UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

Editor de mapas mentales online con HTML5 y Javascript

Realizado por

José Luis Molina Soria

Dirigido por

Juan Antonio Falgueras Cano

Departamento

Lenguajes y Ciencias de la Computación

Málaga, Noviembre de 2013

UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

Reunido el tribunal examinador en el día de la fecha, constituido por:

Presidente/a D^o/D^a. _____

Secretario/a D^o/D^a. _____

Vocal D^o/D^a. _____

para juzgar el proyecto Fin de Carrera titulado: _____

_____

_____

del alumno/a D^o/D^a : _____

dirigido por D^o/D^a : _____ ,

y, en su caso, dirigido académicamente por D^o/D^a : _____

_____

ACORDÓ POR _____ OTORGAR LA CALIFICACIÓN

DE _____

Y PARA QUE CONSTE, SE EXTIENDE FIRMADA POR LOS COMPARECIENTES DEL TRIBUNAL, LA PRESENTE DILIGENCIA

Málaga a ___ de _____ de 20___

El/La Presidente/a | El/La Secretario/a | El/La Vocal

Fdo: | Fdo: | Fdo:

Agradecimientos

A mi familia y padres por su paciencia

Tabla de contenidos

Índice de figuras

Capítulo

1

Abstract

El editor de mapas mentales on-line es un sistema web desarrollado única y exclusivamente en HTML5 para diseñar y elaborar mapas mentales con formato FreeMind. Formato el cual, es considerado un estándar en el mundo de los mapas mentales.

La idea que subyace en la realización de este editor de mapas mentales, es probar las nuevas tecnologías existentes alrededor de HTML5 y comprobar el estado actual de dicha tecnología tras el interés y la relevancia que está adquiriendo en estos últimos años.

La metodología Ágil, utilizada para llevar a cabo el proyecto, se adapta muy bien al desarrollo web. Permitiendo un feedback constante desde la versión inicial hasta la actual. Conjuntamente con la metodología Ágil se utilizado otras metodologías y paradigmas de programación como el BBD[1], patrones de diseño, etc ... También se ha hecho uso de tecnologías ya existentes como soporte al desarrollo, entre ellas destacar KineticJS, Mocha, GruntJS, NodeJS, JSHint, JSDoc y GitHub. Todas estas tecnologías han propiciado una experiencia de desarrollo satisfactoria.

En resumen, se ha podido constatar un gran avance en HTML5 con respecto a la versión anterior. A pesar de ello, sigue existiendo, aunque en mucho menor grado, una gran dependencia con el navegador.

[1] Como sistema de pruebas y verificación del código fuente

Capítulo

2 Motivación y objetivos

2.1. Motivación

Desde hace ya unos años, estamos viviendo una revolución en el desarrollo web, que a provocado un cambio en nuestro estilo de vida, la forma de comunicarnos, en los flujos de información e incluso en nuestras relaciones diarias. HTML[1] y JavaScript son una parte importante de esta revolución, y es por ello, que decidí dar el paso y crear una aplicación que funcionará única y exclusivamente en el navegador (sin necesidad de un servidor).

Estamos viendo como día a día aplicaciones que han sido por antonomasia nativas (editores de texto, hojas de cálculo, aplicaciones de gestión, juegos ...), están siendo implementadas con tecnologías web con gran éxito. Los editores de mapas mentales también han dado ese paso existen aplicaciones como text2mindmap[2], MindMeister[3], etc., tan versátiles o más que las propias aplicaciones nativas.

Hace ya tiempo que vio la luz los primeros borradores de HTML5[4] (ver figura 2.1) pero su implantación está siendo lenta, no sólo por parte de la comunidad de desarrolladores y diseñadores, sino también por parte de los navegadores.

[1] Hypertext Markup Language
[2] http://www.text2mindmap.com/
[3] http://www.mindmeister.com/es
[4] El primer borrador de HTML5 fue publicado en 2008

HTML5 ha tomado en cuenta los defectos de su versión anterior[5] y mejorar otras características como:

