Universidad de las Ciencias Informáticas

Facultad 4

Título: Desarrollo de una Librería de Componentes JavaScript basado en Dojo Toolkit.

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autor(es): Lilian Teresa Castro Mecías.
Rolando Bermúdez Peña.

Tutor: Ing. Jorge Ernesto Martínez Cabrera.
Consultante: Ing. Madelin Haro Pérez.

"Año 50 de la Revolución"

Ciudad de La Habana

Mayo 2008

“¿Para qué, sino para poner paz entre los hombres, han de ser los adelantos de la ciencia?”

José Martí

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los __15__ días del mes de _Mayo__ del año _2008____.

_____
Rolando Bermúdez Peña.
(Autor)

_____
Lilian Teresa Castro Mecías.
(Autor)

_____
Jorge Ernesto Martínez Cabrera
(Tutor)

DATOS DE CONTACTO

Ing. Jorge Ernesto Martínez Cabrera graduado de Ingeniero en Ciencias Informáticas de la Universidad de Ciencias Informáticas (UCI) en el año 2007, miembro del proyecto Humanización Penitenciaria (SIGEP) durante 3 cursos.

Ing. Madelin Haro Pérez. Graduada en el 2002 de la carrera Ingeniería Informática en el ISPJAE. Ha trabajado en la Empresa de Servicios Informáticos (ESI) de Cienfuegos y a partir de enero de 2003 en la UCI. Ha sido tutora de tesis durante los 5 cursos de la UCI y cotutor o consultor. Los temas trabajados han estado relacionados con el ciclo de vida del software (análisis, base de datos, arquitectura, prototipos, requisitos,…), Gestión de Proyecto, Seguridad de sistemas informáticos y Linux. Ha participado en eventos como Fórum de Ciencia y Técnica, UCIENCIA y Universidad 2008.
Opta actualmente por la categoría docente de asistente y es integrante en la maestría de Gestión de Proyecto. Se ha vinculado a los proyectos productivos en el rol de Jefe de Proyecto, Analista y Jefe de Capacitación. Ha impartido cursos de postgrado a funcionarios de empresas, profesores y trabajadores de la UCI y a los Directores de los IPI. Ha impartido cursos de capacitación en dos proyectos de producción en el extranjero. Ha recibido curso de postgrado en el instituto TATA en la India recibiendo el certificado internacional de habilidades. Es asesora del Departamento Docente Central de Práctica Profesional.

AGRADECIMIENTOS

A mis padres, por guiarme en la vida.

A mi familia por sentir orgullo de mí.

A mi otra familia de Las Villas por su cariño y atención.

A mi amiga Ani y su mamá.

A mis amigas y compañeras por ser parte de mi familia durante estos 5 años Yadira, Yaimí, Arturo, Tito, Carmen, Maylen, Yanet, Edicta, Meylin, Linet, Adriana, Yaima, Yoana, Ismaray, Yilian, Ana, Meylin, Félix, Yuni y todos los amigos con los que compartí en todo este tiempo.

A nuestro tutor.

A la UCI por hacerme parte de su tropa de futuro.

A mis amigos del proyecto Identidad por superar juntos la difícil etapa en Venezuela Yenin, Eylen, Iván, Irving, Alinita, Odeimis y su novio.

A todos los que se preocuparon.

Lyly

A mis Padres Ada y Roly, por quererme y apoyarme.

A mi hermano grande, porque eso es lo que es, a mi abuela Edith, Adita, Anays, Miguel Angel, Miguel y mi familia toda por pensar siempre en mí.

A mi novia Lyly por amarme y apoyarme tanto.

A Jorge Ernesto, nuestro tutor.

A mis amigos del Proyecto SIGEP los cuales aprecio mucho por el tiempo compartido.

A mi tía Marisol, Jorge y los muchachos, a Onelia y Guido.

A mi otra familia, Orestes, Orestico e Idelina, los quiero mucho.

A todos aquellos que siempre creyeron en mí y nunca olvidan.

Roly

DEDICATORIA

A mi mamá por estar siempre pendiente, siendo mis ojos en la oscuridad.

Mi papá por consentirme tanto y apoyar todas mis decisiones.

A mi hermano, el pintor de la familia.

A mis eternos amigos de la UCI.

A mi novio por brindarme su apoyo incondicional y su Amor, finalmente lo logramos.

Lyly

A mi mami chiquita querida, creo haber cumplido tu sueño de hacerme alguien en la vida.

A mi papá, por ser mi guía, por su apoyo y amor.

A mi hermano Frank por ser más que un hermano para mí, va a ser un gran hombre.

A mi Lyly, por su comprensión y amor infinitos.

A mi abuela Edith por quererme y pensar tanto en mí.

Roly

RESUMEN

Como parte del modelo de producción seguido en la Universidad de Ciencias Informáticas en el desarrollo de sistemas de software, el estudio y la utilización de nuevas tecnologías, surge la necesidad de construir una Librería de componentes JavaScript basado en Dojo Toolkit con el objetivo de satisfacer las necesidades de componentes que requieren diseñadores y programadores para desarrollar interfaces de aplicaciones web.

En el proceso de desarrollo se identificaron los proyectos productivos que utilizan la librería de Dojo Toolkit en la construcción de sus interfaces gráficas de usuario y por tanto la necesidad de componentes que presentan los mismos. La librería propuesta brinda la implementación de un componente que maneja dos listas de elementos, un Wizard, un Calendario de eventos y un Menú vertical con estructura de árbol que puedan ser utilizados en el desarrollo de aplicaciones web.

PALABRAS CLAVE

Librería, Componentes, JavaScript, Widget, Dojo Toolkit.

Índice de Contenidos

INTRODUCCIÓN

Es fundamental en el desarrollo de aplicaciones web la creación de interfaces con la usabilidad, interactividad y funcionalidad requerida por el usuario, exigencias de las cuales son responsables tanto el diseñador como el programador de interfaz de usuario. Existen herramientas como las Librerías de componentes JavaScript que facilitan el cumplimiento de tales propósitos, teniendo en cuenta que todo el tiempo se necesitan componentes que resuelvan determinados problemas. Las Librerías de componentes JavaScript tienen un rol determinante en la construcción de interfaces, tal es el caso de Dojo Toolkit, que brinda una gama de componentes para trabajar con elementos de formularios, tablas editables, menús contextuales, arboles de elementos, entre otros que son utilizados en un número significativo de aplicaciones.

En la Universidad de Ciencias Informáticas existen proyectos productivos que desarrollan aplicaciones web y utilizan la Librería de Dojo Toolkit en la construcción de sus interfaces de usuario. Los proyectos Humanización Penitenciaria, en su Sistema de Gestión Penitenciaria (SIGEP) de la Facultad #4 y el proyecto Centro de Tratamiento y Análisis de la Información de Seguridad Ciudadana (CTAISC), en su Sistema de Gestión Policial (SIGEPOL) de la Facultad #2 necesitan de nuevos componentes que la librería no brinda y que tanto los diseñadores como los programadores de interfaces requieren en el desarrollo de las mismas, surge entonces la necesidad de una Librería que brinde implementación para dichos componentes.

Entre los componentes identificados, para los cuales Dojo Toolkit no brinda implementación se encuentran los siguientes:

- Un calendario web, con manejo de eventos que pueden ser recurrentes.
- Dos listas de elementos que permitan mover y arrastrar elementos de una lista hacia la otra y viceversa.
- Un Menú desplegable vertical con estructura de árbol.
- Un Wizard que se integre con el servidor de forma dinámica.

Por lo que se puede plantear que el **problema de investigación** a resolver es:

Necesidad de componentes JavaScript basados en Dojo Toolkit que puedan ser usados por diseñadores y programadores en la construcción de interfaces web.

El **aporte práctico** de esta investigación es brindar la implementación de una Librería de componentes web que provea las funcionalidades antes mencionadas.

En este sentido se puede determinar que el **objeto de estudio** lo constituyen:
Los componentes para librerías Web.
Del Objeto de estudio analizado, se puede definir que **el Campo de Acción** lo constituyen
Los Componentes JavaScript basado en Dojo Toolkit para aplicaciones Web.

Se puede definir que **el Objetivo general** es:
Construir una librería de componentes JavaScript basado en Dojo Toolkit que pueda ser usada por diseñadores y programadores en la construcción de interfaces web.

A partir del cual se definen como **objetivos específicos** diseñar e implementar:

- Un Menú vertical con estructura de árbol.
- Un Componente que maneje dos listas de elementos
- Un Wizard.
- Un Calendario de eventos.

Para el cumplimiento de los objetivos específicos se desarrollaron un conjunto de tareas:

1. Investigar el Estado del Arte y elaborar el marco teórico referencial sobre el tema.
2. Buscar información sobre herramientas existentes que potencian el desarrollo de interfaces web.
3. Investigar los estándares que se usan actualmente para el desarrollo de componentes web.
4. Definir cada una de las funcionalidades que brindarían los componentes.
5. Documentar cada uno de los componentes con la finalidad de que tengan un uso provechoso para el desarrollador.
6. Implementar los widgets que conforman la librería

El presente documento consta de 3 capítulos:

Capítulo 1 Fundamentación teórica: En este capítulo se hace un estudio de carácter teórico que permita lograr un entendimiento de los aspectos más importantes del domino del problema, se hace un

análisis de algunas componentes y aplicaciones que manejan funcionalidades similares y se exponen las tecnologías y herramientas usadas en el desarrollo de la propuesta.

Capítulo 2 Descripción de la solución propuesta: En este capítulo se expone una descripción de las características y funcionalidades de los componentes que conforman la librería que permita conducir a la construcción de una solución que cumpla con los objetivos propuestos, se realiza un estudio de los patrones de diseño, se muestran además los Diagramas de UML necesarios, con la intención de guiar el trabajo en la implementación de la Librería.

Capítulo 3 Construcción de la solución propuesta: En este capítulo se describe cómo desarrollar componentes utilizando la librería de Dojo Toolkit, se especifican algunas de sus facilidades para el trabajo con JavaScript, se realizan las pruebas correspondientes a los componentes que integran la Librería con el objetivo de obtener un producto confiable en correspondencia con las funcionalidades antes previstas, también se genera la documentación con la intención de orientar al programador en su trabajo con la Librería.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En este capítulo se realiza un estudio de carácter teórico que permita lograr un entendimiento de los aspectos más importantes del dominio del problema, se hace un análisis de algunos componentes y aplicaciones que manejan funcionalidades similares y se exponen las tecnologías y herramientas usadas en el desarrollo de la propuesta.

1.2 Conceptos asociados al dominio del problema.

1.2.1 Librerías de software

Las bibliotecas o librerías de componentes de software son repositorios donde se guardan o conservan documentos, ideas, fragmentos de código y todos aquellos componentes (widgets) que participan en el desarrollo de un software, que puedan ser reutilizados, transformados o consultados. Se puede decir que tienen como objetivo asegurar la disponibilidad de activos, para apoyar el desarrollo de un producto de software.

Mantienen información relevante de cada componente tales como especificación técnica, clasificación, documentación, etc. Constituyen además una forma de acelerar los tiempos de programación, implementación, bajar costos y agregar funcionalidades utilizando soluciones ya probadas.

En todos los lenguajes de programación existen librerías de funciones que sirven para hacer diversas cosas a la hora de programar. Las librerías de los lenguajes de programación ahorran la tarea de escribir las funciones comunes que por lo general pueden necesitar los programadores. En ocasiones es más complicado conocer bien todas las librerías que aprender a programar en el lenguaje. (1)

1.2.2 Widget.

Un widget es un componente gráfico con el cual el usuario interactúa; usualmente presentado en archivos o ficheros pequeños, como por ejemplo, una ventana, una barra de tareas o una caja de texto.

Los widgets se combinan para construir las interfaces gráficas de usuario. El programador los adapta según sus necesidades sin tener que escribir más código que el necesario para definir los nuevos valores de las propiedades de los widgets. (2)

1.2.3 Toolkits

Existen herramientas que facilitan la creación de interfaces gráficas de usuario. Estas son conocidas como toolkits. Un Toolkit consiste en una biblioteca de utilidades, las cuales pueden también incorporar componentes de interfaz de usuario (IU) o widgets.

El uso de toolkits tiene grandes beneficios, como los que se mencionan a continuación: (3)

- Independencia de la IU: Separa la parte correspondiente al diseño de la IU de las complejidades de la programación. Esta separación le permite a los diseñadores el realizar prototipos rápida y fácilmente, hacer revisiones en minutos, y de esta manera agilizar los procedimientos de prueba. La programación necesaria para el sistema se empieza a escribir una vez que el diseño de la IU se ha estabilizado.
- Código más confiable: El código generado por los toolkits es frecuentemente más confiable que el código creado por un implementador humano, por lo que requiere una menor cantidad de debugging. Esto se debe a que el código generado automáticamente tiene un alto nivel de especificación.
- Elaboración rápida de prototipos: Con un toolkit las IUs pueden ser construidas y modificadas rápidamente. Gran parte del trabajo de diseño y construcción de un programa se debe a la creación de la IU. Al utilizar un toolkit para su elaboración, el tiempo de fabricación se reduce de una manera considerable. El tiempo ganado puede ser utilizado para hacer una mayor cantidad de pruebas de evaluación (probar, revisar, probar, revisar), y tener al final un producto apropiado para el usuario final.
- Interfaces de usuarios más fáciles de crear y de mantener: Esto es porque las especificaciones de la interfaz pueden ser representadas, validadas y evaluadas fácilmente gracias al empleo de los widgets incluidos en el toolkit, la cantidad de código a escribir es mucho menor.
- Facilidad para hacer modificaciones: El empleo de un toolkit permite que la incorporación de cambios a la interfaz después de haber hecho las pruebas correspondientes sea más sencilla.

1.2.4 Dojo Toolkit

Dojo es un toolkit open source DHTML (Dynamic HTML) escrito en JavaScript destinado a facilitar el rápido desarrollo de JavaScript o de las aplicaciones basadas en Ajax y sitios web. Fue iniciado por Alex Russell en el año 2004 y es doble licenciado bajo la licencia BSD [1]y la Licencia Libre Académica. Su idea es la de abstraer al desarrollador de las complejidades del DHTML y de las discrepancias existentes entre navegadores, que hacen que el código JavaScript a utilizar sea diferente. Dojo provee gran cantidad de funcionalidades las cuales divide en tres grandes proyectos, Dojo Core, Dijit y DojoX. Dijit y DojoX están basadas en la sólida base que Dojo Core brinda. Dijit incorpora un conjunto de widgets con los cuales se pueden crear amigables interfaces en muy poco tiempo, y escribiendo casi o ningún código JavaScript. DojoX es donde se desarrollan extensiones para Dojo Toolkit, sirve de cuna para nuevas ideas y pruebas de funcionalidades adicionales para el Toolkit principal. Estas son las principales funcionalidades que brinda Dojo Toolkit:

- **Clases de ayuda para la programación orientada a objetos**: Javascript es basado en prototipos, no basado en clases. Dojo esencialmente construye un sistema de clases adicionando grandes funcionalidades como la herencia, encapsulamiento y polimorfismo entre clases.
- **Módulos**: Al tiempo que el código del cliente crece, es difícil de mantener los nombres de variables globales, por eso Dojo brinda un sistema de módulos como parte de su sistema de clases, similares a los paquetes[2] en Java.
- **Métodos de Dijit**: Puedes crear componentes de Dijit programáticamente. También puedes crear tus propios widgets y extender las clases existentes en Dijit.
- **Eventos**: Dojo generaliza el sistema de eventos existente en Javascript. Puedes conectar eventos en código y la aplicación resultante funciona en Firefox, Internet Explorer y Safari sin modificación alguna.
- **XHR (Ajax)**: Dojo adiciona una buena fachada para los servicios nativos de XmlHttpRequest, con el uso de una sola función puedes pasar parámetros a una petición vía Ajax y obtener la respuesta en texto plano, en XML o en un objeto Javascript vía JSON[3].

[1] Berkeley Software Distribution

[2] Paquete: se refiere a la agrupación de código fuente usado en java, que pudiera estar compuesto por uno o varios componentes

[3] JSON: Javascript Object Notation, es un formato que representa a los objetos de javascript.

- **Drag and Drop:** Los servicios de Drag and Drop [4]son esenciales para una interacción fácil con el usuario. La capa de Drag and Drop que Dojo ofrece es rápida y funciona en múltiples navegadores.
- **Dojo.data**: El módulo de datos es una capa abstracta que hace el acceso a datos desde la base de datos o un servicio web de una forma consistente. Tú puedes crear tus propios módulos de datos desde tus propias fuentes.
- **Dojo.query:** Buscar y manipular nodos HTML suele ser una tarea difícil, pues con dojo.query puedes hacerlo tan fácil como lo es trabajar con selectores CSS.
- **i18n:** Los servicios de internacionalización hace todo el código globalmente accesible con fechas entendibles, formatos de números y monedas y recursos traducidos al idioma elegido.
- **Manejo del botón de atrás del navegador:** dojo.back salva la aplicación de usuarios nerviosos tratando de ir una página atrás. Aplicaciones de una sola página pueden perder datos al ir atrás por el navegador, dojo.back cambia el comportamiento del botón de atrás para evitar la pérdida de datos. (4)

1.3 Tendencias tecnológicas.

1.3.1 HTML

HTML es un lenguaje muy sencillo llamado HyperText Markup Lenguaje que como su nombre lo indica es un lenguaje de marcas pensado para la creación de hipertextos los cuales no son más que un texto al cual se le puedan incluir elementos multimedia es decir audio, video y gráficos, también acepta la presencia de hiperenlaces que permiten relacionar otras fuentes de información. Es un lenguaje de marcas ya que en él las instrucciones son trozos de texto resaltados convenientemente, que definirán la estructura lógica del documento. Es decir un documento HTML constará de texto que será el contenido, la información del documento y de instrucciones HTML que resaltarán este contenido y le darán un formato fácil de leer y con la posibilidad de relacionar documentos y fuentes de información mediante hipervínculos.

HTML es, sin duda, el primer paso para todo aquel que se interese por el diseño de páginas web, de ahí la importancia de este lenguaje que es absolutamente imprescindible. Una página web básicamente es una colección de contenedores de información situados en un documento, siguiendo las instrucciones marcadas mediante un lenguaje de etiquetas denominado HTML. Los documentos

[4] Arrastrar y soltar elementos de interfaz, muy conocido del inglés Drag and Drop

así escritos son interpretados por un programa llamado explorador o navegador que se encarga de presentar el documento al usuario final.

