Detección de Problemas de Accesibilidad en la Utilización de Lectores de Pantalla en Aplicaciones Web

Tesis presentada para obtener el grado de Magister en Ingeniería de Software

AUTOR

FERNANDO DURGAM

DIRECTORA

DRA. ALEJANDRA GARRIDO

ASESOR CIENTIFICO

DR. JULIAN GRIGERA

La Plata
Mayo de 2020
UNIVERSIDAD NACIONAL DE LA PLATA

Dedicatoria

A Victoria y Miqueas quienes llegaron a mi vida durante el desarrollo del proyecto y son la motivación que me empuja a trabajar por alcanzar metas.

Gracias Hijos.

Agradecimientos

Por la ayuda, la motivación y la paciencia que me brindaron, porque sin la dirección de Alejandra y el asesoramiento de Julián, no hubiese sido posible este proyecto,

No ha sido fácil y Uds fueron mis guías, gracias a su apoyo he logrado superar los momentos complicados.

Resumen

La Accesibilidad es un factor muy importante en las aplicaciones web. Una web accesible ofrece acceso equitativo e igualdad de oportunidades a las personas con discapacidad. Es necesario garantizar el derecho a expresión, opinión, y no discriminación en el acceso a los contenidos de páginas web a todas las personas con capacidades reducidas.

Esta propuesta facilita el reconocimiento automático de problemas de accesibilidad Web y ofrece soluciones a ellos, aplicando una extensión metodológica de Self Refactoring (Grigera, 2017). De esta manera y reutilizando la herramienta Kobold, se implementan procesos de detección de obstáculos en la accesibilidad para usuarios con dificultades visuales que utilizan el screen reader NVDA. Esto es posible definiendo, a partir de las interacciones de los usuarios, accessibility smell (problemas) y accessibility refactorings (soluciones) como transportaciones a las presentaciones web aplicables a esos problemas, cuya implementación tecnológica es una extensión de Kobold que ofrece Accesibilidad como servicio (SaaS – Software as a Service). Gracias a esto no se requiere tener conocimientos específicos sobre accesibilidad, sino que la herramienta se instala fácilmente en los sitios web y a medida que estos son utilizados se reportan automáticamente los problemas detectados.

Pero al utilizar NVDA algunas interacciones se producen sobre representaciones de los documentos web en estructuras internas que facilitan la navegación de los elementos mediante acciones de acceso directo. Para estos casos, fue necesario desarrollar un complemento para NVDA que detecta interacciones virtuales de los usuarios y las reporta para su análisis a un componente de servidor a fin de detectar dificultades que puedan estar presentes en el buffer virtual.

Con el fin de mejorar la precisión en las definiciones de los eventos que dan cuenta de los problemas de los usuarios se desarrolló un sistema y un complemento para NVDA que se utiliza en la estimación de los valores representativos de las acciones de los usuarios, con los cuales llevar adelante experimentos que permitan obtener mejores valores para los parámetros definidos en los algoritmos de detección de Accessibilty Event.

Los problemas de accesibilidad, Accessibility Smell, reconocidos en estudios de casos son catalogados junto con la evaluación de los problemas de accesibilidad presente y cuya detección es posible automáticamente. Así también para las soluciones a esas dificultades denominadas Accessibility Refactorings que se definen como modificaciones a las presentaciones web, alguna de las cuales pueden aplicarse con algún grado de automatización.

Índice Temático

Capítulo 1. Introducción 1
1.1 Problemas de Accesibilidad Web 2
1.2 Motivación 3
1.3 Contribuciones 10
1.4 Organización de la Tesis 11
Capítulo 2. Marco Teórico 12
2.1 Estado del Arte del Tema 12
2.2 Trabajos Relacionados 14
2.3 Software para la Accesibilidad Web 15
Capítulo 3.Extendiendo Kobold para Accesibilidad 18
3.1 Arquitectura de Kobold 18
3.2 Arquitectura Extendida de Kobold para Accesibilidad 21
3.3 Eventos de Accesibilidad 22
3.4 El Proceso de Detección de Accessibility Smells 23
3.5 Aplicación de Refactoring 24
Capítulo 4. Eventos de Interacción y Accesibilidad 25
4.1 Catálogo de Eventos de Accesibilidad 25
4.2 Implementación de la Detección de Eventos de Accesibilidad 25
4.1.1 Eventos Detectables Mediante Incrustación de Componente 27
4.1.2 Eventos de Accesibilidad Asociados a los Accesos Directos y Tareas comunes en Firefox 46
4.1.3 Eventos y Bad Smell en Modo Revisión del NVDA 47
Capítulo 5. Análisis de Valores de Parámetros para los Eventos de Accesibilidad 52
5.1 Características de salida para los usuarios 52
5.2 La tecnología de asistencia 53
5.3 Software para Recolección de Datos Empíricos 57
5.4 Snippet incrustado 58
5.5 NVDA-Accessibility-Parameters 58
5.6 Tratamiento de los Valores Atípicos (outlier).: 59
5.7 Experiencia 60
5.8 Detección de Interacción en la Web 61
Capítulo 6. Complemento NVDA 77
6. 1 Captura de Eventos y NVDA - Modo Navegación de Objetos 77
6.2 Complementos NVDA 77
6.4 Definiciones en el Contexto de NVDA 78
6.5 Scripts y Asociación de Gestos 79
6.6 Modos de Navegación con NVDA 79
6.7 Navegando en Modo Revisión 80
6.8 Buffers Virtuales 81
6.9 Entradas de Usuario de Interés en el Modo Revisión 81

6.10 Captura de Eventos de Interacción en el Modo Revisión....82
6.11 Complemento NVDA Accessibility....83
Capítulo 7- Bad Smells....87
7.1 Definición de Accessibility smells de interacción de usuario....87
7. 2 Consideraciones para la Detección....87
7.3 Catálogo de Bad Smells de Accesibilidad....89
7.3.1 Bad Smell de Accesibilidad Descubiertos....90
7.3.2 Bad Smell de Accesibilidad Asociados a los Accesos Directos y Tareas comunes en Firefox....99
7.3.3. Bad Smell Asociados al Modo Revisión del NVDA....100
7.4 Detección de los Bad Smell....102
7.5 Accessibility Events y los Accessibility Smells....105
Capítulo 8. Refactoring....106
8.1 Definición de Accessibility Refactoring....106
8.2 Consideraciones para la aplicación Accessibility Refactorings en Kobold....106
8.3 Catálogo de Refactoring....108
8.4 Accessibility Events, Accessibility Smells, y Accessibility Refactorings....130
Capítulo 9. Conclusiones....132
9.1 Aportes y Contribuciones....133
9.2 Trabajos Futuros....134
Bibliografía....136

Capítulo I

Introducción

Las formas de publicación de la información han cambiado con la expansión exponencial de la World Wide Web. El inadecuado uso y las limitaciones en los diseños tecnológicos pueden imposibilitar el acceso a la información a los usuarios con capacidades diferentes. Esta situación agrava la infoexlusion (Hassan Montero, Yusef; Martín Fernández, Francisco J, 2003) o brecha digital y supone la discriminación de una gran parte de usuarios.

> *"La usabilidad trata sobre el comportamiento humano; reconoce que el humano es emotivo, no está interesado en poner demasiado esfuerzo en algo, y generalmente prefiere las cosas que son fáciles de hacer contra las que son difíciles de hacer."*
> David McQuillen ” Taking Usability Offline" Darwin Magazine, junio 2003.

Las actividades específicas que hacen a ese comportamiento humano pueden resultar presumiblemente fáciles de realizar y en realidad requerir habilidades y esfuerzos adicionales a las personas con alguna discapacidad.

Esta carga adicional se conoce como barreras de accesibilidad y se deben principalmente a discapacidades que podemos agrupar en:

- Dificultad visual: ceguera, visión reducida y problemas en visualización de color.
- Dificultad auditiva: debido a que el canal sonoro es mucho menos utilizado en interfaces web que el canal visual.
- Dificultad motriz: usuarios que no suelen ser capaces de interactuar con el sistema a través de dispositivos de entrada tradicionales.
- Dificultad cognitiva y del lenguaje: usuarios que presentan problemas en el uso del lenguaje, la lectura, percepción, memoria o salud mental.

Desde la perspectiva de la actividad humana el término accesibilidad resulta amplio, y se vincula con las limitaciones que pueden existir en la relación entre movilidad, comunicación y comprensión (Alonso 2002; Alonso 2003). Lograr que se comprendan y utilicen bienes y servicio requiere de estrategias de diseño que tiendan a lograr la "Accesibilidad Universal”(Alonso López, 2002), para que todas las personas puedan utilizarlas de manera natural y autónoma. Alcanzar un

grado aceptable de accesibilidad Web es posible (Hassan Montero, Yusef; Martín Fernández, Francisco J, 2003). La Real Academia Española (RAE) considera la accesibilidad como “Calidad de ser de fácil acceso“. Para que un producto o servicio Web sea considerado fácil de usar, debe ser universalmente posible para todos, independientemente de las limitaciones del individuo y del contexto.

A menudo las investigaciones en accesibilidad Web consisten en estudios exploratorios que contrastan guías de recomendación estándar en desarrollos tecnológicos alcanzables desde la Web. Analizan el grado de cumplimiento de las recomendaciones en su diseño de forma estática y establecen indicaciones sobre las posibles mejoras necesarias. Aunque un sitio supere las pruebas definidas según las especificaciones estándar, como las WCAGs, no necesariamente será utilizable por todos (Nielsen, J. 2000). La superación de los peritajes técnicos estáticos definidos en las evaluaciones de los diseños no garantizan la accesibilidad. Las personas con discapacidad pueden aún no navegar e interactuar con el contenido y los procesos de una manera cómoda, fácil y efectiva (Garrido, A., Firmenich, 2013). Resulta necesario evaluar la accesibilidad en las interacciones de los usuarios al experimentar el uso de las páginas web, porque podrían existir estrategias de adaptación a un mal diseño que no serían posibles de detectar por un observador del contenido estático (Grigera, 2017).

1.1 Problemas de Accesibilidad Web

La incorporación de nuevos obstáculos en los contenidos Web presentan desafíos para los navegantes con capacidades reducidas de visión. Una alternativa para superar estos problemas de accesibilidad, consiste en utilizar dispositivos tecnológicos de asistencia para los usuarios de computadoras como lectores de pantallas. Sin embargo, la diversidad de ambientes de despliegue y libre edición de los contenidos Web, hacen que estas tecnologías no siempre tengan un funcionamiento esperado.

La accesibilidad web debe considerar los aspectos de contenido y las condiciones de operatividad desatendidos por la usabilidad (Masri, F & Luján-Mora, S., 2010). Esta situación hace complejo el análisis y detección automática de las barreras de accesibilidad.

Por la semejanza entre conceptos, durante el proceso de evaluación muchos investigadores utilizan métodos surgidos del análisis de la usabilidad (Masri, F & Luján-Mora, S., 2010). Pero las inspecciones heurísticas del código no garantizan la plena accesibilidad (Brajnik, G., 2008). Las herramientas automáticas no son una “solución final y completa” (Díaz, B., Cachero,C., 2009).

Algunos investigadores definen la accesibilidad como usabilidad para discapacitados (Diblas, N., et al, 2004). En términos generales es correcto (Masri, F & Luján-Mora, S., 2010)., pero a nivel técnico no implica las mismas condiciones de diseño, desarrollo y evaluación.

A diferencia de la usabilidad, una falla en la accesibilidad impide continuar haciendo uso de la aplicación. Superar las barreras de accesibilidad demanda transformaciones automáticas sobre las presentaciones web que se adapten a los usuarios. La factibilidad de estas modificaciones depende de algunas consideraciones que los desarrolladores pueden hacer en la etapa de diseño web. Las herramientas de evaluación automática controlan el cumplimiento de estándares que facilitan esas transformaciones.

La solución a los problemas de accesibilidad en aplicaciones web busca incorporar pautas, en los navegadores y tecnologías de asistencia, intentando mitigar las consecuencias de no observar buenas prácticas y estándares para accesibilidad.

1.2 Motivación

El esfuerzo invertido en buscar soluciones a las dificultades de accesibilidad redunda en beneficio de toda la comunidad de navegantes. Las dificultades de acceso están presentes en el diseño desde los inicios de la web y aún continúan siendo una demanda insatisfecha por la industria del software.

> *"El poder de la Web está en su universalidad. Un acceso a la Web para todos independientemente de su discapacidad, es un aspecto esencial."* Tim Berners-Lee, Director del W3C e inventor de la World Wide Web, Washington DC, EE. UU., 22 de octubre de 1997.

Una página web accesibles permite que una mayor cantidad de usuarios, independientemente de sus capacidades físicas, intelectuales o tecnológicas, descubran sus contenidos textuales, gráficos y multimedia. Pensar en accesibilidad conlleva una reflexión sobre la inclusión y sobre quienes tienen dificultades visuales, auditivas, motrices o problemas de aprendizaje, e incluso de quienes no disponen de parlantes, tiene una baja calidad de conexión y utilizan dispositivos móviles.

El 20 de diciembre de 1993 con el fin de garantizar que por su calidad de miembros de la sociedad todas las personas con discapacidad tuvieran los mismos derechos y obligaciones que los demás, se aprobaron en Naciones Unidas las "Normas Uniformes sobre la igualdad de oportunidades para las personas con discapacidad". Estas son medidas tendentes a que los estados reconozcan la importancia global de posibilitar el acceso de todos a la tecnología y así lograr la igualad de oportunidades en todas las esferas de la sociedad.

Para democratizar el acceso a la información se necesita dotar de accesibilidad a los portales web favoreciendo la inclusión de grupos con discapacidades. Esto potencia el teletrabajo, mejora la velocidad de navegación y facilita el acceso independiente de los dispositivos que se utilicen.

Considero que las temáticas de accesibilidad web deben ser puesta en relieve por parte de la comunidad científica. Además de las razones éticas y solidarias, buscar soluciones para quienes atraviesan diariamente dificultades promueve el bienestar general y la accesibilidad permite la participación de un público más amplio en la búsqueda de las metas sociales.

Existen razones económicas que movilizan a los desarrolladores a incluir pautas de accesibilidad. En el comercio electrónico, potenciales clientes pueden verse afectados por algún obstáculo que les impida adquirir algún producto o servicios. También existen razones técnicas, porque cada vez más dispositivos salen al mercado con características diversas. Entre ellos teléfonos, kioscos interactivos y televisores con sistemas de navegación que los sitios accesibles adaptaran más fácilmente.

> *Es sabido que el mundo real y virtual presenta infinidad de similitudes y diferencias. Es sabido también que cuando no vivimos personalmente alguna situación limitante, nos cuesta ponernos en el lugar de otras personas que si conviven a diario con ellas. No todos los interesados cuentan con las mismas posibilidades a la hora de pretender acceder a un sitio o portal de Internet.*
>
> "Acceso sin barreras...por Humberto Demarco, 2009".

Existes diversas formas de acceder a los contenidos disponibles en la World Wide Web. Nos centraremos en la más tradicional en la que se utiliza un software navegador, particularmente Mozilla Firefox ejecutándolo sobre algún dispositivo de hardware local de propósito general. Los sitios web HTML (HyperText Markup Language) ofrecen una interfaz que incluye elementos definidos por código dentro de una jerarquía de página que contienen textos, imágenes, vídeos, juegos, etc. cuyo estándar esta a cargo del World Wide Web Consortium (W3C).

La población de usuarios a considerar en este estudio está constituida por personas que tiene algún grado de dificultad visual y necesitan utilizar tecnología de asistencia por software para leer la pantalla en el navegador (screen reader). Estos programas identifican e interpretan lo que se presenta en la pantalla de una computadora, transmitiendo al usuario su contenido mediante síntesis de voz, estímulos sonoros o salidas braille.

Desarrollado por NV Access, NonVisual Desktop Access (NVDA) es el lector de pantalla para Sistemas Operativos Windows que utilizaremos en este trabajo. Es una herramienta de código abierto, con salida de voz y braille, que posibilita a las personas acceder a los contenidos Web sin necesidad de disponer de contacto visual. Además de su distribución gratuita, cuenta con una amplia comunidad de colaboradores activos y se ha descargado más de 70.000 veces en 43 idiomas.

En el paradigma de programación dirigido por eventos, el flujo de ejecución del programa es determinado por sucesos que ocurren en el sistema y permite al usuario interactuar en

cualquier momento. Uno de los lenguajes de programación por eventos y el más utilizado en la web es JavaScript. Se ejecuta del lado del cliente (client-side) y se implementa directamente en los navegadores web para mejorar la interfaz del usuario. A través de la API Document Object Model (DOM o Modelo de Objetos del Documento) los programas como Mozilla Firefox modifican el contenido y la presentación del documento HTML e invocan funciones incluidas dentro de la página. Cuando se produce un evento de interacción son invocados los manejadores de eventos ligados a cada elemento HTML, que ejecutan tareas definidas por los desarrolladores.

En definitiva cualquier acción posible sobre la presentación web tiene asociado un evento al cual el navegador responde ejecutando algún proceso.

Actualmente existen innumerables dificultades de accesibilidad presentes en las páginas web. En el sistema de Educación Universitario Argentino sus páginas institucionales no cumplen los criterios de conformidad de nivel A de las WCAG y por ello no cumplen con el nivel AA exigido por la legislación Argentina (María Inés Laitano, 2015)

Ejemplos de sitios de acceso masivo son:

Municipalidad de Rosario de Santa Fe	
Contiene imágenes sin texto alternativo	 *Figura 1: Web Page Municipalidad de Rosario*
Plataforma Edu.ar	
Vídeos educativos y contenidos federales sin subtítulos.	 *Figura 2: Web Page Edu.ar*
Nuevo Banco de Santa Fe	
Página principal que no cumple con las pautas de accesibilidad: insuficiente contenido no textual, no se evita el uso de bloques para la accesibilidad por teclado y se combinan colores con bajo radio de contraste.	 *Figura 3: Web Banco de Santa Fe*

Tabla 1: Web Page que incumplen los criterios de conformidad de WCAG.

Ministerio de Salud del Gobierno de Santa Cruz	
La página de contactos es inaccesible por no contener identificación de errores, etiquetas o instrucciones suficientes, sin sugerencias ante errores y falta de contenido no textual.	 *Figura 4: Web Page Ministerio de Salud de Santa Cruz*
Página web del Banco Bica	
No cumple los criterios de nivel A de accesibilidad, con estilos en línea o maquetado con tablas y colores con bajo radio de contraste.	 *Figura 5: Web Page Banco Bica*

Tabla 1: Web Page que incumplen los criterios de conformidad de WCAG.

Dificultades concretas de accesibilidad web podemos encontrarlas en el sitio principal de amazon (www.amazon.com). Examinando las pautas de accesibilidad WCAG 2.0 con la herramienta Online TAW, se obtuvieron los siguientes resultados

Fecha: 29/05/2019 19:06 Pautas WCAG 2.0 Nivel del análisis: AA Tecnologías: HTML, CSS

Categoría	Reportes	Cantidad	Criterios Afectados	Principios Afectado	Dificultades Detectadas
Problemas	Son necesarias correcciones	77	5	Perceptibles	22
				Operable	25
				Robusto	30
Advertencias	Es necesario revisar manualmente	246	12	Perceptibles	182
				Operable	36
				Comprensible	18
				Robusto	10
No verificados	Comprobación completamente manual	17	17	Perceptibles	4
				Operable	8
				Comprensible	5

Tabla 2: Reporte de dificultades de accesibilidad Online TAW

Según estos reportes, con relación a la directriz de perceptibilidad, pauta 1.1 Textos alternativos, no se cumple con el criterio 1.1.1 Contenido no textual. Se informan 18 problemas que indican que en la página principal de Amazon existen elementos no textuales que carecen de texto alternativo que NVDA no podrá reproducir.

Analizando manualmente la misma página con el navegador Mozilla Firefox y utilizando NVDA, se pueden visualizar ofertas que son agrupados en listas HTML. Los productos se presentan dentro de un carrusel deslizante responsivo, donde para cada elemento se dispone de una vista previa. Se accede a este recurso haciendo click sobre un botón, que se visualiza al posicionar el puntero sobre el producto. Esta funcionalidad no es accesible a todos los usuarios, dado que quienes tiene dificultades visuales y utilizan el teclado, no hacen uso del mouse. Sin embargo, aun cuando una combinación de acciones de teclado permitiera posicionar el foco en el botón, este elemento no dispone de texto electrónico, motivo por el cual NVDA reproduce el texto “En blanco”.

Figura 6: Lista de Productos incluidos en un carrusel deslizante.

Un caso similar ocurre con el texto electrónico poco descriptivo que se sintetiza al posicionar el foco en el cuadro de búsquedas de contenidos de la página. Aquí se reproduce el audio del texto electrónico “Buscar edición”, correspondiente al texto de la etiqueta Label asociada al campo de entrada.

Figura 7: Cuadro de búsquedas de contenidos en la Web Page de Amazon.

Idéntica situación se produce con el botón que exhibe la típica imagen de una lupa, presente en la figura 7, del cual sé esperar que NVDA indique que procesará la búsqueda solicitada. Sin embargo, al posicionar el foco sobre este botón se sintetiza “ir botón” como resultado del sintetizar el texto del campo valué del botón, que contiene el texto “ir” y el tipo de elemento reconocido por NVDA.

Los vínculos tampoco están exentos de problemas cuando se trata de guiar al usuario, se observa en la presentación un enlace que facilita cambiar el idioma en la interfaz, pero al posicionar el foco, se sintetiza “es enlace”, debido a que el elemento contiene un tag de tipo Span con texto “Es” y luego NVDA indica el tipo de elemento accedido.

Figura 8: Web Page de Amazon conteniendo enlaces con dificultades de acceso para NVDA.

Estos son claros ejemplos de textos electrónicos que resultan insuficientes para los usuarios con dificultades visuales que utilizan NVDA, porque no les proporcionan adecuada información del contexto sobre el cual están operando y pueden entorpecer el uso de la web.

Como navegante habitual encuentro frecuentemente diseños que pueden obstaculizar el acceso a las personas con dificultades visuales que necesitan utilizar teclados y lectores de pantalla. De la interacción con estudiantes de educación media que participan en las olimpiadas especiales y allegados con movilidad reducida, comprendí los contratiempos que deben atravesar a diario y como las pequeñas acciones pueden favorecer a su integración y mejorar su calidad de vida.

Esta experiencia es una oportunidad para ofrecer respuesta a la necesidad de los desarrolladores, aplicando un enfoque alternativo basado en la información sobre el uso de las aplicaciones Web. Este conocimiento pueda ser utilizado para el desarrollo de herramientas de evaluación de accesibilidad, además constituye un aporte por velar por los derechos de toda persona y alcanzar una condición de igualdad en el acceso a la sociedad de la información.

Motivado por el creciente interés por la accesibilidad de la sociedad y de la comunidad científica, plasmado en las investigaciones de los últimos años he llevado adelante la propuesta reflejando la problemática con el anhelo de lograr una herramienta, que mejore los procesos de ingeniería y ayude a mitigar los efectos negativos sobre la accesibilidad de los inadecuados diseños web.

El mecanismo elegido fue enriquecer las captura y análisis de los eventos de interacción (Grigera, 2017), reportando problemas concretos mediante la agregación de **Eventos de Interacción.** Se pueden definir unidades de nivel superior denominada **Eventos de Accesibilidad** que consisten en una secuencia ordenada de eventos de interacción que se producen sobre determinados elementos de la interfaz web o un conjunto de URL.

Capturando **Eventos de Interacción** y analizando **Eventos de Accesibilidad** es factible buscar patrones que permitan detectar dificultades y sugerir posibles soluciones. Emulando el enfoque utilizado en Usability Self Refactoring y adaptando nuevamente los términos de la jerga del refactoring de código (Grigera, 2017), podemos denominar a los problemas de accesibilidad **"Accessibility Smells"** y sus posibles soluciones **"Accessibility Refactorings"**. Extendiendo el marco de desarrollo y aplicando una estrategia similar es posible detectar automáticamente obstáculos a la accesibilidad, a partir de los **eventos de interacción** de los usuarios finales que utilizan teclados y lectores de pantalla.

1.3 Contribuciones

Este trabajo tiene la finalidad de diagnosticar y proponer mejoras a los problemas de accesibilidad web que surgen en la interacción con el teclado de usuarios con dificultades visuales que utilizan lectores de pantalla. Fue necesario estudiar la factibilidad de que una herramienta orientada a detectar problemas de usabilidad, fuera extensible para la detección automática de barreras de accesibilidad.

Partiendo de la arquitectura reutilizable de la plataforma Kobold (Grigera, J, Garrido, A, Rossi, G, 2017), se logró construir una herramienta simple de configurar y de acceso web, que consiste en:

1. Una extensión a la estrategia implementada en Kobold: Para procesar interacciones que incluya eventos de accesibilidad.
2. Un catálogo de **accessibility smells:** Con dificultades visuales que puedan detectarse automáticamente al utilizar el teclado con un lector de pantalla. Un conjunto de esos smells son específicos para **NVDA**.
3. Un catálogo de **accesibility refactorings:** Que aplican transformaciones a la interfaz como solución a la presencia de un **accesibility smells**.

4. Una extensión al algoritmo de Kobold: Para detectar la similitud entre elementos de las páginas web afectados por el mismo **accesibility smell.**

5. Un conjunto de casos reales que contienen los **accesibility smells** catalogados en este trabajo: En los que se describe la dificultad y como afectan a los usuarios.

6. Un banco de prueba para simular la interacción en la que se presentan los smells.

7. Una extensión para el lector de pantalla **NVDA** con arquitectura reutilizable: Que permite capturar los eventos asociados con pulsaciones de teclas y que controlan la navegación en **NVDA**.

8. Una herramienta que extiende Kobold para automatizar la "Accesibilidad como Servicio": Detectando los **accesibility smells** y proponiendo soluciones que se pueden aplicar semi-automáticamente como **accessibility refactorings.**

1.4 Organización de la Tesis

A continuación en el Capítulo 2 se expone el marco teórico de la investigación, junto con algunas herramientas de accesibilidad y se caracterizan las dificultades tratadas durante el resto del trabajo.

En el Capítulo 3 se analiza la arquitectura de **Kobold** y la forma en que integra sus componentes. Se describe la implementación del proceso de detección con el objeto de extenderlo a los fines de la accesibilidad.

En el Capítulo 4 se detalla un catálogo con captura de eventos para detectar problemas de accesibilidad. Incluye estudio de caso e implementación en el componente del lado del cliente.

En el Capítulo 5 se describe el sistema desarrollado para estimar parámetros de los algoritmos de detección y las tareas llevadas a cabo para la recolección de datos.

En el Capítulo 6 se abordan las consideraciones a tener en cuenta durante la programación de complementos para **NVDA** y los desarrollos requeridos para acceder a los eventos en el **buffer virtual**.

En el Capítulo 7 se detalla un catálogo de dificultades de accesibilidad explicando los aspectos técnicos para el reconocimiento de cada problema junto con un estudio de caso.

En el Capítulo 8 se describe la aplicación de soluciones en términos de **Accessibillity Refactorings,** incluyendo un catálogo con estudio de caso.

Para finalizar, el Capítulo 9 contiene las conclusiones, los aportes realizados y algunos trabajos futuros que han quedado abiertos a partir de esta investigación.

Capítulo II

Marco Teórico

2.1 Estado del Arte del Tema

En Argentina la Ley 26.653 exige que las páginas web acaten las normas sobre **accesibilidad** facilitando el acceso a sus contenidos para todos los usuarios con discapacidad.

Las limitaciones propias de cada individuo incluyen deficiencias visuales, auditivas, motrices, cognitivas, del lenguaje, derivadas del contexto o del dispositivo de acceso (hardware y/ o software). La **accesibilidad universal** es difícil alcanzar y pueden quedar excluidos grupos de usuarios con algún tipo de discapacidad (Garrido et al., 2014).

En particular, las personas con dificultades visuales (UDV) enfrentan obstáculos de accesibilidad Web, asociados a las siguientes categorías:

- Percepción del Usuario: Por la necesidad de percibir gráficos, diseños o señales visuales.
- Navegabilidad de las Aplicaciones: Por la dependencia del teclado para acceder a las funcionalidades.
- Comprensibilidad del Usuario: Al ser necesario comprender el orden lógico lineal que se le presenta.
- Robustez de las Aplicaciones: Disponibilidad de ayudas para no videntes, que no pueden ser accedidas por toda la gama de tecnologías modernas.

Estudios a gran escala sobre calidad de accesibilidad web (López, Gómez, et al., 2010), muestran que las páginas menos complejas tienden a lograr mejores niveles de accesibilidad. Pero estos casos son minoritarios y existe una tendencia al desarrollo web cada vez más orientado a lo visual.

En este contexto, los UDV han optado por utilizar herramientas de interacción acordes a sus capacidades, como lectores de pantallas, reconocimiento de voz, dispositivos apuntadores, teclados alternativos y pantallas Braille (Santana Mansilla et all., 2015). Sin embargo, las dificultades siguen allí y todas las adaptaciones no han resuelto los problemas de accesibilidad.

Las condiciones de acceso están relacionadas con las características de **usabilidad**. Un software usable es un producto que puede ser utilizado por los usuarios para la consecución de

sus metas específicas de forma efectiva, eficaz y satisfactoria. El mismo producto es accesible cuando su uso resulta posible independiente de las limitaciones del individuo y de su contexto.

La **accesibilidad social** fue un intento por reducir el tiempo para generar contenidos accesibles (Takashi Itoh, Kawanaka, et al., 2008). Buscaba que los usuarios utilizando una plataforma colaborativa, generen reportes de las dificultades encontradas. Estos reportes debían ser revisados por voluntarios, que proponían soluciones agregando metadatos a las páginas con técnica de **transcodificación**. Sin embargo, el enfoque resulto limitado en cuanto a los problemas que se pueden resolver.

En el caso de las Aplicaciones Web Enriquecidas (RIA), que cambian previamente su presentación antes de desplegarse (Fernandez, Batistam et al., 2013), los desencadenamientos producidos por cada evento del usuario, hacen que las evaluaciones de accesibilidad varíen considerablemente. Una evaluación completa resulta imposible, porque requiere procesar automáticamente todo el grafo de estados, que representa las posibles interacciones. Una evaluación tradicional resulta insuficiente, por la imposibilidad de que los evaluadores automáticos accedan al lenguaje de script subyacente y al contenido dinámico. Estudios comparativos a gran escala, con más de 8000 RIAs, consideran las propiedades de accesibilidad desde tres perspectivas:

1) Evaluación previa al procesamiento del navegador (enfoque tradicional).

2) Evaluación después del procesamiento (carga dinámica).

3) Lo que ocurre después del procesamiento del navegador (considerando el desencadenamiento de los eventos).

Se observa que las evaluaciones, después de procesamiento del navegador proporcionan resultados más precisos y profundos de la accesibilidad de una página; pero a la vez incompletos, debido a que pasan por alto los estados por los que atraviesan las RIAs, ignorando numerosos problemas de accesibilidad.

Con el objeto de facilitar el acceso a las personas con discapacidad, se han elaborado recomendaciones para mejorar las herramientas de evaluación y reparación de accesibilidad. Son pautas de Accesibilidad al Contenido de la Web (WCAG) que las entidades gubernamentales, alrededor de todo el mundo, infructuosamente han tratado de imponer reglando el marco normativo para la producción de aplicaciones (Theofanos, Redish 2006).

A partir de las WCAG, se ha desarrollado las directrices del Gobierno de EEUU http://www.section508.gov y las herramientas tecnológicas disponibles para evaluar e identificar fallas de accesibilidad como Bobby, RAMP, InFocus y A-Prompt (Ivory, Mankoff, 2003), W3C Markup Validation Service, TAW3 desarrollado por el Centro Tecnológico de la Información y la

Comunicación, HiSoftware Cynthia Says y aDesigner(Serrano Mascaraque, 2009). Sin embargo, no todos los criterios de WCAG puede ser automatizados y requieren la comprobación manual (ej: Lo descriptivo de un texto alternativo dependerá de su contexto).

Por este motivo, son necesarios métodos empíricos y pruebas de usuario a partir de la captura y análisis de eventos en la interfaz (Rubin, J., Chisnell, 2008). Estos eventos, que se pueden capturar automáticamente, son indicadores del comportamiento del usuario con respecto a la interfaz y son una fuente potencialmente de información sobre el uso de la aplicación.

A partir de esos comportamientos, es posible reconocer las necesidades de reestructuración de código para mejorar las características externas de usabilidad, accesibilidad y resguardar ese conocimiento (Garrido, Rossi, et al., 2011).

2.2 Trabajos Relacionados

Existen trabajos previos que proponen mejorar de forma incremental la accesibilidad web utilizando refactoring (Garrido, Rossi, et al., 2013). Son modificaciones de las características perceptibles de una aplicación web asociadas a las dificultades de visión. Se han catalogado algunos de los problemas de accesibilidad, “bad smells” (tomando el término de la jerga de refactoring), y algunas alternativas de respuestas de refactoring de interfaz web (WIR)(Garrido, Rossi, et al., 2013). La aplicación de patrones a las interfaces (WIR), facilita las transformaciones presentando un componente más accesible sobre el elemento afectado. Estas soluciones son posibles gracias a la potencia del Modelo de diseño de Hypermedia Orientado a Objeto.

Un refactoring en la interfaz (Garrido, A., Firmenich, 2013) genera vistas alternativas personalizadas y adaptadas en presentación de la página, como repuesta a la presencia de algún bad smells y del feedback de los usuarios.

Un importante aporte para que los equipos de desarrollo sería un inventario de problemas y soluciones de accesibilidad exhaustivo, enriquecido por los usuarios y orientado a las modificaciones del software.

Usability Smells Finder (USF) es una herramienta que posibilidad cotejar y mejorar las páginas buscando bad smells en las experiencias de los usuarios. Analizando eventos de interacción (UI) descubre bad smell de usabilidad y reportando posibles soluciones automáticamente (Grigera, Garrido, 2014).

2.3 Software para la Accesibilidad Web

Los software de asistencia para la revisión de la accesibilidad asisten, con algún grado de automatización, a los usuarios y desarrolladores para evaluar la presencia de barreras en un sitio web.

Entre las más populares disponibles por Internet se encuentra Test Accessibilidad Web (TAW); ideada y desarrollada por la Unidad de Accesibilidad Web de la Fundación CTIC (Centro Tecnológico de la Información y la Comunicación). Son un conjunto de herramientas de análisis automático y semi-automático que verifican el grado de adecuación de un sitio web a las pautas de accesibilidad **W3C WCAG 2.0.** Aunque requieren complementarse con la participación de expertos en accesibilidad permiten analizar integralmente los elementos y las páginas web detectando problemas en los diseños del sitio.

HERA es un desarrollado del Seminario Iberoamericano sobre Discapacidad y Accesibilidad en la Red (SIDAR) escrito en lenguaje PHP con tecnología HTML y CSS . Permite a desarrolladores y diseñadores la revisión de la versión 1.0 de las W3C WCAG. HERA Xp es una versión simplificada útil durante el proceso de creación de un sitio y puede ser utilizada para revisiones puntuales. Ambas versiones ofrecen información sobre la evaluación de puntos de control de accesibilidad, necesidades de corrección y usuarios afectados. Una tercera alternativa es HERA FFX, que funciona como una extensión de HERA para Firefox basada en una versión de Hera alojada en un servidor externo. Esta extensión está desarrollada en lenguaje JavaScript, XUL, XML, CSS y se utiliza para automatizar las tareas de análisis manual preliminares de una página web. No fue diseñado para pruebas concretas y debe su dinamismo al formato XML, en el que se resguardan las comprobaciones externamente.

En términos prácticos la accesibilidad y la usabilidad son conceptos discutidos por separado. Existen autores (Leporini, Paternó 2003) que sostienen la existencia de una relación estrecha entre ambos. Consideran que se puede aplicar criterios de usabilidad para mejorar la navegabilidad de los usuarios con dificultades visuales. Entienden que un sitio es accesible si puede ser utilizado por todos y principalmente por aquellas personas con algún tipo de discapacidad. Para los autores, la usabilidad es multidimensional, donde la importancia de cada aspecto depende del dominio de la aplicación y su efecto sobre la navegabilidad por parte de usuarios con necesidades especiales. Tienen en cuenta el contexto del usuario (lectores de pantalla, lupa de pantalla) aludiendo que "*La accesibilidad técnica es una condición previa para la usabilidad*". Argumentan que las metas de accesibilidad reales se vinculan a las necesidades del usuario y no solo al cumplimiento de estándares preestablecidos. Para ellos, el software de evaluación WebSAT (Web Static Analyzer) no proporciona suficiente información a los desarrolladores sobre las dificultades encontradas. Hacen referencia a las herramientas basadas en criterios y directrices, como Bobby y LIFT, argumentando que generan reportes demasiados

extensos, difíciles de comprender y sobre un número limitado de dificultades. En ese marco, con el objeto de ofrecer una guía compacta a usuarios y desarrolladores, partiendo de un trabajo empírico han identificado criterios de eficacia, eficiencia y satisfacción que asocian a soluciones técnicas en términos de puntos de control de usabilidad y accesibilidad.

Al desarrollar Multi-Analysis of Guidelines by an ENhanced Tool for Accessibility (MAGENTA)(Leporini B., Paternò F., Scorcia A.,2006) sus precursores sostenían la inexistencia de una herramienta integrada con soporte para múltiples guías de accesibilidad. Los informes resultaban insuficientes para los desarrolladores y no ofrecían opciones de reparación automática. Con MAGENTA entran en escena formas más flexibles de respaldar las evaluaciones. Los métodos de inspecciones y las pautas de usabilidad ofrecen soporte a múltiples conjuntos de directrices, corrección de código e informes. Para aplicar de forma fácil y consistente criterios de evaluación, utiliza un lenguaje de abstracción de guía (GAL), donde las pautas se especifican en un esquema XML que se resguarda independiente y externamente a la aplicación para su posterior interpretación automáticamente. Esta implementación separada de la definición de pautas fomenta la reutilización facilitando la definición e incorporaron de nuevas pautas y conjuntos de directrices.

