabajos futuros.

Capitulo II
Background y trabajos relacionados

2.1 Background

Accesibilidad web

La accesibilidad web tiene como objetivo lograr que las páginas web sean utilizables por el mayor número de personas posible, independientemente de sus conocimientos o capacidades personales e independientemente de las características técnicas del equipo utilizado para acceder a la Web [Alicante]. Trata de eliminar las barreras que previene a usuarios con ciertos tipos de discapacidades o limitaciones en el uso y beneficio de la información presente en internet.

La inaccesibilidad se presenta de muchos modos en internet. La misma varia, desde la imposibilidad de acceder al recurso web, hasta la inaccesibilidad presentada a una persona que carece de conocimientos computacionales necesarios para interpretar la interfaz de usuario.

Usabilidad

Usabilidad es un atributo evalúa que tan fácil es una interfaz de usar [Nielsen 2012]. Esta se define por medio de 5 componentes:

- Facilidad de aprendizaje: Nivel de facilidad para aprender el diseño.
- Eficiencia: La velocidad con la que se puede realizar las tareas.
- Facilidad para recordar: Que tan fácil es recordar el diseño por el usuario.
- Facilidad de cometer errores: Frecuencia en la que el usuario comete errores.
- Nivel de satisfacción: La satisfacción del usuario al utilizar el diseño.

Tal como se explico anteriormente, la accesibilidad y la usabilidad están íntimamente relacionadas. Uno de los principales objetivos de este trabajo es mejorar la accesibilidad por medio de la mejora de la usabilidad.

Refactoring

Refactoring es una técnica que se aplica sobre el código fuente con el objetivo de mejorar su estructura sin alterar la funcionalidad del mismo [Fowler 1999]. Estas modificaciones abarcan desde cambios pequeños, como renombrar funciones o variables, hasta cambios más complejos, como aplicar los patrones de diseño correspondientes.

Si bien la funcionalidad del programa no se ve alterada, el refactoring es una herramienta que ayuda en distintos aspectos a los programadores. Suele mejorar la legibilidad y simpleza del programa, permitiendo que este puede ser entendido más fácilmente.

El concepto de refactoring se ha expandido a otros aspectos de la calidad del software tal como la usabilidad [Garrido et al. 2011] y la accesibilidad [Garrido et al. 2014]. Aquí, las mejoras no se encuentran orientadas al código de los programadores sino a la experiencia del usuario con el uso del programa, a fin de hacerlo más usable y/o accesible. Cada refactoring propone alternativas de implementación a casos particulares en donde se detectan problemas de usabilidad y/o accesibilidad, de este modo, se pretende que los niveles de usabilidad y accesibilidad crezcan a medida que dichos refactorings son aplicados. Estos cambios afectan directamente a las páginas web, modificando las interfaces originales y sin alterar su funcionalidad, para mitigar los problemas hallados.

Transcoding

Se denomina transcodificar a la conversión directa de un formato a otro, esta puede realizarse con o sin pérdida de calidad. El termino de "transcoding" también a sido llevado a otros ámbitos del software como la accesibilidad de páginas web [Asakawa and Takagi 2000] donde los autores proponen un método para mejorar la accesibilidad web para usuarios con discapacidades visuales. Este método se basa en convertir las interfaces web (transcodificarlas) a un formato más apto para los lectores de pantalla, más adelante de este capítulo continuaremos hablando de este trabajo. Por ahora solo destacaremos que cuando se utiliza el transcoding como herramienta para resolver la inaccesibilidad, se hace hincapié en la interfaz y en cómo los usuarios acceden a los datos por medio de ella. En este tipo de transcoding, las páginas web originales son sometidas a un proceso de conversión en donde su contenido pasa a formar parte de nueva página web con una interfaz más adecuada para accederlo.

Diferencias entre refactoring y transcoding

El refacoring se basa en detectar problemas de usabilidad y/o accesbilidad, analizarlos y proponer reglas o patrones de diseño con el objetivo de mitigarlos sin cambiar la funcionalidad de la aplicación, de esta forma, la mejora se logra de forma gradual sobre la interfaz original. Por otro lado el transcoding se basa en la generación de una nueva interfaz como resultado de un proceso de conversión sobre la interfaz original. Una nueva interfaz significa poder **cambiar por completo la forma en el que el usuario interactúa con la información.** A continuación mostraremos un mejor ejemplo de esta diferencia en forma esquemática:

Al utilizar la técnica de "**refactoring**", partimos de la interfaz "A" a la cual decidimos aplicarle un conjunto de cambios en base a reglas y/o patrones (cada cambio sujeto a mejorar algún aspecto en la accesibilidad/usabilidad especial y detectado gracias a dicha regla o patrón definida por el refactoring), para luego obtener una interfaz " A' " (A prima), que se diferencia de la anterior solo por dichas reglas/patrones que se han aplicado sobre ella.

Figura 2.1.1: Representación gráfica del ejemplo de refactoring.

Al utilizar la técnica de "**transcoding**", partimos de una interfaz "A" la cual utilizaremos como entrada para un proceso de conversión, el cual recolectará la información y la presentará en un nuevo formato, es decir, una nueva interfaz "B" con el mismo contenido o parte de el (recordemos que por definición de transcoding puede existir perdida de calidad, en el caso de interfaces web, la falta de contenido en la nueva interfaz equivale a esta perdida de calidad).

Figura 2.1.2: Representación gráfica del ejemplo de transcoding.

Finalmente se encuentra la **técnica que proponemos**, la cual podemos decir que toma aspectos de las anteriores. Partimos de una interfaz "A" y realizamos una conversión que da como resultado, y diferencia del transcoding convencional, la misma interfaz pero con metadatos adicionales. Dichos metadatos podrán ser utilizados posteriormente para realizar mejoras en la interfaz, ya sea porque estos metadatos corresponden a información "estándar" utilizable por aplicaciones (como etiquetas "ARIA" para lectores de pantalla), o porque son metadatos personalizados que podrán ser utilizados por otro tipo de programas específicos (como por ejemplo el script que ofreceremos más adelante para gestionar la navegación sobre las páginas web). Veremos más adelante que esto último podría asociarse a la técnica de refactoring.

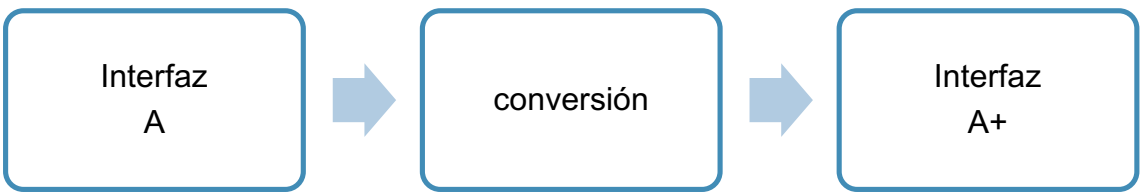


Figura 2.1.3: Representación gráfica de lo que hará nuestra herramienta.

Crowdsourcing

El crowdsourcing consiste en externalizar tareas a resolver por un conjunto de personas ajenas al proyecto con el objetivo de agilizar y/o distribuir la carga del esfuerzo necesario para cumplir el objetivo común [Infocrowdsourcing 2013]. Existen diversos escenarios de crowdsourcing, pero todos ellos se asemejan en 2 características principales:

- Existe una persona/empresa/entidad interesada en cumplir un objetivo final y la cual organiza el sistema de Crowdsourcing.
- Hay un conjunto de personas que participan colaborando en la resolución de tareas las cuales aportan a la realización del objetivo final.

Junto con el crowdsourcing surgen diversos obstáculos a resolver para poder ser llevado a cabo, entre ellos se encuentran: Como lograr la participación de colaboradores (voluntarios o contratados), si se requiere de capacitación previa para realizar las tareas, como evitar acciones malintencionadas en el desarrollo de tareas por parte de la red de colaboradores, si se requiere de una participación explicita o implícita, etc. [Doan et al. 2011].

El crowdsourcing es una herramienta poderosa que resulta muy conveniente para el foco de este trabajo. Dado que nuestro proceso de transcidificación requiere de información adicional sobre el sitio web, no podemos pretender generar esta información para todos los sitios web nosotros mismos. Que nuestro sistema brinde una plataforma de crowdsourcing es lo ideal para lograr una solución que se fundamente en el trabajo colaborativo como base para resolver el problema planteado.

WCAG 2.0

Las Pautas de Accesibilidad para el Contenido Web (WCAG) 2.0 cubren un amplio rango de recomendaciones para crear contenido Web más accesible. Los criterios de conformidad de las WCAG 2.0 están escritos como enunciados verificables no específicos de ninguna tecnología. Proporcionan varios niveles de orientación:

- **Principios**: Comprende los 4 principios de la accesibilidad
  - Perceptible: el contenido debe presentarse de forma perceptible.
  - Operable: la interfaz y la navegación de la misma debe ser operable.
  - Comprensible: el contenido y las operaciones de la interfaz deben poder ser comprendidas fácilmente.
  - Robusto: El contenido debe ser lo más robusto posible a modo de poder ser interpretado por distintos exploradores así como tecnologías de asistencia.

- **Pautas generales**: Comprende 12 pautas (cada una asociada a un principio) que los autores deben lograr con el fin de crear un contenido más accesible para los usuarios con distintas discapacidad.

    1. Alternativas textuales (Perceptible): Proporcionar alternativas textuales para todo contenido no textual.
    2. Medios tempo-dependientes (Perceptible): Proporcionar alternativas para los distintos medios, ejemplo: Si se trata de solo video, proporcionar un audio equivalente, y viceversa.
    3. Adaptable (Perceptible): Crear contenido que pueda ser presentado de diferentes formas sin perder información o estructura.
    4. Distinguible (Perceptible): Facilitar al usuario ver y oír el contenido, incluyendo la separación entre el contenido y el fondo.
    5. Accesibilidad por teclado (Operable): Proporcionar acceso a toda la interfaz por medio del teclado.
    6. Tiempo suficiente (Operable): Proporcionar al usuario el tiempo suficiente para leer y usar el contenido.
    7. Convulsiones (Operable): No diseñar el contenido de un modo que pueda provocar ataques, espasmos o convulsiones.
    8. Navegable (Operable): Proporcionar medios para ayudar a los usuarios a navegar, encontrar contenido y determinar donde se encuentran.
    9. Legible (Comprensible): Hacer que los contenidos textuales resulten legibles y comprensibles.
    10. Predecible (Comprensible): Hacer que las páginas operen de forma predecible.
    11. Entrada de datos asistida (Comprensible): Ayudar a los usuarios a evitar y corregir errores.
    12. Compatible (Robusto): Maximizar la compatibilidad con las aplicaciones de usuarios actuales y futuras, incluyendo las ayudas técnicas.

- **Criterios de conformidad**: Para cada pauta se proporcionan los criterios de conformidad verificables, un conjunto d requisitos los cuales determinan el nivel de conformidad a medida que estos son satisfechos. Se definen tres niveles de conformidad: A (el más bajo), AA y AAA (el más alto).

- **Colección de técnicas suficientes y recomendables**: Para cada pauta y criterio de conformidad a su vez se ofrece una amplia variedad de técnicas, las cuales se pueden distinguir entre "suficientes" (para satisfacer los criterios de conformidad) y "recomendables" (que van más alla de los requisitos de conformidad).

WAI-ARIA

WAI-ARIA (**A**ccessible **R**ich **I**nternet **A**pplications Suite) [W3C 1997] es una iniciativa de la W3C que define un medio para lograr hacer el contenido web y las aplicaciones web más accesibles para personas con discapacidades. Especialmente cuando se trata de contenido dinámico y controles de interfaces avanzados desarrollados con tecnologías Javascript, Ajax y otras similares.

Más específicamente, WAI-ARIA provee un framework para añadir atributos adicionales que ayudan a identificar características de la interacción del usuario con la interfaz, como los elementos se relacionan entre si, en que estado se encuentran, etc. WAI-ARIA describe nuevas técnicas de navegación para marcar regiones y estructuras web comunes como por ejemplo: menús, contenido principal, contenido secundario, banners informativos, y otros tipos de estructuras comunes. Por ejemplo, con WAI-ARIA los programadores pueden identificar regiones de páginas y permitir a usuarios moverse fácilmente entre ellas tan solo con el uso del teclado en vez de tener que presionar la tecla TAB varias veces. WAI-ARIA también permite mapear controles, regiones dinámicas y eventos a APIs de accesibilidad. Provee a los autores web con lo siguiente:

- Roles para describir el tipo de componente (menú, treeitem, slider, etc.).
- Roles para describir la estructura de la página (encabezados, regiones, tablas, etc.).
- Propiedades para describir el estado de los componentes (si esta habilitado o no, si posee popup, etc).
- Propiedades para definir regiones que son propensas a modificarse repetidamente.
- Propiedades para definir la existencia de drag and drops.
- Un método para proveer navegación por teclado de los objetos y eventos mencionados anteriormente.

Los roles WAI-ARIA

Los roles que ofrece WAI-ARIA pueden clasificarse en 3 grupos:

- **Roles abstractos** (*Abstract roles*): Son roles definidos para abarcar conceptos generales de los cuales otros roles de la taxonomía heredan. No deben ser utilizados nunca por programadores.

- **Roles de componentes navegacionales** (*Widget roles*): Son roles que definen tipos de componentes simples y compuestas (widgets y composite widgets) que se caracterizan por proveer diferentes tipos de navegación. Por ejemplo: El rol "tree" (componente compuesta) define una componente cuya interfaz de navegación se asemeja a la de un árbol (o directorio), compuesta por elementos de rol "treeitem" (componente simple) que representa a los nodos internos que forman parte del árbol de navegación (como las carpetas y archivos de un directorio).

- **Roles de estructura** (*Structure roles*): Son roles que describen estructuras que organizan el contenido interno del sitio. Por ejemplo: El rol "heading" define a un elemento que representa el título de una sección.

- **Roles de punto de referencia** (*Landmark roles*): Son roles que denotan regiones comunes de las página web, estas sirven como punto de referencia para la interpretación del sitio. Por ejemplo el rol "main" define la sección que posee el contenido principal de la página.

Los roles a la vez vienen acompañados de ciertos atributos adicionales que permiten determinar su estado y/o propiedades. Vale la pena destacar que el solo hecho de agregar estos atributos al código HTML no es suficiente, el programador web debe hacerse cargo de actualizar dichos valores para que estos sean consistentes con el estado de la aplicación.

2.2 Trabajos relacionados

Reorganizando las páginas web a esquemas más accesibles para los lectores de pantalla a partir del uso de anotaciones.

Con el objetivo de mejorar la accesibilidad de las páginas web para usuarios con problemas de visión que precisan de la utilización de lectores de pantalla, los autores Asakawa y Takagi proponen una solución mediante el uso de anotaciones sobre **grupos de información** que componen las páginas web [Asakawa and Takagi 2000]. En este trabajo se definen un conjunto de grupos de información por medio de ***anotaciones estructurales*** (información sobre los elementos que componen al grupo) y ***anotaciones de comentario*** (información semántica sobre cada grupo) las cuales deben ser especificadas por usuarios con conocimientos técnicos sobre el desarrollo web. Luego utilizan dichos **grupos de información** como base para reestructurar el código HTML de las paginas web, es decir, realizan una transcodificación (conversión) de la página web a un formato más apropiado para los lectores de pantalla. Finalmente, los usuarios con problemas visuales pueden acceder a las páginas transcodificadas mediante un proxy. A esta técnica la denominan "Web Annotation-based Transcoding", lo cual puede traducirse como Transcodificación web utilizando anotaciones, se trata de una herramienta que permite mejorar la accesibilidad de los sitios "en vivo" (on the fly), ya que no requiere de ningún tipo de instalación por parte del usuario final, sino que solo se necesita que exista suficiente información sobre los grupos de información que componen a una página web como para lograr un buen proceso de transcodificación. Durante esta conversión puede que no toda la información sea transformada al nuevo formato, es decir, puede que se pierda cierta "calidad" de la información durante el proceso.

En las siguientes imágenes tomadas de [Asakawa and Takagi 2000] se puede ver un ejemplo de este procedimiento, en donde la interfaz de un página web de noticias es transformada a otra interfaz más accesible para lectores de pantalla.

Figura 2.2.1: Imagen tomada del articulo [Asakawa and Takagi 2000] *donde se encuadran los focos de información que serán transcidificados por la herramienta propuesta en el articulo.*

Figura 2.2.2: Imagen tomada del articulo [Asakawa and Takagi 2000] *que muestra el nuevo formato de la página tras haber aplicado la transcodificacíon sobre la página que figura en la imagen 2.2.1. Vemos que el contenido es reorganizado dentro de una tabla, ofreciendo una nueva interfaz que facilita el acceso a la información por medio de lectores de pantalla.*

La solución que proponemos en este trabajo se asemeja mucho al del *"Annotation-based Transcoding"*, pero con varias diferencias:

1. Utilizaremos el termino de "**etiquetas**" en vez de "**anotaciones**" para la agrupación de información a modo de poder diferenciarlas de nuestro método y porque nos parece una palabra más natural a utilizar para usuarios no técnicos.

2. Ellos utilizan dos tipos de anotaciones, unas para definir los grupos estructuralmente y otras anotaciones para definir la semántica de cada grupos definido anteriormente. Estas anotaciones deben ser aplicadas por usuarios con ciertos conocimientos técnicos. Nosotros buscamos encontrar un método más intuitivo: tras la agrupación de un conjunto de elementos el usuario solo deberá seleccionar la etiqueta semántica que más se asemeje a lo que acaba de agrupar, luego nuestro algoritmo de similitud reconocerá el grupo estructuralmente e identificará si el mismo patrón estructural se encuentra presente en más partes del mismo sitio, a modo de asignarle a estos nuevos grupos similares reconocidos la misma etiqueta semántica y así agilizar el proceso de etiquetado.

3. En el trabajo citado, la transcodificación sucede en una única etapa, cuando un usuario hace un llamado al proxy solicitando la página web transcodificada, nosotros proponemos una transcodificación constante que sucederá del lado del cliente, sin necesidad de acceder a dominios diferentes de la página web. Con tan solo poseer instalada nuestra herramienta, el usuario podrá descargar automáticamente las etiquetas almacenadas en nuestros servidores para realizar la transcodificación en vivo.

4. Con una única transcodificación inicial se complica el reconocimiento de elementos dinámicos pertenecientes a la web 2.0, dado que varios de los elementos de la aplicación pueden no estar presentes en tiempos determinados del ciclo de vida de la aplicación y por lo tanto no estar a la hora de realizar la transcodifiación. Mediante nuestra técnica de reconocimiento de elementos en vivo, la intercepción de cambios en el DOM y la identificación de patrones, seremos capaces de detectar la aparición de dichos elementos dinámicos en cualquier momento del ciclo de vida de la aplicación, dado que nuestra transcodificación es constante.

5. Gracias a que nuestra transcodificación sucede en el lado del cliente, la misma se puede personalizar, es decir, nuestro script puede ser editado para evitar o agregar diferentes cambios durante la transcodificación, diferentes usuarios pueden obtener resultados de transcodificaciones distintas en un mismo sitio web.

Uso de comunidades para resolver la accesibilidad a gran escala.

La solución ofrecida por la técnica de Annotation-based Transcoding [Asakawa and Takagi 2000] no es muy escalable a la infinita cantidad de páginas web existentes. Es por eso que en [Takagi et al. 2008] los autores proponen la utilización del Crowdsourcing para aplicar dicho método a gran escala a través de un sistema que permite a los usuarios reportar problemas de accesibilidad y en el cual otros usuarios voluntarios pueden hacerse cargo del problema creando el conjunto de anotaciones necesarias con el objetivo de mitigarlo.

Esta solución descentraliza la responsabilidad de la mejora de accesibilidad. A mayor la comunidad de voluntarios mayor será el poder de la herramienta para abarcar más variedad de sitios. Aquí entra un nuevo concepto en juego, el de "Autoría de documentos Colaborativa" (Collaborative document authoring), el cual significa que los documentos creados para cada sitio que poseen las anotaciones (metadata) que permiten mejorar la accesibilidad del sitio, no son creados por autores únicos, sino que es el resultado del esfuerzo y participación de un conjunto de voluntarios.

Este trabajo nos demuestra que nuestro enfoque de utilizar el Crowdsourcing como modelo para una transcodificación a mayor escala es adecuado y completamente aplicable al problema que tratamos de resolver. En nuestro sistema permitiremos a usuarios reportar páginas inaccesibles para que otros usuarios voluntarios puedan realizar el proceso de etiquetado necesario y compartir la información necesaria para que nuestra herramienta pueda realizar el transcoding adecuado sobre el sitio solicitado. A su vez, dado que la información compartida es pública entre todos los usuarios, usuarios voluntarios podrán crear copias de los trabajos de etiquetación de cada sitio para realizar sus propias modificaciones y así contribuir a la autoría colaborativa.

Refactoring personalizados del lado del cliente.

Por otro lado en [Garrido et al. 2014] los autores proponen una solución diferente a la técnica de transcoding para resolver problema de usabilidad mediante la aplicación de refactorings sobre las interfaces web, denominados "WIR" (Web interface refactoring). Tal como lo mencionamos recientemente, el termino de "Refactoring" ha sido expandido a otros aspectos de la calidad de software, en este caso al de la accesibilidad web. Los WIRs no alteran la funcionalidad o comportamiento de las aplicaciones web sino que se emplean con el propósito de mejorar la forma en que los usuarios acceden a su contenido o facilitar el uso de sus funcionalidades. Cada WIR esta enfocado a resolver un problema de accesibilidad específico, estos comprenden un patrón de diseño a emplear para resolverlo y diferentes WIRs pueden implementarse en paralelo para resolver distintos problemas de forma incremental. Por ejemplo, uno de los problemas de accesibilidad/usabilidad destacados en este trabajo es el de "páginas sobrecargadas" , es decir, cuando una página web posee demasiado contenido en una misma sección, o peor aún, cuando combina varios conceptos o

operaciones que deberían haberse presentado en etapas separadas y la sobrecarga de información puede confundir al usuario. A este problema los autores proponen un WIR llamado "Split Page" que básicamente consiste en dividir una página web en secciones distintas para reducir el impacto de información que esta puede ocasionar en el usuario.

Una vez identificados cuales son los WIRs que deben ser utilizados dentro de una página web, existen 2 métodos por los cuales estos pueden ser integrados:

1. Mediante la creación de una página web en paralelo que integre cada uno de los cambios dictados para cada WIR.
2. Mediante la aplicación de scripts adicionales en el lado del cliente que apliquen estos cambios.

Nuestra herramienta también hará uso de scripts del lado del cliente para aplicar cambios en las páginas web, pero existen ciertas diferencias en relación a la técnica de los WIRs:

1. Los WIRs a veces pueden significar cambios muy específicos por lo cual los programadores deberán crear scripts dedicados a un sitio web en particular. Si se desea emplear un mismo script en otro sitio web, o si las características iniciales del sitio original cambian, esto puede generar que el programador deba reconfigurar parcial o completamente el script para adaptarlo al nuevo contexto.
2. En cambio los scripts que nosotros incluiremos en el lado del cliente no se basarán en información provista directamente por el sitio web, sino que utilizarán metadatos adicionales previamente incluidos tras el proceso de transcodificación para comprender el sitio web y así poder aplicar los cambios deseados.

A continuación se muestra un esquema gráfico para diferenciar estos puntos, en la Figura 2.2.3 podemos ver como el script actúa utilizando únicamente los datos que pueden obtener de la página web original para lograr la aplicación del WIR. Por otro lado en la Figura 2.2.4 vemos como el script no solo posee los datos de la página web, sino que también dispone de metadatos adicionales incluidos por el proceso de transcodificación para realizar sus acciones. Si además este script solo utilizara metadatos para cumplir con su objetivo, entonces en el caso de que la página web objetivo cambie, solo sería necesario actualizar estos metadatos para que el script continúe funcionando correctamente. Más adelante veremos como se produce la generación de estos metadatos por medio de nuestra herramienta lo que puede significar un proceso mucho más simple que el de tener que reconfigurar por completo el código de un script.

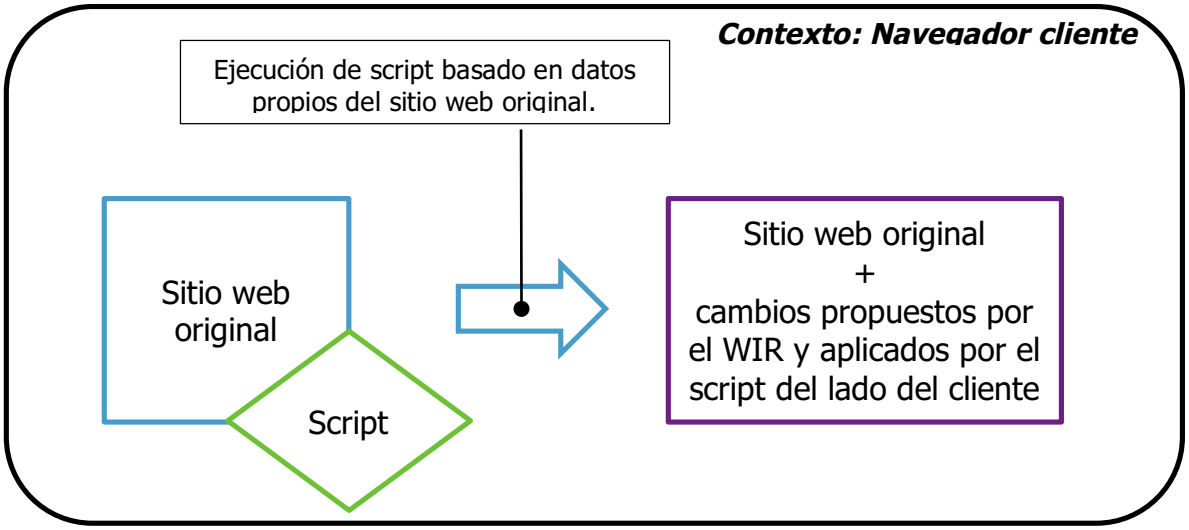


Figura 2.2.3: Gráfico representativo de cómo un script es incluido en una página web para la incorporación de un WIR, todo esto sucediendo del lado del cliente.

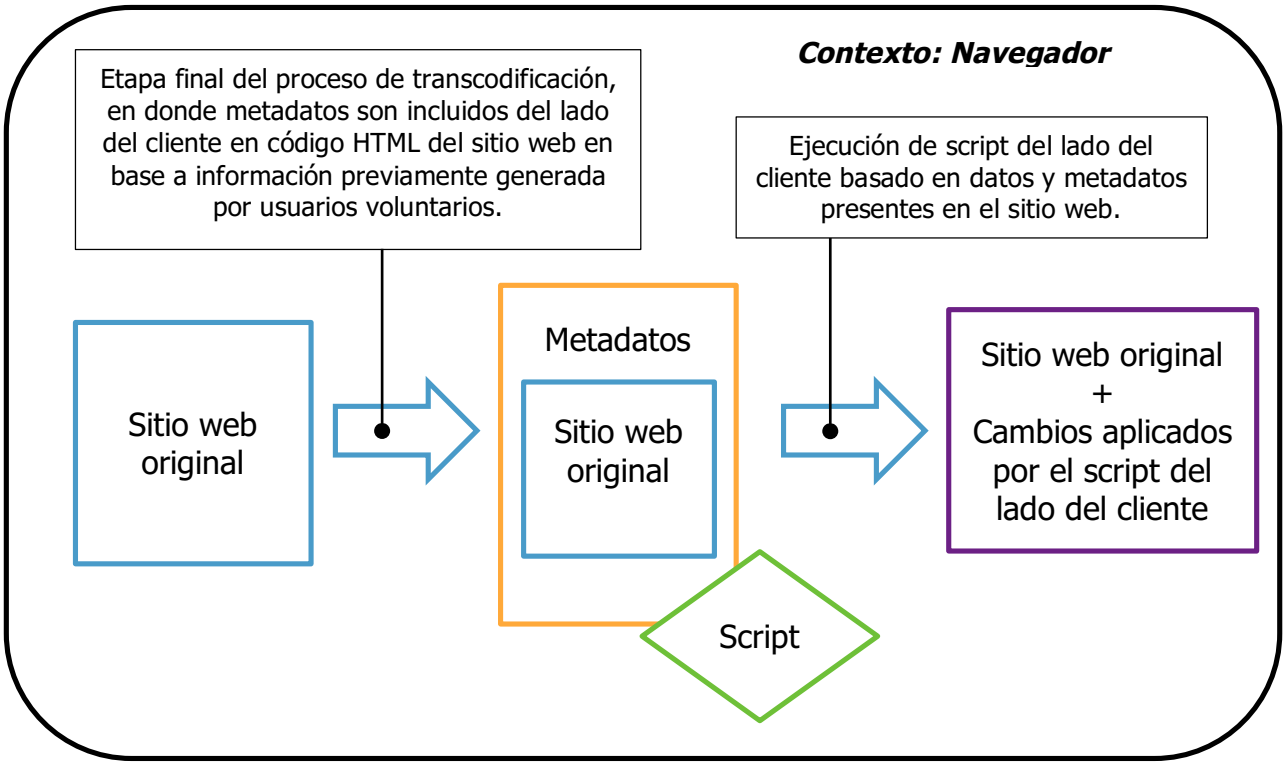


Figura 2.2.4: Gráfico representativo de como un script es incluido en una página web que dispone de metadatos adicionales para facilitar el accionar del script, todo esto sucediendo del lado del cliente.

Un editor visual para corregir el orden de lectura

En [Sato et al. 2009] se presenta una herramienta que permite corregir el orden en el que los elementos son leídos dentro de los sitios web. Esta herramienta brinda una visualización del flujo en el que los elementos web son recorridos y leídos por los lectores de pantalla y permite alterar el mismo mediante dicha visualización. En otras palabras, se trata de una técnica de visualización que permite obtener y organizar el orden de recorrido de los elementos que componen a las páginas web. Gracias a esta visualización resulta muy simple para los usuarios hallar problemas en el flujo del recorrido y así ajustarlo inmediatamente.

El orden de lectura es representado por medio de un conjunto de flechas dibujadas sobre los mismos elementos HTML y las cuales se encuentran conectadas entre sí, la Figura 2.2.5 muestra un ejemplo.

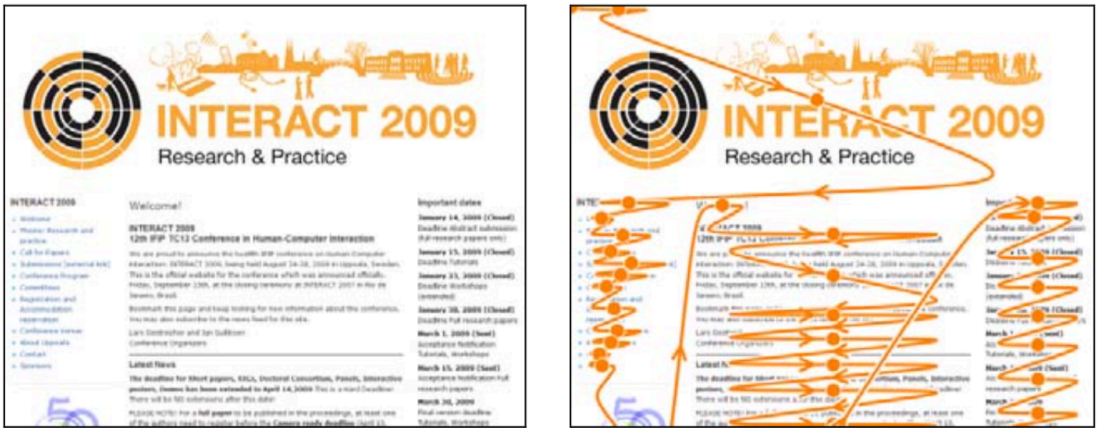


Figura 2.2.5: Imagen tomada del artículo [Sato et al. 2009], a la izquierda se visualiza una página web y a la derecha se visualiza la misma página web pero con un conjunto de flechas que han sido dibujadas sobre ella. Así es como la herramienta presenta al usuario el flujo de lectura del sitio web.

Cada elemento web perteneciente al flujo de lectura provee a su vez un "circulo" o nodo que sirve a modo de conector. Mediante este conector, los usuarios pueden "arrastrar" las flechas hacia ellos para así alterar el flujo de lectura, tal como se puede apreciar en la Figura 2.2.6.

Figura 2.2.6: Imagen tomada del artículo [Sato et al. 2009], se puede apreciar como el usuario "edita" el flujo de lectura mediante arrastrar una flecha hacia el conector deseado.

La herramienta a su vez controla que el flujo de lectura sea correcto, es decir, que no existan ciclos en el recorrido, que los elementos siempre se recorran en una única dirección (de izquierda a derecha, de arriba a abajo, no retrocediendo), etc.

Por último, la herramienta permite determinar el nivel de granularidad del flujo de lectura. A menor nivel de granularidad los nodos conectores del flujo de lectura se posicionarán en elementos más específicos permitiendo realizar un recorrido más detallado, mientras que a mayor nivel de granularidad, la herramienta agrupará elemento dentro de un mismo nodo conector (la agrupación de elementos se realiza mediante similitudes geométricas de los elementos), de esta forma cada nodo conector pasará a representar a conjuntos de elementos más grandes ofreciendo un recorrido de lectura más genérico. La Figura 2.2.7 muestra un ejemplo.

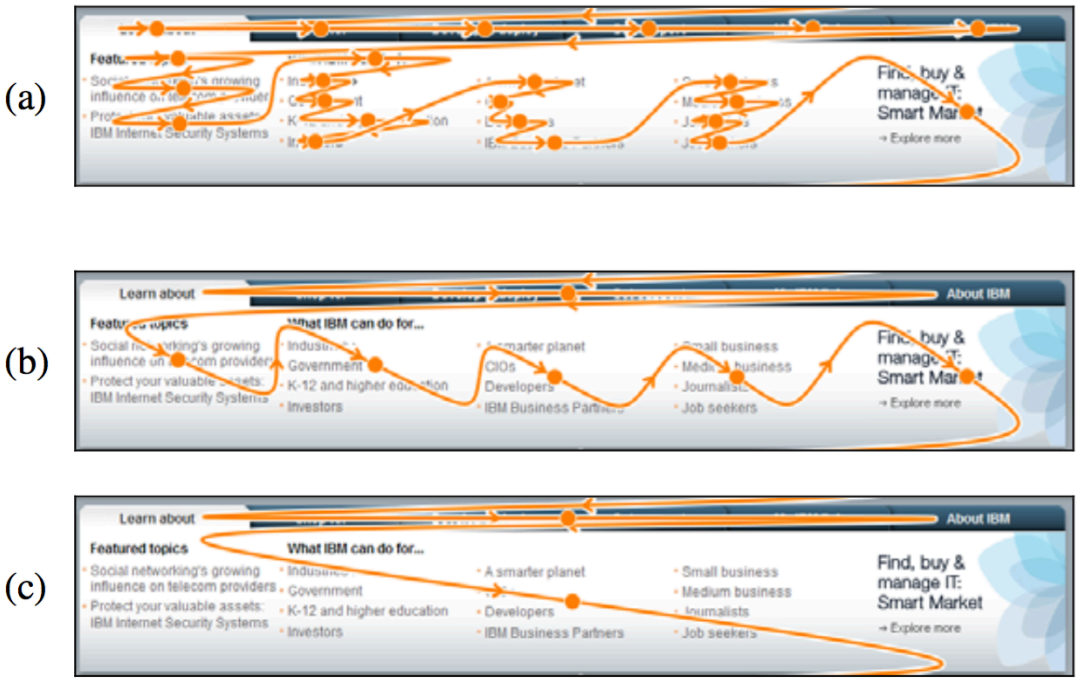


Figura 2.2.7: Imagen tomada del artículo [Sato et al. 2009], la figura (a) muestra un nivel de lectura de grano fino, la figura (b) muestra un nivel de lectura de grano intermedio y la figura (c) muestra un nivel de lectura de grano grueso. A medida que la granularidad crece, elementos consecutivos que comparten ciertas características geométricas son agrupados en mismos nodos conectores.

Esta herramienta resulta muy interesante para los objetivos de este trabajo, dado que se asocia de cierto modo a la herramienta que pretendemos desarrollar, existen varios aspectos a detallar:

- Esta herramienta se enfoca en mejorar el orden en que los elementos son leídos sin hacer mucho énfasis en la granularidad, mientras que nosotros

priorizaremos la granularidad, es decir, la forma en que agruparemos los elementos ofreciendo una navegación por niveles de granularidad.

- El aumento de granularidad de esta herramienta se basa en un algoritmo que recolecta información geométrica para determinar si dos o más elementos deben agruparse. No está muy clara la eficacia de esta técnica, sin embargo nosotros realizaremos esta agrupación mediante la colaboración de usuarios voluntarios, por lo que esperamos obtener agrupaciones más "correctas" (con más sentido semántico) que las brindadas por un algoritmo.

- La herramienta almacena el nuevo orden del recorrido en lo que denominan una "capa semántica" separada del contenido original, luego dicha información semántica es agregada al contenido por medio de metadatos. Esto se asemeja a lo que realizaremos en nuestra herramienta, ya que también almacenaremos la información semántica por separado para luego insertada del lado del cliente en forma de metadatos, veremos esto más en detalle a lo largo del capítulo 3.

- Los metadatos utilizados por esta herramienta para la reorganización del los elementos del sitio web comprenden el uso del atributo WAI-ARIA llamado "aria-flowto", atributo dedicado precisamente a definir el flujo de navegación de los elementos web. El problema esta en que dicho atributo aún no ha sido soportado por la mayoría de los lectores de pantalla, por lo que si bien dicha información es insertada por la herramienta, no podremos apreciar esta nueva funcionalidad mediante el uso de la mayoría de los lectores de pantalla disponibles. En el capítulo 4 veremos como utilizamos nuestra herramienta para ofrecer la misma información semántica, sin embargo, al igual que la herramienta mencionada en este artículo, propondremos esta funcionalidad como un trabajo a futuro a la espera de que el atributo estándar "aria-flowto" sea respetado por la mayoría de los lectores de pantalla.

- El trabajo de este artículo también considera al Crowdsourcing como estrategia fundamental para lograr expandir los resultados brindados por la herramienta a mayor cantidad de sitios web, lo cual refuerza nuestra motivación para utilizarla en el desarrollo de nuestro trabajo.

- Finalmente, en el articulo se menciona "la visualización es un factor tecnológico clave para permitir a usuarios regulares trabajar en la mejora de la accesibilidad web en la estrategia del Crowdsourcing", esta frase resulta muy importante para nosotros, dado que consideramos lo mismo y es por eso que buscamos ofrecer un método de trabajo visual para facilitar los aportes de nuestros usuarios voluntarios en nuestro sistema de Crowdsourcing.

2.3 Clasificación de herramientas

Existen diversas herramientas de utilidad para la accesibilidad web en todos sus aspectos (es decir, no solo para la accesibilidad de usuarios con problemas visuales), entre las mismas se pueden diferenciar:

Validadores

Son aquellas aplicaciones que permiten validar ciertas características de los sistemas. Por un lado tenemos los **validadores de aspecto** que muestran como se vería la página web en distintos navegadores o para personas con ciertas discapacidades, haciendo foco en la forma en que se visualiza la aplicación. Mientras que por otro lado están los **validadores de accesibilidad** que son herramientas que permiten revisar varias de las pautas de accesibilidad según la WCAG, haciendo foco en como se estructura y que metadatos de accesibilidad están disponibles en la aplicación. Un ejemplo de este último es:

- **WAVE** [WebAIM]: es una herramienta de evaluación online que permite crear un reporte sobre las problemáticas de accesibilidad que presenta un sitio. Con tan solo ingresar la URL de la página a evaluar, se muestra la pagina original junto a indicadores que muestran que esta bien y que esta mal respecto de la accesibilidad. Tal como se puede apreciar en la siguiente figura, a la izquierda aparece una columna con el reporte realizado por WAVE, y a la derecha el sitio web que esta siendo analizado, en donde ser remarcan los elementos que están siendo analizados.

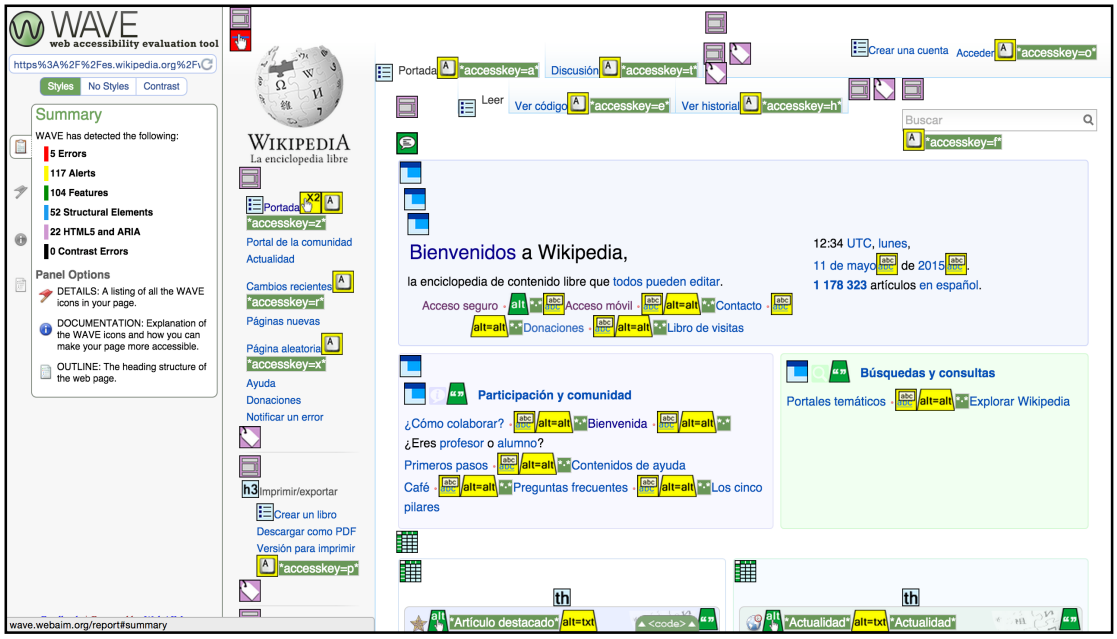


Figura 2.3.1: Captura de pantalla de la aplicación WAVE al analizar el sitio web Wikipedia.

Barras de desarrollo

Son herramientas que se instalan junto a un navegador web, permitiendo a los desarrolladores web comprobar la accesibilidad de las páginas web que están desarrollando y encontrar los errores cometidos con respecto a los estándares. Por ejemplo:

- **Web Accessibility Toolbar** [ThePacielloGroup]: Permite la evaluación manual de las webs en relación a varios aspectos de accesibilidad. Las funciones comprenden, identificar los componentes de una página web, proveer acceso a vistas alternativas de la página web y facilitar el uso de aplicaciones online de terceros. En la siguiente figura se puede apreciar la barra de herramientas en el borde superior del explorador web, ofreciendo un conjunto de herramientas de accesibilidad para el programador.