Los documentos creados con este lenguaje son estáticos es decir la información de la página que se presenta al usuario siempre es de la misma forma, siendo necesario que el diseñador modifique la página si se desea que tenga una apariencia diferente. Además de ser estáticas, las páginas web así escritas permiten muy poca interacción con los usuarios, básicamente los enlaces y los formularios. El panorama mejora mediante las hojas de estilo en cascada (CSS). (5)

1.3.2 CSS

Las Hojas de Estilo en Cascada o CSS(Cascading Style Sheets) surgen por la limitación del lenguaje HTML a la hora de darle forma a un documento, permitiendo dar solución a estos problemas ya que son mecanismos simples que posibilitan aplicar formato a los documentos escritos en HTML y en lenguajes estructurados como XML, dando la posibilidad de separar el contenido de la presentación, es decir, la información estará incluida en la página HTML pero sin ser este el archivo que defina como será visualizada la información, las indicaciones referentes a cómo quedará comprendido visualmente el documento estarán especificadas en los archivos CSS. Describe cómo se verá un documento en pantalla, siendo esta forma de descripción de estilos la vía que ofrece a desarrolladores el control total de estilos y formato de sus documentos así como de múltiples páginas web al mismo tiempo. Un cambio en el estilo para un elemento en el CSS implica que todas las páginas vinculadas a ese CSS en la que aparezca ese elemento serán afectadas. Funciona a través de reglas, o sea declaraciones sobre el estilo de uno o más elementos. Las hojas de estilo están compuestas por varias de esas reglas aplicadas a un documento HTML o XML. Es importante resaltar el cuidado que se debe tener a la hora de usar esta tecnología ya que muchos usuarios puede que no vean los formatos que son aplicados a las páginas con CSS. (6)

1.3.3 JavaScript

JavaScript es un lenguaje de programación que ha permitido el desarrollo de la Web, ha sido el avance más significativo en el logro de páginas Web dinámicas y exactas en cuanto a posición y presentación de su contenido, es un lenguaje robusto y a la vez ligero, el cual a pesar de ser considerado por muchos como un lenguaje no orientado a objetos permite implementar varias de las características de este paradigma de programación, basado en prototipos, las nuevas clases se generan clonando las clases bases y extendiendo su funcionalidad, cualquier navegador puede interpretar el código

JavaScript dentro de las páginas Web, permite la máxima interactividad entre el usuario y la página, además la verificación de los datos introducidos por el usuario antes de enviar el formulario al servidor, el manejo de applets y plugins[5] dentro de múltiples marcos de HTML.

Este lenguaje de programación tiene algunas limitaciones:

- JavaScript por definición no es un lenguaje orientado a objetos y debido a la potencia de esta programación se ha conseguido un funcionamiento similar, intuitivo y potente.
- No se precompila.
- No es obligatorio declarar las variables.
- Verifica las referencias en tiempo de ejecución.
- No tiene protección del código, ya que se descarga en texto claro.
- Es un lenguaje interpretado, o sea, que el sistema lo lee y traduce al mismo tiempo que lo va ejecutando, no confundir con el Java.

Con JavaScript se pueden lograr efectos realmente mágicos en una página, así como formularios, imágenes, textos, etc., tiene la ventaja de que la mayoría de los navegadores no tienen problemas para interpretarlo.

1.3.4 AJAX

Ajax es una nueva tecnología, nombrada por primera vez por Jesse James Garret, sus siglas son el acrónimo para Asynchronous JavaScript + XML, que en realidad no es una tecnología sino la combinación de muchas tecnologías trabajando en conjunto deviniendo en un resultado moderno más poderoso. (7)

AJAX incorpora:

- Presentación basada en estándares usando XHTML y CSS.
- Exhibición e interacción dinámicas usando el Document Object Model.
- Intercambio y manipulación de datos usando XML and XSLT.
- Recuperación de datos asincrónica usando XMLHTTPRequest.
- JavaScript poniendo todo junto.

[5] Plugin: es una aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica y es ejecutada por la aplicación principal.

Una aplicación AJAX elimina la naturaleza “arrancar-frenar- arrancar-frenar” de la interacción en la Web introduciendo un intermediario -un motor AJAX- entre el usuario y el servidor. Parecería que sumar una capa a la aplicación la haría menos reactiva, pero la verdad es lo contrario.

En vez de cargar un página Web, al inicio de la sesión, el navegador carga al motor AJAX (escrito en JavaScript y usualmente “sacado” en un frame oculto). Este motor es el responsable por renderizar la interfaz que el usuario ve y por comunicarse con el servidor en nombre del usuario. El motor AJAX permite que la interacción del usuario con la aplicación suceda asincrónicamente (independientemente de la comunicación con el servidor). Así el usuario nunca estará mirando una ventana en blanco del navegador y un icono de reloj de arena esperando a que el servidor haga algo.

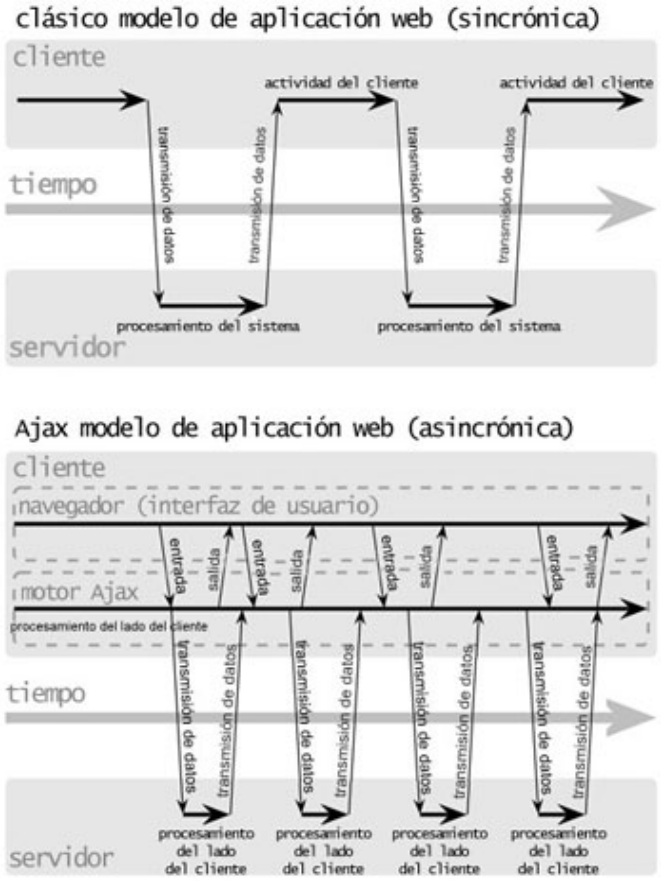


Figura 1. El patrón de interacción sincrónica de una aplicación Web tradicional (arriba) comparada con el patrón asincrónico de una aplicación AJAX (abajo).

Cada acción de un usuario que normalmente generaría un requerimiento HTTP toma la forma de un llamado JavaScript al motor AJAX en vez de ese requerimiento. Cualquier respuesta a una acción del usuario que no requiera un viaje de vuelta al servidor (como una simple validación de datos, edición de datos en memoria, incluso algo de navegación) es manejada por su cuenta.
Si el motor necesita algo del servidor para responder (sea enviando datos para procesar, cargar código adicional, o recuperando nuevos datos) hace esos pedidos asincrónicamente, usualmente usando XML, sin frenar la interacción del usuario con la aplicación.
El modelo Ajax provee un nivel intermedio llamado motor (**engine**) que no es más que un objeto o función de JavaScript, que será llamado cada vez que se necesite una respuesta del servidor.

1.3.4.1 Ventajas de Ajax

Como principales ventajas de Ajax es importante que se mencione:

- Tráfico mínimo: Las aplicaciones Ajax deben enviar y recibir una pequeña cantidad de información desde y para el servidor, por lo que se minimiza el tráfico entre el cliente y el servidor, asegurándose que no reciba o envíe información innecesaria.
- Convenciones establecidas: No gasta tiempo inventando nuevos modelos de interacción de usuario con los que no estarán además familiarizada.
- El usuario primero: diseñar las aplicaciones con el usuario en mente antes que nada.
- Multi-Browser y Multi-Plataforma.

Aplicaciones que utilizan Ajax

Tradicionalmente se ha considerado la primera aplicación Ajax al cliente Web que tiene la herramienta de trabajo en grupo Microsoft Exchange Server aunque sin lugar a dudas Google es uno de los grandes responsables de la popularización de Ajax, al usarla en varias de sus aplicaciones, entre las que se cuentan Google Groups, Google Suggests, Google Maps y el servicio de correo electrónico gratuito Gmail. Así como también empresas en crecimiento que actualmente están desarrollando aplicaciones basadas en Ajax.

- **Gmail:** Correo electrónico de la empresa de Google. Utiliza tecnología AJAX, siendo pioneros en emplearla, aunque también dispone de una interfaz basada en HTML+CSS útil para navegadores antiguos o no compatibles.
- **Google Groups**: Es una herramienta donde se puede crear listas de correo y grupos de discusión.

- **Google Suggests**: Servicio que ofrece diversas sugerencias de términos de búsqueda, es posible gracias a los algoritmos que posee el buscador para predecir las búsquedas de los usuarios.
- **Google Maps**: Es un servicio que ofrece mapas de ciudades de diversos países. Muestra fotografías aéreas de todo el planeta de mayor o menor resolución dependiendo si se trata o no de importantes núcleos urbanos.
- **Orkut:** Es la Red Social de Google diseñada para permitir a sus integrantes mantener sus relaciones existentes y hacer nuevos amigos, contactos comerciales, etc.
- **Amazon (A9)**: Motor de búsqueda de Internet de la empresa Amazon.com.
- **Flickr**: Servicio para crear álbumes de fotos online.

1.3.5 Eclipse SDK

Metafóricamente, Eclipse es como una tienda para herreros, donde no solamente se hacen productos, sino que además se hacen las herramientas para hacer los productos. Cuando se descarga el Eclipse SDK, se obtiene un equipo de instrumentos para desarrollo en Java o Java Development Toolkit (JDT) para escribir y depurar programas en Java; además se obtiene un ambiente de desarrollo de plugin Plug-in Development Enviroment para heredar de Eclipse. Si todo lo que se quiere es un IDE para Java, no se necesita nada además que el JDT. Esto es para lo que la mayoría las personas usan Eclipse. (8)

Aunque Eclipse es escrito en Java y su principal uso es como IDE[6] para Java, este es un lenguaje neutral. El soporte para desarrollo en Java es proveído por un componente enchufado o plugin, pero además están disponibles plugins para otros lenguajes, como C/C++, Cobol, C#.

En principio permite ejecutar un programa sobre cualquier plataforma. Es una extensible plataforma de código abierto (open source) para desarrollar herramientas. Es administrado y dirigido por un consorcio de compañías de desarrollo de software con un interés comercial en promover Eclipse como plataforma compartida para herramientas de desarrollo de software.

[6] IDE: Entorno de desarrollo integrado, pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica de usuario.

1.3.6 UML como lenguaje de modelación visual

El Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, *Unified Modeling Language*) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad.

Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables. (9)

1.3.7 Rational Rose como herramienta de modelado

Rational Rose es una herramienta de modelado visual que permite el análisis y diseño de sistemas de software orientados a objetos. Es usado para modelar un sistema antes de escribir el código, de modo que usando el modelo se pueden capturar errores de diseño tempranamente.

1.4 Análisis de otras soluciones existentes

Dada la tendencia de implementar componentes de software reutilizables, útiles en el desarrollo de aplicaciones web, se realiza un estudio de algunas aplicaciones que brindan la implementación de componentes web con funcionalidades similares al Calendario, el Wizard, el componente que maneja las dos listas de elementos y el Menú vertical con estructura de árbol que sirvan de referencia en el desarrollo de la solución propuesta.

1.4.1 Soluciones con funcionalidades asociadas al Calendario de eventos.

1.4.1.2 Calendario para Dreamweaver

TSCalendar es un componente desarrollado para ser usado en Dreamweaver MX y versiones posteriores.

Está concebido para ser usado en páginas web que manejen fechas**.** El calendario permite la definición de días restringidos, los números de estos días aparecerán en caracteres tachados cuando se aceda al mes en el cual ellos están y no serán accesibles a los usuarios del sitio Web. Esta opción es muy útil cuando se trata de actividades o servicios que no son usualmente ofrecidas en determinados días.

Permite seleccionar el lenguaje en el cual este aparecerá (el lenguaje de los meses y días de la semana pueden ser seleccionados entre el español, inglés, alemán y japonés). TSCalendar permite la definición de días especiales para destacarlos cuando se acceda al mes.

TSCalendar aunque brinda una variedad extensa de funcionalidades no maneja eventos recurrentes y su distribución no es gratis. (10)

1.4.1.3 Google Calendar

Google Calendar permite crear calendarios online, los cuales pueden ser compartidos a un número de usuarios simultáneamente. Es fácil de usar, e incorpora una interfaz simple y robusta (realizada en Ajax) la cual permite una fácil introducción de datos, tarea que se puede realizar también a través de importación de datos externos, provenientes de aplicaciones externas como 'Microsoft Outlook Calendar' o "Yahoo! Calendar", o incluso desde agendas de otros usuarios de 'Google Calendar', puesto que se utiliza un estándar de transmisión de datos basado en XML o ICAL.

Fácil para trabajar con eventos recurrentes con el mínimo de esfuerzo. Múltiples calendarios se pueden crear y ver en una misma vista posibilitando crear, gestionar y compartir eventos. (11)

- Aunque proveen un gadget[7] que puedes utilizar si deseas incluir este calendario en tu propia página no forma parte de ninguna librería open source por lo tanto tiene sus limitaciones en cuanto a reutilización y personalización del calendario.
- Google Calendar aunque es gratis no podemos desarrollar sobre él y estudiar su código fuente en nuestras aplicaciones.
- Podemos usarlo, pero siempre utilizando su servidor web donde reside nuestro calendario, no podemos usarlo para que sirva de interfaz a un calendario residente en nuestra base de datos.

1.4.2 Soluciones con funcionalidades asociadas al Wizard.

1.4.2.1 Wizard que brinda Dojo Toolkit 1.1

Dojo brinda la implementación de un Wizard en el módulo de extensión de Toolkit llamado dojox, se encuentra en el paquete dojox.widget.wizard. Es un Wizard en JavaScript el cual básicamente brinda las siguientes funcionalidades.

[7] Gadget: Un gadget de Google es como un widget o componente web, este puede ser usado desde las aplicaciones web para trabajar directo con calendarios desde tu página.

- Provee un flujo de páginas secuencial hacia delante y hacia atrás entre estas, posibilidad de terminar el Wizard cuando sea el momento y de cancelarlo cuando el usuario desee.
- Permite customización de los botones de control, atrás, siguiente, terminar y cancelar.
- Cada panel que compone el Wizard puede cargar contenido dinámico desde el servidor.
- Permite validar una página antes de salir de esta.

Provee una manera sencilla de implementación de un Wizard dando al programador la posibilidad de extender de este para crear uno más sofisticado si así lo desea.

1.4.3 Acerca del Menú y el componente que maneja las dos listas de elementos.

Existen varias Librerías JavaScript entre las que están Ext, YUI y la propia Dojo Toolkit que ofrecen la implementación de componentes de tipo Menú, cada una brindando menús con diversas formas, por ejemplo Menús contextuales, botones que despliegan menús y menús horizontales que brindan diferentes opciones. (Ver Anexos 5 y 6)

En cuanto al componente que maneja las dos listas de elementos; en la web hay varios componentes con funcionalidades similares pero son publicadas por usuarios anónimos y en sitios o blogs editables en los cuales no se responsabilizan con la información publicada y los cuales no poseen una interfaz agradable para el usuario.

Conclusiones

En este capítulo se presentó un esbozo de los principales conceptos que permiten comprender el dominio del problema. Se realizó un resumen de las tendencias tecnológicas utilizadas. Se describieron las características principales de Dojo Toolkit y se expuso un estudio acerca de diferentes aplicaciones que manejan funcionalidades similares, se explica la importancia que representa el desarrollo de una Librería de componentes basada en Dojo Toolkit en cuanto a reutilización no solo en los proyectos productivos mencionados sino cualquier otro proyecto o programador que decida utilizar Dojo Toolkit en sus aplicaciones.

CAPÍTULO 2: DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA.

2.1 Introducción

En este capítulo se exponen las funcionalidades y características principales que brindan cada uno de los componentes que conforman la Librería, no se especifican mediante requisitos funcionales debido a que dichos componentes son utilizados por el programador para satisfacer los requerimientos que tenga el sistema en el cual se utilice.

Se muestran los Diagramas de clases correspondientes a cada uno de los componentes y los Diagramas de Interacción por cada una de las funcionalidades.

2.2 Principales características y funcionalidades

La librería incluye cuatro componentes y todos necesitan manejar datos, por ejemplo el Menú necesita la información para crear cada uno de los menús hijos, el Calendario requiere gran cantidad de información proveniente del servidor para manejar los eventos, el Wizard a su vez necesita los datos a mostrar en cada página y las dos listas de elementos los datos que se muestran en cada una de ellas, provenientes generalmente del servidor, es por ello que en cada uno de los componentes se usa el api de dojo.data, el cual provee una manera de obtener los datos del servidor y manejar estos mediante una interfaz común.

2.2.1 Descripción del componente que maneja dos listas de elementos.

El componente que maneja dos listas de elementos debe permitir mover y arrastrar uno, varios o todos los elementos de una lista hacia la otra y viceversa, también debe permitir revertir todos los cambios efectuados en ambas listas, volviendo al estado inicial.