Con el paso del tiempo la mayoría de las herramientas desarrolladas para evaluar la accesibilidad queda obsoleta por falta de adecuación a la evolución de los estándares, a los cambios de la legislación y las normas sobre accesibilidad y tecnologías de la información (Schiavone, A.G. & Paternò, F. Univ Access Inf Soc 2015). Los autores distinguen las herramientas que mantienen la definición de directrices formales externamente y separadas de la lógica de la evaluación. La mayoría contiene su definición internamente e integrada a la implementación por lo que son más difíciles de actualizar. En estos casos una nueva directriz requiere de la reescritura del código de la aplicación. Tal es el caso de Achecker, desarrollado por la Universidad de Toronto, que permite la validación de documentos HTML con directrices BITV, la Ley Stanca, WCAG 1.0 y WCAG 2.0, Test Accessibilidad Web (TAW) y mobileOK promovido por el W3C para usabilidad e interoperabilidad de las páginas web en dispositivos móviles.

En MAUVE (Multiguideline Accessibility and Usability Validation Environment) (Schiavone, A., Paternó 2015) se ofrecen un entorno de evaluación para accesibilidad y usabilidad web que permite la validación de pautas con los estándares recientes (como HTML5 y CSS 3) y admite versiones para una variada diversidad de dispositivos (como tabletas, teléfonos inteligentes, consolas y Televisores inteligentes). En estos casos se especifican y actualizan pautas sin necesidad de cambios en la implementación. Las directrices web se definen en lenguaje XML y permite verificar tanto HTML como CSS para detectar problemas de accesibilidad. Tiene soporte para sitios dinámicos a través de complementos para los navegadores más populares. Su estrategia se basa en reconocer automáticamente para cada guía puntos de control

asociados a sus pautas y corroborar que cada construcción y atributo proporcione la información necesaria. Cada criterio de evaluación definido en XML se analiza en el DOM (Modelo de objeto de documento) determinando si se debe corregir a través de un soporte semiautomático. La informatización de las pautas se especifica en un lenguaje de alto nivel denominado Lenguaje para la Definición de Directrices Web (LWGD) y los resultados se reportan en archivos XML que puede almacenarse como bibliotecas.

Con WaaT evaluator se propuso un enfoque de base de datos de ontologías con un motor de inferencia que utiliza reglas para extraer definiciones de directriz de accesibilidad (Fernandes, N y otros 2014). La descripción de reglas y las consultas a la base de datos se pueden realizan a través de SWRL y SPARQL. Las directrices se expresan en lenguaje de procedimiento en lugar de declarativo.

El consorcio World Wide Web (W3C) ha catalogado las herramientas de evaluación de accesibilidad web[1], de forma que se puedan obtener la que mejor se ajuste a las necesidades del usuario y de los desarrolladores. Este catálogo cuenta con herramientas que soportan directrices WCAG. BITV (Estándar del gobierno alemán), RGAA (Norma del gobierno francés), JIS (estándar japonés de la industria), Sección 508 (norma federal de adquisiciones de EE. UU.), Ley Stanca (Legislación Italiana de Accesibilidad), EPUB (El formato se diseñó redimensionable para adaptarse a distintos tamaños de letra y pantalla), MAAG 1.0 (Estándar del Gobierno de Corea), pautas nacionales irlandesas de accesibilidad de TI. y cubre tecnologías como WAI-ARIA, CSS, XHTML, PDF, SVG, MLILL, Texto, vídeo, entre otras.

Pocos textos académicos y científicos se refieren a la detección de barreras implícitas en la experiencia del usuario con dificultades visuales que utiliza la web. A pesar del gran número de aplicaciones disponibles, la amplitud de las tecnologías y los esfuerzos por cubrir más barreras de accesibilidad, se ha relegado el tratamiento de los problemas que no pueden ser detectados en los diseños y/o análisis estáticos.

Estas dificultades de accesibilidad que aparecen con el uso y que se manifiestan en las instancias de interacción entre el usuario y la interfaz son las que trataremos a lo largo de este trabajo.

1- Catálogo de Herramientas de Accesibilidad W3C https://www.w3.org/WAI/ER/tools/index.html

Capítulo III

Extendiendo Kobold para Accesibilidad

En este capítulo se describe la arquitectura de la herramienta de base utilizada (Kobold), su funcionamiento para detectar problemas de usabilidad y la extensión realizada para capturar problemas de accesibilidad a partir de eventos de interacción.

3.1 Arquitectura de Kobold

El software **Kobold** detecta automáticamente dificultades de usabilidad en aplicaciones web reportando sus posibles soluciones basadas en acciones que los usuarios realizan sobre la interfaz (Grigera et al. 2017). Su autor adapta términos del **refactoring** de código definiendo problemas de usabilidad como **usability smells** a partir de los cuales puede aplicar soluciones que denomina **usability refactorings**.

La herramienta ofrece “Usabilidad como Servicio”, caso específico de Software como Servicio” o SaaS (SaaS – Software as a Service). A medida que se utiliza el sistema se escanean las interacciones de los usuarios en las aplicaciones buscando problemas de usabilidad. Cuando alguna dificultad es detectada y se supera el valor definido para su umbral de tolerancia, se genera un reporte acompañado de recomendaciones en formas de **refactorings** que se pueden aplicar.

El acceso al sistema requiere que los usuarios se registren en una cuenta y generen un código **snippet** que debe ser incorporado en la aplicación objeto del análisis.

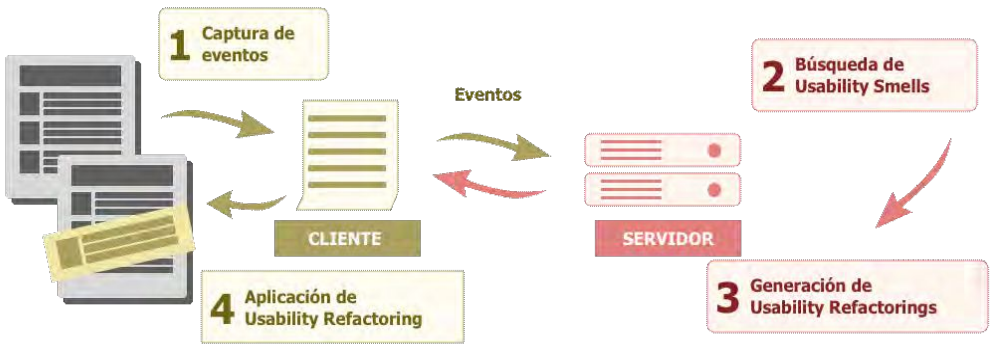


Figura 9: Arquitectura de Kobold. Tomado de Self Refactoring (p. 37), por J. Grigera

En el proceso intervienen dos elementos principales, un componente del lado del cliente y otro del lado del servidor. El primero se incrusta en la aplicación para capturar los eventos de interacción y aplicar las refactorizaciones sobre la interfaz. El segundo, procesa los eventos capturados por cliente, detecta los **usability smells** y reporta sugerencias de **refactorings** algunas de la cuales pueden aplicarse automáticamente.

Al aplicar una solución automáticamente se crea una instancia del **usability refactoring** en el servidor, conteniendo código JavaScript que es invocado por el componente cliente para modificar la interfaz ante la presencia de algún elemento afectado por un **usability smell**.

Mediante la cuenta de usuario se puede acceder a los resultados de las capturas de eventos. La incorporación del snippet de código en la aplicación que se desea analizar proveer el análisis de usabilidad en forma de servicio, simplificando su adopción para los usuarios con la intención de reducir posibles costos de análisis y reparación de problemas de usabilidad.

La estrategia utilizada en **Kobold** consiste en combinar componentes basados en el cliente, dentro del navegador web, con componentes en un servidor aplicando un análisis de tres pasos: Captura de Eventos, Detección de **Usability Smells** y **Refactoring**.

La captura sucede en el cliente, mientras que el análisis, en el servidor. Es el componente del lado del cliente quien evalúa las interacciones de los usuarios, filtrándolas y agrupándolas en eventos más abstractos llamados **eventos de usabilidad.** El componente del lado del servidor clasifica y analiza esos **eventos de usabilidad** para descubrir problemas de usabilidad como **usability smells**.

Para cada dificultad detectada pueden existir diferentes soluciones según sea posible aplicar uno o más **refactorings** que resuelven el smell. Si la solución es aplicable automáticamente se genera el código necesario en el componente del servidor y se inserta en la aplicación web utilizando el componente del cliente.

De esta manera, se compone un framework con una arquitectura extensible para el manejo de eventos de interacción y de elementos en el DOM, que facilita incorporar nuevos eventos de usabilidad del lado del cliente.

Los pasos siguientes son implementados del lado del servidor en Pharo Smalltalk 9 donde existe un mecanismo reutilizable que simplifica la tarea de definición de nuevos **usability smells**, sus buscadores y sus instancias de **refactoring** para generar código JavaScript.

Kobold proveer reportes en tiempo real de lo que ocurre en la aplicación recolectando información relativa al contexto del usuario cuando suceden los eventos de usabilidad. Realiza un análisis preliminar, filtrado y agregando eventos potenciales de interacción útiles para detectar **usability smells.**

Los eventos de usabilidad fueron diseñan luego de observar y estudiar los comportamientos que pueden indicar la presencia de **usability smell.** Esos comportamientos fueron desagregados en acciones más atómicas que pueden ser capturadas durante las sesiones del usuario. Los eventos de usabilidad se analizan en el servidor determinando si el mismo evento sobre el mismo conjunto de elementos alcanzan un nivel de reporte suficientes que indique la presencia del **usability smell.**

Existen algoritmos implementado en entidades llamadas **usability smell finders** (buscadores de usability smells) que son los encargados de detectar un único tipo de **usability smell.** Para cada aplicación bajo análisis se genera un conjunto de buscadores que consumen y analizan uno o más tipos de eventos de usabilidad. Se configuran con ciertos parámetros hallados mediante experimentación y definen el número, proporción, o combinación de eventos de usabilidad que desencadenan la presencia de un smell específico en cada buscador.

Para obtener resultados instantáneos sin depender de la cantidad de **usability events** acumulados, los buscadores procesan los eventos en tiempo real reevaluándolos para detectar potenciales nuevos **usability smells**. En cada caso, los clasifican según el elemento que afectan y extraen información a modo de síntesis para evitar re-procesarlos. Con estos datos se reevalúa la presencia de los smell sobre cada elemento afectado y se agregan los nuevos smells al reporte como se muestra en la figura 10.

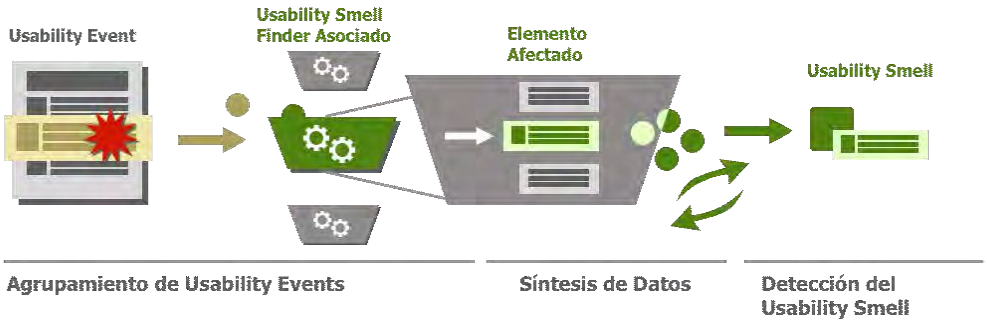


Figura 10. Proceso de detección de Usability Smell. Tomado de Self Refactoring(p. 43), por J. Grigera .

Los círculos representan **usability events** que viajan desde el cliente al componente del servidor, donde son clasificados y almacenados para detectar **usability smells**.

Los elementos buscadores pueden extenderse simplemente definiendo el tipo de **usability event** a tratar, su criterio de agrupamiento y la lógica de detección. Esto es posible dentro de una arquitectura preparada para facilitar la incorporación de nuevos buscadores más específicos de cada nueva clase de **usability smell**.

Para conectar de manera sencilla los problemas, **usability smells,** con las soluciones, **Usability refactorings,** el autor aplica **refactorings de usabilidad** (Garrido, Firmenich, et al., 2013; Grigera et al., 2016), aprovechando el trabajo previo sobre Refactorings Web del lado del Cliente (o CSWR por sus siglas en inglés: Client Side Web Refactorings) (Garrido, Firmenich, et al., 2013) transforma la interfaz, modificando el código HTML y JavaScript, sin necesidad de alterar o conocer el código interno.

Esta arquitectura extensible para incorporar nuevos eventos de usabilidad, bad smells y refactorings, hacen de Kobold una alternativa técnicamente viable que puede evolucionar integrando a su sistema de detección las barreras de accesibilidad y sus soluciones mediante refactoring.

3.2 Arquitectura Extendida de Kobold para Accesibilidad

La solución de software desarrollada ha seguido la línea original definida en el framework de Kobold, extendiendo sus funciones aprovechando las posibilidades que ofrece para la manipulación simplificada de xpaths, manejo de eventos en los elementos del DOM, y principalmente sus facilidades de implementación en el servidor mediante Pharo Smalltalk 9.

El resultado consiste en una nueva rama de desarrollo compuesta por dos componentes extendidos: uno del lado de cliente y otro del lado del servidor, con las mismas responsabilidades que sus análogos de usabilidad, pero adaptados a los propósitos de la accesibilidad.

En general se mantuvo el diseño original, utilizando idénticas estrategias de despliegue e instalación para la detección y corrección de las barreras de accesibilidad. Se agregó una fuente de eventos adicionales, implementada en mismo lector de pantallas **NVDA,** para eventos que están fuera del alcance del navegador web.

Una de las prioridades durante el desarrollo, fue mantener el mismo principio de "*Software como Servicio*" propuesto inicialmente en Kobold (Grigera, J, Garrido, A, & Rossi, G., 2017), de manera que la arquitectura permita ofrecer la "*Accesibilidad como Servicio*". Considerando los problemas de usabilidad como parte del comportamiento humano (David McQuillen, 2003) y la accesibilidad como una condición universal para que un producto o servicio resulte usable para todos, se retomó la propuesta de *usabilidad como servicio (Grigera, 2017),* y se utilizó la arquitectura de Kobold para proveer una implementación que ofrezca la accesibilidad como servicio.

En esta versión, el método de acceso al sistema continúa disponible mediante cuentas de usuarios, que vinculan la prestación del servicio en las páginas web con la aplicación en el servidor. Esto se implementó reutilizando el código como una especialización del snippet original, pero en este caso, para la captura de eventos de interacción de interés para la accesibilidad.

Respecto a las interacciones con el teclado sobre **NVDA** que no son posibles de detectar con el snippet incrustado, se desarrolló un complemento instalable en la aplicación **NVDA,** que extiende su comportamiento posibilitando capturar y remitir eventos de interacción específicos del NVDA sin requerir ninguna modificación en el componente de servidor.

El análisis de las interacciones de los usuarios está centrado en la búsqueda de eventos útiles para detectar problemas de accesibilidad. Cuando la cantidad de eventos detectados supera un nivel de control preestablecido se reportan la presencia de una barrera en forma de **Accessibility Smell** junto con alternativas de corrección como **Accessibility Refactorings** para usuarios con dificultades visuales.

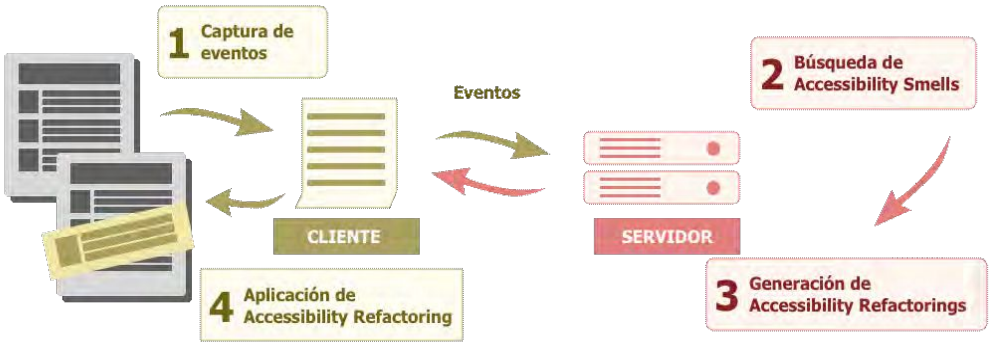


Figura 11: Arquitectura extendida de Kobold para los procesos para accesibilidad.

En la figura 11 se observa el flujo de comunicación donde el snippet incrustado y complemento **NVDA** remiten eventos al componente de servidor, que responde con transformaciones a la interfaz como refactorings de accesibilidad.

Igual que en la arquitectura original, cuando un evento llega al servidor es analizado por buscadores, pero en este son buscadores de **Accessibility Smells** específicos y ante la presencia de alguna dificultad la reportan para que el usuario decidir si aplica la sugerencia generada en los **Accessibility Refactoring**.

3.3 Eventos de Accesibilidad

Los eventos de accesibilidad, al igual que los eventos de usabilidad, se diseñaron en función de las barreras que se esperan detectar y basándonos en la observación de **Accessibility Smells** presentes en casos reales. Se estudiaron las acciones forzadas que deben realiza los usuarios que utilizan el teclado con el lector de pantallas NVDA y la posibilidad de capturarlas a través de interacciones sobre la interfaz.

El Pre-Procesamiento del lado del cliente

La captura de eventos de accesibilidad se realiza paralelamente tanto en el componente cliente en el navegador como en el complemento NVDA, delegando la aplicación de refactorings al Snippet incrustado en la aplicación.

Estas capturas son resultado de la integración de eventos de bajo nivel generados en la interfaz de usuario mediante el uso del teclado. Conjuntamente componen eventos de mayor nivel de abstracción evitando sobrecargar el servidor y facilitando conjugar más información sobre las acciones del usuario.

Los refactorings también se aplican mediante la evaluación de código embebido que se recupera del servidor. Las **instancias de refactoring** en cada página se deben aplicar realizando las modificaciones requeridas sobre la interfaz.

En síntesis, la nueva versión integra eventos para accesibilidad con nuevos buscadores, basados en heurísticas que reutilizan la lógica de generación de código JavaScript para los refactoring.

El diseño resulta similar al de Kobold para usabilidad, salvo que los eventos de interacción provienen de dos fuentes diferentes. Esta rama de desarrollo incluye un plugin de **NVDA** para capturar eventos de interacción no alcanzables desde el navegador. Luego de detectados en el Complemento NVDA, eventos de interacción también son agrupados en eventos de mayor nivel de abstracción como **eventos de accesibilidad** y siguen la misma lógica de procesamiento una vez remitidos al componente de servidor.

3.4 El Proceso de Detección de Accessibility Smells

Durante la captura de los **Accessilibity Smells** se emulan los tres pasos del modelo original: clasificación, agregación y análisis de eventos de accesibilidad. Las responsabilidades de detección se asignaron a especializaciones de las entidades **usability smell finders** originales (buscadores de usability smell de framework), que podemos denominar buscadores de accesibilidad y contienen algoritmos para detectar cada **Accesibility Smell** específico.

A los propósitos de accesibilidad, dado que cada aplicación define su propio conjunto de buscadores se realizaron ajustes a ciertos parámetros originales del framework asociados al número, proporción, o combinación de valores que desencadenan la presencia de cada **Bad Smell** y **Accessibility Event.**

Se mantuvo el modelo de 3 etapas de Kobold para la Clasificación de Eventos, Síntesis de Datos y Evaluación de Smells. Inicialmente solo los buscadores asociados a cada tipo de eventos de accesibilidad analizan y clasifican eventos en colecciones de la misma naturaleza (que afectan al mismo elemento). Luego según cada clase de evento de accesibilidad se sintetizan los

datos actualizando los valores en las cuentas de usuario y las proporciones involucradas en la lógica de detección. Por último, se evalúa la información recibida en búsqueda de **Accesibility Smell**, en cuyo caso se generan reportes a los usuarios.

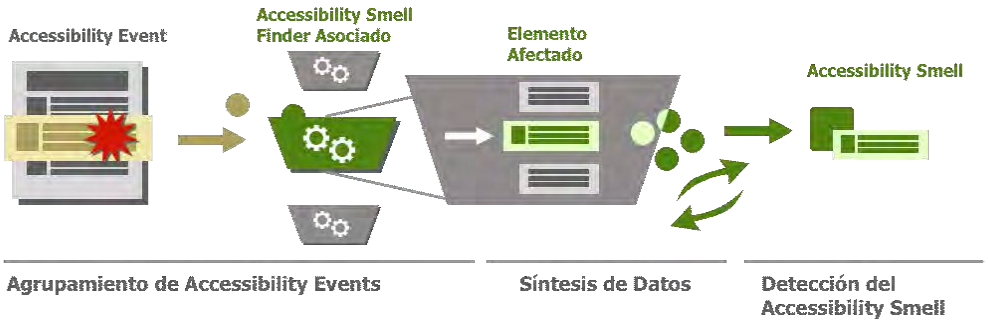


Figura 12: Instancia del Proceso de Detección para Accessibility.

3.5 Aplicación de Refactoring

La estrategia de modificar las presentaciones web mediante refactoring de interfaz utilizada en **Kobold** para la usabilidad (Grigera 2017), también es aplicada para la solución de Accessibility Smells. Como se explica en los párrafos anteriores, al menos técnicamente la accesibilidad puede ser tratada como la usabilidad de personas con algún tipo de dificultad. Por ello, se puede aplicar igual principio que el propuesto originalmente en **Kobold** y considerar que los comportamientos de usuarios afectados por bad smell de accesibilidad puedan ser solucionados por la aplicación de Refactorings Web del lado del Cliente (CSWR) (Garrido, Firmenich, et al., 2013). Para cada dificultad se reportan soluciones que puede ser aplicadas manualmente o automáticamente mediante refactoring, cuyo proceso continúa el mismo proceso descripto para **Kobold** donde se crea una instancia del refactoring que el componente cliente utiliza para modificar la interfaz.

Capítulo IV

Eventos de Interacción y Accesibilidad

En este capítulo se describen los eventos **Accessibility Event,** un caso real donde se presentan y la implementación del proceso de detección. Finalmente, se analizan los umbrales para los parámetros al momento de cada captura y el procedimiento para determinar sus valores.

4.1 Catálogo de Eventos de Accesibilidad

Los eventos de accesibilidad y los eventos de usabilidad se componen de interacciones que los usuarios realizan sobre la interfaz de una página web. Ambos poseen características comunes, que son necesarias para su procesamiento en el componente del servidor. Es así que se reutilizan las estructuras comunes definidas en los metadatos para todos los eventos de **Kobold**, como por ejemplo el momento en el que ocurre un evento o *timestamp.* Esto facilita la implementación de los **Accessibility Event** como especializaciones de los **Usability Event** que afecta a cada tipo determinado de elemento.

4.2 Implementación de la Detección de Eventos de Accesibilidad

La lógica para capturar las interacciones en **Kobold** consiste en incrustar en la página un **snippet** JavaScript conteniendo un objeto **Logger.** Este se asocia a una colección de objetos específicos, donde cada uno configura manejadores de eventos en la interfaz y reconocen la ocurrencia de una secuencia determinada de acciones del usuario que denota la ocurrencia de un **Usability Event**.

A cada **Usability Event** capturado se le adicionan datos del DOM según el elemento afectado y se los combinan con otros metadatos antes de ser enviados al servidor. Una vez en el componente de servidor, un servicio RESTful, recibe además de la información específica de cada evento:

1. El token: para identificar la cuenta de usuario.
2. La clase del evento: Para reconocer que instancia de clase de evento debe crear en el modelo.

3. Los datos particulares necesarios para cada evento como timestamp y la url donde se originó el evento.

Para los eventos basados en un elemento DOM también se dispone de:

1. xpath absoluto de elementos.
2. Contenidos HTML del elemento.
3. Dimensiones y posiciones de los elementos dentro de la página.
4. Lista de sus contenedores del elemento en el árbol DOM.

Cada evento se remite vía Ajax al endpoint del servidor en formato JSON de manera asincrónica y se realiza la conversión a un objeto del modelo en el servidor de la siguiente manera:

1. Se busca la cuenta de usuario que coincida con el token enviado.
2. Se crea el evento de la clase correspondiente en el modelo.
3. Se distribuye entre los buscadores de Usability smells y se recalcula la presencia de nuevos problemas de usabilidad.

Los **Accessibility Event** se pueden clasificar según su fuente en: *Eventos Detectables Mediante Incrustación de Componente*, para los cuales se puede utilizar código JavaScript embebido en las páginas web, y *Eventos de Accesibilidad del Modo Revisión de NVDA* que surgen de la posibilidad de navegar directamente elemento a través del lector de pantalla NVDA. En el primer conjunto se distingue un grupo particular de *Eventos Asociados a los Accesos Directos y Tareas comunes en Firefox* que surgen de la posibilidad de navegación alternativa, mediante teclado, que ofrece el navegador y son específicos Firefox.

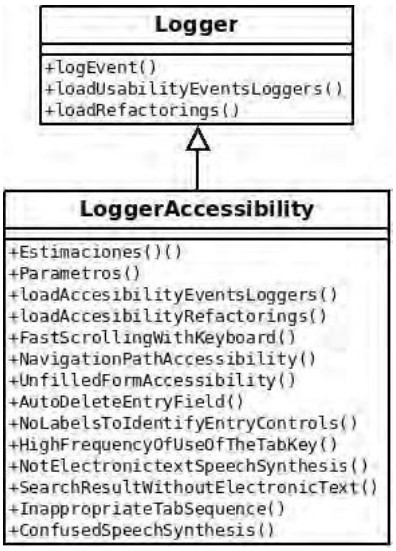


Figura 13: Herencia de Prototipos JavaScript para detección de Eventos de Accesibilidad

Para extender la detección de los **Usability Event** a los **Accessibility Event** mediante prototipado entre objetos JavaScript, se reutiliza el proceso en el caso de los eventos *Detectables Mediante Incrustación de Componente*. Se implementa en JavaScript el objeto **LoggerAccessibility** que sustituye al objeto **Logger,** heredando su estado y comportamiento, reemplazando la colección de objetos encargados de la detección de cada **Usability Event** por otros específicos que se configuran de acuerdo a los **Accessibility Event** definidos en este trabajo.

Para los Eventos de Accesibilidad del Modo Revisión de NVDA donde la comunicación entre el usuario y la página está mediada por el lector de pantallas **NVDA (**Modo Revisión), la detección se realiza en el complemento **NVDA Accessibility** descripto en el capítulo 6.

Independientemente de la fuente de eventos de accesibilidad el método de envío, al servicio RESTful en el componente de servidor, y la conversión a objetos del modelo se mantiene de la misma forma que la descripta previamente para **Kobold**.

4.1.1 Eventos Detectables Mediante Incrustación de Componente

E01 - Auto Delete Entry Field (Auto borrado de campo de entrada)

Este evento ocurre en los campos de entrada de texto, cuando los valores ingresados por el usuario son posteriormente eliminados automáticamente por la página web, posiblemente como consecuencia de algún tipo de validación de datos.

Caso Real: Destino de Vuelos en la Página de Aerolíneas Argentinas

Acciones:

Paso 1: El usuario de NVDA busca un vuelo en el tramo ida y vuelta, entre la ciudad de Jujuy y el destino Aeroparque ingresando los valores en los campos de entrada de origen y destino.

Paso 2: El destino ingresado no forma parte de los valores pertenecientes al dominio del campo de entrada y cuando el foco lo abandona el valor es eliminado por la página web.

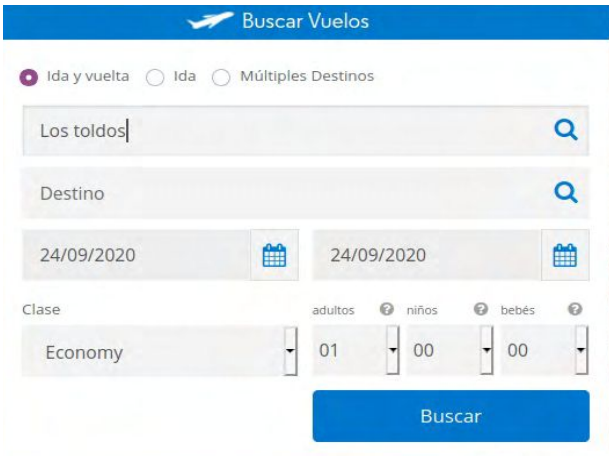


Figura 14: Página de reserva de vuelos de Aerolíneas Argentinas

La navegación puede proseguir al siguiente campo, sin que el usuario se notifique de la situación. La página rellena el campo con un valor vacío, y la interacción continua en el formulario hasta presionar la tecla <Enter> en el botón de búsqueda.

Paso 3: Al detectarse la inconsistencia de destino vacío, se reporta un mensaje visual con la leyenda “Debe ingresar un Destino”, sin que se reproduzca síntesis de voz.

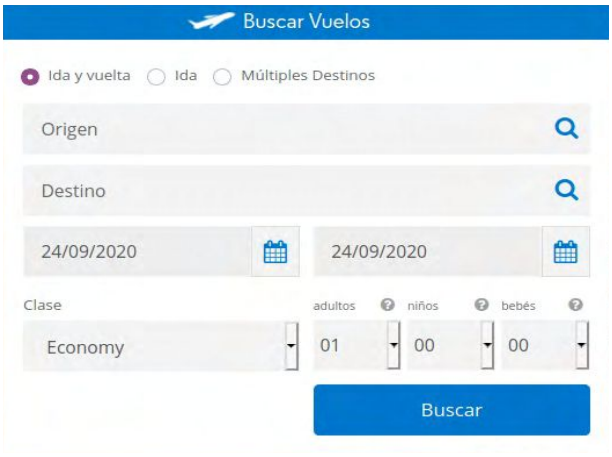


Figura 15: Página de reserva de vuelos de Aerolíneas Argentinas .

En esta condición el usuario desconoce que se ha producido un error y queda desconcertado debido a que la página ha suprimido su valor de entrada sin notificarlo.

Implementación

Los campos de entrada de texto contienen un valor por defecto, que puede ser vacío. Su valor inicial puede ser capturado en el evento J*avaScript focus*, y cualquier modificación que el usuario introduzca se puede detectar en el evento J*avaScript change* que suministra el nuevo dato. Si este último es alterado automáticamente al retirar el foco del campo de entrada, es posible detectar la situación y reportarla al servidor, comparando el último valor con el disponible en el evento *JavaScript blur*.

E02 - Electronic text for non-significant speech synthesis - Texto electrónico para síntesis de voz no significativa.)

Estamos ante este evento cuando no existe una etiqueta HTML <label> asociada a un elemento de entrada que lo describa convenientemente. Es importante que las etiquetas resulten significativas y claras sobre el dato de entrada a ingresar. Disponer de contenido textual útil a los usuarios con dificultades visuales es necesario para que puedan determinar la naturaleza del valor esperado.

La intención con este evento no es solo detectar los campos de entrada sin etiquetas HTML <label> asociadas. Reconocer esta situación se podría realizar a priori mediante la evaluación de código. Lo que se intenta es identificar aquellas circunstancias con las cuales interactúa el usuario en escenarios de aplicaciones que generan campos de entrada dinámicamente. Por ej: aplicaciones enriquecidas.

Caso Real: El usuario reserva un vuelo en la página de Aerolíneas Argentinas

Acciones:

Paso 1: El usuario accede al formulario de reservas de vuelos.

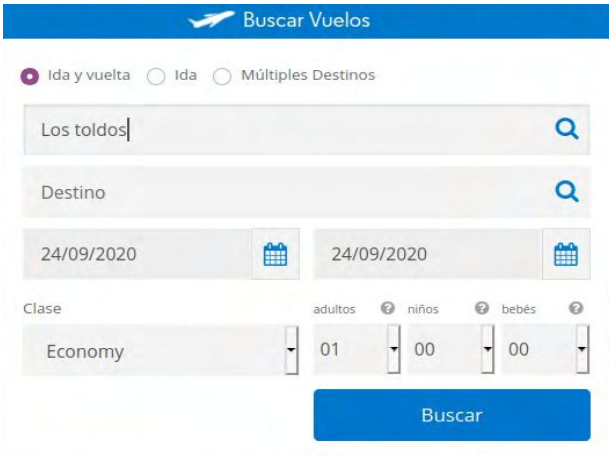


Figura 16: Página de reserva de vuelos de Aerolíneas Argentinas

Paso 2: Los campos de entrada que exhibe el literal origen y destino no disponen de etiquetas descriptivas <label>. Mientras que los elementos de entrada Clase, cantidad de adultos y niños, si cumplen el requisito.

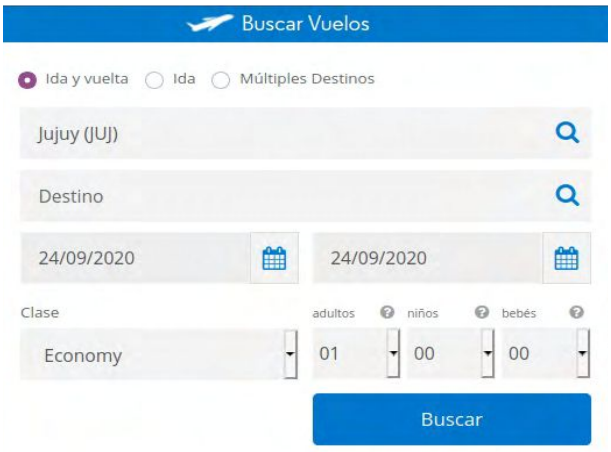


Figura 17: Página de reserva de vuelos de Aerolíneas Argentinas con los datos ingresados por el usuario

Se debe aplicar la etiqueta HTML <label>, adecuadamente para describir un texto relacionado con algún campo de entrada. En algunos casos los diseñadores utilizan las etiquetas HTML , para realizar identifica función que la etiqueta HTML <label>, pero es esta última, la diseñada para este propósito y el motivo por el cual los estándares recomiendan su uso.

Los lectores de pantalla utilizan en primera instancia el texto de las etiquetas HTML <label> para los campos de entrada como su texto electrónico a sintetizar.

Implementación

La detección del evento se presenta cuando el foco abandona el campo de entrada, en el evento *Javascript focusout*, momento en que se evalúa si el usuario no realizó ninguna entrada

por teclado. Si el tiempo de espera en el componente se encuentra dentro del umbral entre un tiempo mínimo y máximo de espera, se asume que el usuario estuvo esperando asistencia que lo orientara por síntesis de voz. Aun cuando la salida de audio su hubiese producido, pudo no resultar significativa según el contenido textual de la etiqueta HTML <label>. Es entonces cuando se verifica si el campo de entrada dispone de etiqueta HTML <label> vinculada por el campo *for* para reportar el evento al servidor.

Parámetros:

- Tiempo de espera mínimo (*MinimumWaitingTime*): Si la acción se produce en un tiempo inferior asumimos que el usuario recibió un mensaje suficientemente significativo que le permitió continuar interactuando.
- Tiempo de espera máximo (*MaximumWaitingTime*): Si la acción se produce en un tiempo superior resulta poco interesante porque el usuario pudo haber dejado de interactuar o incluso haber abandonado la página.

E03 - Electronic text for non-existent speech synthesis (texto electrónico para síntesis de voz inexistente)

El evento se reporta cuando los campos de entrada de texto, botones de opción o casillas de verificación, representados por las etiquetas HTML <Input> en los tipos text, radio y checkbox respectivamente, no tienen texto electrónico que se pueda sintetizar. Existe una secuencia de búsqueda para los textos electrónicos que incluyen el atributo *aria-label*, el texto de alguna etiqueta <label> vinculada o, en el caso de los textos, el atributo *placeholder*.

A diferencia de del evento **E02 - Electronic text for non-significant speech synthesis e**n este caso elemento debe carecer de la etiqueta HTML <label>.

Consideremos un campo de entrada de texto que no dispone de valores en sus atributos *aria-label* ni *placeholder.* En este caso estamos ante un posible **E02 Electronic text for non-significant speech synthesis** o **E3 - Electronic text for non-existent speech synthesis**. Todo dependerá si dispone de una etiqueta HTML <label> asociada y lo descriptivo de su contenido de texto.

La siguiente tabla de decisión representa este caso:

	Tiene etiqueta HTML <lavel> asociada	El texto contenido en la etiqueta HTML <lavel> es significativo al usuario	Evento que se Produce
El elemento afectado	NO	*	E3
	SI	NO	E2
	SI	SI	No se produce evento porque no existe barrera para el usuario

Tabla 3: Condiciones del Evento ***Electronic text for non-existent speech synthesis.***

Caso Real: Sistema de consultas del registro patrimonial de la Universidad Nacional de Salta

Acciones:

Paso 1: El usuario consultas del registro patrimonial de la Universidad Nacional de Salta visualizando los reportes del patrimonio que tienen a su cargo.

Figura 18: Sistema de Patrimonio SABUM de la Universidad Nacional de Salta

Paso 2: Al ingresar el foco al primer campo de entrada **NVDA**, solo indica la posición estructural del elemento sin sintetizar texto que describa el dato se espera ingresar. El elemento de la interfaz no dispone de atributo *aria-label*, ni de texto de una etiqueta <label> vinculada, ni de valor en el atributo *placeholder*.

Implementación

La detección del evento se produce cuando el foco abandona el campo de entrada en el evento *JavaScript focusout*, momento en el que se verifica lo siguiente:

1. Que el usuario no hubiese realizó ninguna entrada por teclado en el elemento.
2. Que el tiempo de espera en el componente se encuentra dentro del umbral entre un tiempo mínimo y máximo de espera.

3. Que el elemento afectado dispone de valores para los atributos *arial-label, placeholder* y de una etiqueta HTML <Label> vinculada.

Cumplidas estas condiciones se asume que el lector de pantalla no dispuso de ninguna de las tres alternativas para la síntesis de voz y se reporta el evento al servidor.

Parámetros:

En este caso los parámetros Tiempo de espera mínimo (*MinimumWaitingTime*) y Tiempo de espera máximo (*MaximumWaitingTime*) tienen idénticas consideraciones que en el evento anterior E02 - Electronic text for non-significant speech synthesis.