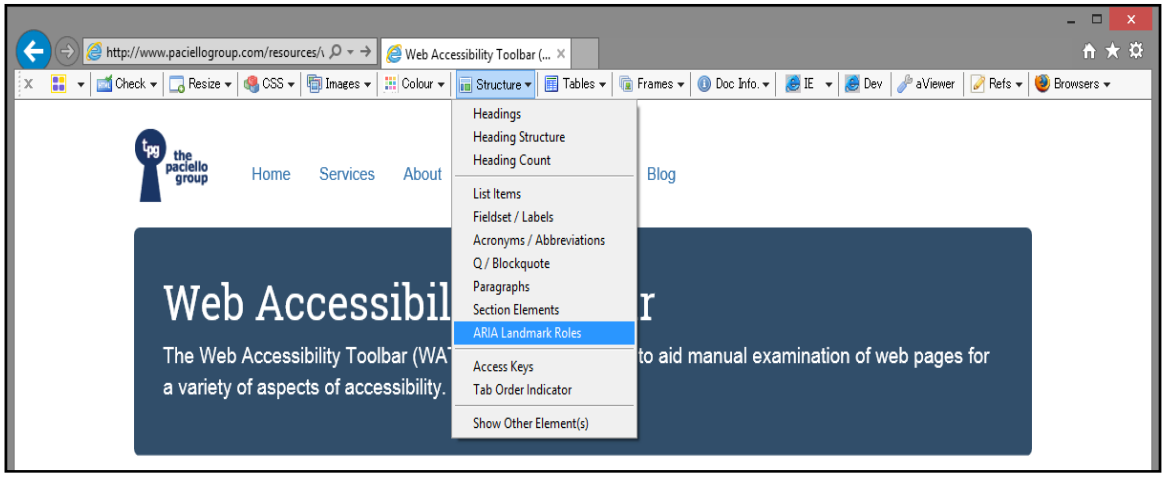


Figura 2.3.2: Captura de pantalla de la herramienta Web Accessibility Toolbar.

Lectores de pantalla

Permiten el uso de la computadora a usuarios con discapacidad visual gracias a un sintetizador de voz que "lee y explica" lo que se visualiza en la pantalla. Otorgan la posibilidad de navegar la interfaz gráfica principalmente mediante comandos del teclado mientras brindan la traducción sonora del texto seleccionado en la navegación. No solo traducen texto visible sino que además pueden ofrecer información adicional de acuerdo a etiquetas o metadatos no visibles para el usuario que puedan estar asociados al contenido. Entre los más conocidos se encuentran:

- Jaws (Windows)
- NVDA (Windows)
- Apple VoiceOver (OS X)
- ORCA (Linux)
- ChromeVox (Google Chrome)

Navegadores

Navegadores para principiantes o que funcionan con símbolos, recomendados para personas con discapacidad cognitiva o problemas de aprendizaje. Algunos ejemplos son:

- **WebbIE** [King]: un navegador para usuarios con discapacidad visual, integra comandos especiales para acceder a información especial como noticias, sintonizar radios, etc.
- **EIA** [ELR]: Un navegador de sistemas de pantalla táctil para el aprendizaje de internet, que además integra otras herramientas especiales para usuarios con discapacidades.

Simuladores

Son herramientas que simulan diferentes discapacidades, a modo de permitirle al desarrollador contemplar las paginas web del modo que lo haría un discapacitado diariamente. Entre las simulaciones, se encuentran simuladores visuales, cognitivos, etc. Un ejemplo de esto encontramos en **WebAIM** [WebAIM] donde podemos participar de simulaciones que nos ayudan a comprender la experiencia de usuarios con ciertas discapacidades.

Modificadores

Los modificadores son aquellas aplicaciones que permiten alterar/personalizar ciertas características del sistema, como por ejemplo los **magnificadores de pantalla** que permiten visualizar la pantalla con un considerable aumento en su tamaño, los **modificadores de color** que permiten alterar los colores de la interfaz para lograr un mejor contraste, o modificadores de otros aspectos del sistema.

Ejemplos de estos últimos son:

- **Farfalla project** [Mangiatordi]: una aplicación que brinda diversas herramientas para mejorar la accesibilidad de las páginas web. Se trata de una barra de herramientas que se instala en el explorador y que permite configurar ciertas características sobre los sitios durante la navegación. Entre estas configuraciones se encuentra: La posibilidad de agregar un teclado virtual, agrandar el tamaño de la fuente de texto o transformarla en formato negrita, cambiar los colores de la pantalla a modo de generar un mayor contraste entre el texto y su fondo y agrandar el tamaño del cursor para que sea más fácil de seguir.

Figura 2.3.3: Capturas de pantalla de la aplicación "Farfalla Project", se muestra como los colores de una página web son modificados por la aplicación para mejorar el contraste de la página.

- Por último, en esta categoría también se encuentran herramientas de refactoring de accesibilidad como la presentada en [Garrido et al. 2014], y herramientas de transcoding, como la presentada en [Asakawa and Takagi 2000], dado que ambos modifican las aplicaciones originales para obtener un sistema más accesible y por lo tanto es donde también ubicaríamos a nuestra aplicación.

Capitulo III
Descripción del proyecto

En este capítulo procederemos a describir el proyecto que realizamos el cual consiste en el desarrollo de una herramienta que llamamos "WAT".

3.1 Introduciendo a WAT

WAT, acrónimo de "***W**eb **A**ccesibility **T**ranscoder"* (Transcodificador de Accesibilidad Web), es un framework de software libre y gratuito que hemos estado desarrollando cuyo objetivo es mejorar el acceso a la información de páginas web de sus usuarios finales, es decir, aquellos usuarios que padecen discapacidades visuales. Nace como iniciativa para **mejorar la experiencia de navegación web de usuarios que utilizan lectores de pantalla** dado que sabemos que una mejor experiencia y usabilidad da como resultado mayor accesibilidad. Buscamos esta mejora de usabilidad ofreciendo herramientas que aumenten el nivel de granularidad en el que los elementos son reconocidos por los lectores de pantalla (de grano fino a grano grueso) pudiendo ofrecer una navegación a más alto nivel y así facilitando la interpretación del contexto en el cual el usuario se encuentra.

Para lograr estas mejoras en accesibilidad WAT provee de información semántica adicional a los elementos presentes en el sitio web. WAT necesita de la participación de usuarios voluntarios sin discapacidades visuales que contribuyan con el aporte de esta información adicional, la cual queda almacenada en archivos de texto que denominaremos **plantillas**. WAT luego hará uso de estas plantillas para realizar un proceso de "parsing" sobre la página web: proceso en el cual se interpreta la información presente en la plantilla y se recorre el código HTML de la página web añadiendo nueva información semántica. De esta forma los usuarios finales pueden, mediante WAT, descargar estas plantillas (cada una relacionada a un único dominio web) y así obtener mayor información semántica para un sitio determinado. Vemos a este conjunto de tareas como un proceso de "transcodificación" dado que WAT recolecta información (en su mayoría se tratará de información semántica) la cual será reintroducida en la página web en un nuevo formato (código HTML) fácilmente interpretable por otros sistemas informáticos.

Para facilitar la solicitud, creación y transferencia de estas plantillas, WAT a su vez comprende una plataforma de Crowdsourcing en donde usuarios voluntarios pueden publicar las plantillas creadas para cada sitio y usuarios finales pueden descargar y solicitar nuevas plantillas para sitios web particulares.

WAT brinda la posibilidad a usuarios voluntarios de contribuir con la generación de mayor cantidad de sitios web accesibles, sin necesidad de la intervención de los programadores propietarios de la página web ni la necesidad de poseer conocimientos

técnicos avanzados. En la Figura 3.1.1 podemos observar donde WAT se posiciona para enriquecer la accesibilidad de las páginas web:

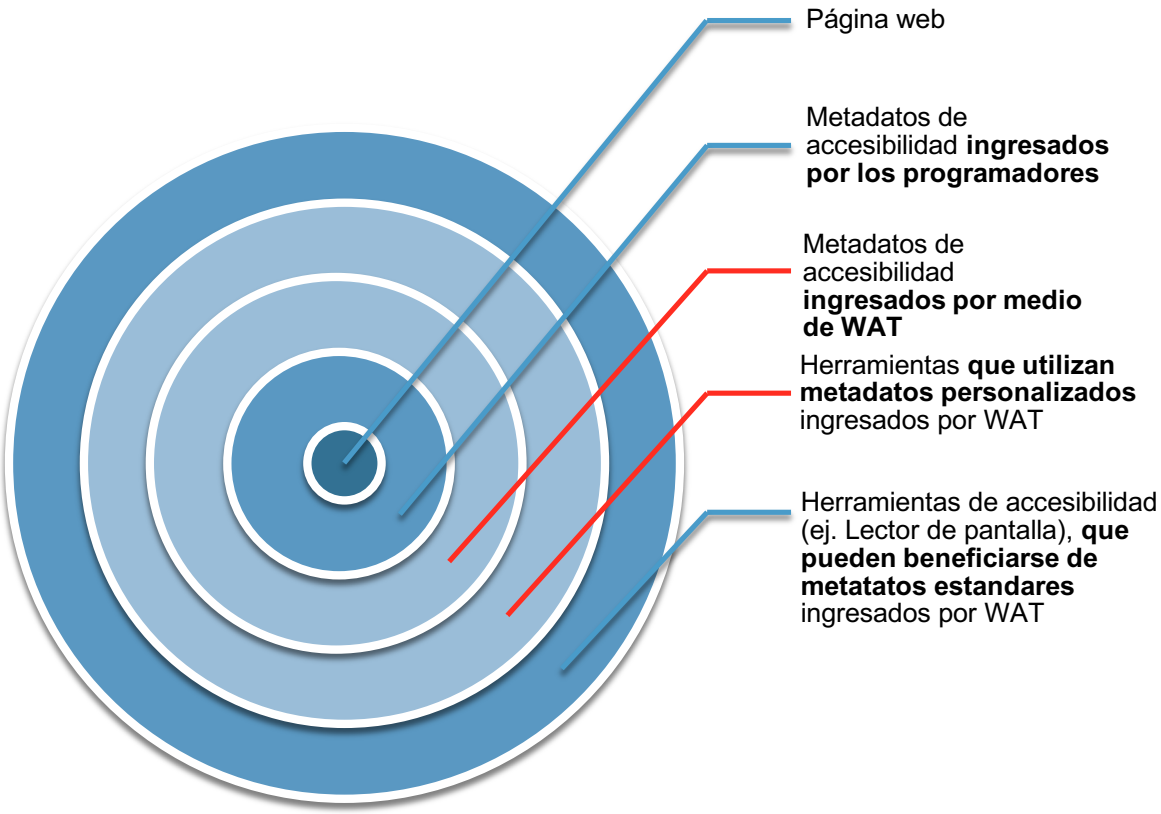


Figura 3.1.1: Ilustración del posicionamiento de WAT en el enriquecimiento de la accesibilidad web.

3.2 ¿Qué ofrece WAT?

En esta sección hablaremos de los conceptos y funcionalidades que comprenden a WAT. Marcaremos cada uno de ellos con un número para poder describirlos en detalle más adelante:

> **WAT** permite **(1) la identificación y agrupación de elementos del DOM** (código HTML) en grupos que denominaremos **(2) "objetos semánticos".** Un objeto semántico define una estructura reconocible dentro del código HTML de una página web como un único objeto, el cual puede o no repetirse. WAT aplicará **(3) reconocimiento de patrones** cada vez que un nuevo objeto semántico sea definido para identificar aquellas estructuras que concuerden con el mismo y así reconocerlas como objetos semánticos similares, de esta forma WAT busca agilizar el proceso de identificación de todos los objetos que componen una página web.

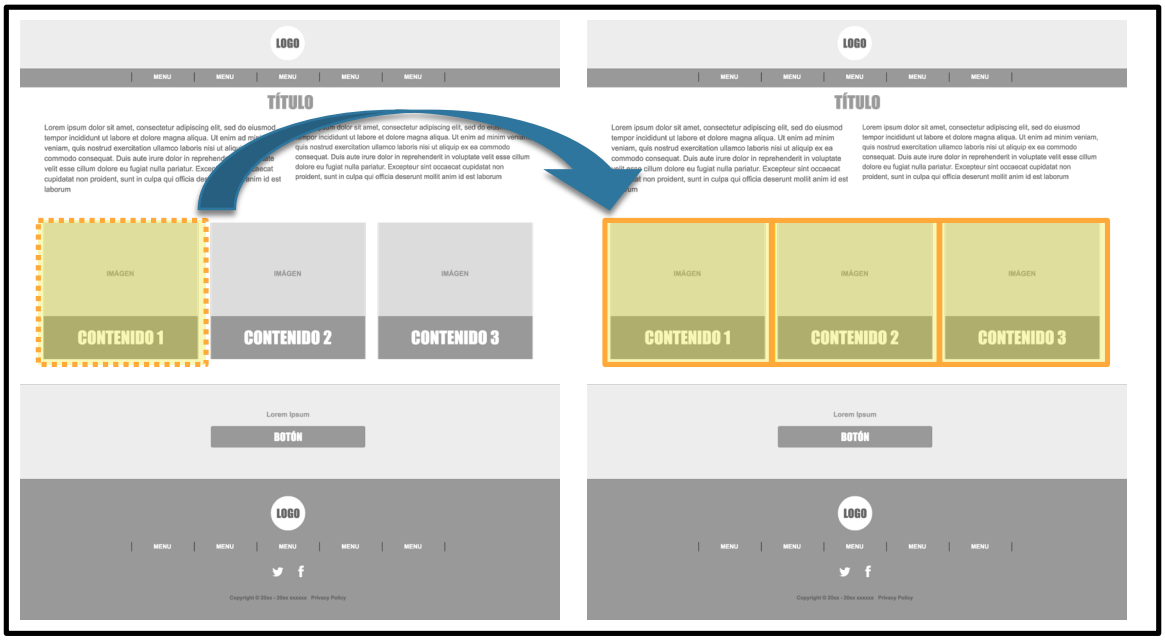


Figura 3.2.1: Ejemplo del reconocimiento de objetos semánticos similares. A la izquierda vemos como el usuario voluntario reconoce un objeto semántico, y a la derecha vemos cómo WAT identificará estructuras similares para asignarles el mismo valor semántico.

> La identificación de objetos semánticos permite **(4) capturar la información visual reconocible únicamente por la percepción humana.** Esta captura de información se logra gracias a la participación de usuarios voluntarios que no solo tienen la tarea de reconocer e indicar cuales son los objetos semánticos que visualizan dentro del sitio web sino además de **(5) introducir información semántica adicional para cada uno de ellos**.

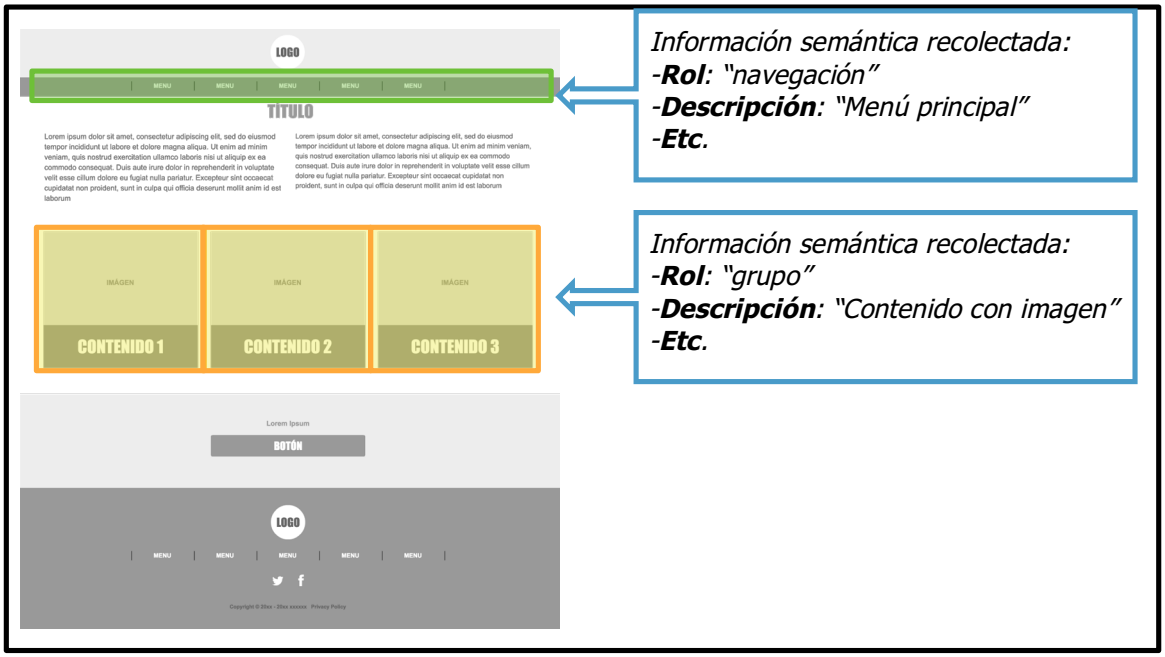


Figura 3.2.2: Ejemplo de la información semántica que se asocia a un objeto semántico.

Una vez que el usuario voluntario considere finalizada la identificación de objetos semánticos, toda esta información será almacenada en una **(6) plantilla** (un archivo de texto que se guardará en el servidor de WAT y estará asociado al dominio de la página web de la cual corresponde).

Finalmente, cuando un usuario final (usuario con discapacidad visual) ingrese a un sitio para el cual ya se dispone de una plantilla creada anteriormente, WAT procederá a descargar e instalar la plantilla en el navegador del usuario final. Dicha instalación consiste en un proceso de **(7) "parsing" que altera el formato original del código HTML de la página web**. Esta alteración se produce introduciendo información semántica adicional en el código HTML de la página para brindar una mejor experiencia al usuario discapacitado. La información adicional puede consistir en **(8) información que mejore la interpretación realizada por herramientas tales como lectores de pantalla** (por ejemplo: insertando atributos WAI-ARIA) o **(9) información a ser utilizada por nuevas herramientas para brindar nuevas funcionalidades.**

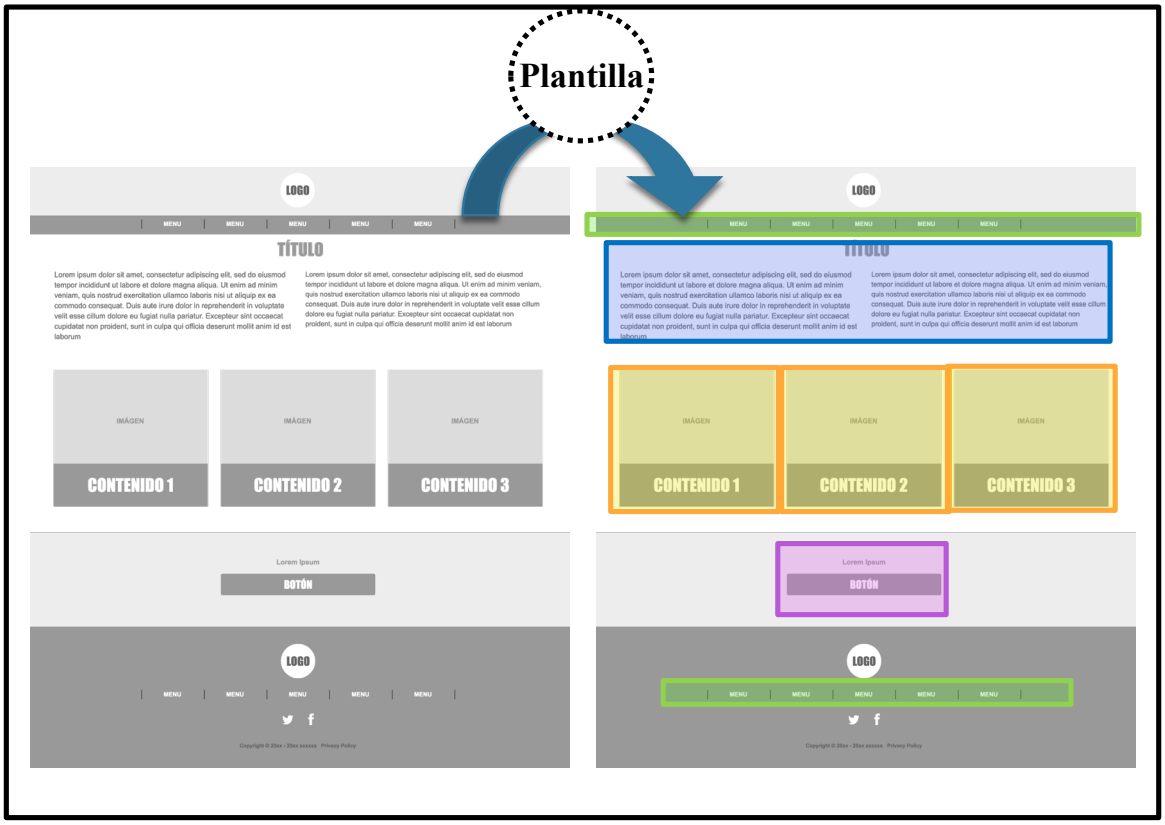
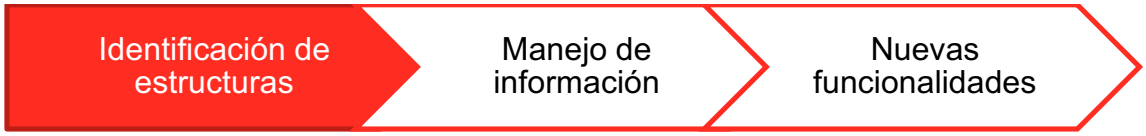


Figura 3.2.3: Ejemplo de instalación de plantilla. A la izquierda vemos la página web original y a la derecha vemos como los objetos semánticos han sido reconocidos tras la instalación de la plantilla donde la información semántica es incluida en cada elemento que concuerda con un objeto semántico definido. Los recuadros de un mismo color indican un mismo tipo de objeto semántico reconocido.

A continuación entraremos en detalle en cada uno de los 9 puntos destacados en los párrafos anteriores. Para facilitar la comprensión los clasificaremos en:

Figura 3.2.1: Clasificación de las etapas destacadas en la descripción de WAT.

3.2.1 Identificación y agrupación de elementos del DOM



Los lectores de pantalla leen el texto presente en el objeto HTML que está seleccionado (es decir que poseen el foco). Tradicionalmente no todos los elementos HTML pueden ser objetivos del foco, aunque esto suele variar dependiendo del navegador web que se utiliza. Generalmente los elementos HTML que corresponden a links (<a>), campos de formulario (<input>, <textarea>, <select>, <radio>, etc) y botones (<button>) pueden ser objetivos del foco por defecto. Pero para aquellos elementos que no poseen esta propiedad por defecto, el desarrollador deberá asignarles el atributo tab-index con un valor numérico positivo para que pase a formar parte de la secuencia de elementos seleccionables por medio del foco. Por otro lado si a un elemento HTML se le asigna el atributo tab-index en -1, este será eliminado de la secuencia de elementos seleccionables, más allá de poseer esta propiedad por defecto [Chrome]. Una vez definida esta secuencia, el usuario puede navegar estos elementos tan solo con presionar la tecla TAB y retroceder en la secuencia por medio de la combinación de teclas SHIFT+TAB.

Supongamos que ahora tenemos la tarea de leer e interpretar un articulo de Wikipedia, no podemos ver la pantalla pero poseemos un lector de pantalla que nos leerá el elemento que posea el foco asignado. Dado que no vemos la pantalla debemos utilizar el teclado para navegar por él. Mediante la navegación utilizada con la tecla TAB trataremos de realizar la tarea de interpretación. La Figura 3.2.1.1 muestra resaltado en recuadros rojos aquellos elementos que se encuentran en la secuencia de elementos seleccionables por medio del foco. Podemos notar que los elementos seleccionables son de grano fino, es decir, que son elementos simples (que en su mayoría no poseen otros elementos HTML en su interior).

Si tratamos de interpretar la página de Wikipedia utilizando los recuadros ofrecidos por la Figura 3.2.1.1 probablemente no solo demoremos un buen rato navegando por la gran cantidad de recuadros, sino que además solo escucharíamos palabras aisladas y no seremos capaces de acceder a gran parte de la información por no ser seleccionable (o mejor dicho accesible) lo que va a dificultar mucho la tarea de interpretación de la información.

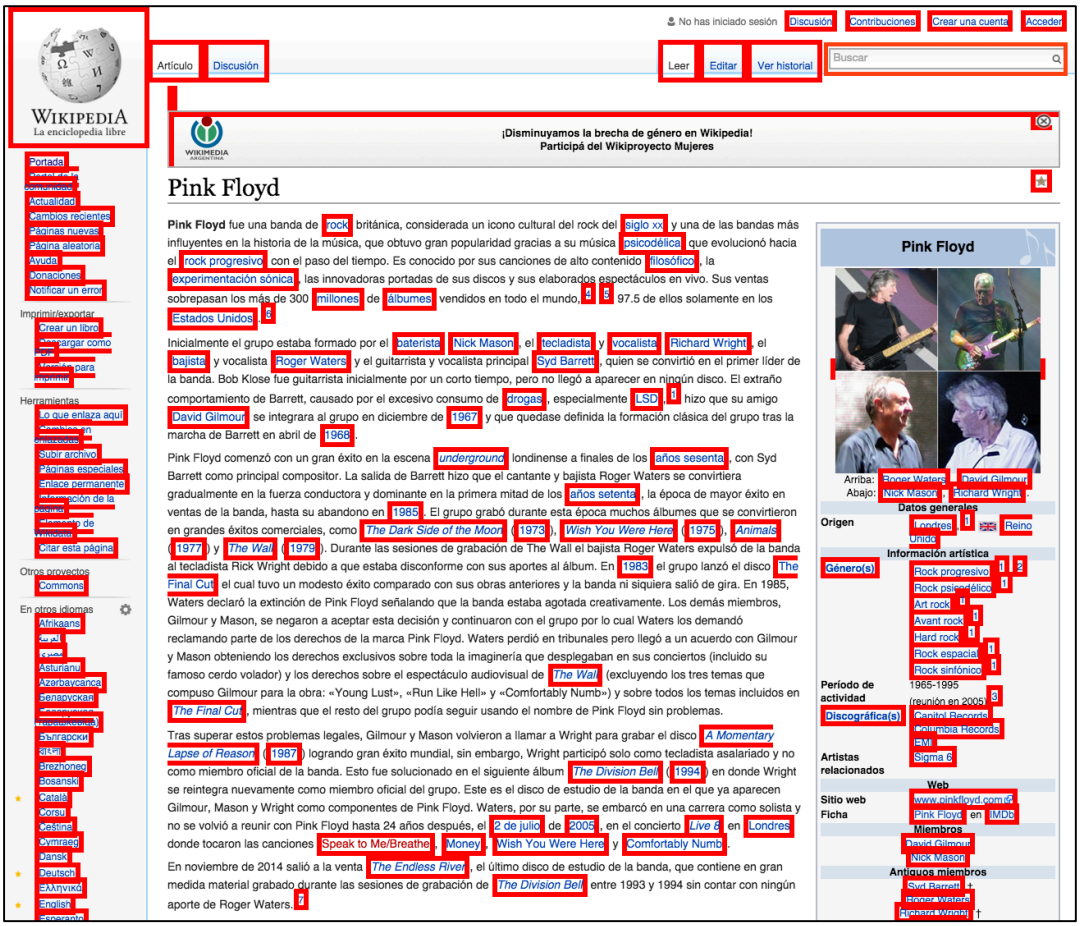


Figura 3.2.1.1: Ejemplo de granularidad fina. Captura de pantalla del sitio web Wikipedia con cuadros rojos que denotan los elementos accesibles por medio de tecla TAB.

Por suerte, los lectores de pantalla ofrecen al usuario combinaciones de teclas personalizadas las cuales no suelen hacer uso de la tecla TAB ni únicamente del atributo TAB-INDEX, sino que navegan todos los elementos del DOM identificando sobre todo aquellos que poseen texto para leer y permitiendo avanzar línea por línea o de a párrafos.

La Figura 3.2.1.2 muestra un ejemplo de cómo sería la granularidad de los elementos tras navegar utilizando los comandos ofrecidos por un lector de pantalla convencional. Se puede apreciar un grado de granularidad más gruesa al ejemplo anterior donde elementos que antes no eran accesibles ahora lo son gracias a los comandos adicionales de los lectores de pantalla.

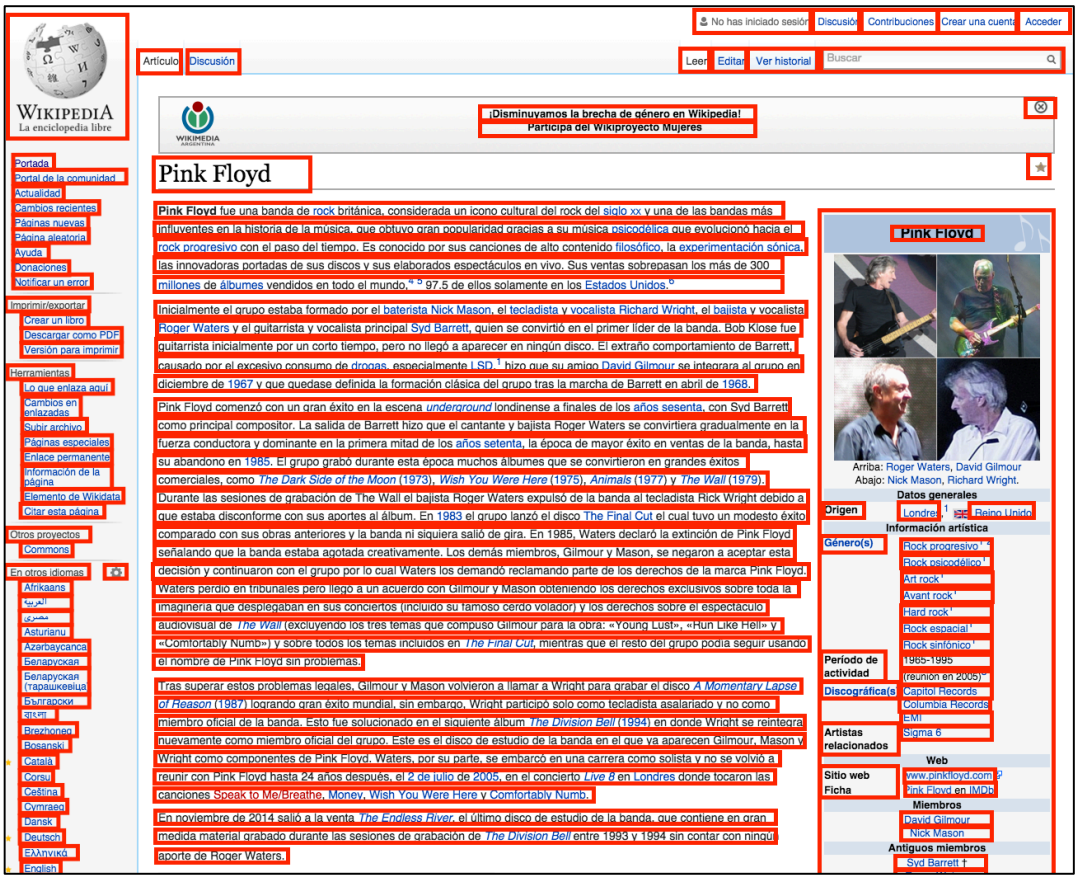


Figura 3.2.1.2: Ejemplo de granularidad media. Captura de pantalla del sitio web Wikipedia con cuadros rojos que denotan que elementos son accesibles por medio de lectores de pantalla.

Si bien los lectores de pantalla ofrecen una interfaz más amigable que el caso de la Figura 3.2.1.1, aun así hemos detectado ciertas problemáticas que ocurren en varios de ellos a la hora de la navegación:

1. La navegación no es muy intuitiva y queda definida por cada lector de pantalla muchas veces sin considerar ningún criterio semántico o no funcionando como se esperase, **dificultando** la tarea de **interpretación del contenido web**.

2. Utilizan un sistema de navegación genérico en vez de una navegación adaptada a cada sitio web particular, por lo cual, la **experiencia de usuario varía demasiado dependiendo de la estructura** de cada sitio web.

3. Hemos llegado a casos en los que el lector de pantalla **no se desplazaba correctamente por los elementos**, se queda estancado en un elemento HTML o se producen **ciclos en la navegación** sin poder continuar con el resto del documento.
4. Varios de los lectores de pantalla ofrecen diferentes comandos: uno para desplazarse hacia el elemento hermano siguiente, otro para desplazarse hacia el elemento hermano anterior, otro para subir en el árbol de DOM y otro para

ingresar a los elementos internos del elemento seleccionado. Si bien estos controles son correctos, dado que el usuario posee dificultades para visualizar la página es imposible saber **cual es la dirección correcta para desplazarse** (Si es que podemos seguir ingresando dentro del elemento o si posee más hermanos, etc.).

5. **Recorrer todo el sitio aún suele demorar mucho tiempo** debido a la lectura secuencial. Esto dificulta no solo la movilización dentro del sitio sino además la interpretación de su contenido (el usuario queda perdido entre tantos elementos que lo rodean).

Finalmente la Figura 3.2.1.3 muestra un ejemplo de cómo podrían agruparse los elementos, considerando como se relacionan los elementos del sitio y obteniendo una granularidad más gruesa, es decir de más alto nivel.

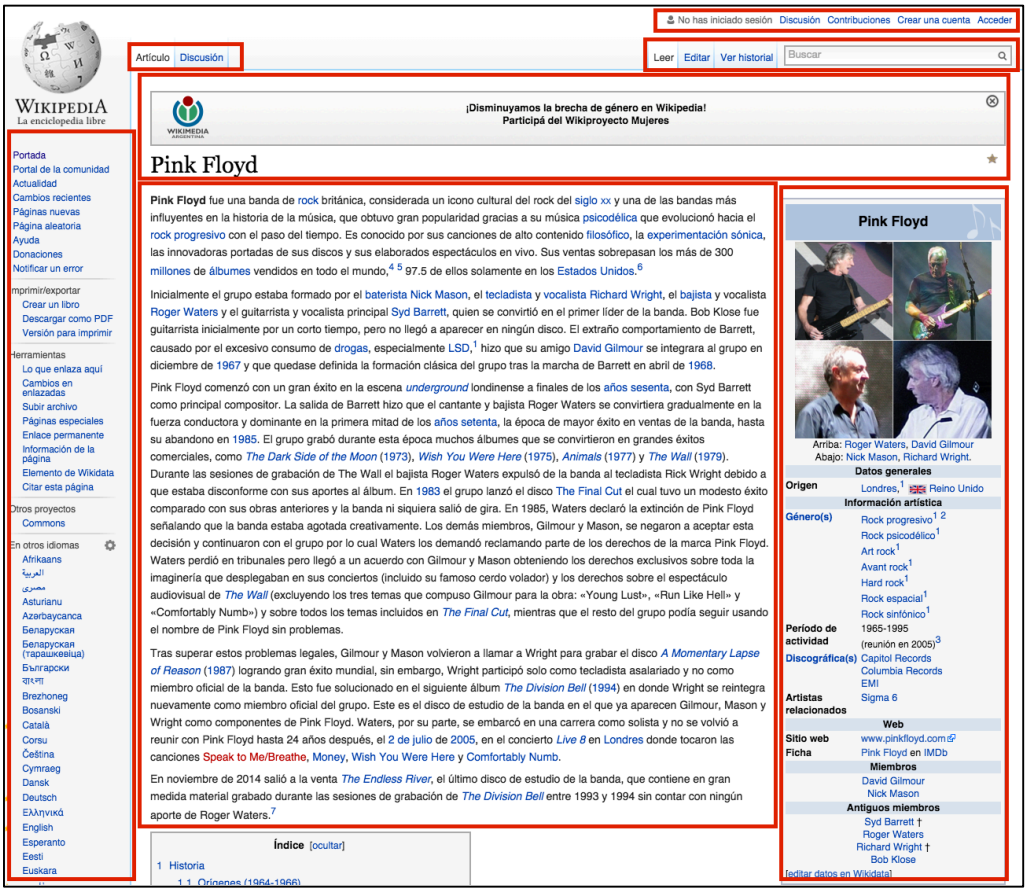


Figura 3.2.1.3: Ejemplo de reconocimiento de elementos con granularidad gruesa. Captura del sitio web Wilkipedia con cuadros rojos que denotan los grupos accesibles en primer instancia, dichas agrupaciones aumentan la granularidad en la navegación.

Si tratamos de interpretar la página de Wikipedia utilizando los recuadros ofrecidos por la Figura 3.2.1.3 la lectura de cada uno de los recuadros sería más extensa pero a la

vez habría muchos menos recuadros por recorrer; sería mucho más fácil para el usuario "escanear" la página y comprender como esta compuesta rápidamente dado que solo debería recorrer menos de 10 recuadros y ayudando a la mitigación de los problemas 1* y 5*. Sin embargo surgen a la vez nuevos:

6. El contenido a leer por el lector de pantalla es **muy largo**
7. El usuario pierde la posibilidad de **interactuar en mayor detalle** con los elementos del sitio.

Si recapitulamos todos los problemas mencionados, estos quedan resumidos en:

1. Dificultad para la interpretación del contenido.
2. Diferentes experiencias de usuario por sitio.
3. Fallas en la navegación (ciclos, o elementos inaccesibles).
4. Navegación poco intuitiva.
5. Navegación muy extensa.
6. Contenidos de lectura largos.
7. Falta de interacción detallada.

Vemos que el aumento de granularidad no ofreció por si sola una gran mejora en la mitigación de los problemas mencionados (a excepción tal vez de 1* y 5*), de hecho introdujo nuevos problemas (6* y 7*). Pero luego veremos que cumple un rol importante en el enfoque que estamos proponiendo y que, en conjunto con otras funcionalidades, pueden brindar una mejor solución para obtener una mayor accesibilidad web. Por ejemplo:

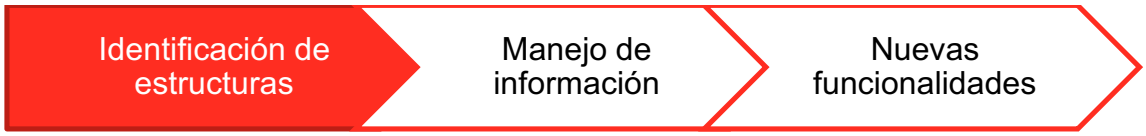
Para resolver **6***: si al mismo tiempo cada agrupación que vemos en la Figura 3.2.1.3 tuviese un nombre o etiqueta asociada (información semántica) tal como una palabra u oración que exprese la esencia del grupo (por ejemplo: "menú lateral", "Contenido principal", "Información básica", "Buscador", etc.), y si el lector de pantalla pronunciase dicho nombre previo a iniciar la redacción del contenido del grupo, entonces el usuario podría comprender rápidamente en que "sección" se encuentra y así decidir si precisa continuar escuchando el resto de ella o no.

Para resolver **7***: si una vez que el usuario hace foco en una agrupación, le permitimos al usuario "ingresar" dentro de la misma para obtener mayor detalle de ella, de esta forma el usuario podría interactuar con las secciones "ingresando" y "saliendo" a distintos niveles de granularidad. Esta vez, al interactuar con elementos de menor granularidad, el usuario podrá saber que dichos elementos corresponden a elementos internos de la "sección" que venía navegando y de esta forma poseer una mejor comprensión del contexto en el que se encuentra posicionado.

Más adelante veremos como aplicamos esas soluciones, lo importante a destacar ahora es que al existir un grado de granularidad mayor se reduce la cantidad de opciones posibles del recorrido y así es más fácil y rápido recordar y comprender las secciones que componen la página. Por lo tanto, una mayor granularidad es bueno, siempre que

podamos garantizar la resolución de los problemas **6*** y **7*** mencionados anteriormente y las secciones por las cuales dividamos la página tengan "sentido", es decir, una lógica por la cual se divide y asocian de tal manera. Es por eso que WAT integra el concepto de "**objetos semánticos**", termino que hemos utilizado para describir a las "agrupaciones con sentido" que a su vez nos permitirán obtener un mayor grado de granularidad.

3.2.2 Objetos semánticos



La mayoría de los elementos de las páginas web no actúan individualmente, sino que forman parte de grupos de elementos cuya combinación dan a la creación de lo que denominamos **objetos semánticos**, agrupaciones que poseen un significado de más alto nivel. Por ejemplo, en la Figura 3.2.2.1 se muestra una captura del e-commerce "Mercado libre", la cual posee un cuadro rojo que encuadra a una serie de elementos (enlaces, palabras, números, imágenes, etc.) que en conjunto forman un **objeto semántico** el cual semánticamente nos representa a "un producto del sitio".

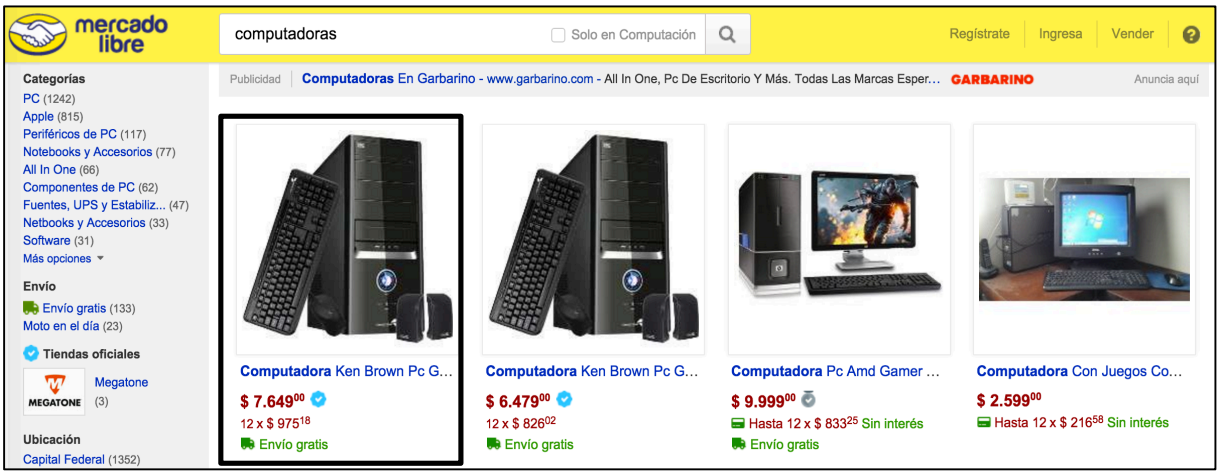


Figura 3.2.2.1: Captura de pantalla del e-commerce "Mercado libre" con recuadro rojo que denota a un objeto semántico.