Funcionalidades principales:

1. Permitir cargar los datos de las listas desde el servidor usando el api de dojo.data mediante un store[8] del tipo dojo.data.ItemFileWriteStore. Este store es una implementación del api dojo.data el cual permite además de leer los datos, hacer cambios sobre los mismos.
2. Permitir mover todos los elementos o solo los que han sido seleccionados de una lista hacia otra, mediante el uso de los botones de control, los cambios se efectúan sobre el store también.
3. Permitir arrastrar y soltar elementos seleccionados desde una lista hacia la otra, reflejando los cambios efectuados en el store utilizado.
4. Permitir guardar los cambios efectuados sobre las listas en el store, los datos pueden ser guardados en el servidor, es función del programador implementar dónde se guardarán los cambios.
5. Permitir revertir los cambios efectuados sobre las listas que no han sido guardados aún.

2.2.2 Descripción de las características del Wizard

Permite manejar un conjunto de páginas de forma secuencial o dinámica en dependencia de las acciones del usuario, brinda validación y diferentes opciones de navegación entre las páginas.

Funcionalidades principales:

1. Permitir cargar los datos que se mostrarán en cada una de las páginas mediante el uso del api de dojo.data, usando un store de tipo dojo.data.ItemFileWriteStore. El programador es el encargado de definir cuales datos mostrar en cada página.
2. Incorporar dos modos para el Wizard, “local” o “remota”, el primero para trabajar con datos que residen en el cliente en objetos JavaScript y en el segundo los datos se cargan y manejan desde el servidor.

[8] store: Almacenador de datos, provisto de una interfaz para acceder a los mismos.

3. Proveer flujos entre páginas de forma secuencial siguiendo el modo tradicional siguiente, siguiente, finalizar o de forma dinámica, tomando diferentes caminos en dependencia de las acciones del usuario.

4. Permitir flexibilidad, posibilitando ir a la página anterior o a la siguiente con datos vàlidos o no.

2.2.3 Descripción de las características del Menú

El componente brinda un Menú vertical estructurado con estructura de árbol, por lo que puede tener el nivel de profundidad de menús hijos que se desee en dependencia de la necesidad del usuario.

Funcionalidades principales:

1. Permitir una estructura de árbol, posibilitando tener múltiples menús hijos.
2. Mostrar un menú a la vez, viñetas a los menús de acción, y viñetas de estado (abierto o cerrado) a los menús contenedores.
3. Permitir mostrar texto en cada menú con la longitud que se desee, el componente permite acortar el texto a una cantidad de caracteres deseado y hace visible el texto completo al pasar el mouse por encima del menú.

2.2.4 Descripción de las características del Calendario

En la actualidad existen aplicaciones como Microsoft Outlook, Apple iCal y Google Calendar que implementan Calendarios mediante los cuales el usuario puede programar citas, reuniones y planificar su actividad diaria, también permiten compartir y exportar información sobre sus calendarios, eventos, citas y tareas. La comunicación entre calendarios es posible debido a que implementan el estándar de calendarios para internet RFC 2445 conocido como iCalendar[9], el cual permite el intercambio de información referente a calendarios y eventos, especifica los métodos y propiedades de un calendario, y propone una forma estándar de manejar las interacciones entre múltiples Calendarios.

[9] iCalendar: Estándar de Calendarios para Internet, define un formato para compartir calendarios mediante internet y una especificación de los objetos de un calendario.

Funcionalidades principales:

1. Permitir crear múltiples calendarios usando el mismo componente.
2. Permitir crear eventos sobre calendarios existentes.
   2.1 Permitir especificar nombre, lugar y descripción del evento.
   2.2 Permitir indicar un patrón de recurrencia para dicho evento.
3. Permitir modificar y eliminar calendarios.
4. Brinda la opción de mostrarse en los idiomas, inglés o español.
5. Permitir mostrar diferentes vistas tales como: vista de Días, Semana, Mes, Crear nuevo evento, Crear nuevo calendario.
6. Permitir manejar series de eventos como uno solo.
7. Permitir especificar alarmas a los eventos deseados.
8. Permitir guardar los cambios realizados sobre el calendario en el servidor.
9. Exportar e importar eventos en formato iCal.

2.3 Referente al Diseño

2.3.1 Diagramas de Interacción.

Los diagramas de secuencia y de colaboración llamados diagramas de interacción se utilizan para modelar los aspectos dinámicos de un sistema.

- Muestran cómo se comunican los objetos.
- Su interacción para realizar colectivamente los servicios ofrecidos por las aplicaciones.
- Su misión, en esencia es localizar el comportamiento de los objetos.

Se utilizarán los Diagramas de secuencia por estar bien adaptados para representar interacciones entre los objetos, es decir mostrar gráficamente cómo los objetos se comunicarán entre ellos a fin de cumplir con las funcionalidades previstas.

2.3.2 Diagrama de clases

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargan del funcionamiento y la relación entre uno y otro. (12)

2.3.3 Para qué utilizar Diagramas de clases del diseño.

Se muestra una representación de Diagramas de clases del diseño con el objetivo de describir gráficamente las especificaciones de las clases de cada uno de los componentes y sus interfaces.
Aunque el lenguaje de UML no define concretamente un elemento denominado "diagrama clases del diseño", sino que utiliza un término más genérico: "diagrama de clases". Se incluye el término "diseño" en "diagrama de clases" para especificar que se trata de una perspectiva desde el punto de vista del diseño y no de una concepción analítica sobre los conceptos del dominio. Se logra además un punto de partida para comenzar la implementación.

Modelan los aspectos estáticos de un sistema.

1. Formalizar el análisis de conceptos.
2. Definir una solución de diseño.
3. Construir componentes de software.

2.4 Patrones de Diseño

Un patrón es una pareja de problema / solución con un nombre, que codifica (estandariza) buenos principios y sugerencias. Es aplicable a otros contextos, con una sugerencia sobre la manera de usarlo en diferentes situaciones. El objetivo de los patrones es crear un lenguaje común a una comunidad de desarrolladores para comunicar experiencia sobre los problemas y sus soluciones. Pueden referirse a distintos niveles de abstracción, desde un proceso de desarrollo hasta la utilización eficiente de un lenguaje de programación. (13)

Para el diseño se utilizaron los patrones de asignación de responsabilidades (GRASP, Patrones de Software para la asignación General de Responsabilidad, en english Acrónimo de General Responsability Assignment Software Patterns.), en especial el patrón Experto que plantea que se debe

asignar una responsabilidad al experto en información -la clase que tiene la *información* necesaria para realizar la responsabilidad. (UML y patrones)

2.5 Descripción del diseño del componente que maneja dos listas de elementos.

2.5.1 Representación de los Diagramas de diseño.

Diagrama de clases del diseño componente que maneja dos listas de elementos.

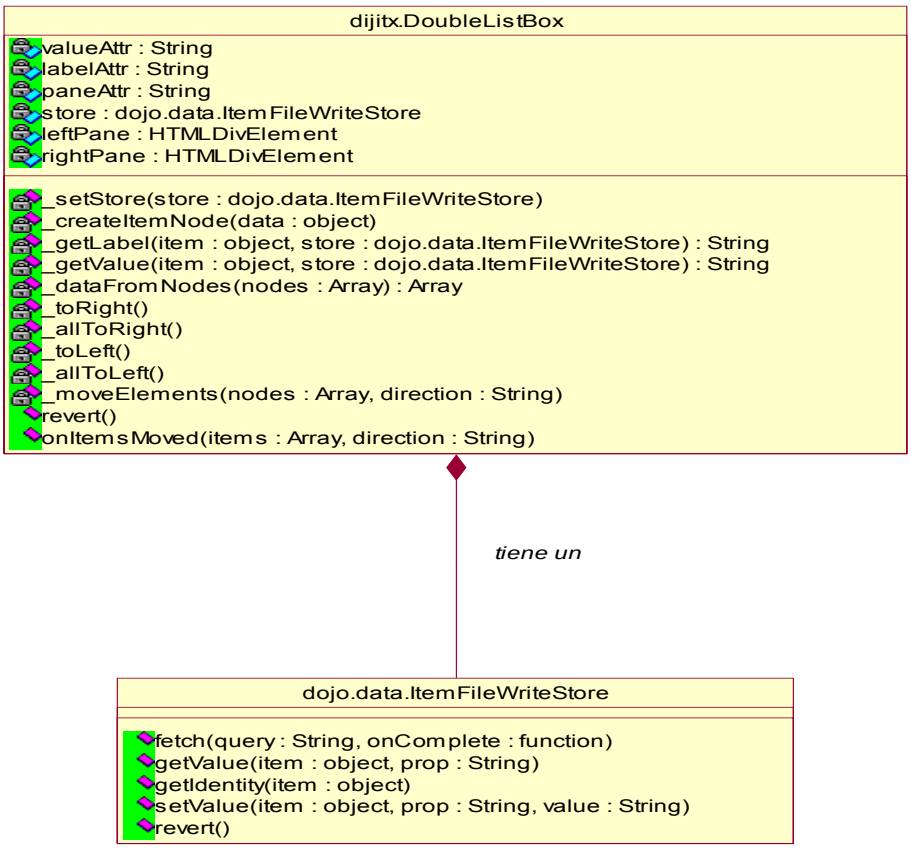


Tabla 1 Componente que maneja dos listas de elementos.

2.5.2 Descripción de las clases del Diseño del componente que maneja dos listas de elementos.

Nombre: dijitx.DoubleListBox	
Descripción de la clase: Representa al componente encargado de manejar dos listas de elementos, muestra la interfaz necesaria que permite al usuario la interacción con el componente y utiliza un dojo.data.ItemFileWriteStore como proveedor de datos.	
Atributo:	**Tipo:**
valueAttr	String
labelAttr	String
paneAttr	String
leftPane	HTMLDivElement
rigthPane	HTMLDivElement
Store	dojo.data.ItemFileWriteStore
Para cada responsabilidad	
Nombre	_setStore
Descripción	Dados los datos básicos del elemento crea el nodo correspondiente que se mostrará en el panel.
Nombre	_getValue
Descripción	Devuelve el identificador del elemento.
Nombre	_dataFromNodes
Descripción	Array con los nodos HTML que representan los elementos que se muestran en ambas listas, derecha e izquierda.
_**toRight**	Se ejecuta cuando se da clic sobre el botón que permite mover los elementos hacia la derecha.
Nombre	_allToRight
Descripción	Se ejecuta cuando se da clic sobre el botón que permite mover todos los elementos hacia la derecha.
Nombre	_toLeft
Descripción	Se ejecuta cuando se da clic sobre el botón que

	permite mover los elementos hacia la izquierda.
Nombre	_allToLeft
Descripción	Se ejecuta cuando se da clic sobre el botón que permite mover todos los elementos hacia la izquierda.
Nombre	_moveElements
Descripción	Permite mover los datos de esos elementos hacia la dirección definida.
Nombre	Revert
Descripción	Revierte todos los cambios que se han realizado, todos los movimientos efectuados y retorna al componente al estado inicial.

Nombre: dojo.data.ItemFileWriteStore	
Descripción de la clase: Es una clase que pertenece al api de dojo.data y sirve como contenedor de datos, permite leer y salvar los datos, ya sea desde el servidor o localmente.	
Para cada responsabilidad	
Nombre	Fetch
Descripción	Se utiliza para realizar consultas a los datos guardados en el store.
Nombre	getValue
Descripción	Retorna el valor de una propiedad de un elemento determinado.
Nombre	getIdentity
Descripción	Retorna el identificador de un elemento dado.
Nombre	setValue
Descripción	Asigna un valor a una propiedad de un elemento determinado.
Nombre	Revert
Descripción	Devuelve el store a su estado inicial.

2.5.3 Representación del Diagrama de secuencia del componente que maneja dos listas de elementos.

Diagrama de secuencia del componente que maneja dos listas de elementos.

Tabla 2 Diagrama de secuencia del componente que maneja dos listas de elementos.

2.6 Descripción del diseño del Wizard.

2.6.1 Representación de los Diagramas de Diseño.

Diagrama de clases del diseño del Wizard.

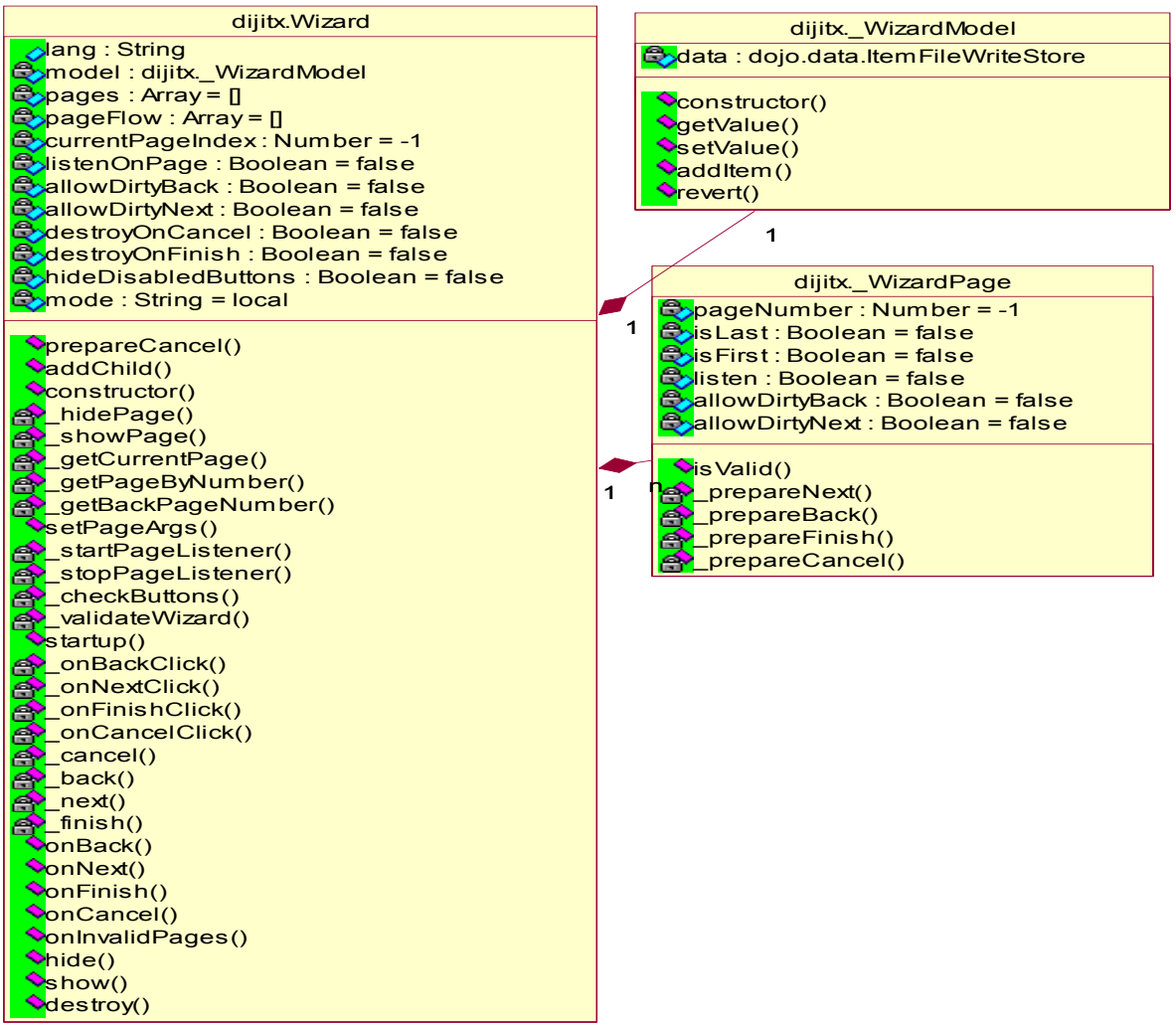


Tabla 3 Diagrama de clases del diseño del Wizard

2.6.2 Descripción de las clases del diseño del Wizard.

Nombre: dijitx._WizardPage	
Descripción de la clase: Representa una página de un Wizard y provee configuraciones específicas para cada una de las páginas que componen el Wizard.	
Atributo:	**Tipo:**
pageNumber	Number
isLast	Boolean
isFirst	Boolean
Listen	Boolean
allowDirtyBack	Boolean
allowDirtyNext	Boolean
Para cada responsabilidad	
Nombre	_prepareNext
Descripción	Se ejecuta antes de ir hacia la página siguiente, devuelve el número de página a la que se pasará.
Nombre	_prepareBack
Descripción	Se ejecuta antes de ir hacia la página anterior, no necesita ser devuelto el número de página, pues ya este se conoce.
Nombre	_prepareFinish
Descripción	Se ejecuta antes de finalizar el Wizard, se supone que esta página sea una página que finaliza el Wizard.
Nombre	_prepareCancel
Descripción	Se ejecuta al cancelar el Wizard, si retorna true se cancela el Wizard, si retorna false no se cancela el Wizard.
Nombre	isValid

Descripción	Devuelve verdadero si la página contiene todos los datos válidos, falso de lo contrario.

Nombre: dijitx._WizardModel	
Descripción de la clase: Clase para soporte del modelo de datos de un Wizard, permite cargar los datos desde el servidor o localmente mediante el uso del api de datos de Dojo.	
Atributo:	**Tipo:**
Data	dojo.data.ItemFileWriteStore
Para cada responsabilidad:	
Nombre	getValue
Descripción	Dado una clave se devuelve el valor asociado guardado en el store.
Nombre	setValue
Descripción	Dado una clave y un valor, se guarda el valor asociado a la clave seleccionada en el store, la clave debe existir.
Nombre	addItem
Descripción	Dado una clave y un valor, se adiciona dicho valor asociado a la clave seleccionada en el store.
Nombre	Revert
Descripción	Hace un revert en el store, los datos que no han sido guardados aún se desechan.