E04- High Frequency Of Use Of The Tab Key (Alta frecuencia de uso de la tecla de tabulación)

Los usuarios con dificultades visuales hacen uso del teclado recorriendo las páginas web para acceder a sus diferentes elementos. Pueden hacerlo utilizando la tecla de tabulación <tab> o <Shift> + <Tab> o recurrir a alternativas de acceso directo mediante una combinación de teclas configuradas en **NVDA** o en el Web Browser Firefox. El evento **High Frequency Of Use Of The Tab Key** se presenta ante una ráfaga de pulsaciones de siguiente elemento con tecla <tab>.

Caso Real: Solicitud de becas de estudio en la Universidad Nacional de Salta a través del sistema Siu Tehuelche.

Acciones:

Paso 1: El usuario accede a la página www.unsa.edu.ar

Figura 19: Página principal de la Universidad Nacional de Salta

Paso 2: El Usuario recorrer 32 elementos, utilizando la tecla <tab>, hasta el enlace “Becas” y así accede a la url “becas.unsa.edu.ar”.

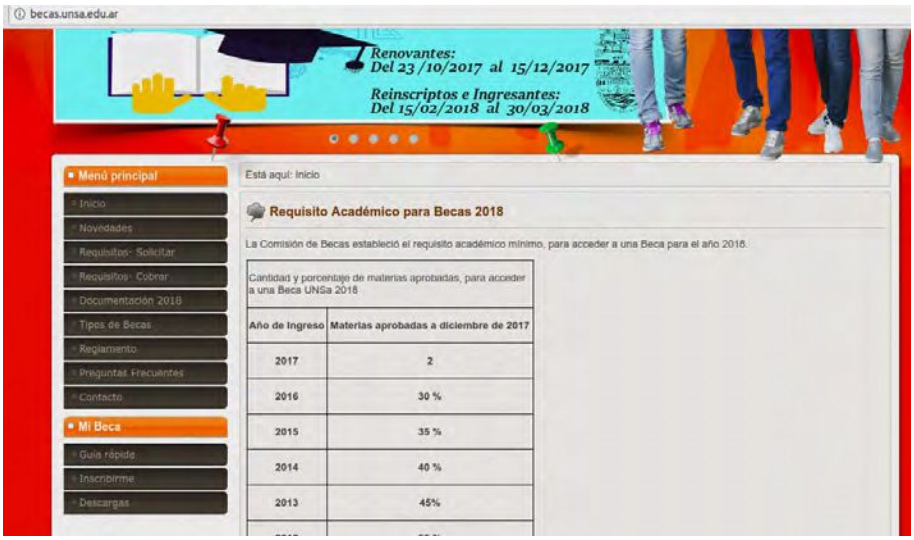


Figura 20: Página de Beca de la Universidad Nacional de Salta

Paso 3: El usuario recorrer 28 elementos, utilizando la tecla <tab>, hasta el enlace “Inscribirme” y acceder a la url becas.unsa.edu.ar/index.php/inscribirme.

Figura 21: Página de Inscripción de Beca de la Universidad Nacional de Salta.

Paso 4: El usuario recorrer 31 elementos, utilizando la tecla <tab>, hasta el enlace “Completar Solicitud” a través del cual acceden al sistema de becas Siu Tehuelche para registrarse y proceder a solicitar.

Figura 22: Sistema Siu Tehuelche de la Universidad Nacional de Salta.

Implementación

La función en el acceso a contenidos de la tecla <tab> es posicionar el foco en el próximo elemento focable de la página web. La detección en este caso está asociada al evento keyup, que se produce cuando el usuario hace uso de la tecla <tab> a modo de scroll. En la configuración de la detección se contabiliza que la cantidad de pasos realizados no supere una cantidad mínima, durante un tiempo de residencia en cada paso y además que la duración total del desplazamiento sea menor que un máximo establecido.

Parámetros:

1. Pasos mínimos (*minimumSteps*): Cantidad de acciones de scroll con tecla <tab> mínimas requeridas para considerar un potencial evento.
2. Tiempo máximo de desplazamiento (*MaximumScrollingTime*): Tiempo máximo de desplazamiento del evento durante las tabulaciones en ráfaga.
3. Tiempo de residencia (*dwellingTime*): Tiempo de espera luego de la última entrada de tecla <tab>.

E06 - Navigation Path Version Accessibility (Ruta de navegación Versión Accesibilidad)

En la versión **Kobold** para usabilidad, la ocurrencia del evento ***Navigation Path*** *(Grigera, 2017)* secuencia de navegaciones, captura una acción en cadena de navegaciones rápidas sobre varios nodos. Este evento puede extenderse para los usuarios con dificultades visuales cuando se produce utilizando el teclado como el evento de accesibilidad ***Navigation Path Version Accessibility.***

Caso Real: Creación de Cuentas de usuario en la Página de CONEAU Global

Acciones:

Paso 1: El usuario accede a la página http://209.13.179.3/coneauglobal/

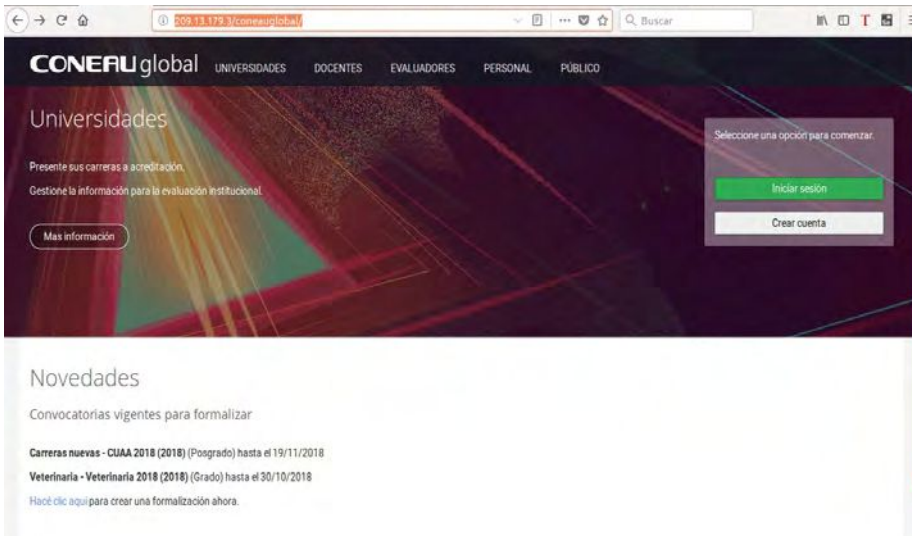


Figura 23: Página principal de CONEAU Global

Paso 2: El usuario accede al enlace Universidades y presiona la tecla <Enter>

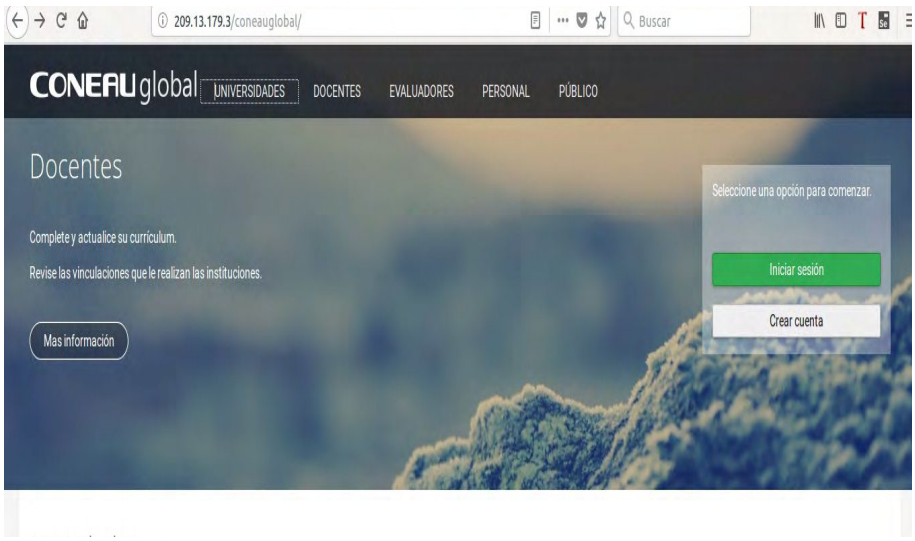


Figura 24: Sección Universidades de la Página de CONEAU global

Paso 3: El usuario accede a http://209.13.179.3/coneauglobal/atenea/acerca-de.aspx y se dirige al enlace *Comenzar* y presiona la tecla <Enter>

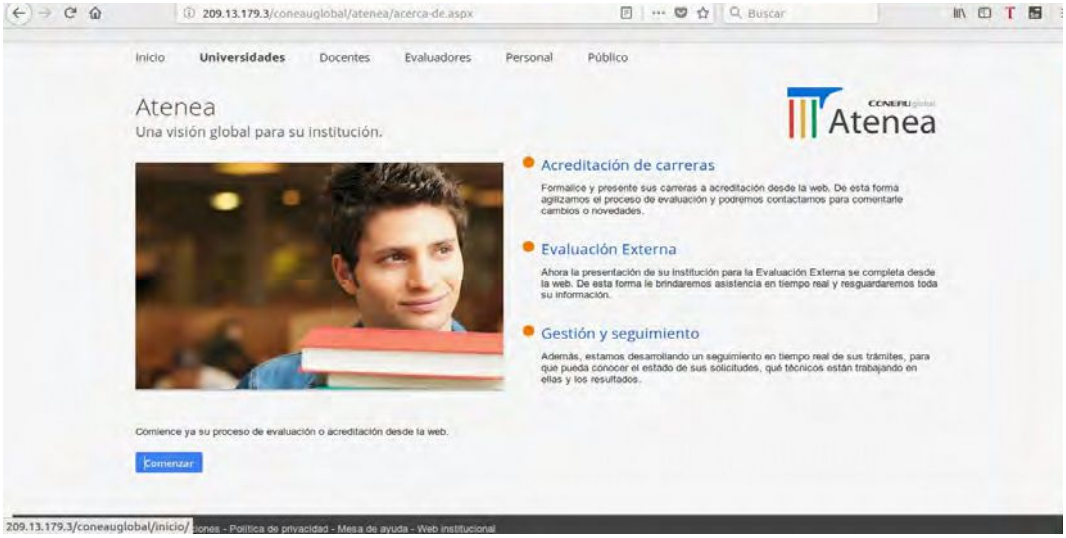


Figura 25: Visión global para las Instituciones de CONEAU global

Paso 4: El usuario accede a http://209.13.179.3/coneauglobal/iniciar-sesion/?ir=/coneauglobal/inicio/, se dirige al enlace Crear Cuenta y presiona la tecla <Enter>.

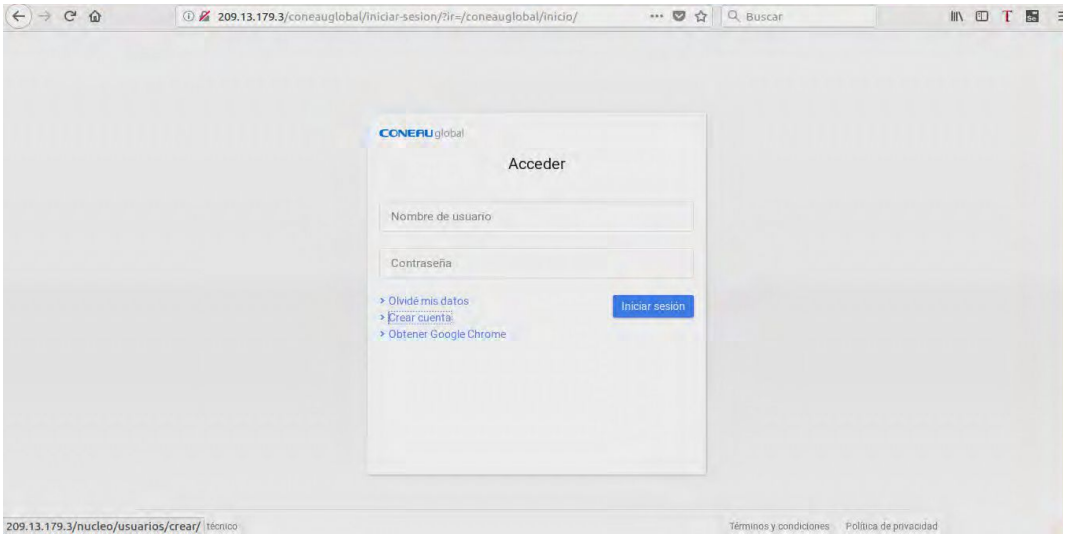


Figura 26: Creación de Cuentas de Usuario de CONEAU Global

Implementaciones

El evento Navigation Path Version Accessibility es un caso especial de Navigation Path definido en Kobold, que ocurre cuando los usuarios utilizan el teclado. Se implementa mediante herencia directa adecuando los parámetros paramMinimumNavigations y paramMaximumTime. Aunque la detección del evento Navigation Path se basa en los eventos de interacción JavaScript Click sobre enlaces que se acceden con el mouse, la detección para usuarios que utilizan el

teclado es posible dado que evento de interacción también se produce al pulsar la tecla <Enter> sobre los elementos de una página web.

Parámetros:

Cantidad Mínima de Navegaciones (*paramMinimumNavigations*): Cantidad mínima entre páginas navegadas para considerar el evento.

Máximo tiempo de Navegación (*paramMaximumTime*): Tiempo máximo que se espera en un nodo para detectar el evento de navegación entre páginas.

E07 - Unfilled Form Accesibility

El evento de usabilidad ***Unfilled Form*** (Grigera, 2017), puede ser reutilizado para reportar el no envío de un formulario. Para los usuarios con dificultades de visión que utilizan el teclado y **NVDA** este evento a diferencia de ***Navigation Path*** es independiente del tiempo y navegación, por lo que es utilizado con adecuaciones que se verán en la implementación.

Caso Real: Solicitud de Clave Única de Identificación Laboral en la Página del Anses.

Acciones:

Paso 1: El usuario accede a https://www.anses.gob.ar/constancia-de-cuil/, y al completar los datos requeridos encuentra un elemento Captcha que debe completar.

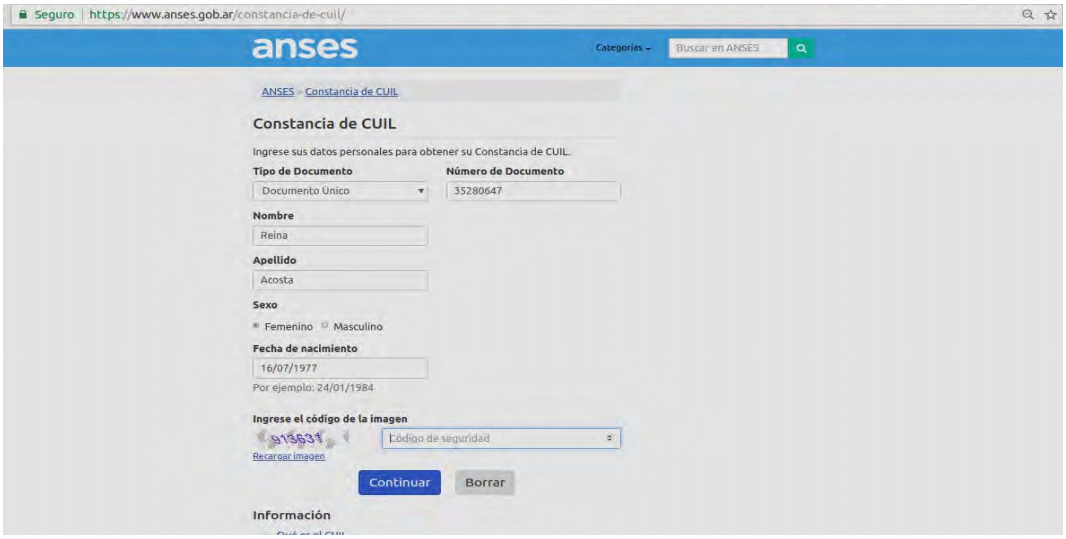


Figura 27: Solicitud de Constancia de Cuil en la Página web de Anses

El usuario NVDA está solicitando una constancia de CUIL (Código Único de Identificación Laboral) de la República Argentina y se le solicita que ingrese un código de seguridad presentado

visualmente como “9113631”. Sin embargo, **NVDA** nunca sintetiza el código porque no tiene acceso, lo que genera desconociendo en el usuario y le impide continuar.

Implementación

A diferencia de *Navigation Path*, este evento fue reutilizado directamente en la implementación de ***Unfilled Form Accesibility***, para dar soporte a los restantes pasos en el proceso de detección de barreras de accesibilidad como se verá en el Capítulo 6 - Bad Smells.

E08 - Search result without electronic text (resultado de búsqueda sin texto electrónico)

Algunos sitios incorporan cuadros de búsquedas asociadas a librerías externas, que facilitan reportar elementos vinculados a determinados descriptores. Podemos citar a *google custom search engine* (búsquedas personalizadas de google) que generan listas de resultados que se incorporan dinámicamente a la página.

La funcionalidad se incrusta en las páginas web, o son generadas por script dinámicamente al cargar la página.

Caso Real: El usuario NVDA busca el texto ”despacito” en la página de youtube.
Acciones:

Paso 1: El usuario accede a la página de youtube e ingresa el texto “despacito” y procesa la búsqueda.

Figura 28: Búsqueda del Texto “Despasito” en la página de Youtube.

El usuario obtiene como resultado una lista de vídeos sin ser informado, mediante síntesis de voz, que su solicitud fue procesada. Esto ocurre porque una vez finalizada la

búsqueda, a **NVDA** no se le suministran textos electrónicos que pueda sintetizar, motivo por el cual, el usuario no es notificado.

Implementación

En estos casos existe un retardo asociado a la carga de los códigos que vinculan las APIs en una página web. Al incluir estas funcionalidades de búsqueda y reportes externo es necesario esperar a que los elementos que constituyen la interfaz del servicio remoto esté disponible en la página. Por este motivo, el componente de cliente incrustado no puede configurar directamente los manejadores de eventos al cargar la página en el navegador. La solución fue posponer, utilizando la función JavaScript setTimeout, la carga de manejadores de eventos en JavaScript, inicializando la detección con un retardo (*delayConfiguringHandlers*) mayor al tiempo de carga de los componentes remotos en la interfaz.

Estas operaciones de búsqueda pueden estar vinculadas a formularios, cajas de cuadro de texto y ejecutarse mediante botones o ingresos por teclado <Enter>.La detección se basa en utilizar los eventos JavaScript *click, keyup y submit,* para encontrar términos que describan operaciones de búsqueda dentro de los elementos del formulario. Tal es el caso de s*earch*, *search_query* y descripciones como *gsc-search-button, gsc-search-button-v2* para los botones o cajas de búsqueda. En caso de ejecutarse la búsqueda presionando la tecla <Enter> sobre los campos de entrada, el algoritmo verifica tiempos máximos y mínimos en las interacciones para descartar falsos positivos.

Parámetros:

Tiempo de mínimo (*MinimumWaitingTime*) y tiempo máximo (*MaximumWaitingTime*) de espera que determina el rango de valores para la residencia en un elemento de la página antes de detectar el evento.

Retardo para manejadores de Eventos (DelayConfiguringHandlers): Tiempo de retardo para configurar los manejadores de eventos en JavaScript para la interacción en el componente de cliente.

E09 - Confused speech synthesis (síntesis de voz confusa)

Pueden existir diversos motivos por los que una síntesis de voz emitida por el lector de pantalla podría resultar difícil de comprender por los usuarios, como ruidos en el ambiente o distorsión en la reproducción del audio. En esta implementación durante el proceso de detección se recolecta información que trata razones vinculadas al idioma de los textos definidos en los elementos de la interfaz.

El proceso incluye específicamente los botones y enlaces, para los cuales los asistentes de navegación deben utilizar el idioma definido en la página web para reconocer que tipo de tratamiento deben realizarle a sus textos. Para sintetizar correctamente texto a voz se requiere reconocer el idioma de cada contenido textual de un elemento. De lo contrario el usuario puede no comprender el audio que se le reproduce, dando lugar a un evento de accesibilidad ***Confused speech synthesis***.

Caso Real: Catálogo de la Biblioteca de la Universidad Nacional de Salta

Acciones:

Paso 1: El usuario NVDA accede al catálogo de la biblioteca y busca el texto “Salud”, obteniendo como resultado una lista de ejemplares bibliográficos que se agrupan según ese descriptor.

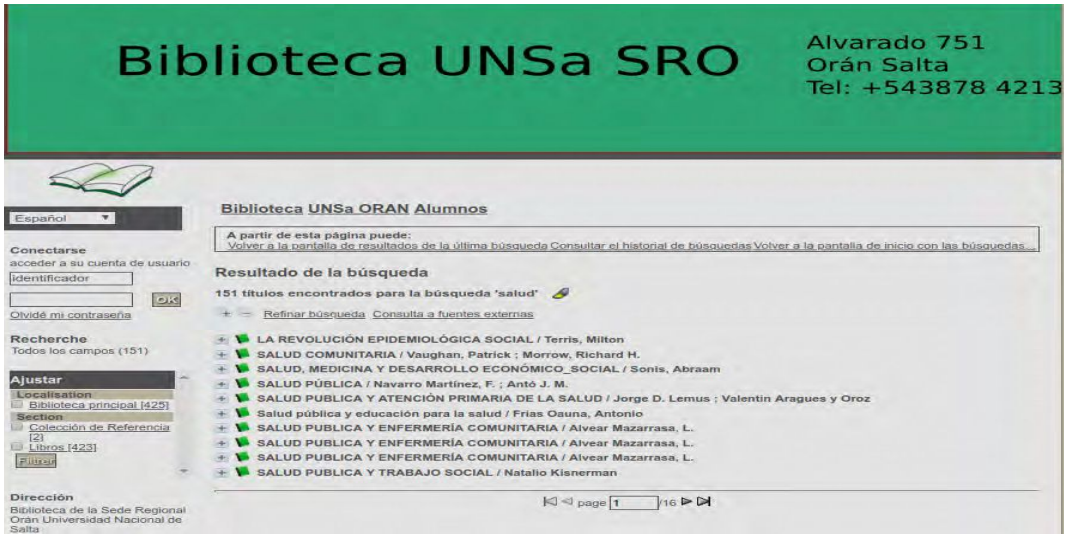


Figura 29: Catalogo de la biblioteca de la Universidad Nacional de Salta.

Paso 2: En el margen izquierdo se puede filtrar la búsqueda aplicando categorías.

Figura 30: Filtros del Catálogo de la biblioteca

Paso 3: Estos filtros se encuentran categorizados, en grupos cuyos textos electrónicos se encuentran en un idioma diferente al de la página actual.

Dado que el idioma de los textos contenidos en los elementos utilizados para el filtrado no se corresponden con el definido en la página web, **NVDA** reproduce un texto que el usuario puede no comprender.

Implementación

En la detección se espera que el foco abandone el componente HTML. Si el tiempo que foco se mantuvo dentro del botón o el enlace, sin que el usuario realice ninguna entrada, se encuentra dentro del umbral entre mínimo (minimumWaitingTime) y máximo (maximumWaitingTime) se inicia el proceso de detección.

Debido a dificultades técnicas y por razones de performance, para no incluir retardos en los navegadores se delega la tarea de detección de idiomas al componente de servidor. Utilizando una API es posible detectar los idiomas de los textos del título de la página y del texto del elemento potencialmente afectado por el evento. A esto se suma el idioma definido en el navegador y el definido en la página. Con esta información se compone un evento en el modelo que luego es procesado por los buscadores como se describe en el Capítulo 6 Bad Smells.

Al momento de capturar el evento se remiten al servidor:

Lenguaje configurado en el navegador (browserLanguage)

Lenguaje definido en la página (pageLanguage, atributo Lang)

Texto del título de la página

Lenguaje definido para el elemento afectado (elementLanguage, atributo Lang)

Contenido del texto de elemento.

El objeto que instancia el evento en el servidor utiliza una API (ws.detectlanguage.com/0.2/detect) para detectar el idioma del texto del título de la página y el del contenido textual del elemento.

Parámetros:

En este caso también son necesarios MinimumWaitingTime y MaximumWaitingTime ambos utilizados de la misma manera descripta en los eventos previos.

E10 - Inappropriate Tab Sequence (secuencia de tabulación inapropiada)

El **orden en la tabulación** de los elementos contenidos en una página, resultan importantes para acceder a los campos de entrada por medio del teclado. En los navegadores, se

utiliza la tecla de tabulación <Tab> para navegar secuencialmente entre enlaces y controles de formulario.

Orden de los elementos en el código HTML: es el orden con el que aparecen los elementos representados por etiquetas HTML dentro del archivo que contiene el código de una página web.

Orden de Presentación: es la secuencia en la que se disponen los elementos de la página web en el navegador que puede no coincidir con su orden dentro del código HTML.

Orden de tabulación: es la secuencia a la que se accede mediante el teclado presionando la tecla <tab> en el navegador web y que puede no coincidir con el **Orden de Presentación**.

Cada uno de los elementos dentro de estas formas de ordenarlos tiene un orden de prelación o secuenciación**.** Por ejemplo, dados los elementos HTML C, B, A, D se obtiene:

	Orden de Secuenciación	**Vista**
Orden de los elementos en código Html	A, B, C, D	Los que codifican los desarrolladores.
Orden de Presentación	A, C, B, D	Lo que visualizan los usuarios en el navegador.
Orden de tabulación	D, B, C, A	Como se accede con la tecla de tabulación <tab>

Tabla 4: Ordenes de Secuencia del Evento ***Inappropriate Tab Sequence***

De forma predeterminada, el **orden de tabulación** inicia en el primer enlace o control de la página continuando con el **orden de presentación** de los elementos. Este **orden de secuencia** puede ser alterado, modificando el atributo "tabindex" o incluso en **orden de presentación** visual de la página puede ser modificado mediante cambios en CSS. Frecuentemente estas alteraciones se realizan sin considerar el impacto en la accesibilidad mediante el teclado. El evento **Inappropriate Tab Sequence** detecta configuraciones inconsistentes entre el **orden de presentación** y **orden de tabulación,** que puede dificultar la interacción de los usuarios cuando dependen del teclado y un Screen reader.

Caso Real: Solicitud de becas del sistema Siu Tehuelche

Acciones:

Paso 1: El usuario intenta crear una cuenta para solicitud de becas en el sistema Siu Tehuelche con la dificultad del acceso al Widget de selección de fecha, debido a que el orden de secuencia no corresponde con el elemento siguiente al campo de entrada para la fecha.

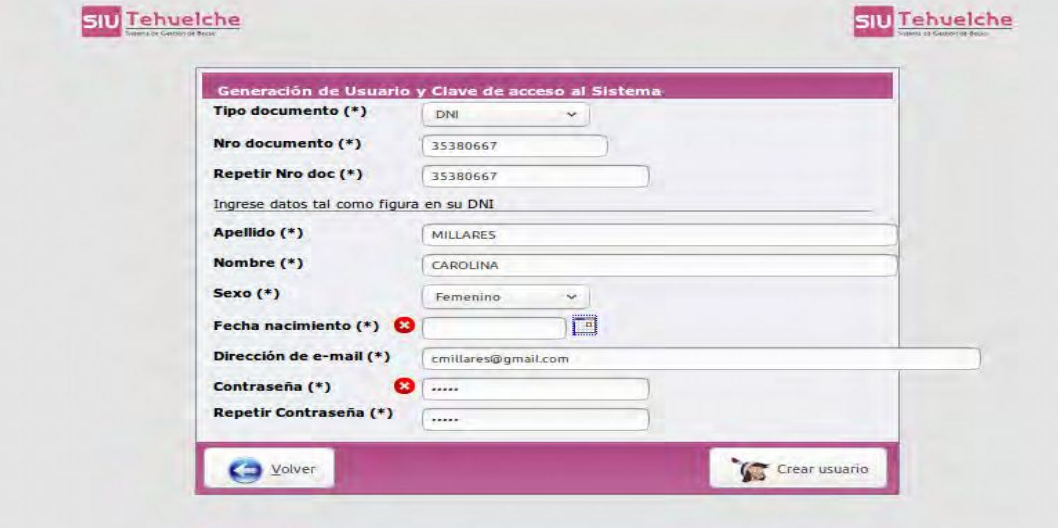


Figura 31: Solicitud de becas del sistema Siu Tehuelche.

Luego de recorrer todos los elementos de entrada del formulario el foco se posiciona sobre el Widget para el ingreso de la fecha de nacimiento. Además el Widget no responde a entradas de teclado para desplazarse y seleccionar una fecha determinada.

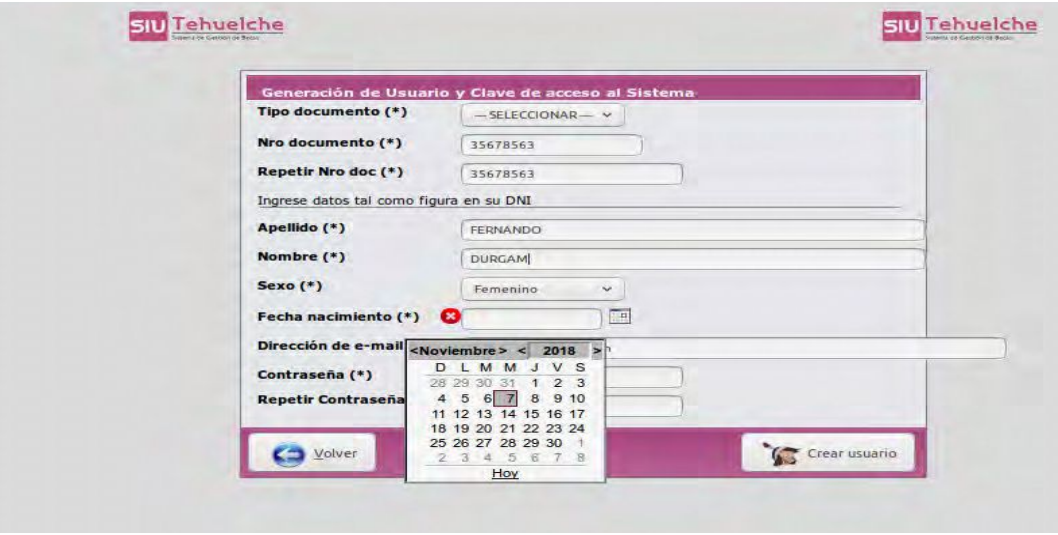


Figura 32: Widget para campo de fecha inaccesible.

El orden de tabulación de los elementos contenidos en el formulario resulta importante para acceder a los campos de entrada por medio del teclado. Un orden inadecuado puede dificultar la interacción de los usuarios con dificultades visuales.

Implementación

En cada entrada de tabulación con tecla <tab> en el evento JavaScript *keyup*, se analizan las posiciones (top y left) de los elementos definidos con etiquetas HTML <input>, <select>, <button>, <a> que recibieron el foco previamente en el evento JavaScript *focus*.

Según el orden en que fueron accedidos los elementos con respecto a sus correspondientes elementos previos y siguientes, se comparan las posiciones top y left, consideran los desplazamientos de altura (height) del elemento. De esta manera se determina si en la secuencia producida de la página, se puede descubrir un orden que pueda dificultar el acceso a los usuarios con dificultades visuales que utilizan el teclado.

El algoritmo de detección requiere procesar 4 (cuatro) elementos accedidos mediante teclado con tecla <tab>.

Actualmente la implementación detecta dos casos:

Caso A	
Valores	< --Left -->
< –-Top---->	A
	C
	B
	D
Secuencia obtenida	A,B, C, D

Tabla 5: Orden de tabulación del Caso A de Evento ***Inappropriate Tab Sequence***

Caso B	
Valores	<---Left--->
< –----Top---->	A, C, B, D
Secuencia obtenida	A,B,C,D

Tabla 6: Orden de tabulación del Caso b de Evento ***Inappropriate Tab Sequence***

La navegación de B a C muestra un retroceso en el orden de secuenciación.

4.1.2 Eventos de Accesibilidad Asociados a los Accesos Directos y Tareas comunes en Firefox

El navegador Mozilla Firefox incluye características que facilitan la accesibilidad al contenido web a todos los usuarios que tienen problemas de visión (total o parcial) o con capacidad limitada para utilizar el mouse. Este acceso es posible mediante atajos de teclado asociados a determinadas pulsaciones. Si bien el navegador cuenta con un gran número de funcionalidades disponibles, a los efectos de considerar esta categoría de eventos se presenta solo el siguiente caso:

E11 – Fast Scrolling With Keyboard

Algunos atajos de teclado disponibles en el browser Mozilla Firefox, permiten a los usuarios saltar rápidamente el contenido presentado en la web mediante acciones de scroll. Para los eventos de usabilidad Flash Scrolling (Grigera, 2017), los usuarios necesitan un buen motivo para saltar el contenido. Este evento también se presente en la accesibilidad pero requiere ser adecuado a usuarios con dificultades visuales y ajustar su detección a interacciones por teclado mediante el evento ***Fast Scrolling With Keyboard***.

Caso Real: Búsqueda de alojamiento en la Página de Booking

Acciones:

Paso 1: El usuario realiza una búsqueda de alojamientos y se desplaza en la lista de resultados hacia abajo presionando la tecla <space> y hacia arriba presionando <sihft> + <space>

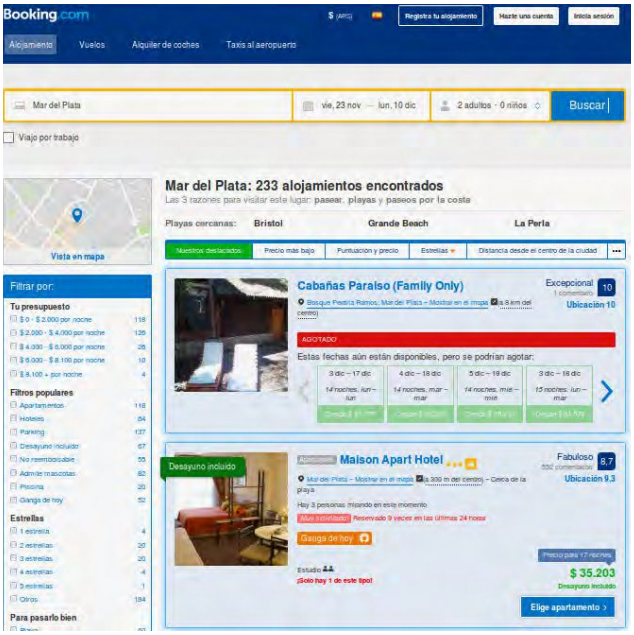


Figura 33: Página de Alojamiento de Booking.

Implementación

El evento se produce sobre la presentación de la página, cuando un usuario realiza un scroll presionando la tecla <space> y hace uso de los Accesos Directos y Tareas comunes en Mozilla Firefox. El algoritmo de detección del evento es una reescritura del evento de usabilidad **FlashScroll** de **Kobold** (Grigera,2017), que verifica entre sus condiciones que el desplazamiento se produzca con la tecla de espacio en el evento JavaScript *keypr*ess. En la configuración de la detección se contabiliza que la cantidad de pasos realizados supere una cantidad mínima (*MinimumSteps*), durante un tiempo de residencia en cada paso (*dwellingTime*) y que el tiempo total de desplazamiento sea menor que un máximo definido (*MaximumScrollingTime)*.

Parámetros

Pasos mínimos (*MinimumSteps*): Pasos mínimos para considerar significativo el desplazamiento.

Tiempo máximo de desplazamiento (*MaximumScrollingTime*): Tiempo máximo de espera para reportar el evento.

Tiempo de residencia (*dwellingTime*): Tiempo durante el cual se espera entre dos entradas sucesivas para contabilizar el evento.

4.1.3 Eventos y Bad Smell en Modo Revisión del NVDA

Esta categoría surge de la necesidad de considerar combinaciones de teclas que el screen reader **NVDA** ofrece para facilitar la exploración de una página sin mover el foco de la aplicación o el cursor del sistema. Incluso permite acceder a representaciones en formato de objetos a elementos de una página web, que no son accesibles mediante el foco utilizando el teclado, por ejemplo los encabezados de primer nivel <h1>. Este modo de utilizar la aplicación hace posible desplazarse lo largo de todo el texto utilizando un cursor virtual, similar al cursor del sistema. Para lograrlo **NVDA** utilizan para representar las páginas webs un **Modo Virtual** denominado **Buffer Virtua**l, como un contenido marcado, navegable con teclas de cursor o atajos de teclados (**gestos**). Esta navegación de modo virtual facilita el acceso directo a los elementos de las páginas web representados jerárquicamente como un modelo de objetos a navegar, por lo cual, para detectar estos eventos, fue necesario desarrollar un complemento de software para **NVDA** como se describe en el capítulo 5.

A continuación se describen dos casos de eventos de accesibilidad que se presentan cuando **NVDA** está configurado en modo revisión:

E5 - Flash Scrolling in Accessibility with NVDA add-on (Ráfaga de desplazamiento con NVDA)

En este modo de acceder mediante un **buffer virtual** a la estructura del documento, los navegantes pueden saltear áreas de la página como desplazamientos de contenido, ingresando

repetidamente comandos de acceso directo, mediante combinaciones de teclas. Una ráfaga de navegación podría indicar un comportamiento condicionado por el contenido de la página por la presencia de un evento de accesibilidad **Flash Scrolling in Accessibility with NVDA add-on.**

Caso Real: Búsqueda en la página de Wikipedia.

Acciones:

Paso 1: El usuario accede a la página de Wikipedia para investigar sobre la historia de Egipto, mediante el enlace https://es.wikipedia.org/wiki/Historia_de_Egipto.