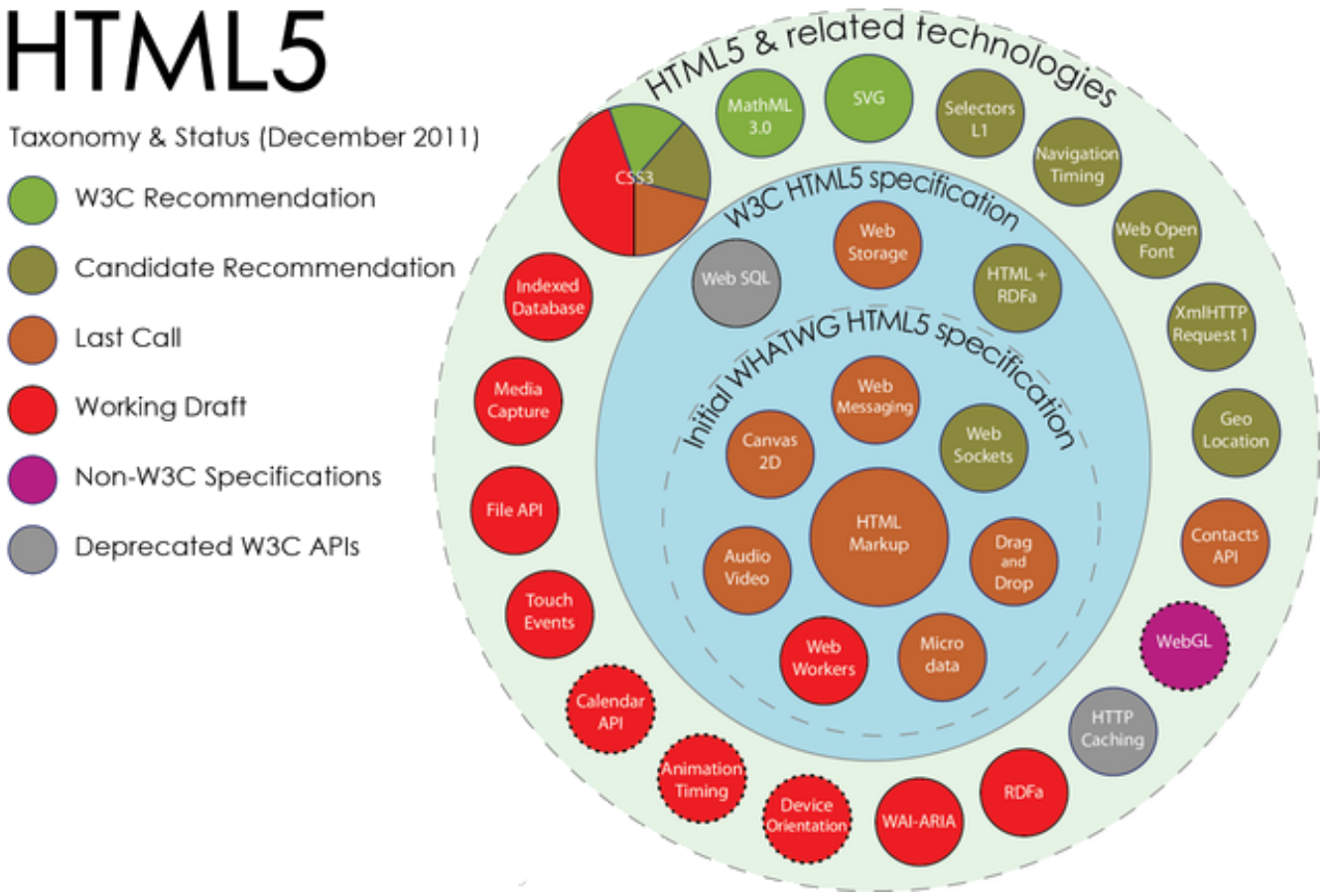

Figura 2.1: Esquema general de la especificación HTML5 a diciembre de 2011

- **Nuevas etiquetas estructurales.** Se ha incorporado un nuevo conjunto de etiquetas pensadas para definir mejor la estructura de una web, entre las más importantes están las de encabezado, barra de navegación, secciones y pie.
- **Manejo de imágenes.** En la versión anterior podía incorporar imágenes ahora, además podemos modificarlas e interactuar con ellas. También disponemos un una etiqueta y un API completo para manejo de canvas[6].
- **Etiquetas de vídeo y audio.** Sin incluir flash ni aplicaciones externas podemos incorporar un reproductor de vídeo y/o audio.
- **Mejora en la semántica web.** HTML5 incluye elementos que permiten dar

[5] HTML 4.01
[6] En principio, sólo en 2D.

información de la página web a los buscadores para obtener resultados adaptados a las necesidades del usuario.

- **Soporte móvil/tabletas.** Mejoras en las hojas de estilos, nuevos manejadores para evento touch, etc.
- **Acceso a ficheros.** Incorpora un API para lectura/escritura de ficheros.

La motivación no puede ser otra que profundizar en las características de HTML5 y aprender de esta tecnología.

2.2. Objetivos

El principal objetivo de este proyecto es la creación de un editor de mapas mentales online. El editor, frontend, debe ejecutarse completamente en el cliente. Para ello, vamos a utilizar como lienzo de dibujos el canvas de HTML5 y Javascript como lenguaje de desarrollo.

El usuario podrá navegar por el diagrama con los cursores partiendo desde la idea central. Interactuará con el diagrama de forma que, dependiendo del nodo en el que se encuentre y la acción que realice podrá insertar, modificar, anotar, plegar, etc...