Pero, ¿Por qué vemos a este conjunto de elementos como un objeto semántico?, ¿Qué es lo que relaciona a estos elementos? Inspeccionemos nuevamente la Figura 3.2.2.1 y pensemos en qué es lo que nos transmite: *vemos una cuadricula con 4 imágenes, debajo de cada una hay un enlace seguido de un número, seguido de otro número y en varias de ellas las palabras "envío gratis".*

Rápidamente (y conociendo la naturaleza de los e-commerce), asociamos que cada una de estas imágenes pertenece un producto en cuestión, que el link debajo de ellas es un enlace que nos guiará a la compra de dicho producto, el primer número representa el precio del mismo, el segundo la opción de pagarlo en cuotas y por último un texto que nos dice si posee envío gratuito o no.

Para comprender esta información los programadores del sitio organizaron dicha información de manera tal que cada producto posea sus características en una misma celda separada por un margen blanco del resto de los productos y de esta forma generarnos la ilusión de que estos **elementos están relacionados** (es decir, que ese precio que vemos hace relación al producto que se encuentra en la imagen superior y no al que se esta a su derecha o izquierda).

Todas estas asunciones no son nada más y nada menos **preconceptos que hemos adquirido** a lo largo del desarrollo de las interfaces web, también llamados metáforas de interfaz en la comunidad de HCI. Estamos acostumbrados a estos diseños y nos resulta natural y "obvio" comprenderlos tal como lo hacemos. Estos conceptos son transmitidos a lo largo de la evolución de las interfaces, la mayoría surgida de elementos de la vida cotidiana (como el símbolo del diskette que simboliza "guardar", existen quienes nunca utilizaron un diskette en su vida, pero aún así asocian e interpretan el símbolo) y otras que fueron evolucionando a niveles más abstractos (como el desplazamiento entre pestañas, muchas aplicaciones ya ni siquiera generan la ilusión de que la pestaña se asemeje a la de un archivo de papel, pero aún así comprendemos cuando nos desplazamos entre pestañas distintas). Los usuarios se han acostumbrado a estos diseños de forma que les resulta muy natural reconocer los distintos significados y roles que cumplen cada conjunto de elementos tan solo por su diseño. Cuando un usuario ve por primera vez una página web, rápidamente realiza un **proceso de escaneo** el cual le permite reconocer grupos de elementos y orientarse para entender el sitio que se encuentra navegando, algo así como una "primera impresión" de la página web. Un usuario con discapacidad visual también precisa de este proceso para comprender el contexto de la página, pero dado que su percepción se encuentra limitada a los elementos reconocidos individualmente por los lectores de pantalla, resulta complicado comprender y reconocer los distintos objetos que la componen.

Llamamos **objetos semánticos** a estructuras de elementos que en conjunto poseen un significado de más alto nivel. Son grupos de elementos relacionados entre si que con tan solo verlos podemos distinguirlos como un todo y nos brindan cierta información semántica ("una barra de navegación", "un cuadro de búsqueda", "el resultado de una búsqueda", "un producto de una página", etc.). Los objetos semánticos pueden a su vez formar parte de objetos semánticos de mayor nivel.

En nuestro framework WAT, la tarea de identificar **objetos semánticos** es otorgada a usuarios voluntarios quienes pueden guiarse fácilmente para el reconocimiento de los mismos por la información visual que perciben: Los **objetos semánticos** suelen estar

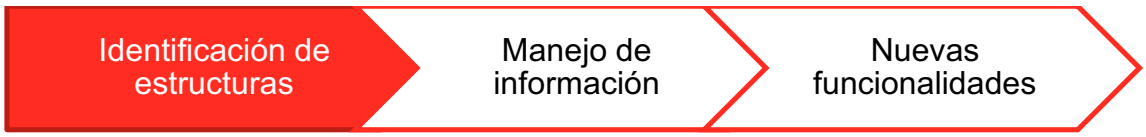
"remarcados", distanciados de otros objetos semánticos. Poseen un patrón que suele asemejarse a otros elementos del mismo sitio y/o de otros sitio similares.

Si la fotos de las computadoras de la Figura 3.2.2.1 estuviesen mezcladas, al igual que los precios y enlaces, difícilmente podríamos comprender como se asocian los datos entre si, pero la disposición creada por el diseño de la página nos permite reconocer estos objetos semánticos fácilmente (los elementos relacionados, suelen estar organizados y poseer cierto grado de cercanía).

El usuario voluntario utilizará WAT para agrupar elementos según la información visual que él mismo percibe tratando de agruparlos en objetos semánticos de más alto nivel (veremos luego como realizar este proceso en el capítulo 4). Esta agrupación nos permite obtener la **información estructural** de los objetos semánticos, es decir, los elementos HTML que los componen, pero aún hará falta capturar su esencia, es decir, la **información semántica** de cada uno.

En la sección 3.2.3 veremos como resolvemos el almacenamiento de dicha **información estructural** para identificar la presencia de los objetos semánticos luego de que han sido definidos, posteriormente en las secciones 3.2.4 y 3.2.5 hablaremos de cómo realizamos la captura de **información semántica** faltante.

3.2.3 Reconocimiento de patrones: Identificando objetos semánticos similares



WAT reduce la granularidad de las páginas web mediante la agrupación de elementos individuales pero, ¿Cómo sabe WAT que elementos agrupar?. WAT ofrece un método por el cual un usuario voluntario puede definir los objetos semánticos que reconoce para que luego sean identificados de tal forma. Sin embargo, esta tarea podría significar mucho trabajo si existieran cientos de objetos que componen a un sitio web. Afortunadamente identificamos que, por lo general, cuando una página web posee un contenido muy extenso (generalmente en páginas web 2.0), se trata de contenido que es cargado dinámicamente. Hoy en día los programadores hacen uso de "templates" (plantillas o modelos) para la generación de elementos que componen las páginas web. Dichos templates permiten definir una estructura que será reutilizada con la diferencia de poseer distintos valores de información en cada reutilización. Por ejemplo, observemos la Figura 3.2.3.1, una captura de pantalla del e-commerce "Mercado libre", donde el listado de productos posee un mismo "patrón" o estructura en la disposición de los elementos que componen a un resultado de la búsqueda.

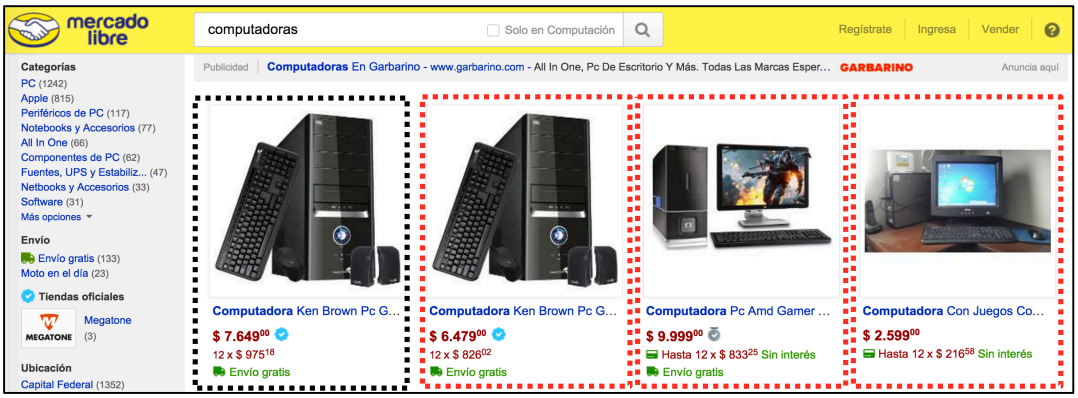


Figura 3.2.3.1: Captura de pantalla del e-commerce "Mercado libre", que muestra como un mismo "template" a sido utilizado para la generación del listado de productos.

Este es el mismo ejemplo que utilizamos en la sección anterior para describir a los objetos semánticos, y es que los objetos semánticos poseen la características de que muchas veces suelen repetirse en distintas partes de un mismo sitio, dado que suelen ser generados en base a un mismo *template*: objetos similares pero con diferente información.

Es por eso que cuando se registra un nuevo objeto semántico, posteriormente WAT intentará encontrar estructuras similares para clasificarlas como el mismo objeto semántico y así agilizar el proceso de identificación de objetos semánticos que componen al sitio web. El objetivo de WAT será entonces reconocer aquellos *templates* que generaron a los objetos semánticos similares. Por ejemplo, la Figura 3.2.3.2 muestra un ejemplo de un código HTML de cómo se vería un *template* al utilizar el framework *AngularJS*.

```
<div ng-repeat="res in resultados">
    <img src="{{res.url}}"></img>
    <h2>{{res.nombre}}</h2>
    <a href="{{res.link}}"></a>
    <p>{{res.descripcion}}</p>
</div>
```

Figura 3.2.3.2: Ejemplo del código de un template escrito con el framework AngularJS. El termino "ng-repeat" itera sobre una colección de resultados. En cada iteración la variable "res" adopta el valor del resultado en la iteración actual. Por medio de las dobles llaves "{{}}" se acceden a los atributos del ítem iterado y se crea un objeto HTML que se diferencia del resto únicamente en los valores que adoptaron los atributos de cada uno de los ítems iterados.

Para realizar la tarea de identificar estas estructuras "similares", partimos de la creación de un algoritmo que respeta las siguientes 3 reglas primordiales:

1. **Importa la estructura** de cómo se compone el objeto y **no la información que almacena**. Por ejemplo, si poseemos dos objetos que en su interior poseen un link, solo nos importa la existencia de dicho link y no a donde referencia cada uno.

2. **No pretendemos que las estructuras sean 100% idénticas**, sino que queremos poder otorgar cierto "margen de error". A veces los *templates* poseen elementos intermitentes que pueden o no estar presentes (por ejemplo como vimos en la Figura 3.2.3.1, algunos de los productos del e-commerce "Mercado Libre" poseen un ultimo elemento que dice "*Envío Gratis*" pero otros no). En estos casos nos interesa reconocer a estos elementos como "similares" siempre y cuando estos elementos intermitentes no cambien la esencia del objeto a reconocer.
3. **Mismas subestructuras pueden repetirse dentro de estructuras mayores**. Nos importa saber qué elementos componen la estructura pero no la cantidad de ellos (si se repiten). Por ejemplo, si quisiéramos identificar los dos objetos presentes en la Figura 3.2.3.3 como un mismo objeto semántico. En una primer impresión ambos objetos se parecen demasiado pero difieren en la información que almacenan (uno hace referencia a información del *proyecto 1* y el otro al *proyecto 2*). Como dijimos en la **regla 1**, no nos importa esta discrepancia de la información sino su **estructura**. Cuando observamos su estructura notamos que existen una serie de "subestructuras" que pueden (o no) aparecer reiteradas veces (en este ejemplo son los comentarios y las tareas) y de hecho, la cantidad de veces que aparecen en cada uno de los proyectos también varia (Proyecto 1 posee 3 comentarios mientras que el proyecto 2 posee 1). Por lo tanto, lo que quiere expresar la **regla 3** es que no ponderaremos la cantidad de veces que una misma subestructura aparece dentro de una estructura mayor, sino que solo nos importara saber que dicha subestructura existe. De esta forma logramos evitar que la similitud de dos objetos falle tan solo porque uno posee más, por ejemplo, comentarios que otro.

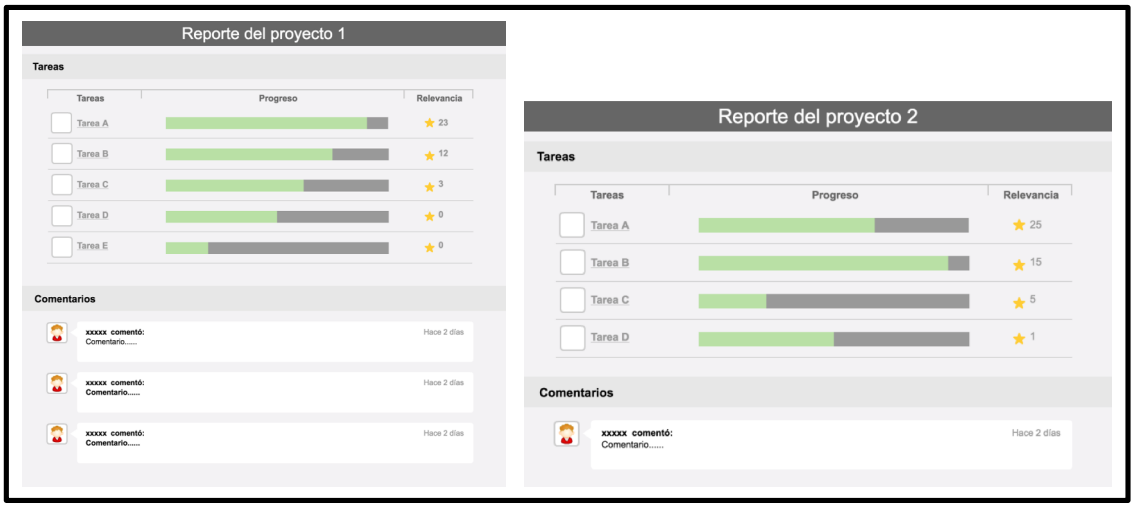


Figura 3.2.3.3: Figura comparativa que muestra dos elementos HTML que se desea reconocer como objetos semánticos similares. Notamos que ciertas subestructuras pueden estar presentes (como los comentarios y las tareas), pero que poseer 1 o varias de ellas aún no altera la esencia del objeto. Por lo que el algoritmo de similitud deberá despreciar la cantidad de veces que una misma subestructura aparece y valorar únicamente la posibilidad de que dicha subestructura exista.

Considerando el ejemplo de la Figura 3.2.3.3, procederemos con un ejemplo de cómo debería verse el patrón a generar por nuestro algoritmo de similitud para cumplir con las 3 reglas mencionadas previamente, la Figura 3.2.3.4 nos da un ejemplo gráfico de aquellas características que el algoritmo deberá apreciar para reconocer estructuras similares como las que vimos en la Figura 3.2.3.3.

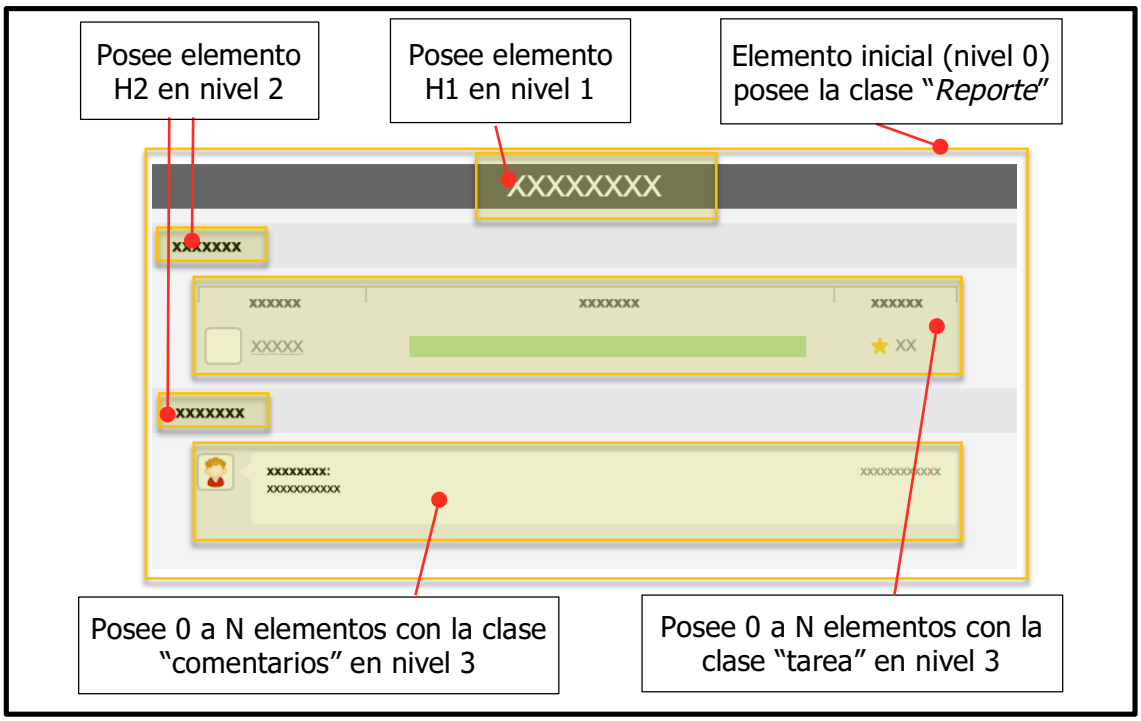


Figura 3.2.3.4: Ejemplo gráfico de la creación de un patrón. Donde se identifican un conjunto de características a cumplir para establecer que un objeto coincide con el patrón.

Siguiendo con el patrón ilustrado en la Figura 3.2.3.4, podemos ver que se establecen 5 reglas a satisfacer para que un objeto "cumpla" con dicho patrón. Si volvemos a observar los elementos de la Figura 3.2.3.3 notamos que ambos satisfacen las reglas del patrón y por lo tanto podemos decir que se trata del mismo objeto semántico.

Esta "flexibilidad" que debemos otorgar para reconocer elementos similares puede a su vez generar algunos **falsos positivos**, como el ejemplo de la Figura 3.2.3.5, en donde se muestra un objeto que semánticamente podría significar algo distinto a los objetos de la Figura 3.2.3.3, por lo que tal vez el usuario desee separarlos en un objeto semántico distinto, pero nuestro algoritmo de reconocimiento de similitud podría marcarlo como el mismo objeto semántico a los ejemplo de la Figura 3.2.3.3.

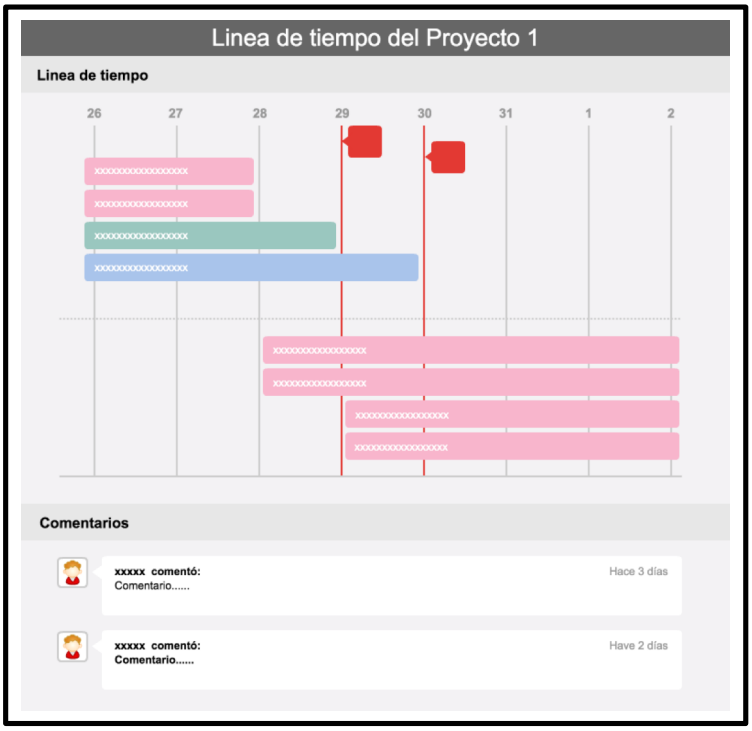


Figura 3.2.3.5: Ejemplo de elemento que podría considerarse similar a los objetos que se muestran en la Figura 3.2.3.3, pero que semánticamente debería ser tratado como un objeto semántico distinto. Si lo comparamos con el patrón de la Figura 3.2.3.4, este objeto sería capaz de cumplir con casi todas las características requeridas menos una (la existencia de "tareas"), dependerá de la flexibilidad del algoritmo para considerar este ejemplo como similar o no.

Este escenario de **falso positivo** (o bien también podrían suceder **falsos negativos**), dependerá de cómo se encuentre configurado el algoritmo de reconocimiento de similitud. El mismo cuenta con una "**variable de flexibilidad**", un numero entre 0 y 1 que expresa el porcentaje de similitud a tener en cuenta para considerar a dos elementos como similares o no. Un porcentaje de flexibilidad muy bajo causará demasiados "falsos positivos" mientras que un porcentaje de flexibilidad muy alto causará demasiados "falsos negativos" (donde 0 representa que no es necesario poseer ningún elemento en común para satisfacer la condición de "similar" y 1 representa que todos los atributos de ambos elementos deben ser idénticos).

Figura 3.2.3.6: Representación gráfica del índice de flexibilidad. Cuando tiende a 0 objetos muy distintos serán reconocidos como similares, a medida que tiende a 1 los objetos deberán parecerse más para ser reconocidos como similares.

Generando patrones

El algoritmo de similitud de WAT comienza con la creación de patrones como el que se mostró en la Figura 3.2.3.4. Un patrón no es más que una serie de "reglas" que el elemento debe satisfacer para poder decir que dicho elemento "cumple" con el patrón. Para la generación de dichas reglas, WAT procede de la siguiente manera:

1. Primero se lee el código HTML del elemento del cual se desea generar el patrón. El código HTML posee una estructura de tipo árbol n-ario como el que se observa en la parte superior de la Figura 3.2.3.7 (el cual representa un ejemplo de cómo se vería el árbol n-ario siguiendo el ejemplo de uno de los elementos mostrados anteriormente en la Figura 3.2.3.3). Para la simplicidad del ejemplo, los nodos que representan las "tareas" y los "comentarios" fueron marcados como nodos terminales para reducir la complejidad del árbol generado.

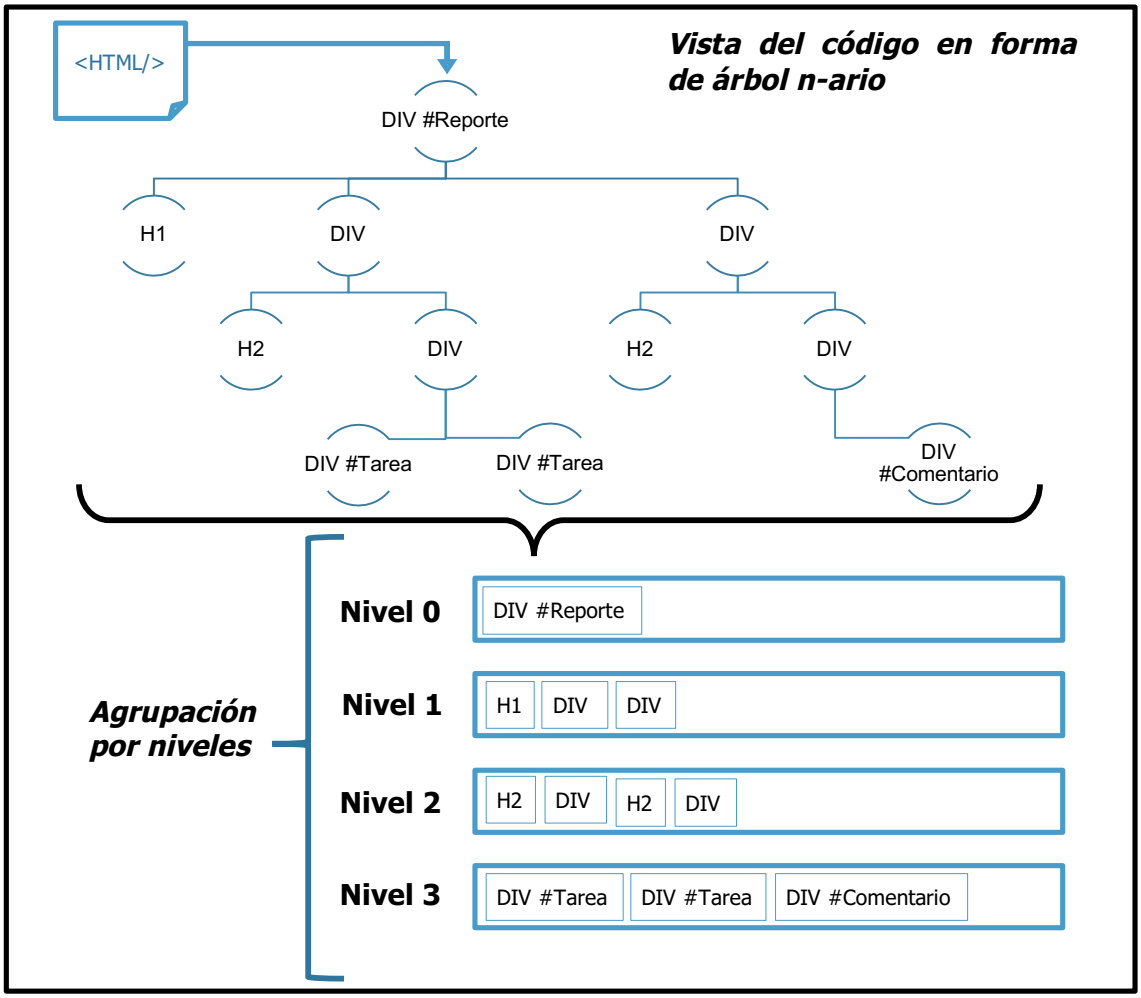


Figura 3.2.3.7: Ejemplo de cómo WAT mapea las etiquetas del árbol HTML de un elemento (ilustrado en la parte superior), organizando cada nodo por el nivel de profundidad que se encuentra (ilustrado en la parte inferior).

2. WAT realiza un recorrido en niveles (BFS) de dicho árbol. Durante este recorrido WAT organiza cada uno de los nodos de acuerdo al nivel de profundidad que se encuentran. En la parte inferior de la Figura 3.2.3.7 vemos un ejemplo de esta organización.

3. La organización mostrada en el inferior de la figura 3.2.3.7 solo sirve a modo de ejemplo para la comprensión del recorrido del algoritmo, pero en realidad WAT guardara esta información en un diccionario (HashMap), el cual brinda la ventaja de poder reconocer si un elemento se encuentra en la colección rápidamente (orden constante). La Figura 3.2.3.8 muestra como los elementos son mapeados en un diccionario utilizando **claves especiales** que identifican el nivel, tipo y valor de propiedad almacenada.

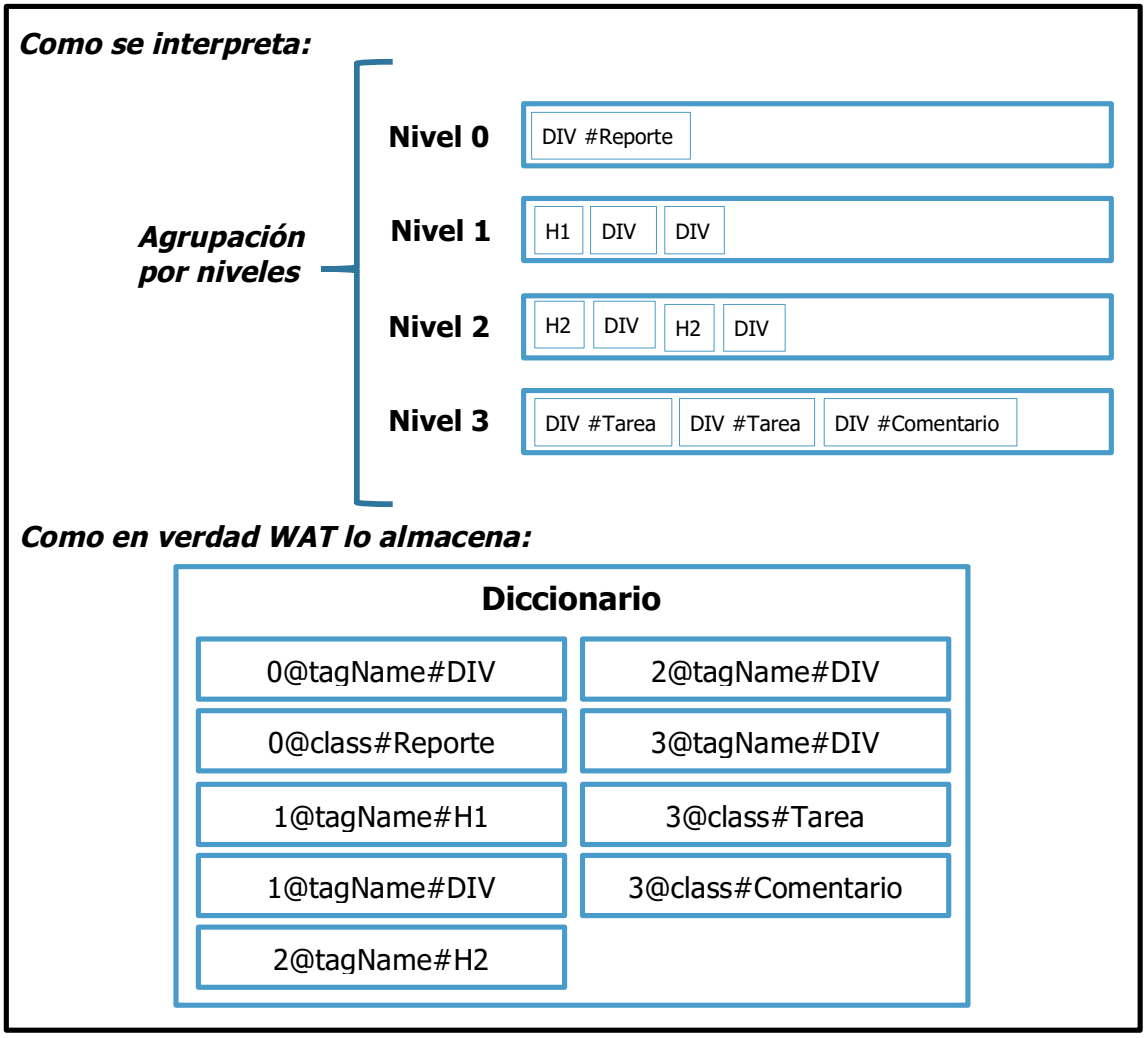


Figura 3.2.3.8: Ejemplo de cómo los atributos HTML son mapeados en un diccionario utilizando claves especiales que permiten almacenar el nivel, tipo y valor de la propiedad mapeada.

Las claves del diccionario poseen un formato especial el cual nos permiten reconocer información sobre la propiedad mapeada:

- El prefijo de la clave almacena un numero delimitado por el carácter '@'. Dicho número representa el nivel o profundidad en la cual el atributo se encuentra.
- La clave continua con un String delimitado por un '#', dicho String es una palabra que representa el tipo de atributo que esta siendo mapeado (en el ejemplo de la figura 3.2.3.8 se puede ver los casos de "tagName" y "class", indicándonos que el valor que esta siendo mapeado corresponde al nombre y clase del nodo respectivamente).
- Finalmente, el sufijo de la clave (posterior al '#') es el valor mapeado, en el ejemplo de la figura 3.2.3.8 son lo valores de los nombres y clases de cada nodo.

4. Por simplicidad, en el ejemplo de la Figura 3.2.3.8 solo se muestran mapeados valores de los atributo "tagName" y "class" de cada nodo, pero en la práctica WAT además realizará el mapeo de los atributos adicionales que encuentre de cada nodo (a excepción de algunos atributos que son excluidos por la configuración del algoritmo):

- Por cada nodo, WAT creara la clave de tipo "*x@**tagName**#NombreDelNodo*" para almacenar el tipo de nodo en cuestión.
- Por cada nodo, WAT creará una clave de tipo "*x@**class**#NombreDeClase*" para almacenar cada una de sus clases.
- Por cada nodo, WAT creará una clave de tipo "*x@**attr**#NombreDelAtributo*", para almacenar ciertos atributos no excluidos del nodo y así detectar que tipos de atributos especiales deberíamos encontrar en cada nivel.

5. Como resultado de este proceso, recibimos como **entrada** un elemento HTML (un árbol n-ario) y obtenemos como **salida** un mapa de sus características (una estructura lineal, fácil de recorrer y con acceso directo). En instantes veremos como haremos uso de este mapa para realizar la comparación de dos elementos y determinar su grado de similitud, pero antes hay unos últimos detalles que debemos aclarar:

- Notar que, dado que utilizamos un diccionario, en los casos en donde dos nodos a un mismo nivel generan exactamente una misma clave para ser almacenada en el mapa la misma será almacenada una única vez, es decir, sin repeticiones. Este comportamiento es deseado, dado que ayuda a respetar la 3er regla discutida al inicio de esta sección "**Mismas subestructuras pueden repetirse dentro de estructuras mayores**". Cuando varias subestructuras se repiten dentro de la generación del nuestro mapa, la mayoría de las claves generadas serán idénticas, de modo que evitaremos que nuestro mapa de claves crezca innecesariamente y almacenaremos estas claves una única vez.

- Finalmente, un diccionario es una estructura que almacena un par "clave -> valor", pero hasta ahora solo mencionamos su utilización para el almacenamiento de claves, entonces, **¿Qué almacenaremos como valores del diccionario?** Notemos que hasta este punto, todos los atributos que hemos mapeado a nuestro diccionario poseen el mismo "peso" independientemente de si se trata de un atributo de nivel 0 o de un nivel más profundo. Durante nuestro análisis en el reconocimiento de elementos similares detectamos que mientras más compleja (grande) es una subestructura, más información suele estar contribuyendo al concepto general del elemento padre. Es otras palabras, ciertas subestructuras de la estructura padre poseen más "**relevancia**" que otras dado que aportan mayor contenido de lo que visualizamos como objeto semántico. Veamos un ejemplo, en la Figura 3.2.3.9 volvemos a ver el mismo patrón otorgado anteriormente. Vemos que hay 2 subestructuras etiquetadas (A y B). **¿Cuál de ellas creen que aporta mayor información a la esencia de este objeto semántico?** Mientras que la subestructura A consta de un simple titulo, la subestructura B representa a una "Tarea", es decir, una componente más compleja y dado que tal como vimos antes, este objeto semántico representa el "reporte de un proyecto", que el mismo posea "tareas" en su interior aporta más a la semántica del objeto que el hecho de que esta posea un titulo. Si bien este análisis puede resultar algo complicado para determinar la relevancia final de un objeto, para los fines de nuestro algoritmo utilizaremos la "**altura**" del nodo en cuestión (no confundir con profundidad) como **heurística para determinar la relevancia** de dicho nodo dentro del mapa. Es decir, mientras mayor altura posea un elemento dentro del árbol, más elementos podría contener en su interior y por lo tanto, a de ser más compleja y aportar mayor información en comparación a otra más pequeña (reiteramos que si bien esto no es cierto en todos los casos, será utilizado a modo de heurística).

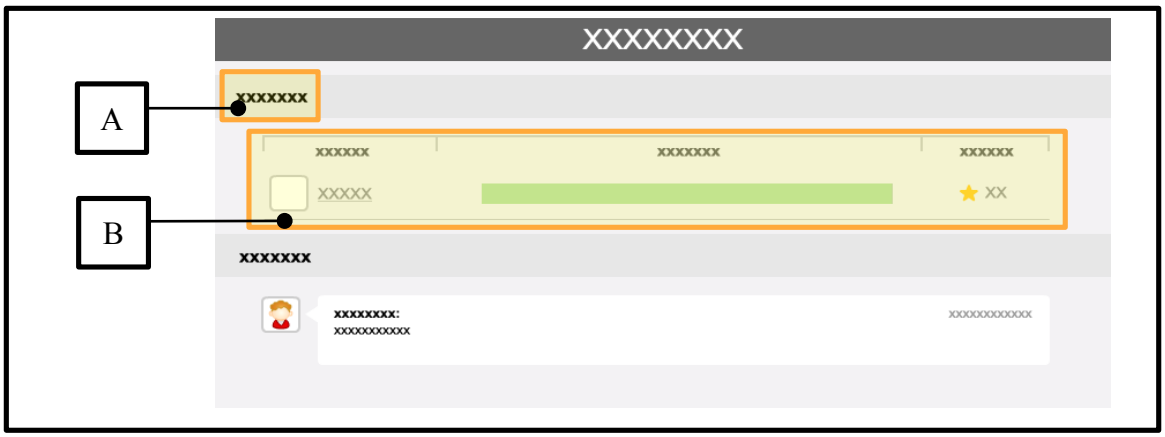


Figura 3.2.3.9: Captura de un elemento semántico, en la cual se resaltan dos subestructuras internas con distinto nivel de relevancia.

Por lo tanto, considerando el ejemplo de la Figura 3.2.3.9, obtendríamos que la subestructura A posee una altura de 1 (dado que se trata de un elemento H1 que no posee hijos), y la subestructura B posee una altura de, por ejemplo, 3 dado que en su interior posee algunos elementos hijos (a fines del ejemplo suponemos que esa es la altura que posee).
Finalmente, luego de estas ultimas consideraciones, dado un elemento concreto del código HTML crearemos un mapa de este estilo:

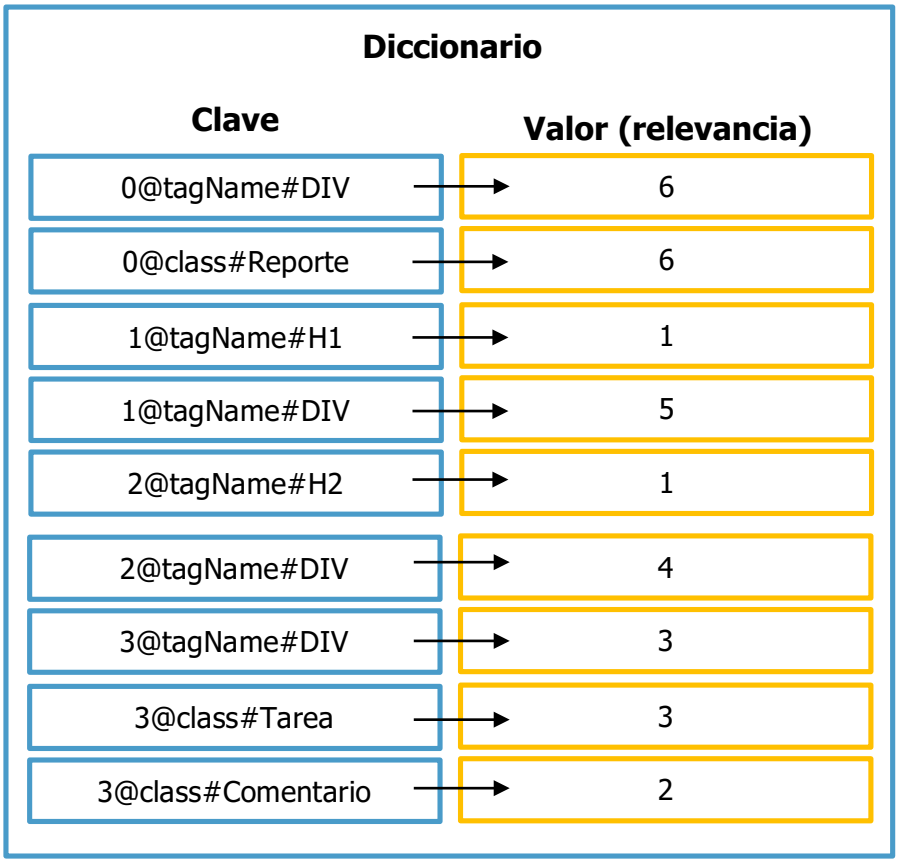


Figura 3.2.3.10: Ejemplo de mapeo generado a partir del árbol visualizado en la Figura 3.2.3.7. Las claves del mapeo dan información del atributo mapeado y los valores la relevancia que posee dicho atributo en la contribución de información del patrón. (La relevancia equivale a la altura del nodo evaluado), recordemos que el árbol de la Figura 3.2.3.7 ha sido simplificado marcando los nodos "Tarea" y "Comentario" como terminales, por esa razón en este ejemplo dichos nodos inician con una relevancia (altura) de 3 y 2 respectivamente para compensar dicha simplificación.

A continuación vemos como utilizaremos esta información para determinar la similitud de dos elementos.

Determinando si dos elementos son similares

Para la determinación de similitud de dos elementos WAT procede de la siguiente manera:

- Se genera el mapeo de atributos como el mencionado en la sección anterior para cada elemento (llamémosle **map1** y **map2**).
- Se suma la intersección de relevancias, es decir, para cada clave *K* existente en ambos mapas, se suman los valores de dicha clave en ambos mapas al total, pseudocódigo:

```
var intersection = 0;
for ( var k in map1.keys )
        if( k is in map2.keys)
                intersection += map1[k] + map2[k];
```

- Se calcula la unión total de las relevancias, es decir, la suma total de todas las relevancias en map1 y map2, pseudocódigo:

```
var union = 0;
for ( var k in map1.keys ) union += map1[k];
for ( var k in map2.keys ) union += map2[k];
```

- *Finalmente, el índice de similitud corresponde a dividir la intersección por la unión:*

```
return intersection / union;
```

- Si el índice de similitud se encuentra por encima del "*índice de flexibilidad*" configurado (también conocido como "*threshold*", o disparador), entonces se considera que ambos elementos son similares, de lo contrario no se considerarán como similares.

Este procedimiento fue desarrollado considerando el coeficiente de similitud de Jaccard [Jaccard 1908; Leskovec et al. 2011] que se formula de la siguiente forma:

$$J(A,B) = \frac{A \bigcap B}{A \bigcup B}$$

Figura 3.2.3.11: Formula del coeficiente de similitud de Jaccard

Tiempo de ejecución

Durante este análisis se utilizará la notación BigO.

Comparar 2 árboles (o elementos HTML) exactamente iguales es una tarea lineal, es decir O(N), se deben recorren todos los nodos de ambos árboles corroborando que cada nodo del árbol A sea idéntico al que se encuentra exactamente en la misma posición en el árbol B. Sin embargo, obtener un índice de similitud de dos árboles es una tarea más compleja ya que debemos considerar que es posible que ambos árboles no sean 100% idénticos, es decir, puede que los nodos de un mismo nivel del árbol A se encuentren ordenados de distinta manera en el árbol B, o que el árbol B directamente carezca de ciertos nodos (o viceversa). A su vez, puede suceder el caso que dado un nodo del árbol A, existan más de un nodo del árbol B el cual podríamos considerar como candidato para la comparación. Esto quiere decir que podríamos obtener distintos índices de similitud de acuerdo a cómo hayamos realizado la comparación entre los distintos nodos candidatos de ambos árboles. Por lo tanto, para encontrar la "comparación óptima" entre ambos árboles deberemos comparar cada subárbol del árbol A, con los subárboles del mismo nivel del árbol B y así obtener el mejor "match", es decir la mejor comparación que nos permita determinar el mejor grado de similitud de dos árboles. El tiempo de ejecución de este problema (comparar cada subárbol A con cada subárbol B) pasa entonces a ser de orden exponencial, algo del estilo $O((K^2)^H)$ donde K es el grado máximo del árbol (cantidad máxima de nodos hijos) y H representa la altura del árbol, en otras palabras: muy ineficiente.