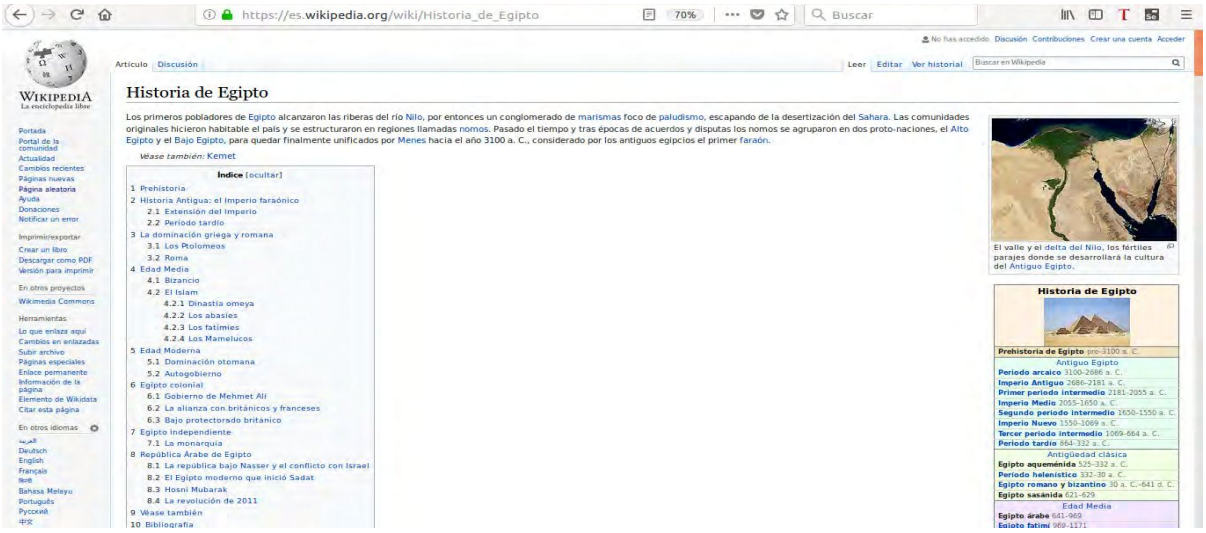


Figura 34: Página de Wikipedia sobre la Historia de Egipto.

La página contiene elementos que pueden ser accedidos directamente por los usuarios de **NVDA**. La presentación incluye un título con un encabezado de nivel 1, representado por la etiqueta HTML <h1>, una lista de enlaces representado por la etiqueta HTML <ul> y que contiene menús de navegación, imágenes y una gran cantidad de secciones de encabezados de nivel 2 y 3.

Configurando el screen reader en **modo revisión** se pueden utilizar las teclas predeterminadas, que se describen en la sección *desarrollo de complemento NVDA Accesibility del capítulo 5*, para acceder mediante a un **buffer virtual** de la estructura del documento y realizar una navegación saltando áreas de la página

Implementación

Los desplazamientos en el **buffer virtual**, se producen cuando los usuarios realizan entradas como gestos (combinaciones de teclas) que son detectados en el complemento. Existen entidades que almacenan referencias a los eventos de interacción que podrían resultar útiles para detectar algún evento de accesibilidad.

Cuando se produce un desplazamiento en el buffer las interacciones sucesivas se agrupan mientras el tiempo transcurrido entre dos pares de desplazamientos no supere un valor mínimo (deltaTime).

De superarse este valor, se analizan las entradas recibidas verificando que la cantidad de pasos realizados supere la cantidad mínima (*minimumSteps*) requerida para considerar significativo el desplazamiento.

En los eventos del **buffer virtual** no es necesario definir tiempos máximos para los eventos de accesibilidad porque el complemento contiene un hilo que chequea la actividad regularmente (cada 40 segundos) procesando todas las interacciones capturadas y reporta los eventos almacenados previamente.

Parámetros:

Pasos mínimos (*minimumSteps*): Cantidad de acciones de desplazamiento mínimas requeridas para considerar un evento de desplazamiento.

Tiempo delta (*deltaTime*): Tiempo transcurrido entre dos interacciones de desplazamiento sucesivas**.**

E12 - Navigation Between Lists of Links for the NVDA add-on (Navegaciones entre listas de enlaces para el complemento NVDA)

NVDA en modo revisión, posibilita el acceso de manera secuencial o directa a grupos de colecciones de enlaces. Por ejemplo, cuando los usuarios que recorren colecciones de listas de enlaces hasta navegar a la estructura de su interés. Este comportamiento da cuenta de un camino recorrido para acceder a una de las listas, lo que se reporta como un evento de accesibilidad **Navigation Between Lists of Links for the NVDA add-on** al servidor.

La recurrencia de seguir una determinada secuencia de acceso entre las listas de menús (circuito de navegación rápida) podría indicar que el contenido se presenta como distante para los usuarios.

Caso Real: Acceso a la página de la Universidad Nacional de Salta deseando conocer el menú del comedor para la semana actual. Ingresa en modo revisión de **NVDA** usando operaciones de navegación rápida

1. Paso 1: El usuario accede a la página www.unsa.edu.ar.

Figura 35: Página de la Universidad Nacional de Salta.

Paso 2: Se realiza la navegación entre las listas con la tecla <l> hasta la 4ta lista en la solapa Bienestar Universitario, posicionándose en el enlace “Becas”.

Figura 36: Solapa de Bienestar Universitario.

Paso 3: Dentro de esta lista se navega hasta el segundo enlace con la tecla <k>, posicionándose en el enlace “Comedores Estudiantiles”.

Paso 5: Expande el menú con la tecla <espacio>

Figura 37: Enlaces de Comedor Estudiantil.

Paso 6. Navega hasta el segundo elemento de la lista, “Menú del Comedor” presionando en dos ocasiones la tecla <i> y accede al menú presionado <ENTER>.

En el ejemplo la página contiene listas de elementos de enlaces como menús de acceso, que son utilizadas por el usuario de **NVDA** en **modo revisión**.

Implementación

En la navegación entre listas de enlaces para el complemento **NVDA**, al igual que en el evento de accesibilidad anterior, se almacenan las referencias a las interacciones sucesivas de desplazamientos entre listas, analizando que contengan una cantidad de elemento superior a un valor mínimo (*minimunChildren*).

Cuando se produce una interacción diferente a la navegación entre listas, o bien el complemento determina que la actividad del usuario ha cesado. Se procesan las interacciones detectadas verificando que la cantidad de pasos de entradas de navegación entre listas supere un valor mínimo requeridos para considerar el evento (*minimunSteps*) y que cada una de las listas contenga al menos un enlace.

Parámetros:

Cantidad mínimo de Elementos Hijos (*minimunChildren*): Cantidad mínima de elemento que debe contener una lista para considerarla en el evento.

Pasos mínimos (*minimunSteps*): Cantidad de acciones de desplazamiento mínimas requeridas para considerar un evento desplazamiento.

Capítulo V

Análisis de Valores de Parámetros para los Eventos de Accesibilidad

Los comportamientos que los navegantes efectúan al interactuar en la web con **NVDA** utilizando un teclado se modelan como **Evento de interacción.** Combinando cada evento y agregándolos en **Accessibility Event** pueden ser detectados **automáticamente.**

Al definir puntos de control que pueden ser evaluados para determinar la presencia de conductas particular del usuario, es posible buscar coincidencias con un conjunto de condiciones esperadas que se definen en cada **Accessibility Event.**

En términos prácticos, estas condiciones se examinan mediante el contraste de valores de parámetros que requieren ser estimados sobre la base de comportamientos observables en usuarios con dificultades de visión.

Con el objeto de ajustar el proceso al comportamiento especificado para cada evento, en **Usability Event** en **Kobold** (Grigera, 2017), los valores utilizados en los algoritmos de detección definen umbrales temporales y valores que asignan mensura las acciones sobre la interfaz.

En el caso de los **Accessibility Event**, también son necesarios estos valores de referencia, cuya definición y medición deben ajustarse a las **características de la salida para los usuarios** y a la **tecnología de asistencia** utilizadas.

5.1 Características de salida para los usuarios

Las dificultades visuales de los navegantes hace que la comunicación visual sea restringida o nula. Los valores para los parámetros en los algoritmos de detección pueden ser diferentes a los encontrados en Usability Event en Kobold (Grigera, 2017).

Existe una relación entre el tiempo requerido para cambiar el registro de texto a voz y la longitud de texto electrónico a reproducir. Puede ser necesario sintetizar una mayor cantidad de texto que la disponible en la prestación Web.

Ante la carencia de comunicación gráfica, los textos electrónicos de una página web, deben contener más información que la visible en la interfaz y los usuarios con dificultades visuales podrían requerir tiempos diferentes para interactuar con la aplicación.

Ante la necesidad de ubicar a los usuarios dentro de la estructura de la página, **NVDA** incluye información sobre el punto de interacción en sus salidas de audio, incorporando más texto a ser sintetizado. De este modo, indica la posición de los elementos dentro de la estructura de la página. Estas situaciones pueden incidir en las estimaciones de los umbrales temporales de cada interacción cuando se desplaza el foco entre los elementos.

La situación se puede ilustrar con el siguiente ejemplo donde, se observa la navegación de los elementos de entrada de texto distribuidos dentro de una estructura de una tabla HTML:

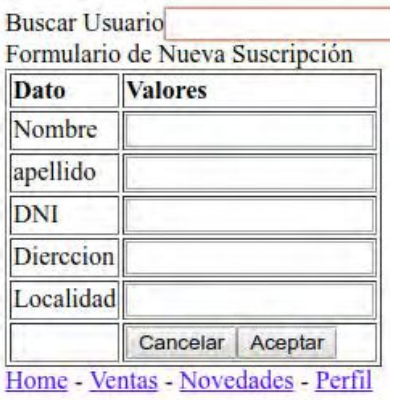


Figura 38: Tabla conteniendo elementos de entrada de texto en una página web.

Los textos que se sintetizan contienen más información que los presentados visualmente en la figura, e incluso más que los que pudieran haber especificado los desarrolladores en la semántica.

Cuando se posiciona el foco sobre un elemento de campo de entrada de texto dentro de una tabla, **NVDA** también indica la ubicación del mismo dentro ese contenedor. Es decir, que dará a conocer la fila y la columna accedidas, que coinciden con la celda que contiene al elemento que recibe el foco.

La incorporación de estos datos al texto visible que se debe informar, influye en el tiempo requerido para la síntesis de voz y el tiempo necesario para que el usuario pueda escuchar e interpretar la información.

5.2 La tecnología de asistencia

Los usuarios con dificultades visuales pueden utilizar diferentes tecnologías para interactuar con las aplicaciones web que van desde anillos para reconocimiento de texto, pantallas táctiles, figuras brailes, teclados brailes e ingreso por comandos de voz.

En este caso, estamos centrados en aquellos que utiliza el lector de pantallas **NVDA** y los teclados convencionales como instrumentos de acceso por lo que es necesario considerar la manera en que estos pueden influir en la estimación de los parámetros.

En principio, una estrategia de estimación basada en la observación directa resultaría conveniente y la misma solo requeriría relevar los instantes iniciales y finales de cada interacción; llevando la cuenta de la cantidad de ocurrencias de las acciones del usuario que se desean evaluar.

Si embargo, durante la navegación, algunas de esas acciones se producen sobre el lector de pantallas y no directamente sobre la interfaz visible, por lo que la observación directa no es suficiente para reconocer las acciones que efectivamente se realizan. Este es el caso de las navegaciones de objetos en el modo revisión que se producen sobre el **buffer virtual** de **NVDA** y no se reflejan sobre la interfaz visible, motivo por el cual el observador nunca notaría la ocurrencia de estos eventos.

La solución a esto es analizar la sucesión de eventos de interacción propios de **NVDA** para conocer lo que sucede en la aplicación cuando el usuario interactúa con una página web.

N.º	Fecha Hora	Evento	Elemento Afectado	ROL NVDA	URL	Fecha Hora Evento Previo	Duración en Segundos	Segundos Trascurridos
18	2019-07-17 20:19:35.188000	FastScrollingWithKeyboard	body	ROLE_DOCUMENT	192.168.1.110/accesibilidad2/otros/NVDAyTablas.html	2019-07-17 20:19:35.063000	0	6
19	2019-07-17 20:19:35.763000	event_becomeNavigatorObjec	input	ROLE_EDITABLETEXT	192.168.1.110/accesibilidad2/otros/NVDAyTablas.html	2019-07-17 20:19:35.188000	0	6
20	2019-07-17 20:19:35.773000	focusEntered	div	ROLE_SECTION	192.168.1.110/accesibilidad2/otros/NVDAyTablas.html	2019-07-17 20:19:35.763000	0	6
21	2019-07-17 20:19:35.804000	focusEntered	label	ROLE_LABEL	192.168.1.110/accesibilidad2/otros/NVDAyTablas.html	2019-07-17 20:19:35.773000	0	6
22	2019-07-17 20:19:35.824000	gainFocus	input	ROLE_EDITABLETEXT	192.168.1.110/accesibilidad2/otros/NVDAyTablas.html	2019-07-17 20:19:35.804000	0	6
23	2019-07-17 20:19:40.698000	typedCharacter	input	ROLE_EDITABLETEXT	192.168.1.110/accesibilidad2/otros/NVDAyTablas.html	2019-07-17 20:19:35.824000	4	11
24	2019-07-17 20:19:41.374000	event_becomeNavigatorObjec	input	ROLE_EDITABLETEXT	192.168.1.110/accesibilidad2/otros/NVDAyTablas.html	2019-07-17 20:19:40.698000	0	12
25	2019-07-17 20:19:41.828000	focusEntered	form	ROLE_FORM	192.168.1.110/accesibilidad2/otros/NVDAyTablas.html	2019-07-17 20:19:41.374000	0	12
26	2019-07-17 20:19:41.844000	focusEntered	table	ROLE_TABLE	192.168.1.110/accesibilidad2/otros/NVDAyTablas.html	2019-07-17 20:19:41.828000	0	12
27	2019-07-17 20:19:41.875000	focusEntered	tr	ROLE_TABLEROW	192.168.1.110/accesibilidad2/otros/NVDAyTablas.html	2019-07-17 20:19:41.844000	0	12
28	2019-07-17 20:19:41.891000	focusEntered	td	ROLE_TABLECELL	192.168.1.110/accesibilidad2/otros/NVDAyTablas.html	2019-07-17 20:19:41.875000	0	12
29	2019-07-17 20:19:41.906000	gainFocus	input	ROLE_EDITABLETEXT	192.168.1.110/accesibilidad2/otros/NVDAyTablas.html	2019-07-17 20:19:41.891000	0	12
30	2019-07-17 20:19:47.617000	typedCharacter	input	ROLE_EDITABLETEXT	192.168.1.110/accesibilidad2/otros/NVDAyTablas.html	2019-07-17 20:19:41.906000	5	18
31	2019-07-17 20:19:48.201000	event_becomeNavigatorObjec	input	ROLE_EDITABLETEXT	192.168.1.110/accesibilidad2/otros/NVDAyTablas.html	2019-07-17 20:19:47.617000	0	19
32	2019-07-17 20:19:48.655000	focusEntered	tr	ROLE_TABLEROW	192.168.1.110/accesibilidad2/otros/NVDAyTablas.html	2019-07-17 20:19:48.201000	0	19
33	2019-07-17 20:19:48.687000	focusEntered	td	ROLE_TABLECELL	192.168.1.110/accesibilidad2/otros/NVDAyTablas.html	2019-07-17 20:19:48.655000	0	19
34	2019-07-17 20:19:48.702000	gainFocus	input	ROLE_EDITABLETEXT	192.168.1.110/accesibilidad2/otros/NVDAyTablas.html	2019-07-17 20:19:48.687000	0	19
35	2019-07-17 20:19:53.843000	typedCharacter	input	ROLE_EDITABLETEXT	192.168.1.110/accesibilidad2/otros/NVDAyTablas.html	2019-07-17 20:19:48.702000	5	24
36	2019-07-17 20:19:54.429000	event_becomeNavigatorObjec	input	ROLE_EDITABLETEXT	192.168.1.110/accesibilidad2/otros/NVDAyTablas.html	2019-07-17 20:19:53.843000	0	25

Tabla 7: – Reporte de Eventos de NVDA

La tabla 3 contiene un fragmento de la secuencia de eventos que se produce al navegar el formulario descripto en el ejemplo de la figura 5. En la fila N.º 19 se indica que un evento event_becomeNavigatorObjec, se produjo a las 0:19:35.763000 horas del día 2019-07-17, en la URL 192.168.1.110/accesibilidad2/otros/NVDAyTablas.html y sobre un elemento del tipo Input vinculado al rol que en **NVDA** se describe como ROLE_EDITABLETEXT.

Esto muestra que el suceso es un evento de preparación para la navegación del foco hacia un elemento de campos de entrada de tipo texto sobre la interfaz. Si tenemos en cuenta que el evento previo, se produjo en la misma fecha pero a las 20:19:35.188000 horas, es posible determinar el tiempo trascurrido entre ambos, que al ser menor a 1 segundo se indica con el valor 0 como el tiempo transcurrido desde el último evento.

En la novena columna se muestran los segundos transcurridos desde el inicio de la detección de los eventos hasta el instante en el que el evento sucede y es equivalente al tiempo total acumulado desde el comienzo de la captura hasta el instante en el que se produce cada evento.

Así, la secuencia continua, con dos eventos focusEntered, en las filas 20 y 21, sobre un elemento div, y un elemento label respectivamente, que resultan ser previos al evento gainFocus en la línea 23. Es recién en este último caso cuando se produce realmente la llegada del foco sobre el elemento visible en la interfaz, que resulta ser un campo de texto de entrada descripto como <Input> con ROLE_EDITABLETEXT.

Los eventos focusEntered requieren tiempo a la aplicación y pueden dar lugar a síntesis de texto como en el caso del posicionamiento de filas y columnas dentro de una tabla HTML de la figura 5.

Es posible categorizar a estos eventos focusEntered como no focables, dado que no producen el movimiento del foco sobre la presentación de la página y al ser previos a la llegada visual del mismo al elemento de la interfaz, son transparentes al observador y no pueden ser detectados visualmente.

Continuando con el análisis, el evento typedCharacter de la línea 23 indica que se produjo una entrada por teclado, pero con la particularidad que desencadena un evento del tipo event_becomeNavigatorObjec, lo que indica que se ha ingresado una orden de navegación al siguiente elemento de la interfaz presionando la tecla de tabulación.

Al igual que en el evento de la fila 23, los eventos focusEntered listados entre las filas 25 y 28, se producen sobre elementos HTML de formulario (form), tabla(table), fila(tr) y columna(td), en un momento previo a que el elemento de entrada de tipo texto sobre la presentación reciba realmente el foco en la fila 29.

Como se observa en la tabla 3, cada uno de estos eventos no focables aunque requieren tiempo a la aplicación tienen una duración inferior a 1 segundo. Aunque resulten útiles para controlar la ejecución del programa su incidencia en la estimación de los parámetros no es significativa.

Debido a la imposibilidad de relevar valores para los parámetros mediante observación y la complejidad del proceso de cálculo requerida, se recurrió a los datos del funcionamiento interno de **NVDA.**

Evaluando los comportamientos de los usuarios programáticamente y automatizando las mediciones se obtuvieron valores confiables para ser utilizados como parámetros en los algoritmos de captura de los **Accessibility Event.**

Se desarrolló un complemento para **NVDA,** que registra las acciones de los usuarios recuperando datos del contexto sobre la interfaz. Este software agrupa eventos para obtener empíricamente datos sobre el comportamiento durante la navegación, mientras se ejecutan rutinas sobre un sitio web. Los datos obtenidos durante estas interacciones fueron utilizados para estimar los parámetros de cada evento de accesibilidad.

5.3 Software para Recolección de Datos Empíricos

El sistema de estimación se compone de:

1. **Snippet incrustado** con la función de:

   Configuración: Incorpora datos a la página web relacionados con el despliegue del sistema, endpoint procesamiento de datos y funciones complementarias.

   Autenticación de usuarios: Identifica mediante un Token a los usuarios que están interactuando con la aplicación.

   Marcaje de elemento: Para posibilitar la identificación de los elementos afectados por los eventos en el **buffer virtual** al utilizar **NVDA** en modo revisión.

   Detección de eventos**:** Envío de datos sobre los eventos a una aplicación API Rest para ser almacenados en una base de datos.

2. **API REST:** Una Api programada en PHP que procesa los datos reportados desde el **Snippet** y desde un plugins **NVDA** y genera reportes sobre los eventos descartando valores atípicos.

3. **Base de datos en PostgresSQL**: para resguardo de datos.

4. **NVDA-Accessibility-Parameters:** Complemento instalable que extiende **NVDA** para reportar los datos necesarios para la estimación de los parámetros**.**

5.4 Snippet incrustado

Consiste en un script Javascript que se incrusta en el sitio Web y contiene métodos que definen objetos encargados de la captura de los datos.

Fuerza la identificación de cada usuario al utilizar el sitio Web con el objeto de facilitar el agrupamiento de los datos y hacer uso de la función de cálculo del Xpaht definida en **Usability Event** en **Kobold** (Grigera, 2017).

5.5 NVDA-Accessibility-Parameters

El complemento se basa en el modelo de eventos interno de **NVDA** para detectar las interacciones de los usuarios en el buffer virtual. Al producirse un evento, **NVDA** determina si requiere propagarlo hacia la aplicación de destino final, o si solo debe realizar alguna acción ligada a sus funcionalidades internas y/o producir alguna salida a través del sintetizador de voz incorporado al software.

En el caso del Navegador Mozilla Firefox algunos eventos sobre el screen reader tienen efectos en la representación del documento Web contenida en el **buffer virtual.** Tal es el caso de la navegación del próximo encabezado de Nivel 1 que además produce como salida de síntesis de voz el texto contenido en el elemento navegado.

Por otra parte, cada elemento de la interfaz presentada en Mozilla Firefox es representado por un objeto predefinido que forman parte de la clasificación de roles que les asigna **NVDA** dentro de su jerarquía interna. Por ejemplo, un enlace HTML contiene un valor de campo Role=ROLE_LINK en la descripción de su atributo en **NVDA**.

NVDA contiene una limitada lista de posibilidades de detección de interacciones sobre el screen reader, a partir de las cuales se pueden derivar acciones más complejas como **Accessibility Event:**

N°	Evento	Se produce Cuando
1	event_gainFocus	Cuando un elemento recibe el foco.
2	event_focusEntered	Cuando un elemento no focable en la interfaz, como por ejemplo un contenedor, incluye un elemento que recibe el foco.
3	event_becomeNavigatorObject	Al convertir un objeto en navegador de objeto.
4	event_typedCharacter:	Al presionarse una tecla sobre un elemento.
5	event_mouseMove	Al mover el mouse sobre la página.
6	event_caret	Al mover el cursor entre elementos en la página.
7	event_valueChange	Al cambiar un valor de un elemento editable.
8	event_stateChange	Al cambiar un estado de un elemento editable.

*Tabla 8: Eventos de interacción internos del Screen Reader **NVDA**.*

A partir de este modelo el complemento captura los gestos de entrada utilizados de manera estándar en el modo revisión y detecta los eventos de navegación directa mediante combinación de teclas.

5.6 Tratamiento de los Valores Atípicos (*outlier*):

Los datos obtenidos por el sistema de estimación pueden contener valores atípicos como consecuencia de alguna interrupción en la interacción. Esta situación puede tener diferentes orígenes, como consecuencia de una distracción del usuario por algún estímulo externo, como una llamada telefónica.

Para subsanar la situación, se utiliza el método de test de Tukey para descartar todos los datos atípicos leves en el cálculo de los estimadores. Es decir que dado un conjunto Q de valores muéstrales para la estimación y los valores Q1, Q3 y RI, donde:

Q1: Es el valor que corresponde al primer cuartil del conjunto de muestras.

Q3: Es el valor que corresponde al tercer cuartil del conjunto de datos del evento.

RI: Es el rango Intercuartílico, donde RI=Q3-Q1.

Se descartan todos los valores del parámetro q tal que:

$$Q3-1,5*RI< q <Q1-1,5*RI$$

5.7 Experiencia

El procedimiento para la estimación de los valores para los parámetros consistió en sesiones de capturas de eventos, durante las cuales un grupo de usuarios con dificultades visuales, utilizando el teclado con **NVDA** y el complemento **NVDA-Accessibility-Parameters**, realizaron una serie de tareas sobre una interfaz Web.

Entre las posibilidades en la elección de la aplicación web para las estimaciones se priorizó que fueran de código abierto, acceso libre y con un grado de dificultad en la accesibilidad, que permita su utilización por parte de los usuarios con dificultades visuales.

Debido a que algunos software presentan un alto grado de dificultad para los usuarios y fueron diseñados para ser utilizados por medio del mouse, fueron descartados. Este es el caso del inventario informático GLPI.

Las aplicaciones utilizadas incluyeron las páginas Web libres Bookmedik y VrtualMart que resultaron convenientes para llevar adelante la recolección de las muestras.

Bookmedik es un sistema para llevar el control de citas médicas, pacientes, médicos, historiales y citas para áreas médicas utilizados por centros médicos, clínicas y médicos independientes. El software se encuentra disponible para descargar en github: https://github.com/evilnapsis/bookmedik. Utiliza como espacio de almacenamiento bases de datos Mysql y está escrita en PHP.

VirtualMart es un complemento para el gestor de contenidos Joomla utilizado para la puesta en marcha de tiendas de comercio electrónico. Desarrollada en PHP, se encuentra disponible en https://www.virtuemart.net/download y requiere un entorno de base de datos Mysql o postgres para almacenamiento.

Las aplicaciones fueron desplegados sobre un servidor dedicado para realizar el experimento del cual participaron 5 usuarios de entre 18 y 25 años, a los que se les explicó verbalmente las tareas que debían realizar en cada aplicación. Luego de proveerles auriculares a cada uno, con el fin de evitar incidencias del sonido ambiente en la comunicación, se les permitió configurar el screen reader **NVDA** con la velocidad de reproducción de su preferencia.

A cada participante se le asignó un nombre de usuario y una contraseña, que el sistema de detección les solicita automáticamente antes de compensar a utilizar las aplicaciones y que se utiliza internamente para el agrupamiento de datos.

Las tareas solicitadas a cada participante consistieron en:

En el caso de BookMedik

- Iniciar la sesión de usuario.
- Agregar un Profesional médico.
- Agregar un paciente.
- Agregar una especialidad Médica.
- Asignar una especialidad médica al profesional creado en el paso anteriormente.
- Coordinar una cita médica ente un profesional y un paciente.
- Generar un reporte de citas a las que no asistieron los pacientes.

En VirtulaMart

- Registrase como usuario de la página.
- Buscar un producto de informática en la tienda virtual.
- Cargar en su carro de compras el producto.
- Verificar sus pedidos.
- Confirmar la compra.
- Registra un fabricante y una forma de pago.
- Cambiar el idioma de la aplicación.

Aunque no todas estas tareas pudieron ser completadas por los usuarios, las mismas generaron datos suficientes para realizar las estimaciones.

5.8 Detección de Interacción en la Web

La estructura de **Snippet** relacionada con la detección de eventos se compone de objetos encargados de recolectar la información para las estimaciones. Estos objetos detectan las acciones de los usuarios y las remiten al componente API REST.

1. **WaitingTime**:

Reporta a la **API REST** los valores de los tiempos de permanencia del foco en los elementos de la interfaz que afectan a los **Accessibility Event Electronic text for non-significant speech synthesis, Electronic text for non-existent speech synthesis y Confused speech synthesis.**

Para estos eventos se requieren estimar los parámetros **MinimumWaitingTime y MaximumWaitingTime** para un conjunto particular de elementos de la interfaz.

Estos elementos HTML son los campos de entrada de texto, botones de chequeo, botones de radio, enlaces, botones estándar y submit de formularios.

Medición de Valores WaitingTime

Cuando el foco abandona un elemento en la página web se remite al servidor el tiempo transcurrido desde que el foco ingreso al elemento hasta que el mismo lo abandona. Los reportes contienen los siguientes valores:

Campo	Descripción	Ejemplo
Time	Instante en que se produce el evento	17/9/2019 18:19:08
Parámetro	Objeto que reporta el dato	WaitingTime
xpaht	Identificador del elemento afectado	//*[@id="lastname"]
Tag	Etiqueta que describe al elemento	INPUT
Type	Tipo de elemento afectado	text
waitingTime	Tiempo en milisegundos de permanencia del foco en el elemento	1544
URL	Dirección de la página que contiene el elemento	http://192.168.1.110/bookmedik/index.php?view=newmedic
Token	Valor de autenticación criptográfico correspondiente al usuario que genero el evento	779dc6059352002d35f35235eaefcab3

*Tabla 9: Datos reportados durante las interacciones por el objeto **WaitingTime**.*

Estimación con WaitingTime:

Dado que el valor de los umbrales de **WaitingTime** se deben calcular para cada tipo de elemento afectado E (enlace, text, button, submit, checkbox y radio) se establece para cada elemento E como valores mínimos y máximos del umbral **WaitingTime** los valores del límite inferior y superior del intervalo de confianza para la media muestral de los valores de la variable **WaitingTime** para ese elemento E luego de descartados los valores atípicos. De tal manera que

los pasos del procedimiento para cada tipo particular de elemento afectado E (enlace, text, button, submit, checkbox y radio), fueron:

1. Descartar los valores **outlier** por el método de Tukey, para la variable **WaitingTime** correspondiente al elemento afectado E.

2. Agrupar lo valores por cada Token, es decir por cada usuario, para la variable **WaitingTime** correspondiente al elemento afectado E.

1. Calcular el promedio de la variable **WaitingTime** para cada usuario correspondiente al tipo de elemento afectado E: **WaitingTime_promedio (E).**

2. Para el tipo de elemento afectado E, calcular **WaitingTime_Estimado(E)** como un estimador para la media de la variable **WaitingTime,** como el promedio de valores obtenidos en el paso anterior promedio de **WaitingTime_promedio(E)**.

3. Obtener un intervalo de confianza del 90% para la media, de cada **WaitingTime_Estimado(E)** del elemento E, estableciendo los valores del límite inferior y límite superior del intervalo como los valores para el umbral de WaitingTime.

Es decir que se obtuvieron intervalos de confianza para los estimadores:

Waiting_Estimado(Enlace): WaitingTime para un elemento HTML de tipo enlace (A)**.**

Waiting_Estimado(Text): WaitingTime para un elemento HTML de entrada de texto (INPUT, Type=Text).

Waiting_Estimado(Checkbox): WaitingTime para un elemento HTML de entrada de botón de chequeo (INPUT, Type=checkbox).

Waiting_Estimado(RadioButton): WaitingTime para un elemento HTML de entrada de botón de radio (INPUT, Type=radio)

Waiting_Estimado(Submit): WaitingTime para un elemento HTML de entrada de botón de envío de formulario (INPUT, Type=submit).

Waiting_Estimado(Button): WaitingTime para un elemento HTML de entrada de botón de interacción (INPUT, Type=button).

Valores Obtenidos para los Parámetros:

		Parámetro		
Eventos	**Elemento**	**MinimumWaitingTime**	**MaximumWaitingTime**	**Unidad de Medida**
Electronic text for non-significant speech synthesis Electronic text for non-existent speech synthesis Confused speech synthesis.	Texto	**4,179**	**7,813**	Seconds
	RadioButton	**5,097**	**8,351**	
	Checkbox	**3,948**	**6,360**	
	Submit	**1,940**	**3,710**	
	Button	**3,177**	**5,635**	
	Enlace	**3,957**	**6,351**	

Tabla 10: Valores para los parámetros de los ***Accessibility Event*** **.**

2. HighFrequencyTab:

Reporta a la **API REST** los valores para estimar los parámetros del **Accessibility Event High Frequency Of Use Of The Tab Key.**

Para el evento se requieren estimar los parámetros **MinimumSteps, MaximumScrollingTime y DwellingTime** donde el evento afecta a las URLs para las cuales es necesario estimar los parámetros.

Medición de Valores

Durante las interacciones se contabilizan las entradas de navegación con el botón de tabulación que efectúa el usuario hasta que finalmente realiza una entrada de datos, momento en el que se remite al servidor un reporte conteniendo los siguientes valores:

Campo	Descripción	Ejemplo
Time	Instante en que se produce el evento	17/9/2019 18:18:47
Parámetro	Objeto que reporta del dato	HighFrequencyTab
xpaht	Identificador del último elemento accedido en la secuencia	html/body/div/div/div/ div/div[2]/form/fieldset/ div[2]/input
scrollingTi me	Tiempo total del scrolling con tecla tab en milisegundos	6541
steps	Cantidad de pasos de tabulación	3
TimeEvent	Tiempos trascurridos entre entradas de tabulación	1202,3196,1375
URL	Dirección de la página afectada por el evento	http://192.168.1.110/ bookmedik/index.php? view=newmedic;

Tabla 11: Datos reportados durante las interacciones por el objeto ***HighFrequencyTab***

Estimación con HighFrequencyTab:

En este caso es necesario calcular tres estimadores para los parámetros **MinimumSteps**, **MaximumScrollingTime** y **DwellingTime**; donde cada evento de interacción que es reportado contiene un conjunto de valores correspondiente a los tiempos entre acciones de tabulación sucesivas almacenados en el campo TimeEvent. En función del tiempo total de duración del evento y el número de tabulaciones realizadas, es posible calcular valores promedios para los tiempos entre las entradas y utilizarlos para la estimación del valor del parámetro **DwellingTime**.

Se puede apreciar esta situación en el evento de la tabla 11, cuyo tiempo total del evento es de 6541 milisegundos durante los cuales se realizaron 3 acciones de tabulación, obteniendo un tiempo medio de DwellingTime de 2180 milisegundos.

Para cada evento se dispone de valores de las variables Steps, ScrollingTime y DwallingTime, siendo posible calcular un intervalo de confianza para la media muestral de cada una luego de descartar los valores atípicos.

Los pasos del procedimiento, para los valores de las variables Steps, ScrollingTime y DwallingTime fueron:

1. Descartar los valores **outlier** por el método de Tukey.
2. Agrupar lo valores por cada Token, es decir por cada usuario.
3. Calcular el promedio de los valores de las variables Steps, ScrollingTime y DwallingTime por cada usuario.
4. Calcular el estimador para las medias de los valores de las variables Steps, ScrollingTime y DwallingTime, como el promedio de los valores obtenidos en el paso anterior.
5. Obtener un intervalo de confianza del 90% para la media de las variables Steps, ScrollingTime y DwallingTime estableciendo los valores de los parámetros como a continuación:

   MinimumSteps: El valor entero igual o menor más próximo al límite inferior del intervalo de confianza del 90% para la media del Steps.

   MaximumScrollingTime: Límite superior del intervalo de confianza del 90% para la media del scrollingTime.

   DwellingTime: Límite superior del intervalo de confianza del 90 % para la media del DwellingTime de los usuarios.

Valores Obtenidos para los Parámetros:

Evento	Parámetros	Valor	Unidad de Medida
High Frequency Of Use Of The Tab Key.	**MinimumSteps**	**4,03 ≈ 4**	Steps
	MaximumScrollingTime	**22,58**	Seconds
	DwellingTime	**4,83**	Seconds

*Tabla 12: Valores para los parámetros del **Accessibility Event High Frequency Of Use Of The Tab Key.***

3. NavigationPath

Reporta a la **API REST** los valores para estimar los parámetros del **Accessibility Event Navigation Path Version Accessibility.**

Para el evento se requieren estimar los parámetros **ParamMinimumNavigations y paramMaximumTime** donde el evento afecta a las URLs para las cuales es necesario estimar los parámetros.

Medición de Valores

Se almacena la secuencia de navegación entre páginas mediante linck sin que los usuarios realicen ninguna otra acción de entrada en las páginas. Cuando la secuencia finaliza con una acción sobre un elemento de una página, se remite al servidor un reporte conteniendo los siguientes valores:

Campo	Descripción	Ejemplo
Time	Instante en que se produce el evento	17/9/2019 18:33:29
Parámetro	Objeto que reporta el dato	NavigationPath
URL	Dirección de la página afectada por el evento	http://192.168.1.110/ bookmedik/ index.php? view=newmedic
WaitingTime Page	Tiempo total en milisegundos de duración del evento	87030
numberOfNo des	Cantidad de páginas de la secuencia	5

*Tabla 13: Datos reportados durante las interacciones por el objeto **NavigationPath**.*

Estimación con NavigationPath:

Es necesario calcular dos estimadores para los parámetros **ParamMinimumNavigations** y **paramMaximumTime**, valores que se asocian a los datos contenidos en los campos de las variables numberOfNodes y WaitingTimePage de cada evento.

Con estos valores es posible calcular, para cada una de las variables un intervalo de confianza para la media muestral de los valores para los usuarios luego de descartar los valores atípicos y obtener valores para los parámetros. De tal manera que los pasos del procedimiento para numberOfNodes y WaitingTimePage, fueron:

1. Descartar los valores **outlier** por el método de Tukey.
2. Agrupar lo valores por cada Token, es decir por cada usuario.
3. Calcular el promedio de los valores de las variables numberOfNodes y WaitingTimePage de cada usuario.

4. Calcular el estimador para la media de las variables numberOfNodes y WaitingTimePage**,** como el promedio de valores obtenidos en el punto anterior.

5. Obtener un intervalo de confianza del 90% para la media de cada valor numberOfNodes y WaitingTimePage estableciendo los valores de los parámetros como a continuación:

   ParamMinimumNavigations: El valor entero igual o menor más próximo al límite inferior del intervalo de confianza del 90% para la media del numberOfNodes.

   ParamMaximumTime: Límite superior del intervalo de confianza del 90% para la media del WaitingTimePage.

   Valores Obtenidos para los Parámetros:

Evento	Parámetros	Valor	Unidad de Medida
Navigation Path Version Accessibility	**ParamMinimumNavigations**	**2,39 ≈ 2**	Nodos
	paramMaximumTime	**80,64**	Seconds

Tabla 14: Valores para los parámetros del ***Accessibility Event Navigation Path Version Accessibility.***

4. FastScrollingWithKeyboard

Reporta a la **API REST** los valores para estimar los parámetros del **Accessibility Event FastScrollingWithKeyboard.**

Para el evento se requiere estimar los parámetros **MinimumSteps, MaximumScrollingTime y DwellingTime** donde el evento afecta a las URLs para las cuales es necesario estimar los parámetros.