Esta fuerte interacción, provoca que dentro de los objetivos del proyecto, se encuentre la elaboración de una extensa librería JavaScript, bien estructurada y testeada.

En todo momento, y en pos de una aplicación lo más estándar posible, se seguirá las especificaciones de la World Wide Web Consortium[7] (W3C) y la especificación

Como objetivo principal está pues, la universalidad, independencia de sistemas y la inmediatez de uso, sin instalación, siempre actualizada, e incluso la posibilidad de uso en forma local con cualquier navegador actual que sigue el estándar HTML5. Entre las posibles plataformas de uso se tratará de incluir las plataformas táctiles, especialmente los tablets.

[7] Web oficial de la W3C http://www.w3.org/

Capítulo

3 Introducción

3.1. ¿Qué es?

Los mapa mentales son un método efectivamente sencillo de asimilar y memoriza información a través de la representación visual de la información.

Por naturaleza, nuestro celebro tiene un potencial ilimitado y que, en muchas ocasiones es desaprovechado o difícil de interpretar. Tenemos dos hemisferios el izquierdo y el derecho, el racional y el creativo, ambos funcionan de forma separada. Los mapas mentales consiguen relacionar ambos hemisferios (racional y creativo) y lograr que funcionen conjuntamente.

Toda persona tiene una forma natural de elaborar sus propias ideas, mediante pensamiento irradiante[1] . El pensamiento irradiante refleja mediante la asociación de ideas nuestros pensamientos y conocimientos sobre una materia concreta. A esta forma de pensamiento podemos acceder mediante los mapas mentales, que irradian y asocian ideas a partir de un concepto central.