Nuestro mapeo de los árboles a una estructura lineal (diccionario), y la comparación de las mismas utilizando el coeficiente de similitud de Jaccard, nos permite mantener el tiempo de ejecución de este algoritmo en orden lineal:

- La generación de cada mapeo equivale a un recorrido BFS de cada uno de los árboles O(N).
- La comparación equivale a un recorrido lineal sobre dichos mapeos, otra vez O(N).
- En conclusión el tiempo de ejecución general del algoritmo continua siendo O(N).

Claro esta, que al realizar este tipo de comparación estamos cediendo precisión, pero aún así ha demostrado otorgar buenos resultados y considerando la gran ventaja que obtenemos en su tiempo de ejecución es prueba suficiente para adoptar este mecanismo como estrategia para la similitud de árboles de elementos HTML.

Búsqueda de los elementos similares

Una vez generado un patrón (mapeo de un elemento HTML designado como representante), debemos proceder a buscar elementos similares (aquellos elementos cuyos mapeos, al ser comparados con el mapeo del elemento representante, estén por encima del índice de flexibilidad). Considerando que en internet podemos encontrar páginas con contenido HTML considerablemente grande y a su vez dinámico (es decir,

que puede que nuevos objetos semánticos aparezcan luego de haber finalizado la etapa de búsqueda de objetos semánticos) en cuyo caso deberemos reiniciar nuevamente el proceso para identificar aquellos nuevos objetos semánticos, es muy posible que precisemos repetir el proceso de búsqueda reiteradas veces y por lo tanto el mismo deberá ser rápido o de lo contrario podría afectar a la performance de la navegación del sitio web. Es por eso que surgimos con 3 estrategias de búsqueda de los objetos semánticos definidos, cada una con ventajas y desventajas diferentes:

1. **Búsqueda por clases**: La búsqueda por clases consiste en recorrer el mapa de atributos del patrón y seleccionar las K entradas con mayor valor de relevancia correspondientes a clases (**@class#**), independientemente del nivel en que se encuentre (**0@**, **1@**, **2@**, etc.). Una vez seleccionadas, procederemos a utilizar los selectores de clases *Javascript* (los selectores de *clase* y *id* en *Javascript* están optimizados y son más veloces que la selección por *atributo* o *tagName*), una vez recuperados todos los elementos que poseen dicha clase, consideraremos el nivel en el que se encuentra para ubicar al elemento a evaluar, es decir, si el nivel es 1 (uno), entonces el elemento a evaluar es el padre del elemento recuperado por la selección de clase. Por ejemplo:

   - Para la clave "**0@class#publicación**": Utilizaremos el selector *Javascript* para encontrar todos los elementos con clase "*publicación*". Dado que el nivel de la clave es 0 (cero) , dichos elementos seleccionados serán los que evaluaremos para integrar o no al patrón.

   - Para la clave "**3@class#comentario**": Utilizaremos el selector *Javascript* para encontrar todos los elementos con clase "*comentario*". Dado que el nivel de la clave es 3 (tres) , tomaremos el elemento que se encuentra 3 niveles arriba de cada elemento recuperado por el selector de clase. Dicho ancestro es el que utilizaremos para evaluar contra nuestro patrón.

   La principal ventaja de esta estrategia es que es rápida dado que utilizamos los selectores de clase de *Javascript*, a su vez, supongamos que el elemento principal (nivel 0) no posee clase alguna, esta estrategia aún es capaz de identificarlo si tan solo un elemento de su descendencia posee una de las clases utilizadas en la búsqueda. La desventaja de esta estrategia se encuentra cuando no existen clases para buscar como referencia (algo que no suele ser muy probable).

2. **Búsqueda por posición (*Xpath*)**: elementos similares deberían encontrarse en rutas similares; cada vez que identifiquemos que un elemento corresponde al patrón guardaremos su *xpath (ruta) genérico* (con *genérico* nos referimos al *xpath* utilizando *wildcards*, de esta forma reducimos la cantidad de *xpaths* a almacenar, dado que varios elementos generalmente producirán *xpaths* genéricos idénticos, el cual se almacena una sola vez). La búsqueda por posición consiste en recorrer todos los *xpaths* genéricos almacenados y recuperar todos los elementos que coincidan con dicho *xpaths*. Cada elementos será luego evaluado contra el patrón.

La ventaja de esta estrategia es que al igual que la estrategia por clases, es bastante rápida, y en la práctica, la cantidad de *xpaths* a almacenar por patrón no suele crecer demasiado. La desventaja es que si existen nuevas ocurrencias del objeto semántico en *xpaths* que no están almacenados, esta estrategia no los encontrará.

3. **Búsqueda total (BFS)**: Tanto la estrategia 1 como 2 poseen ventajas y desventajas similares. La ventaja es que son rápidas, pero las desventaja es que dependen de los datos almacenados, por lo que si dichos datos no están completos, entonces nunca encontrarán nuevas ubicaciones y clases del objeto semántico. Es por eso que existe la estrategia 3. La búsqueda total, consiste en realizar un BFS sobre todo el documento HTML (aunque se puede establecer un limite de profundidad para evitar que el algoritmo tarde demasiado), por cada elemento HTML recorrido, se evaluará al mismo contra el patrón. Esta estrategia es muy lenta, pero posee la ventaja que nos permitirá encontrar nuevas clases y posiciones de los objetos semánticos.

Por lo tanto, la estrategia 3 debido a su lentitud es utilizada un menor número de veces, generalmente durante la carga inicial del patrón. Luego, cada vez que debamos "refrescar" la búsqueda en caso de nuevas apariciones (debido al contenido dinámico), utilizaremos las estrategias 1 y 2 únicamente para evitar afectar la performance de la navegación del usuario.

Otras optimizaciones

- **Cache:** Cada vez que realicemos un recorrido de búsqueda de elementos, tanto si el elemento concuerda o no con el patrón, asociaremos información a dicho elemento para "recordar" dicha solución (cache). De esta forma, si en el futuro tratamos de evaluar nuevamente al mismo elemento contra el mismo patrón, podemos determinar rápidamente que el atributo o bien ya se encuentra asociado al patrón o no va a coincidir con el mismo gracias a la información insertada la primera vez.

- **Reforzamiento:** El reforzamiento es una optimización clave para la "*búsqueda por clase*", cada vez que un elemento coincida o no con el patrón procederemos a realizar un reforzamiento positivo o negativo según corresponda. Es decir, si el atributo coincide con el patrón, entonces incrementaremos la relevancia de aquellos atributos que tanto el mapa del patrón como el mapa del elemento poseen en común; por otro lado, si el elemento no coincidió con el patrón, entonces procederemos a decrementar (hasta llegar a cero, no más) aquellos atributos que poseen en común. De esta forma, a medida que incluimos elementos en el patrón, aquellas clases que los elementos del patrón poseen en común, irán tomando relevancia mientras que las clases que también poseen elementos diferentes al patrón se irán debilitando a tal punto de no poseer influencia en el patrón.

3.2.4 Capturando la información visual

La información visual cumple un rol vital en la navegación web, dado que influye directamente en la interpretación de los usuarios sobre los datos representados, una malinterpretación del contenido web es nada más y nada menos un problema de accesibilidad. En la Figura 3.2.4.1 trata de demostrar como la información visual puede interponerse entre los datos y la comprensión semántica de los elementos web. Vale la pena aclarar que no toda información depende si o si de características visuales, pero si podemos estar de acuerdo en que gran parte de ella lo hace, al menos para facilitar su correcta interpretación.

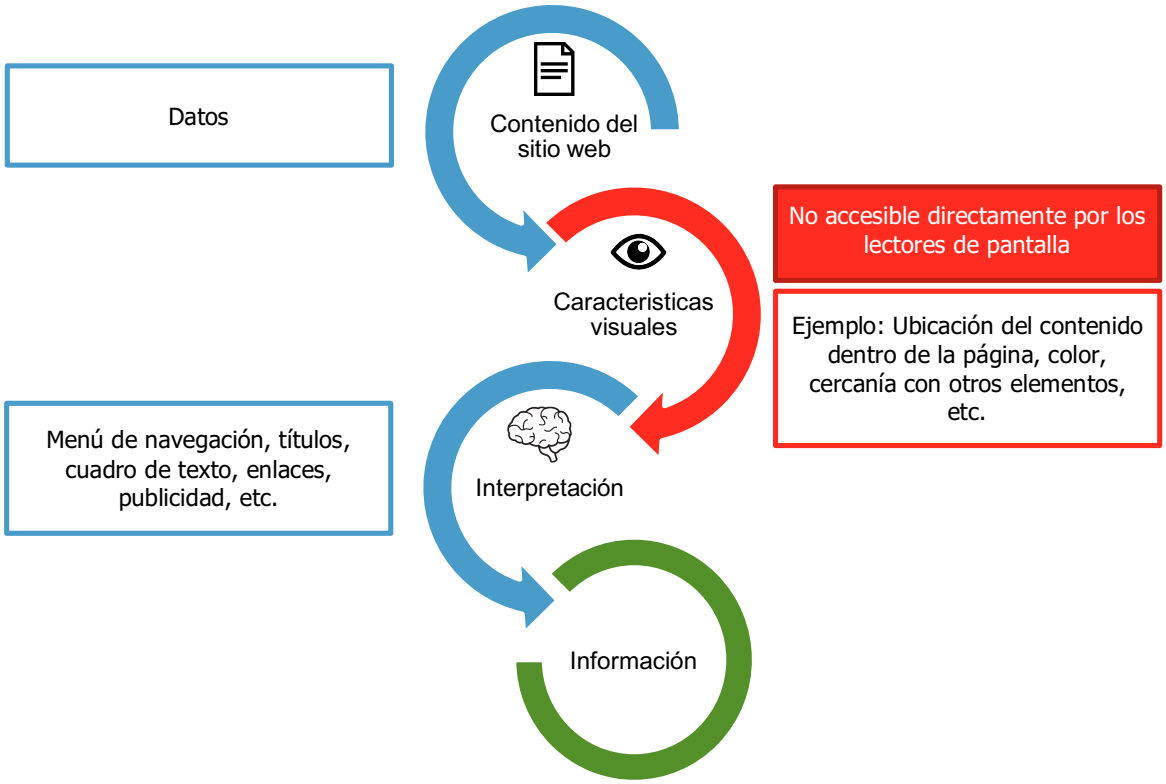


Figura 3.2.4.1: El medio visual interpuesto entre los datos y la interpretación

Demos un ejemplo de lo mencionado recientemente: Supongamos que poseemos código HTML que **sintácticamente** representa a un conjunto de enlaces. Dicha estructura sintáctica podría ser interpretada **semánticamente** de distintas formas como por ejemplo:

1. Podría representar un conjunto de acciones (como el de la Figura 3.2.4.2 que posee las acciones "Me gusta", "Comentar", "Compartir") que al presionarlas no produce una navegación sino una interacción.
2. Podría significar un menú de navegación (como el de la Figura 3.2.4.3 que posee los enlaces de "Noticias", "Mensajes" y "Eventos")
3. Podría ser un índice con enlaces internos (como el de la Figura 3.2.4.4), etc.

Figura 3.2.4.2: Ejemplo de código HTML que podría representar un conjunto de acciones en Facebook.

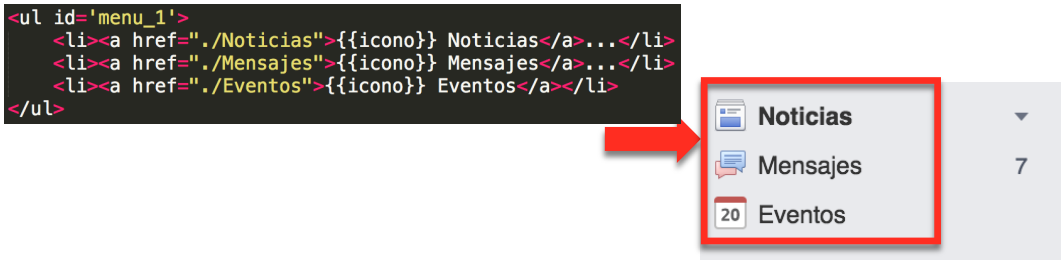


Figura 3.2.4.3: Ejemplo de código HTML que podría representar un menú en Facebook.

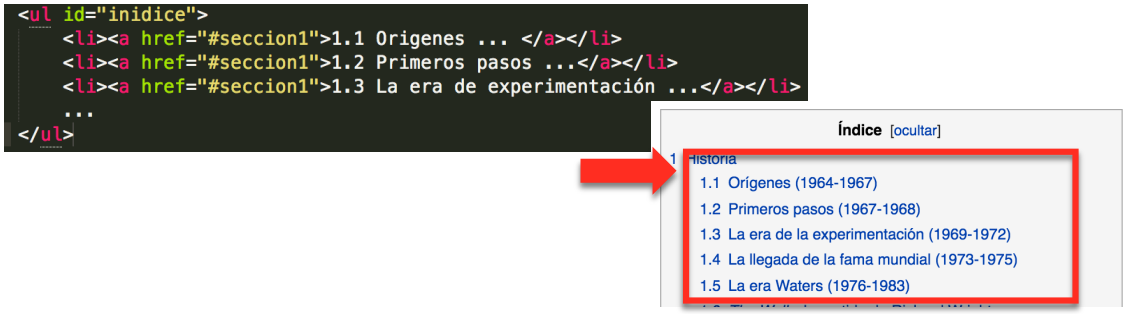


Figura 3.2.4.4: Ejemplo de código HTML que podría representar índice de enlaces internos en Wikipedia.

Aclaración: Los códigos HTML de los ejemplos anteriores no son reales sino que fueron creados a modo de ejemplo teniendo el conocimiento de que dichos elementos podrían ser escritos de tal manera y que sus resultados solo se distingan por el código CSS que no esta presente en el código HTML.

Lo importante a notar es que si bien la forma de escribir los datos en el código HTML puede ser muy similar, los resultados finales pueden variar mucho debido al código Javascript y/o CSS que alteran el funcionamiento y la forma en que se grafican los datos en pantalla respectivamente, estas alteraciones modifican nuestra percepción del elemento y por lo tanto, el significado semántico que le otorgamos. El problema que surge ahora es que lo lectores de pantalla solo leen código HTML.

Entonces **¿cómo podemos capturar esta información visual perdida?**, ya hemos hablado de como pretendemos hacer uso de la ayuda de usuarios voluntarios para el reconocimiento de los "**objetos semánticos**" que integran una página web. La información visual que da sentido a gran parte de los objetos que integran la web (y es muy difícil de interpretar programáticamente) es nada más y nada menos que información semántica y por lo tanto, si los usuarios voluntarios deben identificar dichos objetos semánticos a su vez poseerán la tarea de **ingresar nueva información semántica**.

Otra pregunta que surge ahora es **¿cómo introduciremos esta información semántica ingresada por los usuarios voluntarios?**, por suerte el WCAG ya dio un paso adelante en este tema generando un estándar de etiquetas HTML adicionales llamado WAI-ARIA, estas etiquetas permiten entre otras cosas definir semántica para ser interpretada por herramientas tales como los lectores de pantalla.

Si bien WAI-ARIA ofrece un gran conjunto de roles y propiedades para definir semántica y mejorar la accesibilidad, vale la pena aclarar que aún no todas han sido implementadas o son respetadas por todos los navegadores web y/o lectores de pantalla, de esta forma, no se puede garantizar que el simple uso de WAI-ARIA provea una mejora en la accesibilidad, dado que dependerá de las herramientas y programas que el usuario final utilice. A su vez, las propiedades definidas por WAI-ARIA no actúan por si solas, sino que es responsabilidad del programador actualizar los valores de las propiedades WAI-ARIA para que las mismas sean consistentes con el estado actual de la aplicación. Por ejemplo: La propiedad WAI-ARIA "aria-hidden" refleja el estado de visibilidad de un elemento HTML. Le corresponde al programador realizar la tarea de alternar el valor de dicha propiedad ("true" – "false") para reflejar el correcto estado del elemento, WAI-ARIA no introduce funcionalidad, más bien estandariza semántica, estados y valores.

En la próxima sección detallaremos más específicamente cuál es la información semántica que vamos a recolectar para cada objeto semántico, mencionando a la vez cuales son las propiedades WAI-ARIA que utilizaremos para esto.

3.2.5 Añadiendo información semántica: Inicio de la transcodificación

Una vez creado un objeto semántico, el usuario voluntario deberá agregar la información semántica correspondiente al objeto. Cada vez que un elemento de la página web sea identificado como un objeto semántico (debido a que concuerda con el

patrón como se hablo en las secciones anteriores) la información semántica ligada al objeto semántico será también asociada al elemento en cuestión.

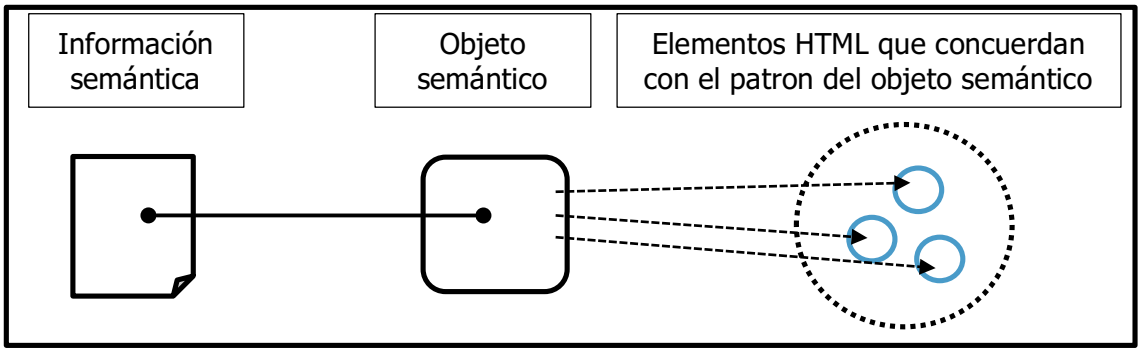


Figura 3.2.5.1: Demostración gráfica de cómo los elementos de la página web se asocian a la información semántica a través del objeto semántico que forman parte.

La información semántica es introducida a través de un formulario que se le mostrará al usuario voluntario durante la creación de un nuevo objeto semántico (aquí es donde vemos la primer instancia de la "transcodificación" donde el usuario voluntario transmite el conocimiento que posee sobre el objeto y lo plasma en datos concretos requeridos por el formulario, capturando así la información semántica del objeto). Esta información semántica luego será ligada a los elementos que concuerdan con el patrón del objeto semántico.

Ahora la pregunta es: **¿Cuál es la información que deberíamos recopilar en dicho formulario?**, tal como mencionamos antes, WAI-ARIA ofrece un conjunto de roles y propiedades que ayudan a mejorar la accesibilidad web. Un buen comienzo para nuestro formulario sería consultar al usuario voluntario que valores deberíamos otorgarles a estas propiedades WAI-AIRA (en caso de que el elemento reconocido como objeto semántico aún no las posea). Por ejemplo, las primeras entradas de nuestro formulario podrían ser las siguiente:

1. "**Nombre del objeto semántico**": Introducir una palabra o frase que ayude a identificar al objeto, dicha frase será leída por el lector de pantalla cada vez que el objeto sea seleccionado. Esta funcionalidad se obtiene agregando dicho texto como valor de la propiedad WAI-ARIA llamada **aria-label**.

2. **"Rol del objeto semántico"**: Un *select* con opciones de roles WAI-ARIA El usuario voluntario selecciona entre un conjunto de roles aquel que más corresponda al objeto semántico.

Información adicional personalizada: A su vez podríamos agregar campos personalizados a este formulario, no con el objetivo de mapear su valor a un atributo WAI-ARIA específico, sino para usar dichos valores en funcionalidades adicionales de accesibilidad. Por ejemplo: podríamos agregar un *checkbox* al formulario con el propósito de incluir/eliminar a un elemento del sitio web y luego crear la funcionalidad de que si un objeto posee dicha opción seleccionada WAT remueva a cada aparición

del objeto semántico dentro de la navegación de la pagina web (esto podría ser útil para la eliminación de publicidades u otros elementos no deseados).
La Figura 3.2.5.2 muestra una primera aproximación de cómo podría verse el formulario de información semántica que se le presentará al usuario voluntario tras la creación de un objeto semántico, notar que las primeras dos entradas se asemejan a aquellas detalladas anteriormente en los puntos 1* y 2*, mientras que el resto de las entradas hacen referencia a datos adicionales personalizados para relacionar nuevas funcionalidades en los objetos semánticos definidos.

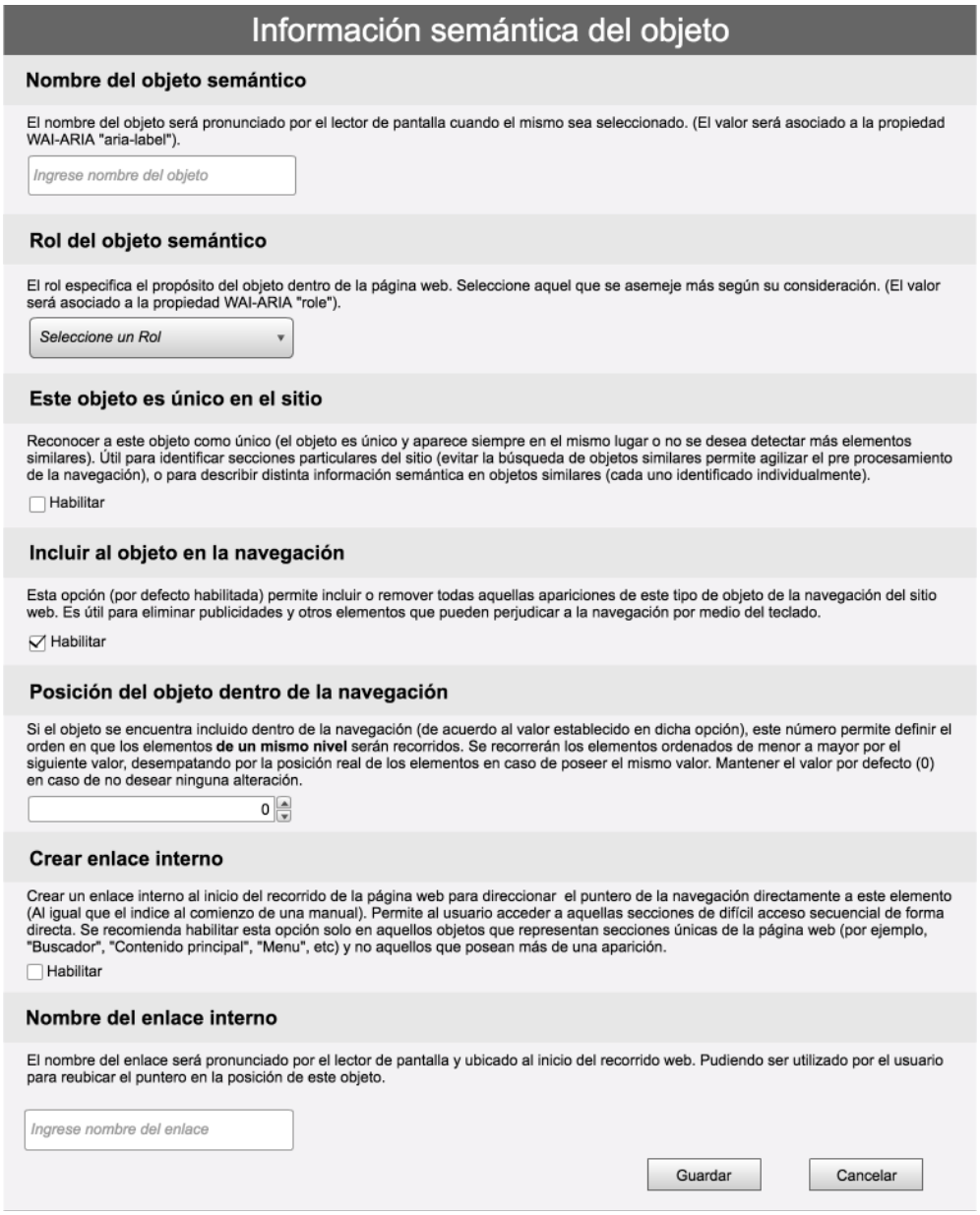


Figura 3.2.5.2: Ejemplo del formulario de información semántica para un objeto semántico de WAT. En la figura se distinguen 2 tipos de campos: Las primeras dos entradas hacen referencia a información destinada a etiquetas WAI-ARIA, el resto de las entradas corresponden a información destinada a etiquetas personalizadas para nuevas funcionalidades.

Por lo tanto, toda la información que el usuario introduzca en este formulario quedará relacionada a los elementos HTML que hayan sido identificados como tales objetos semánticos, brindando así información específica que podrá ser utilizada para otorgar al usuario interfaces más accesibles.

Selección de los roles WAI-ARIA

En total existen alrededor de unos 60 roles definidos por WAI-ARIA. Conocer todos los roles ofrecidos de WAI-ARIA no es una tarea fácil, y considerando que planeamos ofrecer un formulario que pregunte a un usuario voluntario que rol asignar a un objeto semántico, un listado de 60 roles no sería algo agradable. Es por eso que hemos realizado una selección de aquellos roles que consideramos más "genéricos" para ofrecerlos como opciones en esta primer versión de nuestro formulario de información semántica:

- **Dialog** (Widget): Define una ventana que emerge de la interfaz para interactuar con el usuario, por ejemplo popups y modals. Permite al usuario voluntario marcar aquellas regiones que aparecen dinámicamente en la página web.
- **Menu** (Widget composite): Define una lista de acciones o funciones que el usuario puede invocar (Dichas opciones deberán poseer el rol "menuitem").
- **Menuitem** (Widget): Define una acción/función dentro de un objeto con rol "menu".
- **Article** (Structure): Sección que define un contenido independiente que puede anidarse con otros contenidos independientes (elementos que también poseen el rol "article"), logrando así que el contenido quede relacionado a sus ancestros (en cascada). Es decir, cuando un sección con el rol "article" posee nodos hijos con el mismo rol, los nodos hijos representan contenido que esta relacionado al contenido del nodo "article" padre. Esto puede ser utilizado por tecnologías de accesibilidad para otorgar sistemas de navegación por jerarquías y otras funcionalidades. (Por ejemplo, con este rol seremos capaces de relacionar los comentarios de una noticia como información relacionada a la noticia en si).
- **Directory** (Structure): Define que la estructura representa una tabla de contenidos, o mejor dicho, un índice del contenido.
- **List** (Structure): Define una estructura con forma de lista.
- **Listitem** (Structure): Define un ítem dentro de un objeto con rol "List".
- **Group** (Structure): Define una sección del sitio web que no es lo suficientemente relevante como para ser incluida en la tabla de contenidos de la página (usar "Region" de lo contrario).
- **Heading** (Structure): Define a un elemento dedicado a ser un título.
- **Region** (Structure): Define una sección importante del sitio web, tan importante que debe ser considerada en la tabla de contenidos del sitio web. Estas secciones suelen estar acompañadas con al menos un elemento con el rol "heading" en su interior o referenciadas mediante la propiedad "aria-labelledby".
- **Banner** (Landmark): Define una sección que posee contenido relacionado al sitio (Por ejemplo las barras superiores que poseen links de navegación y el logo del sitio).

- **Form** (Landmark): Define una sección correspondiente a un formulario.
- **Main** (Landmark): Define la sección que posee el contenido principal de la página.
- **Navigation** (Landmark): Define una sección que posee únicamente links de navegación.
- **Search** (Landmark): Define una sección cuya objetivo es proveer un formulario de búsqueda.

Aclaración: para aquellos roles no incluidos, no se trata de que no los consideremos importantes, sino que buscamos simplificar la entrada de información que realizarán los usuarios voluntarios y por lo tanto tratamos de reducir la variedad de opciones WAI-ARIA a aquellas que consideramos más generales. Por otra parte, no buscamos que nuestro proceso de transcodificación resuelva completamente la falta de propiedades WAI-ARIA que deberían ser introducidas por los desarrolladores, sino que hacemos uso de ellas con el objetivo de mejorar ciertos aspectos básicos de accesibilidad.

En su mayoría hemos seleccionado aquellos roles que definen estructuras organizacionales (Structure roles) y de referencia (Landmark roles), dado que son las que nos permiten organizar la información general de las páginas web. Por otro lado los roles de componentes navegacionales (Widget roles) suelen requerir de mayor configuración, y por lo cual no sería fácil definir una forma "genérica" de agregar dicha funcionalidad a los sitios, es por eso que para este tipo de rol solo hemos optado por aquellas más simples y que suelen ser muy comunes como lo son los roles "menu" y "menuitem" (para definir las barras de menú de la aplicación web) y los dialogs (para proveer una forma de definir aquellas ventanas emergentes que suelen ser intermitentes dentro de la aplicación).

3.2.6 Creación de plantillas

Los objetos semánticos junto con su información semántica asociada definidos por los usuarios voluntarios son guardados en una plantilla. Las plantillas son archivos de texto en formato JSON que representan un objeto Javascript con la siguiente información:

- **Información del proyecto** (datos del autor, dominio de la página web, descripción, idioma).
- **Colección de objetos semánticos:** Cada objeto semántico a su vez se compone de:
    - Información semántica asociada

- Un patrón correspondiente para identificar elementos HTML que coincidan con la definición del objeto semántico.
- Conjunto de xpaths donde los elementos suelen estar ubicados (para agilizar el reconocimiento de elementos similares al patrón).

Estas plantillas son guardadas en un servidor online que hemos configurado. Este servidor ofrece 2 funcionalidades principales:

1. La posibilidad de guardar una plantilla para un dominio específico
2. La posibilidad de consultar y recuperar plantillas para un dominio específico.

De esta forma, los usuario voluntarios compartirán las plantillas que ellos generen y usuarios finales podrán descargarlas para utilizarlas mientras navegan por sitios web diferentes.

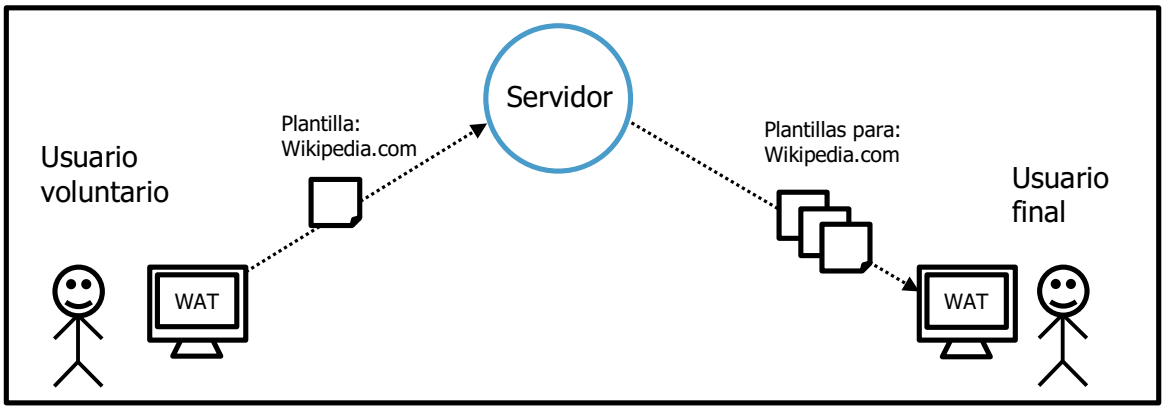


Figura 3.2.6.1: Demostración gráfica de cómo las plantillas son generadas, compartidas y utilizadas

3.2.7 Etapa de "parsing": concluyendo la transcodificación

Cuando un usuario final descarga una plantilla para un sitio web en particular la misma procederá a "instalarse". Esta instalación comprende que WAT recorra todos los objetos semánticos definidos en la plantilla y por cada uno:

a. Buscará elementos HTML de la página que coincidan con el patrón asociado al objeto semántico.
b. Por cada elemento HTML encontrado en el paso anterior se le asociará la información semántica definida en el objeto semántico. Esta

información semántica alterará el código HTML del elemento de la página web. Veamos un ejemplo:

Supongamos que poseemos el siguiente elemento HTML:

*<div **class**="comment"> Esto es un comentario *

Y en la información semántica poseemos los siguientes datos:

Aria-label	"Comentario"
wat-custom	"Un valor personalizado"

Dichos valores son datos que se ingresaron en el formulario durante la creación del objeto semántico.
WAT entonces modificará al elemento HTML para que se vea de la siguiente forma:

*<div **class**="comment"*
***aria-label**= "Comentario"*
***wat-custom** = "Un valor personalizado"*
*> Esto es un comentario *

En este ejemplo vemos como WAT añade 2 atributos al elemento HTML, por un lado añade el atributo "***aria-label***", un atributo del estandar WAI-ARIA para otorgar un nombre semántico al elemento, por otro lado añade otro atributo llamado "***wat-custom***", a modo de ejemplo para demostrar que mediante WAT es posible añadir atributos personalizados. En la sección 3.2.9 veremos para qué nos es útil añadir estos atributos personalizados.

A esta etapa de instalación también se la denomina "parsing" (termino ingles referido al análisis de una cadenas de símbolos) dado que se recorren y analizan todos los elementos HTML en búsqueda objetos semánticos y se les incorporan la información semántica almacenada.

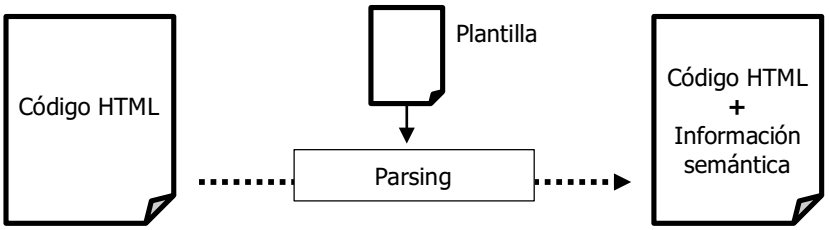


Figura 3.2.7.1: Ejemplo grafico del proceso de parsing.

Este punto comprende la etapa final de la transcodificación, partimos de la información visual que observan los usuarios voluntarios, luego ellos transcriben dicha información semántica en valores de un formulario, finalmente WAT vuelca dichos valores en todos los elementos que coinciden con la definición del patrón de dicho objeto semántico.

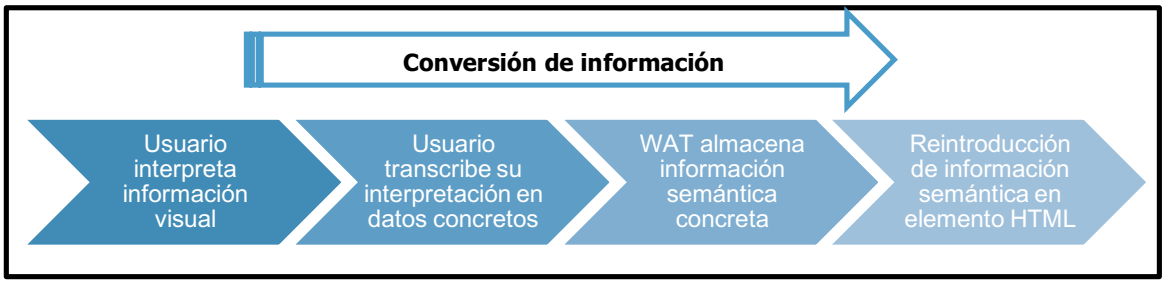


Figura 3.2.7.2: El proceso de transcodificación completo.

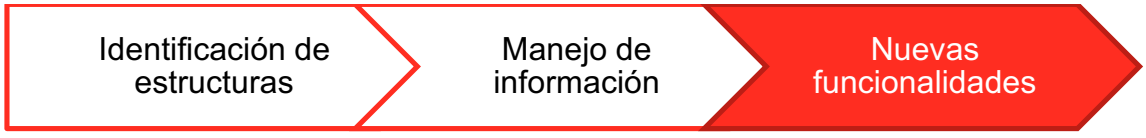
Ahora solo queda aclarar: **¿Cómo es que esta información adicional ofrece una mejora en la accesibilidad web?**

En esta sección vimos como WAT "vuelca" la información semántica en los elementos HTML. Dicha información se traduce a nuevos atributos HTML que son insertados en los elementos que coinciden con los patrones de los objetos semánticos. Esta información adicional puede utilizarse para 2 motivos:

- Para mejorar la interpretación del código HTML por parte de herramientas de accesibilidad ya existentes (como lectores de pantalla)
- Para ser utilizadas por nuevas herramientas desarrolladas para funcionar en conjunto con WAT.

Los atributos a ser añadidos en los elemento HTML estarán definidos en la configuración del "formulario de información semántica" utilizado durante la creación de objetos semánticos mencionado en la sección 3.2.5 y el cuál continuaremos detallando en el capítulo 4.

3.2.8 Mejorando la interpretación de herramientas de accesibilidad



Los atributos HTML que WAT incorpora a los elementos de una página web pueden ayudar a mejorar la funcionalidad de herramientas de accesibilidad como por ejemplo los lectores de pantalla. Por ejemplo, volvamos al mismo código mostrado en la sección anterior:

<div ***class****="comment" > Esto es un comentario *

Si un lector de pantalla debe interpretar dicho código, dado que el mismo no posee ningún tipo de información semántica, el lector de pantalla procederá a leer el texto incluido en el interior del elemento en este caso la frase *"Esto es un comentario".* En este caso el usuario será afortunado dado que dicha frase indica a su vez de que se trata, pero no siempre sería así:

<div ***class****="comment" > Si, por supuesto. *

Si bien el elemento posee una clase llamada "comment" esto no indica ningún parámetro de accesibilidad.

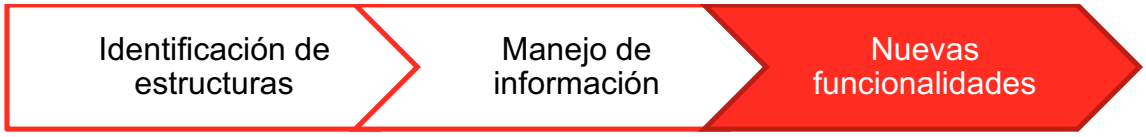
Mediante WAT podemos crear un objeto semántico para dicha estructura, y agregar información semántica como por ejemplo con el atributo "aria-label", posteriormente el código resultado tras la etapa de *parsing* será el siguiente:

<div ***class****="comment"* ***aria-label****= "Comentario"> Si, por supuesto. *

A este punto, se logró una mejora en la semántica del elemento, dado que ahora el lector de pantalla será capaz de leer la etiqueta "*Comentario*" previo a proceder con la lectura del texto interno.

Este es un simple ejemplo de cómo un atributo añadido por WAT puede beneficiar herramientas de accesibilidad, pero bien que existen otros, como por ejemplo añadiendo el atributo WAI-ARIA "role" u otros atributos pertenecientes a estándares de accesibilidad.

3.2.9 Utilizando información adicional para nuevas funcionalidades



Los atributos HTML que WAT incorpora a los elementos de una página web pueden ser utilizados por nuevas herramientas de accesibilidad que requieran de la existencia de dichos atributos personalizados. Por ejemplo, observemos el siguiente código:

```
<div    class="article" >
        <div class="content"> Contenido del articulo </div>

        <div    class="comment" > Comentario 1</div>
        <div    class="comment" > Comentario 2 </div>
</div>
```

Este ejemplo de código hace referencia a un artículo de una página web que a su vez posee comentarios. Ahora supongamos que queremos diseñar una nueva herramienta de accesibilidad capaz de reconocer los artículos junto a sus comentarios para brindar una mejor interfaz al usuario discapacitado. Creamos nuestra herramienta y la configuramos para que busque elementos con las clases "*article*", "*content*" y "*comment*" respectivamente para reconocer los elementos del artículo y así procesar su información. Para el caso del código anterior esto va a funcionar, pero luego queremos probar nuestra herramienta en un nuevo sitio web cuyo código se ve de la siguiente forma:

```
<section>
        <div class="principal"> Contenido del articulo </div>

        <div   class="comentario" > Comentario 1</div>
        <div   class="comentario" > Comentario 2 </div>
</section>
```

Vemos como un objeto que trata de representar lo mimo que en el ejemplo anterior fue escrito de forma diferente. Para que nuestra herramienta funcione con esta nueva estructura deberemos reconfigurarla manualmente, en este caso, en búsqueda de elementos "*section"* en cuyo interior encuentre clases "*principal*" y "*comentario*".

WAT puede ser utilizado para resolver este tipo de problemas, mediante la aplicación de atributos personalizados. Veamos como lucirían ambos códigos tras la aplicación de atributos personalizados:

```
<div   class="article" wat-role = "article" >
        <div class="content"  wat-role="content"> Contenido del articulo </div>

        <div class="comment" wat-role="comment"> Comentario 1</div>
        <div class="comment" wat-role="comment"> Comentario 2 </div>
</div>
```

```
<section wat-role="article">
        <div class="principal" wat-role="content"> Contenido del articulo </div>

        <div   class="comentario" wat-role="comment" > Comentario 1</div>
        <div   class="comentario" wat-role="comment" > Comentario 2 </div>
</section>
```

Mediante WAT podemos crear objetos semánticos y configurar WAT para que inserte información específica en dichos objetos. Podemos configurar nuestra herramienta

para que funcione considerando esta información adicional en vez de utilizar información específica de cada sitio.

Finalmente lo único que debemos hacer es crear plantillas por medio de WAT para cada uno de los sitios. Las plantillas guardarán los patrones que WAT necesita para reconocer a dichos objetos semánticos en cada caso y así introducir la información semántica adicional. Podríamos ver esto como un patrón *decorator*:

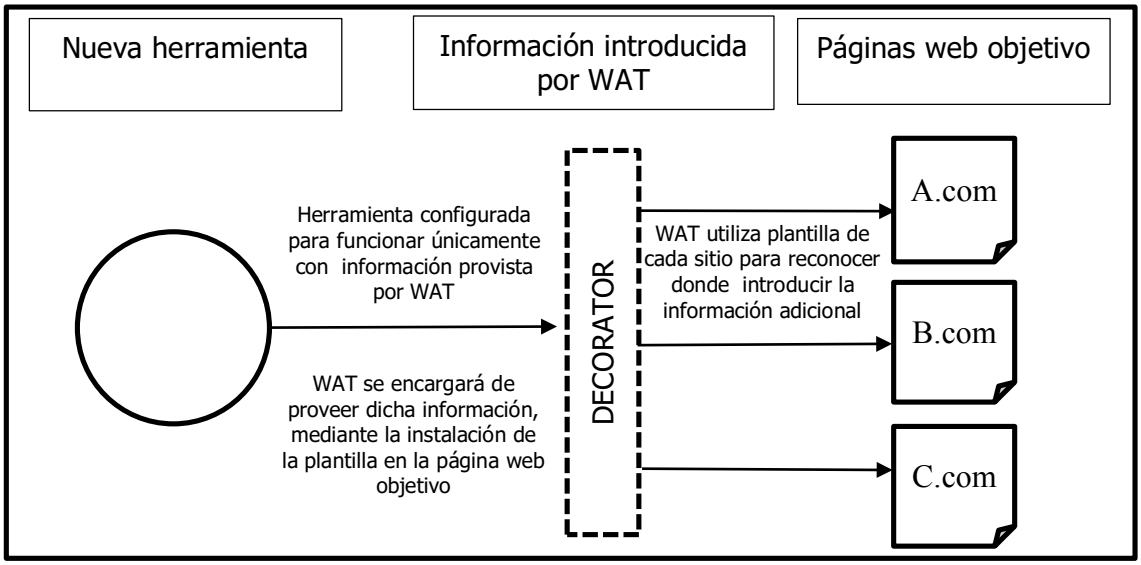
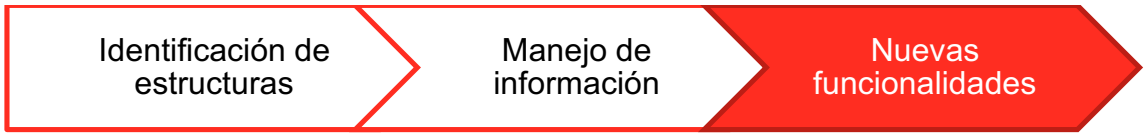


Figura 3.2.9.1: Demostración gráfica de cómo este sistema se asemeja al patrón "decorator". La nueva herramienta solo debe configurarse para funcionar con la información personalizada que WAT incluirá en el sitio.