Medición de Valores

Durante la navegación se almacenan las secuencias de desplazamiento con barra espaciadora que se producen en la página hasta que el usuario realiza una acción diferente. Cuando la secuencia finaliza se remite al servidor un reporte conteniendo los siguientes valores:

Campo	Descripción	Ejemplo
Time	Instante en que se produce el evento	17/9/2019 19:10:05
Parámetro	Objeto que reporta el dato	FastScrollingWithKeyboard
Steps	Cantidad de acciones de desplazamiento realizadas por el usuario	5
StartingTop	Posición del Top de la página al inicio del desplazamiento vertical	251.25
ScrollFinal	Posición del Top de la página al final del desplazamiento vertical	810
TimeStampInicial	Instante de inicio del evento	1568758202020
TimeStampFinal	Instante de finalización del evento	1568758205105
Duración	Tiempo durante el cual se produce el desplazamiento vertical	3085
TiempoEntreEvento	Tiempo en milisegundos trascurrido entre cada acción de desplazamiento	0,151,313,402,562
URL	Dirección de la página afectada por el evento	http://192.168.1.110/AccessibilityParametros/15%20-%20NavigationBetweenListsAndLink/15%20-%20NavigationBetweenListsAndLink.php

*Tabla 15: Datos reportados durante las interacciones por el objeto **FastScrollingWithKeyboard**.*

Estimación con FastScrollingWithKeyboard:

Es necesario calcular tres estimadores **MinimumSteps, MaximumScrollingTime y DwellingTime,** valores que se asocian a los datos de los campos que contienen las variables Steps, Duración y TiempoEntreEvento respectivamente. En el caso de TiempoEntreEvento se obtiene para cada evento un valor representativo de DwellingTime como el promedio de los valores de TiempoEntreEvento, que corresponden a los tiempos entre las pulsaciones de barra espaciadora.

Una vez de descartar los valores atípicos, es posible calcular para cada una de las variables un intervalo de confianza para la media muestral y obtener valores para los parámetros. De tal manera que los pasos del procedimiento, fueron:

1. Descartar los valores **outlier** por el método de Tukey.
2. Agrupar lo valores por cada Token, es decir por cada usuario.
3. Calcular para cada variable el promedio de los valores de Steps, Duración y DwellingTime de cada usuario.
4. Calcular el estimador para la media de las variables Steps, Duración y DwellingTime**,** como el promedio de valores obtenidos en el paso anterior.
5. Obtener un intervalo de confianza del 90% para la media de cada variable Steps, Duracióny DwellingTime estableciendo los valores de los parámetros como a continuación:

   MinimumSteps: El valor entero igual o menor más próximo al límite inferior del intervalo de confianza del 90 % para la media de la variable Steps.

   MaximumScrollingTime: Límite superior del intervalo de confianza del 90% para la media de la variable Duración.

   DwellingTime: Límite superior del intervalo de confianza del 95% para la media de la variable DwellingTime de los usuarios.

Valores Obtenidos para los Parámetros:

Evento	Parámetros	Valor	Unidad de Medida
FastScrollingWithKeyboard	MinimumSteps	**2**	Step
	MaximumScrollingTime	**4,65**	Seconds
	DwellingTime	**2,139**	Seconds

Tabla 16: Valores para los parámetros del ***Accessibility Event FastScrollingWithKeyboard.***

Detección de Interacciones para Eventos en el Modo Revisión de NVDA

Para la estimación de los parámetros para los eventos que se producen en el **buffer virtual** de **NVDA** el complemento **NVDA-Accessibility-Parameters** detecta las interacciones de los usuarios y ante un **Accessibility Event Flash Scrolling in Accesibilidad with NVDA add-on**

o Navigation Between Lists of Links for the NVDA add-on remite al servidor los datos necesarios para la estimación de los parámetros**.**

Durante la navegación de los objetos en el modo revisión el complemento almacena las secuencias de eventos de interacción hasta tanto se ingresan datos de entrada por teclado vinculados a una interacción directa sobre la interfaz. Es decir hasta que las acciones dejan de ocurrir durante el modo revisión en el **buffer virtual** y se producen directamente sobre la página web, momento en el que se reporta la secuencia de eventos de navegación realizadas por el usuario a la **API REST**.

El complemento contiene dos objetos principales **AppModule** y **Logger**. El primero se encarga de la interfaz de entrada modelada mediante eventos de **NVDA**, detecta los gestos ingresados por los usuarios como combinaciones de teclas y propaga las acciones que se ejecutan sobre la aplicación. Además se comunica con el objeto **Logger** remitiendo datos sobre el contexto de las acciones del usuario.

Campo	Descripción	Ejemplo
time	Instante en que se produce el evento de interacción	17/9/2019 20:10:05
shift	Valor que indica si el usuario presiono la tecla Shift	false
gesture	Gesto de entrada que da lugar al evento	l
xpath	Identificador del elemento Html destino de la navegación en el buffer virtual	/html/body/div[9]
url	Url de la página donde se produjo el evento	http://192.168.1.110/ AccessibilityParametros/15%20-%20NavigationBetweenListsAndLink/15%20-%20NavigationBetweenListsAndLink.php
dirección	Indica si la navegación es del siguiente elemento o del elemento previo	next
children	Cantidad de Nodos Hijo de elemento Html objeto de la navegación en el buffer vitual	4 (solo aplica a los eventos **Navigation Between Lists of Links for the NVDA add-on)**

Tabla 17: Datos que remite el objeto ***AppModule a Logger***

Por su parte el objeto **Logger** almacena los eventos de interacción, clasificándolos en eventos de **FlasScrollingWithNVDAadd-on** o **ListLinckforNVDAadd-on** que luego agrupa en **Accessibility Event** y remite a la **API REST** para su procesamiento. Los datos enviados a la **API REST** consisten la siguiente información:

Campo	Descripción	Ejemplo
fechaHora	Instante en el que se finaliza el evento	2019-12-13 11:14:26
parameterName	Identificador del evento	FlasScrollingWithNVDAadd-on
first_event	Primera navegación (evento de interacción) en el modo revisión del evento de accesibilidad	Objeto json conteniendo datos remitidos por el objeto AppModule como se discribe en la tabla 11
step	Cantidad de interacciones de navegación en modo revisión	4
Interaction_Event	Conjunto de interacciones de navegación en modo revisión	Conjunto Objeto json conteniendo datos remitidos por el objeto AppModule como se discribe en la tabla 11
url	URL de la página en la que se produjo la navegación	http://192.168.1.110/ AccessibilityParametros/15%20-%20NavigationBetweenListsAndLink/15%20-%20NavigationBetweenListsAndLink.php
token	Identificación del Usuario que generó el evento	d204dbe03bae6f041124bdf0f8881fda

Tabla 18: Datos que remite el objeto ***Logger*** *a la* ***ApiRest***

5 Flash Scrolling in Accesibilidad with NVDA add-on

En este caso el complemento **NVDA** reporta a la **API REST** los datos para estimar los parámetros **MinimumSteps, MaximumScrollingTime y DwellingTime** donde el evento afecta a las URLs para las cuales es necesario estimar los parámetros.

Medición de Valores

Cuando la **API REST** recibe los datos realiza dos acciones complementarias: Calcula la variable Duración de cada **Accessibility Event Flash Scrolling in Accesibilidad with NVDA add-on** en función de sus valores de campo fechaHora, que corresponde al instante final del evento y el valor correspondiente al tiempo de registro contenido en el campo first_event que corresponde al instante inicial del **Accessibility Event**.

Calcula el tiempo promedio trascurrido entre los eventos de interacción contenidos en el campo Interaction_Event de cada **Accessibility Event Flash Scrolling in Accesibilidad with NVDA** con el fin de obtener un valor representativo para la variable DwellingTime del evento.

Estimación para Flash Scrolling in Accesibilidad with NVDA:

En este caso es necesario calcular los estimadores para los parámetros **MinimumSteps, MaximumScrollingTime y DwellingTime.** Para cada **Event Flash Scrolling in Accesibilidad with NVDA add-on** se asocian a la variable Steps con el estimador de **MinimumSteps** y los dos siguientes **MaximumScrollingTime y DwellingTime** con los valores calculados por la **API REST** para Duración y DwellingTime.

Luego es posible calcular, para cada una las variables Steps, Duración y DwellingTime, un intervalo de confianza para la media muestral de los valores de los usuarios luego de descartar los valores atípicos y obtener valores para los parámetros. De tal manera que los pasos del procedimiento, fueron:

1. Descartar los valores **outlier** por el método de Tukey.
2. Agrupar lo valores por cada Token, es decir por cada usuario.
3. Calcular para cada variable el promedio de los valores de Steps, Duración y DwellingTime de cada usuario.
4. Calcular el estimador para la media de las variables Steps, Duración y DwellingTime**,** como el promedio de valores obtenidos en el paso anterior.
5. Obtener un intervalo de confianza de 90% para la media de cada variable Steps, Duración y DwellingTime estableciendo los valores de los parámetros como a continuación:

   MinimumSteps: El valor entero igual o menor más próximo al límite inferior del intervalo de confianza del 90 % para la media de la variable Steps.

   MaximumScrollingTime: Límite superior del intervalo de confianza del 90% para la media de la variable Duración.

   DwellingTime: Límite superior del intervalo de confianza del 90% para la media de la variable DwellingTime de los usuarios.

Valores Obtenidos para los Parámetros:

Evento	Parámetros	Valor	Unidad de Medida
Flash Scrolling in Accesibilidad with NVDA.	MinimumSteps	**2,71≈ 2**	Steps
	MaximumScrollingTime	**16,99**	Seconds
	DwellingTime	**5,33**	Seconds

*Tabla 19: Valores para los parámetros del **Accessibility Event Flash Scrolling in Accesibilidad with NVDA add-on**.*

6 Navigation Between Lists of Links for the NVDA add-on

En este caso se reportan a la **API REST** los datos para estimar los parámetros **MinimunChildren y MinimunSteps, MaximumScrollingTime y DwellingTime** donde el evento afecta a las URLs para las cuales es necesario estimar los parámetros.

Medición de Valores

La **API REST** recibe los datos y realiza las siguientes acciones complementarias:

1. Calcula la variable Duración de cada **Accessibility Event Navigation Between Lists of Links for the NVDA add-on** en función de sus valores de campo fechaHora, que corresponde al instante final del evento y el valor correspondiente al tiempo de registro contenido en el campo first_event que corresponde al instante inicial del **Accessibility Event**.
2. Calcula el tiempo promedio trascurrido entre los eventos de interacción contenidos en el campo Interaction_Event de cada **Accessibility Event Navigation Between Lists of Links for the NVDA add-on** con el fin de obtener un valor representativo para la variable DwellingTime.
3. Calcula el valor de Children de cada **Accessibility Event Navigation Between Lists of Links for the NVDA add-on** con el promedio de los valores de los campos children contenido en cada evento de interacción dentro de Interaction_Event.

Estimación para Navigation Between Lists of Links for the NVDA:

Para la estimación de **MinimunChildren y MinimunSteps, MaximumScrollingTime y DwellingTime** es posible asociar los valores de las variables Steps del evento de accesibilidad con el estimador del parámetro **MinimumSteps,** los valores calculados por la **API REST** para la variable Duración con el estimador de **MaximumScrollingTime,** los de la variable Children con el

estimador de **MinimunChildren** y los de la variable DwellingTime con el estimador del parámetro **DwellingTime.**

A partir de esto es posible calcular, para cada una de estas variables, un intervalo de confianza para la media muestral de los usuarios, luego de descartar los valores atípicos y obtener valores para los parámetros. De tal manera que los pasos del procedimiento, fueron:

1. Descartar los valores **outlier** por el método de Tukey.
2. Agrupar lo valores por cada Token, es decir por cada usuario.
3. Calcular para cada variable el promedio de los valores de Steps, Duración, Children y DwellingTime de cada usuario.
4. Calcular el estimador para la media de las variablesSteps, Duración, Children y DwellingTime**,** como el promedio de valores obtenidos en el paso anterior.
5. Obtener un intervalo de confianza de 90% para la media de cada variable Steps, Duración, Children y DwellingTime estableciendo los valores de los parámetros como a continuación:

   MinimumSteps: El valor entero igual o menor más próximo al límite inferior del intervalo de confianza del 90 % para la media de Steps.

   MinimunChildren: El valor entero igual o menor más próximo al límite inferior del intervalo de confianza del 90 % para la media de Children.

   MaximumScrollingTime: Límite superior del intervalo de confianza del 90 % para la media de Duración.

   DwellingTime: Límite superior del intervalo de confianza del 90% para la media del DwellingTime de los usuarios.

Valores Obtenidos para los Parámetros:

Evento	Parámetros	Valor
Navigation Between Lists of Links for the NVDA	**MinimumSteps**	**2,32≈2**
	MaximumScrollingTime	**24,54**
	DwellingTime	**5,96**
	MinimunChildren	**1,84 ≈1**

Tabla 20: Valores para los parámetros del ***Accessibility Event Flash*** **Navigation Between Lists of Links for the NVDA**

Capítulo VI

Complemento NVDA

La captura de cada evento de accesibilidad se define en función de las interacciones que lo componen y la tecnología de asistencia. Al utilizar el teclado junto con el lector de pantalla **NVDA** se producen eventos que solo pueden ser detectados a partir de acciones sobre el **buffer virtual**.

En este capítulo se describe la arquitectura, y tipos de extensiones disponibles para desarrollar complementos **NVDA**.

6. 1 Captura de Eventos y NVDA - Modo Navegación de Objetos

En Kobold para accesibilidad, los sucesos del grupo de *Eventos de Accesibilidad del Modo Revisión de NVDA* se capturan mediante un complemento NVDA. Estos eventos son el resultado de una secuencia de interacciones sobre una representación virtual de la presentación web, que no es accesible desde el objeto incrustado Javascript LoggerAccesibility.

La navegación de objetos por teclas de acceso rápido facilitan navegar directamente los elementos de una página sin recorrerlos secuencialmente. Estas órdenes de entradas por teclado producen interacciones no detectables modificando el script incrustado en la página web.

Para su detección fue necesario extender las funciones del screen reader desarrollando un complemento instalable en **NVDA**.

6.2 Complementos NVDA

El desarrollo de complementos para NVDA posibilita extender el comportamiento de la aplicación dotándola de nuevas y/o mejores características.

NVDA es software open source que integra jerarquías de clases y APIs para adicionar paquetes con funciones y controles más específicos. Su arquitectura posibilitó implementar soluciones para detectar eventos vedados al componente javascript.

6.3 Tipos de Extensiones Disponibles

Módulos de aplicación: Son específicos para una aplicación concreta, como por ejemplo Firefox. Reciben eventos destinados a la aplicación ejecutando órdenes ,que el módulo vincula con pulsaciones de teclado y tipos de entradas.

Extensiones globales: Utilizadas para capturar las operaciones sobre todas las aplicaciones. Reciben los eventos para todos los controles del sistema operativo.

Extensiones de controlador: Son específicas para aplicaciones dependiente del hardware (Por ejemplo teclado braille).

En este trabajo fue necesario desarrollar un complemento de Módulo **de Aplicación** específicamente para el Navegador Web Mozilla Firefox.

6.4 Definiciones en el Contexto de NVDA

- Caret: Cursor del sistema.
- Interceptador de árbol: Estructura que permite a **NVDA** utilizar el "árbol" de objetos como si fuese solo uno.
- Foco: Región resaltada de la pantalla donde el sistema establece el foco del cursor.
- Script: Función que se ejecuta en respuesta a una entrada de usuario o alguna orden de comando.
- Entrada de Usuario: Pulsaciones de teclado, manipulación de controles en pantallas braille o toques en pantalla táctil.
- Gesto o gesture: Unidad mínima de entrada. Por ejemplo, la orden de teclado “<control>+<t>”.
- Texto descriptivo: Texto electrónico que los Screen Reader anuncian mediante un sintetizador de voz.
- Evento: Suceso en la aplicación vinculado a un procedimiento que debe ejecutarse ante la presencia de ciertas condiciones.
- Widget: Componente individual en una GUI con el cual interactuar. (ejemplo botones, campos de texto editable, cuadros de lista, etc).

6.5 Scripts y Asociación de Gestos

Los módulos que complementan **NVDA** contienen métodos especiales asociados a unidades de entradas por teclado. Reciben como parámetro un objeto **gesture** que representa entradas del usuario que se vinculan a invocaciones de **scripts** ejecutables.

Los scripts se enlazan en cada módulo mediante una estructura de diccionario con variables de clases que contienen cadenas de gestos. Estas unidades de entrada se representan textualmente mediante códigos separados por el carácter ":", seguido por teclas escalonadas con el carácter "+". Estos códigos indican la fuente o dispositivo utilizado y los nombres de entradas por teclado.

Algunos ejemplos de cadenas identificadoras de gestos son:

- "kb:NVDA+shift+v"
- "br(freedomScientific):leftWizWheelUp"
- "kb(laptop):NVDA+t"

Donde las fuentes de entradas son:

- kb: entrada desde el teclado del sistema.
- br: controles de la pantalla braille.
- ts: pantalla táctil.
- bk: entrada por teclado braille.

6.6 Modos de Navegación con NVDA

Navegando con el Cursor del Sistema

Cuando en una aplicación el foco se posa sobre un objeto que contiene un cursor de edición (cursor del sistema), es posible desplazarse con las flechas de teclado y modificar su contenido editable. En estos casos, **NVDA** anuncia descripciones según el cursor se posicione en los caracteres, palabras, líneas, o en una selección de texto.

A diferencia del modo revisión, que trataremos a continuación, al desplazar el cursor para que las entradas del usuario y los eventos se propaguen sobre la aplicación objetivo, la posición de revisión se actualiza automáticamente emparejándose con el cursor del sistema.

Modos Virtuales

Los buffers virtuales se utilizan para representar las páginas web en un Modo Virtual (**Virtual Buffer**). Son contenidos marcados y navegables con teclas de cursor o atajos de teclado (gestos).

La navegación virtual incluye dos modos de interacción con una página web.

- El Modo Revisión: Es posible desplazarse a lo largo de todo el texto utilizando un cursor virtual que simula al cursor del sistema. En este modo de operación, funcionan todas las teclas de órdenes del cursor del sistema, como leer todo, anunciar formato, órdenes de navegación de tablas, etc.
- El Modo Foco: La interacción es directa sobre los controles (campo de edición, cuadro combinado, botones de opción) utilizando las teclas habituales.

Modo Foco

Es el modo más habitual de navegar por el Sistema Operativo Windows. **NVDA** anuncia el objeto que tiene el foco, y se utilizan las órdenes de teclado <tab> y <shift>+<tab> para desplazarse hacia adelante y atrás entre los controles. Se puede desplegar la barra de menú con <alt> y utilizar las teclas para navegar los elementos.

6.7 Navegando en Modo Revisión

En este caso, al navegar objetos con **NVDA** no es posible capturar los eventos de interacción de teclado utilizando Javascript.

El modo de revisión de objetos, le permite al usuario desplazase y obtener información de los objetos individuales, sin moverse secuencialmente entre ellos. Los objetos se estructuran jerárquicamente agrupados en contenedores, que ofrecen funcionalidad de revisión de los objetos contenidos.

Un objeto en revisión se denomina navegador de objetos, y se puede obtener su información utilizando órdenes de revisión de texto, como su nombre o valor de un campo de entrada.

Estos movimientos del cursor de revisión son independientes del cursor del sistema. Es posible navegar y revisar elementos, manteniendo la posición de edición actual (foco) dentro de la página presentada en el navegador.

La secuencia de órdenes de navegación ofrece un efecto de acceso virtual a los elementos de la página en función de una representación alternativa del contenido denominado **buffer virtual**.

Al completarse la secuencia de navegación o abandonarse el modo de revisión, se sincronizan automáticamente las posiciones de edición en la página y la posición de navegación en el **buffer virtual**. La actualización de la ubicación tiene efecto de focado que se puede observar sobre el navegador web y es detectable por los objetos incrustados Javascript.

Sin embargo todas las interacciones intermedias en el **buffer virtual** entre dos actualizaciones de posición se producen sin ser detectadas.

6.8 Buffers Virtuales

Al procesar documentos complejos, como las páginas web, los navegadores necesitan crear una representación plana en función de la jerarquía de objetos propia del navegador. Estas representaciones dinámicas no son navegables linealmente, de arriba hacia abajo, mediante caret (cursor del sistema). Con idéntico mecanismo los screen reader generan sus propias representaciones y ofrecen funciones para recorrerlas.

En **NVDA** se denomina **buffer virtual** a esas representaciones que por razones de performance se crea mediante código dentro del mismo proceso, evitando los retardos asociados a las consultas externas.

Un **buffer virtual** (**VirtualBuffer**) en **NVDA** se representa mediante una jerarquía de clases, que hereda de la clase base virtualBuffers.VirtualBuffer y son un tipo de **interceptor de árbol** disponible internamente en **NVDA**.

6.9 Entradas de Usuario de Interés en el Modo Revisión

El acceso rápido a objetos en un **VirtualBuffer** se realiza ingresando ordenes de teclado:

Teclas disponibles de acceso: “Evento Navegar hasta el próximo….”

5. h: encabezado.
6. l: lista.
7. i: elemento de lista.
8. t: tabla.
9. k: enlace.
10. n: texto que no es enlace.
11. f: campo de formulario.
12. u: enlace no visitado.
13. v: enlace visitado.
14. e: campo de edición.
15. b: botón.
16. x: casilla de verificación.

17. c: cuadro combinado.
18. r: botón de opción.
19. q: cita.
20. s: separador.
21. m: marco.
22. g: gráfico.
23. d: zona ARIA.
24. o: objeto empotrado.
25. 1 a 6: encabezados HTML de 1 a 6 respectivamente.
26. *En todos los casos, anteponiendo la entrada la tecla <shift> se puede navegar al elemento anterior del mismo tipo. Ej: <shift>+b, navega al botón anterior.*

6.10 Captura de Eventos de Interacción en el Modo Revisión

NVDA utiliza **Buffers Virtuales** para representar documentos de Mozilla Firefox, Microsoft Internet Explorer y Adobe Reader. Estas estructuras se almacenan en el dispositivo del cliente quedando ocultas a Javascript y al resto de las aplicaciones.

Los **eventos virtuales** que suceden en el **buffer virtual,** se producen al navegar por el documento durante el modo revisión con los gestos asociados a las teclas descriptas en la sección anterior.

Sin embargo, en algunas ocasiones, se producen en la aplicación web eventos indirectos sobre la interfaz, a raíz de las acciones del usuario que son reconocidas por el componente Javascript. En estos casos es necesario diferenciar los eventos que se producen en el componente de cliente Javascript que son detectables por **Kobold** y los eventos que debemos detectar en el **buffer virtual** extendiendo **NVDA**.

Un ejemplo es el caso del **evento virtual** “Navegar hasta el próximo encabezado de nivel 1”. El **evento virtual** se produce luego de configurar **NVDA** en modo revisión, mediante las teclas <NVDA (Bloq Mayus)>+<espacio>, y presionar la tecla correspondiente al número 1 <1>.

En ese momento, **NVDA** captura el evento vinculado al ingreso por teclado en el nivel de aplicación y no lo propaga hacia el navegador. En su lugar, desplaza hacia abajo el cursor virtual en el **virtualBuffer** hasta la posición donde se encuentra el próximo encabezado de nivel 1 y dependiendo de que el objeto se encuentra fuera del área cubierta por la pantalla sincroniza la vista con el navegador.

Esta sincronización solo procede de ser necesario un desplazamiento de pantalla. En Javascript se puede detectar con un evento de scroll pero no disponemos de medio para determinar el origen del desplazamiento. En todo caso, resulta necesario detectar la acción del usuario para navegar hasta un elemento particular que posee característica determinadas (en el ejemplo un encabezado de nivel 1).

En estas condiciones la detección de la interacción es dependiente de los aspectos visuales de presentación sobre la pantalla.

Idéntica situación se presenta cuando no se requiere desplazamiento de pantalla. La interacción del usuario “Navegar hasta el próximo encabezado de nivel 1” (presionando la tecla <1> en el modo revisión) quedaría oculta al componente Javascript, debido a que la navegación se produce en el **buffer virtual**.

La situación es más compleja si consideramos los componentes de formulario que pueden obtener el foco y alterar el modo de funcionamiento. La navegación de siguiente elemento se produce por distintas acciones en cada modo de operación (en modo foco con tecla <*tab*>, en modo revisión con tecla <*e*> y tecla <x> para campos de casilla de verificación).

Con el objetivo de reconocer la intención real en la acción realizada, resultó necesario diseñar e implementar una solución que detecte las operaciones primarias, reales y directas producidas en el modo virtual, para capturar las interacciones particulares de cada usuario.

6.11 Complemento NVDA Accessibility

Ante la imposibilidad de acceder a las acciones específicas a través del componente Javascript, se recurrió al sistema de extensiones por complementos de aplicación del Screen Reader **NVDA.** Esta alternativa resultó viable para detectar la ocurrencia de eventos en el **buffer virtual,** dando lugar al desarrollo de un módulo de aplicación **NVDA** para Mozilla Firefox llamado **NVDA Accessibility.**

El complemento, al igual que **NVDA**, está desarrollado íntegramente en el lenguaje de programación Phyton, versión 2.7.14, 32-bit para Windows y se distribuye con extensión .nvda-addon para su posterior instalación en la aplicación.

La extensión captura los gestos de entrada en el modo revisión y al igual que el componente Javascript, las procesa buscando eventos de accesibilidad que reportar al servidor Kobold.

Eventos Detectables y Modelo de NVDA-Accessibility

La extensión intercepta todos los eventos de navegación asociados al modo revisión, mencionados en el punto 4.1.3, y los resguarda en eventos de interacción del modelo como herencia de la Clase NavigationbyKey.

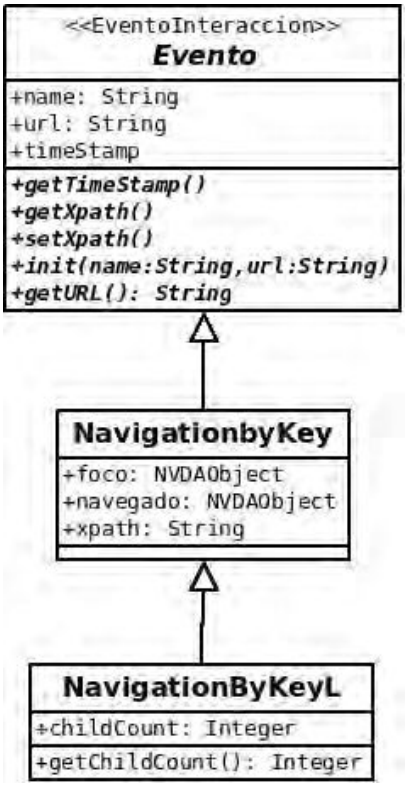


Figura 39: Vista parcial del Modelo de Eventos de Interacción del Complemento NVDA Accessibility

Los buscadores procesan la colección de eventos de interacción detectando eventos de accesibilidad como los descriptos en el capítulo 4 Eventos en Modo Revisión de NVDA.

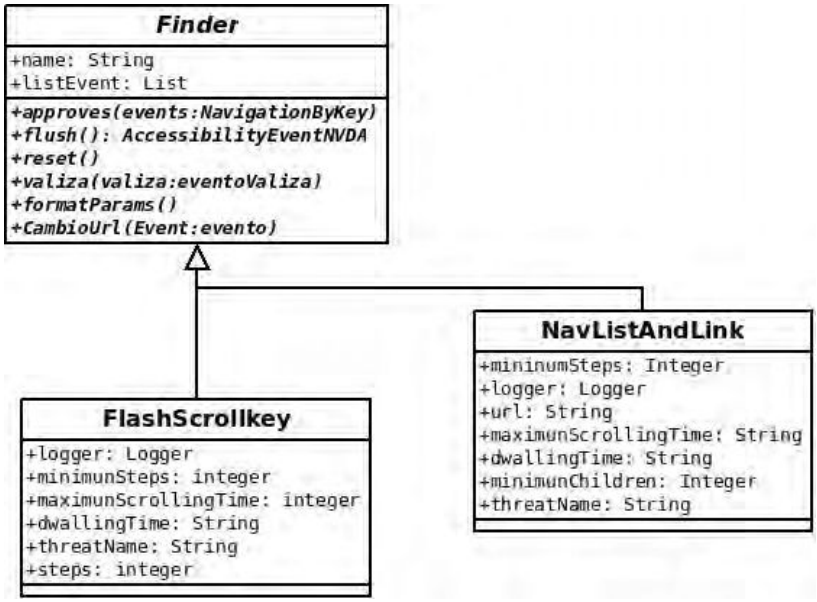


*Figura 40: Buscadores de **Accessibility Event** del Complemento **NVDA Accessibility***

Al detectar la presencia de un evento de accesibilidad el complemento lo reporta, para la URL de la página representada en el **buffer virtual**, al servicio API REST del componente servidor en una solicitud HTTP.

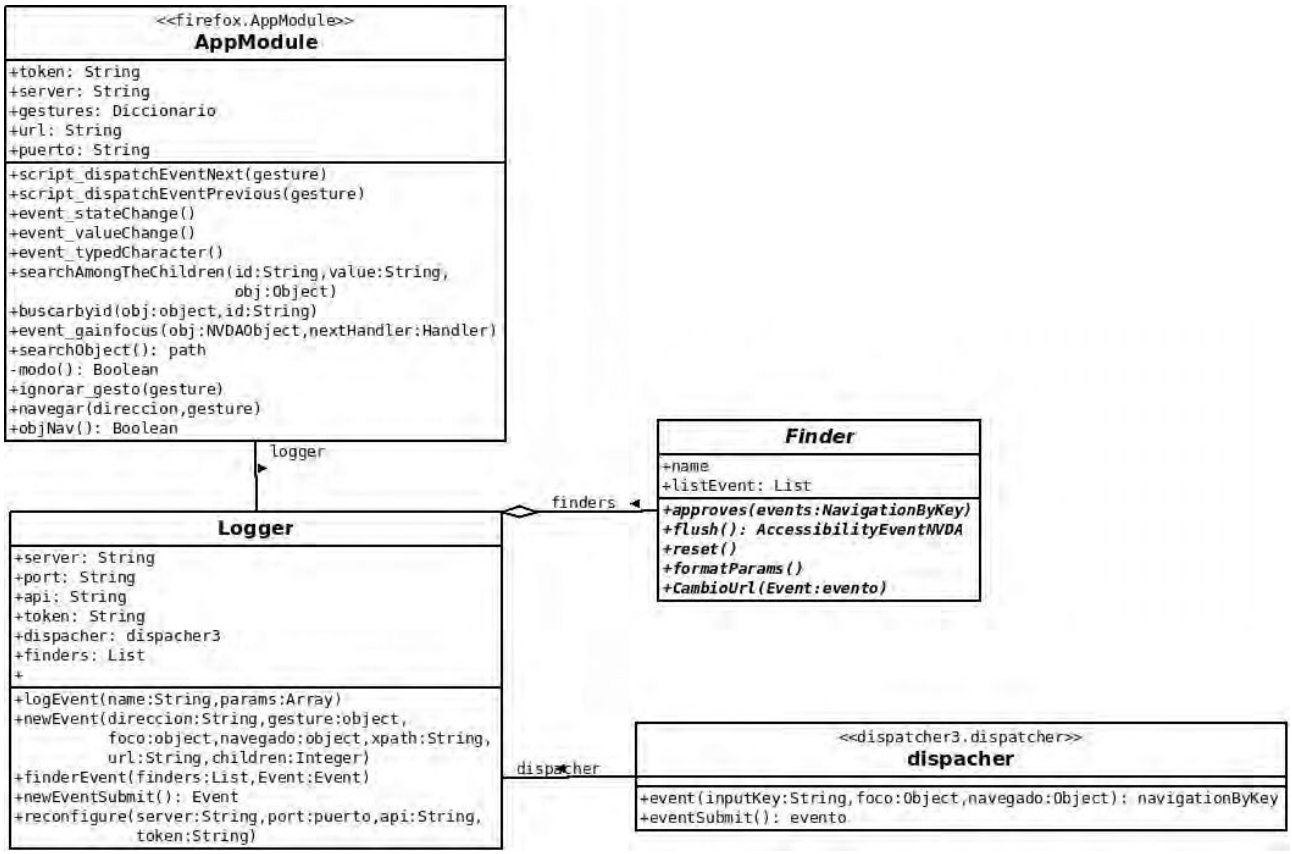


*Figura 40: Arquitectura del Complemento **NVDA Accessibility***

La arquitectura del complemento facilita extender la cobertura de **Accessibility Event** agregando nuevos eventos e incorporando gestos de entrada y buscadores. Estos se implementa como especializaciones de las clases **NavigationbyKey** y **Finfer** dentro de modelo.

Complemento NVDA y Snippet JavaScript

Para identificar cada elemento de la página web dentro del buffer virtual, el complemento se apoya en el snippet incrustado JavaScript. Este último incluye, en los campos identificadores “ID” vacíos de cada elemento navegable en modo revisión, el resultado de calcular el valor de su Xpath. Por ejemplo, un encabezado de nivel 2 que puede ser accedido en el cursor virtual se identifica dentro de la página con su campo ID, que el snippet completa con el valor <h2 id="*/html/body/h2*"> Egipto</h2>.

De esta manera, se puede reconocer unívocamente cada elemento utilizando la interfaz estándar del lector de pantallas y remitir la información al servidor sin requerir modificación del

núcleo de **NVDA**. Utilizando la misma estrategia, el snippet pone a disposición del complemento **NVDA**, información sobre la configuración como el endpoint del servicio de Kobold.

Capítulo VII

Bad Smells.

En este capítulo se catalogan las dificultades de accesibilidad junto a casos reales y se describe la detección de **Accessibility Smells** a partir de su relación con los eventos de accesibilidad.

7.1 Definición de Accessibility Smells de interacción de usuario.

Los usuarios con dificultades visuales que utilizan el lector de pantallas **NVDA** se enfrentan a potenciales problemas de accesibilidad. De igual manera que un **Usability Smell** señala deficiencias de diseño que impide o incomoda al usuario (Grigera, 2017), un **Accessibility Smell** lo hace para problemas de accesibilidad que pueden ser detectados y corregidos en la interfaz.

En tanto sea posible vincular una dificultad de accesibilidad con una solución en términos de refactoring, también es posible automatizar progresivamente el proceso y mejorarlo de forma incremental, reduciendo costos y plazos de desarrollo de aplicaciones, liberando a los desarrolladores de la necesidad de contar con experiencia en accesibilidad.

El catálogo de **Accessiblity Smells** contiene problemas de accesibilidad que se pueden detectar analizando el comportamiento de usuarios como Eventos de Accesibilidad catalogados en el Capítulo 4**.**

7. 2 Consideraciones para la Detección.

Los eventos de accesibilidad capturados en las interacciones son procesados por buscadores que implementan variadas heurísticas de detección de **Accessibility Smells**. Forman parte de una jerarquía de herencia de *Usability Smell Finder,* que implementa Kobold, e incluye el comportamiento básico para todos los *Finders*. Cada Finder específicos reutilizan las definiciones e incluyen su propio comportamiento en la detección de los **Bad Smell** de Accesibilidad.

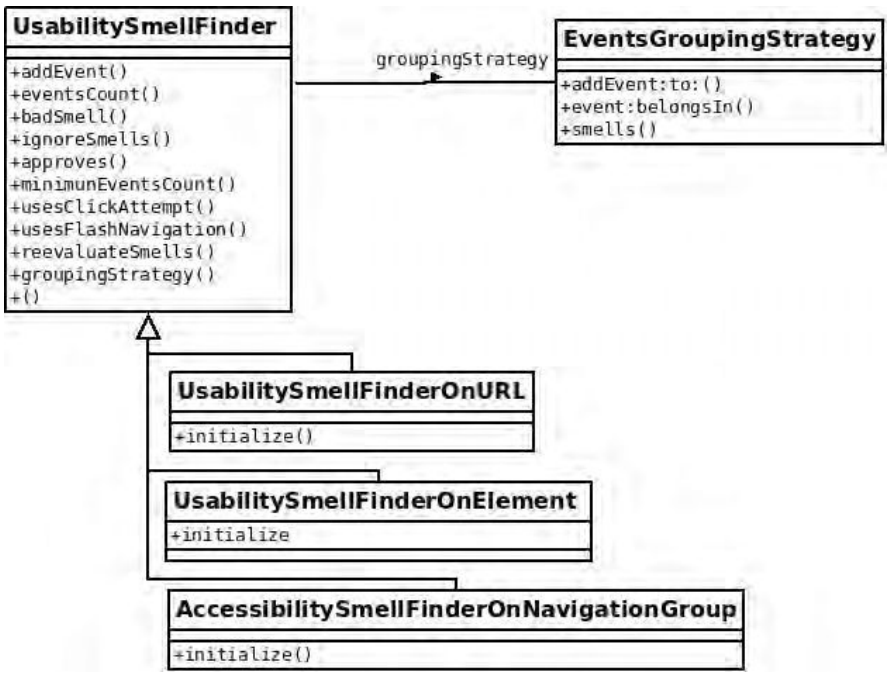


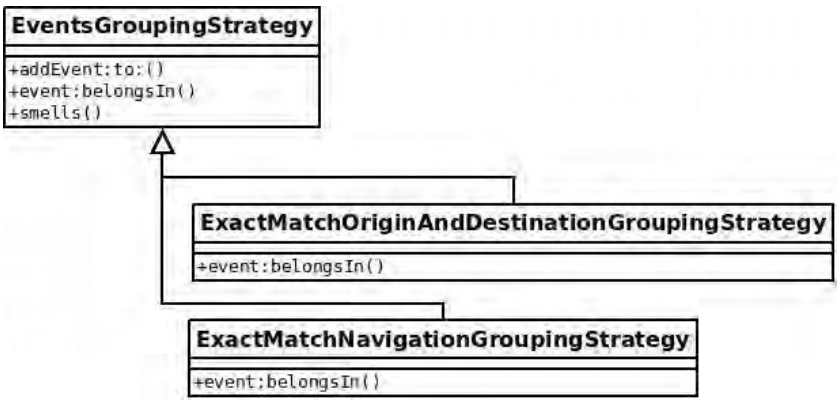
*Figura 41: Modelo de buscadores extendido para **Accessibility Smells**.*

La figura 41 muestra la herencia de la Clase Abstractas UsabiliySmellFinder que cuenta con estrategias de agrupamiento para los eventos y se representa en la clase Abstracta EventsGroupingStrategy. Esta arquitectura refleja la búsqueda que afecta a elementos de la interfaz o URL mediante las clases UsabilitySmellFinderOnElement y UsabilitySmellFinderOnURL respectivamente. Junto con las instancias de la clase AccessibilitySmellFinderOnNavigationGroup procesan eventos provenientes del Complemento **NVDA,** descripto en el capítulo 5, y conforman una jerarquía de la cual derivan todos los buscadores implementados en la versión de accesibilidad.