[1] que irradia

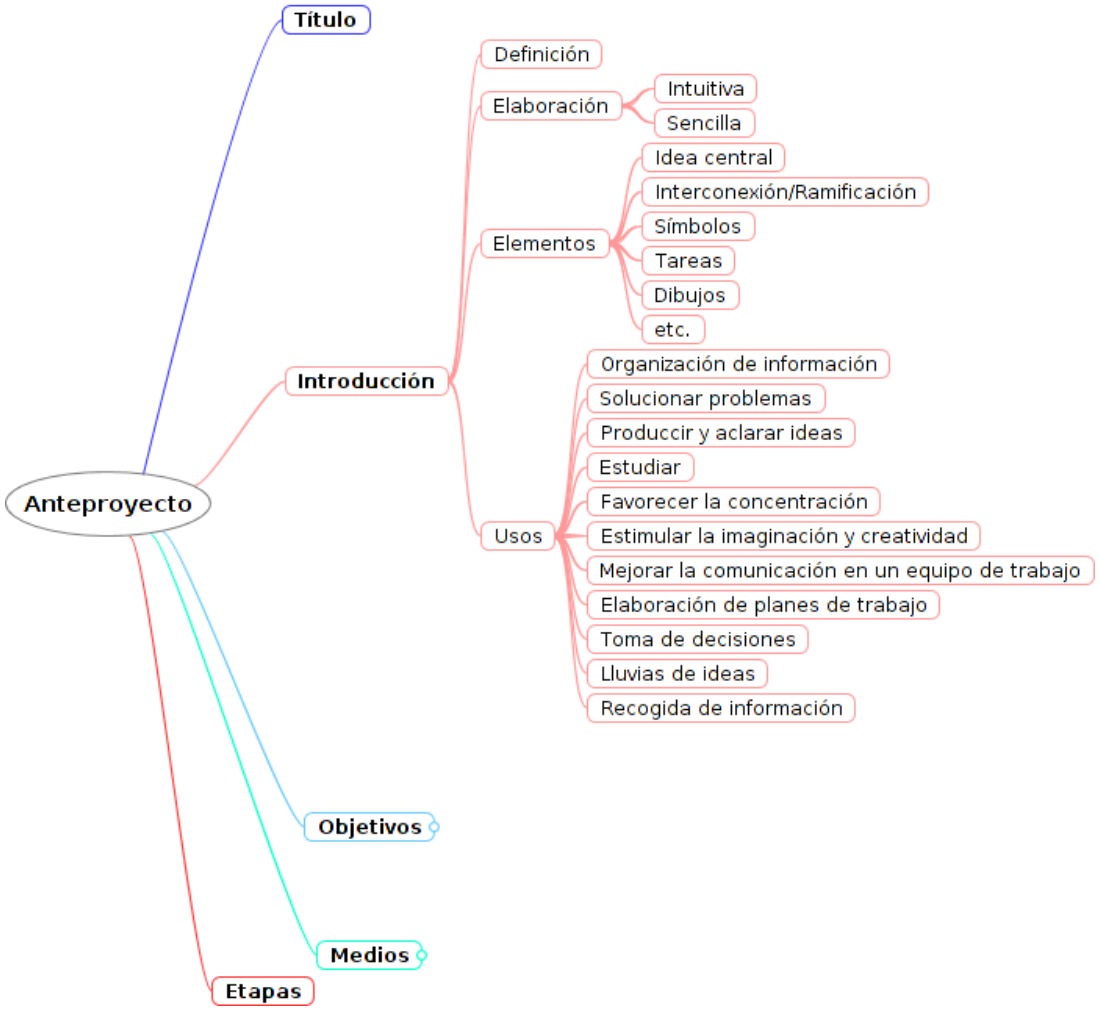

Figura 3.1: Mapa mental de FreeMind

3.2. Aplicaciones y beneficios.

Los campos de aplicación y los beneficios de los mapas mentales son muchos y muy diversos. Entre los más destacados tenemos:

- Estimular la memoria, imaginación y creatividad.
- Organizar información.
- Concentrarnos en la resolución de un problema.
- Tomar notas y apuntes.
- Producir y aclarar ideas o conceptos.
- Visualizar escenarios complejos.
- Consolidar procesos de estudios y aprendizaje.
- Favorecer la concentración.
- Proyectos. Organizar el proyecto y priorizar el plan de trabajo.
- Mejorar la comunicación en un equipo de trabajo.
- Preparar y dirigir una reunión.
- Toma de decisiones.
- Lluvias de ideas.
- Recogida de información.
- Expresar ideas complejas y difíciles de redactar.
- Diseñar el contenido de un escrito o informe.
- Preparar una presentación en público.
- Elaboración de sitios webs.

3.3. Partes de un mapa mental.

3.3.1. Según su estructura.

Un mapa mental tiene las siguientes estructuras esenciales (figura 3.2):

1. Idea central.
2. Aristas. Establece una asociación de ideas.
3. Nodo. Ideas segundarías o asociada a otra idea.

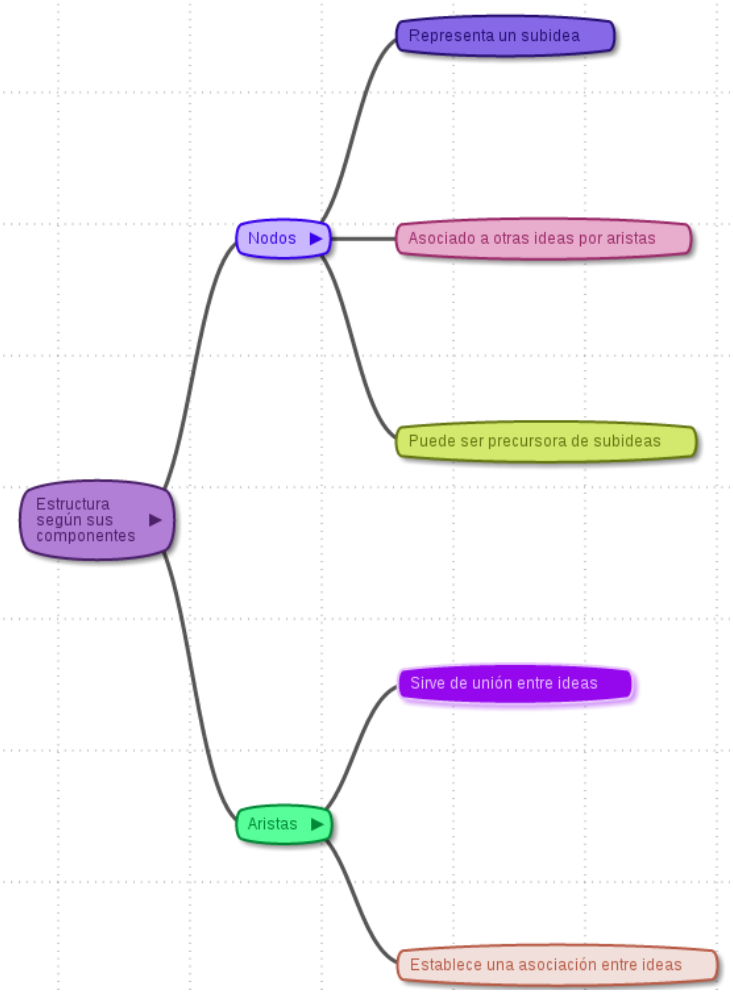

Figura 3.2: Partes de un mapa mental

3.3.2. Por contenido.

Un mapa mental podemos estructurarlo según su contenido (figura 3.3).