3.2.10 Navegación web asistida



Retomando las ideas planteadas en las secciones anteriores: hablamos de cómo una mayor granularidad puede facilitar la interpretación de aquellos usuarios que deben utilizar lectores de pantalla, luego hablamos de que dicha granularidad mayor debería a su vez poseer un significado a más alto nivel (evitando así la agrupación sin sentido de elementos que carecen de relación), dicha "agrupación con sentido" se tradujo en lo que definimos como los objetos semánticos. Luego vimos como WAT es capaz de reconocer objetos semánticos similares, relacionar información a ellos y guardarlos en una plantilla. Dicha plantilla puede ser utilizada en el futuro para que usuarios finales puedan reintroducir la información semántica almacenada en las páginas web que navegan, dicha información puede ser utilizada por herramientas como lectores de pantalla o por herramientas más especificas como la que estamos a punto de describir

ahora. Un caso de uso para la información que integra WAT en las páginas web es el de ofrecer a usuarios finales una navegación basada en la interpretación de objetos semánticos (una navegación que surge de la idea de aumentar la granularidad de los elementos que componen la web, al igual que lo hablado en la sección 3.2.1).

Introduciendo a la "NavigationTool"

La **NavigationTool** es un script que desarrollamos y que utiliza la información brindada por WAT para ofrecer una navegación sectorizada de más alto nivel, es decir, evitando que los lectores de pantalla lean elementos individuales sino que primero reconozcan los objetos semánticos definidos como un todo (grano grueso) para ayudar al usuario a comprender el contexto y luego poder recorrer los elementos que componen al objeto semántico de manera más granular. La Figura 3.2.10.1 muestra una representación abstracta sobre como el usuario final accedería a la información al utilizar esta herramienta de navegación. Al ingresar al sitio web el usuario se encuentra posicionado en el "nivel 1" de abstracción pudiendo observar únicamente los objetos semánticos presentes en el "nivel 2". Estos objetos semánticos brindan información de alto nivel al usuario lo cual facilita y agiliza la tarea de "escaneo"; luego el usuario decide cual objeto seleccionar para continuar explorando a un nivel más fino de granularidad. Una vez seleccionado un objeto del "nivel 2", el usuario podrá acceder a los objetos de menor granularidad presentes en el "nivel 3" que se encuentran dentro del objeto semántico seleccionado en el "nivel 2"; este proceso se repite de acuerdo a la cantidad de elementos semánticos que hayan sido configurados, una vez que no hay más objetos semánticos de menos granularidad definidos el usuario podrá navegar dicha sección normalmente (tal como lo hacía previamente), con la diferencia que ahora solo se encuentra navegando una pequeña sección del sitio. En todo momento el usuario puede retroceder a los niveles superiores para optar por diferentes caminos. Se puede ver a este método de navegación como el de navegar por una estructura de tipo árbol, en donde cada nivel de profundidad concuerda con un nivel de menor granularidad.

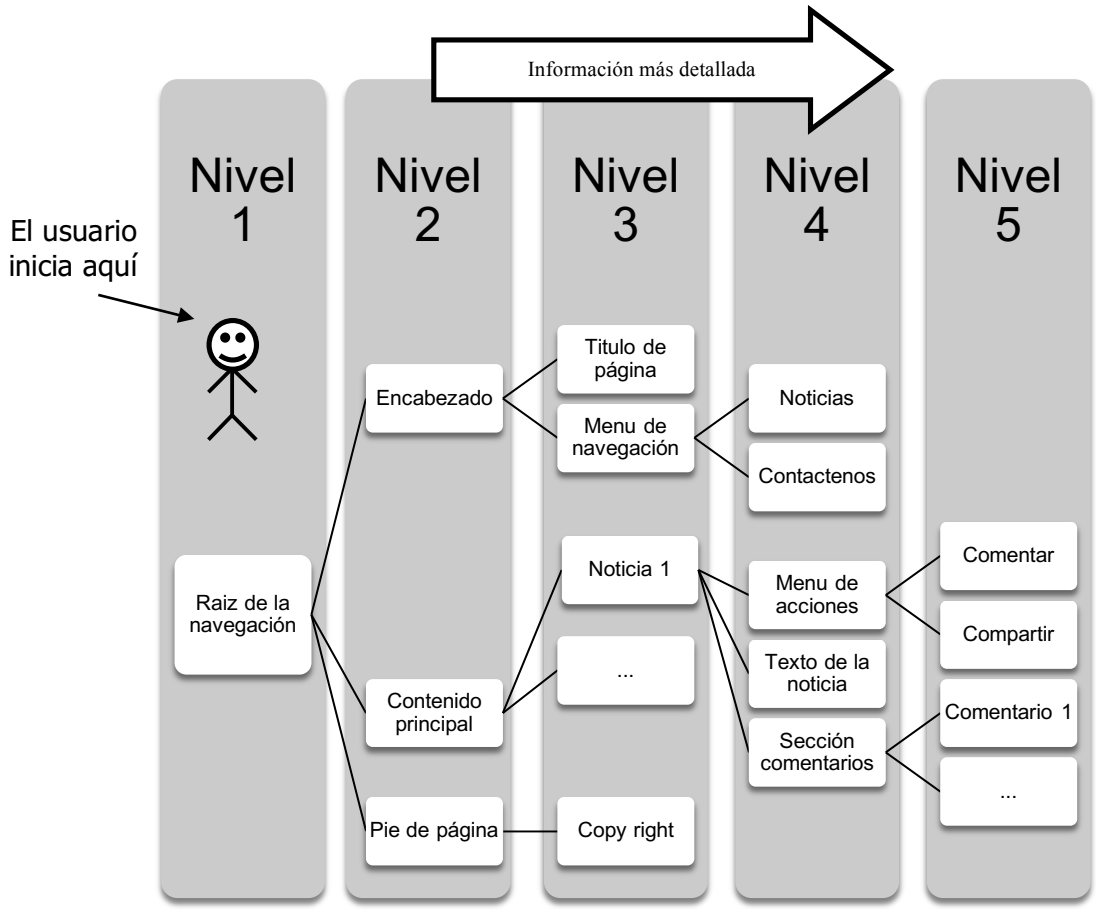


Figura 3.2.10.1: Gráfico representativo de cómo sería la navegación ofrecida por WAT en una página de noticias.

La Figura 3.2.10.2 muestra una captura de pantalla del e-commerce "Mercado libre" que sirve como referencia para apreciar como en la figura 3.2.10.3 los elementos son agrupados (enmarcados en cuadros de colores) por medio de la herramienta WAT, considerando su semántica y delimitando así a los objetos semánticos. Así podemos reconocer cuales son los distintos tipos de objetos presentes en la página y luego crear el flujo de navegación similar al representado en la Figura 3.2.10.1.

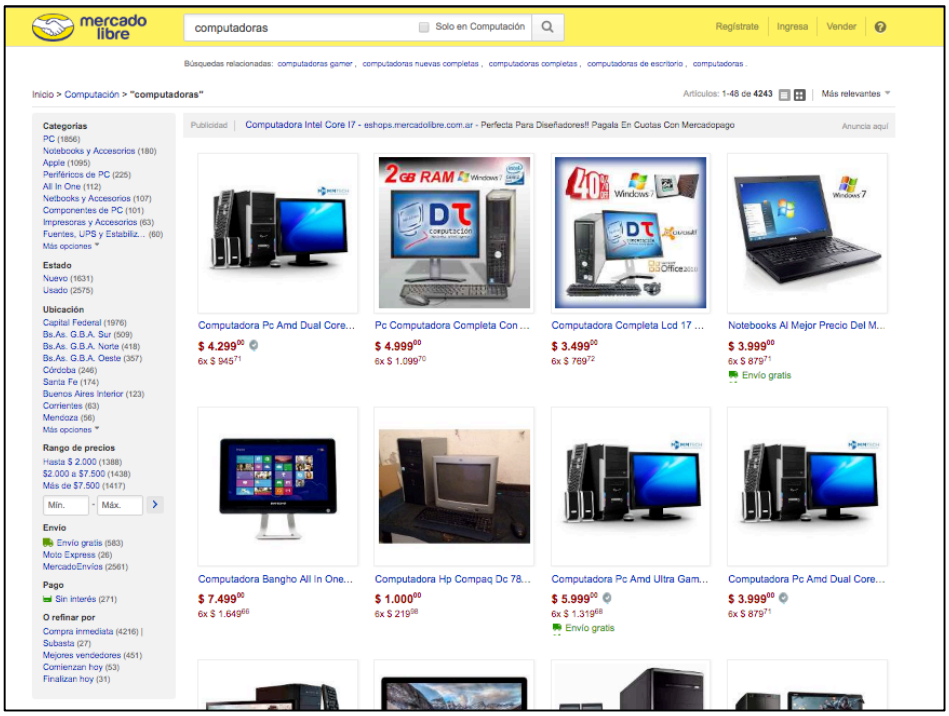


Figura 3.2.10.2: Captura de pantalla de e-commerce "Mercado Libre" que sirve como referencia para la Figura 3.2.10.3

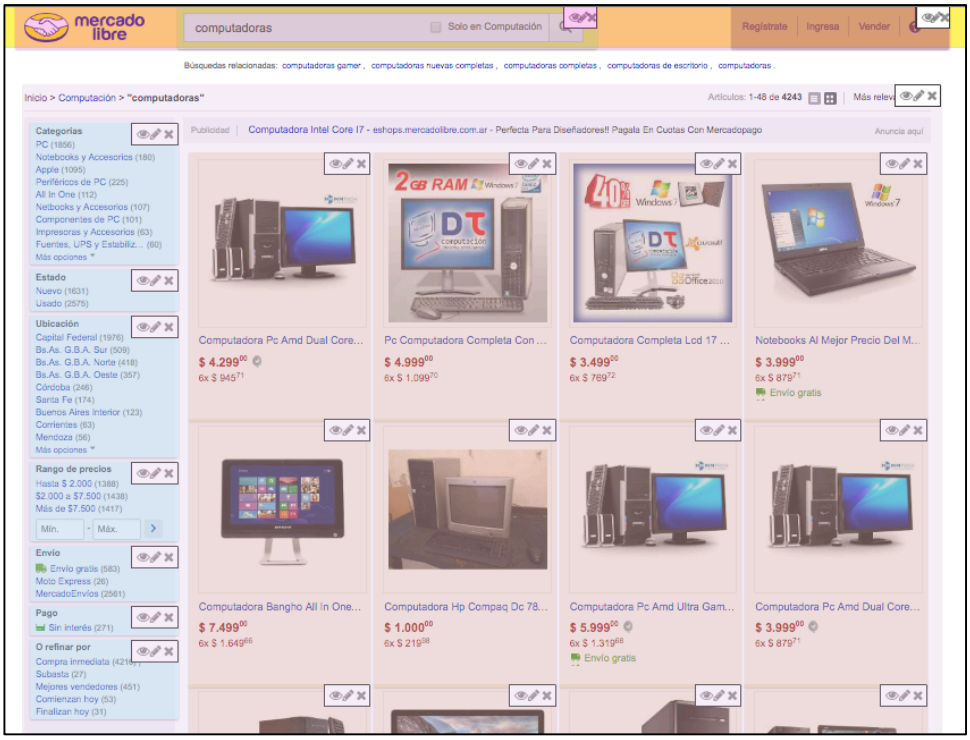


Figura 3.2.10.2: Captura de pantalla de e-commerce "Mercado Libre" al utilizar WAT para reconocer objetos semánticos. Notar que objetos semánticos similares son remarcados con un mismo color y que existen objetos semánticos definidos dentro de otros objetos semánticos.

En la Figura 3.2.10.4 vemos una representación abstracta de cómo sería la navegación del usuario final si consideramos los objetos etiquetados en la captura de pantalla de la Figura 3.2.10.3, los cuadros remarcados en rojo representan a lo que denominaremos como "**E**lementos **n**o **d**efinidos" (**E.N.D.**), es decir aquellos elementos HTML que no son reconocidos como objetos semánticos (dado que no fueron etiquetados como tal), pero que aún así podrán ser accedidos (sin el beneficio del manejo de granularidad ofrecido por la *NavigationTool*). A su vez, un objeto semántico puede estar compuesto o bien por otros objetos semánticos de menor nivel o por nodos E.N.D.

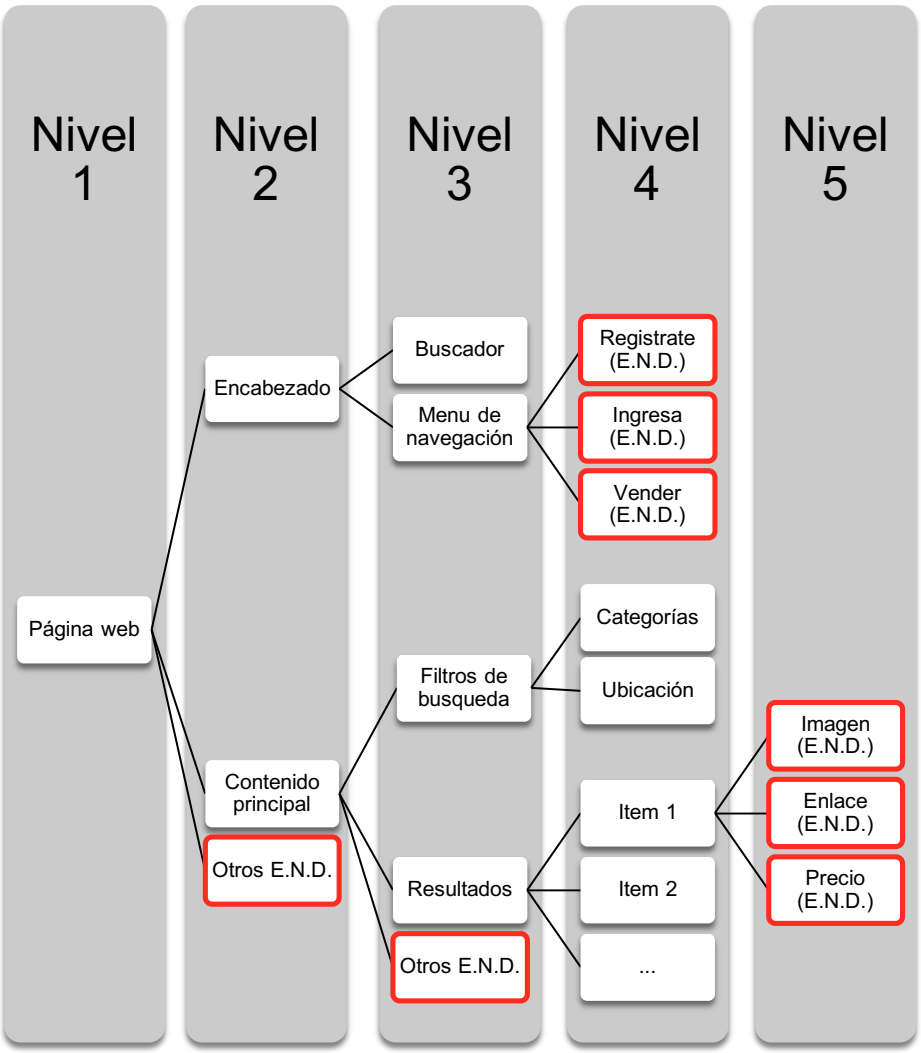


Figura 3.2.10.4: Gráfico representativo de la navegación ofrecida por la NavigationTool utilizando objetos reconocidos en Figura 3.2.10.3. Los cuadros enmarcados en rojo representan elementos no definidos (END), es decir, aquellos elementos que no están definidos como objetos semánticos pero que el usuario es libre de recorrerlos normalmente.

En la sección 3.3 podremos comprender donde es que se ubica esta NavigationTool.

3.3 Arquitectura de WAT

En esta sección nos dispondremos a organizar todos los conceptos detallados en las secciones anteriores en una arquitectura final como la que se muestra en la Figura 3.3.1:

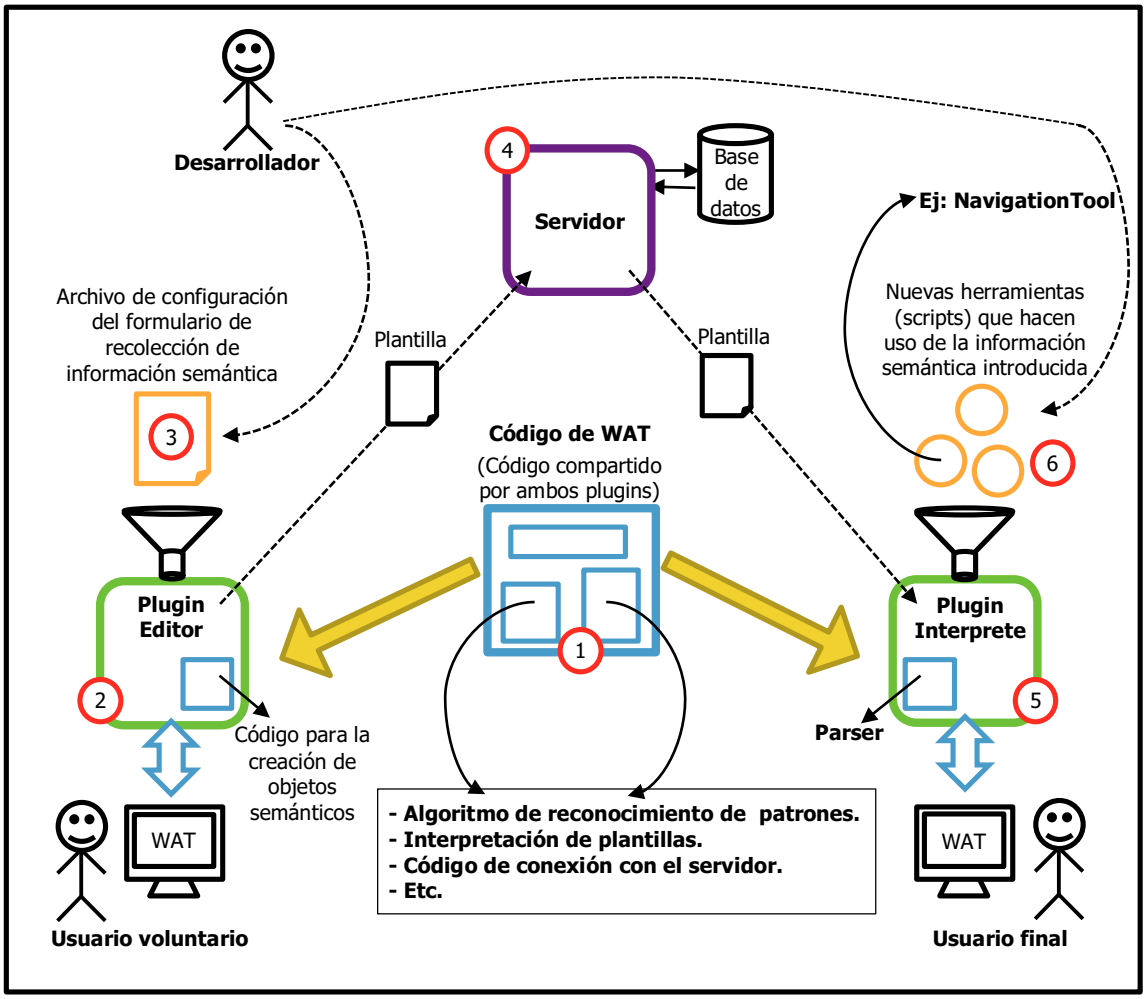


Figura 3.3.1: Arquitectura de WAT

Descripción de sus partes

Notar que hemos etiquetado cada etapa del gráfico con un número del 1 al 6 a continuación explicaremos cada una de ellas:

1. **Código de WAT:** Código principal del framework de WAT. Este código representa la base inicial por sobre donde los plugins son construidos, es una librería de código compartido entre todos los plugins de WAT. Aquí se aloja código que es reutilizado a lo largo de toda la aplicación como por ejemplo el

algoritmo de reconocimiento de patrones, la creación y lectura de plantillas, una API para comunicarse con el servidor, etc.

2. **Plugin editor**: Plugin basado en la librería WAT cuyo objetivo es brindar una herramienta al usuario voluntario para que el mismo pueda identificar objetos semánticos y "etiquetarlos" añadiendo la información semántica solicitada. La información semántica a solicitar se encuentra configurada en el "archivo de configuración del formulario". Una vez finalizado el reconocimiento de objetos semánticos el editor provee la funcionalidad para subir la plantilla a nuestro servidor.

3. **Archivo de configuración del formulario**: Comprende una de las entradas del framework, donde un desarrollador debe/puede configurar la información a solicitar durante la creación de objetos semánticos. Dicha configuración permite añadir atributos personalizados a los objetos semánticos durante la etapa de "parsing".

4. **Servidor**: Es una aplicación básica que provee un CRUD para plantillas y usuarios de WAT. Dispone de una API REST para que los plugins puedan comunicarse con él y una aplicación web para que usuarios puedan acceder a él y gestionar sus datos y plantillas. La aplicación Web provee 2 tipos de interfaces, una para usuarios voluntarios donde pueden ver las plantillas que ellos mismos crearon y otra para usuarios finales donde pueden buscar plantillas para sitios específicos y/o solicitar nuevas plantillas. A su vez los usuarios finales pueden seleccionar cual plantilla utilizar en caso de que existan más de una plantilla para un mismo sitio. (No se hará demasiado hincapié en esta parte, dado que la participación del servidor dentro de la aplicación es simplemente para la gestión de datos y no representa ningún tipo de complejidad más que la de un simple CRUD).

5. **Plugin interprete:** También llamado plugin "player", es un segundo plugin basado en la librería WAT cuyo objetivo es brindar una herramienta a los usuarios finales para que puedan "instalar" plantillas almacenadas en nuestro servidor dentro de la página web objetivo. Para lograr esto dispone de un "parser", código encargado de detectar objetos semánticos definidos en la plantilla e insertar la información semántica correspondiente. El parser se ejecutará cada vez que ocurran cambios en la página web a modo de poder detectar nuevos objetos semánticos que puedan aparecer, a este ciclo lo denominamos "refresh". El plugin interprete a su vez sirve de entrada para definir un conjunto de herramientas adicionales, estas hacen uso de la información semántica adicional que el parser provee. Cada vez que un nuevo ciclo de "refresh" sucede, el plugin interprete alertará a cada herramienta definida sobre los cambios ocurridos (en caso de reconocer nuevos objetos semánticos). Por último, el plugin interprete permite al usuario final habilitar y deshabilitar cada una de las herramientas adicionales (en caso de que no las desee).

6. **Herramientas que hacen uso de la información semántica introducida:** Comprende otra de la entradas del framework, donde los desarrolladores pueden insertar código (scripts) que hagan uso de la información semántica introducida por WAT para ofrecer nuevas funcionalidades de accesibilidad a los usuarios finales. Cada herramienta se conecta al plugin interprete mediante un interfaz de 3 funciones (**start**, **close** y **refresh**), el plugin interprete se encargará de iniciar (start) o cerrar (close) dicha herramienta de acuerdo a si el usuario final la habilita o deshabilita, así como de ejecutar la función "refresh" cada vez que ocurre un cambio en la información semántica provista por WAT, a modo de alertar a la herramienta sobre los cambios ocurridos. Luego cada herramienta se encargará de proveer su funcionalidad de manera independiente. Un ejemplo de estas herramientas es la del ***NavigationTool***.

Tipos de usuario

En esta arquitectura podemos diferenciar 3 tipos de usuarios:

1. **Usuarios voluntarios**: Son usuarios sin discapacidad visual quienes crean y aportan plantillas generadas por medio del plugin editor brindado por WAT.

2. **Usuarios finales**: Son los usuarios con discapacidad visual, quienes pueden hacer uso del plugin interprete brindado por WAT y a su vez de las herramientas adicionales que este provee (las cuales pueden ser habilitadas o deshabilitadas).

3. **Desarrolladores**: Son usuarios técnicos, los cuales pueden hacer uso del framework que provee WAT para mejorar o expandir sus funcionalidades. Esto se logra ya sea editando alguna de las entradas del framework (archivo de configuración del formulario o creando nuevas herramientas asociadas al plugin interprete) o editando el código fuente de los plugins y librería de WAT.

En el capítulo 4 se mostrará un manual de usuario para cada uno de los plugins editor e interprete. Mientras que el capítulo 5 estará dedicado a los desarrolladores, un capítulo técnico con todo lo que necesitan saber sobre como se organiza el código de WAT para continuar con su desarrollo.

3.4 Concluyendo

Ahora que hemos explicado los conceptos y funcionalidades que integran a WAT y como estos se organizan, retomaremos los problemas detectados en la sección 3.2.1 sobre el uso de los lectores de pantalla para comprender como es que buscamos mitigarlos:

1. **Dificultad para la interpretación del contenido:** Esta problemática se verá reducida gracias al aumento de granularidad en el recorrido de los elementos web. La interpretación del usuario debería verse beneficiada gracias a la existencia de objetos de más alto nivel que ayudan a reducir la cantidad de elementos por interpretar del sitio web.

2. **Diferentes experiencias de usuario por sitio:** Las estructuras de distintos sitios web son distintas, sin embargo, durante la definición de objetos semánticos, estamos "etiquetando" estas estructuras distintas con información semántica similar. La información provista en el formulario de información semántica permite "estandarizar" cierta cantidad de objetos semánticos por reconocer de cada sitio distinto. A su vez, gracias a la introducción de nuevas herramientas como la "NavigationTool", podemos crear funcionalidades que ayuden a estandarizar la navegación de distintas páginas web bajo un mismo conjunto de elementos. Por ejemplo: Podríamos incluir una herramienta con la cual, cada vez que presionemos "Ctrl+B" nos posiciones el puntero dentro del cuadro de búsqueda (si es que hay uno presente), de misma forma, utilizar otros comandos para posicionarnos en otras secciones del sitio. Esta herramienta puede reconocer fácilmente estas secciones gracias a la información semántica de cada plantilla, que de cierto modo "estandariza" las componentes de distintos sitios para que se reconozcan de igual forma. Es así, como diferentes herramientas pueden utilizar a WAT para buscar otorgar una experiencia de usuario más similar en todos los sitios web que posean plantillas creadas.

3. **Fallas en la navegación (ciclos, o elementos inaccesibles):** Muchos lectores de pantalla poseen errores a la hora de navegar distintos sitios web, dado que cada uno posee estructuras distintas, o no poseen la correcta información de accesibilidad. Gracias a la estandarización de la información y a herramientas como la "NavigationTool" este problema puede ser fácilmente mitigado.

4. **Navegación poco intuitiva:** Nuevamente, herramientas como la "NavigationTool" pueden ofrecer mecanismos de asistencia que ayuden a comprender mejor la navegación. Un ejemplo, es el uso de una herramienta adicional que reproduce distintos sonidos durante la navegación. De esta forma el usuario con discapacidad visual puede saber cuando ha alcanzado el final del documento, o cuando se esta desplazando por distintos objetos semánticos no

solo por la lectura, sino también por la reproducción de un sonido particular para cada escenario.

5. **Navegación muy extensa:** Al igual que 1*, la existencia de objetos semánticos de mayor nivel, reduce la cantidad de elementos por recorrer. Gracias a esto, recorrido general del sitio debería verse reducido.

6. **Contenidos de lectura largos:** Gracias a la implementación de nombres "labels" en cada objeto semántico, los lectores de pantalla pueden utilizar este texto alternativo para dar información al usuario sobre el contenido que se encuentra dentro del objeto semántico, sin la necesidad de recurrir a la total lectura del mismo.

7. **Falta de interacción detallada:** La posibilidad de expandir objetos semánticos para navegar a objetos semánticos de menor nivel (o elementos simples E.N.D.), habilita a que el usuario pueda obtener una navegación más detallada tras haber navegado los objetos de mayor nivel. El hecho de haber recorrido los objetos de mayor nivel para acceder a los de menor nivel, ayuda a que una vez accedidos a los elementos de menor nivel, el usuario posea conciencia de donde se encuentra ubicado el puntero del lector de pantalla.

Capitulo IV
Uso de las herramientas provistas por WAT

Este capítulo sirve a modo de manual de usuario para el uso de todas las herramientas brindadas por WAT, procederemos a dividirlo en 3 partes:

1. Plugin Editor
2. Plugin Interprete
3. Aplicación web (Servidor)

4.1 Plugin Editor

El Plugin Editor (de ahora en más lo llamaremos simplemente Editor) es la herramienta de WAT que permite la generación de plantillas para cada sitio web visitado. El Editor debe ser utilizado por usuarios sin discapacidades visuales que llamaremos "usuarios voluntarios" quienes, una vez en ingresado a un sitio web, podrán hacer uso de las funcionalidades provistas por el Editor para reconocer y etiquetar nuevos objetos semánticos en dicho sitio.

Una vez instalada la extensión del explorador Chrome, un nuevo ícono aparecerá en la barra superior derecha.

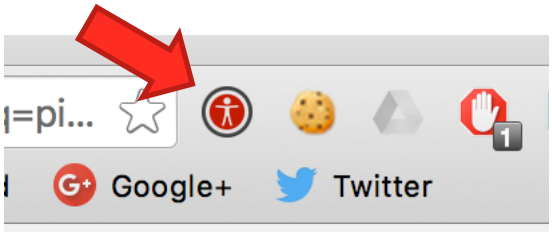


Figura 4.1.1: Captura de pantalla del explorador Chrome, donde se visualiza el nuevo ícono que aparece al instalar el Plugin Editor.

Al presionarlo veremos un *popup* con la opción de habilitar/deshabilitar el Editor.

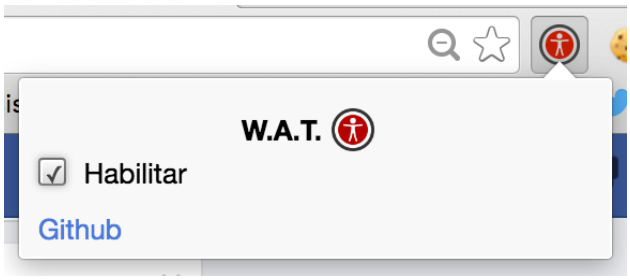


Figura 4.1.2: Captura de pantalla del explorador Chrome, donde se visualiza el popup que emerge cuando el ícono del Plugin Editor es presionado.

Mientras el Editor se encuentre habilitado una barra de herramientas aparecerá en el lado derecho del explorador.

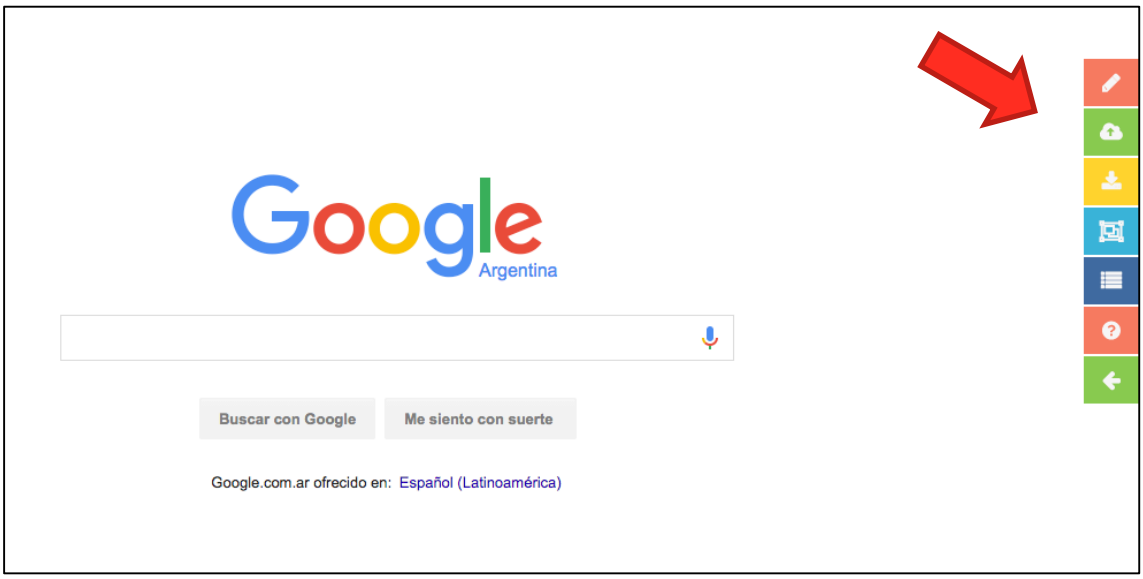


Figura 4.1.3: Captura de pantalla del sitio Google.com, luego de habilitar el Editor. Una barra de herramientas aparecerá en el costado derecho del sitio web.

Esta barra de herramientas posee distintos "estados", es decir, mostrará diferentes funcionalidades de acuerdo a cual sea su estado actual. La primera vez que utilicemos la barra de herramientas en una página web, veremos únicamente las siguientes opciones para ser utilizadas:

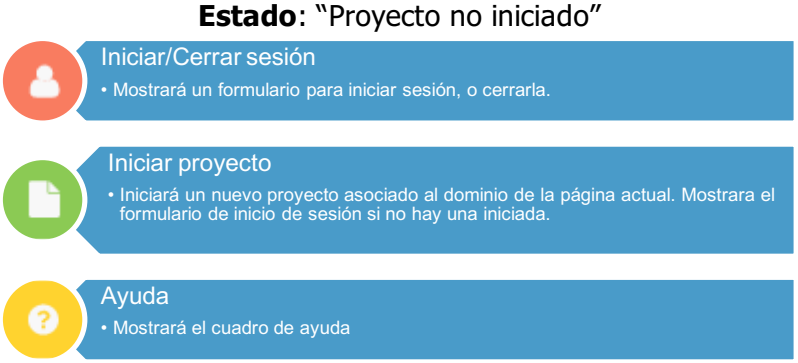


Figura 4.1.4: Lista de opciones de la barra de herramientas cuando el usuario aún no ha iniciado un proyecto en el Editor.

4.1.1 Ayuda

La opción de ayuda se mostrará en todos los "estados" de la barra de herramientas. Cuando el usuario la seleccione, se mostrará un dialogo con la descripción de cada una de las opciones que se encuentran disponibles en el estado actual.

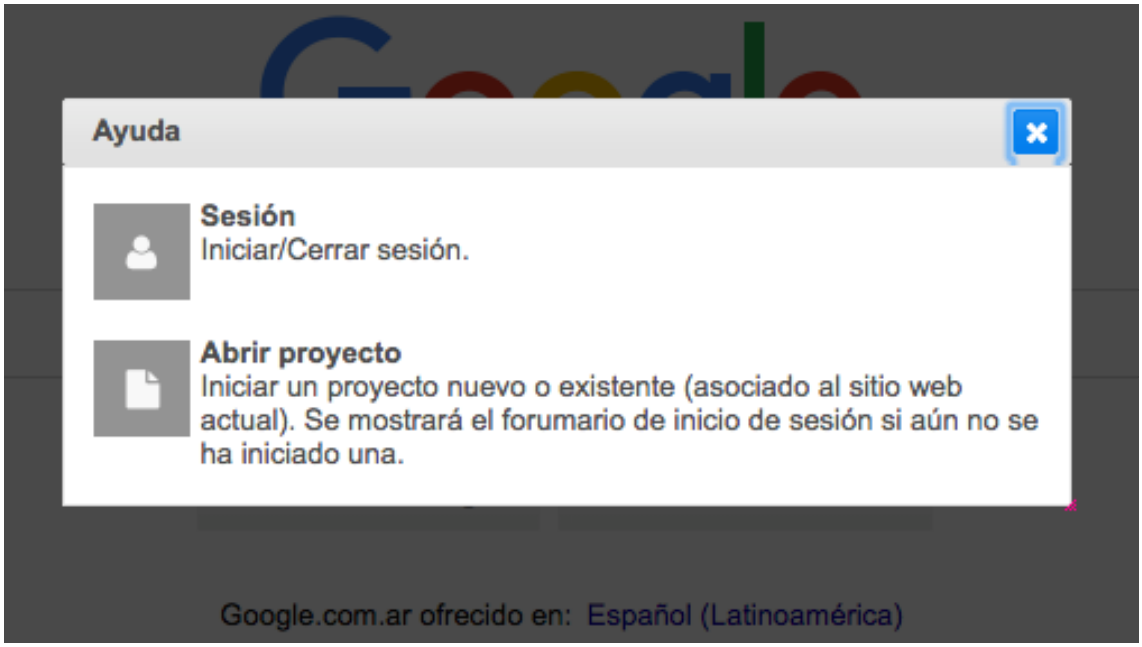


Figura 4.1.1.1: Ejemplo de dialogo que aparece al seleccionar la opción "Ayuda" en el estado "Proyecto no iniciado", dicho dialogo muestra una descripción sobre cada una de las opciones disponibles en dicho estado.

4.1.2 Iniciar sesión

Al seleccionar esta opción, si el usuario no posee una sesión iniciada se mostrará el diálogo de inicio de sesión como el que se muestra en la Figura 4.1.2.1, por otro lado, si el usuario ya poseía una sesión iniciada, se mostrará el diálogo de cierre de sesión como el que se muestra en la Figura 4.1.2.2.

Figura 4.1.2.1: Captura de pantalla del diálogo de inicio de sesión que aparecerá cuando el usuario presione el botón de inicio de sesión y la sesión aún no se encuentre iniciada.

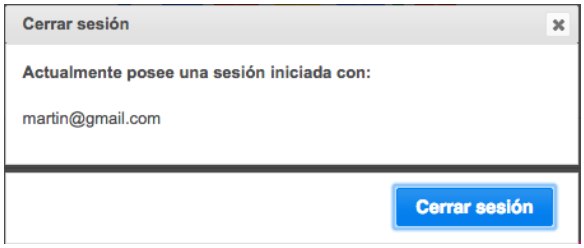


Figura 4.1.2.2: Captura de pantalla del diálogo de cierre de sesión que aparecerá cuando el usuario presione el botón de inicio de sesión y la sesión ya se encuentre iniciada.

4.1.3 Iniciar proyecto

Cuando aún no hayamos iniciado ningún proyecto para el sitio en cuestión, la barra de herramientas mostrará esta opción para iniciar uno. Al seleccionar "Iniciar proyecto" el Editor nos mostraría una serie de formularios:

1. Si aún no poseemos una sesión iniciada, mostrará el mismo dialogo que el de la Figura 4.1.2.1.

2. Nos preguntará si deseamos iniciar un nuevo o volver a abrir alguno existente (siempre que poseamos proyectos antiguos relacionados al mismo dominio).

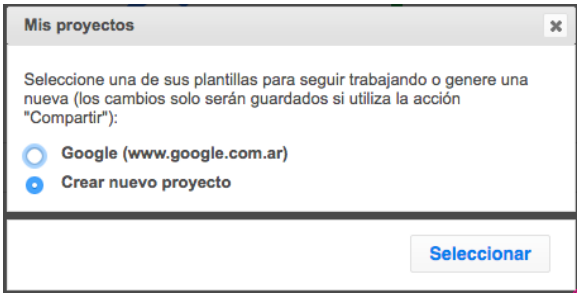


Figura 4.1.3.2: Captura de pantalla del dialogo de selección de proyecto que aparecerá luego de seleccionar la opción de "Iniciar Proyecto". Entre las opciones se puede optar por un proyecto anterior con el mismo dominio, o iniciar uno nuevo.

3. Si seleccionamos un nuevo proyecto, nos mostrará un formulario para ingresar la información básica del proyecto.

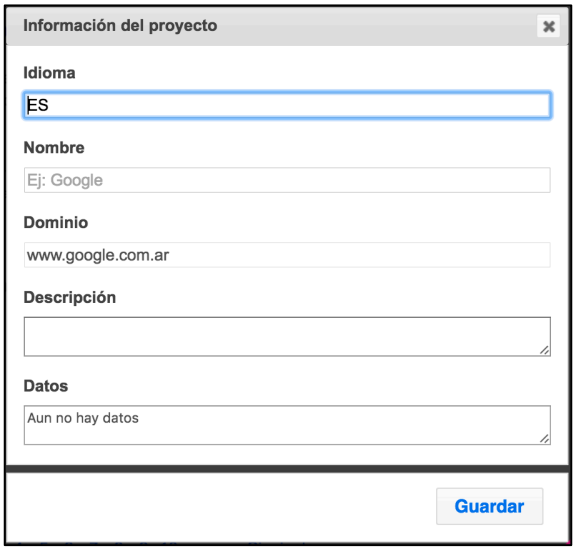


Figura 4.1.3.3: Captura de pantalla del dialogo de información básica del proyecto. El mismo aparecerá si el usuario opta por la creación de un nuevo proyecto.

Cuando hayamos finalizado la inicialización del proyecto (o si ya poseíamos un proyecto iniciado para el sitio web actual), la barra de herramientas mostrará nuevas opciones para continuar trabajando con el proyecto iniciado:

Figura 4.1.3.1: Lista de opciones de la barra de herramientas una vez que el usuario ha iniciado un proyecto en el Editor.

A continuación repasaremos cada una de estas opciones.