La mecánica básica y las consideraciones de optimización de performance continúan de la misma manera que en la versión original de Kobold. Al recibir los eventos, el servicio los deriva a los finders de su cuenta asociada. Existe una instancia de cada finder para cada tipo de smell a detectar que responde al mensaje #addEvent. A su vez, cada finder delega la acción a su estrategia de agrupamiento, que reúne los eventos según se afecte un elemento del DOM o una URL. Por último, cada estrategia de agrupamiento clasifica los eventos y los examina nuevamente para detectar la presencia de un **Accessibility Smell**.

Para mantener la velocidad del sistema se mantuvo el diseño ampliando las estrategias de agrupamiento originales mediante ExactMatchOriginAndDestinationGroupingStrategy y ExactMatchNavigationGroupingStrategy que son clases necesarias para eventos específicos provenientes del **complemento NVDA Accessibility**.

El modelo incluye las subclases de EventGroupingStrategy como se observa en la figura 42.



F*igura 42: Estrategias de Agrupamiento de Eventos para* ***Accessibility Smells*** *provenientes del* ***Complement NVDA.***

Ambas estrategias agrupan los eventos que reciben del complemento **NVDA Accessibility** conteniendo colecciones de acciones de desplazamiento en el buffer virtual como se describe a continuación:

- Exact Match Navigation Grouping Strategy *Estrategia de navegación de coincidencia exacta del grupo de interacciones.*

Agrupa eventos por secuencias de desplazamiento en el **buffer virtual** de **NVDA**, verificando que los eventos de interacción incluidos y su orden de prelación sean idénticos. Permite reconocer secuencias de navegación en el **buffer virtual** que frecuentemente realizan los usuarios siguiendo un camino.

- Exact Match Origin And Destination Grouping Strategy *Estrategia de navegación de origen y destino exacto del grupo de interacciones.*

Agrupa eventos por secuencias de desplazamiento en el **buffer virtual** de **NVDA**, verificando que los elementos de origen y de destino sea idénticos independientemente del camino recorrido.

7.3 Catálogo de Bad Smells de Accesibilidad.

Los **Bad Smells** de Accesibilidad se identifican por un código y un nombre; y se clasifican en tres grupos. El primer grupo lo conforman los *Bad Smell de Accesibilidad Descubiertos* en las interacciones y que no dependen de la tecnología de asistencia utilizada, particularmente de lectores de pantalla. En segundo lugar están los *Bad Smells Asociados a los Accesos Directos y Tareas comunes en Mozilla Firefox*, que surgieron con el uso de las funciones de asistencias a la accesibilidad que ofrece el navegador. Por último, el grupo de los *Bad Smell Asociados al Modo*

Revisión del NVDA que se manifiestan al utilizar las opciones que el screen reader ofrece para facilitar la navegación directa entre los elementos de una página web.

A continuación se describe cada **Bad Smells** junto a los casos reales tratados en el capítulo 4 y un análisis de la dificultad de Accesibilidad que presentan.

7.3.1 Bad Smell de Accesibilidad Descubiertos.

Estas dificultades fueron descubiertas empíricamente durante la navegación en la nube buscando **Bad Smells** de Accesibilidad con el screen reader **NVDA** y el navegador Web Mozilla Firefox.

D01- **Validation test without electronic text** *Prueba de validación sin texto electrónico.*

Las notificaciones visuales no acompañadas de texto electrónico son difíciles de percibir por los usuarios con dificultades visuales que utilizan **NVDA**. Ante una *Prueba de validación sin texto electrónico* no superada, los lectores de pantalla no pueden sintetizar ningún reporte de error y el usuario desconoce que su entrada de datos fue considerada incorrecta en la página.

Retomando el Caso Real 1 referido al Destino de Vuelos en la Página de Aerolíneas Argentinas.

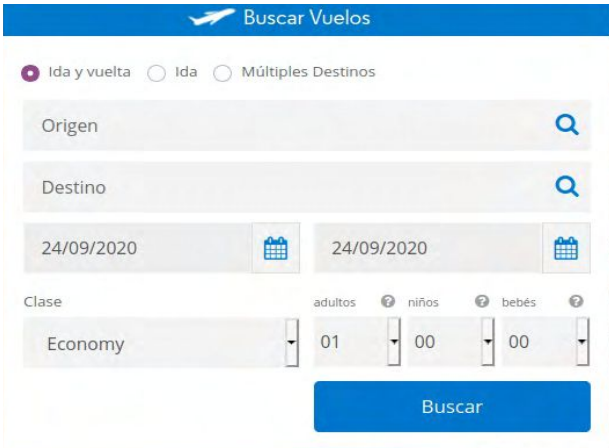


Figura 43: Página de reserva de vuelos de Aerolíneas Argentinas .

El usuario trata de reservar un ticket de vuelo, pero la página web eliminó automáticamente el valor de origen sin reportarle ningún mensaje. Solo cuando el usuario peciona click sobre buscar se visualiza “Debe ingresar un Destino”.

En este caso estamos ante una *Prueba de validación sin texto electrónico* en la cual los lectores de pantalla no pueden sintetizar ningún reporte de error. Un mensaje visual que no se acompaña de texto electrónico puede no ser percibido por los usuarios con dificultades visuales dificultando o impidiendo la interacción con la página web.

D02 - **Input field without label that describes it** *Campo de entrada sin etiqueta que lo describa.*

Ante un C*ampo de entrada sin etiqueta que lo describa* un usuario con dificultades visuales, podría no saber como continuar utilizando la página web. Los valores de estas etiquetas ayudan al usuario a continuar navegando informando que tipo de ingreso se espera en un campo de entrada. Los usuarios con dificultades visuales requieren que estos campos dispongan de un contenido textual asociado que resulte detectable por las herramientas de asistencia. En algunos casos, los programadores utilizan la etiqueta “span” para este propósito, pero no resultan útiles porque ante un campo de entrada de texto los lectores de pantalla utilizan el contenido de sus etiquetas “label” como texto electrónico. Los estándares promueven el uso de <label> que fue diseñada con este propósito y deben ser vincularlas con los cuadros de entrada de texto mediante un atributo For.

Retomando el Caso Real 2 referido a las Reservas de un vuelo en la página de Aerolíneas Argentinas, donde los campos de entrada de origen y destino carecen de etiquetas <label> asociadas.

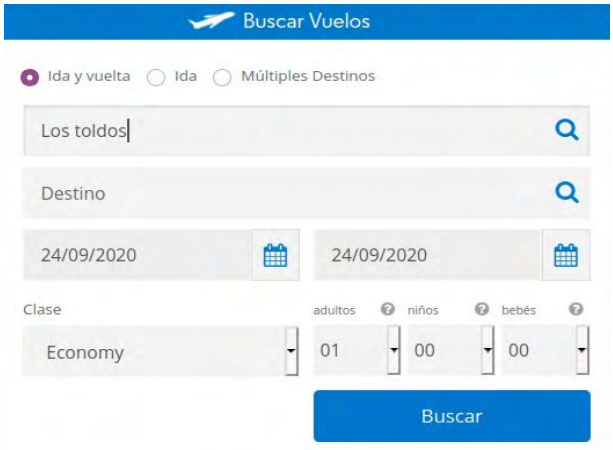


Figura 44: Página de reserva de vuelos de Aerolíneas Argentinas con los datos ingresados por el usuario.

Dado que los lectores de pantalla utilizan como texto electrónico a sintetizar, en primera instancia el texto de las etiquetas HTML <label> asociada, al carecer de ellas no es posible ofrecer al usuario información sobre el tipo de entrada, lo que puede dificultar o impedirle utilizar la aplicación.

D03 - **Absence of descriptive text for text entry fields** *Ausencia de texto descriptivo para los campos de entradas.*

Al posicionar el cursor sobre un campo de entrada de datos, el lector de pantalla **NVDA** reproduce textos electrónicos asociados al elemento. Estos textos deben ser claros indicadores del dato esperado. La ausencia de texto descriptivo, o un contenido textual ambiguo, constituye una barrera de accesibilidad **Absence of descriptive text for text entry fields** para el usuario

con dificultades visuales, debido a que **NVDA** reporta que el foco se encuentra en un campo de edición, pero no dispone de información que le indique al usuario que entrada se espera que ingrese, situación que le impide continuar con la interacción.

Retomando el Caso Real 3 referido al Sistema de consultas del registro patrimonial de la Universidad Nacional de Salta.

Al posicionar el foco sobre los elementos de entrada de datos no se sintetiza ningún texto.

Figura 45: Sistema de Patrimonio SABUM de la Universidad Nacional de Salta.

Esta *Ausencia de texto descriptivo para los campos de entradas* se debe a que el elemento de la interfaz no dispone de atributo *aria-label*, ni de texto en una etiqueta <label> vinculada, ni de valor en el atributo *placeholder*. Al igual que en el caso anterior, no es posible ofrecer información sobre como continuar utilizando la aplicación con idénticas consecuencias para los usuarios.

D4 - **Distant content for keyboard use on a page** *Contenido distante para uso del teclado en una página.*

Este Bad Smell es una versión adaptada de Distant Content (Grigera 2017), para elementos que se distribuyen en una página web, e indica que el camino secuencialmente seguido con tecla <Tab> o <Shift> + <Tab>, desde el nodo inicial hasta el final puede ser demasiado largo. Al utilizar el teclado los usuarios pueden recorrer una gran cantidad de elementos para acceder a la función requerida. Esta situación reporta que un determinado elemento debería estar más accesible, más arriba en la secuencia de navegación, dado que la disposición actual es **Distant content for keyboard use on a page** y requiere un esfuerzo adicional.

Retomando el Caso Real 4 referido a la Solicitud de becas de estudio en la Universidad Nacional de Salta en el sistema Siu Tehuelche.

Figura 46: Página principal de la Universidad Nacional de Salta.

El interés del usuario se centra en un *Contenido distante para uso del teclado en una página* por lo que debe recorrer secuencialmente con tecla <Tab> un número elevado de elemento de la interfaz navegando a través de un camino extremadamente largo que le requiere realizar un mayor esfuerzo.

D06 - **Distant Content in Accessibility** *Contenido distante en accesibilidad.*

La dificultad **Distant Content in Accessibility**, se presenta como una barrera asociada a un camino de navegación entre páginas que los usuarios deben recorrer, permaneciendo intervalos breves de tiempo en los nodos intermedios, para acceder al nodo final de información. Esta dificultad es la versión de Accesibilidad del Smell de usabilidad Distant Content (Grigera, 2017).

Retomando el Caso Real 1.4 referido a la Creación de Cuentas de usuario en la Página de CONEAU Global.

Figura 47: Creación de Cuentas de Usuario de CONEAU Global.

La creación de cuentas presenta la dificultad *Contenido distante en accesibilidad* en la cual es necesario recorrer un camino de navegación entre diferentes páginas permaneciendo innecesariamente en nodos intermedios requiriendo mayor esfuerzo que si estuviera accesible directamente.

D07 - **Captcha in the form** *Captcha en el formulario.*

La presencia de un **Captcha in the form** ocurre cuando, para verificar que la interacción es realizada por un usuario humano y no por software automático malicioso, algunos sitios solicitan que se añada un código de seguridad antes de enviar un formulario Web. Estos códigos son exhibidos visualmente al usuario en el mismo formulario, en un formato que dificulta su descubrimiento de manera automática.

Se espera que, en el campo de entrada del código de seguridad, los usuarios logren ingresar el dato remitido previamente al procesar la solicitud del formulario.

Sin embargo, los lectores de pantalla no pueden acceder al texto de estos códigos para sintetizarlos. Esto códigos se presentan en formatos conocido como captcha y el usuario nunca puede conocer que información se le solicita impidiéndole continuar. Aunque algunos sitios resuelven esta dificultad facilitando versiones de audio de los códigos, no están siempre presentes o incluso pueden no ser accesibles.

Retomando el Caso Real 1.5 referido a la Solicitud de Clave Única de Identificación Laboral en la Página del ANSES.

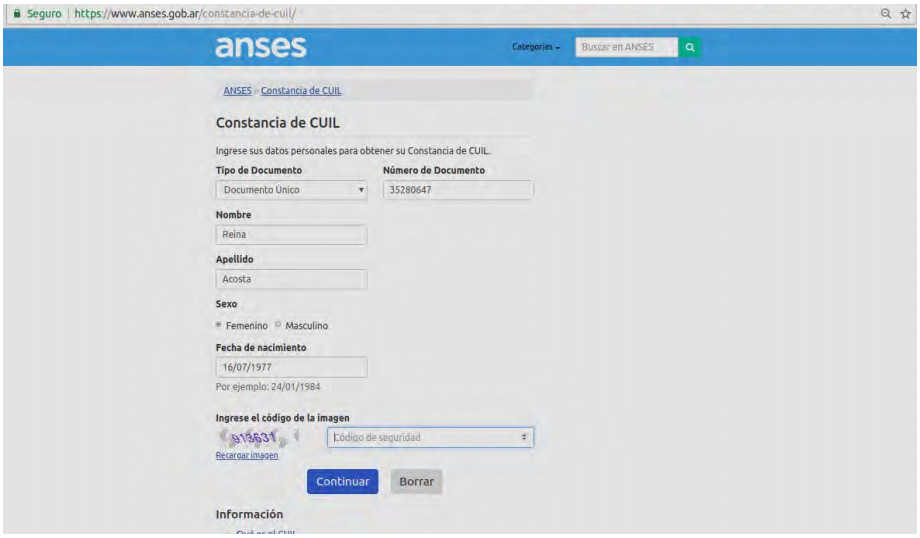


Figura 48: Solicitud de Constancia de CUIT en la Página web de Anses.

El Captcha en el formulario impide al usuario continuar utilizando la aplicación debido a que no existe posibilidad de acceder a su código, ni se dispone de texto electrónico que se pueda sintetizase para indicarle su contenido textual.

D08 - **Forms without send button** *Formularios sin botón enviar.*

Es frecuente encontrar sitios con alternativas visualmente elaboradas y distintas a las presentaciones estándar de un botón de envío asociado a un formulario. La función puede estar presente dentro de un elemento diferente, por ejemplo un que se visualiza como botón y responder al evento click. Estas propuestas, orientadas a lo visual, no siempre son accesibles mediante el uso del teclado en el evento keypress, situación que impide a los usuarios con dificultades visuales enviar el formulario.

En este caso se analiza la Búsqueda de Vuelos en la Aerolínea Boliviana de Aviación donde el usuario NVDA desea conocer los vuelos disponibles entre las ciudades de Barcelona y Buenos Aires para las fechas indicadas en pantalla.

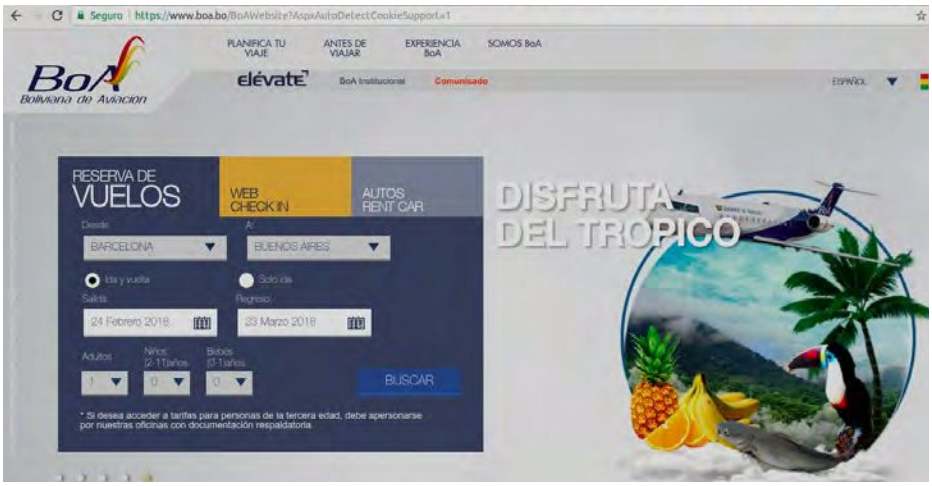


Figura 49: Página de Reserva de Vuelos de la Aerolínea Boliviana de Aviación.

Sin embargo no es posible confirmar la búsqueda desde el teclado, debido a que se encuentra ante un *Formulario sin botón enviar*.

Los desarrollos donde los envío de formularios Web se construyen con estructuras diferentes a botones de envío que puedan activarse mediante el teclado, impiden enviar el formulario a los usuarios que utilizan un screen reader.

D14 - **Datepicker inaccessible using the keyboard** *Datepicker inaccesible usando el teclado.*

El componente Datepicker es un pop-up que ofrece una interfaz gráfica de usuario para seleccionar fechas en un calendario. Pero la mayoría de estos widgets responden a eventos asociados al ratón, motivo por el cual los usuarios que utilizan el teclado no puede desplazarse entre las fechas ni seleccionarlas correctamente. Incluso existen casos en que los lectores de pantalla no pueden sintetizar su contenido.

Analizando el caso de un usuario que intenta crear una cuenta en el sistema Siu Tehuelche, se observa la dificultad de acceso al Widget de selección de fecha de nacimiento.

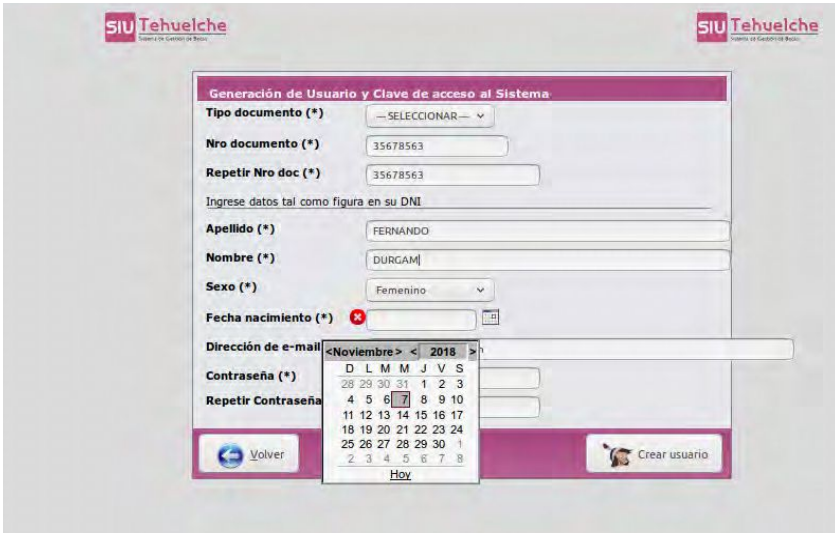


Figura 50: Página del Sistema de Becas Siu Tehuelche.

La presencia de un *Datepicker inaccesible* que no responde a entradas de teclado, sino a eventos de mouse, impende ingresar el dato requerido.

D09- **Search without feedback** Búsqueda sin retroalimentación.

Cuando las páginas que incluyen cajas de búsqueda no suministran textos electrónicos de los resultados, los screen reader no sintetizan descripciones a los usuarios con dificultades visuales, que pueden quedar desconcertados ante la falta de retroalimentación.

Retomando el Caso Real 1.6 donde El usuario NVDA busca el texto ”despacito” en la página de youtube.

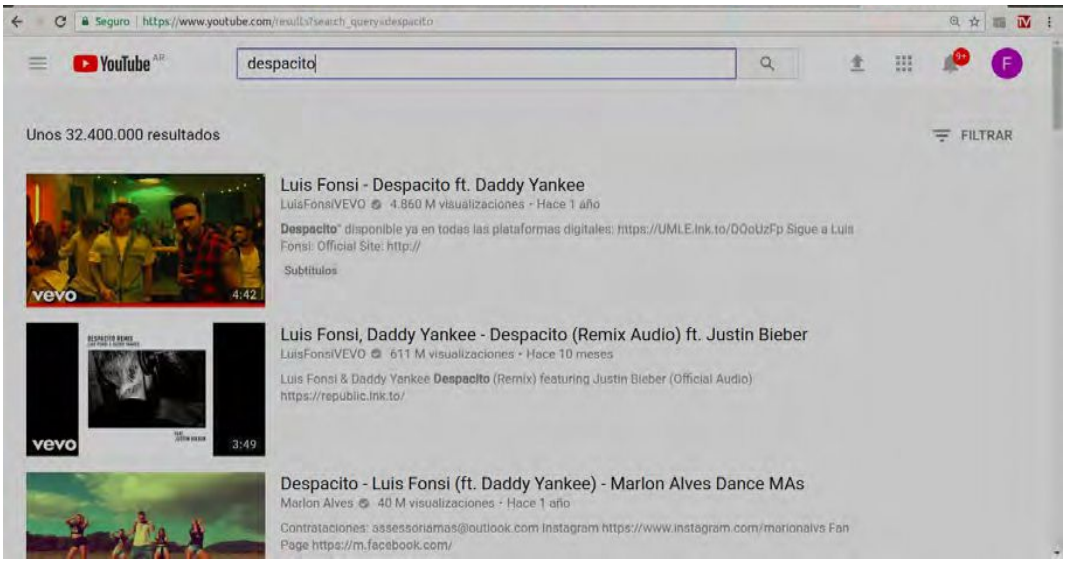


Figura 51: Búsqueda del Texto “Despacito” en la página de Youtube.

Al no ser informados sobre lo ocurrido con la acción requerida, los navegantes con dificultades visuales pueden interrumpir la interacción en la web. Estas *Búsquedas sin retroalimentación* puede desconcertar al usuario e imposibilitarle comprender lo que ocurre sobre la interfaz.

D10- **Unexpected language change** C*ambio de idioma inesperado*

La dificultad puede presentarse al sintetizar literalmente texto en un idioma diferente al esperado por el usuario, como consecuencia de no declararlos adecuadamente en la página. Cada elemento puede contener texto en un idioma determinado y debe ser indicado para una correcta síntesis de voz. No se trata de traducir el texto mediante servicios de **NVDA**, sino de sintetizarlo en el idioma correcto**.**

Retomando el Caso Real 1.7 referido al Catálogo de la biblioteca de la Universidad Nacional de Salta.

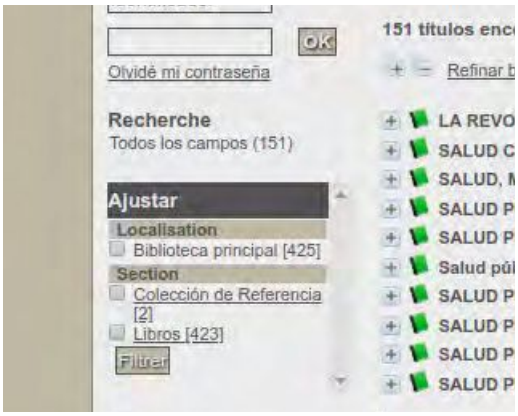


Figura 52: Filtros del Catálogo de la biblioteca PMB.

Los filtros que se puede aplicar a las búsquedas bibliográficas están en un idioma diferente al resto de los elementos de la interfaz. Debido al C*ambio de idioma inesperado* el usuario puede no comprender la función ofrecida en la página.

D11 - **Disposition of confusing elements** *Disposición de elementos confusa.*

Al desplegar elementos sobre una presentación web, debe cuidarse que cada uno transmita su mensaje teniendo en cuenta tanto a los requerimientos funcionales y las características del usuario, evitando cualquier ***Disposition of confusing elements***. Una navegación puede fracasar si se presenta una estructura caótica y los usuarios no pueden continuar utilizando el sitio. Una estructura jerárquica y tabular puede resultar útil en términos operativos, para los usuarios con dificultades visuales que utilizan el teclado, pero además debe reflejar la sucesión de pasos y la perspectiva del usuario con respecto al sitio, su información y sus servicios.

Retomando el Caso Real 1.8 referido a la Solicitud de becas del sistema Siu Tehuelche.

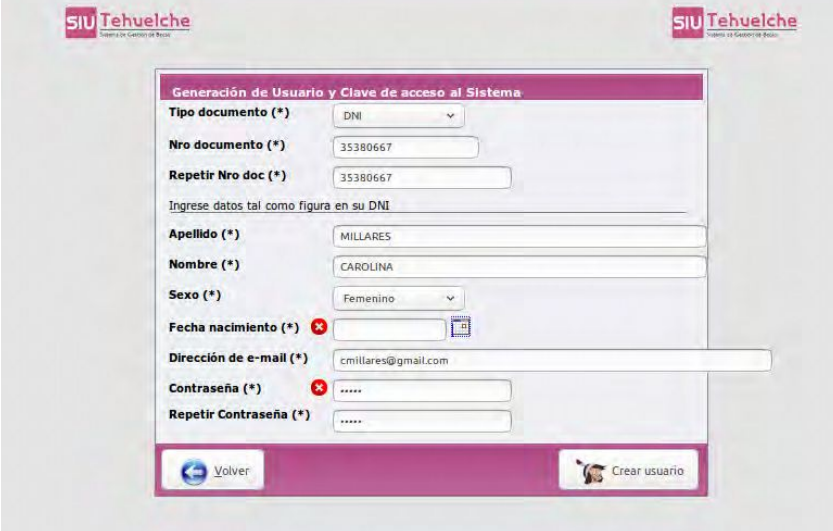


Figura 53: Solicitud de becas del sistema Siu Tehuelche.

Al intentar crear una cuenta en el sistema Siu Tehuelche, el usuario accede secuencialmente a los elementos de la página, pero una *Disposición de elementos confusa* puede no permitirle enviar el formulario. En la figura 53, incluye un Widget para fechas, con una secuencia cuyo orden de tabulación es posterior al botón de submit. Por este motivo, al intentar enviar el formulario y ante la carencia de un valor requerido, pueden no ser superadas las pruebas de validación.

7.3.2 Bad Smell de Accesibilidad Asociados a los Accesos Directos y Tareas comunes en Firefox.

Las facilidades de interacción para la accesibilidad que ofrece Mozilla Firefox agregan nuevas posibilidades a detectar en el comportamiento a los usuarios con dificultades visuales. Estas interacciones pueden ser fuente de nuevas barrera de accesibilidad.

D12 – **Overlooked Content** *Contenido pasado por alto.*

El bad Smell ***Overlooked Content*** definido en la versión de Kobold para usabilidad, también se presenta en la versión de accesibilidad. Esta dificultad aparece en contenidos que suelen ser rápidamente salteados con acciones rápidas de scroll. Se redefine el bad smell para los usuarios con dificultades visuales que utilizan el teclado, dato que sus motivos para saltarse el contenido y las soluciones pueden ser diferentes.

Retomando el Caso Real 2.1 referido a la Búsqueda de alojamiento en la Página de Booking.

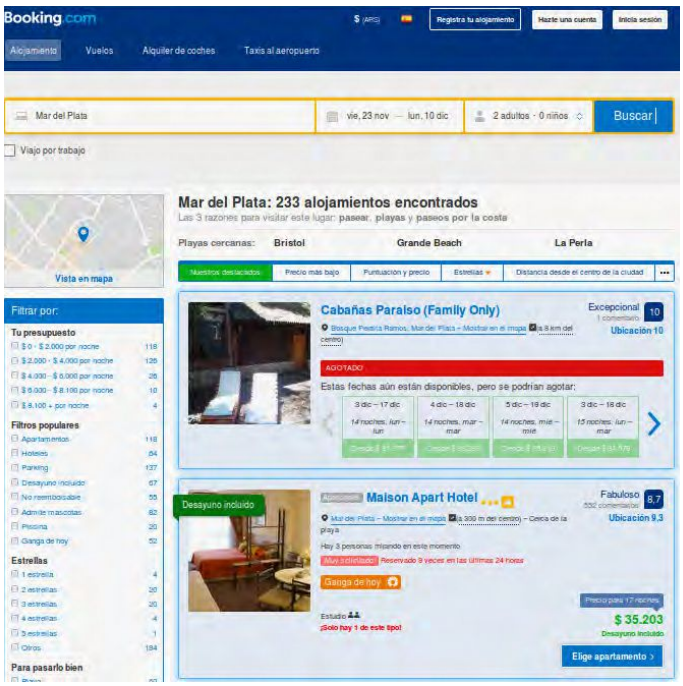


Figura 54: Página de Alojamiento de Booking.

Realizada la búsqueda en el sitio de alojamientos Booking el usuario se desplaza entre los resultados presionando la tecla <space> y <sihft> + <space> pasado por alto contenido no accedido.

7.3.3. Bad Smell Asociados al Modo Revisión del NVDA.

El modo revisión de **NVDA** con sus funciones de acceso rápido, aporta nuevas formas de comportamiento que pueden ser utilizadas para descubrir barreras de accesibilidad como ocurren en los siguientes casos.

D05 - **Overlooked Content For NVDA** *Contenido pasado por alto para* ***NVDA.***

No todos los contenidos presentados en el **buffer virtual** son accedidos por los usuarios. Algunos no son tenidos en cuenta y los usuarios acceden directamente a elementos que se encuentran más adelante en la estructura de página web. La razón para ignorar frecuentemente contendidos, puede tener origen en la presencia de una barrera de accesibilidad ***Overlooked Content For NVDA*** que agrega acciones de interacción innecesarias a los usuarios. Esta dificultad es una versión modificada de *Overlooked Content* (Grigera,2017) adaptado para el uso en el **buffer virtua**l con **NVDA**.

Analizando el Caso Real 3.1 referido a la Búsqueda en la página de Wikipedia sobre la historia de Egipto, es posible acceder mediante el **buffer virtual** a la estructura del documento y realizar una navegación donde el *Contenido pasado por alto para NVDA* es ignorado.

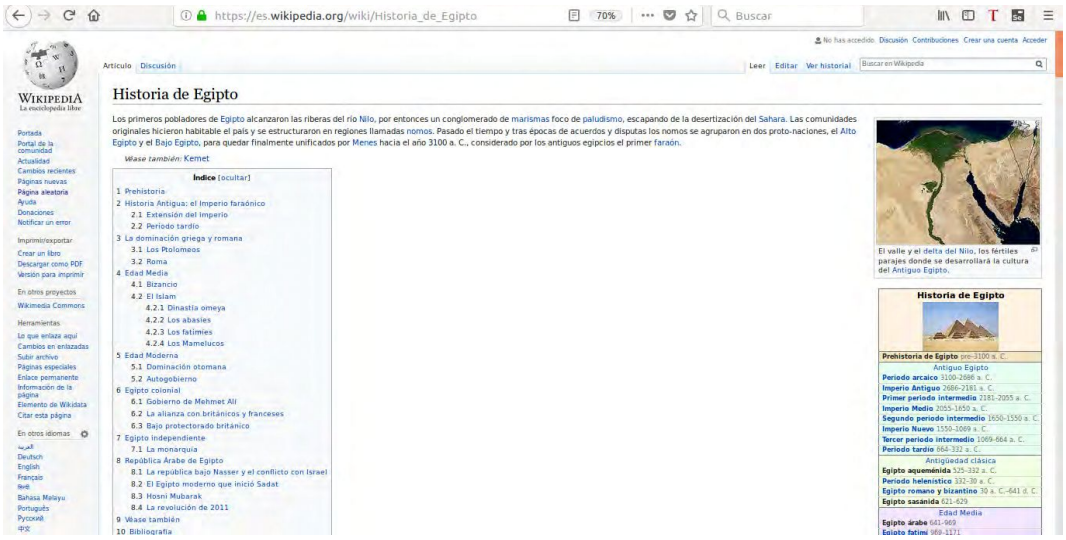


Figura 55: Página de Wikipedia sobre la Historia de Egipto.

D13 - **Distant Content For List Menu for Complement NVDA** *Contenido distante para el menú de lista del complemento* ***NVDA.***

La representación jerárquica de elementos accesibles directamente por teclado con **NVDA**, **buffer virtual,** resulta conveniente para los usuarios con dificultades visuales que acceden a una ubicación especifica en la página. El acceso directo frecuente a un determinado contenido, ignorando los elementos previos en la secuencia, puede estar reportando que el elemento de interés se encuentra distante. En este sentido***, Distant Content For List Menu for Complement NVDA*** es una versión modificada del Usability Smell Distant Content (Grigera,2017) para elementos de listas de enlaces navegadas en una página, en la cual los usuarios recorren un largo camino de nodos intermedios por un breve instante hasta llegar al nodo final.

Retomando el Caso Real 3.2 referido al Acceso a la página de la Universidad Nacional de Salta para conocer el menú del comedor.

Figura 56: Enlaces de Comedor Estudiantil.

Al encontrarse con *Contenido distante para el menú de lista del complemento NVDA* los usuarios acceden a los elementos web mediante operaciones de navegación rápida del modo revisión de **NVDA**, cuando deben navegar a lo largo de un conjunto de listas de enlaces que representan menús de acceso al contenido de la página.

7.4 Detección de los Bad Smell

Los comportamientos de los usuarios en la interfaz web, modelados como eventos de accesibilidad, se utilizan para detectar las barreras de accesibilidad presentes en los sitios web. A continuación se describe la relación entre los eventos de accesibilidad descriptos en el capítulo 4 y los bad Smell de Accesibilidad catalogados.

E1 - **Auto Delete Entry Field** *Auto borrado de campo de entrada* para detectar D01- **Validation test without electronic text** *Prueba de validación sin texto electrónico.*

Un evento de accesibilidad **Auto Delete Entry Field,** se produce cuando sobre un campo de entrada se modifica automáticamente el valor ingresado al desplazar el foco, probablemente por un error en el ingreso de datos. Una motivo para esto puede ser la presencia de un Smell de Accesibilidad **Validation test without electronic text.** Cuando las páginas web reportan el problema mediante mensajes visuales, generalmente texto desplegable o remarcando los bordes, los screen reader no pueden interpretarlos y los usuarios con problemas visuales no pueden detectar la situación.

E02 - **Electronic text for non-significant speech synthesis** *Texto electrónico para síntesis de voz no significativa* para detectar D02 -**Input field without label that describes it** *Campo de entrada sin etiqueta que lo describa.*

La ocurrencia de **Electronic text for non-significant speech synthesis,** puede condicionar los comportamientos de los usuarios al desorientarlos sobre que dato se espera que ingresen, o el formato que deben utilizar al ingresar información. La inexistencia de una etiqueta Label que describa el campo de entrada, por ejemplo cuando se solicita una contraseña mediante una imagen, o la etiqueta no está enlazada al elemento, puede constituir una barrera **Input field without label that describes it** que dificulte o impide continuar utilizando la página web.

E3 - **Electronic text for non-existent speech synthesis** *Texto electrónico para síntesis de voz inexistente* para detectar D03 **- Absence of descriptive text for text entry fields** *Ausencia de texto descriptivo para los campos de entrada.*

Un evento **Electronic text for non-existent speech synthesis** se presenta cuando el usuario intentar ingresar un dato en la página sin contar con un feedback adecuado. La presencia de un **Bad Smell Absence of descriptive text for text entry fields** es consecuencia de no contar ninguna de las alternativas de texto arialabel, Placeholder o Label descriptiva para el elemento de entrada.

E4- **High Frequency Of Use Of The Tab Key** *Alta frecuencia de uso de la tecla de tabulación para detectar* D4 - **Distant content for keyboard use on a page** *Contenido distante para uso del teclado en una página.*

En el navegador web se evalúan las acciones al producirse secuencia de entradas en ráfaga correspondiente a la tecla <Tab> como un evento **High Frequency Of Use Of The Tab Key.** Este evento se remite al servidor, donde se actualizan y evalúan las cantidades reportadas para determinar la presencia en la página un bad Smell **Distant content for keyboard use on a page**.

E6 - **Navigation Path Version Accessibility** *Ruta de navegación para la Versión de Accesibilidad* para detectar D06 - **Distant Content in Accessibility** *Contenido distante en accesibilidad.*

El evento de accesibilidad **Navigation Path Version Accessibility,** es una adaptación del evento de usabilidad Navigation Path, original de Kobold, que en el escenario del usuario con dificultades visuales puede requerir consideraciones diferentes de detección y corrección para el bad Smell **Distant Content** detectable en Kobold. Con el objeto de mantener la arquitectura extensible y permitir que la versión de accesibilidad contenga la detección de esta dificultad, se desarrolló una instancia del proceso para los usuarios con dificultades visuales. De esta manera y gracias al diseño arquitectónico original, tanto la detección de las dificultades como su posible soluciones quedan abiertas a nuevas especializaciones y estrategias.

E7 - **Unfilled Form** *Formulario sin completar* para detectar:

D07 - Captcha in the from *Captcha en el formulario.*

D08 - Forms without send button *Formularios sin botón de envío.*

D14 - Datepicker inaccessible using the keyboard *Datepicker inaccesible usando el teclado.*

Cuanto se presenta un **Unfilled Form** en accesibilidad se utilizan eucarísticas para examinar el elemento afectado por el evento, un formulario HTML, en búsqueda de descripciones útiles a fin de inferir la presencia algun **bad Smell**.

Para detectar un **Captcha in the from** se examina la presencia de algún elemento Captcha en el formulario. Para el caso de **Forms without send button** se verifica la ausencia de un botón de submit y para detectar **Datepicker inaccessible using the keyboard** se examina la existencia de un calendario dentro del formulario.

E8 - **Search result without electronic text** *Resultado de búsqueda sin texto electrónico* para detectar D09- **Search without feedback** *Búsqueda sin retroalimentación.*

Ante la presencia de un evento de accesibilidad, **Search result without electronic text,** los buscadores verifican que la cantidad de ocurrencias del evento para el elemento afectado supere la cantidad mínima requerida para reportar el **bad Smell Search without feedback**.