1. Idea central

2. Los temas principales del asunto irradian de la imagen central como ramas.

3. Cada rama contiene una imagen o una palabra clave asociada.

4. Las ramas forman una estructura nodal conectada.

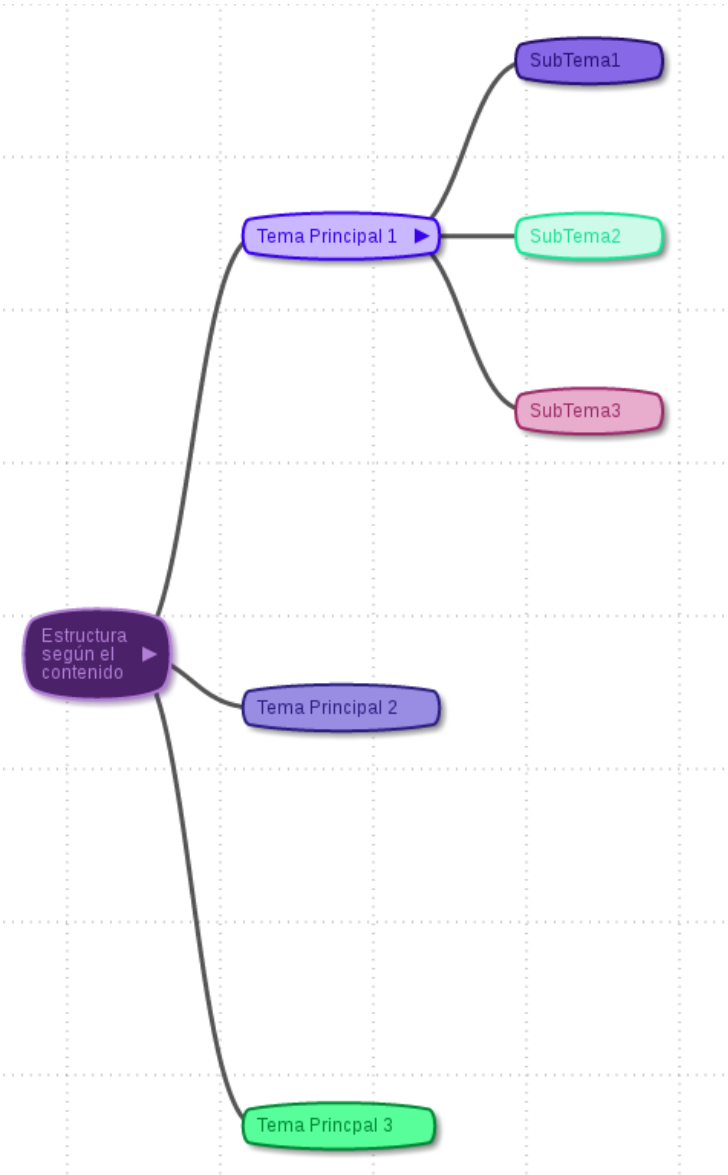

Figura 3.3: Partes de un mapa mental considerando su contenido

3.4. Elaboración.

La elaboración de un mapa mental es un gesto sencillo y casi intuitivo, sólo necesitamos partir de una idea central, de la cual vamos ramificando asociando o interconectando símbolos, palabras, tareas o dibujos.