Nombre: dijitx.Wizard	
Descripción de la clase: Clase que maneja un conjunto de páginas como un Wizard, provee navegación entre páginas de forma dinámica se integra también al servidor.	
Atributo:	**Tipo:**
Lang	String
Model	dijitx._WizardModel
Pages	Array
pageFlow	Array

hideDisabledButtons	Boolean
Mode	Boolean
Para cada responsabilidad:	
Nombre	_prepareCancel
Descripción	Se ejecuta al cancelar el Wizard, si retorna true se cancela el Wizard, si retorna false no se cancela el Wizard.
Nombre	addChild
Descripción	Adiciona una nueva página al Wizard, se debe adicionar páginas vía javaScript antes de inicializar el Wizard.
Nombre	Constructor
Descripción	Método donde se asigna el valor de algunas propiedades por defecto.
Nombre	_checkButtons
Descripción	Chequea la validez de la página actual y en consecuencia habilita o deshabilita los respectivos botones.
Nombre	Startup
Descripción	Inicia al Wizard, selecciona la primera página, asigna números de página e inicializa el componente.
Nombre	_onBackClick
Descripción	Método que se ejecuta al dar clic sobre el botón que va hacia la página anterior.
Nombre	_onNextClick
Descripción	Método que se ejecuta al dar clic sobre el botón que va hacia la página siguiente.
Nombre	_onFinishClick
Descripción	Método que se ejecuta al dar clic sobre el botón que termina el Wizard.
Nombre	_onCancelClick
Descripción	Método que se ejecuta al dar clic sobre el botón

	que cancela el Wizard.
Nombre	onBack
Descripción	Se ejecuta al dar clic en el botón que va hacia la página anterior, se pasa como parámetros el número de página actual y el número de página a la que se navega.
Nombre	onNext
Descripción	Se ejecuta al dar clic en el botón que va hacia la página siguiente, se pasa como parámetros el número de página actual y el número de página a la que se navega.
Nombre	onFinish
Descripción	Se ejecuta al dar clic en el botón que termina el Wizard, se pasa como parámetro el número de página actual.
Nombre	onCancel
Descripción	Se ejecuta al dar clic en el botón que cancela el Wizard, se pasa como parámetro el número de página actual.

2.6.3 Representación del Diagrama de Secuencias del Wizard.

Diagrama de secuencia del Wizard.

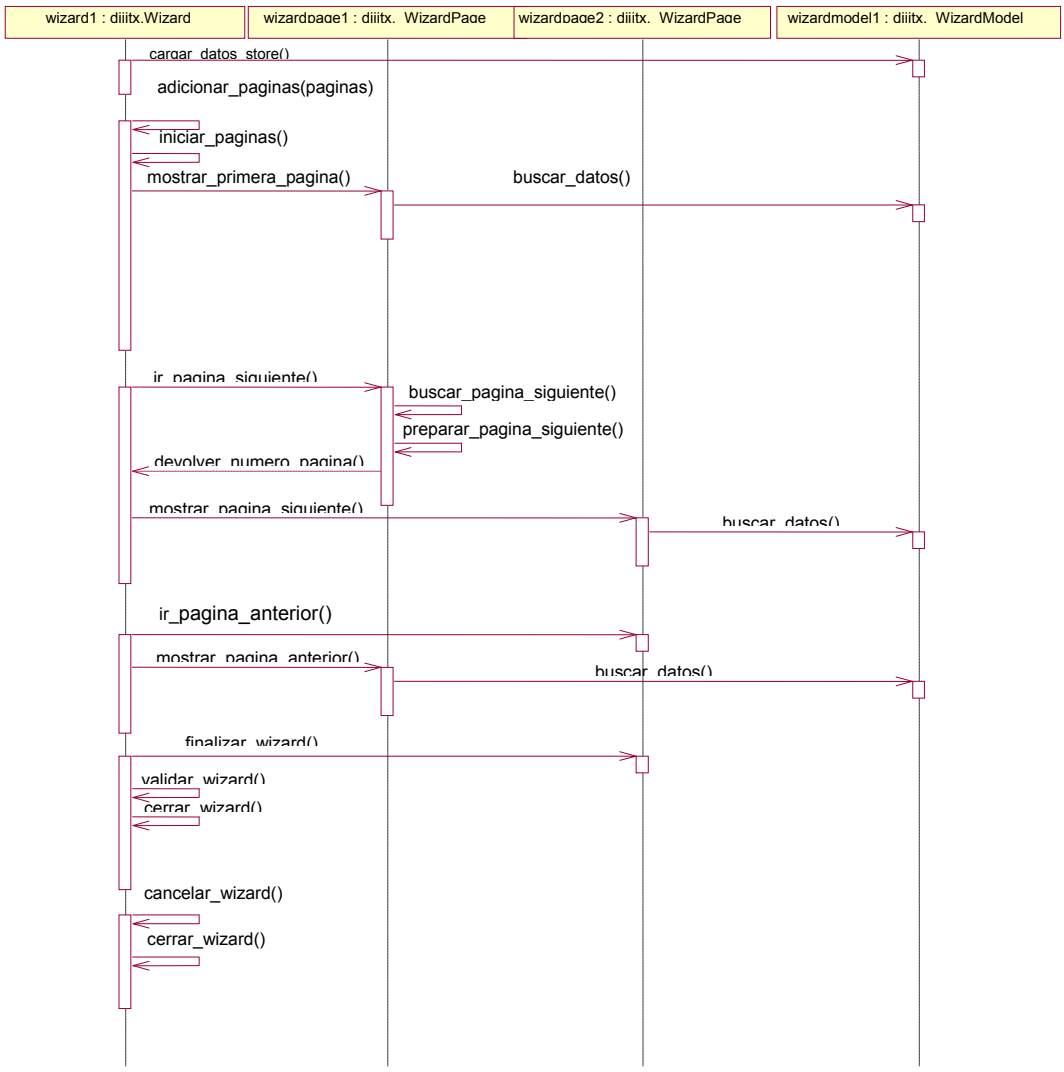


Tabla 4 Diagrama de secuencias del Wizard.

2.7 Descripción del diseño del Menú vertical con estructura de árbol.

2.7.1 Representación de los Diagramas de Diseño.

Diagrama de clases del diseño del Menú desplegable.

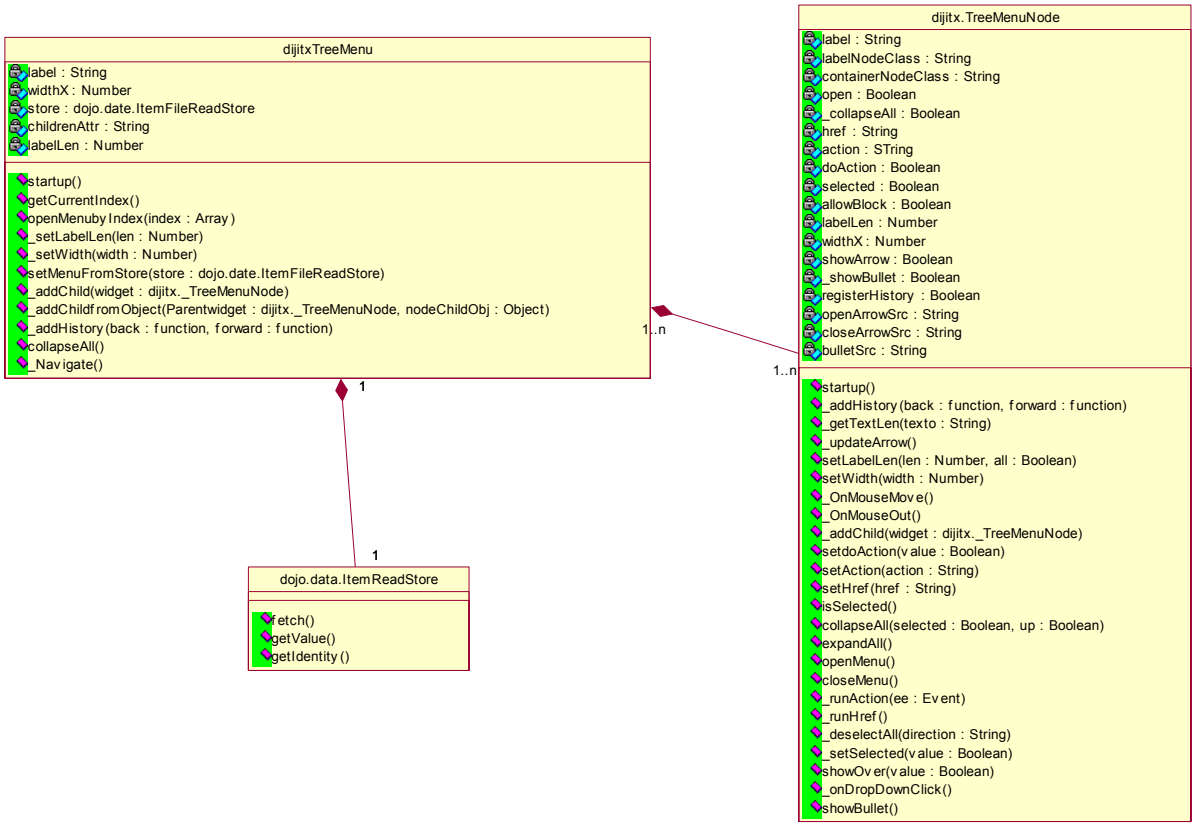


Tabla 5 Diagrama de diseño del del Menú desplegable

2.7.2 Descripción de las clases del diseño del Menú vertical.

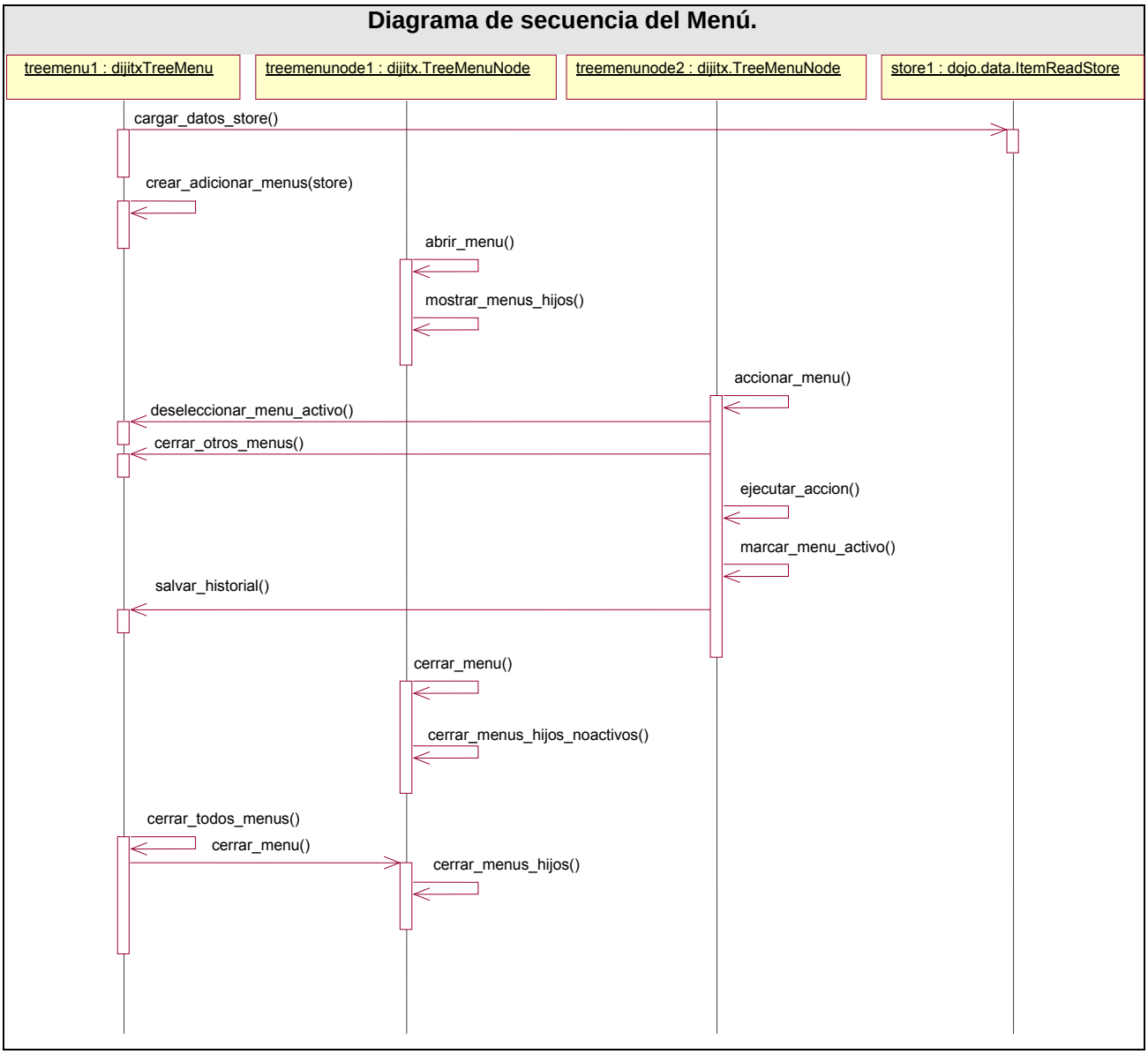
Nombre: dijitx.TreeMenu
Descripción de la clase: Clase que representa un menú con estructura de árbol, es la raíz del árbol, contiene menús hijos de tipo dijitx._TreeMenuNode, estos a su vez pueden tener más menús hijos.

Atributo	**Tipo**
Label	String
widthX	Number
Store	dojo.date.ItemFileReadStore
childrenAttr	String
labelLen	Number
Para cada responsabilidad:	
Nombre:	Startup
Descripción:	Método que inicia el menú, se ejecuta luego de creado este. Lee la información del store, crea y adiciona los menús hijos.
Nombre:	getCurrentIndex
Descripción:	Retorna un array con los índices de los menús que componen el camino hasta el menú seleccionado actualmente.
Nombre:	openMenubyIndex
Descripción:	Selecciona el menú especificado por el array de índices que indica el camino a seguir.
Nombre:	_setLabelLen
Descripción:	Específica la nueva longitud de caracteres máxima para los label de cada menú.
Nombre:	_setWidth
Descripción:	Específica el nuevo ancho de cada menú.
Nombre:	_addChild
Descripción:	Adiciona un nuevo menú como hijo de este.
Nombre:	setMenuFromStore
Descripción:	Crea y adiciona los menús obtenidos de la información en el store. Remueve los menús que ya existían.
Nombre:	collapseAll
Descripción:	Cierra cada uno de los menús hijos y los hijos de estos.

Nombre: dijitx._TreeMenuNode	
Descripción de la clase: Clase que representa un elemento del menú, un menú que puede tener un padre y además muchos menús hijos, provee la funcionalidad básica del menú.	
Atributo:	**Tipo:**
Label	String
Open	String
_collapseAll	Boolean
Href	String
Selected	Boolean
labelLen	Number
showArrow	Boolean
Para cada responsabilidad:	
Nombre:	startup
Descripción:	Método que inicia el menú, se ejecuta luego de creado este. Inicializa propiedades y conecta eventos.
Nombre:	_addHistory
Descripción:	Adiciona al historial del navegador la entrada correspondiente a este menú.
Nombre:	_updateArrow
Descripción:	Actualiza la flecha que indica que el menú está abierto o cerrado, en dependencia del estado actual del mismo.
Nombre:	setWidth
Descripción:	Específica un nuevo ancho para el menú, se aplicará el mismo ancho a todos sus menús hijos.
Nombre:	isSelected
Descripción:	Devuelve true si el menú esta seleccionado, false de lo contrario, un menú esta seleccionado si alguno de sus hijos lo está.
Nombre:	collapseAll
Descripción:	Cierra todos los menús hijos de este, y opcionalmente todos sus hermanos

	recursivamente hasta cerrar todos los menús, exceptuando opcionalmente los menús seleccionados.
Nombre:	expandAll
Descripción:	Expande todos los menús hijos recursivamente.
Nombre:	openMenu
Descripción:	Abre el menú, muestra los menús hijos.
Nombre:	closeMenu
Descripción:	Cierra el menú, oculta los menús hijos.

2.7.3 Diagrama de secuencia del Menú desplegable.



2.8 Descripción de los Diagramas de diseño del Calendario.

2.8.1 Representación de los Diagramas del Diseño.

Diagrama de clases del diseño del Calendario

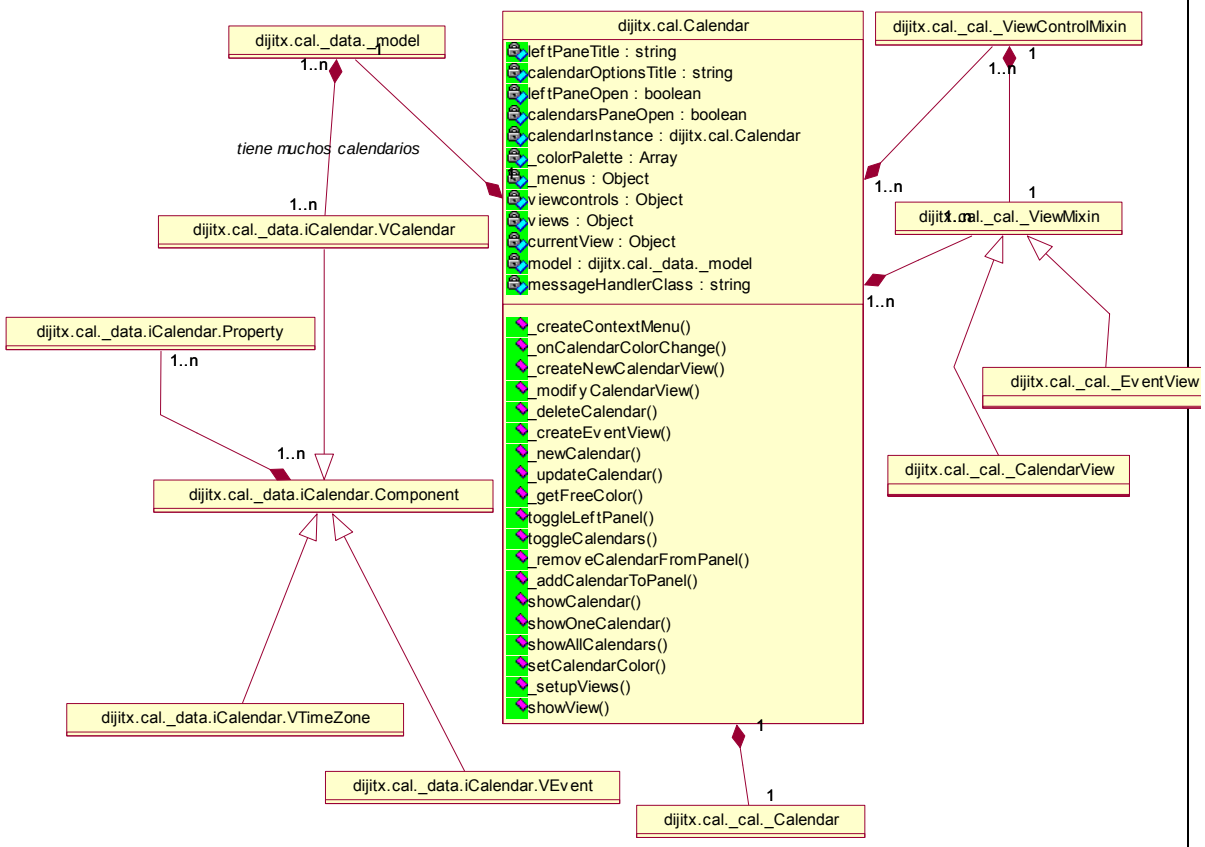


Tabla 7 Diagrama de clases del diseño del Calendario

2.8.2 Descripción de las clases del Diseño del Calendario.

Nombre de la clase: dijitx.cal.Calendar	
Descripción de la clase: Clase principal del componente del Calendario, implementa el estándar RFC2445, llamado iCalendar. Provee una interfaz para trabajar con calendarios.	
Atributo	**Tipo**
views	Object
model	dijitx.cal._data._model

leftPaneTitle	String
calendarOptionsTitle	String
_colorPalette	Object
viewcontrols	Object
Por cada responsabilidad	
Nombre:	_createContextMenu
Descripción:	Crea un menú contextual a partir de la información proporcionada, los nodos sobre los cuales funcionará, la información de las opciones del menú y el id que tendrá asociado.
Nombre:	_getIdFromMenu
Descripción:	Devuelve el id del calendario en menús contextuales que accionan sobre calendarios.
Nombre:	_onCalendarColorChange
Descripción:	Método que se ejecuta cuando se selecciona un color nuevo para un calendario.
Nombre:	_contextCreateNewCalendar
Descripción:	Método que se ejecuta cuando se selecciona la opción de crear un nuevo calendario desde el menú contextual.
Nombre:	_contextDeleteCalendar
Descripción:	Se ejecuta cuando se selecciona la opción de eliminar un calendario desde el menú contextual.
Nombre:	_contextNewEvent
Descripción:	Se ejecuta cuando se selecciona la opción de crear un nuevo evento desde el menú contextual.
Nombre:	_contextShowAllCalendar
Descripción:	Se ejecuta cuando se selecciona la opción de mostrar todos los calendarios desde el menú contextual.
Nombre:	_createNewCalendarView
Descripción:	Muestra la vista para crear un nuevo calendario.
Nombre:	_modifyCalendarView