E9 -**Confused speech synthesis** *Síntesis de voz confusa* para detectar D10 - **Unexpected language change** *Cambio de idioma inesperado.*

El evento ***Confused speech synthesis*** remitido al servidor entrega al buscador información asociada a los posibles idiomas del elemento afectado. El buscador debe ser capaz de determinar si la confusión se debe a la incorrecta definición del idioma para el elemento afectado reportando un **bad smell *Unexpected language change***.

E10 -**Inappropriate Tab Sequence** *Secuencia de tabulación inapropiada* para detectar D11 -**Disposition of confusing elements** *Disposición de elementos confusa.*

En la presentación de una página web, donde las interacciones están guiadas por el orden de tabulación de los elementos, una **Inappropriate Tab Sequence** puede requerir un esfuerzo adicional de los usuarios con dificultades visuales que dependen del teclado e impedirles comprender la funcionalidad. El motivo puede deberse a una retroalimentación confusa como consecuencia de un Bad Smell **Disposition of confusing elements.**

E11 – **FastScrollingWithKeyboard** *desplazamiento rápido con teclado* para detectar D12–**Overlooked Content** *Contenido pasado por alta.*

Similar al caso de usabilidad, en el cual se define un buscador que detecta ***Overlooked Content*** pero utilizando como recursos instancias de evento de accesibilidad ***FastScrollingWithKeyboard*** sobre una URL. Es importante en esta versión definir todas las clases que intervienen en la traza del proceso de detección con el fin de mantener extensible la aplicación.

E5 - **Flash Scrolling in Accesibilidad with NVDA add-on** *Ráfaga de desplazamiento en el complemento **NVDA*** para detectar D12 – ***Overlooked Content*** *Contenido pasado por alto.*

Las acciones de interacción que caracterizan al evento **Flash Scrolling in Accesibilidad with NVDA add-on** son una sucesión de solicitudes de navegación directa a elementos dentro del **buffer virtual** de **NVDA**. Entre cada par de interacciones de navegación sucesivas se accede a la posición de dos elementos del grupo definidos en la jerarquía del buffer. Entre ambas posiciones puede existir contenido que los usuarios han decidido ignorar deliberadamente. Desde el primer hasta el último desplazamientos los usuarios han obviado síntesis de textos electrónicos de los contenidos intermedios. Esta situación puede estar indicando que los contenidos no resultan significativos o que su ubicación actual no es conveniente. Con esa información se puede inferir la presencia de un **Bad Smell** de accesibilidad ***Overlooked Content For NVDA.***

E12 - **Navigation Between Lists of Links for the NVDA add-on** *Navegación entre listas de Enlaces en el complemento NVDA* para detectar *D13* -**Distant Content For List Menu for Complement NVDA** *Contenido distante para el menú de lista del complemento NVDA.*

La ocurrencia de **Navigation Between Lists of Links for the NVDA add-on** se produce cuando los usuarios, han accedido en el modo revisión a una lista de enlaces de algún conjunto de listas existentes en el **buffer virtual**. La frecuencia de esta acción puede indicar que la lista en la que se interesan debería encontrase más adelante en la presentación. La distancia puede ser

excesiva porque requiere recorrer las listas una a una hasta llegar a la de su interés. En estos casos podemos asumir que estamos ante un **Bad Smell** de accesibilidad **Distant Content For List Menu for Complement NVDA.**

7.5 Accessibility Events y los Accessibility Smells

En la Tabla 1 se incluye **Accessibility smells** junto a **Accessibility events** utilizados para su detección.

Accessibility events	Accessibility smells
E01- Auto Delete Entry Field	D01- Validation test without electronic text
E02 - Electronic text for non-significant speech synthesis	D02- Input field without label that describes it
E03- Electronic text for non-existent speech synthesis	D03- Absence of descriptive text for text entry fields
E04- High Frequency Of Use Of The Tab Key	D04- Distant content for keyboard use on a page
E05- Flash Scrolling in Accesibilidad with NVDA add-on	D05- Overlooked Content For NVDA
E06- Navigation Path Version Accessibility	D06- Distant Content in Accessibility
E07- Unfilled Form	D07- Captcha In The Form
	D08- Forms without send button
	D14- Datepicker inaccessible using the keyboard
E08- Search result without electronic text	D09- Search without feedback
E09- Confused speech synthesis	D10 - Unexpected language change
E10- Inappropriate Tab Sequence	D11- Disposition of confusing elements
E11- FastScrollingWithKeyboard	R12 Overlooked Content
E12- Navigation Between Lists of Links for the NVDA add-on	D13-Distant Content For List Menu for Complement NVDA

Tabla 21. Accessibility Smells y su relación con los Accessibility Events.

Los **bad Smell** de accesibilidad son señales de posibles dificultades, que al igual que los **Usability Smell**(Grigera, 2017), solo ofrecen pistas y dependiendo de factores como el contexto podrían no indicar un problema real.

Capítulo VIII

Refactoring

8.1 Definición de Accessibility Refactoring

Self Refactoring (Grigera, 2017) ofrece un catalogado de refactorings específicamente diseñados para mejorar la usabilidad de aplicaciones web, como AddTooltip que agrega texto significativo. Aplicar refactorings a problemas de accesibilidad no requiere tener acceso al servidor de la aplicación (Garrido, Firmenich, et al., 2013).

Un **Accessibility Refactoring** es una solución que se puede aplicar mediante modificaciones en el front end para resolver las dificultades de interacción presentes en una página web y que afectan la accesibilidad de los usuarios con dificultades visuales que utilizan el teclado en NVDA.

En este capítulo se describe la extensión de Kobold para ofrecer soluciones a **Accessibility Smells** en términos de **Accessibility Refactorings,** y como pueden aplicarse a casos reales en la Web.

8.2 Consideraciones para la aplicación Accessibility Refactorings en Kobold

Los **Accessibility Refactorings** se implementan como extensiones de clases existentes en Kobold, manteniendo JavaScript como lenguaje de programación del lado del cliente y delegando en cada **Accessibility Smell,** la responsabilidad de recolectar la información necesaria para generar el código.

De los 17 **Accessibility Refactoring** definidos en el catálogo**:**

4 Son automáticos y no requieren intervención del usuario.

11 Son semi automatizados y requieren algún grado menor de intervención.

3 Requieren aplicación manual por parte de desarrolladores expertos.

El mecanismo de generación de código no cambia en esta versión de Accesibilidad de Kobold. Los refactorings modifican la presentación web independientemente del agente de navegación y el lector de pantalla, manteniendo el modelo de clases base como se observa a continuación:

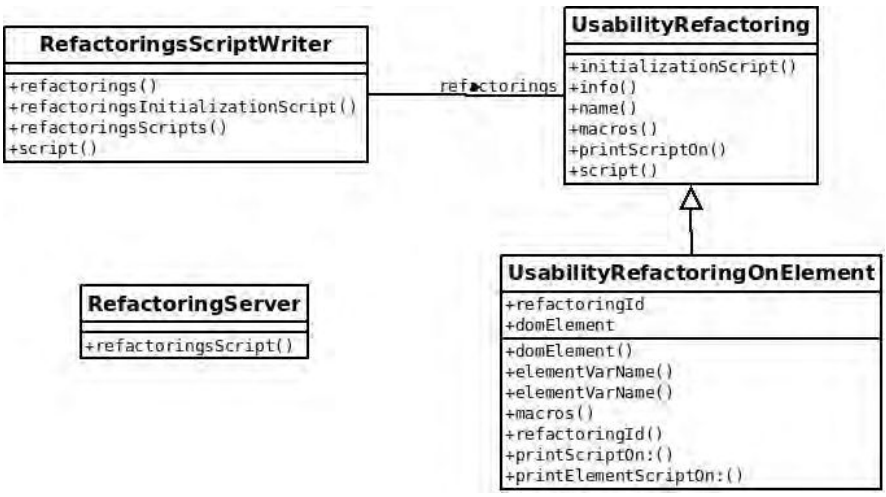


Figura 57. Modelo parcial de Usability Refactorings.

La aplicación de los refactorigs es invocada desde el componente del lado del cliente, utilizando un servicio de Api Rest. Mediante request a la clase RefactoringsServer se obtienen instancias en cada cuenta de usuario, con el código JavaScript generado. La clase RefactoringsScriptWriter inicia el proceso creando un conjunto de refactorings y generando el código completo mediante el método #script.

A su vez, este método invoca otros dos en cada refactoring:

#refactoringsInitializationScript que carga el código único para cada familia de refactorings mediante el método de clase #initializationScript.

#refactoringsScripts que genera el código particular requerido para cada instancia del refactoring.

Con el método #script cada refactoring invoca su mecánica de generación de código. En la clase UsabilityRefactoringOnElement contiene el código para los refactorings que se aplican sobre un elemento DOM e implementa la generación del script agregando una variable para el elemento afectado.

```
var element2db3bee8e7100d00a4bab16a0fda3de9 =

 $(xpathInstance.getElementByXpath(

'/html/body/div[3]/div[2]/div/div/div[2]/form'

)

);
```

Figura 58. Código Javascript generado por la clase UsabilityRefactoringOnElement

El nombre de la variable element2db3bee8e7100d00a4bab16a0fda3de9, en la figura 58, corresponde al identificador único para el elemento afectado, obtenido a partir de su xpath en la página utilizando la función getElementByXpath del objeto xpathInstance transformado en un objeto jQuery.

Las subclases de UsabilityRefactoringOnElement hacen referencia a esta variable y generan código para manipular el elemento dentro del DOM. Estos scripts se basan en el uso de dos herramientas: streams y macros. Los streams son objetos que facilitan la concatenación de strings que funcionan como canales que consumen strings. Los macros son cadenas especiales que pueden introducirse en un string para ser sustituidas por valores específicos.

8.3 Catálogo de Refactoring

A continuación se presenta cada **Accessibility Refactoring** del catálogo describiendo su implementación y la aplicación de cada uno en un caso real en la Web.

R1 – NotifyScreenReadersUsingAlertRole: *Notificar a los lectores de pantalla usando el rol de alerta.*

Se inserta en la página Web un contenedor con un rol de alerta, que exhibe en la pantalla un párrafo con texto configurado en la instancia del refactoring. Este texto es suministrado al screen reader que lo sintetiza al usuario.

La implementación se aplica en los casos en que el **Accessibility Smell Validation Test Without Electronic Text Validation** (prueba de validación sin texto electrónico), afecta a un campo de entrada de texto dentro de un formulario HTML.

Este refactoring se realiza en dos pasos:

1. Se crea un elemento de párrafo oculto dentro de un y se agrega al formulario.
2. Se asignan acciones en los eventos del campo de entrada afectado, para que el párrafo se visualice en caso de que las condiciones del **Accessibility Smell** se presenten y el screen reader necesite sintetizar su contenido.

Una aplicación del refactoring se observa en el Caso Real 1 en el cual el usuario deseaba ingresar un destino para su vuelo en la Página de Aerolíneas Argentinas.

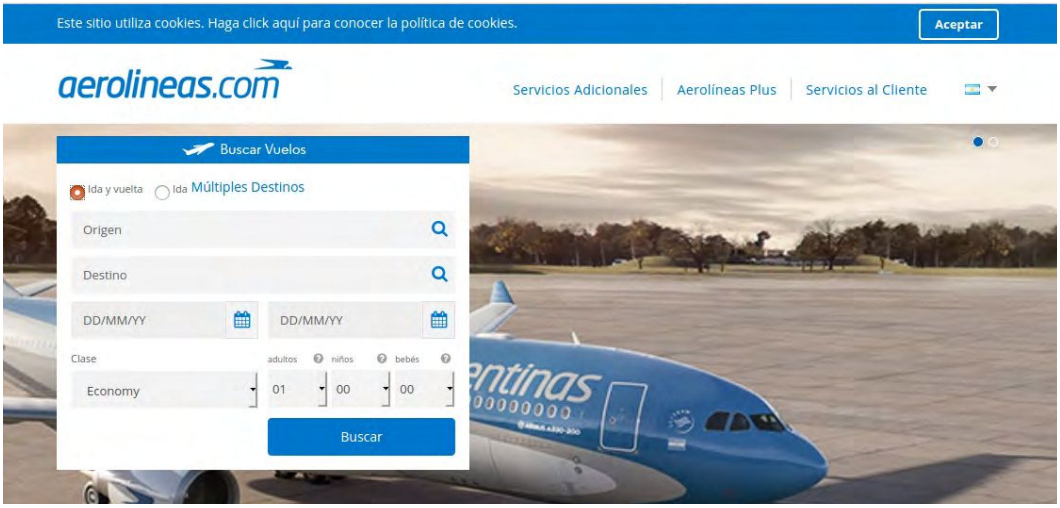


Figura 59: Página Web de Aerolíneas Argentinas antes del Refactoring ***NotifyScreenReadersUsingAlertRole****.*

Ante la pérdida del foco en el campo de entrada de origen se elimina automáticamente el dato ingresado por lo que debe desplegar el texto configurado en la instancia del refactoring.

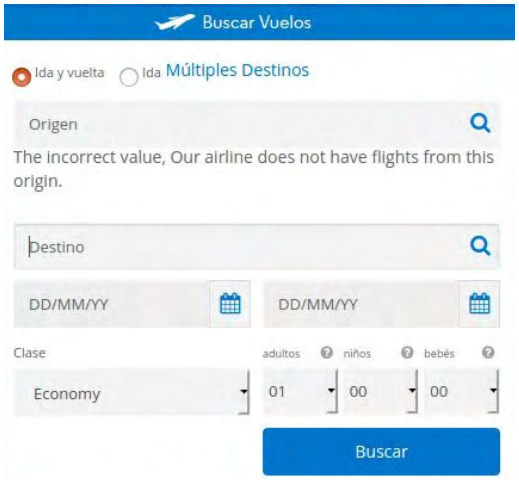


Figura 60: Página Web de Aerolíneas Argentinas después del Refactoring NotifyScreenReadersUsingAlertRole.

Como se exhibe en la figura se despliega el texto "The incorrect value, Our airline does not have flights from this origin.", reconocido por NVDA como electrónico y sintetizado al usuario.

R2 – AddDescriptionToEntryConfrols Agregar descripción a los controles de entrada

Al crear una instancia del refactoring se configura texto dentro de una etiqueta <label> vinculada con el elemento afectado por el **Accessibility Smell**. Su contenido puede ser suministrado al Screen reader **NVDA** como texto electrónico que sintetizar a los usuarios.

El refactoring se aplica en casos en que el **Accessibility Smell Input field without label that describes it** (*Campo de entrada sin etiqueta que lo describa)* afecte a un campo de entrada

representado por una entrada de texto (<input type=text>), cajas de chequeo (checkbox) o botones de radio (radiobutton).

El refactoring se realiza en dos pasos:

1. Se crea una nueva etiqueta <label> en la posición de la página previa al campo de entrada afectado.
2. Se vincula la etiqueta mediante el atributo “for” con el campo de entrada.

Una aplicación del refactoring se puede observar en el Caso Real 2 referido a las Reservas de vuelos en la página de Aerolíneas Argentinas.

Analizando la síntesis de voz del formulario web, durante el ingreso del origen para el vuelo, se detecta que no cuenta con una etiqueta, <label> vinculada.

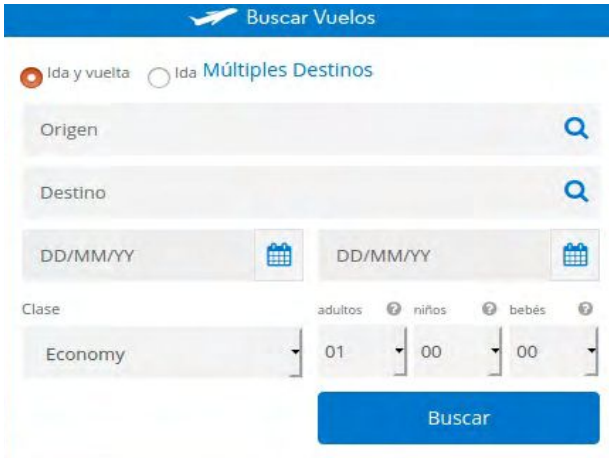


Figura 61: Página Web de Aerolíneas Argentinas antes del Refactoring AddDesciptionToEntryConfrols.

Dentro del código HTML del formulario se observa que existe la etiqueta <label>, pero vinculada con un elemento de selección, con valor de ID=”Id_Origen”, y no con el campo de entrada para el origen del vuelo. Por ese motivo al posicionar el foco **NVDA** reconoce como texto electrónico el valor de placeholder="Origen" y sintetiza al usuario el texto “Origen edición en blanco”.

```
<fieldset class="form-group validationTooltip">
 <label for="Id_Origen" class="sr-only">Origen</label>
  <select data-val="true" data-val-number="El campo Id_Origen debe ser un número." id="Id_Origen" name="Id_Origen" title="Origen" tabindex="-1" style="display: none;" class="selectized">
<div class="selectize-control single plugin-restore_on_backspace">
<div class="selectize-input items not-full has-options">
<input type="text" autocomplete="off" tabindex="" placeholder="Origen" style="width: 69px; opacity: 1; position: relative; left: 0px;">
```

Figura 62: Fragmento de código HTML del Formulario de Reserva de Vuelos de la Página de Aerolíneas Argentinas.

Aplicado el refactoring se inserta en el formulario una etiqueta <label> vinculada al campo afectado por el Accessibility Smell que incluye un texto configurado por el usuario.

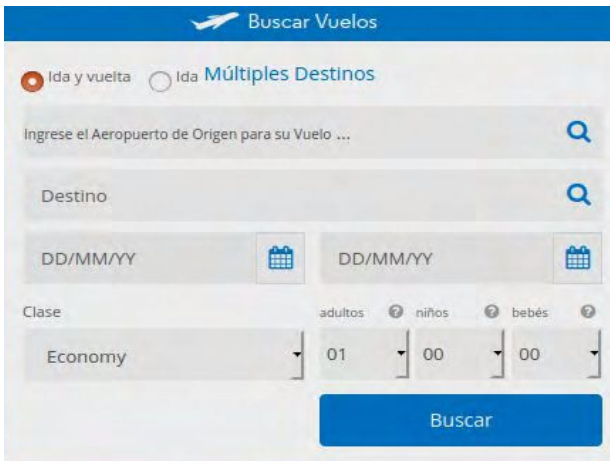


Figura 63: Página Web de Aerolíneas Argentinas luego del Refactoring AddDesciptionToEntryConfrols

Ahora el campo afectado cuenta con su etiqueta <label>. En la imagen anterior no se visualiza el cambio debido a los estilos modifican visualmente la presentación web. Sin embargo la etiqueta esta presente y el texto “Ingrese el Aeropuerto de Origen para su Vuelo” se sintetiza al usuario.

R3 – AddAriaLabel *Añadir etiqueta Aria*

Accessible Rich Internet Applications (ARIA), es una especificación técnica de la W3C destinada para aumentar la accesibilidad de los contenidos e interfaces dinámicas en HTML.

ARIA describe como agregar semántica y metadatos al contenido HTML con el fin de hacerlo disponible especialmente para personas con alguna capacidad reducida.

Este refactoring, se vale de las propiedades que ofrece la implementación de ARIA al posibilitar, con solo agregar propiedades y estados como roles HTLM, sea posible informar al usuario de las condiciones y/o acciones sobre un elemento de la presentación web.

La implementación se aplica para los casos en que el **Accessibility Smell Absence of descriptive text for text entry fields** (*Ausencia de texto descriptivo para los campos de entrada)*, afecte a un campo representado por una entrada de textos (<input type=text>), cajas de chequeo (checkbox) o botones de radio (radiobutton).

El refactoring inserta texto en el campo de entrada en su atributo **aria-label** configurado durante la creación de la instancia del refactoring. El etiquetado arial tiene soporte en los lectores de pantallas y **NVDA** reproduce su contenido, como texto electrónico cuando el foco se posa sobre el elemento al cual se aplica.

Retomando el Caso Real 3 referido al Sistema de consultas del registro patrimonial de la Universidad Nacional de Salta, al posicionar el foco sobre el elemento de entrada de datos destinado al Nro. de Documento no se sintetizan ningún texto que sea descriptivo de la naturaleza del dato requerido.

Figura 64: Sistema de Patrimonio SABUM de la Universidad Nacional de Salta.

Analizando el código HTML se observan la presencia de condiciones del **Accessibility Smell Absence of descriptive text for text entry fields** (*Ausencia de texto descriptivo para los campos)* ante la existencia de una etiqueta <label> vinculada al campo de datos, ni valores para sus atributos aria-label ni *placeholder.*

```
<input name="documento" size="8" value="">
```

Figura 65: Código HTML del campo destinado al ingreso del Número de Documento antes del Refactoring ***AddAriaLabel*** *- Sistema de Patrimonio SABUM de la Universidad Nacional de Salta.*

Luego de aplicado el refactoring el campo cuenta con un valor de aria-label, que **NVDA** reconoce como texto electrónico y lo sintetiza al usuario al momento de posicionar el foco dentro del campo afectado por el **Accessibility Smell**.

```
<input name="documento" size="8" value="" aria-label="Ingrese el Número de Documento del Agente de la Universidad">
```

Figura 66: Código HTML del campo destinado al ingreso del Número de Documento después del Refactoring ***AddAriaLabel*** *- Sistema de Patrimonio SABUM de la Universidad Nacional de Salta.*

R4 -AddPlaceholderText *Añadir texto de ayuda*

En HTML el atributo de marcador **Placeholder** ofrece pistas breves que describen el valor esperado en un campo de entrada. Esas pistas consisten en un texto que se exhibe al usuario al solicitarle el ingreso de un valor.

Este refactoring se aplica ante la presencia del **Accessibility Smell Absence of descriptive text for text entry fields (***Ausencia de texto descriptivo para los campos de entrada)*, que afecta a un campo representado por una entrada de textos (<input type=text>), cajas de chequeo (checkbox) o botones de radio (radiobutton).

El refactoring inserta para el campo de entrada en su atributo **Placeholder** texto que es configurado durante la creacion de su intancia. El atributo **Placeholder** tiene soporte por los lectores de pantallas y **NVDA** reproduce su contenido, como texto electrónico, al posicionar el foco sobre el elemento al cual se aplica.

Este refactoring, también se puede aplicar al caso tratado anteriormente para el *Sistema de Patrimonio SABUM de la Universidad Nacional de Salta* dado que consideramos el mismo **Accessibility Smell**.

```
<input name="documento" size="8" value="">
```

Figura 67: Código HTML del campo destinado al ingreso del Número de Documento antes del Refactoring ***AddPlaceholderText*** *- Sistema de Patrimonio SABUM de la Universidad Nacional de Salta.*

Luego de aplicado el refactoring el campo cuenta con un valor de *placeholder* que **NVDA** reconoce como texto electrónico y lo sintetiza al usuario al momento de posicionar el foco dentro del campo afectado por el Bad Smell.

```
<input name="documento" size="8" value="" placeholder="Ingrese el Número de Documento del Agente de la Universidad">
```

Figura 68: Código HTML del campo destinado al ingreso del Número de Documento después del Refactoring AddPlaceholderText - Sistema de Patrimonio SABUM de la Universidad Nacional de Salta.

La ventaja en este caso, comparado con ***AddAriaLabel*** radica en que también se visualiza el texto del PlaceHolder en la presentación web, lo que aumenta la accesibilidad para los usuarios con un bajo grado de dificultar visual.

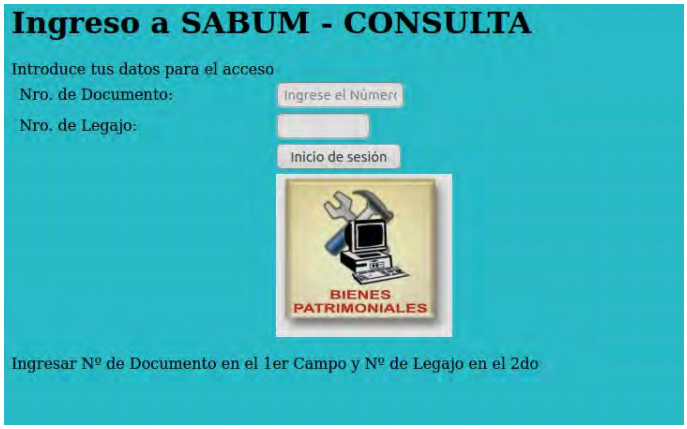


*Figura 69: Sistema de Patrimonio SABUM de la Universidad Nacional de Salta luego de aplicado el Refactoring **AddPlaceholderText**..*

R5 – NewLayoutOfInterfaceElement *Nuevo diseño del elemento de interfaz*

El refactoring posiciona el elemento afectado por el **Accessibility Smell Distant content for keyboard use on a page** (*Contenido distante para uso del teclado en una página)* a una nueva ubicación en la presentación web. Se altera el orden y la ubicación del elemento en la página para nuevos valores de top y left, reduciendo la distancia en el orden de tabulación para el usuario.

La implementación se realiza en dos pasos:

1. Se modifica la propiedad tabindex del elemento afectado.
2. El elemento se agrega en un contenedor que se incluye en la página en la nueva posición.

Para visualizar el refactoring retomamos Caso Real 4, referido a la página web de la Universidad Nacional de Salta, enfocándonos en un vídeo vinculado al sitio YouTube. Este elemento está contenido dentro de un enlace afectado por el **Accessibility Smell** dado que se requieren 34 tabulaciones desde el inicio de la página para acceder a su visualización.

En la tabla a continuación se puede compara la página antes de aplicar el refactoring **NewLayoutOfInterfaceElement** y después de modificar la interfaz. La flecha indica el desplazamiento del vídeo en la presentación.

Aplicación del Refactoring **NewLayoutOfInterfaceElement**	
Antes	Después
	

*Figura 70: Refactoring **NewLayoutOfInterfaceElement** en la Página Web de la Universidad Nacional de Salta.*

R6 – AddLink *Añadir enlace*

En una lista de páginas web navegadas la aplicación del refactoring inserta en la primera página de la secuencia un enlace directo hacia la última, evitando al usuario recorrer los pasos intermedios. Este refactoring se aplica cuando el **Accessibility Smell Distant Content in Accessibility (***Contenido distante en accesibilidad)* dificulta el acceso a un contenido requiriendo un camino más largo de navegación para acceder a la información incluida en el nodo final.

El enlace directo en la primera página es creado en la ubicación siguiente al enlace utilizado para acceder a la segunda página de la secuencia.

La implementación se realiza en dos pasos:

1. Se identifica el enlace dentro de la primera página de la secuencia que recorren los usuarios.

2. Se inserta un enlace en la primera página con destino a la última en la ubicación posterior al enlace identificado en el paso anterior.

En el caso de la consulta del menú del comedor universitario, en la página de la Universidad Nacional de Salta, los usuarios recorren tres páginas.

N°	Páginas Recorridas
1	http://www.unsa.edu.ar/web/index.php
2	http://www.unsa.edu.ar/web/index.php? ption=com_content&view=category&id=14&Itemid=230
3	http://www.unsa.edu.ar/web/index.php?option=com_content&view=article&id=15&Itemid=229

Figura 71: Secuencia de páginas recorridas por los usuarios en el ***Accessibility Smell Distant Content in Accessibility***

La aplicación del refactoring para los usuarios que utilizan el teclado y **NVDA** resuelve una importante dificultad. Durante el recorrido en cada página intermedia deben recorrer una gran cantidad de elementos, escuchando las síntesis de voz hasta posicionar el foco sobre al enlace que los deposita en la siguiente página.

En la figura 72, se inserta un nuevo enlace con el texto "Menú *del Comedor*" a continuación del enlace "Comedores Estudiantiles".

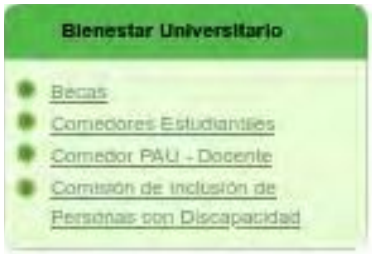
Aplicación del Refactoring **AddLink**	
Antes	Después
	
Menú en la Primera página de la Secuencia http://www.unsa.edu.ar/web/index.php	Menú en la última página de la Secuencia http://www.unsa.edu.ar/web/index.php? option=com_content&view=article&id=15&Itemid=229

Figura 72: Aplicación del refactoring ***AddLink*** *en la Página Web de la Universidad Nacional de Salta.*

De esta manera, el usuario puede acceder al menú sin necesidad de pasar por la segunda página de la secuencia. En el ejemplo los pasos intermedios que los usuarios evitan

recorres incluyen un solo nodo, pero en la web la implicación de una camino más largo significa una mayor dificultad para los usuarios de **NVDA**.

R7 – AddSubmitButton *Agregar botón Enviar*

El refactoring agrega a un formulario en la página web un botón de envío “submit”. La implementación resuelve los inconvenientes que se presentan ante el **Accessibility Smell Forms without send button (***Formularios sin botón enviar) y* consiste en insertar un botón “submit” dentro del formulario con texto descriptivo que el usuario suministra al crear la instancia del refatoring.

Un ejemplo es la consulta de los resúmenes de cuenta del Sistema de Autogestión Web de la Obra Social OSUNSa.

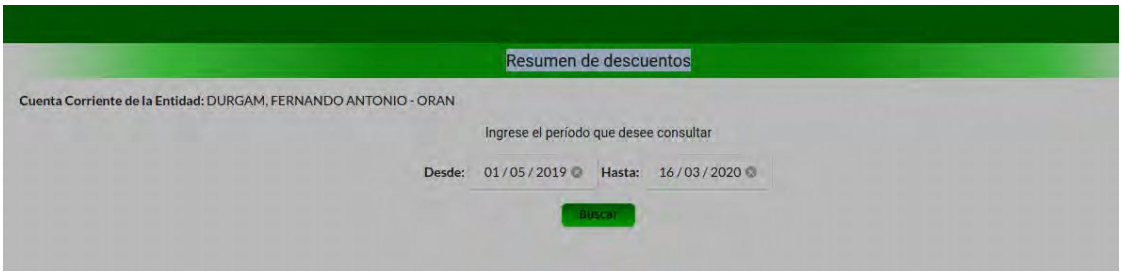


Figura 73: Sistema de Autogestión Web de la Obra Social OSUNSa antes del Refactoring ***AddSubmitButton****.*

Estos resúmenes son inaccesibles para los usuarios con dificultades visuales que utilizan **NVDA** con el teclado, debido a que el botón de búsqueda no corresponde a un elemento HTML de tipo *Submit.* Es un botón que invoca una función como respuesta a un evento *click y* el formulario no puede ser enviado. Esta situación se describe en el **Accessibility Smell Forms without send button (***Formularios sin botón enviar).*

Al aplicar el refactoring se inserta un botón de tipo submit en el formulario que los usuarios pueden acceder con el teclado.

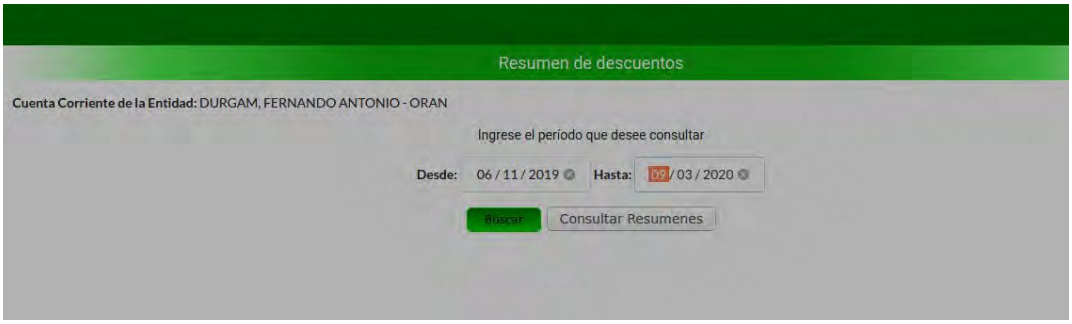


Figura 74: Sistema de Autogestión Web de la Obra Social OSUNSa después del Refactoring AddSubmitButton

De esta manera los usuarios que utilizan teclados podrá recorrer los elementos hasta encontrar el botón de *submit* y utilizarlo para enviar del formulario.

El nuevo botón no invoca ni suprime los manejadores del evento click del botón original, situación por la cual podrían no ejecutarse tareas vinculadas al evento.

R8 – AddFeedbackToSearch *Añadir comentarios a la búsqueda*

El refactoring agrega feedback a los procesos de búsquedas en una página web, incluyendo textos electrónicos que son accedidos por los lectores de pantalla. El refactoring se aplica para resolver la dificultad **Accessibility Smell Search without feedback (***Búsqueda sin retroalimentación)* cuando una búsqueda no suministra textos electrónicos sobre los resultados.

La implementación se realiza en dos pasos:

1. Se incrusta un elemento de audio en la página web que accede a un servicio externo para sintetizar texto y reproducirlo al usuario. El servicio disponible en spanishdict.com sintetizar a voz del texto configurado por el usuario al crear una instancia del refactoring.

2. El texto configurado se almacena en un párrafo dentro de un contenedor que permanece oculto en la página y se sintetiza a voz al presionar <enter> sobre un cuadro de texto, un botón, o al enviar un formulario.

En el Caso Real 1.6, el usuario de **NVDA** busca el texto ”despacito” en la página de Youtube y se presenta el **Accessibility Smell Search without feedback (***Búsqueda sin retroalimentación)*. Resulta necesario incorporar una síntesis de voz a la búsqueda que le indique al usuario que su solicitud fue procesada.

Este refactoring no produce ningún cambio en la presentación web, inserta elementos en la página extendiendo su funcionalidad por medio de una API, disponible en audio1.spanishdict.com, para incorporar el texto electrónico configurado al crear la instancia del refactoring.

R9 -Modify Focus Sequence *Modificar secuencia de enfoque*

El refactoring modifica la secuencia de navegación por teclado de los elementos en la página a un nuevo orden más accesible.

La implementación se aplica ante la presencia del **Accessibility Smell *Disposition of confusing elements (****Disposición de elementos confusa)* en el cual una incorrecta secuencia de navegación con tecla de tabulación puede dificultar o impedir al usuario interactuar con la página web.

La implementación consiste 2 pasos:

1. Identificar los elementos focables afectados.

2. Asignar a cada elemento un número de secuencia en el campo tabIndex, acorde a su

orden de aparición en el DOM de la página.

Retomando el Caso Real 1.8 referido a la Solicitud de becas del sistema Siu Tehuelche, se observa un acceso secuencial que dificulta la interacción del usuario debido a una secuencia de tabulación no coincidente con la lógica de despliegue en la presentación web, **Accessibility Smell *Disposition of confusing elements*** *(Disposición de elementos confusa).*

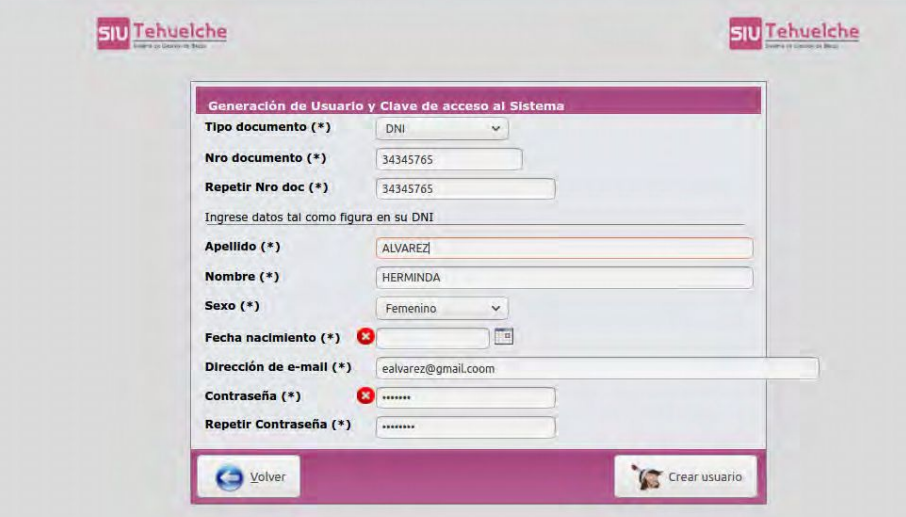


Figura 75: Sistema de Solicitud de becas del sistema Siu Tehuelche antes del Refactoring ***Modify Focus Sequence***

Al *datepicker* que facilita el ingreso de la fecha de nacimiento se accede luego del botón *submit* etiquetado como “Crear usuario”. La aplicación del refactoring reajusta la secuencia de tabulación de los elementos con un orden lógico más conveniente para la interacción.

Figura 76: Fragmento del formulario del sistema Siu Tehuelche después del Refactoring ***Modify Focus Sequence***

La aplicación del refactoring altera la secuencia del foco donde:

(1) Al abandonar el campo de entrada para la fecha de nacimiento, el foco se desplaza hacia el datepicker para seleccionar una fecha.

(2) Al abandonar el datepicker, el foco se desplaza al campo de ingreso para el email.

(3) Al abandonar campo de ingreso para el email, la secuencia continua con el orden de tabulación hacia el siguiente elemento en la presentación de la página.

R10 – AddDatePickerAccessible *Agregar selector de fecha accesible*

El refactoring altera los campos de entrada de texto, utilizados para el ingreso de fechas en un formulario. La especificación HTML 5 incluye un tipo de campo de entrada *Date* que puede ser accedido por los screen reader y responder a las entradas de teclado. Actualmente la versión de **Usabitity Smell de Kobold** incluye un refactoring no accesible, que agrega un datepicker que no configura manejadores de eventos de teclado.

La implementación resuelve los inconvenientes que se presentan ante el **Accessibility Smell Datepicker inaccessible using the keyboard (***Datepicker inaccesible usando el teclado)* y consiste en modificar los tipos de campo de entradas de textos, transformándolos en entradas accesible para los valores de fecha.

La implementación consiste en 2 pasos:

1. Reemplazar todos los campos de entradas de tipo texto, destinados a valores de fecha con campos tipo *Date*.
2. Ocultar las imágenes destinadas a desplegar calendarios popup *DatePicker,* para la selección de fechas.