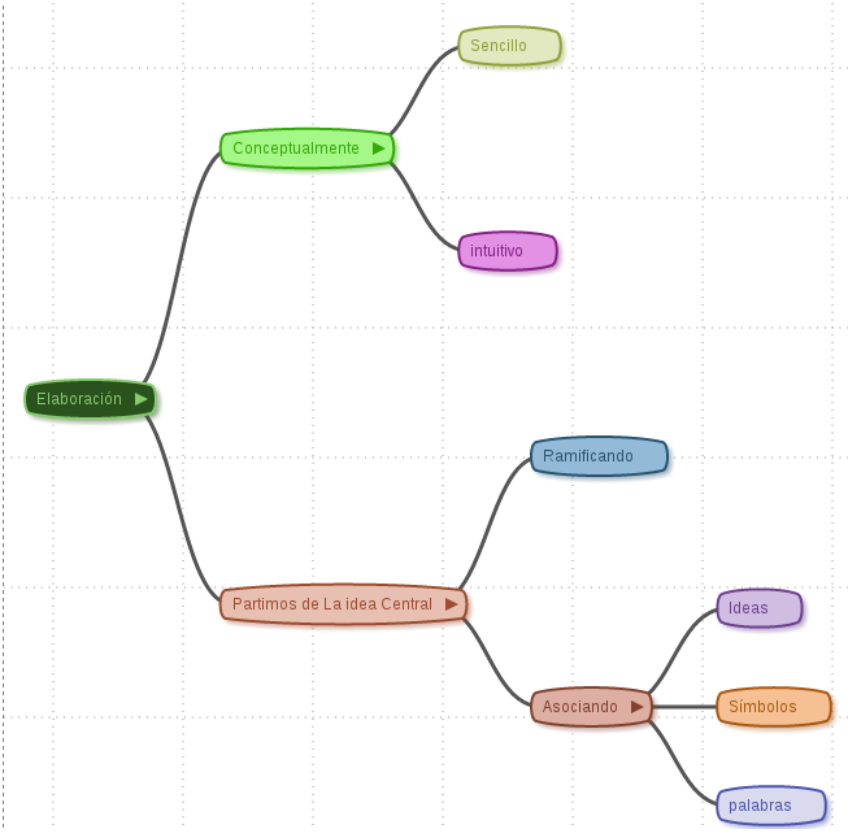

Figura 3.4: Mapa mental de elaboración de mapas mentales

En definitiva, se trata de un diagrama radial que permite a una persona, o grupo de ellas, plasmar su percepción sobre un tema, o idea, mediante la asociación de conceptos palabras y/o imágenes.

Capítulo

4 Materiales y métodos

4.1. Metodología y etapas del desarrollo.

4.1.1. Metodología de desarrollo ágil.

En 2001, de un reunión celebrada en EEUU por 17 expertos en la industria del software nace el término "ágil" aplicado al desarrollo de software. El propósito de estos expertos era la elaboración de un manifiesto y principios que permiten a los equipos de desarrollar software rápidamente y responder a los cambios que surjan a lo largo del proyecto. The Agile Alliance, organización surgida de esta reunión, se dedica a promover los conceptos relacionados con el desarrollo ágil y cómo punto de partida tiene un manifiesto con los siguientes 4 puntos:

- Individuos e interacciones sobre procesos y herramientas
- Software funcionando sobre documentación extensiva
- Colaboración con el cliente sobre negociación contractual
- Respuesta ante el cambio sobre seguir un plan

Quizás estemos ante una de las metodologías de desarrollo más importantes del momento. Se trata de un modelo desarrollo iterativo e incremental donde en cada iteración se elabora una nueva versión para usuario final.

El modelo de desarrollo Ágil tiene como principal objetivo la satisfacción del cliente y la elaboración de un software de calidad. Para ello, involucra al usuario en todas las etapas del desarrollo, aportando ideas y realizando pruebas de los productos de cada iteración. El usuario consigue así un software adaptado a sus necesidades, quedando completamente satisfecho del producto final. Con esta estrecha colaboración entre usuario final y el equipo de desarrollo se busca aunar esfuerzos en pos de un objetivo común.

En cada ciclo se pretende minimizar los riesgos. Es por ello que, para cada iteración se incorpora un conjunto reducidos de funcionalidades. Buscando, no sólo minimizar el riesgo intrínseco al desarrollo, sino que los ciclos de desarrollo sean cortos y se dinamice el proceso productivo. A este respecto, y según los principios de la metodología Ágil, es preferible una versión incompleta a una con errores.

Otro aspecto importante, es que la solución y los requerimientos evolucionan de forma continua. Provocando en ocasiones cambios profundos en los diseños preliminares, algo inconcebible en las metodologías clásicas. La refactorización de código se convierte en algo habitual y deseable, si ello nos lleva a una mejor solución.

Un ciclo de desarrollo en la metodología Ágil consta de la siguientes fases:

- Planificación.
- Análisis de requerimientos.
- Diseño.
- Codificación.
- Revisión.
- Documentación.

La metodología Ágil se adapta muy bien al desarrollo web. Por esto, y por las características que presenta este paradigma, el proyecto seguirá este modelo de desarrollo.

4.1.2. Etapas del desarrollo.

Viendo la dependencia entre librerías a implementar, lo más apropiado, es seguir un diseño ascendente (bottom-up). Siempre que el estadio anterior haya sido verificado y comprobado su completud, se podrá afrontar con éxito la siguiente etapa. Dicho de otra forma, cada etapa es dependiente de la etapa inmediatamente anterior. Siempre siguiendo la metodología ágil se ha decidido afrontar el proyecto en dos fases:

Primera fase: se encargará de llevar a buen término la implementación de las librerías Javascripts necesarias para la aplicación. En cada ciclo tendremos que realizar una planificación, análisis de riesgos, implementación, pruebas unitarias y documentación de cada librería. La primera fase constará pues, de seis ciclos bien definidos. El orden de los ciclos es el que sigue:

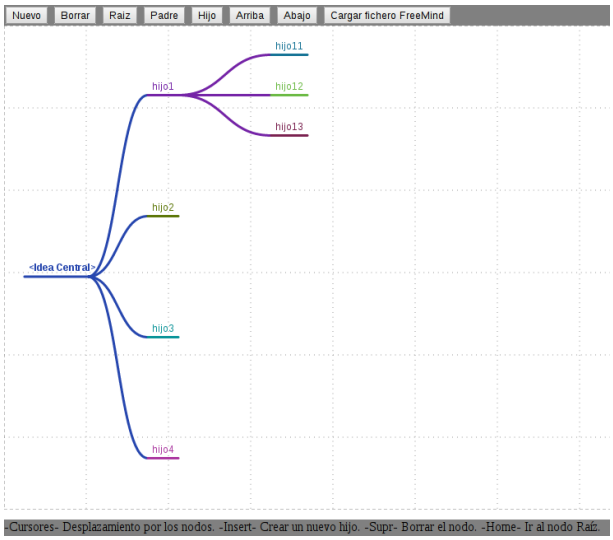

Figura 4.1: Versión inicial.

- **Librería base con soporte para herencia.** Esta librería debe tener toda la funcionalidad básica (bindings, curryings, etc) y debe estar muy optimizada ya que el perfecto funcionamiento de la aplicación dependerá en buena medida de ella.
- **Librería para manejo de árboles n-arios.**
- **Librería para el manejo de ficheros.** Será la encargada de manejar ficheros, a partir de ella, realizaremos las clases de exportación e importación de mapas mentales de la aplicación.

- **Librería gráfica.** Ciñéndonos al contexto 2D, necesitamos un wrapper sobre la librerías propias del canvas. Esta librería, nos debe permitir pintar, cada uno de los elementos de nuestro árbol. Además de configurar, atributos visuales tales como color del trazo, relleno, etc. No se descarta el uso de alguna librería estándar. Para ello, se realizará una pruebas de concepto sobre ellas.
- **Librería para el manejo de eventos del canvas.** El canvas debe reaccionar tanto al teclado, ratón y touch. El canvas viene desprovisto de eventos sobre los elementos pintados en él y es aquí donde entra en juego esta librería.
- Por último, las librerías propias del mapa y **prototipo** o primera versión.

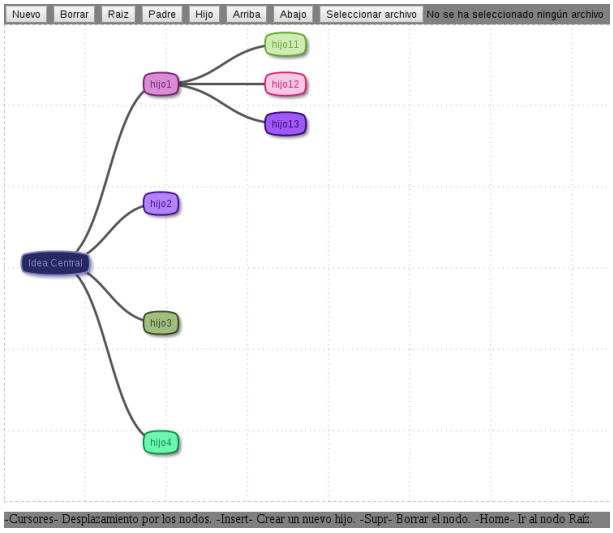

Figura 4.2: Versión inicial.

Segunda fase: una vez implementadas todas las librerías necesarias y una primera versión (inoperativa), nos encontramos en disposición de ir elaborando la aplicación. Revisión de aspectos visuales de la aplicación tales, como un editor ajustable, zoom, y mejoras en el funcionamiento en general.

Tercera fase: nuevas funcionalidades como plegado, hacer-deshacer y un mejor ajuste del en la redistribución de nodos y escalado.

4.2. Casos de uso.

En la figura 4.3 podemos ver de forma general el diagrama de casos de usos.