Descripción:	Muestra la vista para modificar el calendario seleccionado
Nombre:	_newCalendar
Descripción:	Crea un nuevo calendario, dados las propiedades y la zona horaria que usará.
Nombre:	showCalendar
Descripción:	Muestra u oculta los eventos del calendario especificado
Nombre:	showOneCalendar
Descripción:	Muestra solamente los eventos del calendario especificado, ocultando todos los restantes.
Nombre:	setCalendarColor
Descripción:	Aplica un color sobre el calendario especificado.

Diagrama de secuencia del Calendario de Eventos.

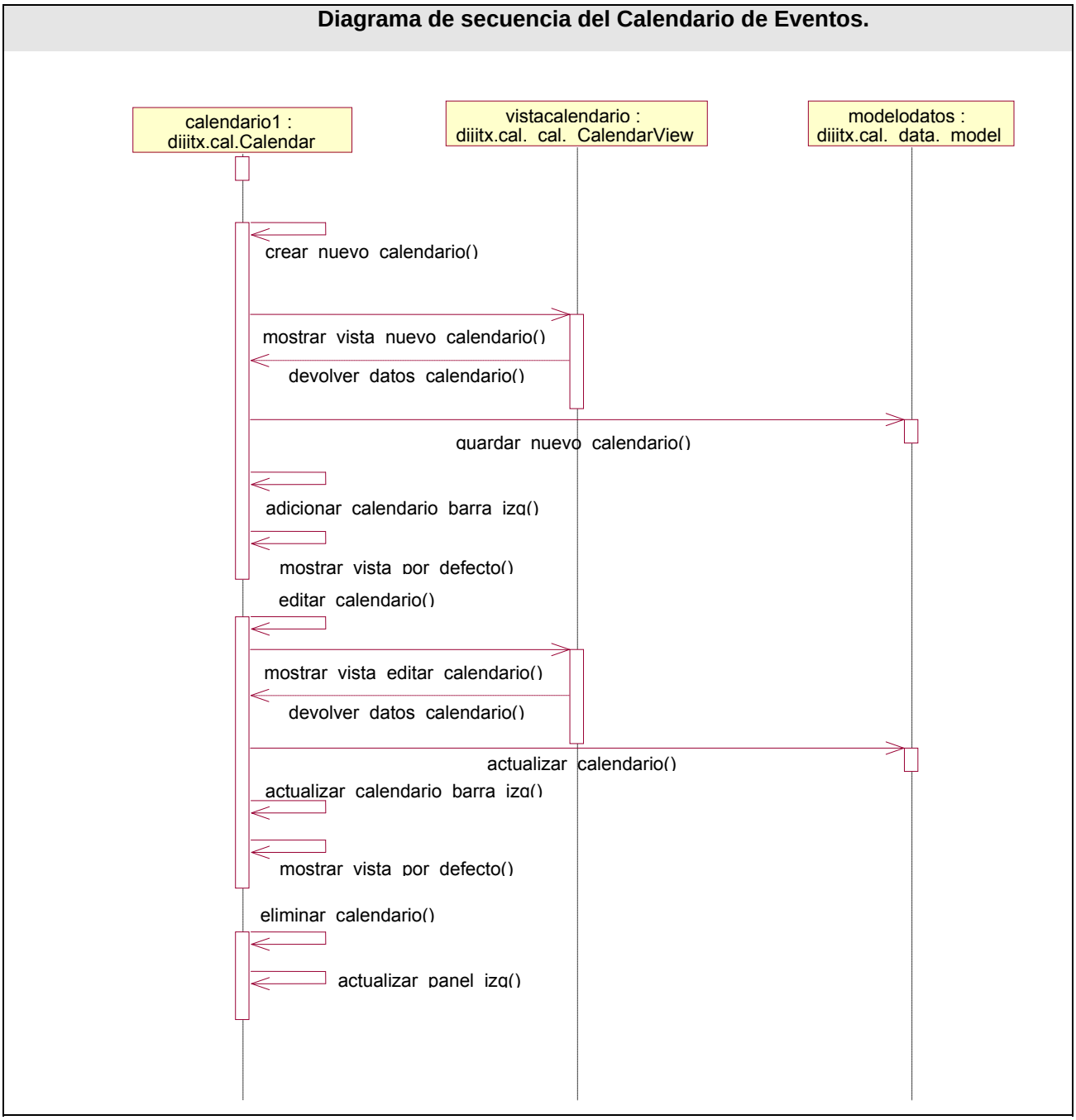


Tabla 8 Diagrama de secuencia. Escenario Gestionar Calendarios.

Diagrama de secuencia del Calendario de Eventos.

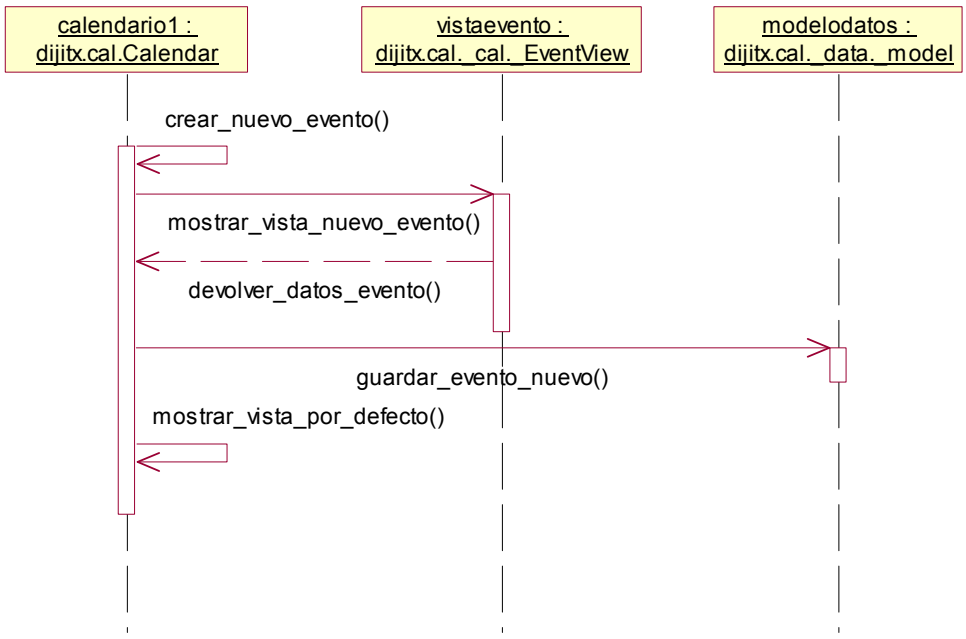


Tabla 9 Diagrama de secuencia. Escenario Crear Evento

CONCLUSIONES

Se ha presentado en el capítulo una descripción que permita conducir a la construcción de una solución que cumpla con los objetivos propuestos, se realizó un estudio de los patrones de diseño, se mostraron los Diagramas de clases y de interacción ambos con la intención de guiar el trabajo en la implementación de la Librería.

CAPÌTULO 3: CONSTRUCCIÒN DE LA SOLUCIÓN.

3.1 Introducción

En este capítulo se describen los estándares utilizados en la implementación de la Librería, la manera de desarrollar componentes utilizando Dojo, se muestra además la estructura física de la Librería mediante los Diagramas de componentes también se le realizan las pruebas al producto final.

3.2 Estándares utilizados en la interfaz

Las interfaces de usuario (IU) en un inicio eran desarrolladas pensando solamente en su correcto funcionamiento, luego se comprobó la necesidad de que fueran fácilmente usables y después se hizo evidente la importancia de la estética en las mismas acorde con las inquietudes del mundo que nos rodea colores, formas, agrupaciones y comunicación forman en la actualidad parte indispensable de una interfaz de usuario.

En los widgets que contiene la librería no se utilizan elementos contrastantes o colores fuertes, la fuente tipográfica tiene el tamaño necesario para que se lea sin esfuerzo. Cada elemento sigue un patrón de tamaño y formas, siendo adecuados con el formato y seriedad, tratando también de no alejar mucho la estructura en que se visualizan, aunque estas opciones se dejan a consideración del programador, recordar que dependen del proyecto en el cual se utilicen dichos componentes.

3.2.1 Estándares de codificación.

Para la implementación se siguió un estándar de codificación a utilizar, dividido fundamentalmente en los siguientes aspectos:

Comentarios:

JavaScript define 2 tipos de comentarios: los de una sola línea y los que ocupan varias líneas.

Ejemplo de comentario de 1 sola línea:

Fragmento de código JavaScript

```
// a continuación se muestra un mensaje
alert("mensaje de prueba");
```

Ejemplo de comentario de varias líneas:

Fragmento de código JavaScript

```
/* Los comentarios de varias líneas son muy útiles cuando se necesita
   incluir bastante información en los comentarios */
alert("mensaje de prueba");
```

Declaraciones:

Se utilizan los Patrones de idiomas comúnmente reconocidos como estándares de codificación y proyecto, los métodos de Camel Casing o Mayúsculas y Minúsculas Camel, en este método la primera letra del identificador está en minúscula y la primera letra de las siguientes palabras concatenadas en mayúscula, por ejemplo:

"camelCasing"

También se utiliza el método de Pascal Casing, el cual a diferencia del Camel Casing la primera letra del identificador no debe estar en minúsculas, por ejemplo:

"**P**ascal**C**asing"

El nombre de la Librería es dijitx, es por eso que las declaraciones de las clases será de la forma nombre de la Librería, nombre de la clase, ejemplo dijitx.DoubleListBox.

Espacios en blanco:

Se colocarán espacios en blanco entre operadores lógicos-aritméticos y sus operandos.

Ejemplo:

Fragmento de código JavaScript

```
if (evento.fechaInicio < fecha) {
     alert('El evento tiene una fecha menor');
}
```

3.3 Componentes con Dojo Toolkit.

Crear componentes web escritos en JavaScript no suele ser sencillo cuando se requiere que estos incorporen usabilidad, sean extensibles y permitan desarrollar interfaces web con la rapidez que se necesita. Por eso Dojo Toolkit incluye todo un proyecto dedicado a la creación de componentes, su nombre es Dijit, este incorpora una amplia gama de componentes, desde un sencilla caja de texto que permite validación hasta complicados componentes como puede ser un Editor de texto con su barra de herramientas. Las aplicaciones web requieren de nuevos componentes todo el tiempo, no basta con los existentes. Dijit brinda una forma de hacer extensible los componentes para crear nuevos o crear estos desde cero.

Un componente está compuesto por varios elementos, tres por lo general, un archivo HTML que servirá como plantilla o esqueleto, un archivo css que tendrá los estilos usados para brindar una presentación agradable y un archivo JavaScript el cual define el comportamiento y las funcionalidades que brindará dicho componente, además de unir todos estos elementos como un todo.

3.3.1 La plantilla

La plantilla es un archivo HTML que contiene información acerca del esqueleto del componente, Dojo provee algunas facilidades para evitar escribir luego código JavaScript adicional al construir plantillas. Especifica dos propiedades:

- **dojoAttachPoint:** Su función es asignar nombres a nodos HTML en la plantilla para luego referenciarlos desde JavaScript usando dichos nombres, por ejemplo en el siguiente código se define que el botón será referenciado mediante JavaScript como la propiedad ‘finishButton’ del componente:

Fragmento de código HTML

```
<button dojoAttachPoint="finishButton">
    Finalizar
</button>
```

- **dojoAttachEvent:** Su función es asignar que método(s) del componente se ejecutará al dispararse cierto evento del nodo seleccionado. Por ejemplo el siguiente código indica que al dar clic sobre el botón referenciado como ‘finishButton se ejecutará el método ‘_onFinishClick’

del Wizard que se ejecuta al dar clic sobre el botón que finaliza el Wizard. Además incorpora otra facilidad, que son el signo de $ y las llaves de la siguiente forma ${}, así se sustituye en tiempo de creación el valor de la propiedad 'finishLabel' del componente por ${finishLabel} en la plantilla. De esta forma el texto mostrado en el botón puede ser configurable al depender del valor de una propiedad y no necesariamente ser estático.

Fragmento de código HTML

```
<div>
      <input type="button" value="${finishLabel} dojoAttachPoint="finishButton"
              dojoAttachEvent="onclick: _onFinishClick" />
</div>
```

3.3.2 CSS

Este archivo css contiene todas las clases que se encargan de la presentación de cada uno de los elementos que están contenidos en los componentes, es donde se especifican el tamaño, color, tipo de letra y demás propiedades que definen cómo se visualizan los elementos en una página. Aunque estas propiedades se pueden definir en el propio archivo HTML es recomendable dedicar un archivo solamente para estas cuestiones, la principal ventaja radica en que el código es más sencillo de mantener, pues un cambio en un solo archivo CSS permite variar de forma instantánea los estilos de todos los documentos HTML que enlazan ese archivo.

En el caso específico de la Librería se creó un archivo css con el nombre tundrax.css que contiene todos los estilos usados por los componentes en su interfaz.

El siguiente código muestra una clase css que define el ancho, la altura y el color de fondo, de forma tal que solo faltaría asignarle esta clase al elemento deseado.

Fragmento de código CSS

```
.tundrax .botonContenedor
{
  width: 200px;
  height: 100px;
  background-color: blue;
}
```

En el siguiente código se le asigna al elemento div el estilo definido en la clase botonContenedor.

Fragmento de código HTML

```
<div class="botonContenedor">
    <input type="button" value="${finishLabel} dojoAttachPoint="finishButton"
           dojoAttachEvent="onclick: _onFinishClick" />
</div>
```

Dojo también incorpora facilidades para el trabajo con las clases css, estas se encuentran en el proyecto Dojo Core. Brinda funciones para adicionar, remover y obtener clases CSS en nodos HTML. Provee además un selector CSS que permite buscar nodos HTML según un selector CSS3[10] especificado, por ejemplo para encontrar todos los nodos que tienen la clase .botonContenedor y que sean elementos de tipo DIV sería de la siguiente manera.

Fragmento de código JavaScript

```
dojo.query("div.botonContenedor");
```

Devuelve un array con todos los nodos HTML que cumplen con la consulta especificada.

3.3.3 JavaScript

Es un archivo (script) con extensión js donde se define el comportamiento y las funcionalidades diseñadas para cada uno de los componentes además de unir como un todo funcional los archivos html y css anteriormente especificados.
Dojo también incorpora una vía para programar en JavaScript orientado a objetos, permite crear clases y extender de existentes, para esto se usan varias funciones, tal es el caso:

dojo.declare: Su función es crear una nueva clase, permitiendo especificar de cuales extenderá si es necesario, luego se especifica el constructor, las propiedades y métodos.

[10] CSS3, Nivel 3 de las Hojas de Estilos en Cascada.

Por ejemplo el siguiente código muestra la declaración de una clase en JavaScript usando Dojo que define al componente que maneja dos listas de elementos.

Cada componente que se encuentran en Dijit son al final una clase Dojo pero que tiene características específicas, pues todos los widget en Dijit extienden de una clase base que provee las propiedades y funcionalidades básicas de un componente, esta clase es **dijit._Widget**. Por lo tanto para crear el widget, se crea una clase para este.