4.1.4 Editar proyecto

Al seleccionar esta opción, el Editor nos mostrará nuevamente el formulario de edición de información básica del proyecto (el mismo que se muestra al iniciar un nuevo proyecto)

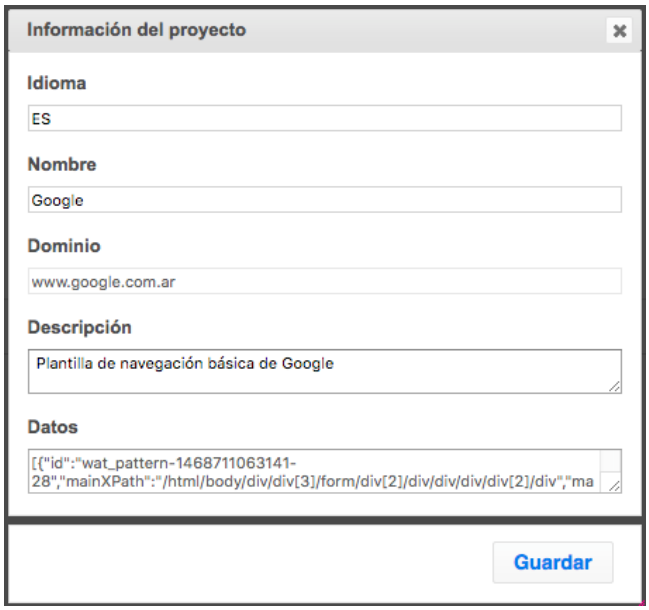


Figura 4.1.4.1: Captura de pantalla del dialogo de información básica del proyecto. El mismo aparecerá si el usuario selecciona la opción de "Editar proyecto".

4.1.5 Compartir proyecto

Se mostrará un dialogo de confirmación para guardar el proyecto en nuestro servidores (de no hacerlo, el proyecto se mantendrá guardado localmente en el explorador, pudiendo perderse si el usuario decide borrar los datos de aplicaciones almacenadas en el explorador). Al presionar "Guardar", el Editor persistirá el proyecto en nuestro servidor, permitiendo que otros usuarios puedan utilizarlo. (Puede suceder que el Editor pida al usuario nuevamente iniciar sesión antes de realizar esta acción, en caso que la sesión actual haya caducado).

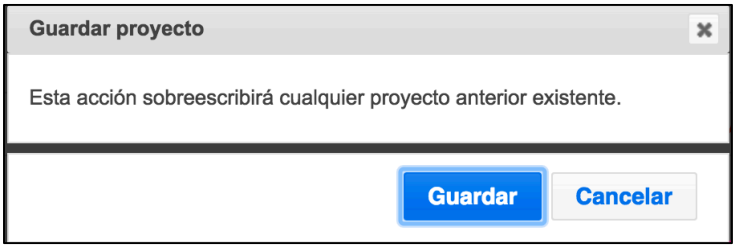


Figura 4.1.5.1: Captura de pantalla del dialogo que aparecerá cuando el usuario seleccione la opción de "Compartir proyecto". Se pide la confirmación al usuario para sobrescribir el proyecto si es que existía una versión anterior del mismo.

4.1.6 Descargar proyecto

Permite descargar el proyecto en un archivo de texto

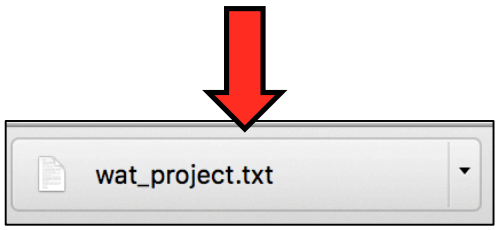


Figura 4.1.6.1: Captura de pantalla de la descarga del explorador Chrome que muestra como un archivo de texto es descargado. El mismo contiene la información del proyecto en cuestión.

4.1.7 Crear nuevo objeto semántico

Esta es la principal opción del Editor, la más importante, dado que nos permite definir nuevos objetos semánticos de un sitio web. Una vez seleccionada, la barra de herramientas pasará a un nuevo estado y nos mostrará las siguientes opciones:

Estado: "Creando nuevo objeto semántico"

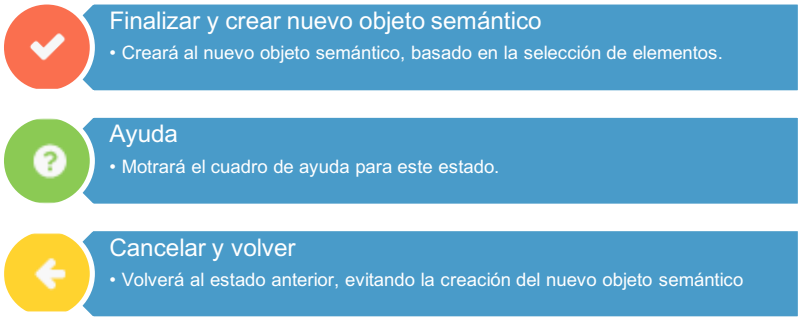


Figura 4.1.7.1: Lista de opciones de la barra de herramientas luego que el usuario selecciona la opción de "Crear nuevo objeto semántico".

En paralelo al cambio de estado de la barra de herramientas, la funcionalidad del mouse dentro del sitio web que navegamos quedará "inhabilitada" momentáneamente, permitiendo al usuario utilizar el mouse para la función de elegir aquellos elementos que pasarán a formar parte del nuevo objeto semántico, en vez de interactuar con el sitio web. Observar que en la Figura 4.1.6.2, cuando el puntero se posiciona sobre un elemento de la página web Google.com, dicho elemento será remarcado por un borde celeste, indicando al usuario que el mismo esta siendo seleccionado. Un click sobre el elemento seleccionado lo seleccionará definitivamente para formar parte del objeto semántico. Notar que la herramienta evita que el click se propague y ejecute alguna funcionalidad del sitio web.

Nota: Para las siguientes figuras, el puntero rodeado por un circulo rojo de contorno punteado indica que el puntero esta posicionado sobre un elemento pero no se ha realizado la acción de click, a su vez, el puntero rodeado por un circulo verde de contorno sólido, indica que el mouse realiza la acción de click sobre el elemento.

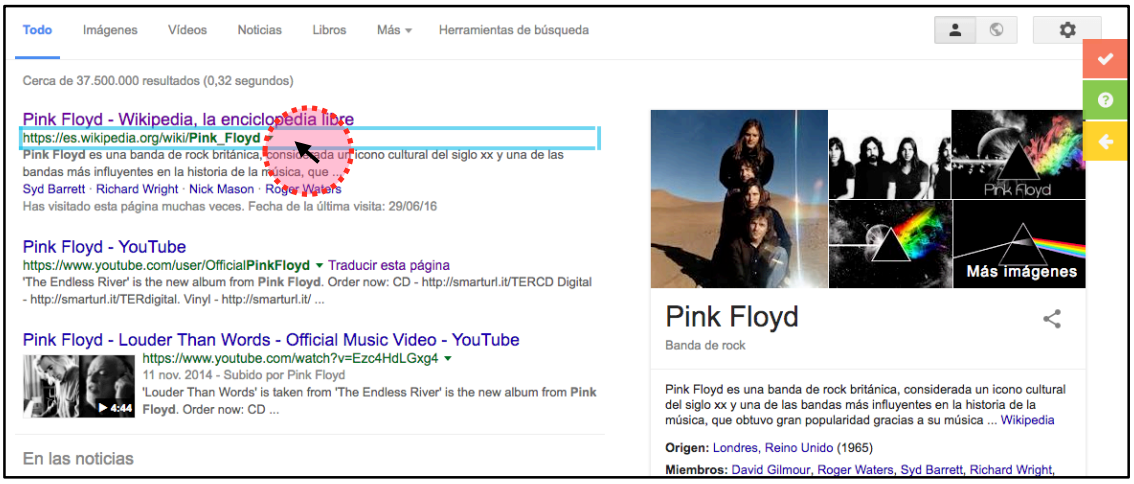


Figura 4.1.7.2: Captura de pantalla del sitio web Google.com, donde se muestra como la funcionalidad del puntero es alterada para permitir al usuario navegar los elementos del sitio mientras la acción de "Crear nuevo objeto semántico" esta activada. A su vez, notar que la barra de herramientas lateral del Editor, solo muestra las opciones descritas por la Figura 4.1.7.1.

Ahora veamos la diferencia de la Figura 4.1.7.2 con la Figura 4.1.7.3, cuando desplazamos el mouse hacia el elemento superior, este último pasa a tener el borde celeste, indicando que ahora estamos seleccionado momentáneamente dicho elemento.

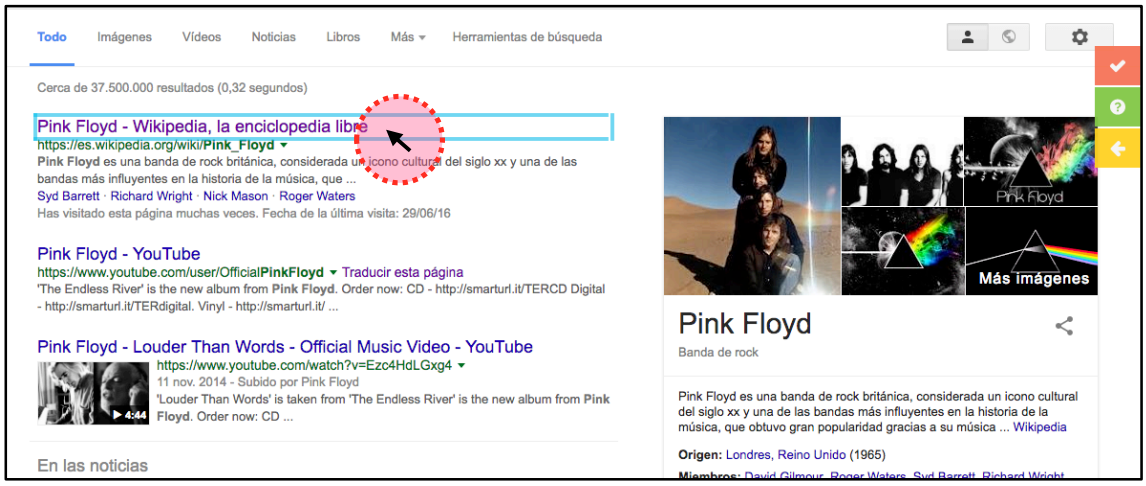


Figura 4.1.7.3: Captura de pantalla del sitio web Google.com, donde se muestra como al desplazar el puntero sobre otro elemento distinto al de la Figura 4.1.7.2, el usuario pasa a "enfocar" un nuevo elemento.

Con esta información, el usuario es capaz de comprender correctamente cual es el elemento seleccionado en cada momento. Posteriormente, el usuario puede realizar un click sobre el elemento tal como muestra la Figura 4.1.7.4.

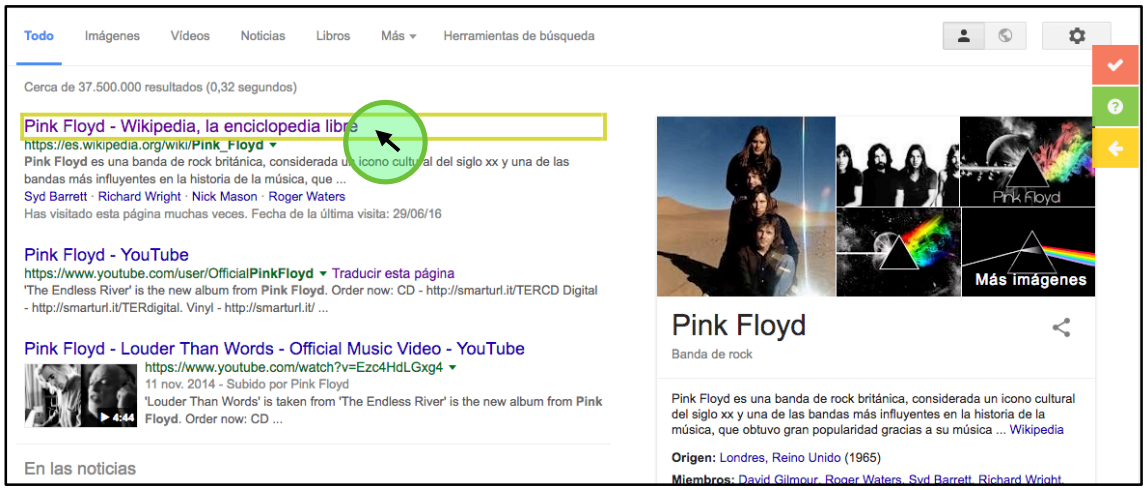


Figura 4.1.7.4: Captura de pantalla del sitio web Google.com, donde se muestra el puntero realizando un click sobre el elemento seleccionado en dicho momento. El selector del elemento se tornará amarillo.

Una vez realizado el click, el contorno del elemento se tornará amarillo, indicando que dicho elemento pasará a formar parte de la estructura que se utilizara para la creación del nuevo objeto semántico. La figura 4.1.7.5, muestra como el elemento donde se realizó un click en la Figura 4.1.7.4, ahora posee un contorno amarillo, pero al mismo tiempo el usuario puede continuar recorriendo otros objetos.

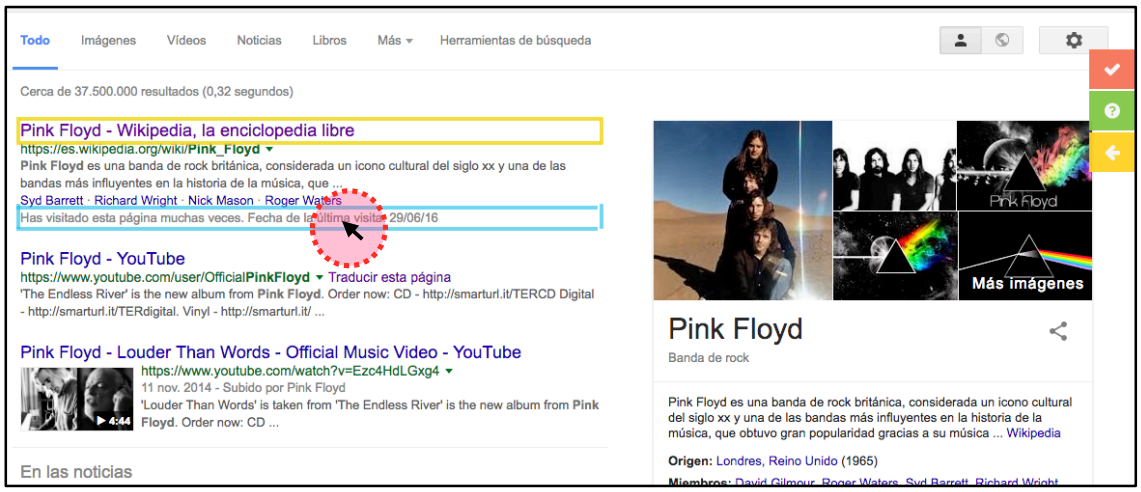


Figura 4.1.7.5: Captura de pantalla del sitio web Google.com, donde se muestra como el elemento de la Figura 4.1.7.4 ahora posee un contorno amarillo, mientras que el usuario puede continuar enfocando otros elementos del sitio.

Ahora veamos que pasa si el usuario selecciona un segundo elemento. La Figura 4.1.7.6 muestra como se realiza un click sobre un segundo elemento.

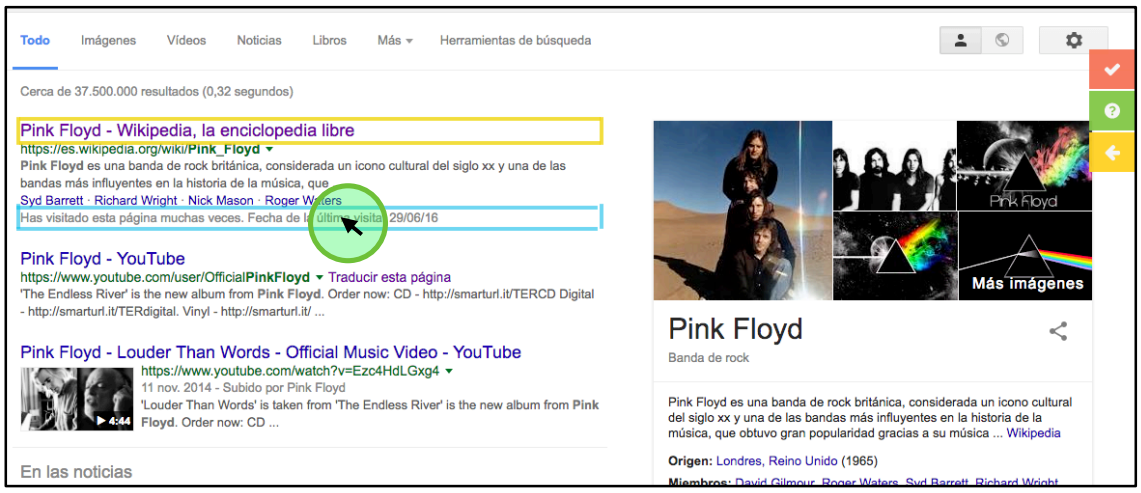


Figura 4.1.7.6: Captura de pantalla del sitio web Google.com, donde se muestra un segundo click sobre otro elemento diferente al de la figura 4.1.7.4.

Una vez seleccionado un nuevo elemento, podemos apreciar en la Figura 4.1.7.7 como cambia el recuadro amarillo, ahora pasará a encuadrar a un elemento padre, indicando que la estructura del nuevo objeto semántico es ahora aquel elemento que posee en su interior a los dos elementos seleccionados anteriormente, o mejor dicho su ancestro más cercano en común (Lowest common ancestor).

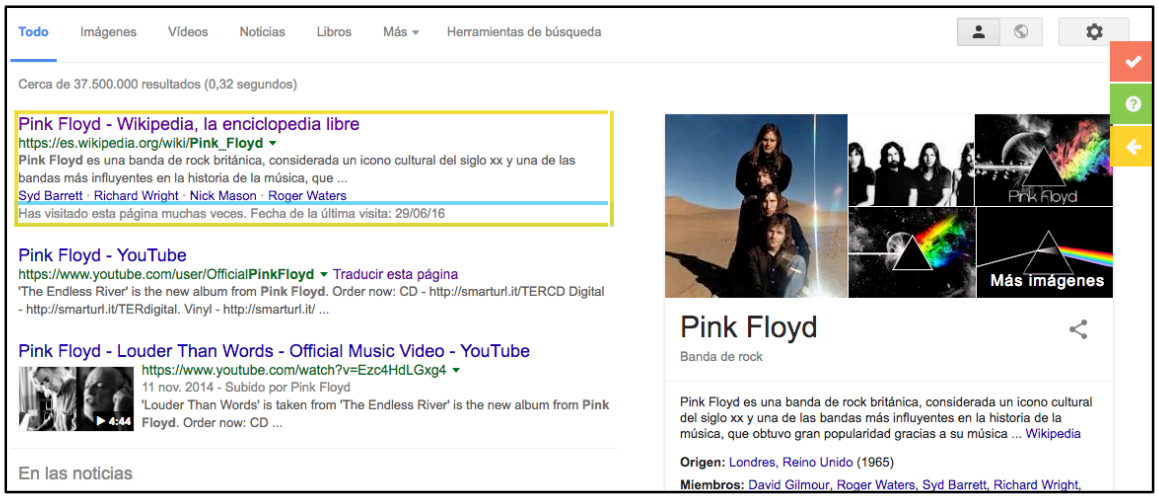


Figura 4.1.7.7: Captura de pantalla del sitio web Google.com, donde se muestra como luego de haber seleccionado un segundo elemento, el cuadro amarillo pasa a seleccionar al ancestro en común más cercano (lowest common ancestor), indicando ahora que el nuevo objeto semántico posee como estructura al padre en común de ambos elementos.

De forma similar, el usuario podrá repetir el proceso anterior cuantas veces quiera, seleccionando tantos elementos como desee. Por cada nuevo elemento seleccionado,

el recuadro amarillo comprenderá siempre al padre más cercano que contenga a todos los elementos seleccionados de esta forma. Una vez finalizada dicha selección, el usuario podrá entonces seleccionar la opción de "Finalizar y crear nuevo objeto semántico" desde la barra de herramientas, tal como muestra la Figura 4.1.7.8.

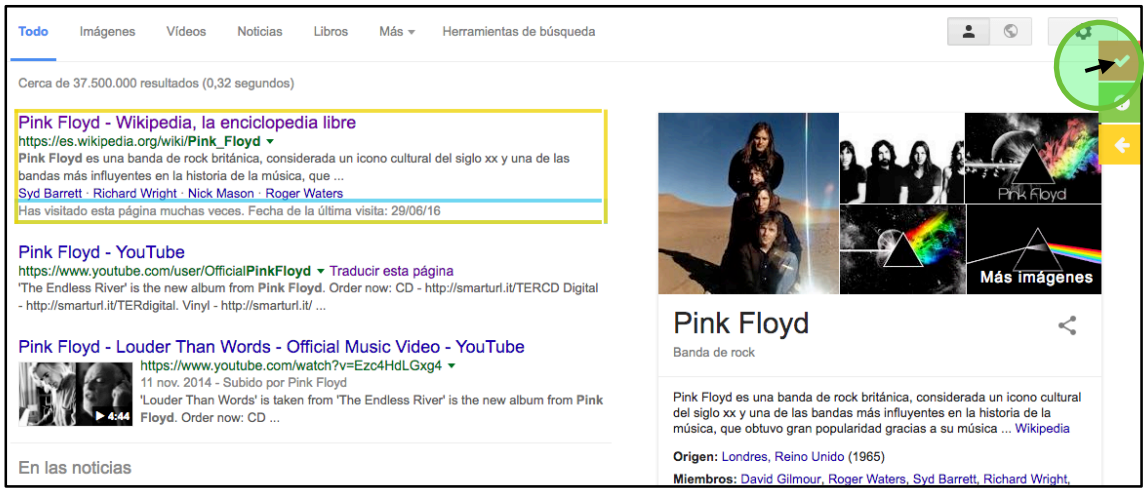


Figura 4.1.7.8: Captura de pantalla del sitio web Google.com, donde se muestra como el usuario realiza un click sobre la barra de herramientas del editor para seleccionar la opción de "finalizar y crear nuevo objeto semántico".

En este punto, el usuario también puede optar por seleccionar la opción de "Cancelar y volver" (no explicaremos el botón de ayuda, dado que ya fue explicado antes), veamos que sucederá en cada caso:

4.1.7.1 Finalizar y crear nuevo objeto semántico

De optar por esta opción, el Editor creará un nuevo objeto semántico que poseerá como estructura principal aquel elemento que haya quedado seleccionado por el recuadro amarillo. Luego se le mostrará el "formulario de información semántica" para que el usuario agregue la información semántica. La misma dependerá de la configuración de dicho formulario, nosotros hemos optado por la siguiente configuración que se visualiza en la Figura 4.1.7.1.1.

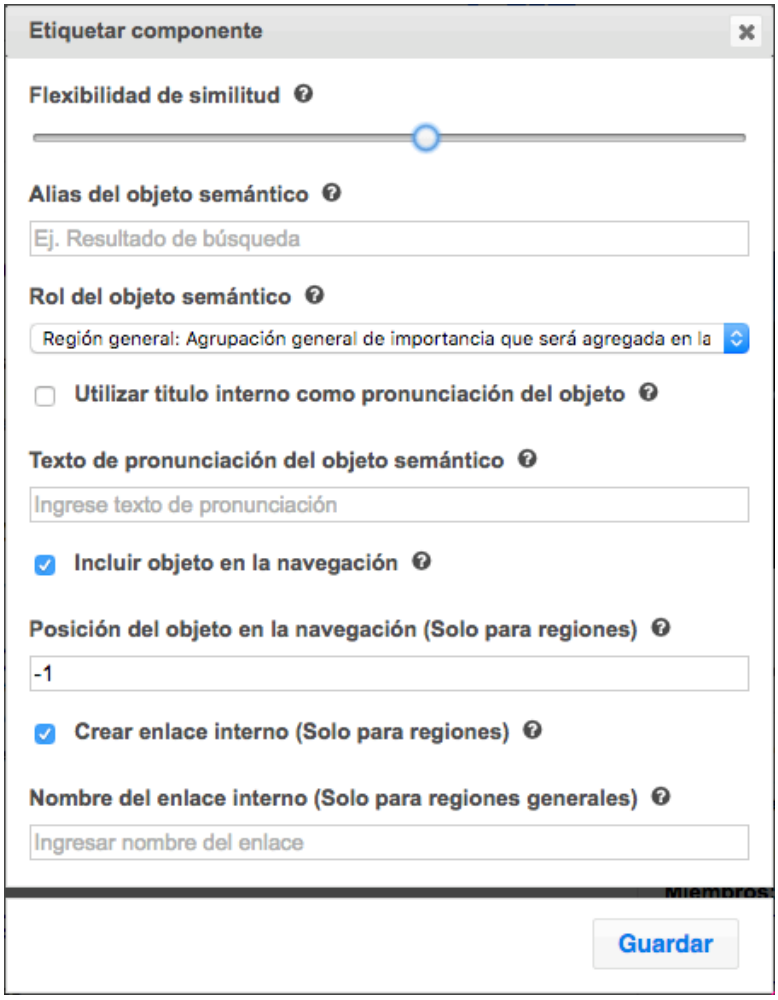


Figura 4.1.7.1.1: Captura de pantalla del formulario de información semántica. El mismo se mostrará luego de que el usuario finalice la creación del objeto semántico.

La información solicitada por el formulario de información semántica es la siguiente:

1. **Flexibilidad de similitud:** La flexibilidad de similitud es un campo "slider" el cual establece un valor numérico que será utilizado a la hora de determinar que tan "idénticos" deben ser los elementos para ser considerados como "similares". Si establecemos un valor muy bajo (posicionando el slider hacia la izquierda) podría suceder que elementos no tan idénticos al seleccionado inicialmente se consideren como similar, mientras que si seleccionamos un valor más alto (posicionando el slider hacia la derecha) esta similitud será más estricta. Si llevamos el slider al su extremo derecho, WAT interpretará que el patrón **solo** debe considerar al elemento

seleccionado inicialmente y a ningún otro. Este valor sólo puede establecerse la primera vez, luego no podrá cambiarse.

2. **Alias del objeto semántico:** Este valor es sólo utilizado para que el usuario pueda darle un nombre (alias) al objeto semántico, el cual se utilizará más adelante en el "listado de objetos semánticos", pero no incluye ningún otro tipo de funcionalidad.

3. **Rol del objeto semántico:** El rol especifica el propósito del objeto dentro de la página web. Se deberá seleccionar de una lista de opciones aquel que mejor corresponda según la consideración del usuario.

4. **Utilizar titulo interno como nombre de objeto:** De habilitar esta opción, entonces el usuario deberá garantizar que se creará un objeto semántico y con el rol de "titulo" dentro de la estructura de este objeto semántico. De ser así, esta opción hará que durante la navegación se utilice aquel elemento interno como "titulo" del elemento que lo posee. El texto que posee el elemento interno de rol "titulo" será pronunciado por el lector de pantalla cuando este objeto sea seleccionado.

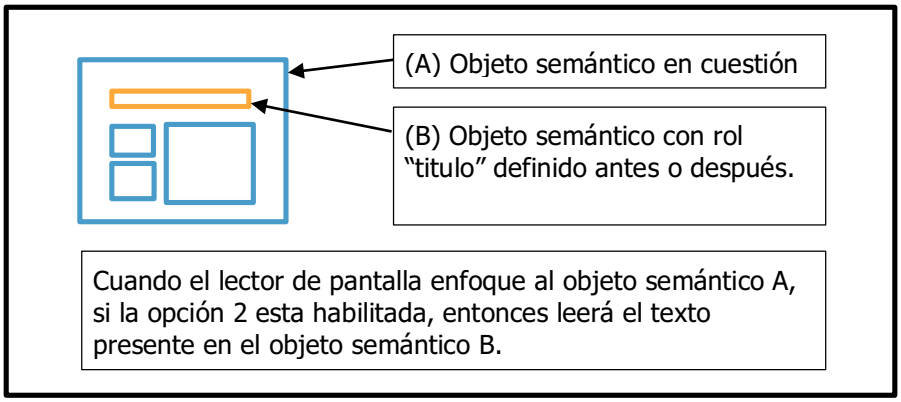


Figura 4.1.7.1.2: Ejemplo gráfico de cómo un elemento de rol "titulo" es utilizado como nombre para un objeto de mayor nivel que posee la propiedad 2 del formulario habilitada.

5. **Texto de pronunciación del objeto semántico:** Si 2* se encuentra deshabilitada, entonces el usuario puede (o no) especificar un nombre para el objeto semántico. El texto ingresado será pronunciado por el lector de pantalla cuando el mismo sea. Mantener campo en blanco para evitar esta funcionalidad.

6. **Incluir objeto en la navegación:** Esta opción, por defecto habilitada, decide si el objeto será "visible" o no durante la navegación. Es decir, que si se encuentra deshabilitada, todos los

objetos semánticos de este tipo serán excluidos (ignorados) de la navegación. Es útil para eliminar de la navegación aquellos elementos como publicidades, u otros elementos que puedan perturbar la navegación de lectores de pantalla.

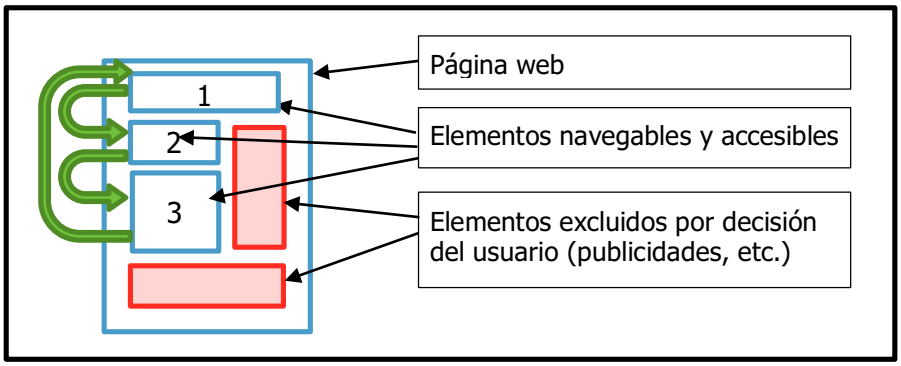


Figura 4.1.7.1.2: Ejemplo gráfico de cómo los ciertos objetos semánticos pueden ser excluidos de la navegación. Las flechas en verde representan el ciclo de navegación y los recuadros rojos aquellos elementos excluidos.

7. **Posición del objeto en la navegación (Solo para regiones):** Esta opción solo será visible si el rol seleccionado es un tipo de "región". Las regiones son objetos semánticos que representan partes del sitio (generalmente grandes) y probablemente únicas, es decir, que no suelen repetirse. El usuario puede "encerrar" varios objetos semánticos dentro de una misma región, para reducir la granularidad de la navegación del sitio. Esta opción permite redefinir el orden en que dichas regiones serán recorridas, mediante la asignación de un número que representará la prioridad de la región. (Se recorrerán de menor a mayor prioridad, desempatando por el orden inicial). Mantener un número negativo para evitar alterar el orden inicial de las regiones.

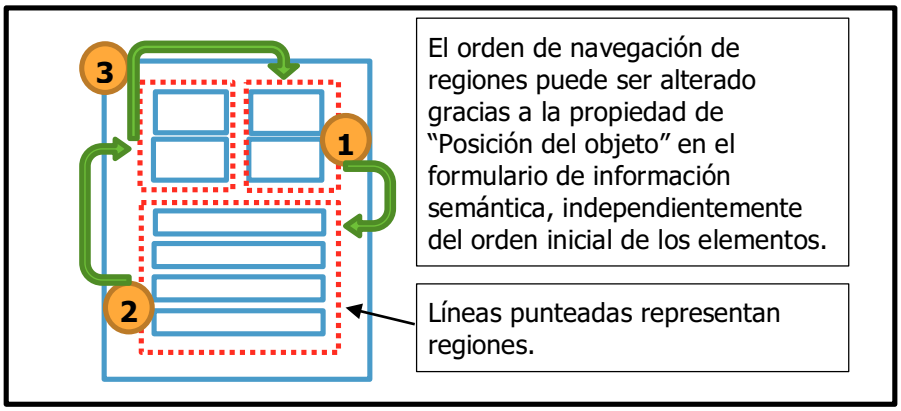


Figura 4.1.7.1.3: Ejemplo gráfico de cómo se altera el orden de navegación de los objetos que representan regiones. Las flechas verdes indican el flujo de navegación, las cuales no respetan el orden natural de los elementos gracias a la alteración realizada.

8. **Crear enlace interno (Solo para regiones):** Esta opción solo será visible si el rol es un tipo de región. Al habilitarla agregará un enlace interno al inicio de la navegación de la página web, permitiendo al usuario reubicar el puntero en la posición donde se encuentra dicha región más rápidamente. Útil para aquellas regiones que son de difícil acceso.

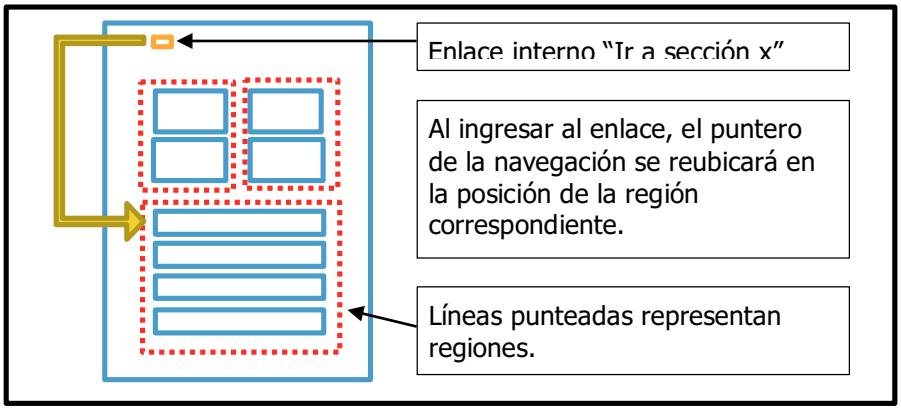


Figura 4.1.7.1.3: Ejemplo gráfico de la utilización de enlaces internos para facilitar el acceso a regiones.

9. **Nombre del enlace interno (Solo para regiones generales):** Esta opción solo será visible si se encuentra habilitada 6*. Los roles de regiones posee un nombre predefinido asociado, pero para las regiones generales, se solicita que el usuario ingrese un nombre. Dicho nombre será pronunciado por el lector de pantalla cuando el mismo haga foco sobre el enlace relativo. Se recomienda utilizar nombres que inicien con "Ir a…" por ejemplo "Ir a sección lateral", de esta forma el usuario identificará fácilmente que se trata de un enlace interno.

Luego de guardar el formulario, el Editor buscará todas las estructuras que se asemejen (considerando el índice de flexibilidad utilizado) a la estructura principal al inicio de la creación del objeto semántico y las reconocerá como objetos semánticos similares, como muestra la Figura 4.1.7.1.2.

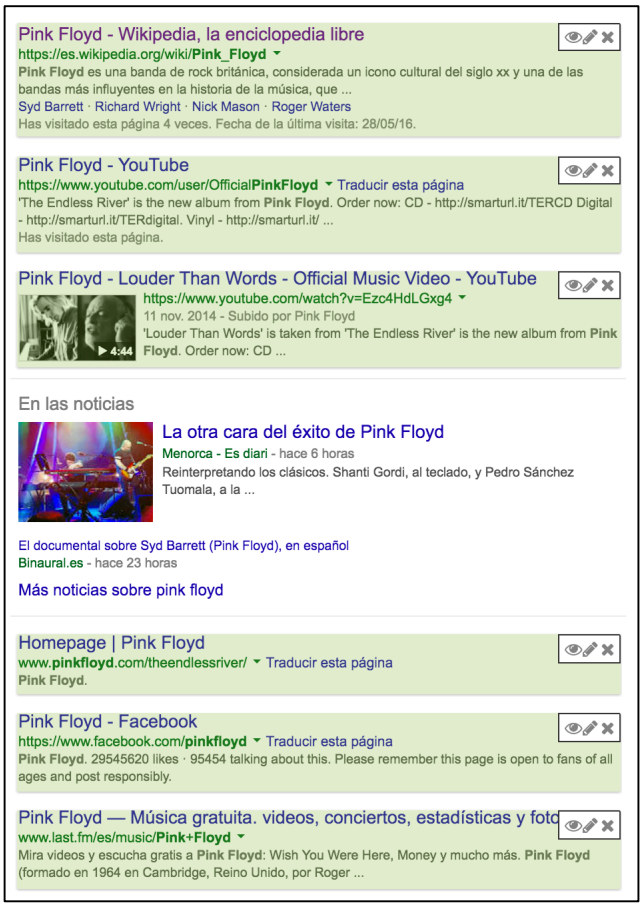


Figura 4.1.7.1.2: Captura de pantalla del sitio web Google.com, donde se muestra como varias estructuras son coloreadas de un mismo color, indicando que pertenecen a un mismo tipo de objeto semántico. La creación de este objeto semántico se realizo seleccionando una sola de estas estructuras como lo mostró la Figura 4.1.7.8, pero el Editor reconoce que existen varias similares, como resultado también les asignará la misma información semántica.

Por último, finalizada la creación del nuevo objeto semántico, se rehabilitará la funcionalidad común del mouse en el sitio web, para que continúe funcionando normalmente (sin la funcionalidad de selección de elementos).

4.1.7.2 Cancelar y volver

De optar por esta opción , el Editor cancela la creación del nuevo objeto semántico, retorna la barra de herramientas al estado anterior como el de la Figura 4.1.2.4 y se rehabilitará la funcionalidad común del mouse en el sitio web para que continúe funcionando normalmente (sin la funcionalidad de selección de elementos).

Finalmente la Figura 4.1.7.9 muestra un ejemplo de cómo varios objetos semánticos han sido reconocidos dentro del sitio web Google, representando distintos tipos de objetos semánticos.

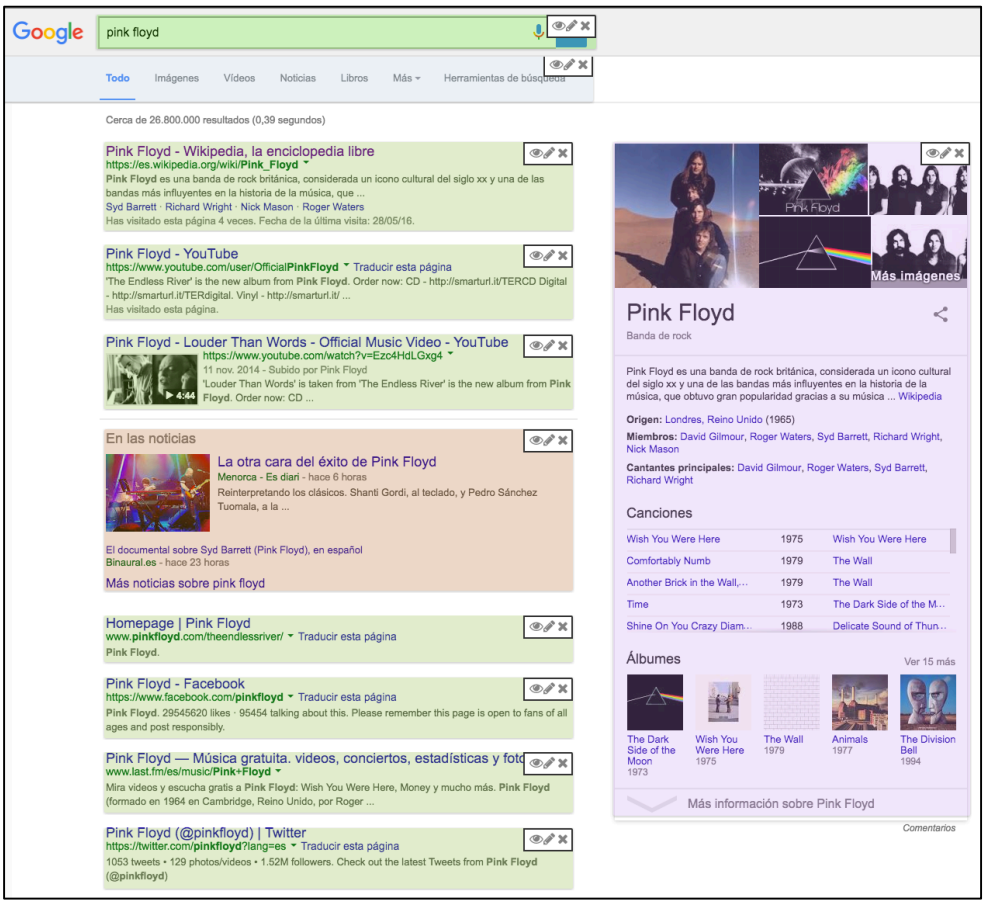


Figura 4.1.7.9: Captura de pantalla del sitio web Google.com, donde se muestra varios objetos semánticos reconocidos. Cada recuadro representa un objeto semántico distinto, y cada color de recuadro representa distintos "tipos" de objetos semánticos.

Notar que por cada objeto semántico reconocido, un recuadro de color aparecerá sobre el elemento, representando así que dicha estructura es un objeto semántico. Se utilizan recuadros de un mismo color para indicar que aquellos elementos con mismo color de recuadro pertenecen a un mismo "tipo" de objeto semántico. Por último, cada recuadro posee a su vez 3 acciones ubicadas en el borde superior derecho, como se ve en la Figura 4.1.7.10. Estas acciones corresponden a:

1. **Ocultar recuadro**: Ocultará el recuadro de color (el mismo puede volver a ser visible), es útil en caso que queramos interactuar con la página web y la misma se vea comprometida por la existencia de dicho recuadro, el cual puede ser un estorbo.
2. **Editar objeto semántico**: Reabrirá el formulario de información semántica correspondiente a dicho tipo de objeto semántico para poder editarlo. Recordar

que una modificación en el formulario afecta a todos los elementos que se categorizan por el mismo tipo (mismo color).

3. **Eliminar objeto semántico**: Eliminara el objeto semántico, y por ende toda repetición del mismo. Al igual que 2*, eliminar el objeto semántico provocará que todos los elementos similares también dejen de ser reconocidos como dicho objeto.

Figura 4.1.7.10: Captura de pantalla del sitio web Google.com, donde se muestra un resultado de búsqueda que fue reconocido como un objeto semántico. El mismo es destacado mediante un recuadro de color con un conjunto de 3 acciones en su borde superior derecho.

4.1.8 Lista de objetos semánticos

Una vez seleccionada, esta acción nos mostrará un diálogo con todos los objetos semánticos definidos en la plantilla, como se muestran en la Figura 4.1.8.1 y 4.1.8.2. Este listado facilita la gestión de todos los objetos semánticos definidos.

Figura 4.1.8.1: Captura de pantalla del diálogo con el listado de objetos definidos en la plantilla cuando no existe ningún objeto definido.