En el caso tratado anteriormente, dentro del formulario para el alta de usuarios existe un componente Datepicker que no responde a los eventos del teclado. El Datepicker es inaccesible utilizando el teclado por la presencia del Accessibility Smell Datepicker inaccessible using the keyboard(Datepicker inaccesible usando el teclado).

Figura 77: Sistema de Solicitud de becas Siu Tehuelche luego del Refactoring AddDatePickerAccessible

El refactoring reemplaza el DatePicker de forma que sea posible utilizar las teclas de desplazamiento (arriba y abajo) a medida que el screen reader **NVDA** sintetizar texto.

R11- ReplaceCaptcha *Reemplazar Captcha*

Las pruebas humanas interactivas brindan seguridad al diferenciar entre usuarios humanos y máquinas para evitar el spam en internet.

Los CAPTCHA (*Completely Automated Public Turing test to tell Computers and Humans Apart*) se implementan, como pruebas de Turing donde el juez es un dispositivo que realiza preguntas que solo usuarios humanos podrán responder.

Basados en inteligencia artificial estos componentes proponen retos con texto, imágenes y/o sonido, con serias limitaciones a la accesibilidad web para los usuarios con dificultades visuales.

Debido a que los CAPTCHA dependen de funciones alojadas en el servidor, cualquier intento de solución al **Accessibility Smell Captcha in the forma** (*Captcha en el formulario)* requiere modificar el back-end de la aplicación, porque la función de reconocimiento del usuario está condicionado por estados que cambian en cada request al servidor.

Este caso excede las soluciones de refactoring sobre las presentaciones web. Se recomienda realizar modificaciones como los desarrollos de Re- CAPTCHA v3 de Google que verifican las solicitudes con una puntuación basada en calificaciones de comportamiento. Al no requerir de forma obligada superar un desafió, una instancia de Re- CAPTCHA v3 de Google puede determinar de forma transparente si se trata de un usuario humano.

Si analizamos nuevamente el Caso Real 1.5 referido a una Solicitud de Clave Única de Identificación Laboral en la Página del ANSES.

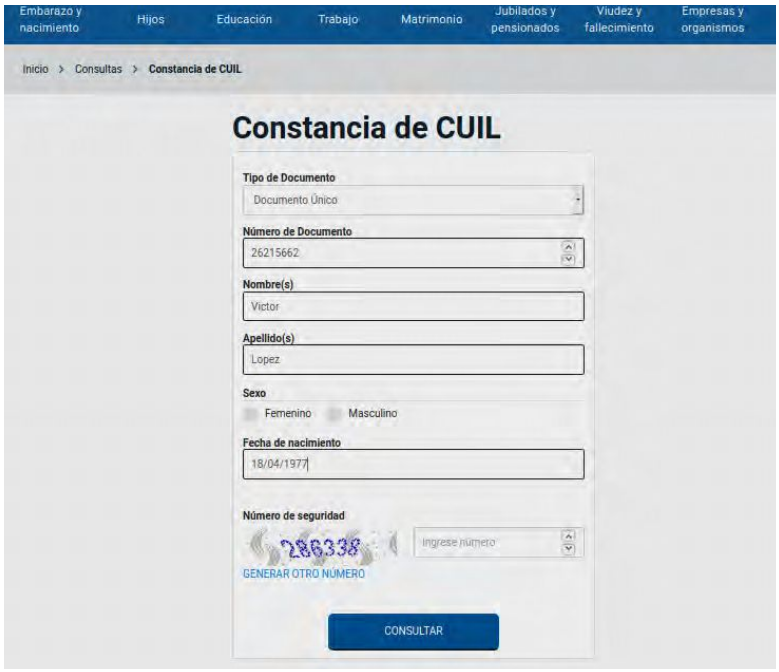


Figura 78: Solicitud de Constancia de Cuil en la Página web de Anses.

El **Accessibility Smell *Captcha in the form (****Captcha en el formulario)* impide al usuario continuar utilizando la aplicación y es necesario reemplazar el Captcha.

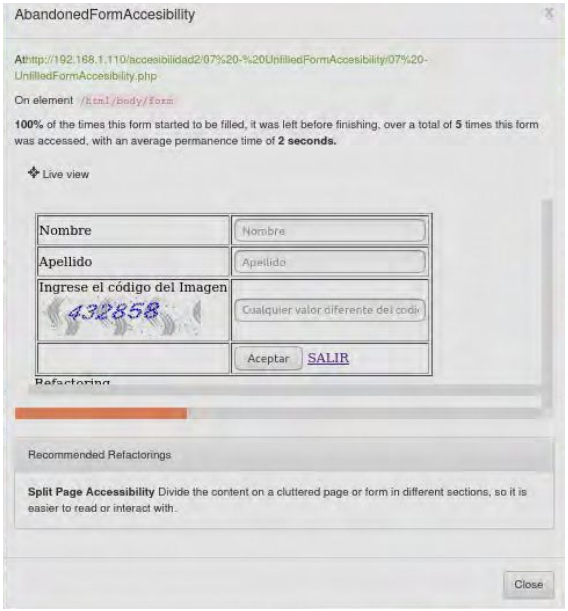


Figura 79: Reporte para Accesibilidad de Kobold sugiriendo la aplicación del Refactoring ***ReplaceCaptcha.***

En caso de eliminar el Captcha, la página ofrecería al usuario un formulario sin requerir explícitamente superar un desafío.

Figura 80: Solicitud de Constancia de CUIL luego de Reemplazar el Capcha en la Página web de Anses.

R12- SplitPage *Dividir Página*

Este refactoring, existente en Kobold (Grigera, 2017), también se aplica a la versión de accesibilidad como respuesta a la presencia de los **Accessibility Smell Distant content for keyboard use on a page** (*Contenido distante para uso del teclado en una página)* y **Overlooked Content** (*Contenido pasado por alto). C*uando las páginas web se encuentran abarrotadas de

contenido, los usuarios deliberadamente deciden ignorarlos. El refactoring consiste en dividir un contenido demasiado extenso en múltiples páginas que se vinculan de alguna manera.

Según el tipo de contenido, puede resultar conveniente diferentes formas de dividir la página, como combinación de link o formularios en etapas. Ante la complejidad de automatización el refactoring Split Page es sugerido.

Figura 81: Reporte para Accesibilidad de Kobold sugiriendo la aplicación del Refactoring SplitPage.

R13- SplitPageForNVDA Página dividida para NVDA

Este refactoring es una adecuación del refactoring Split Page, para los contenidos salteados en el **buffer virtual** de **NVDA**. Ante un **Accessibility Smell Overlooked Content For NVDA** *(contenido distante para NVDA)* se genera una sugerencia de refactorización. A diferencia del refactoring anterior los desplazamientos posicionan el cursor virtual en elementos intermedios específicos no focables. Informar a los desarrolladores sobre esos elementos puede resultar de gran utilidad al momento de diseñar el Split. En cada sugerencia se incluyen referencias a los elementos recorridos.

Figura 82: Reporte para Accesibilidad de Kobold sugiriendo la aplicación del Refactoring ***SplitPage*** *conteniendo los identificadores de los elementos recorridos por los usuarios durante los desplazamientos.*

R14–AddLanguageFieldElement *Agregar el valor de idioma al campo de entrada*

El refactoring agrega o modifica el atributo de lenguaje *lang* del elemento afectado por el **Accessibility Smell** ***Unexpected language change*** *(Cambio de idioma inesperado)* y de sus etiquetas y <p> contenidas.

El valor adecuado del idioma se obtiene a partir de un sistema de detección disponible como API. Para alterar la definición, la implementación se basa en el idioma informado por el **Accessibility Smell,** que se determina durante los eventos de accesibilidad.

Las instancias de la extensión **HttpRequestIdiomas** utilizan el servicio ofrecido por el sitio ws.detectlanguage.com para detectar el idioma de los textos contenidos en el elemento afectado y en el título de la página web.

Para aplicar los cambios se comparan el idioma de la página Web, el idioma de su título y el idioma configurado en el navegador.

Retomando el Caso Real 1.7 referido al Catálogo de la biblioteca de la Universidad Nacional de Salta, la aplicación del refactoring no altera la presentación de la página web. Incluye atributos de idioma *lang* en los elementos afectados por el bad Smell ***Unexpected language change*** *(Cambio de idioma inesperado).*

En la siguiente figura se reporta un elemento de tipo enlace <a> afectado por el Bad Smell.

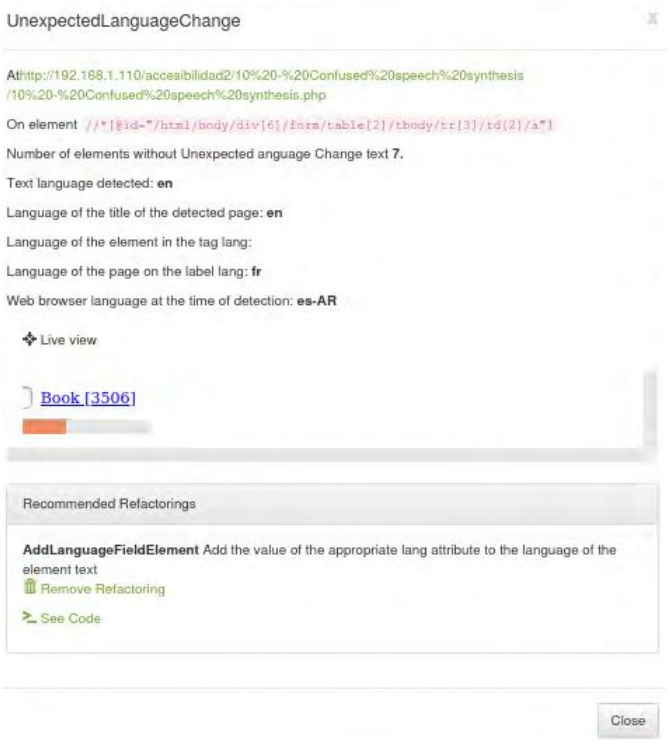


Figura 83: Reporte del Bad Smell ***Unexpected language change*** *sobre un enlace Book.*

El reporte indica que el idioma detectado tanto para el texto del enlace como para el título de la página es el inglés (“en”), que la página tiene definido el idioma francés (“fr”) y el navegador web está configurado en español (es-Ar).

El código HTML del enlace afectado en la página es el siguiente:

```
<a href="./index.php?
lvl=more_results&amp;mode=extended&amp;facette_test=1&amp;name=Section&amp;value=Libros&amp;champ
=90&amp;ss_champ=3" id="/html/body/div[6]/form/table[2]/tbody/tr[3]/td[2]/a">

<span class="facette_libelle">Book</span>

<span class="facette_number">[3506]</span>

</a>
```

*Figura 84: Código HTML del enlace antes de aplicar el Refactoring **AddLanguageFieldElement**.*

Luego de aplicado el refactoring los elementos afectados disponen de atributos que indican su idioma <lang=”en”>.

```
<a href="./index.php?
lvl=more_results&amp;mode=extended&amp;facette_test=1&amp;name=Section&amp;value=Libros&amp;champ=9
0&amp;ss_champ=3" id="/html/body/div[6]/form/table[2]/tbody/tr[3]/td[2]/a" lang="en">

<span class="facette_libelle" lang="en">Book</span>

<span class="facette_number" lang="en">[3506]</span>

</a>
```

*Figura 85: Código HTML del enlace después de aplicar el Refactoring **AddLanguageFieldElement**.*

R15-New Position For List On The Page *Nueva posición para la lista en la página*

El refactoring desplaza una lista de enlaces a una nueva ubicación dentro de una página. La solución se aplica ante la presencia del **Accessibility Smell *Distant Content For List Menu for Complement NVDA*** *(Contenido distante para el menú de lista del complemento NVDA)* cuando los usuarios recorren en una página, siempre el mismo camino que contienen listas de enlaces intermedias.

La implementación consiste en 3 pasos:

1. Identificar los elementos inicial y final del camino de listas de enlaces.
2. Determinar la inclusión de los elementos en contenedores HTML (por Ej.).
3. Reubicar en la página al elemento final o su contenedor, en la posición inmediata anterior al elemento inicial o su contenedor.

Retomando el Caso Real 3.2 referido la página Web de la Universidad Nacional de Salta y una secuencia de listas recorridas ante el **Accessibility Smell *Distant Content For List Menu for Complement NVDA.*** La navegación incluye identificadores de cada lista en el camino:

- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div[3]/div/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div[3]/div[2]/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div[3]/div[3]/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div[3]/div[4]/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div[3]/div[5]/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div[3]/div[4]/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div[3]/div[3]/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div[3]/div[2]/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div[3]/div/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div/div[4]/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div/div[3]/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div/div[3]/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div/div[2]/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div/div/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div/div[2]/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div/div[3]/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div/div[4]/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div[3]/div/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div[3]/div[2]/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div[3]/div[3]/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div[3]/div[4]/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div[3]/div[5]/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div[3]/div[6]/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div[3]/div[7]/div/div[2]/div[10]/ul
- <Lista:>/html/body/div[6]/div/div[10]/div[3]/div[3]/div[8]/div/div[2]/div[10]/ul

Los usuarios navegan entre 25 listas de enlaces antes de acceder a la sección deseada.

Aunque no afectan la aplicación del refactoring, pueden reiterarse listas de enlaces en el camino recorrido. En los casos enumerados en 1 y 9 las interacciones avanzan y retroceden en el **buffer virtual.**

El refactoring desplaza la lista ubicada en el orden 25, hacia la ubicación anterior de la lista ubicada en el orden 1. Se puede observar visualmente su impacto comparando las imágenes a continuación:

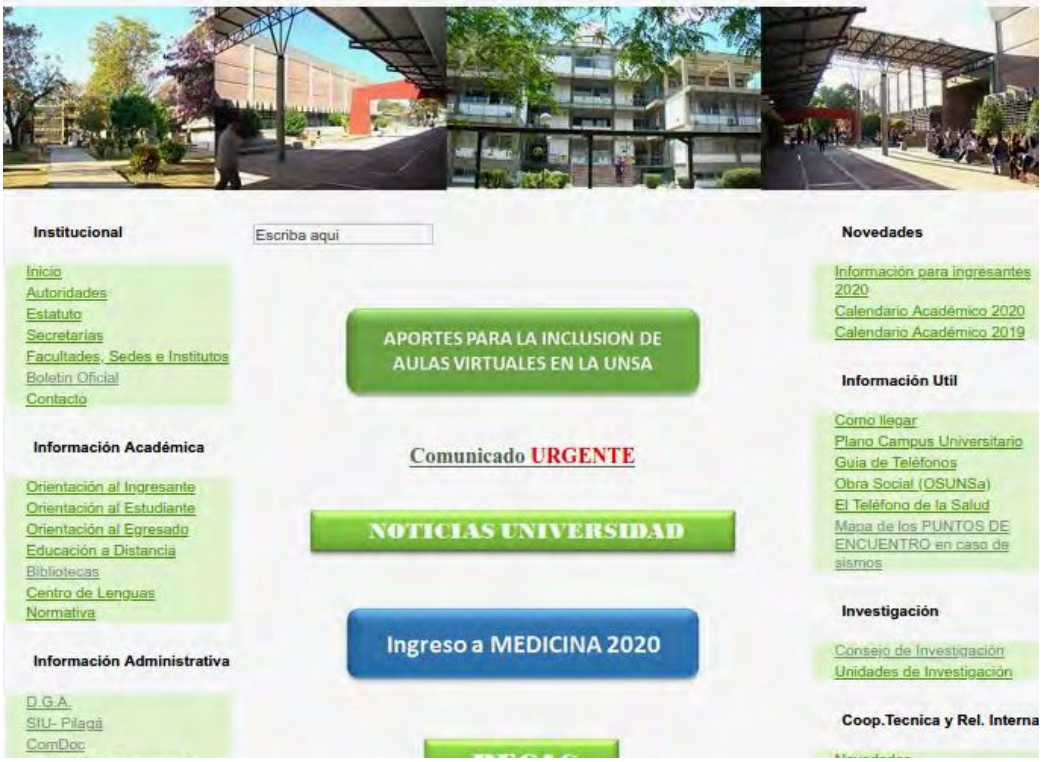


Figura 86: Página Web de la Universidad Nacional de Salta antes del Refactoring ***New Position Fo List On The Page.***

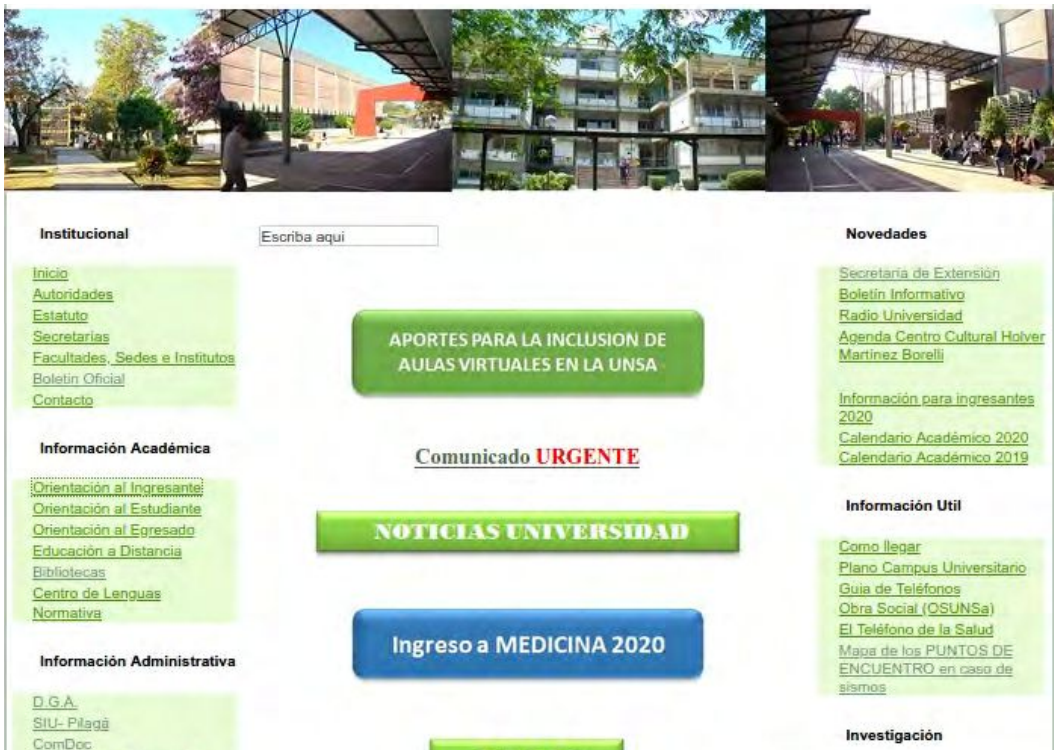


Figura 87: Página Web de la Universidad Nacional de Salta despues del Refactoring ***New Position For List On The Page.***

En la lista de *Novedades* se ha desplazado la *Secretaria de Extensión, Boletín Informativo, Radio Universidad y Agenda Centro Cultural Holver Martines Borrelli,* haciendo los contenidos más accesibles.

R16 - Add Anchor on Page *Agregar ancla en la página.*

El refactoring agrega un ancla hacia una lista de enlaces. A igual que el caso anterior, esta solución también se aplica al **Accessibility Smell *Distant Content For List Menu for Complement NVDA (****Contenido distante para el menú de lista del complemento NVDA),* pero en lugar de reubicar el elemento final de la lista, inserta un ancla hacia la ubicación del mismo.

La implementación consiste en 2 pasos:

1. Identificar los elementos inicial y final del camino en la lista de enlaces.

2. Crear, en la ubicación anterior al elemento inicial, un enlace hacia el elemento final con el texto descriptivo que el usuario configura durante la creación de la instancia del refactoring.

En el caso anterior, con idéntica navegación entre listas se puede aplicar el mismo el refactoring.

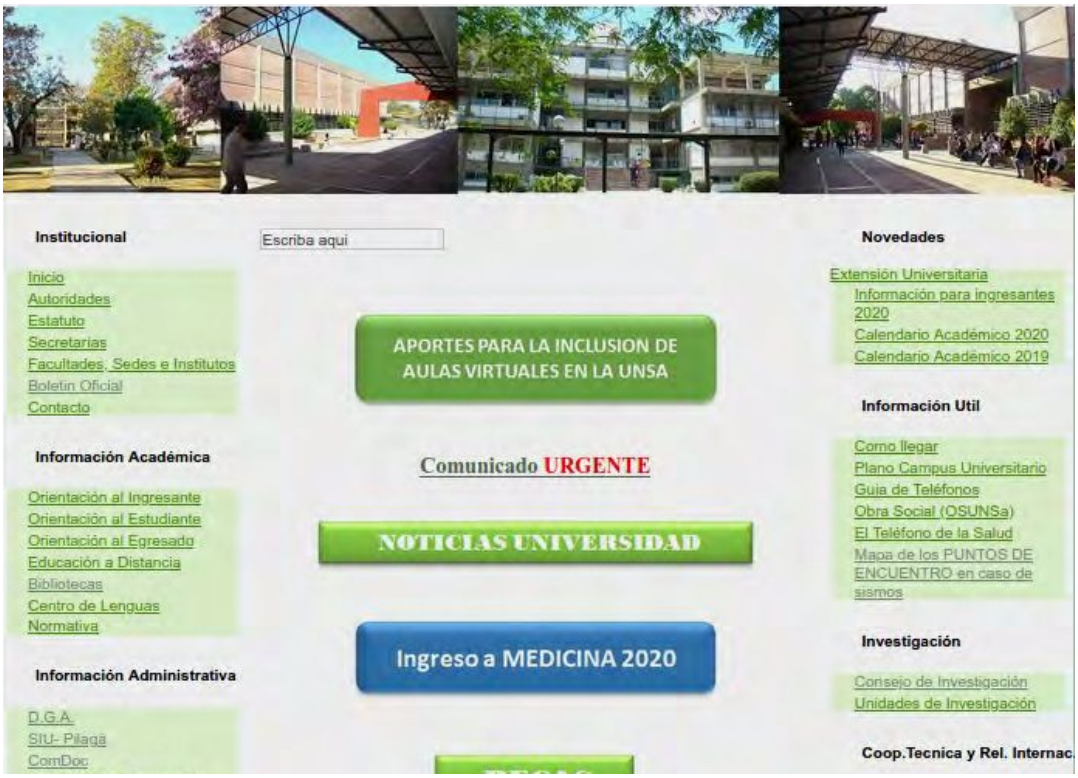


Figura 88: Página Web de la Universidad Nacional de Salta luego del Refactoring ***Add Anchor on Page****.*

Luego de aplicado el refactoring, en la ubicación previa a la lista de *Novedades* de la página Web, se ha insertado un enlace con el texto "*Extensión Universitaria*" que se vincula con la lista que contiene los enlaces de Secretaria de Extensión, Boletín Informativo, Radio Universidad y Agenda Centro Cultural Holver Martines Borrelli.

R17 - Date Input Into Selects

Este refactoring disponible en la versión de usabilidad de Kobold, se propone como alternativa de solución al **Accessibility Smell Datepicker inaccessible using the keyboard (**Datepicker inaccesible usando el teclado), cuya aplicación consiste en insertar en el formulario de la página tres listas de selección, una para cada uno de los datos que componen el valor de una fecha (día, mes y año).

Ante el **Accessibility Smell** se recomienda el refactoring **AddDatePickerAccessible** que no requiere aumentar la cantidad de elementos a recorrer y solo transforma el elemento existente manteniendo vinculados los manejadores de eventos.

8.4 Accessibility Events, Accessibility Smells, y Accessibility Refactorings

En la tabla 22 se representa la relación entre los **Accessibility Events, Accessibility Smells, y Accessibility Refactorings.** Cada events puede utilizarse para detectar varios smells y cada smels puede ser resuelto por varios Refactorings.

Accessibility Events	Accessibility Smells	Refactoring
E01- Auto Delete Entry Field	D01- Validation test without electronic text	R01- NotifyScreenReadersUsingAlertRole (S)
E02 - Electronic text for non-significant speech synthesis	D02- Input field without label that describes it	R02-_AddDesciptionToEntryConfrols_(S)
E03- Electronic text for non-existent speech synthesis	D03- Absence of descriptive text for text entry fields	R03- AddAriaLabel (S) R02-AddDesciptionToEntryConfrolsRefactoring (S) R04- AddPlaceholderText (S)
E04- High Frequency Of Use Of The Tab Key	D04- Distant content for keyboard use on a page	R12- SplitPage (N) R05- NewLayoutOfInterfaceElement (S)
E05- Flash Scrolling in Accesibilidad with NVDA add-on	D05- Overlooked Content For NVDA	R13- SplitPageForNVDA (N)
E06- Navigation Path Version Accessibility	D06- Distant Content in Accessibility	R06- AddLink (S)
E07- Unfilled Form	D07- Captcha In The Form	R11- ReplaceCaptcha (N)
	D08- Forms without send button	R07- AddSubmitButton (S)
	D14- Datepicker inaccessible using the keyboard	R10 – AddDatePickerAccessible (A) R17- Date Input Into Selects(A)
E08- Search result without electronic text	D09- Search without feedback	R08- AddFeedbackToSearch (S)
E09- Confused speech synthesis	D10 - Unexpected language change	R14 -AddLanguageFieldElement (A)
E10- Inappropriate Tab Sequence	D11- Disposition of confusing elements	R09- Modify Focus Sequence (S)
E11-FastScrollingWithKeyboard	R12 Overlooked Content	R12- SplitPage (N)
E12- Navigation Between Lists of Links for the NVDA add-on	D13-Distant Content For List Menu for Complement NVDA	R15-New Position For List On The Page (A) R16-Add Anchor on Page (S)

(S) Semi Automático – (A) Automático – (N) Sugerido.

*Table 22: Lista con las relaciones entre todos los **Accessibility Events, Accessibility Smells, y Accessibility Refactorings.***

Capítulo IX

Conclusiones

El enfoque del trabajo estuvo centrado en una visión de la accesibilidad vinculada con los obstáculos que experimentan los usuarios con dificultades visuales. La mayoría de los avances en la detección automática de incumpliendo de las “Web Content Accessibility Guidelines” están centrados en problemas estáticos. Las herramientas para detectar problemas en el código HTML o CCS, no se adaptan o resultan insuficientes para las dificultades de interacción sobre sitios web.

En ese marco, se propuso aplicar una versión adaptada a la metodología utilizada de Self Refactoring (Grigera, 2017) para detectar problemas de accesibilidad. El desarrollo incluye una extensión a la arquitectura de Kobold que detecta interacciones definida como **Accessibility Events** por medio de los cuales descubrir dificultades como **Accessibility Smells** y aplicarles soluciones como **Accessibility Refactorings**.

Durante la definición de los eventos de accesibilidad se presentaron problemas asociados con las acciones del usuario sobre los **Buffers Virtuales NVDA** que no resultan accesibles desde el componente javascript en la aplicación web.

Desde ese momento se distinguieron los eventos detectables con el snippet incrustando y los eventos del modo revisión de **NVDA**. Para detectar ambos fue necesario complementar la estrategia original de Kobold, analizando la arquitectura completa de **NVDA** e implementando una extensión que no requiriera modificaciones al núcleo.

Como resultado, el complemento **NVDA-Accessibility** agrupa interacciones del usuario en el **Buffer Virtual** y reportar eventos de accesibilidad a Kobold. Su integración combinada con el snippet incrustado, implementa acciones de marcado de elementos no focables que se remiten al componente servidor para ser utilizados en los Refactorings.

La necesidad de adecuar parámetros en los algoritmos de detección requirió modelar y desarrollar un sistema para estimación. A partir de la experimentación, se suministraron datos para mejorar la estimación de los valores vinculados a la detección de **Accesibilidad Events**.

A partir de mediciones con un snippet javascript y el complemento **NVDA-Accessibility-Parameters,** se obtuvieron resultados diferentes a los utilizados originalmente en Self Refactoring

probablemente como consecuencia de la combinación de dificultades visuales y el uso del lector de pantalla.

9.1 Aportes y Contribuciones

Un inventario de problemas y soluciones de accesibilidad orientadas a las modificaciones del software que incluye:

1) Una nueva versión de la herramienta Kobold, específica para accesibilidad, que ofrece la “Accesibilidad como Servicio” a partir del diagnóstico de smell con soluciones de refactoring del lado del cliente. Arquitectura descripta en el capítulo 3.

2) Un catálogo de estrategias para detectar **Accessibility Events** en aplicaciones web durante las acciones detectables mediante un snippet incrustado y las producidas sobre el **Buffer Virtual** de **NVDA.** Catálogo y ejemplos descripto en el Capítulo 4.

3) Una herramienta de software como extensión de **NVDA** para detectar **Accesibility Events** en el **Buffer Virtual** que permite reconocer acciones inaccesibles desde el snippet y reportarlas al componente Servidor.

4) Un catálogo de **Accessibility Smells** detectables automáticamente para usuarios con dificultades visuales que describe los problemas de accesibilidad reconocidos por la herramienta.

5) Un catálogo de **Accesibility Refactorings** como transformaciones de interfaz web, que pueden aplicarse a inconveniente catalogados como **Accessibility Smells**.

6) Un conjunto de estudios de casos, donde visualizar cada **Accessibility Events, Accesibility Smells** y **Accesibility Refactorings,** describiendo la dificultad, su forma de detección y una aplicación para su tratamiento.

7) Un sitio web que contiene un banco de pruebas para simular la detección de eventos de accesibilidad y scripts para implementar los **Accesibility Refactorings** mediante consolas de Javascript. Esto posibilita observar los comportamientos y los elementos de la interfaz donde se manifiestan las dificultades identificadas.

8) Un sistema para estimar parámetros de detección de **Accesibility Events.** Incluyendo un snippet con código Javascript**,** un plugins instalable que extiende **NVDA,** para acceder a acciones en el **Buffer Virtual** y una Api REST con autenticación de usuarios. En conjunto estos componentes procesan y reportan información necesaria para ajustar los estimadores involucrados en los procesos de detección.

9.2 Trabajos Futuros

Existen líneas de investigación que surgiendo durante la tesis, han quedado abiertas y son posibles de continuar a futuro. Algunas como resultado de la investigación y otras que exceden el alcance del trabajo y no fueron abarcadas con suficiente profundidad, dando lugar a las siguientes preguntas:

- - ¿Qué resultados se obtendrían aplicando la metodología a gran escala?

Cuando se implementa en aplicaciones de uso masivo, con miles de usuarios concurrentes, como el caso de Facebook, G suite o Moodle, podría estudiarse el comportamiento de la herramienta y evaluar la forma en que se detectan y reportaran las acciones para aplicar los refactorings.

- - ¿Qué indicadores de parámetros resultan más convenientes en la detección de los **Accessibility Smell**?

Considerando la medición de las acciones de los usuarios sobre base de información estadística, se podría buscar mejorar el intervalo de confianza y/o utilizar otro método para seleccionar los umbrales.

- - ¿De qué manera es posible incorporar el aprendizaje automático a la estimación de los parámetros?

Las soluciones de Machine Learning podrían utilizarse para dinamizar las detecciones y ajustarlas durante el uso a cada aplicación web, tipo de elemento y usuario específico.

- - ¿Qué patrones se presentan en los **Accesibility Events** y **Accessibility Refactoring** para modelarlos, automatizarlos y aplicarlos bajo demanda?

Self Refactoring (Grigera,2017) plantea la posibilidad de utilizar Lenguajes Específicos del Dominio (DSL) en la definición de los refactoring. Podemos agregar su posible aplicación a los **Accessibily Event** con un esquema de jerarquías extensibles que pueda compartirse entre aplicaciones.

- - ¿Cómo puede ajustarse en tiempo real la aplicación de los refactorings?

El análisis podría generar herramientas de evaluación permanente, sobre las páginas y los refactorings, requiriendo experimentos a gran escala que involucren aplicaciones de uso masivo.

- - ¿De qué manera se compromete la Seguridad Web al aplicar detecciones automáticas y soluciones por refactoring?

Un estudio de las limitaciones de análisis, por ejemplo en las aplicaciones bancarias, que restringen las solicitudes de request a servicios del dominio y donde la aplicación de refactoring vía snippet constituiría una seria infracción a las condiciones de seguridad. Podrían clasificarse los refactorings como seguros y específicos para cada dominio acorde al tipo de aplicación web.

- - ¿Cómo extender la metodología y su implementación independientemente del dispositivo de acceso y la tecnología de asistencia utilizada?

Este trabajo se limitó al screen reader **NVDA,** usando el teclado convencional y el navegador Web Mozilla Firefox. Una generalización a diferentes Web Browser, incluido el acceso por consola de texto, el uso de diferentes sistemas operativos, lectores de pantalla y dispositivos móviles podría requerir versionado y/o refactoring de scripts. Incorporar otras tecnologías de asistencia como reconocimiento de voz, dispositivos apuntadores alternativos, teclados alternativos y pantallas Braille, podrían requerir nuevos complementos más complejos y la necesidad de diseñar un método balanceado de carga de procesamiento entre el cliente y el servidor.

- - ¿Qué limitaciones se presentan en el caso de las aplicaciones enriquecidas?

Durante el estudio se presenta condiciones que dependen de la conexión y la velocidad de procesamiento local. Dificultades vinculadas a los servicios en la infraestructura que requieren ser analizadas y resueltas para todos los ambientes.

- - ¿Cómo podemos evaluar y aplicar refactoring basados en el aprendizaje automático?

Aplicaciones de Machine Learning podrían evaluar el uso, en tiempo real, de cada refactoring, estimando el más conveniente en cada contexto y momento, aplicado en forma alternativa, personalizada y adaptada repuesta a cada smells.

Bibliografía

Alonso López, F. (dir.). 2002.Libro Verde de la Accesibilidad en España. Diagnóstico y bases para un plan integral de supresión de barreras. Imserso. ISBN: 84-8446-048-7.

Alonso López, F. (dir.). 2003.Acceplan. Plan de accesibilidad 2003-2010. Libro Blanco.Ceapat.

Fernandes, Nádia & Kaklanis, Nikolaos & Votis, Konstantinos & Tzovaras, Dimitrios & Carriço, Luis. (2014). An analysis of personalized web accessibility. W4A 2014 - 11th Web for All Conference. 10.1145/2596695.2596698.

Garrido, A., Firmenich, S., Rossi, G., Grigera, J., Medina Medina N., Harari, I.: Personalized web accessibility using client-side refactoring. IEEE Internet Comput. To appear (2013)

Garrido, A., Rossi, G., Distante, D.: Refactoring for usability in web applications. IEEE Softw. 28(3), 60–67 (2011).

Garrido, A., Rossi, G., et al. (2013) 'Improving accessibility of Web interfaces: refactoring to the rescue', Universal Access in the Information Society, pp. 1–13. doi: 10.1007/s10209-013-0323-2.

Garrido, Alejandra & Rossi, Gustavo & Medina, Nuria & Grigera, Julián & Firmenich, Sergio. (2014). Improving Accessibility of Web Interfaces: Refactoring to the Rescue. Universal Access in the Information Society. 13. 387-399. 10.1007/s10209-013-0323-2.

Grigera, Julián & Garrido, Alejandra & Rivero, José & Rossi, Gustavo. (2016). Automatic Detection of Usability Smells in Web Applications. International Journal of Human-Computer Studies. 97. 10.1016/j.ijhcs.2016.09.009.

Grigera, Julián & Garrido, Alejandra & Rossi, Gustavo. (2017). Kobold: Web Usability as a Service. 10.1109/ASE.2017.8115717

Hassan Montero, Yusef; Martín Fernández, Francisco J. (2003). Qué es la Accesibilidad Web. En: No Solo Usabilidad, nº 2, 2003. <nosolousabilidad.com>. ISSN 1886-8592

Henry, Shawn Lawton. (2002). Understanding Web Accessibility. En Constructing Accessible Web Sites. Glasshaus: April 2002. ISBN: 1904151000.

Leporini B., Paternò F. (2003) Criteria for Usability of Accessible Web Sites. In: Carbonell N., Stephanidis C. (eds) Universal Access Theoretical Perspectives, Practice, and Experience. UI4ALL 2002. Lecture Notes in Computer Science, vol 2615. Springer, Berlin, Heidelberg

Leporini B., Paternò F., Scorcia A. (2006). Flexible tool support for accessibility evaluation. Interacting with Computers, Volume 18, Issue 5, 1 September 2006, Pages 869–890

Lopes, R., Gomes, D., & Carrico, L. (2010). Web not for all: A large scale study of web accessibility. In Proceedings of the 2010 International Cross Disciplinary Conference on Web

Masri, Firas & Luján-Mora, Sergio. (2010). Análisis de los métodos de evaluación de la accesibilidad web. Conferencia llevada a cabo en el 7º Congreso Internacional de Educación Superior, La Habana Cuba.

McQuillen, D., "Taking Usability Offline", Darwin Magazine , June 2003

Nielsen, J.: Designing Web Usability: The Practice of Simplicity. New Riders Publishing, Indianapolis (2000)

Rubin, J., Chisnell, D., 2008. Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests. Wiley, Hoboken, New Jersey, USA.

Santana Mansilla, Pablo & Lescano, Germán & Costaguta, Rosanna. (2015). Accesibilidad de aplicaciones móviles para discapacitados visuales: problemas y estrategias de solución.

Schiavone, Antonio Giovanni & Paternò, Fabio. (2015). An extensible environment for guideline-based accessibility evaluation of dynamic Web applications. Universal Access in the Information Society. 14. 111-132. 10.1007/s10209-014-0399-3.

Serrano Mascaraque, E, 2009. Herramientas para la evaluación de la accesibilidad Web. UCM Documentación de las Ciencias de la Información 2008, vol. 31.

TAW. Test de Accesibilidad Web. Disponible en: http://www.tawdis.net/. Fecha de consulta: 08/08/2012. W3C, 2008. Validador. http://validator.w3.org/errores/avisos

Catálogo de herramientas de accesibilidad W3C https://www.w3.org/WAI/ER/tools/index.html

W3C: Web Content Accessibility Guidelines (WCAG) 2.0, http://www.w3.org/TR/WCAG20 (2008). Accessed 10 July 2012.