Fragmento de código JavaScript

```
dojo.declare("dijitx.DoubleListBox", [dijit._Widget, dijit._Templated], {
    valueAttr: "value",
    labelAttr: "label",
    paneAttr: "pane",
    revert: function(){
            //revertir cambios
    },
    destroy: function(){
            //destruir componente
    }
});
```

Solo falta enlazar la plantilla para que quede conformado el componente. El uso de plantillas no es obligatorio, pueden existir componentes que no tengan una interfaz visible y que por consiguiente no usen una plantilla para su presentación. Ahora, para usar una plantilla en el componente, la clase del widget debe extender a **dijit._Templated** como se especificó en el ejemplo anterior**,** pues esta es la clase que brinda Dijit que contiene las funcionalidades necesarias para trabajar con plantillas. Luego a la clase se le agrega la propiedad **templatePath** para especificar el camino al archivo HTML que servirá como plantilla. Por ejemplo:

Fragmento de código JavaScript

```
dojo.declare("dijitx.DoubleListBox", [dijit._Widget, dijit._Templated], {
    templatePath: "js/templates/DoubleListBox.html",
});
```

De esta forma el componente conoce cuál es la plantilla a usar, y mediante los dojoAttachPoint y dojoAttachEvent especificados, cuales nodos debe referenciar en propiedades y cuales eventos debe

conectar a que métodos del componente. Solo resta implementar las funcionalidades trazadas para el componente en cuestión una vez construido una plantilla y los estilos css apropiados. Estas funcionalidades se resumen a propiedades y métodos que se adicionan a la clase que representa al componente. Los estilos CSS son referenciados desde la propia plantilla haciendo uso de las clases directamente desde el HTML. Es posible que algunos estilos se apliquen en tiempo de ejecución, en esos casos Dojo sirve de ayuda con las funcionalidades que brinda para trabajar con CSS antes mencionadas.

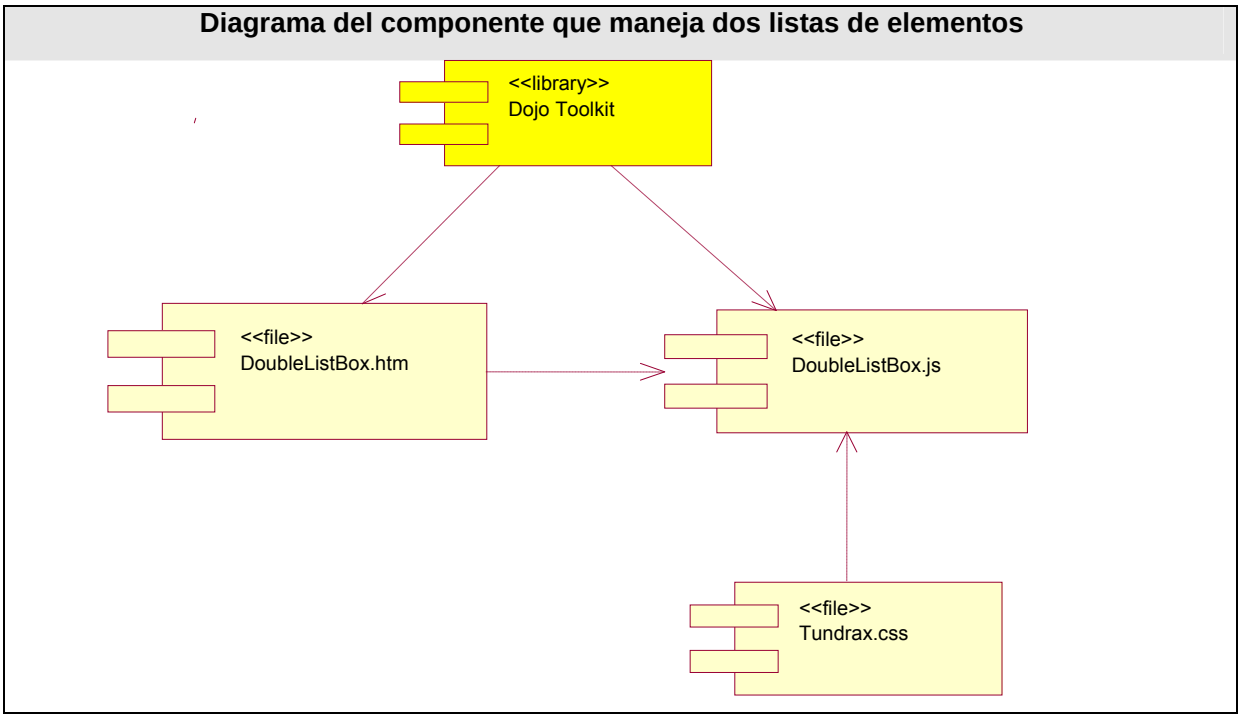
3.4 Estructura física mediante Diagramas de Componentes

Las vistas físicas modelan la estructura de la implementación de la aplicación por sí misma. Estas vistas proporcionan una oportunidad de establecer correspondencias entre las clases y los componentes de implementación, su organización en componentes (Diagrama de componentes) y su despliegue en nodos ejecución (Diagrama de Despliegue), no se presenta un Diagrama de Despliegue porque es imposible distribuir el sistema asignando componentes ejecutables a nodos en el diagrama de despliegue debido a que los componentes implementados no constituyen por sí solos una aplicación en su totalidad, se debe tener en cuenta en el momento de integrarse a una aplicación.
Los diagramas de componentes modelan la vista estática de un sistema. Muestra la organización y las dependencias lógicas entre un conjunto de componentes software, sean éstos componentes de código fuente, librerías, binarios o ejecutables.

3.5 Representación de los Diagramas de componentes.

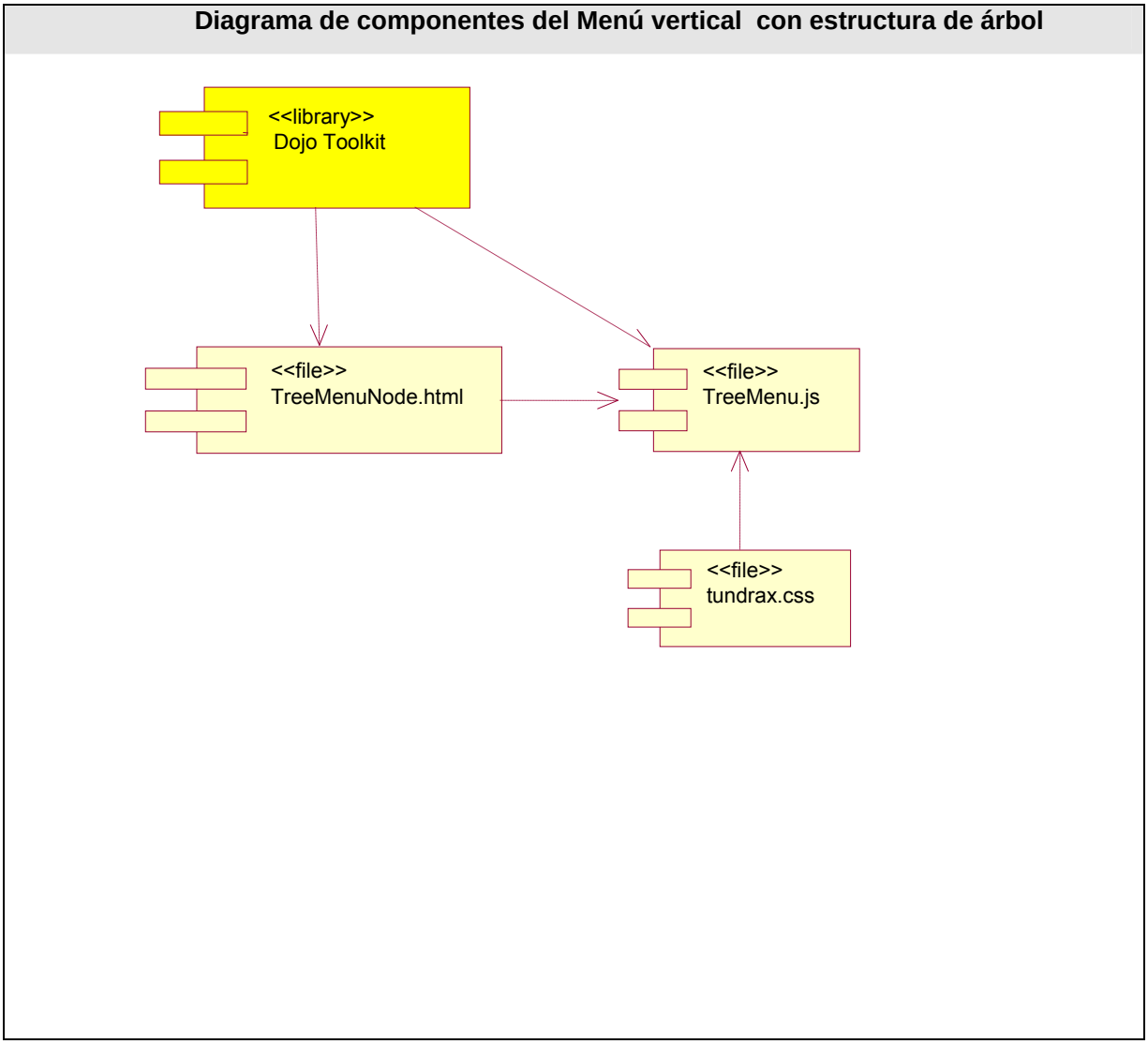
Representan la forma o estructura física que se utilizó en la implementación de la Librería. Cada uno de los componentes contiene un archivo HTML (plantilla), css (presentación) y js (Javascript) todos basados en Dojo Toolkit, que en su conjunto definen el comportamiento y visualización de la Librería.

3.5.1 Diagrama del componente que maneja dos listas de elementos.

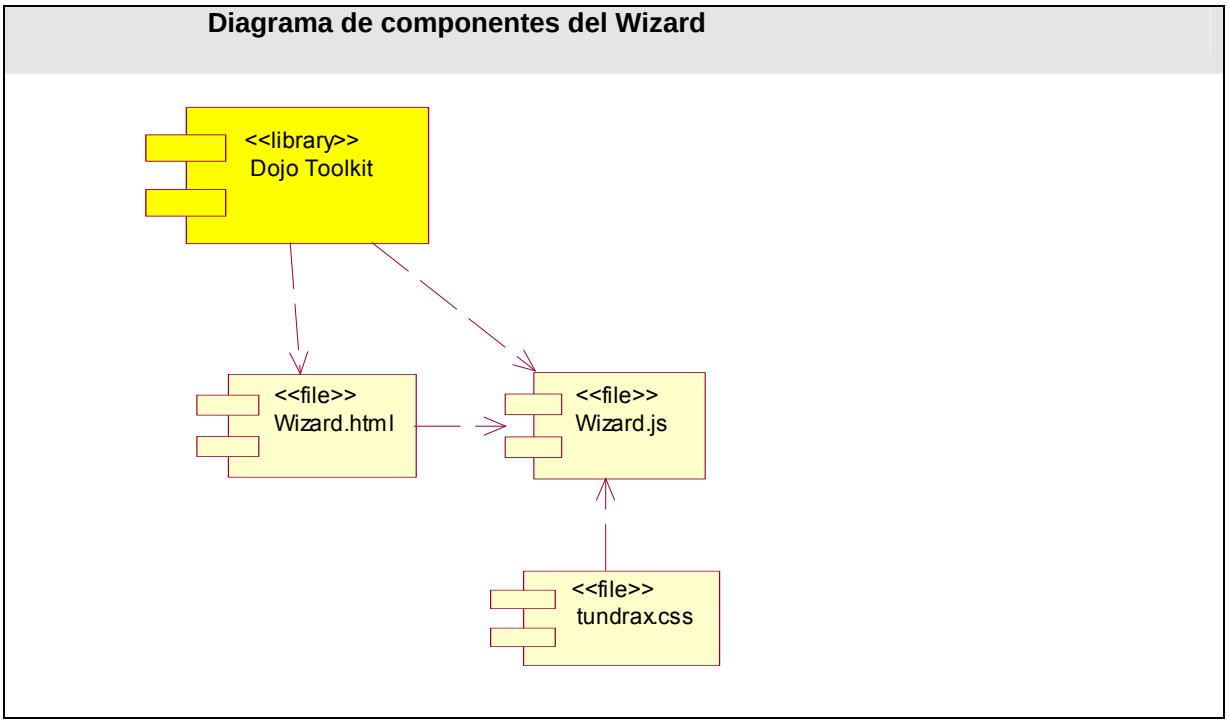


3.5.2 Diagrama de componentes del Menú vertical con estructura de árbol.

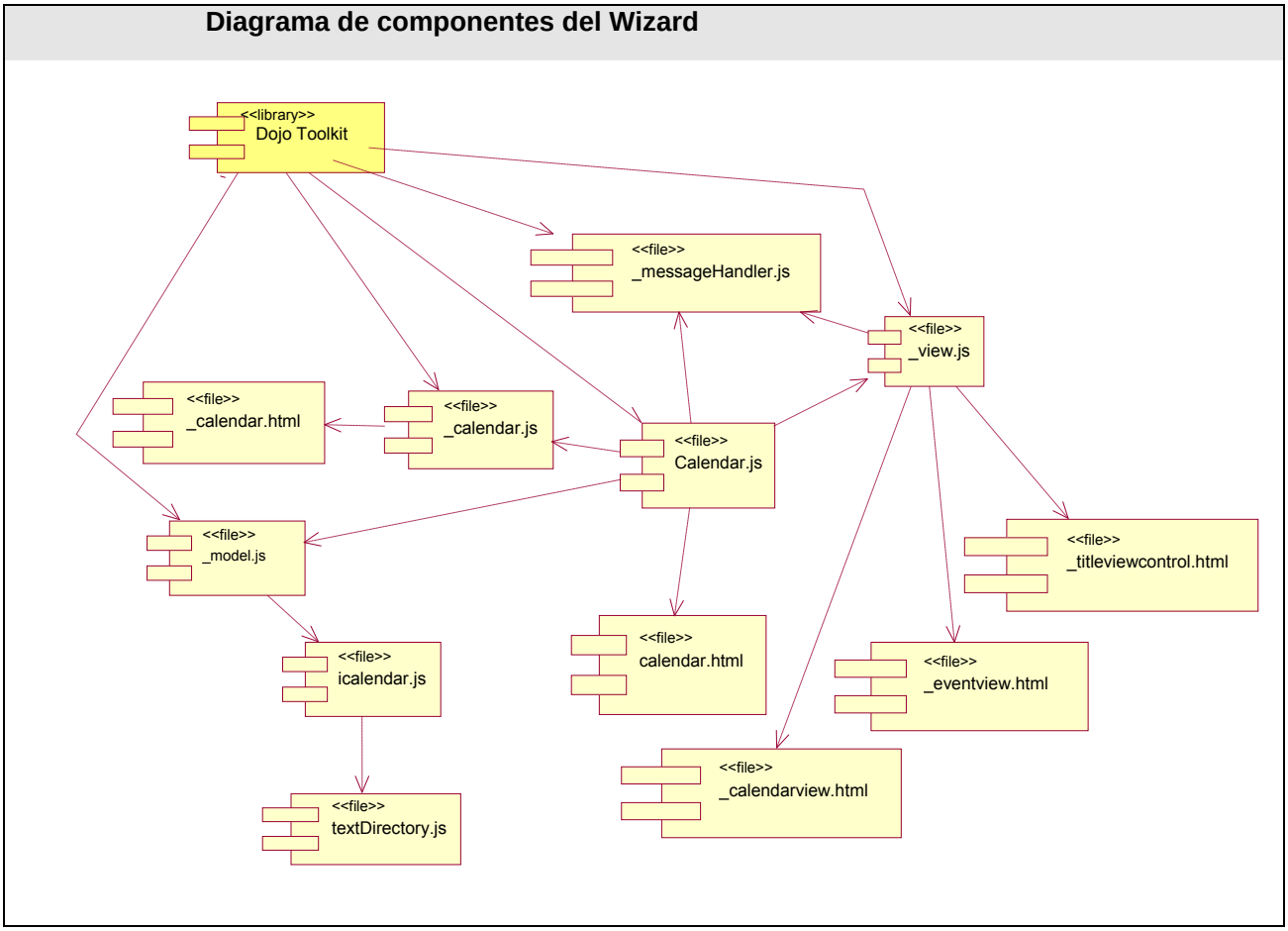
Diagrama de componentes del Menú vertical con estructura de árbol



3.5.3 Diagrama de componentes del Wizard.



3.5.4 Diagrama de componentes del Calendario.



3.6 Concepción de la Ayuda.

La necesidad de contar con la documentación que permita orientar al programador en el momento de utilizar la Librería, llevó a la creación de un API[11] que lograra construir de forma automática la Ayuda, parecido al javadoc de Java, que genera la documentación a partir del código fuente.

[11] Una **API** (del inglés **Application Programming Interface - Interfaz de Programación de Aplicaciones**) es el conjunto de funciones y procedimientos que ofrece cierta librería para ser utilizado por otro software como una capa de abstracción.

La ayuda consiste en un sitio web donde se encuentran referenciadas todas las clases usadas en los componentes, describiendo para cada una de ellas su objetivo, propiedades y métodos.
A continuación se describe el API creado, el cual genera la ayuda en formato HTML a partir de los comentarios en los ficheros JavaScript.

3.6.1 Uso de Ant

La documentación fue generada utilizando una herramienta de construcción de proyectos, conocida como Ant, diseñado para ayudar a los equipos de software en el desarrollo de grandes programas, automatizando el trabajo de compilar el código, correr pruebas y empaquetar los resultados de la distribución.
Ant es multiplataforma, fácil de usar, extensible, escrito en Java, con una sintaxis de XML. Se eligió Ant para la generación de la documentación debido a que es una herramienta que permite automatizar el proceso mediante tareas escritas en Java.

3.6.2 API

El API consiste en una tarea de Ant, varias plantillas HTML para crear el sitio y define una sintaxis de los comentarios requeridos para generar la ayuda.
Se creó una tarea de Ant, llamada dijitx.jsFileDocMaker, es una clase Java encargada de leer los ficheros JavaScript y generar la documentación usando las plantillas HTML definidas. Además se crearon clases para soportar cada tipo de los comentarios que se definen más adelante.

Nombre de Clase	**Descripción**
dijitx.WidgetClass	Soporta comentarios de tipo clase.
dijitx.WidgetProperty	Soporta comentarios de tipo propiedad.
dijitx.WidgetMethod	Soporta comentarios de tipo método.
dijitx.WidgetParam	Soporta comentarios que describe un parámetro.

Los comentarios en el código a partir de los cuales se genera la documentación se escriben de la siguiente manera:
Se especifican tres tipos de comentarios, todos múltiples empezando con '/**' y terminando con '*/'.