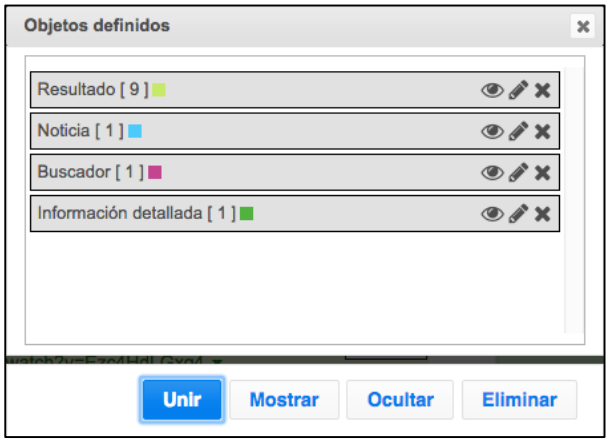


Figura 4.1.8.2: Captura de pantalla del diálogo con el listado de objetos definidos en la plantilla cuando existen objetos ya definidos.

Notar en la Figura 4.1.8.2, como se muestra una lista de objetos semánticos (identificados por su alias), junto con un número entre corchetes que representa la cantidad de objetos semánticos de ese tipo identificados en ese momento y un cuadrado de color que indica cual es el color de identificación asignado a dicho objeto semántico. A la derecha de cada fila, se encuentran 3 acciones:

- **Ocultar/Visualizar**: Permite mostrar/ocultar los cuadros de colores que se posicionan sobre cada elemento para reconocer visualmente a los objetos semánticos.
- **Editar**: Permite abrir el formulario de información semántica de dicho objeto semántico.
- **Eliminar**: Permite eliminar el objeto semántico.

Adicionalmente, el usuario puede optar por utilizar las acciones que se muestran en la parte inferior del dialogo para aplicar dicha funcionalidad a más de un objeto semántico en simultaneo. El usuario puede utilizar el mouse para seleccionar más de un objeto semántico como se muestra en la Figura 4.1.8.3.

Figura 4.1.8.3: Captura de pantalla del diálogo con el listado de objetos definidos, donde se aprecia la selección de más de un objeto semántico.

Luego el usuario puede utilizar las acciones que se muestran en el inferior del dialogo:

- **Mostrar**: Mostrará los recuadros de colores de todos los objetos semánticos que estén seleccionados (de no existir selección alguna, la acción se aplicará sobre todos los objetos semánticos).
- **Ocultar**: Ocultará los recuadros de colores de todos los objetos semánticos que estén seleccionados (de no existir selección alguna, la acción se aplicará sobre todos los objetos semánticos).
- **Eliminar**: Eliminará los objetos semánticos que estén seleccionados (de no existir selección alguna, la acción se aplicará sobre todos los objetos semánticos), en ambos casos un cuadro de confirmación se mostrará previamente.
- **Unir**: Para esta acción, si o si deberán existir al menos 2 objetos semánticos seleccionados. Esta acción asociará a todos los objetos semánticos seleccionados como un mismo objeto semántico. Esta acción no puede

deshacerse una vez hecha. La asociación de objetos semánticos solo asocia su información, es decir, cada objeto semántico seguirá actuando independientemente en el reconocimiento de elementos similares, pero la información semántica de todos ellos estará ligada. Esto es útil para cuando poseemos objetos muy distintos estructuralmente, pero que semánticamente aún representan lo mismo. La Figura 4.1.8.4 muestra como se ve el listado de objetos semánticos luego de unir los objetos de la selección que se mostró en la Figura 4.1.8.3. Vemos que ahora tanto la "noticia" como el "resultado" pasan ambos a llamarse "resultado" y poseen el mismo color de identificación. Sin embargo en el listado siguen figurando como objetos individuales. Ahora comparten su información semántica.

Figura 4.1.8.4: Captura de pantalla del diálogo con el listado de objetos definidos, donde se aprecia como los 2 objetos semánticos seleccionados en la Figura 4.1.8.3 ahora poseen la misma información semántica asociada luego de utilizar la acción de "Unir".

4.1.9 Cancelar y volver

Al igual que siempre, esta acción retornará la barra de herramientas al estado anterior, en esta instancia, se dará por finalizada la creación del proyecto, eliminando los datos locales (se perderá aquello que no haya sido guardado), y volveremos al estado de la Figura 4.1.4, donde podremos dar inicio a un nuevo proyecto.

4.1.10 Concluyendo

Usuarios voluntarios utilizarán el editor para crear o modificar plantillas. Su objetivo es definir todos los objetos semánticos que considere necesarios, asignándoles la información semántica más apropiada. Todo esto se logra fácilmente mediante el uso de las acciones "Crear nuevo objeto semántico" y "Combinar objetos semánticos", una vez que el usuario se acostumbre a la utilización de estas funcionalidades, el proceso de creación de plantillas será muy simple.

4.2 Plugin Interprete (Player)

El Plugin Interprete, o Plugin Player (de ahora en más lo llamaremos simplemente Player) es la herramienta de WAT que permite la ejecución de plantillas previamente generadas para sitios webs visitados. El Player fue programado para ser utilizado por usuarios que precisan de tecnologías de asistencia para mayor accesibilidad web, que llamaremos "usuarios finales", y ofrece herramientas de asistencia para la accesibilidad web basadas en las plantillas generadas por los usuarios voluntarios.

Una vez instalada la extensión del explorador Chrome, un nuevo ícono aparecerá en la barra superior derecha.

Figura 4.2.1: Captura de pantalla del explorador Chrome, donde se visualiza el nuevo ícono que aparece al instalar el Plugin Player.

Al presionarlo veremos un *popup* con la varias opciones entre ellas la de habilitar/deshabilitar el Player.

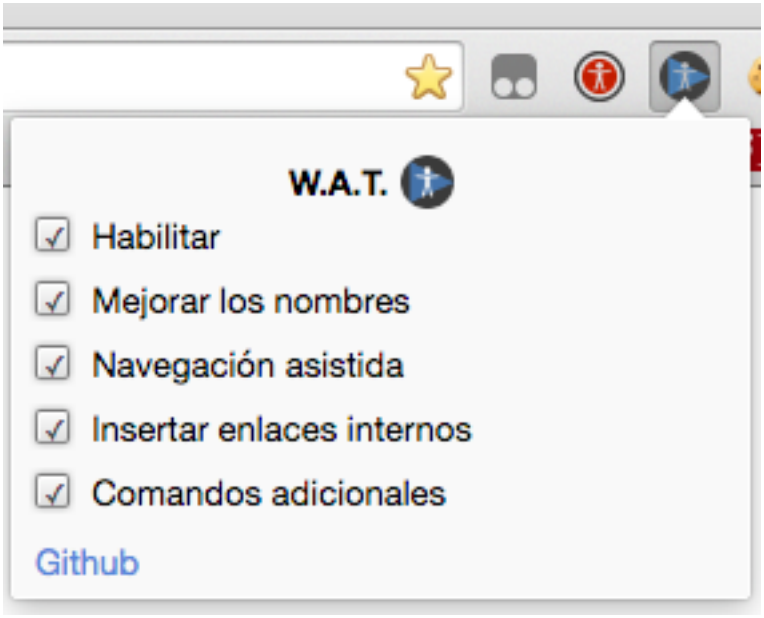


Figura 4.2.2: Captura de pantalla del explorador Chrome, donde se visualiza el popup que emerge cuando el ícono del Plugin Player es presionado.

Mientras el Player esté habilitado las tecnologías de asistencia también podrán estarlo. Pero si se deshabilita el Player, entonces ninguna funcionalidad adicional surtirá efecto. Cada funcionalidad adicional poseerá la opción de ser habilitada/deshabilitada, la misma se encuentra dentro del popup que se visualizó en la Figura 4.2.2. De esta forma, cada usuario final podrá optar por elegir cuáles son las funcionalidades que desea utilizar.

Que sucede al habilitar el Player

Cuando el Player se encuentra habilitado, cada vez que el usuario navegue a un nuevo sitio, el Player consultará a nuestro servidor por plantillas asociadas al mismo dominio. Cuando esto suceda, el Player emitirá un sonido y mostrará una pantalla con el texto "Cargando datos de accesibilidad", el Player intentará enviar el foco de la pantalla hacia dicho texto, de esta forma, si el usuario posee un lector de pantalla habilitado, el mismo debería pronunciar dicha frase, la cual, sumado al sonido emitido, ofrece parámetros necesarios para que el usuario interprete que el Player estar cargando.

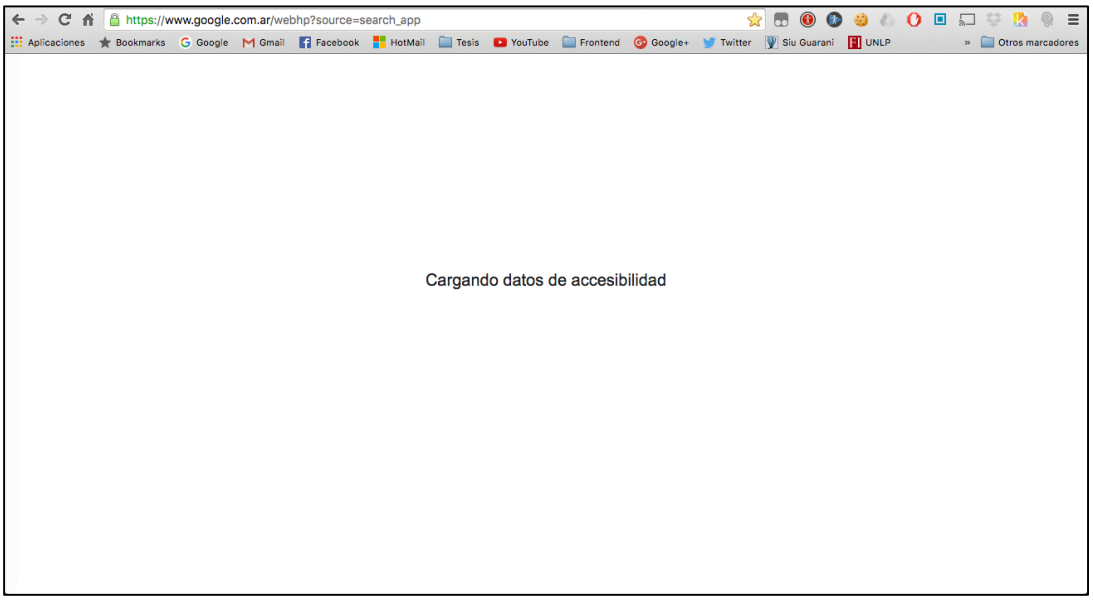


Figura 4.2.3: Captura de pantalla del plugin Player al navegar a un sitio web nuevo. Se mostrará una pantalla con la frase "Cargando datos de accesibilidad" y se emitirá un sonido particular para que el usuario interprete que el Player esta cargando.

Si el usuario no posee una sesión iniciada, no podrá acceder a nuestro servidor, es por eso cuando el Player detecta que el usuario aún ha iniciado sesión un diálogo de inicio de sesión será mostrado en pantalla, como se muestra en la Figura 4.2.4.

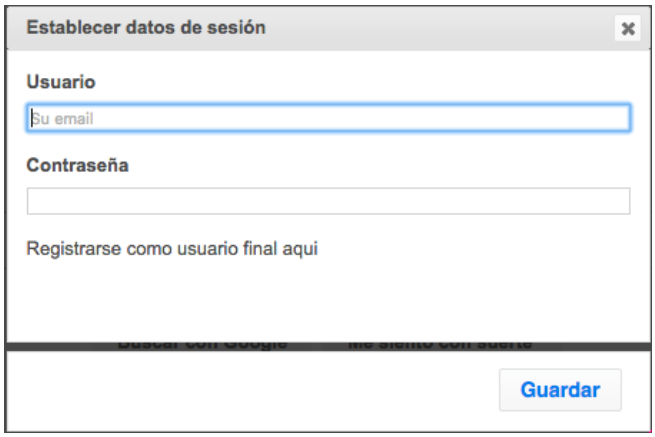


Figura 4.2.4: Captura de pantalla del diálogo de inicio de sesión del Player. El cual se mostrará únicamente si el usuario no ha iniciado sesión previamente.

Dicho diálogo posee la información de accesibilidad necesaria para poder ser completado fácilmente al utilizar lectores de pantalla, por lo que el usuario final debería ser capaz de iniciar sesión fácilmente. Este dialogo no se mostrará si el usuario ya posee una sesión iniciada. En caso de que el usuario desee iniciar sesión con otra cuenta, podrá acceder a este dialogo tras presionar las teclas "*Ctrl+L"*.

Una vez iniciada la sesión, y luego de que el Player consulte al servidor 2 cosas pueden suceder.

1. El Player no encuentra ninguna plantilla para el dominio visitado. En cuyo caso emitirá un sonido de "fallo" (distinto al de carga), que sirve como parámetro para que el usuario sepa que no se ha logrado cargar ningún dato de accesibilidad.
2. El Player encuentra una plantilla para el dominio visitado. En cuyo caso emitirá un sonido de "éxito" (distinto a los anteriores), que sirve como parámetro para que el usuario sepa que se ha logrado cargar información de accesibilidad y que por lo tanto las nuevas tecnologías de asistencia funcionarán. La información semántica definida en dicha plantilla ahora ha sido insertada automáticamente en el sitio web visitado.

¿Qué plantilla se utiliza?

Puede que existan más de una plantilla para un mismo dominio. En ese caso el Player tendrá la siguiente prioridad:

1. Utilizar la plantilla marcada como "instalada" por el usuario final (dicha marca se realiza en la aplicación servidor).
2. Utilizar la plantilla con mayor puntaje. (Los usuarios finales pueden puntuar plantillas dentro de la aplicación servidor).

En al sección 4.3 veremos como los usuarios finales pueden especificar qué plantilla instalar para cada sitio y como puntuar las plantillas existentes. Por ahora es suficiente

saber que una plantilla poseerá un puntaje dado por usuarios finales, y que usuarios finales pueden especificar que plantilla instalar para cada sitio.

La Figura 4.2.5 muestra un esquema de esta decisión de elección de plantilla.

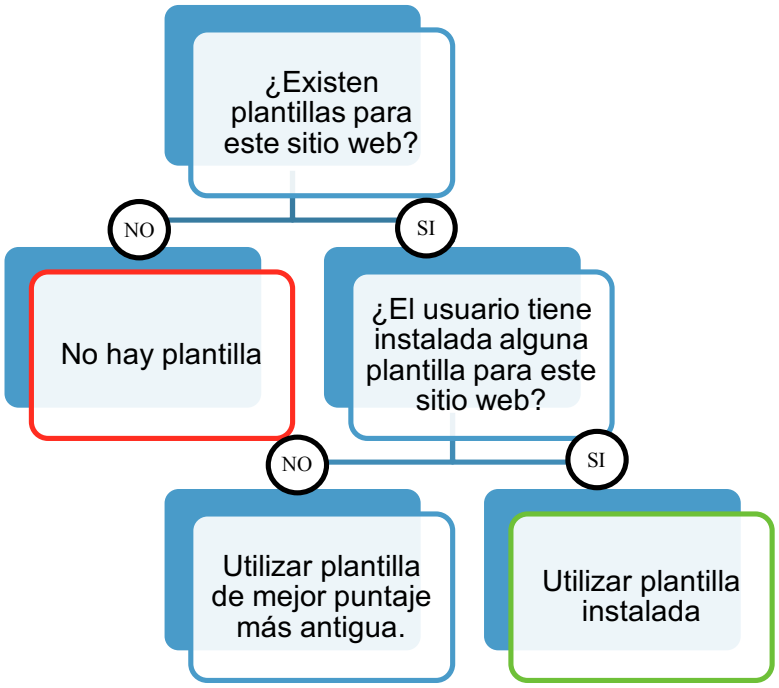


Figura 4.2.5: Esquema de decisión para la sección de la plantilla a utilizar por el Player.

Las funcionalidades adicionales que integra el Player varían de acuerdo a si el usuario las posee habilitadas o no, entre ellas se encuentran:

4.2.2 Mejorar los nombre

Al habilitar esta opción, el Player reemplazará los atributos aria-label y aria-labelledby por los valores establecidos en el formulario de información semántica. Ya sea el "texto de pronunciación" establecido, o si se marco la opción de "Utilizar titulo interno como pronunciación de objeto" en cuyo caso, el Player buscará el primer atributo de tipo titulo que se encuentre dentro del objeto semántico para asignarlo como nombre del objeto.

4.2.3 Navegación asistida

Al habilitar esta opción, se introducirá la funcionalidad de "Navegación asistida", la misma permite recorrer el sitio web considerando los objetos semánticos identificados de acuerdo a la plantilla utilizada (La sección 3.2.10 muestra más detalle de cómo funciona esta navegación, definida como "NavigationTool"). La navegación asistida

permite al usuario recorrer la página web de a "niveles", del mismo modo que recorreríamos un directorio de carpetas. Básicamente toda la navegación web podrá ser resumida en la utilización de 4 comandos:

- **Flecha abajo**: Siguiente elemento (Desplazarse al próximo hermano).
- **Flecha arriba** : Volver al elemento anterior (Retroceder al hermano anterior).
- **Flecha derecha o ENTER**: Ingresar al elemento (Nivel inferior).
- **Flecha izquierda o ESCAPE**: Salir del elemento (Regresar al nivel superior).

La navegación asistida también vendrá acompañada de la emisión de sonidos partículas que permiten al usuario reconocer cuando ingresa o sale de distintos niveles e granularidad así como cuando se desplaza entre objetos semánticos.

4.2.4 Insertar enlaces internos

Al habilitar esta opción, aquellos links internos que estén definidos en la plantilla serán insertados al inicio de la página web. Los links internos permitirán acceder más rápidamente a ciertas regiones de la página web (por ejemplo: buscador, contenido principal, etc). Que enlaces serán insertados dependerá de la información contenida en la plantilla. Los enlaces internos están limitados únicamente a regiones, esto evita que los autores de plantilla abusen de la funcionalidad y mantenga el índice corto a solo las regiones importantes.

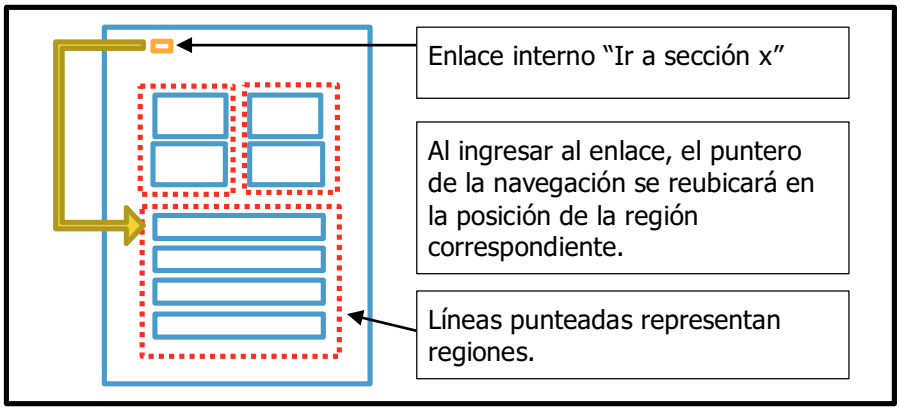


Figura 4.2.3: Ejemplo gráfico de cómo funcionan los enlaces internos.

4.2.4 Comandos adicionales

Al habilitar esta opción, ciertos comandos adicionales estarán disponibles para realizar acciones particulares. Estos comandos serán definidos por los programadores. Por el momento el único comando que se ha definido es el "*Ctrl+L*" el cual permite reabrir el formulario de inicio de sesión del Player. Aunque quedará como trabajo a futuro incorporar nuevos comandos, tales como "navegar a la región principal, o del buscador", o "puntuar positivamente a la plantilla", etc.

4.2.6 Concluyendo

Usuarios finales utilizarán el Player para utilizar las plantillas generadas por usuarios voluntarios y así obtener un conjunto de herramientas de asistencia para la navegación web. Su objetivo es hacer más accesible a los sitios web mediante el uso de la información adicional provista por las plantillas previamente almacenadas. Se busca optimizar el uso de esta información para que día a día estas herramientas provean una mejor usabilidad de los sitios web. Si bien puede que existan limitaciones, como el caso de páginas web muy complejas, las herramientas de asistencia buscan ofrecer una solución general para la navegación de páginas web simples, intentando no interponerse con las características de accesibilidad que ya existen en dichas páginas.

4.3 Aplicación web (Servidor)

Ingresando a http://transcoding.herokuapp.com/ los usuarios podrán acceder al servidor donde todas las plantillas son gestionadas. Inicialmente veremos un formulario de inicio de sesión, tal como lo muestra la Figura 4.3.1.

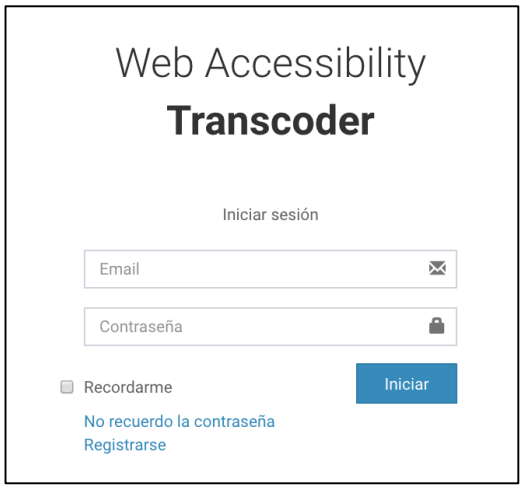


Figura 4.3.1: Captura de pantalla del formulario de inicio de sesión de nuestra aplicación servidor.

Nuevos usuarios podrán crearse siguiendo el enlace "Registrarse" del formulario de Inicio de sesión, el cual nos re-direccionara al formulario que vemos en la Figura 4.3.2.

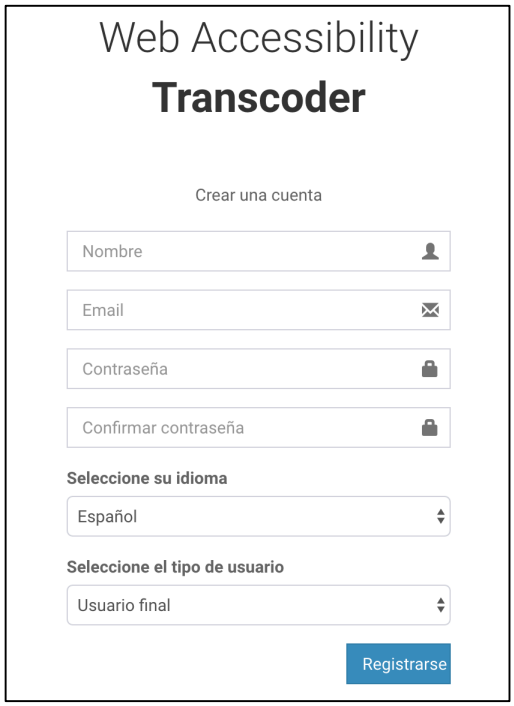


Figura 4.3.2: Captura de pantalla del formulario de registro de nuestra aplicación servidor.

Lo más importante en la creación de un nuevo usuario es la selección de "tipo de usuario", el cual ofrece dos opciones:

1. **Usuarios finales**: Aquellos usuarios que harán uso de las plantillas.
2. **Usuarios voluntarios**: Aquellos usuarios que proveerán plantillas al servidor.

4.3.1 Utilización de la aplicación servidor para usuarios finales

Pantalla principal

Una vez iniciada la sesión con un usuario final, se visualizará la siguiente pantalla tal como lo muestra la Figura 4.3.1.1:

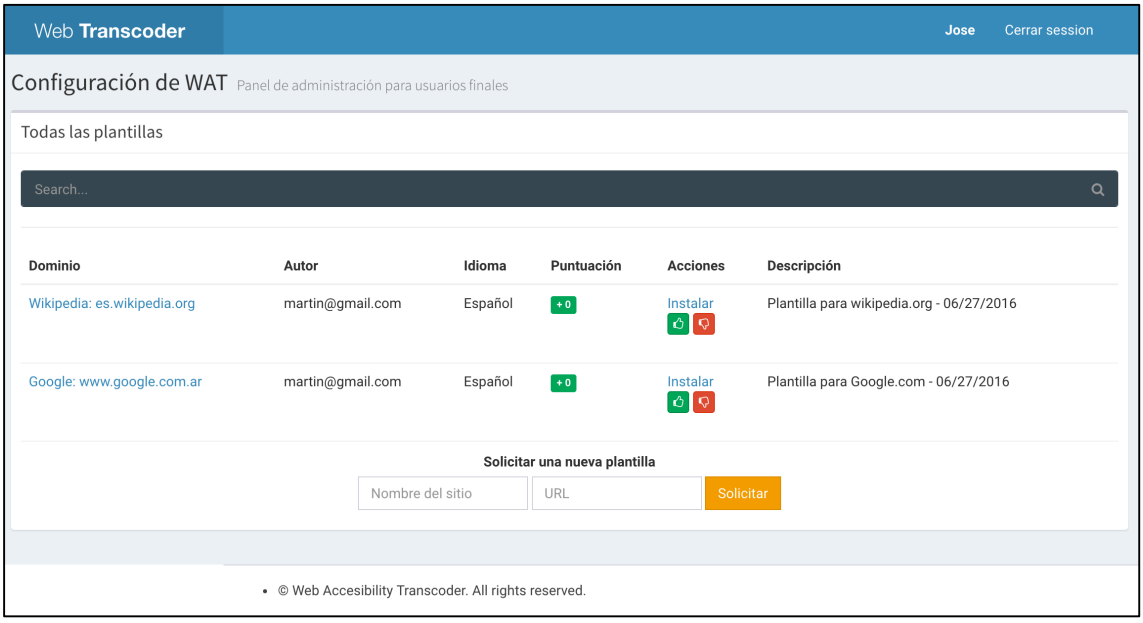


Figura 4.3.1.1: Captura de pantalla de la aplicación servidor al ingresar como usuario final.

En la pantalla de la Figura 4.3.1.1 observamos un cuadro llamado "Todas las plantillas" que a su vez se integra por 3 secciones:

1. Un buscador: Que permite filtrar plantillas por nombre y/o dominio.
2. Una lista de plantillas: Con todas las plantillas guardadas en el servidor. Esta lista cambiará de acuerdo a si el usuario ingresa algún valor en el campo de búsqueda.
3. Un formulario para la solicitud de una nueva plantilla: Donde el usuario final podrá realizar una solicitud para que usuarios voluntarios generen nuevas plantillas para paginas web determinadas.

Solicitud de nuevas plantillas

Los usuarios finales podrán solicitar nuevas plantillas, para ello deberán completar el formulario de solicitud de nueva plantilla que figura en la pantalla principal de la aplicación servidor. La Figura 4.3.1.2 muestra un ejemplo:

Figura 4.3.1.2: Captura de pantalla del formulario de solicitud de nueva plantilla para usuarios finales.

Al enviar el formulario, un nuevo cuadro aparecerá en la pantalla principal con las solicitudes pendientes que ha realizado el usuario final. En la figura 4.3.1.3 podemos ver esta nueva información que aparece a la derecha de la pantalla principal.

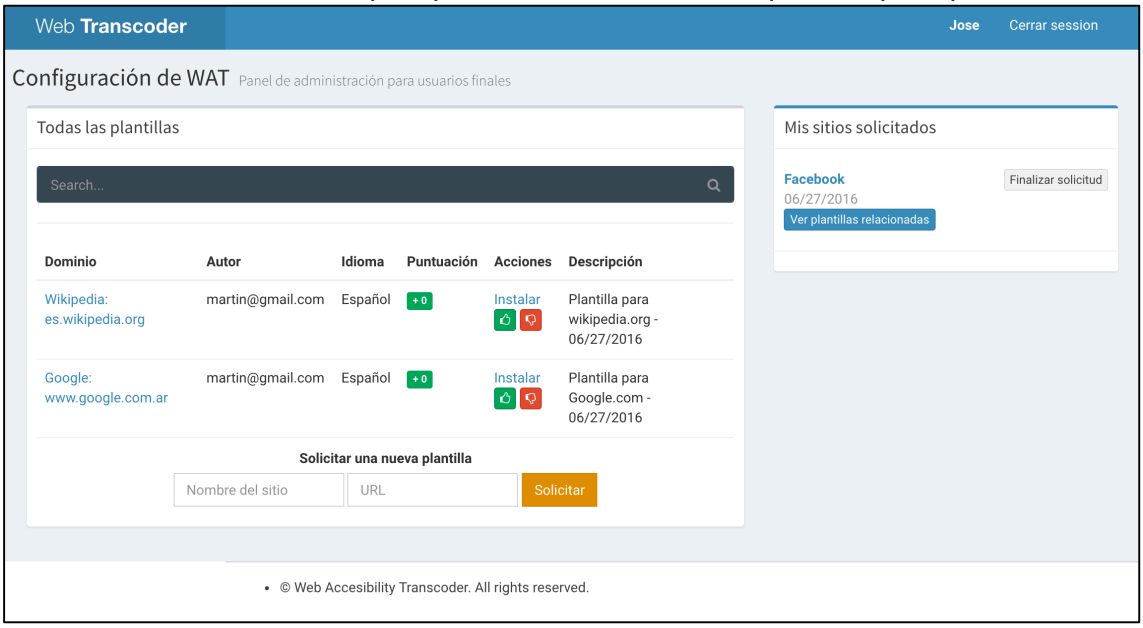


Figura 4.3.1.3: Captura de pantalla de la pantalla principal de la aplicación servidor al ingresar como usuario final. Cuando el usuario final posee solicitudes pendientes, las mismas aparecerán en un recuadro llamado "Mis sitios solicitados" a la derecha de la pantalla.

Instalación y puntuación de plantillas

El usuario final podrá instalar/desinstalar y puntuar plantillas desde la lista de plantillas que se visualiza en la página principal. En la columna acciones se se distinguen 3 opciones:

1. **Instalar/desinstalar la plantilla**: Una plantilla instalada indica que esa será la plantilla a utilizar cuando el usuario navegue por dicho sitio. El botón de instalación/desinstalación cambiará de acuerdo a si la plantilla ya se encuentra instalada o no (permitiendo la acción contraria al estado actual). Cuando un usuario instala una plantilla de un dominio para el cual ya poseía otra plantilla instalada, la aplicación automáticamente desinstalará la otra plantilla dejando únicamente instalada la ultima seleccionada, por ejemplo: Supongamos que existe la plantilla g1 y g2 para el sitio www.google.com. Primero el usuario instala la plantilla g1, y luego instala la plantilla g2, al hacerlo la plantilla g1 será desinstalada automáticamente, dejando siempre una única plantilla seleccionada para un sitio determinado. Aquellas plantillas que se encuentren instaladas poseerán una etiqueta verde "Instalada" al lado del nombre de la plantilla.
2. **Puntuación positiva**: Cuando el usuario presiona este botón, sumara un punto a la plantilla que corresponde. Si el usuario ya había otorgado un punto negativo, el mismo será removido para asignar el punto positivo.
3. **Puntuación negativa**: Del mismo modo que la puntuación positiva, pero otorgando un punto negativo. Si el usuario ya había otorgado un punto positivo, el mismo será removido para asignar el punto negativo.

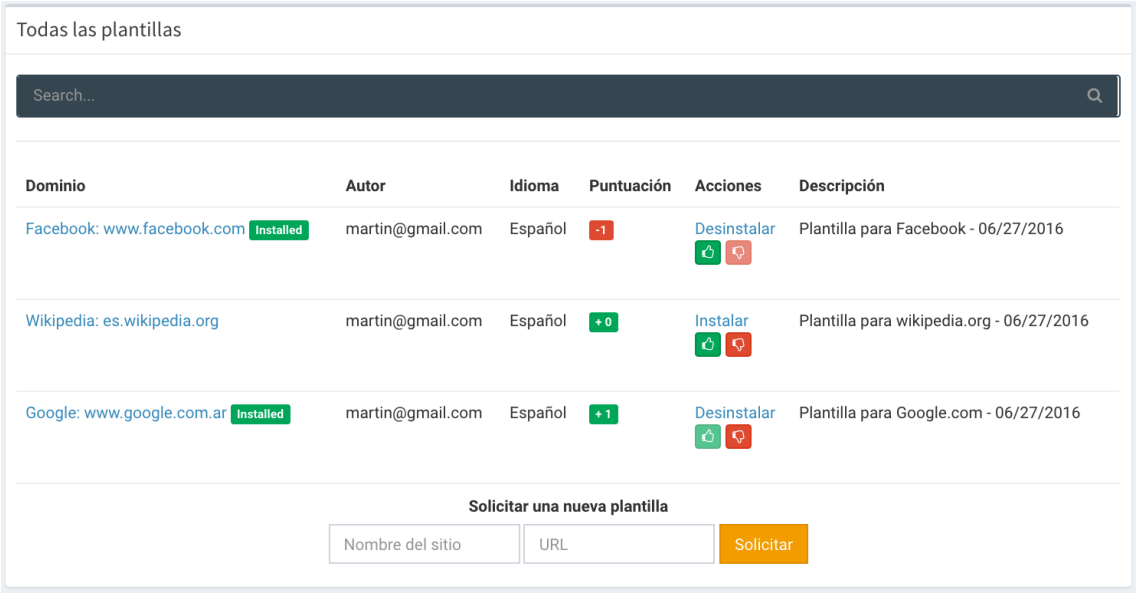


Figura 4.3.1.4: Captura de pantalla de la lista de plantillas de la pantalla principal de la aplicación servidor al acceder como usuario final. Se puede observar como algunas plantillas se encuentran "Instaladas", a la vez que poseen puntajes positivos y negativos.

4.3.2 Utilización de la aplicación servidor para usuarios voluntarios

Pantalla principal

Una vez iniciada la sesión con un usuario voluntario, se visualizará la siguiente pantalla tal como lo muestra la Figura 4.3.2.1:

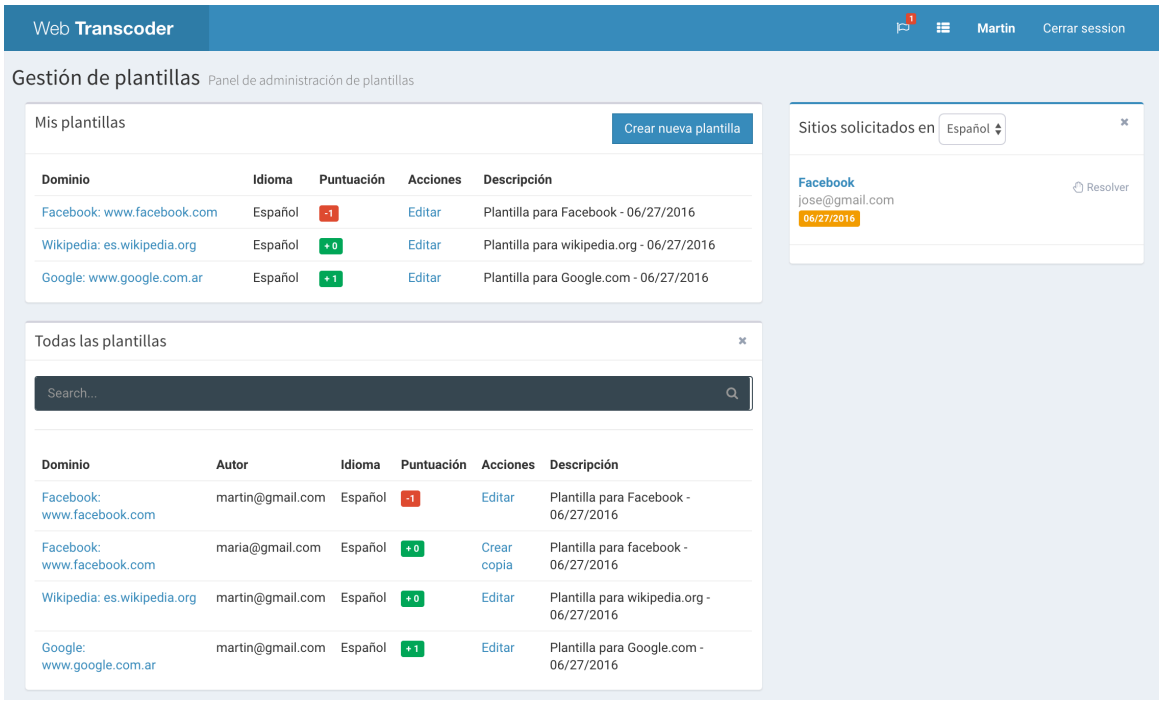


Figura 4.3.2.1: Captura de pantalla de la aplicación servidor al ingresar como usuario voluntario.

En la pantalla de la Figura 4.3.2.1 observamos tres cuadros:

1. **Un cuadro llamado "Mis plantillas"**: Aquí se visualizan todas las plantillas propias del usuario voluntario en cuestión. Por cada una de ellas, el usuario puede utilizar la acción de "editar" para ingresar a un formulario de edición de los datos de la plantilla. A su vez se encuentra un botón para la creación de una nueva plantilla. Aunque el usuario también puede lograr estas acciones desde el Plugin Editor.

2. **Un cuadro llamado "Todas las plantillas"**: Con todas las plantillas guardadas en el servidor (incluyendo las propias). Esta lista cambiará de acuerdo a si el usuario ingresa algún valor en el campo de búsqueda. Para aquellas plantillas de las cuales el usuario no es el propietario, el mismo puede realizar una copia mediante la acción "Crear copia", de esta forma los usuarios voluntarios pueden colaborar entre si en la creación de plantillas, reutilizando los trabajos de otros.

3. **Un cuadro llamado "Sitios solicitados"**: Donde el usuario voluntario podrá ver todas las solicitudes realizadas (filtradas por idioma), así podrá ver las solicitudes de acuerdo a los idiomas que conoce. Cuando un usuario final solicita una plantilla, el idioma de la solicitud corresponderá al idioma con el cual haya creado su cuenta, del mismo modo cuando los usuarios voluntarios crean una nueva plantilla, por defecto también poseerá el mismo idioma con el cual hayan creado su cuenta. Sin embargo los usuarios voluntarios pueden también colaborar en la creación de plantillas con idiomas diferentes al configurado por defecto en su cuenta, (siempre que el usuario conozca el idioma).

Resolviendo solicitudes de usuarios finales

El usuario voluntario puede utilizar la acción llamada "Resolver" presente en el cuadro de "Sitios solicitados", dicha acción lo re-direccionará al formulario de creación de nueva plantilla, para que proceda con la creación de una plantilla para resolver la solicitud. La Figura 4.3.2.2 muestra dicho formulario.

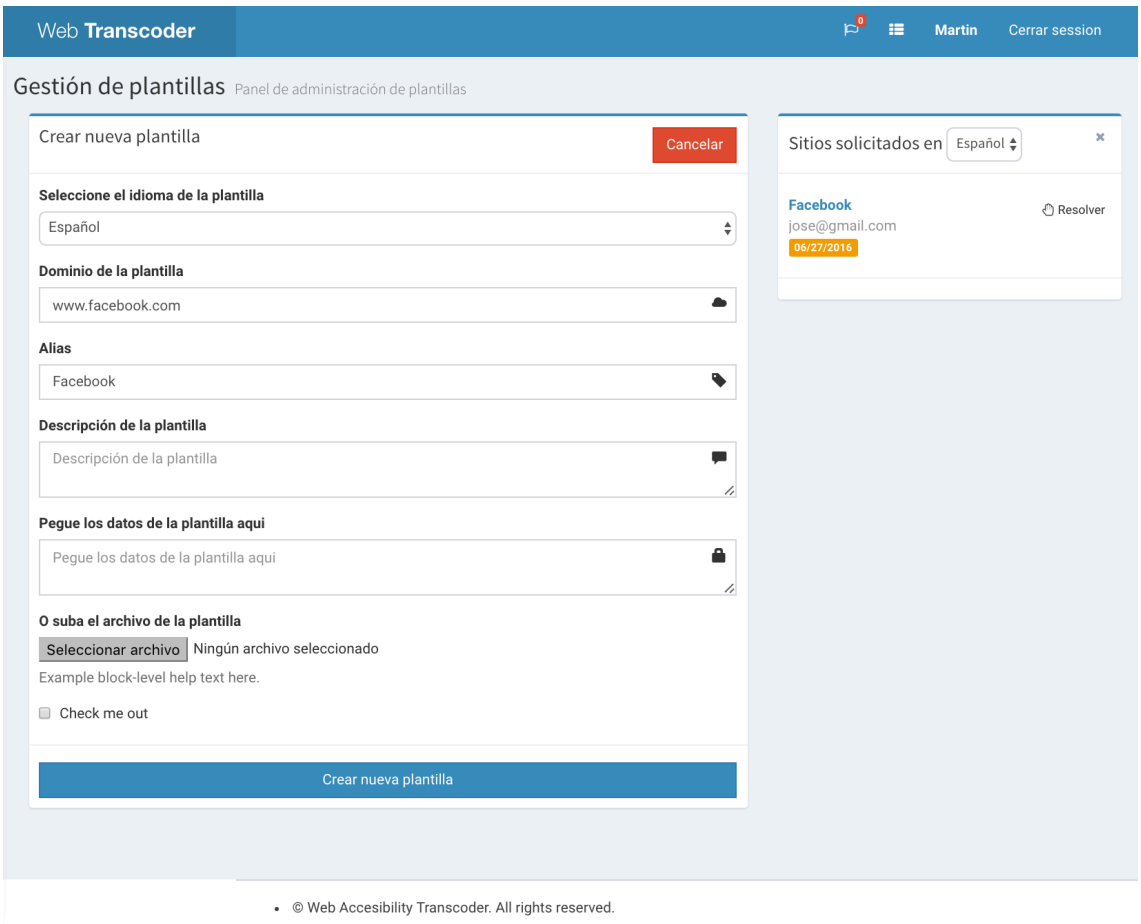


Figura 4.3.2.2: Captura de pantalla del formulario de creación de nueva plantilla.

4.3.3 Concluyendo

La aplicación web del servidor ofrece un acceso a la gestión de plantillas tanto para usuarios voluntarios como para usuarios finales. A su vez cumple un rol fundamental como plataforma de crowdsourcing, ya que es aquí donde surgen las solicitudes de nuevas plantillas y usuarios finales pueden puntuar las mejores plantillas, lo cual tendrá efecto en la descarga automática de plantillas por parte del Plugin Interprete.

Capitulo V
Manual técnico

En este capítulo describiremos la organización del desarrollo de todo este trabajo, su instalación y detalles técnicos. Vamos a dividirlo en dos partes:

1. **Organización del proyecto Cliente:** proyecto que incluye la librería WAT y los plugins Editor e Interprete que son instalados en el explorador cliente.

2. **Organización del proyecto Servidor:** proyecto que incluye la aplicación web para la gestión de usuarios y plantillas en el servidor.

Ambos proyectos fueron desarrollados utilizando el lenguaje de programación Javascript:

- En el caso del proyecto cliente se hizo uso además de la librería "Jquery", que provee varias funcionalidades que permiten agilizar el desarrollo.

- Por otro lado, para la aplicación web del servidor utilizamos el framework "Seals.js", un framework MVC que agiliza la creación de aplicaciones CRUD y APIs REST.