- Para definir una clase
- Una propiedad

- Un método y sus parámetros

De la clase se define:

Palabra Clave	Significado	Sintaxis
class:	El nombre completo de la clase.	class:”NombreClase”
extends:	Los nombres de las clases de la cual hereda separados por coma.	extends: ”Clase1,Clase2”
sumary:	Descripción de la clase.	sumary:”La clase NombreClase sirve para ….”

Ejemplo:

Fragmento de comentario

```
/**
    class: "dijitx.TreeMenu"
    extends: "dijit._Widget,dijit._Templated,dijit._Container"
    sumary: "Clase que representa un menú con estructura de árbol, es la raíz del
    árbol, contiene menús hijos de tipo dijitx._TreeMenuNode, estos a su vez
    pueden tener más menús hijos."
*/
```

De una propiedad se define:

Palabra Clave	Significado	Sintaxis
property:	El nombre de la propiedad en cuestión.	property:”propiedad1”
type:	El tipo de datos de la propiedad.	type:”String”
default:	Valor por defecto de la propiedad.	type:”valor1”
sumary:	Descripción de la propiedad.	sumary:”Descripción de la propiedad propiedad1”

Ejemplo:

Fragmento de comentario

```
/**
      property: "label"
      type: "String"
      default: "label1"
      sumary: "Representa el texto que serviría como label de este menú."
*/
```

De un método se define:

Palabra Clave	**Significado**	**Sintaxis**
method:	Nombre del método en cuestión.	method:”Metodo1”
returns:	Tipo de dato que retorna si es que retorna algo. Se puede omitir.	returns:”String”
param:	Se describe un parámetro según se muestra en la tabla de parámetros.	param:” name:’nombreparametro’ type:’tipodato’ sumary:’descripción’ ”
sumary:	Descripción del método.	sumary:”descripción del metodo1”

De un parámetro de un método se define:

Palabra Clave	**Significado**	**Sintaxis**
name:	El nombre del parámetro.	name:’parametro1’ (Con comillas simples)
type:	Tipo de dato del parámetro.	type:’String’ (Con comillas simples)
sumary:	Descripción del parámetro.	sumary:’Descripción de parametro1’(Con comillas simples)

Ejemplo de un método con parámetros:

Fragmento de comentario

```
/**
      method:"setMenuFromStore"
      param: "
            name:'store'
            type:'dojo.date.ItemFileReadStore'
            sumary:'Store con la información de los menús.'"
      sumary:"Crea y adiciona los menús obtenidos de la información en el store.
      Remueve los menús que ya existían."
*/
```

En un mismo archivo Javascript se pueden comentar varias clases, se comienza comentando la primera clase con un comentario de clase, luego se escriben tantos comentarios de propiedades y métodos como se necesiten e inmediatamente se puede comentar otra clase definiendo nuevamente otro comentario de tipo clase seguido por sus propiedades y métodos. Así se pueden definir cuántas clases se necesite en un mismo archivo. El resultado final después de que se han definido los comentarios, mediante la tarea especificada de Ant es generada la documentación. (Ver figura).

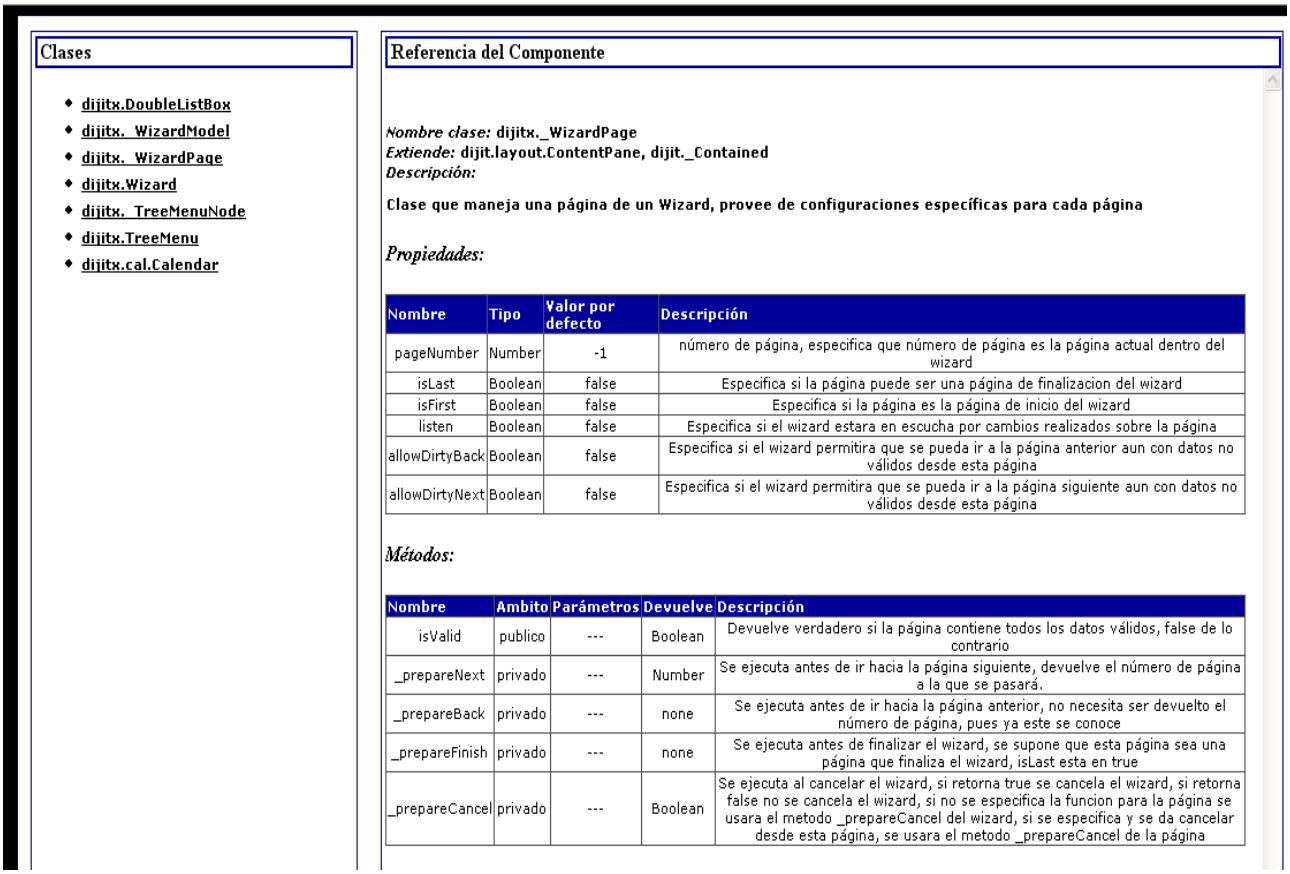


Fig. 2 Sitio web que muestra la documentación de los componentes.

3.7 Prueba

En las pruebas verificamos el resultado de la implementación. El objetivo de las pruebas es verificar el producto de software para comprobar si el mismo cumple los requisitos. Se pueden desarrollar varios tipos de pruebas en función de los objetivos de las mismas, entre ellas, las pruebas funcionales de aplicación con interfaces gráficas que verifican si el software ofrece o no las funcionalidades especificadas.

3.7.1 Las Pruebas con Dojo Toolkit

Dojo Toolkit provee una utilidad que permite realizar pruebas haciendo uso de una página donde se utiliza el componente, teniendo en cuenta que cada componente tiene una o más páginas de prueba, las cuales abarcan cada una de las funcionalidades de cada widget, es posible verificar en tiempo real

cada una de las funcionalidades que brindan los componentes. Esta herramienta es llamada DOH (Dojo Objectives Harness), un módulo que no depende de Dojo, el cual permite realizar dos tipos de pruebas:

- Pruebas al código JavaScript, generalmente llamadas a métodos y verificación de resultados.
- Pruebas manuales, se ejecutan una o varias páginas de prueba donde se comprueba de forma manual que se cumplen con las funcionalidades previstas.

3.7.1.1 Pruebas manuales

Las pruebas manuales se ejecutan mediante DOH. Cada componente tiene una o varias páginas de prueba, las cuales se encuentran en la carpeta “tests” dentro de la librería.
Por ejemplo el componente del Menú tiene dos páginas de prueba, test_TreeMenu.html y test_TreeMenuDOH.html, así como los demás componentes también tienen sus páginas de prueba. DOH brinda una interfaz capaz de cargar las páginas de pruebas especificadas mediante la siguiente configuración.

Fragmento de código JavaScript

```
/* doh.registerUrl registra una prueba manual, se le pasan el nombre del grupo de
prueba,
   la url del archivo HTML a probar y la cantidad de milisegundos máximo
   que se espera hasta que la prueba cargue completamente. */

doh.registerUrl("Pruebas del componente menú", "dijitx/tests/test_TreeMenu.html",
10000);

doh.registerUrl("Pruebas del componente menú",
"dijitx/tests/test_TreeMenuDOH.html", 10000);

doh.registerUrl("Pruebas del componente de dos Listas", "dijitx",
"dijitx/tests/test_ DoubleListBox.html ", 10000);
```

Se ejecutan las pruebas correspondientes al componente de las dos listas y el menú.

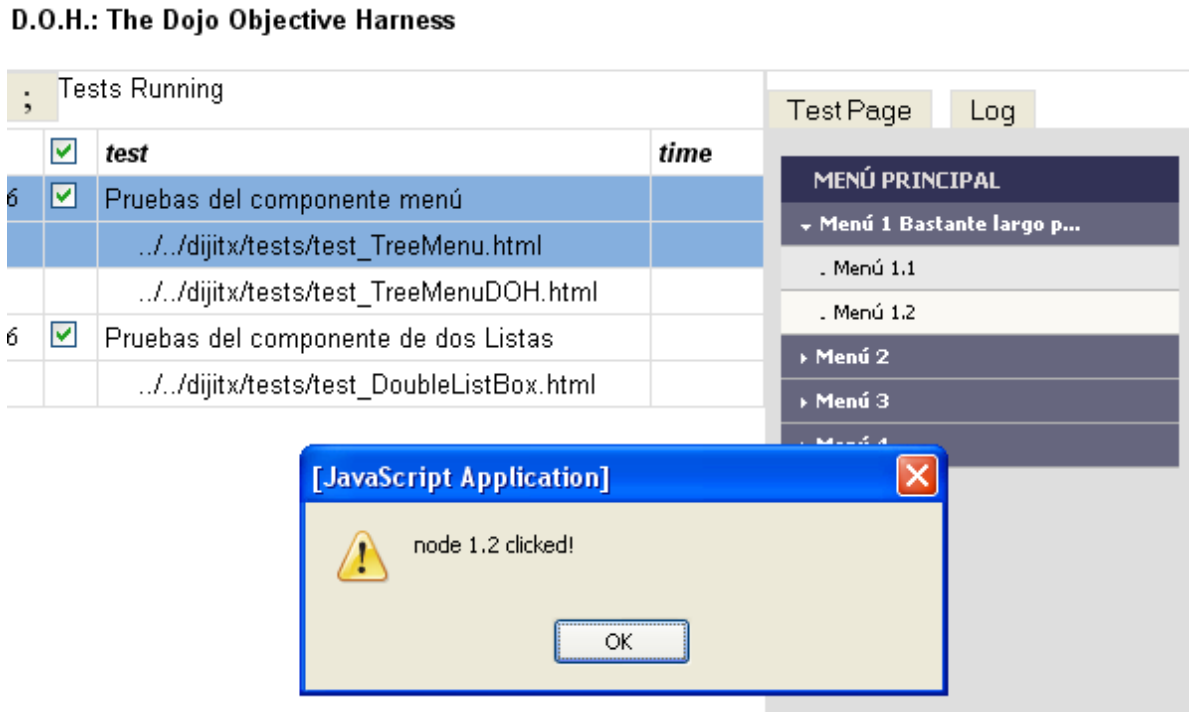


Fig. 3 Pruebas Manuales realizadas al Menú vertical con estructura de árbol.

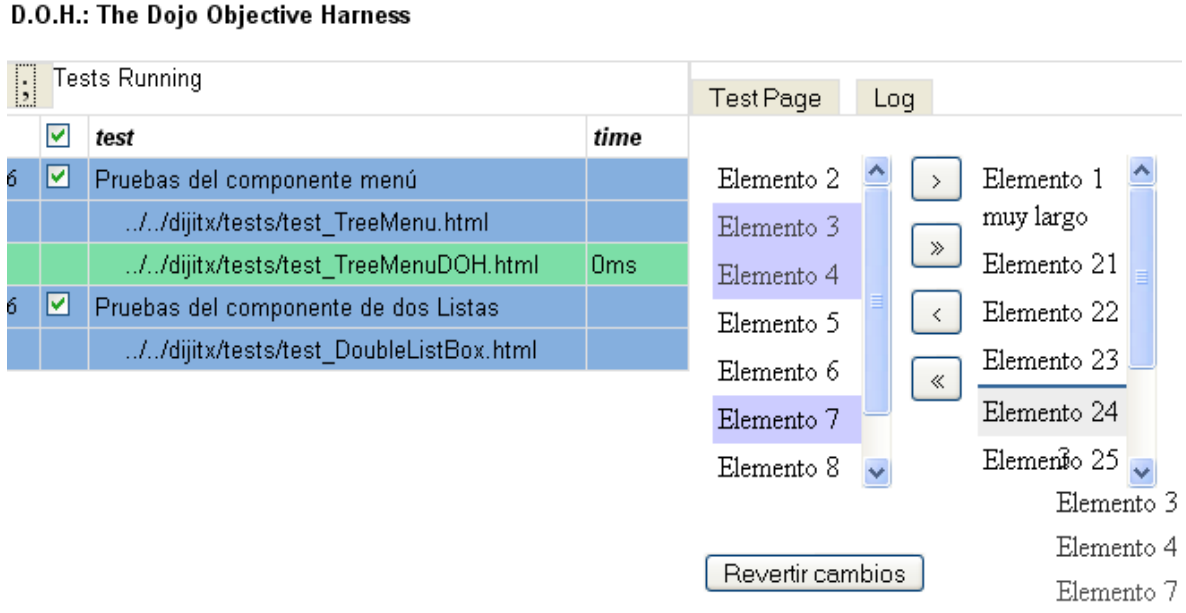


Fig. 4 Pruebas manuales realizadas al componente que maneja dos listas de elementos.

3.7.1.2 Pruebas al código

Las pruebas realizadas al código se simplifican a verificar que los métodos realizan lo debido y devuelven lo que deben retornar en el momento adecuado y que las variables tomen los valores esperados. DOH provee soporte para hacer esto, además de permitir probar métodos asincrónicos, es decir que requieren cierto tiempo de espera para obtener un resultado, por ejemplo, los métodos que hacen peticiones asincrónicas al servidor. El ejemplo siguiente se muestra una prueba al código del componente del Menú, la cual consiste en lo siguiente.

Al cargar la página se crea un componente de tipo dijitx.TreeMenu, que no tiene ningún menú hijo todavía, por lo tanto se prueba que la longitud de los menús hijos sea '0', luego se prueba que el componente carga cuatro menús del servidor mediante una petición asincrónica usando Ajax, para verificar esto se prueba la longitud de los menús hijos nuevamente, esta vez debe ser igual a 4 y finalmente se prueba que al colapsar el menú principal, ninguno de sus menús hijos se encuentra seleccionado.

Fragmento de código JavaScript

```
//Registra un grupo de pruebas con el nombre de 'prueba' la cual tiene 3 pruebas
// 'getChildren', 'setMenuFromStore' y 'collapseAll'
doh.register("prueba", [{
    name: "getChildren",
    runTest: function(prueba){
        var m = dijit.byId("menu1");
            //Se prueba igualdad
        prueba.assertEqual(0, m.getChildren().length);
    }
}, {
    name: "setMenuFromStore",
    runTest: function(prueba){
        var m = dijit.byId("menu1");
        m.setMenuFromStore(store);
            //Usando un objeto doh.Deferred se pueden probar métodos asincrónicos
        var d = new doh.Deferred();
        setTimeout(function(){
            try {
                if (prueba.assertEqual(4, m.getChildren().length)) {
                    d.callback(true);
                }
                else {
                    d.errback(new Error("Hay un error en la prueba"));
                }
            }
            catch (e) {
```

```
                d.errback(e);
            }
        }, 100);
        return d;
    }
}, {
    name: "collapseAll",
    runTest: function(prueba){
        var m = dijit.byId("menu1");
        m.collapseAll();
        m.getChildren().forEach(function(child){
                  //Se prueba que es 'false' la selección
                  //de cada unos de los menús hijos
            prueba.assertFalse(child.isSelected());
        });
    }
}]);
//Se ejecutan las pruebas registradas anteriormente
doh.run();
```

Luego DOH muestra un sumario con el resultado de las pruebas, mostrando cuáles pruebas pasaron, cuáles fallaron debido a un error o porque no se obtuvo el resultado esperado. Por ejemplo como muestra la figura, este es el resultado de la prueba anterior al código del menú.

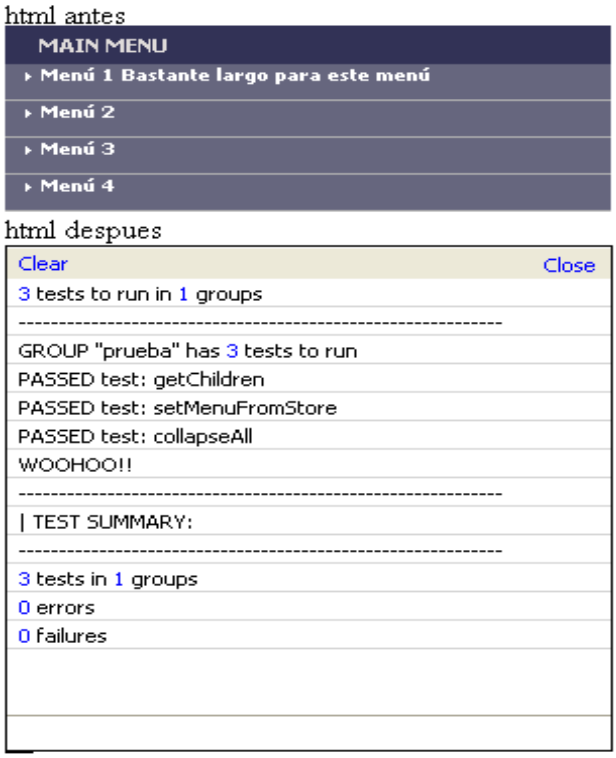


Fig5 Resultado de las pruebas realizadas al código del menú.

De está forma se puede probar el código de los componentes de la Librería, estas pruebas también se pueden integrar junto con las pruebas manuales en la interfaz antes mostrada, donde aparecen las pruebas manuales, lo que permite poder realizar pruebas manuales y al código desde la misma página.

CONCLUSIONES

En este capítulo se describió cómo desarrollar componentes utilizando la librería de Dojo Toolkit, se especificaron algunas de sus facilidades para el trabajo con JavaScript, se le realizaron las pruebas correspondientes a los componentes que integran la Librería con el objetivo de obtener un producto confiable en correspondencia con las funcionalidades antes previstas, también se generó la documentación con la intención de orientar al programador en su trabajo con la Librería.

CONCLUSIONES GENERALES:

Con la realización de la Librería se cumple con los objetivos propuestos de proveer a diseñadores y programadores de interfaz de usuario de componentes JavaScript que facilitarán la construcción de sus interfaces, proporcionando la implementación de un Menú vertical con estructura de árbol, un Calendario de eventos, un componente que maneja dos listas de elementos y un Wizard.

Provee además de un sitio de Ayuda que describe las características y funcionalidades de las clases que definen a los componentes con el objetivo de orientar el trabajo con la Librería.

Se creó un Api que permite generar documentación de clases a partir de los comentarios en los ficheros JavaScript.

Recomendaciones

- Adicionar funcionalidades a los componentes en función de las necesidades que se presenten.
- Identificar la falta de otros componentes e implementarlos en la Librería proporcionada.
- Estudiar la Librería de Dojo Toolkit con el objetivo de explotar todas sus utilidades.
- Publicar este trabajo como referencia para el trabajo con esta Librería.

GLOSARIO

Toolkit: Un Toolkit consiste en una biblioteca de utilidades, las cuales incorporan componentes de interfaz de usuario (IU) o widgets.

Dojo Toolkit: Es un toolkit open source DHTML escrito en JavaScript destinado a facilitar el rápido desarrollo de JavaScript o de las aplicaciones basadas en Ajax y sitios web.

Widget: Componente gráfico con el cual el usuario interactúa ejemplo, una ventana, una barra de tareas o una caja de texto.

IDE: Entorno de desarrollo integrado, pueden ser aplicaciones por si solas o pueden ser parte de aplicaciones existentes.

Plugin: Aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica y es ejecutada por la aplicación principal.

Eclipse: Escrito en Java aunque su principal uso es como IDE para Java, este es un lenguaje neutral. El soporte para desarrollo en Java es proveído por un componente enchufado o plugin, pero además están disponibles plugins para otros lenguajes, como C/C++, Cobol, C#, etc.

Script: Cada uno de los programas, aplicaciones o trozos de código que se compilan en tiempo de ejecución linealmente.

Web 2.0: Corresponde a una segunda generación de comunidades basadas en la Web y de servicios residentes en ella; buscan facilitar la creatividad, la colaboración y dan la posibilidad de compartir contenidos y otros recursos entre usuarios.

XML: Por sus siglas en inglés de Extensible Markup Language («lenguaje de marcas extensible»). Propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo, etc.

API: Del inglés **Application Programming Interface - Interfaz de Programación de Aplicaciones**) es el conjunto de funciones y procedimientos que ofrece cierta librería para ser utilizado por otro software como una capa de abstracción.

BSD: Berkeley Software Distribution.

Json: Javascript Object Notation, formato que representa a los objetos de Javascript.

Bibliografía

1. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *El proceso unificado de desarrollo de software.* Madrid : Addison Wesley, 2000.

2. **Alvarez, Miguel Angel.** *desarrolloweb.com.* [En línea] http://www.desarrolloweb.com/articulos/391.php.

3. pixelco.us. [En línea] [Citado el: 25 de Enero de 2008.] http://pixelco.us/blog/recopilacion-de-frameworks-bibliotecas-y-otros-recursos-para-desarrollo-web/.

4. [En línea] 26 de Enero de 2008. (http://www.fismat.umich.mx/~crivera/tesis/node20.html).

5. ServiceRules. [En línea] Enero de 2008. http://www.servicerules.com.ar/articles.do?pageId=ajax.

6. *The Dojo Toolkit.* [En línea] [Citado el: 28 de enero de 2007.] http://dojotoolkit.org/.

7. *Trio Solutions.* [En línea] 2008. http://componentes.developers4web.com/calendario.

8. **Gallardo, David, y otros.** *Eclipse in Action: A Guide for Java Developers .* 2003.

9. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *El proceso unificado de desarrollo de software.* Madrid : s.n., 2000.

10. **Monteiro Lazaro, Juliana.** [En línea] 23 de enero de 2008. http://www.desarrolloweb.com/articulos/26.php.

11. **Vilalta, Josep.** *Diagramas de Clases.* 2006.

12. **Schmuller, Joseph.** *APRENDIENDO UML EN 24 HORAS.* México : PEARSON EDUCACION, 2000. ISBN: 968-444-463-X.

13. [En línea] http://google.dirson.com/o.a/google-calendar.

14. **Loughran, Steve y Hatcher, Erik.** *Ant in Action.* s.l. : Manning Publications Co. 1-932394-80-X.

15. The Book of Dojo | Dojo Toolkit. [En línea] 2008. http://dojotoolkit.org/book/dojo-book-1-0.

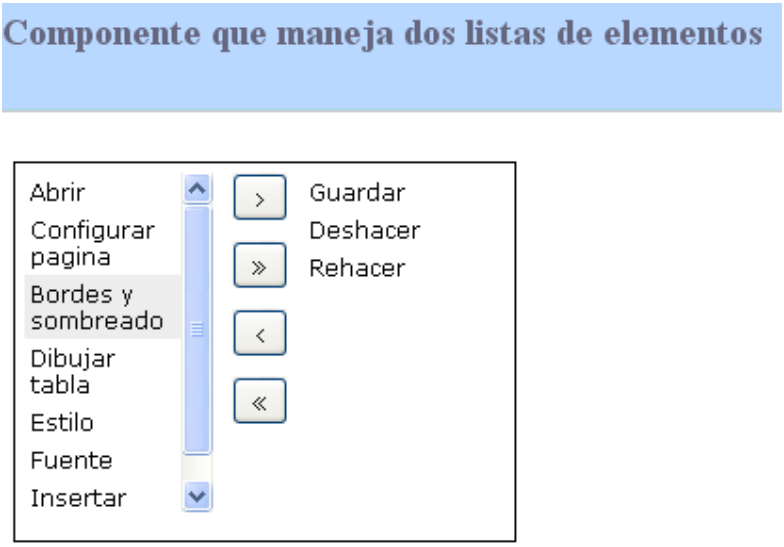
16. Internet Calendaring and Scheduling Core Object Specification(iCalendar). [En línea] 1998. http://tools.ietf.org/html/rfc2445.

17. **Zakas, Nicholas C.** *Professional JavaScript™ for Web Developers.* s.l. : Wiley Publishing, Inc., Indianapolis, Indiana, 2005. 978-0-7645-7908-0.

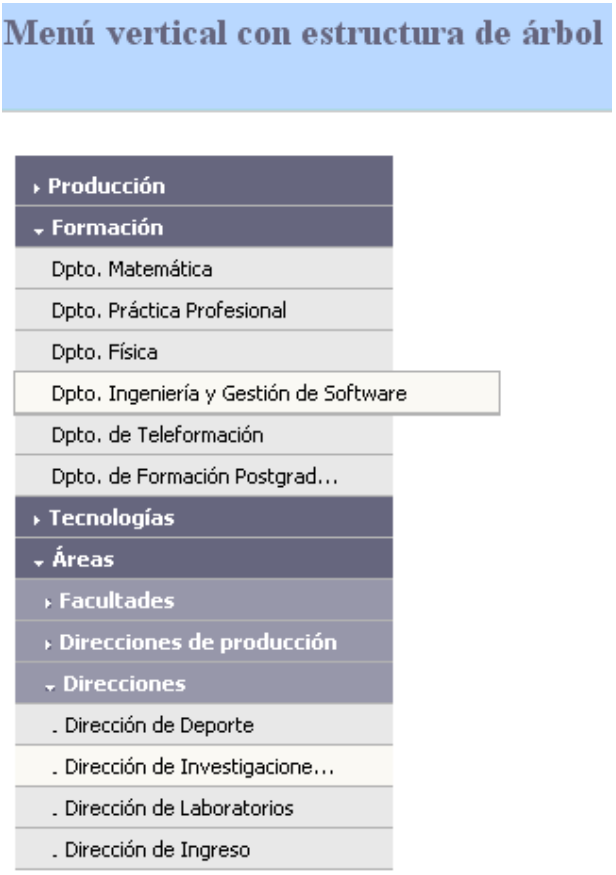
18. **Goodman, Danny, Morrison, Michael y Eich, Brendan.** *JavaScript™ Bible.* s.l. : Wiley Publishing, Inc., Indianapolis, Indiana, 2004. 0-7645-5743-2.

19. **Boggs, Wendy y Boggs, Michael.** *UML with Rational Rose 2002.*

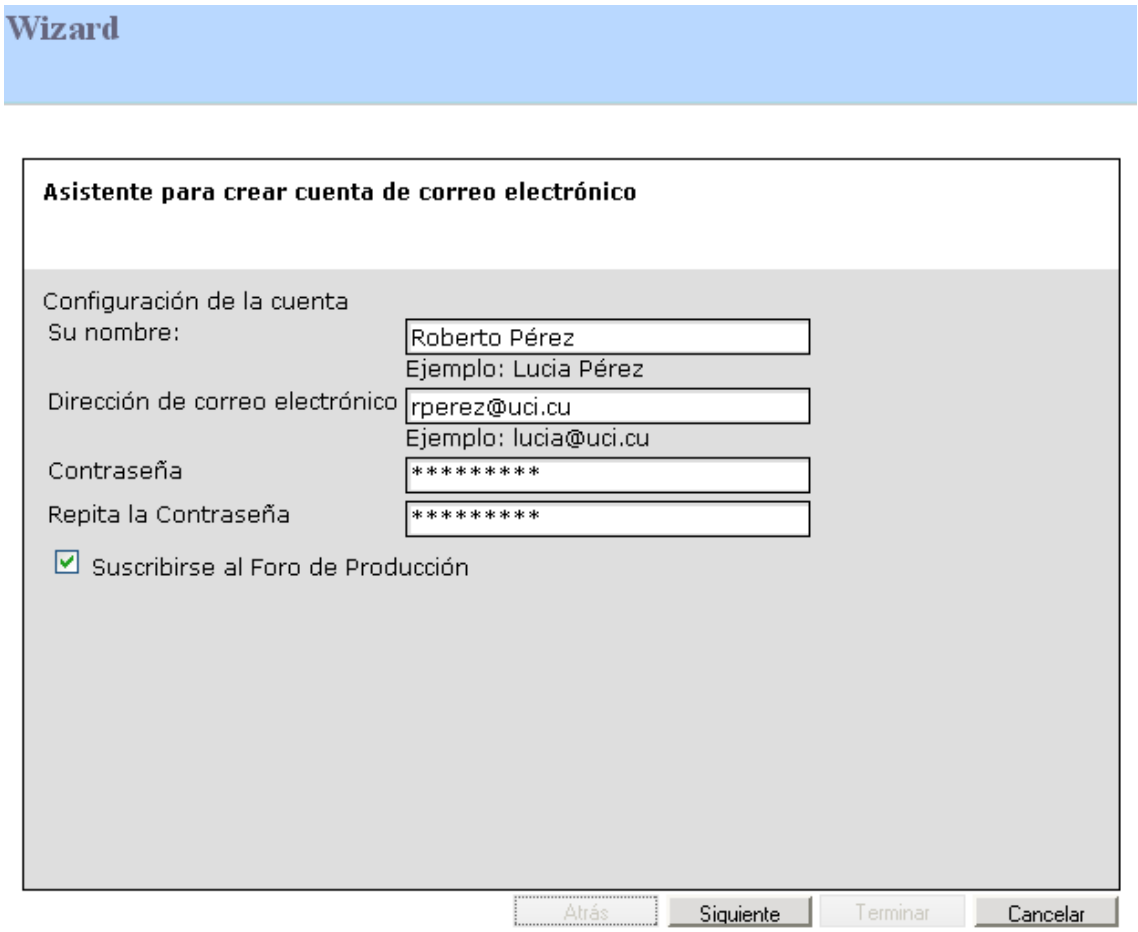
Anexos



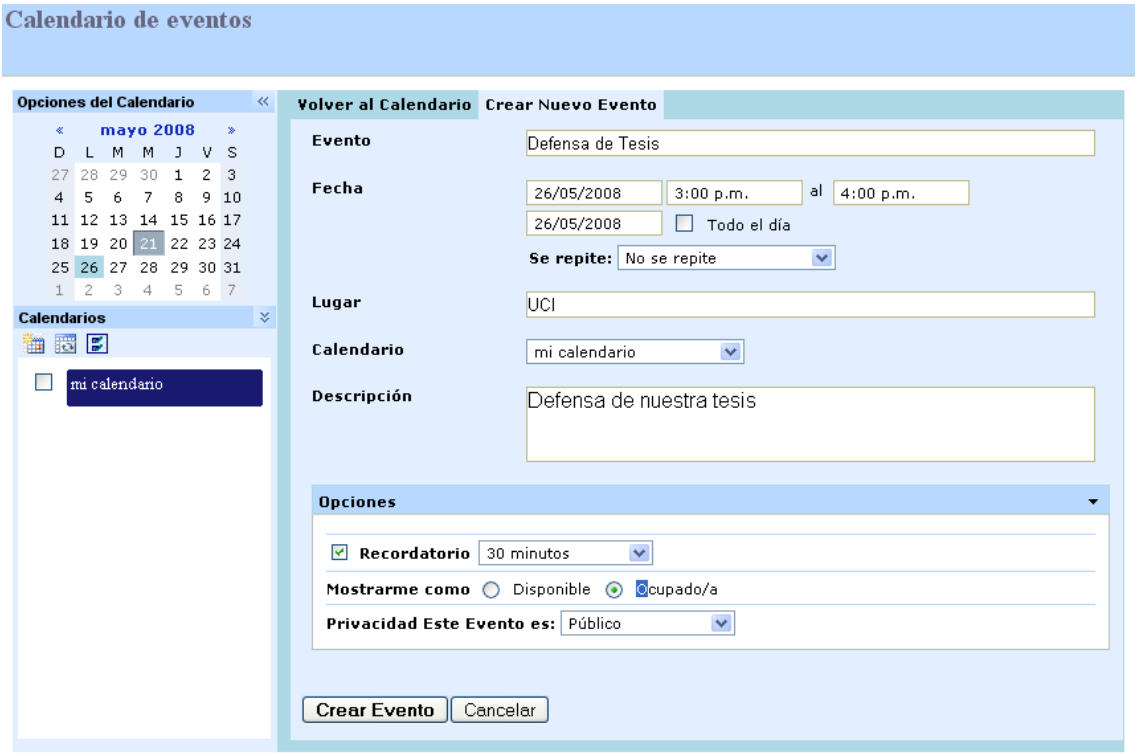
Anexo 1 Componentes que maneja dos listas de elementos.



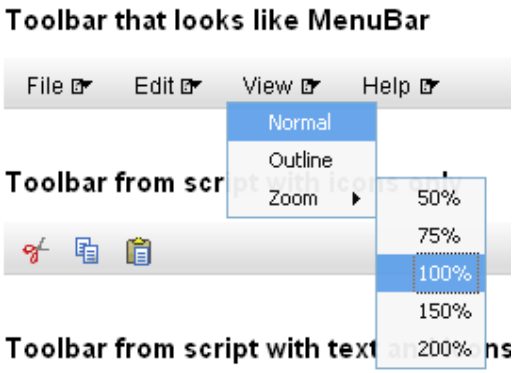
Anexo 2 Menú vertical con estructura de árbol.



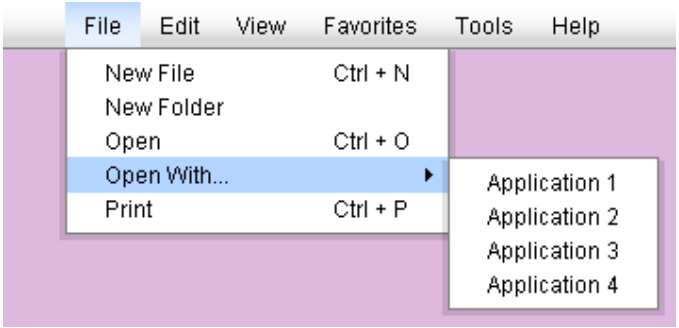
Anexo 3 Wizard



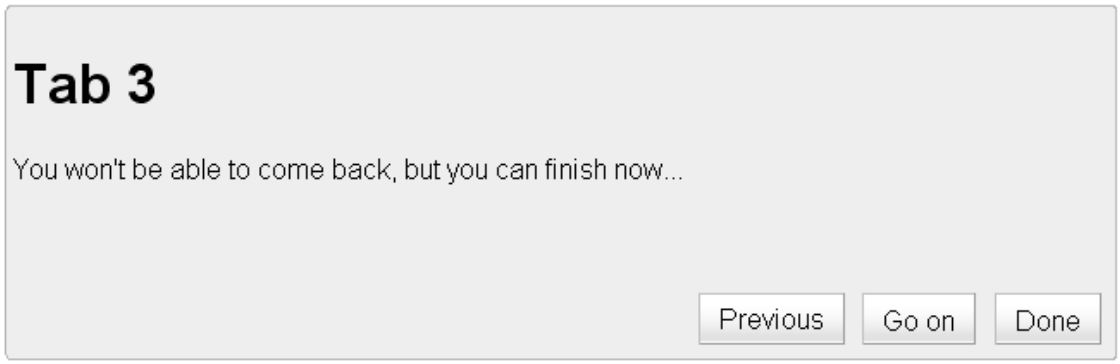
Anexo 4 Calendario de eventos.



Anexo 5 Menú que brinda Dojo Toolkit 1.1.



Anexo 6 Menú que brinda la Librería de YUI.



Anexo 7 Wizard que brinda Dojo Toolkit 1.1.