La librería WAT incluida en el proyecto cliente ofrece métodos de comunicación con la API REST de la aplicación web Servidor, y es así como ambos proyectos logran conectarse.

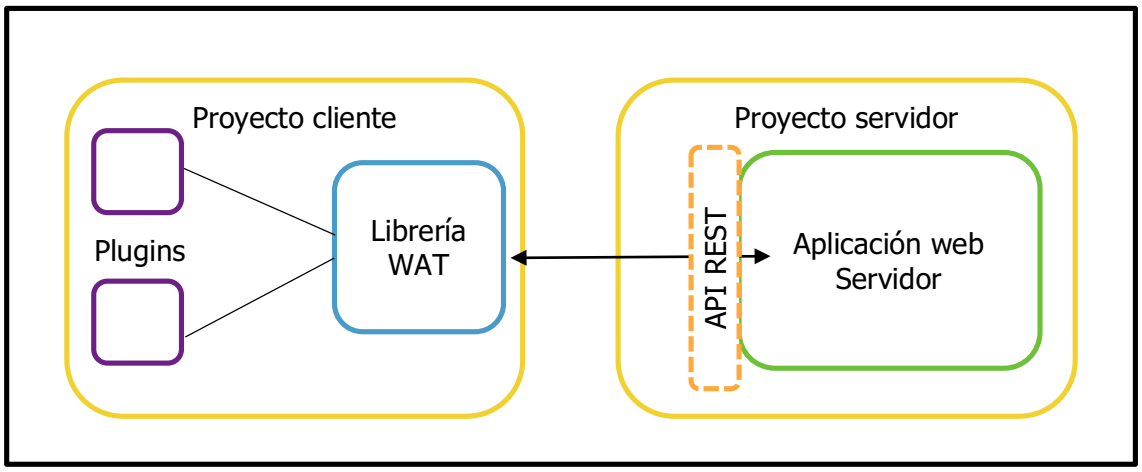


Figura 5.1: Esquema que muestra cómo se comunican ambos proyectos.

A continuación se describirá como se organiza cada proyecto y como proceder para su instalación en un entorno local.

5.1 Organización del proyecto Cliente

Este proyecto incluye:

- Librería general de WAT (código compartido por todos los plugins)
- Código correspondiente a cada uno de los plugins (Editor e Interprete).

Fue desarrollado utilizando el lenguaje de programación Javascript junto a la librería Jquery. Javascript es un lenguaje interpretado, débilmente tipado y dinámico, lo que otorga mucha flexibilidad al programador, por lo tanto, al no disponer de una única estructura de desarrollo la complejidad de los programas escritos en Javascript puede tornarse un problema cuando la cantidad de código comienza a crecer. Esto puede generar archivos de gran tamaño confusos para el programador. Si bien la librería de Jquery reduce la cantidad de código Javascript, tampoco ofrece una estructura única de trabajo.

Hoy en día existen diversos frameworks Javascript que buscan estructurar los desarrollos para obtener un proyecto más organizado y comprensible. En el desarrollo de este trabajo nos dispusimos a realizar nuestro propio método de organización: una organización modular de nuestro código.

5.1.1 Organización del proyecto cliente, desarrollo modular

A fin de mantener nuestro código organizado, optamos por un desarrollo modular. La creación de módulos nos provee los siguientes beneficios:

- Encapsulamiento de código complejo en interfaces sencillas.
- Simplicidad para emplear inyección de dependencias.
- Facilidad para testear, optimizar y reemplazar código mediante el intercambio de módulos que respeten las mismas interfaces.
- Facilita el uso de *mocks* en los test de unidad.
- Posibilidad de almacenar cada módulo en archivos físicos distintos, con nombres representativos y organizados en una jerarquía de carpetas. Esto facilita la comprensión de la organización del proyecto.

Por lo tanto, cada archivo Javascript representará entonces un módulo distinto que será dinámicamente cargado. Cada vez que definimos un nuevo módulo deberemos especificar la siguiente información:

1. Un **String** que representa **ID del módulo**, que deberá ser único entre todos los módulos cargados.
2. Un **vector** de **Strings** con los **IDs de las dependencias** del módulo.
3. Una **función de callback**, que será ejecutada una vez resueltas sus dependencias y recibirá como parámetros todas las dependencias especificadas en el vector de dependencias en el mismo orden que fueron escritas.

La lógica de carga de estos módulos se encuentra en el archivo "src/wat.js". El mismo es el responsable de la inicialización de cada nuevo módulo y de proveer la inyección de dependencias adecuada. Vale la redundancia aclarar que no debe existir dependencias circulares entre los módulos, y que las mismas deberán poder ser representadas mediante un "Sort topológico". Así, el código presente en "wat.js" cargará los módulos en el orden en que sus dependencias ya hayan sido cargadas previamente. "wat.js" ofrece una función para la definición de un nuevo módulo, la cual se encuentra definida en el *namespace* "WAT", por ejemplo:

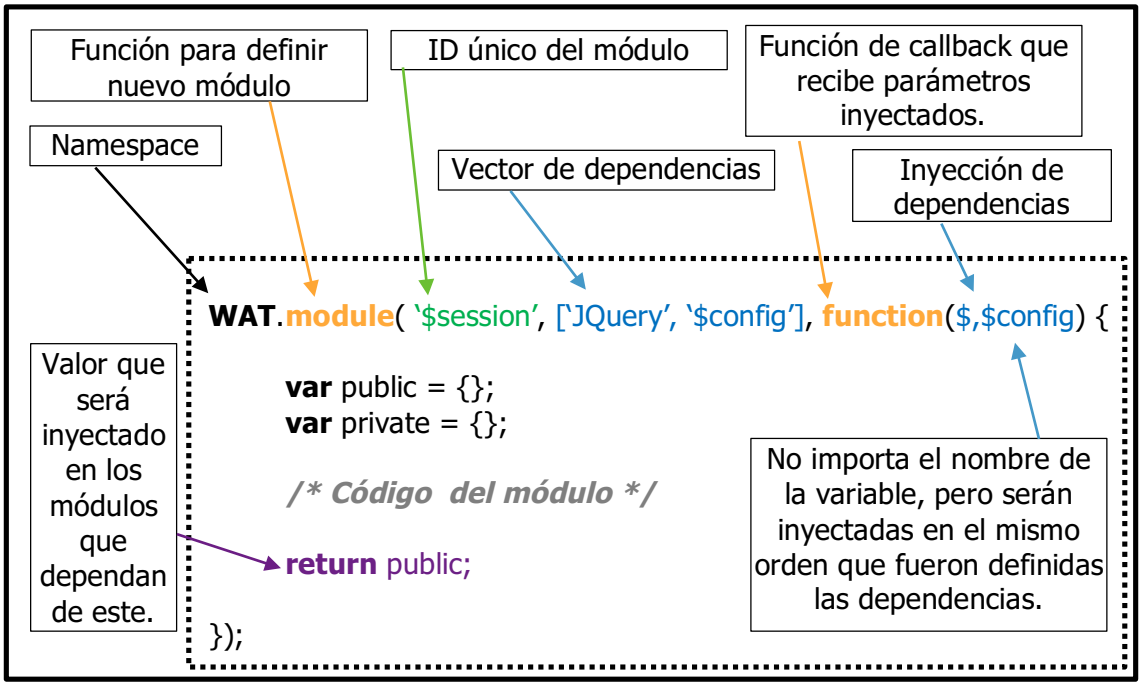


Figura 5.1.1.1: Demostración de cómo escribir un nuevo módulo de WAT.

Cada módulo es entonces un "nuevo mundo" de código, donde el programador es libre de implementar cualquier tipo de estructura siempre y cuando ofrezca una interfaz pública organizada hacia el exterior y respete la organización modular, es decir, estar consiente de que si el módulo crece demasiado, tal vez parte de su código deba ser externalizado nuevamente hacia otro módulo independiente e incluir dicha dependencia en la definición del módulo actual. Esta "libertad de escritura" nos permite utilizar los módulos de distintas formas, nosotros los hemos utilizado de 3 formas:

- **Utilizar un módulo para definir una Clase**: Si bien Javascript no integra completamente el concepto de Clase (aunque nuevas versiones lo harán), por el momento las Clases pueden ser imitadas en Javascript mediante el uso de prototipos. De esta forma podemos pensar al módulo como un método para encapsular Clases las cuales puedan ser instanciadas dentro de otros módulos.

Como se define:

```
WAT.module( 'AClass' , ['JQuery'], function($) {

        var AClass = function(){ … };
        AClass.prototype.getValue = function(){ … };

        return AClass;
});
```

Como se usa:

```
WAT.module( 'AModule' , ['JQuery', 'AClass'], function($,AClass) {
        …
        var instance = new AClass();
        instance.getValue();
        …
});
```

Figura 5.1.1.2: Demostración de cómo escribir un módulo que integra una nueva clase.

- **Utilizar el módulo para definir un *Singleton***: Bien podríamos necesitar que una Clase posea una única instancia en toda la aplicación, de esta forma podríamos utilizar al módulo para encapsular la definición de la Clase y mantenerla privada ofreciendo una única instancia hacia el resto de la aplicación.

Como se define:

```
WAT.module( '$singleton' , ['JQuery'], function($) {

        var AClass = function(){ … };
        AClass.prototype.getValue = function(){ … };

        var singleton = new AClass();
        return singleton;
});
```

Como se usa:

```
WAT.module( 'AModule' , ['JQuery', '$singleton'], function($,$singleton) {
        …
        var value = $singleton.getValue();
        …
});
```

Figura 5.1.1.3: Demostración de cómo escribir un módulo que integra un nuevo singleton.

- **Utilizar el módulo para definir un servicio/librería**: Tal vez no sea necesario implementar toda una Clase, y solo queramos utilizar el módulo para ofrecer un conjunto de funciones estáticas a modo de servicio o librería.

Como se define:

```
WAT.module( '$serive' , ['JQuery'], function($) {

        var service = {};
        service.doSomething = function(){ … };

        return service;
});
```

Como se usa:

```
WAT.module( 'AModule' , ['JQuery', '$service], function($,$service) {
        …
        $service.doSomething();
        …
});
```

Figura 5.1.1.3: Demostración de cómo escribir un módulo que integra una nueva librería/servicio.

Estas diferencias de módulos nos lleva a establecer un conjunto de convenciones adicionales para mantener la organización del código.

5.1.2 Convenciones en la organización de módulos

A modo de mantener una buena organización en la utilización de módulos hemos establecido las siguientes convenciones:

- Aquellos módulos cuya función sea integrar una nueva Clase instanciable por otro módulo de la aplicación deberán poseer como ID un nombre en CamelCase empezando en mayúscula. Ejemplo: "Foo". Luego podrá ser utilizados al estilo "new Foo()".
- Aquellos módulos cuya función sea integrar una librería de funciones o una instancia deberán poseer como ID un nombre en CamelCase empezando con una minúscula y anteponiendo el prefijo '$'. Ejemplo: "$session". Luego podrán ser utilizados al estilo "$session.doSomething()".
- Los archivos JS de cada módulo deberán llamarse de igual forma que el ID del módulo, con la excepción de no utilizar el prefijo $ en caso que el ID lo posea. Ejemplo: El módulo "Foo" estará en el archivo llamado "Foo.js", mientras que el módulo "$session" en el archivo llamado "session.js".

- Los módulos deberán de ser lo más abstractos posibles para permitir su reutilización haciendo uso de la inyección de dependencia y declarando sus tests correspondientes siempre que sea posible.

5.1.3 Organización de carpetas

El proyecto se organiza en varias carpetas, a continuación se describirá brevemente a que se dedica cada una:

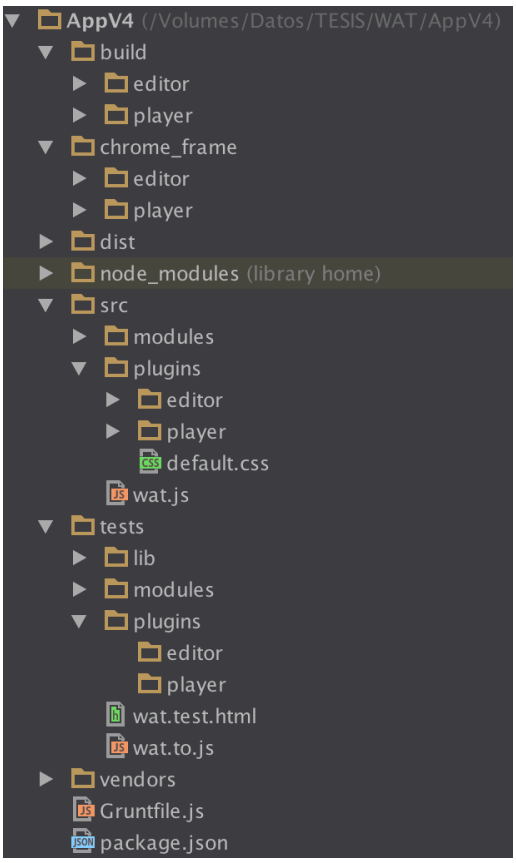


Figura 4.1.3.1: Organización de carpetas en el proyecto cliente de WAT

- **Gruntfile.js**: Grunt es un "task runner", el cual permite realizar un conjunto de tareas definidas en su archivo de configuración Gruntfile.js. Estas tareas consisten en concatenación de archivos, ofuscación de código, copiado de archivos y carpetas etc. El archivo Gruntfile.js esta configurado para servir de "compilador" de la aplicación WAT. Luego de cada cambio o finalizado el desarrollo, el programador puede compilar la aplicación tras correr comando "grunt" por consola desde la carpeta raíz del proyecto. El mismo creará dos carpetas dentro de la carpeta build, que corresponden al plugin "Player" y "Editor" de WAT para el explorador Chrome. Una vez compilado se correrán los test. Análogamente se pueden ejecutar los comandos "grunt build" y "grunt test" para disparar una sola de las operaciones detalladas anteriormente.

- **build**: Aquí se guardan los plugins para Chrome listos para instalar, luego de ejecutar grunt.

- **chrome_frame**: Carpeta donde se aloja la estructura de cada uno de los plugins de Chrome. Estos son copiados directamente en la carpeta build y se combinan con los archivos js y css de la carpeta dist.

- **dist**: Carpeta donde se aloja el código js y css concatenado y ofuscado generados durante la ejecución de grunt. Estos archivos luego serán copiados automáticamente dentro de las carpetas presentes en build.

- **node_modules**: Carpeta donde se alojan los módulos del ambiente NodeJs.

- **src**: Archivos fuente del desarrollo.

- **wat.js**: Archivo primordial del desarrollo, es el encargado de gestionar la carga de módulos y la inyección de dependencias.

- **modules**: Contiene módulos generales de la aplicación, cada uno responsable de tareas especificas. Los archivos js cuyos nombres de archivo inician con mayúsculas corresponden a módulos que integran clases instanciables por otros módulos, mientras que los archivos js cuyos nombres de archivo inician con minúsculas corresponden a módulos que integran una instancia singleton o librería.

- **plugins**: Aquí se alojan las carpetas con archivos fuentes correspondientes a cada plugin.

- **plugins/player**: Módulos de aplicación correspondientes al plugin "Player".

- **plugins/editor**: Módulos de aplicación correspondientes al plugin "Editor".

- **tests**: Respeta una subestructura similar a la carpeta src, con la diferencia que en vez de alojar archivos js, posee archivos .html. Cada archivo html posee en su interior código javascript con test de unidad correspondientes a cada uno de los módulos definidos en src.

- **vendors**: Archivos js de terceros.

- **package.json**: Archivo de paquete de NodeJs. Aquí se declaran las dependencias NodeJs de la aplicación. Para instalar las dependencias ejecutar el comando "node install" desde la raíz del proyecto.

5.1.4 Configuración del entorno para desarrollo local

Para el desarrollo de este proyecto requerimos previamente poseer instalados:

- El explorador Chrome.
- Node.js

Dado que la instalación de estos programas puede variar de acuerdo a cada sistema, obviaremos la explicación de cómo realizar cada instalación.

Una vez instalado node.js, dirigirse a la carpeta del proyecto y ejecutar el siguiente comandos: "**npm install**".

Las dependencias de este proyecto se encuentran en el archivo package.json por lo que node.js sabrá cómo resolverlas. Finalizado el proceso anterior, el entorno estará listo.

5.1.5 Compilación e instalación de plugins

Como vimos, el proyecto cliente se organiza en varios módulos separados en archivos físicos distintos. Utilizamos Grunt para unir todos estos archivos separados en un único compilado (un archivo js minificado y ofuscado). El archivo de configuración Gruntfile.js especifica todos los pasos realizados durante esta "compilación". Por lo cual solo deberemos correr el siguiente comando por consola desde la carpeta raíz del proyecto: "***grunt***" o *"**grunt release**".*

Este comando esta configurado para realizar los siguiente pasos:

1- **Limpiar** todo el contenido anterior de las carpetas "dist/" y "build/".
2- **Concatenar** todos los archivos JS y CSS a un único archivo correspondiente:

- Unirá el archivo wat.js y todos los archivos js presentes en las carpetas "vendors" y "src/modules" en un único archivo "dist/wat-framework.js"

- Unirá el archivo "dist/wat-framework.js" con todos los archivos js presentes en la carpeta "src/plugin/editor" en un único archivo "dist/wat-editor.js"

- Unirá el archivo "dist/wat-framework.min.js" con todos los archivos js presentes en la carpeta "src/plugin/player" en un único archivo "dist/wat-player.js"

- Unirá todos los archivos css presentes en las carpetas "src/modules", "src/plugin/editor" y "vendors" en un único archivo "dist/wat-editor.css"

- Unirá todos los archivos css presentes en las carpetas "src/modules", "src/plugin/player" y "vendors" en un único archivo "dist/wat-player.css"

3- **Ofuscar** todos los archivos js dentro de la carpeta "dist/", a un archivo del mismo nombre pero con la extensión min.js y min.css según corresponda.
4- **Limpiar** todos los archivos de la carpeta "dist/" que no posean extensión "min.js" o "min.css"
5- **Copiar** las **carpetas** "editor" y "player" de "Chrome_frame" dentro de "build". Cada una de estas carpetas representa una maqueta con la estructura y archivos necesarios para crear una extensión del explorador de Chrome.
6- **Copiar archivos** min.js y min.css de la carpeta "dist", a su respectiva carpeta "build/editor" y "build/player".
7- **Ejecuta todos los test** configurados en la carpeta "test/".

Finalizado este proceso poseeremos dos carpetas dentro de la carpeta "build". Cada carpeta corresponde a una extensión (plugin) de Chrome, el Editor y el Player respectivamente. Luego podremos montar el plugin desde el explorador Chrome en la sección de extensiones, utilizando el botón de "cargar extensión descomprimida" y seleccionando la ubicación de cada una de las carpetas dentro de "build".

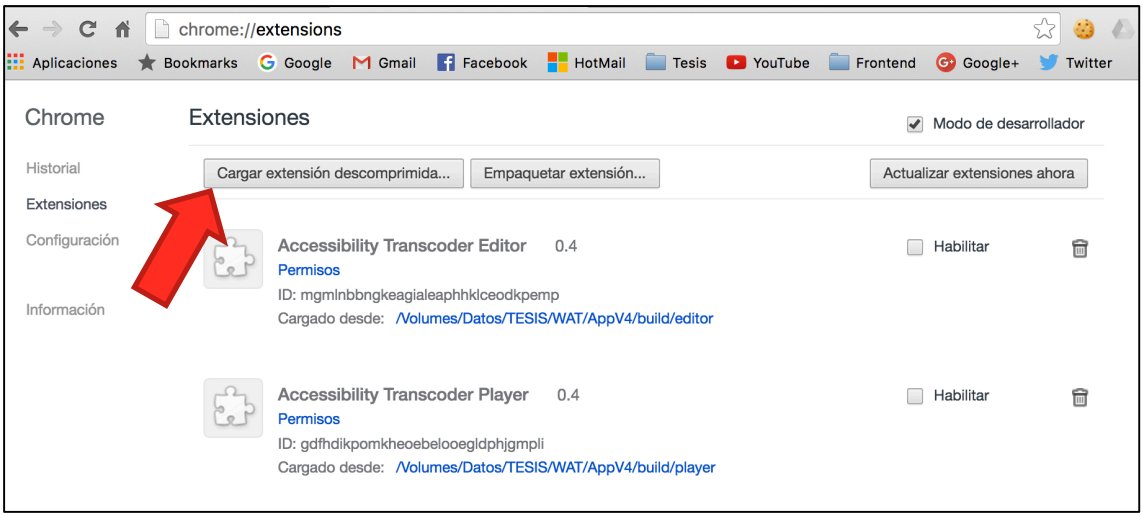


Figura 5.1.5.1: Captura de pantalla del explorador Chrome en la sección de extensiones para agregar nuevas extensiones al explorador.

Alternativamente podemos ejecutar los siguientes comandos configurados en el archivo Gruntfile.js:

1. "**grunt test**": Ejecuta únicamente los test.
2. "**grunt build**": Concatena los archivos correspondientes y los mueve a su respectivo destino dentro de "build/", sin ofuscarlos (para verlos durante el desarrollo) y sin ejecutar los test.
3. "**grunt watch**": Ejecutará el comando "grunt build" cada vez que un archivo js sea modificado, esto es útil durante el desarrollo.

5.2 Organización del proyecto Servidor

Este proyecto fue desarrollado respetando la estructura del framework Sails.js, por lo que no entraremos en detalle sobre su estructura y organización dado que toda esa información puede encontrarse en su sitio oficial http://sailsjs.org. El objetivo de esta sección será simplemente mostrar como configurar el entorno local para su desarrollo.

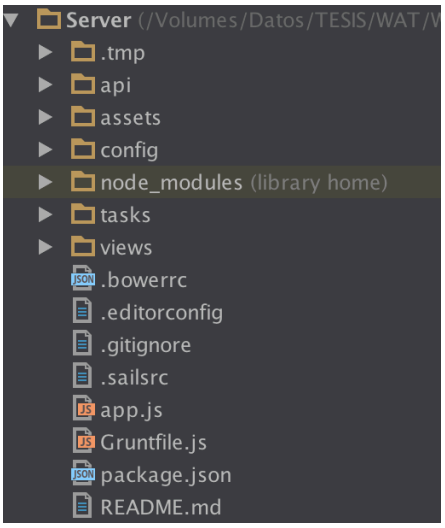


Figura 5.2: Organización de carpetas en el proyecto servidor de WAT

El objetivo de este proyecto es tan solo proveer de una API REST a la aplicación Cliente y de un Front-End para los usuarios para la persistencia de datos, gestión de usuarios y plantillas. Por lo que no se hará mucho hincapié en describir como esta implementado.

5.2.1 Configuración del entorno para desarrollo local

Para este proyecto requeriremos poseer instalados:

- Node.js
- Sails.js

Dado que la instalación de estos programas puede variar de acuerdo a cada sistema, obviaremos la explicación de cómo realizar cada instalación, la cual puede ser encontrada en la página web oficial de cada uno de los programas.
Una vez instalados los programas anteriores, nos dirigiremos a la carpeta del proyecto servidor. Ingresaremos al archivo "config/models.js" y nos aseguraremos de que ambas propiedades "connection" y "migrate" se encuentren comentadas. Luego desde una consola de comandos ejecutaremos: "**sails lift**". El programa se ejecutará utilizando una base de datos temporal. Para una correcta configuración de la base de datos visitar la documentación del framework sails.js en http://sailsjs.org.

Capitulo VI
Conclusiones y trabajos futuros

6.1 Conclusiones

6.1.1 Introducción de nueva información para la accesibilidad

La accesibilidad web no es algo simple de lograr sobre todo si los programadores de cada sitio web no se hacen cargo de ella. El enfoque presentado en este trabajo otorga una solución a este problema permitiendo a otros usuarios definir e incluir su propia información adicional aunque ellos no posean acceso para la edición del sitio web, ni tengan amplios conocimientos de programación. Hemos logrado abstraer los aspectos técnicos mediante la introducción del formulario de información semántica y el concepto de "objetos semánticos".

6.1.2 La importancia del concepto de objetos semánticos

La introducción del concepto de "objeto semántico" es útil para comprender que es lo que los usuarios voluntarios deben "etiquetar". Una vez inculcado el termino, los usuarios voluntarios deberían agrupar aquellos elementos HTML que poseen cierto grado de relación semántica y no simplemente realizar agrupaciones arbitrarias.

6.1.3 La importancia del formulario de información semántica

El uso del "formulario de información semántica" es muy importante para lograr abstraer todos los detalles técnicos a los usuarios voluntarios que carezcan de dichos conocimientos. Por ejemplo: el formulario podría contener el campo "Introduzca una breve descripción de este objeto", posteriormente se utilizará dicho valor para asignárselo al atributo "aria-label" del elemento HTML en cuestión, logrando así abstraer dicho detalle técnico a una simple pregunta de formulario no técnica.

La creación de este formulario es responsabilidad del desarrollador, quien debe asegurarse de ser claro en la creación de nuevas preguntas (asociadas a valores técnicos) que puedan ser fácilmente comprendidas por usuarios no técnicos.

6.1.4 El uso de la nueva información semántica para mejorar las funcionalidades de accesibilidad

Por el momento, y durante el desarrollo de este trabajo, hemos configurado el formulario de información semántica para que usuarios voluntarios completen un conjunto de campos particulares, entre los cuales podemos destacar 2, el primero utilizado para detectar qué rol (atributo "role") asignar al elemento HTML en cuestión, y el segundo para obtener un nuevo valor a insertar en el atributo "aria-label". Tanto

el atributo "role", como el "aria-label", poseen gran importancia para las herramientas de lectura de pantalla, dado que el primero le permite identificar el tipo de elemento que se esta navegando, mientras que la segunda asigna un texto para ser pronunciado y ayudar al usuario a comprender de qué elemento se trata. Por lo tanto, así como hemos dado uso al formulario para obtener esta información, otros desarrolladores pueden incluir nuevas propiedades (WAI-ARIA u otra) que permitan ir mejorando poco a poco nuevos aspectos para la accesibilidad de los sitios web.

6.1.5 El uso de la nueva información semántica para simplificar la interpretación de páginas web

Casi todas las herramientas ajenas a los sitios web desarrolladas para brindar mayor accesibilidad poseen la misma desventaja: deben ser capaces de funcionar y comprender códigos HTML completamente distintos.

La inserción de información semántica personalizada nos permite estandarizar ciertos aspectos de dichos códigos HTML tras el reconocimiento de objetos semánticos los cuales son asociados a información detallada. Luego las nuevas herramientas de accesibilidad podrán hacer uso de esta información adicional independientemente del sitio en donde se ejecuten, confiando en que WAT provea la información necesaria para su correcta ejecución. Hemos visto un ejemplo de esto con la herramienta de navegación desarrollada en la sección 3.2.10, la misma recorría los elementos HTML que posean la clase llamada 'wat-navigate', información insertada por la herramienta WAT previo a la ejecución de la herramienta de navegación. Por tanto, de no poseer esta información "ya procesada", la herramienta de navegación hubiese tenido que interpretar por si misma el sitio web para comprender que elementos debía navegar y cuales no.

Esta ventaja de "inyectar" nueva información personalizada a las páginas web, sumada a la rapidez con la que nuevas plantillas pueden crearse utilizando el *plugin* Editor*,* nos permite simplificar el desarrollo de nuevas herramientas de accesibilidad en cuanto a la interpretación del código HTML.

6.1.6 Crowdsourcing como método de expansión

El Crowdsourcing como base para la generación de contenido de accesibilidad puede ser una herramienta muy poderosa, si bien no hemos logrado explotarla al máximo debido al enfoque en otras funcionalidades, quedará como un trabajo futuro poder obtener mayor provecho del mismo.

6.1.7 Algoritmo de reconocimiento de estructuras similares

El algoritmo de reconocimiento de estructuras HTML similares es uno de los mayores aportes de este trabajo y su aplicación no queda limitada al campo de la accesibilidad, sino que puede ser útil para cualquier otro. Otros trabajos similares pueden ser encontrados en artículos [Buttler 2004; Nierman and Jagadish 2002; Omer et al.

2012], pero la novedad de nuestro algoritmo radica en su estructura lineal así como en las distintas estrategias de búsqueda aplicadas que en conjunto ofrecen una solución estable y rápida.

6.1.8 Desarrollo Javascript basado en módulos

El desarrollo de una aplicación Javascript a gran escala también represento un reto. El desarrollo modular permitió mantener la estructura del código de forma organizada y facilitar la inyección de dependencias y reutilización del código al máximo. Finalmente, la necesidad de la aplicación de los test de unidad fue fundamental, dado que facilitó mantener la consistencia del programa durante todo su desarrollo.

6.2 Contribuciones

6.2.1 Estudio y clasificación de diferentes métodos y herramientas actuales que buscan mejorar la accesibilidad de las páginas web

En el capítulo 2 hemos realizado un análisis a modo de poder describir y clasificar otras herramientas relacionadas a la accesibilidad web.

6.2.2 Análisis comparativo de las técnicas de refactoring y transcoding

Debido a que el método ofrecido en este trabajo combina aspectos de las técnicas de refactoring y transcoding, en la sección 2.1 ofrecemos una comparación de las mismas para comprender en mayor detalle que comprende cada una y poder diferenciarnos de ellas.

6.2.3 Enumeración de problemáticas halladas en el uso de lectores de pantalla

En la sección 3.2.1 ofrecemos un listado de las problemáticas que hemos detectado en el uso de lectores de pantalla y que afectan directamente a la experiencia de usuario.

6.2.4 Definición de un método para mejorar la accesibilidad web mitigando las problemáticas definidas anteriormente

En la sección 3.4 explica como los conceptos y funcionalidades descritas a lo largo del capitulo 3 pueden ser utilizadas como un método para mitigar las problemáticas encontradas y definidas en la sección 3.2.1.

6.2.5 Implementación de un framework extensible que implementa dicho método

La sección 3.3 describimos una arquitectura que implementa el método descrito a lo largo del capítulo 3, la cual a su vez incluye a su vez el desarrollo de 2 plugins Javascript:

- Un plugin para definir nueva información semántica para distintas páginas web.
- Un plugin para incluir dicha información semántica y nuevas funcionalidades a lo largo de la navegación web.

Toda esta arquitectura puede verse como un framework:

- De caja negra: si los desarrolladores modifican/extienden las entradas ya definidas, como el formulario de información semántica o conectando nuevos scripts (herramientas) al plugin Interprete.
- O de caja blanca: los desarrolladores pueden definir nuevos plugins y/o extender las funcionalidades internas respetando la estructura actual.

6.2.6 Desarrollo e implementación de un algoritmo de similitud de elementos HTML para la definición de objetos semánticos.

En la sección 3.2.3 introducimos un nuevo algoritmo de reconocimiento de estructuras HTML similares. Durante la creación de objetos semánticos, el usuario voluntario selecciona un elemento HTML que actúa como "representante" del nuevo objeto semántico definido. Hacemos uso de este nuevo algoritmo de similitud para encontrar estructuras similares a dicho "representante" y etiquetarlas con el mismo objeto semántico, logrando así agilizar el proceso de creación de plantillas.
A su vez, relacionamos a este nuevo algoritmo con 3 estrategias de búsqueda propuestas para optimizar la tarea de encontrar aquellos elementos similares.

6.2.7 Plataforma de crowdsourcing.

Debido a la inmensidad de la internet, utilizamos el crowdsourcing como estrategia para escalar nuestra solución. La arquitectura presentada en la sección 3.3 se encuentra diseñada en base a este concepto, donde usuarios finales pueden solicitar la generación de nuevo contenido mientras que los usuarios voluntarios serán los encargados de resolver dichas solicitudes.

6.2.8 Manual de usuario con distintos roles

En el capítulo 4 hacemos una descripción detallada de todas las funcionalidades brindadas por los plugins ofrecidos para los distintos tipos de usuarios.

6.2.9 Documentación detallada del código para programadores.

El capítulo 5 brinda toda la información necesaria para que otros programadores puedan continuar con el desarrollo de las herramientas. A su vez, todo el código desarrollado a lo largo de este trabajo posee su propia documentación correctamente detallada.

6.3 Trabajos futuros

Tras el extenso desarrollo de este trabajo quedan abiertos nuevos trabajos futuros por implementar:

6.3.1 Autoría colaborativa en el desarrollo de plantillas

Como se mencionó en el capítulo 4, el *plugin* Editor permite la creación de nuevas plantillas asociadas a un único dominio. Dichas plantillas poseen objetos semánticos definidos por un único autor. Quedará abierta la posibilidad de idear un desarrollo colaborativo, en donde las plantillas se generen a partir de conjuntos de objetos semánticos definidos por múltiples autores, reduciendo la carga individual en la creación de nuevas plantillas.

6.3.2 Mejora en la integración de la navegación asistida con otros lectores de pantalla

Durante nuestro desarrollo, la mayoría de nuestras pruebas de integración se basaron en la combinación del lector de pantalla *ChromeVox* (versión 53.0) en el explorador Chrome (versión 51.0 - 64-bit). Optamos por esta configuración, dado que no solo es gratuita, sino que a su vez puede ser utilizada en cualquier sistema operativo y configurada muy fácilmente. Nuevas pruebas podrían realizarse para comprobar la integridad de nuestra aplicación con otras combinaciones de exploradores/lectores, haciendo las modificaciones necesarias para obtener una mejor integración entre ambos sistemas.

6.3.3 Implementaciones propuestas no concluidas

Entre las ideas propuestas pero no implementadas se encuentran:

-El uso del campo de "posición del objeto semántico dentro de la navegación" incluido en el formulario de información semántica, el cual permitiría alterar el orden del recorrido de los objetos semánticos. Ya hemos mencionado anteriormente un trabajo relacionado [Sato et al. 2009], en el cual los autores se encuentran con el obstáculo de que el atributo WAI-ARIA "aria-flowto" aún no es soportado por la mayoría de los lectores de pantalla y el cual representaría la mejor estrategia (estandarizada) para aplicar dicha funcionalidad. Por lo tanto, consideramos que lo mejor es dejar esta implementación como un trabajo a futuro para el momento en que dicho atributo sea correctamente respetado por todos los lectores de pantalla.

-Selección de ventanas emergentes: A veces, en la creación de un nuevo objeto semántico, resulta dificultoso seleccionar ciertas ventajas emergentes que desaparecen al momento de eliminar el foco sobre ellas. Por ejemplo, supongamos que una ventana emerge cuando hacemos foco en un elemento. Cuando queremos marcar dicha ventana como objeto semántico, al momento de dirigirnos a la barra de herramientas del Editor, dicha ventana ya desaparece. Por lo tanto, queda como trabajo futuro buscar un método para resolver estos casos particulares de elementos intermitentes.

6.3.4 Implementación de un lector de pantalla propio

Dado que el *plugin* Interprete (o *plugin* Player) ofrece en si, un mecanismo de navegación (Sección 3.2.10) no estaría muy lejos la posibilidad de la implementación de nuestro herramienta de lectura de pantalla. Como vimos en la Sección 4.2.2, la navegación asistida es un código independiente que se introduce al *framework* como una herramienta que el usuario puede activar o desactivar. De este mismo modo, podríamos ofrecer una herramienta que lea el código HTML que esta siendo seleccionado por la herramienta de navegación haciendo uso de algún tipo de sistema externo que provea la pronunciación de texto. Esto último evitaría la necesidad del trabajo futuro 6.3.2, dado que ya no requeriremos integrar WAT con un lector de pantalla, sino que proveeríamos el nuestro.

6.3.5 Eliminación de la aplicación servidor

La aplicación web desarrollada en el servidor podría ser reemplazada si incluimos sus funcionalidades dentro de los *plugins* Editor e Interprete (es decir, creación de cuenta, gestión y solicitud de nuevas plantillas). Por simplicidad no hemos realizamos esta tarea en el desarrollo de este trabajo pero bien podría hacerse dejando el uso del servidor únicamente como almacenamiento de datos.

6.3.6 *Prunning* en el algoritmo de similitud

Retomando lo hablado en la Sección 3.2.3, Optimizaciones; otra idea que decidimos no implementar es la introducción de un *Prunning* en el algoritmo de similitud. Cuando el usuario reconoce objetos semánticos que se encuentran dentro de otros objetos semánticos de mayor nivel podemos utilizar esta información a nuestro favor. Si sabemos que el objeto semántico B se encuentra dentro del objeto semántico A y si sabemos que esto se cumple para todos los objetos A y B, entonces podemos eliminar ("*prune*" o podar) todos los atributos del Mapa de atributos del objeto A que hacen referencias a atributos del objeto B e insertar una única nueva entrada al mapa de A que corresponda al identificador del objeto B, a su vez, marcamos a A como padre de B. De esta forma, no solo reducimos el tamaño del mapa A, sino que cada vez que encontremos un objeto B, sabemos que debería existir un objeto A, padre de B, a determinada distancia (y viceversa).

6.3.7 Implementación de nueva información semántica y herramientas que hagan uso de ellas

Nuevas investigaciones podrían hacerse para refinar o determinar nueva información a detallar dentro del formulario de información semántica presentado en la Sección 3.2.4. A su vez, nuevas herramientas (como la mencionada en la Sección 3.2.10) podrían idearse que actúen en base a esta nueva información para proveer nuevas funcionalidades de accesibilidad.

6.3.8 Mejora en la accesibilidad del *plugin* Interprete y de la plataforma de Crowdsourcing

Como vimos en la Sección 4.2, ciertas características del *plugin* Interprete aún no son del todo accesibles y puede que requieran de asistencia de un usuario sin discapacidad para realizar su correcta configuración. Quedará por trabajar una mejor accesibilidad para que la configuración del *plugin* Interprete y las acciones brindadas por la plataforma de crowdsoucing puedan realizarse más fácilmente para usuarios con discapacidad visual utilizando lectores de pantalla. En especial la funcionalidad de "comandos adicionales", la cual no ha sido muy explotada en el desarrollo de este trabajo y bien podría significar un buen avance para la accesibilidad a muchas acciones tanto del *plugin* como de la navegación web.

6.3.9 Realización de pruebas adicionales

Si bien las partes más importantes del código desarrollado ha sido testeado (mediante la implementación de test de unidad) aún hace falta la realización de otros tipos de pruebas, pero considerando la extensión del trabajo actual y lo que conllevaría la realización de las pruebas faltantes, hemos optado por no desarrollarlas en el marco de este trabajo. Entre estas pruebas se destacan: pruebas con usuarios finales, pruebas con usuarios voluntarios, pruebas para el refinamiento de la nueva información semántica a introducir, pruebas con distintos exploradores y herramientas de lectura de pantalla.

Referencias bibliográficas

ALICANTE, U. ¿Qué es la accesibilidad web? http://accesibilidadweb.dlsi.ua.es/.

ASAKAWA, C. AND TAKAGI, H. 2000. Annotation-based Transcoding for Nonvisual Web Access. *Proceedings of the Fourth International ACM Conference on Assistive Technologies*, 172–179.

BALL, D. 2013. I thought title text improved accessibility, I was wrong. http://silktide.com/i-thought-title-text-improved-accessibility-i-was-wrong.

BUTTLER, D. 2004. A short survey of document structure similarity algorithms. *International Conference on Internet Computing*, 3–9.

CHROME, G. Accessibility (a11y). https://developer.chrome.com/extensions/a11y#controls.

DOAN, A., RAMAKRISHNAN, R., AND HALEVY, A.Y. 2011. Crowdsourcing systems on the World-Wide Web. *Communications of the ACM 54*, 4, 86.

ELR. A specialized Web Browser suitable for touchscreen systems. http://elr.com.au/eiad/.

FOWLER, M. 1999. Refactoring: improving the design of existing code. .

GARRIDO, A., ROSSI, G., AND DISTANTE, D. 2011. Refactoring for usability in web applications. *IEEE Software 28*, 3, 60–67.

GARRIDO, A., ROSSI, G., MEDINA, N.M., GRIGERA, J., AND FIRMENICH, S. 2014. Improving accessibility of Web interfaces: refactoring to the rescue. *Universal Access in the Information Society 13*, 4, 387–399.

INDEC. 2003. *Encuesta Nacional de Personas con Discapacidad*. .

INFOCROWDSOURCING. 2013. ¿Qué es Crowdsourcing? http://www.infocrowdsourcing.com/que-es-el-crowdsourcing-crau-que/.

JACCARD, P. 1908. Nouvelles recherches sur la distribution florale. *Bull Soc. Vaud. Sci. Nat 44*, 223–270.

KING, A. WebbIE. http://www.webbie.org.uk/index.htm.

LESKOVEC, J., ANAND, R., AND ULLMAN, J. 2011. Finding Similar Items. *Mining of Massive Datasets*, 70–128.

MANGIATORDI, A. Farfalla Project. http://www.farfalla-project.org.

MARÍA, M. AND PAZ, L. 2012. " Accesibilidad y Usabilidad : los requisitos para la inclusión digital ." .

NIELSEN, J. 2012. Introduction to Usability. https://www.nngroup.com/articles/usability-101-introduction-to-usability/.

NIERMAN, A. AND JAGADISH, H. V. 2002. Evaluating Structural Similarity in XML Documents. *Complexity*, 61–66.

OMER, B., RUTH, B., AND SHAHAR, G. 2012. A New Frequent Similar Tree Algorithm Motivated by DOM Mining Using RTDM and its new variant – SiSTeR A NEW FREQUENT SIMILAR TREE ALGORITHM MOTIVATED BY DOM MINING Using RTDM and its new variant — SiSTeR. .

OMS. 2014. Ceguera y discapacidad visual. http://www.who.int/mediacentre/factsheets/fs282/es/.

PETER KRANTZ. 2005. Browsing habits of screen reader users. http://www.standards-schmandards.com/2005/browsing-habits/.

SATO, D., KOBAYASHI, M., TAKAGI, H., AND ASAKAWA, C. 2009. What's next? A visual editor for correcting reading order. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 5726 LNCS*, PART 1, 364–377.

SENADO Y CÁMARA DE DIPUTADOS DE LA NACIÓN. 2010. *Ley de accesibilidad de la*

información en las páginas web. Argentina.

TAKAGI, H., KAWANAKA, S., KOBAYASHI, M., ITOH, T., AND ASAKAWA, C. 2008. Social Accessibility: Achieving Accessibility Through Collaborative Metadata Authoring. *Proceedings of the 10th International ACM SIGACCESS Conference on Computers and Accessibility*, 193–200.

THEPACIELLOGROUP. Web Accessibility Toolbar. www.paciellogroup.com/resources/wat/.

TOLEDO, G.A., GONZALEZ, A.H., AND MALBRÁN, C. 2005. Accesibilidad digital para usuarios con limitaciones visuales Resumen Introducción TIC y sociedades de aprendizaje. .

W3C. World Wide Web Consortium. https://www.w3.org/.

W3C. 1997. Web Accessibility Initiative (WAI). https://www.w3.org/WAI/.

WEBAIM. WAVE. http://wave.webaim.org/.

WEBAIM. Simulations. http://webaim.org/simulations/.