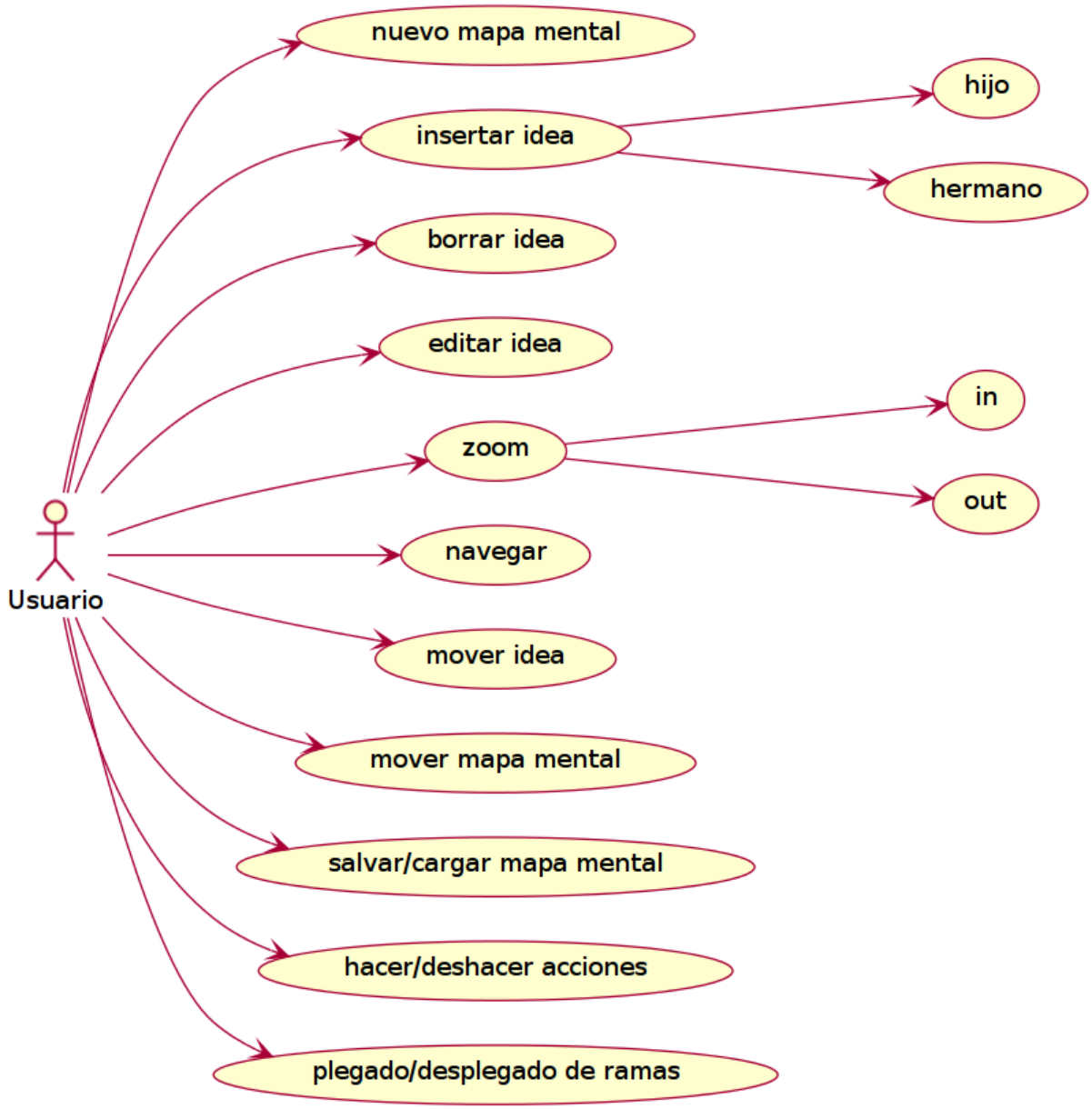

Figura 4.3: Casos de uso

4.2.1. Nuevo mapa mental

El usuario debe de poder reiniciar el editor y empezar un nuevo mapa en cualquier momento. El sistema deberá limpiar la zona de edición eliminando cualquier resto de ediciones anteriores. Una vez borrado se presentará una idea central por defecto que el usuario podrá modificar en todo momento.

Las acciones que desencadenará esta funcionalidad será un botón y/o la secuencia de teclas <Shift+n>.

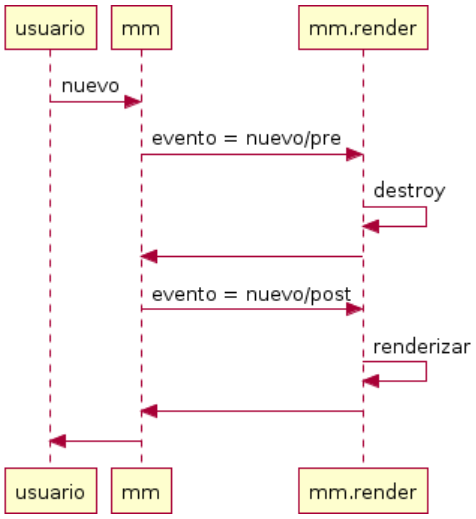

Figura 4.4: Diagrama de secuencia nuevo

4.2.2. Insertar idea

Mediante el uso de teclado o ratón el usuario podrá crear nuevas ideas. Esta nueva idea podrá ser tanto hija como hermana de la idea actualmente seleccionada. Debe quedar distribuida en función de los nodos existentes en el mapa mental.

La secuencia de teclados designadas para la creación de ideas. Son <ins> para ideas hijas y <Shift+Enter> para crear una idea hermana.

4.2.3. Borrar idea

Con el teclado (<supr>) y/o ratón el usuario siempre podrá eliminar un idea del mapa mental. Si existen otras ideas que dependan de la idea a borrar estas también se borrarán.

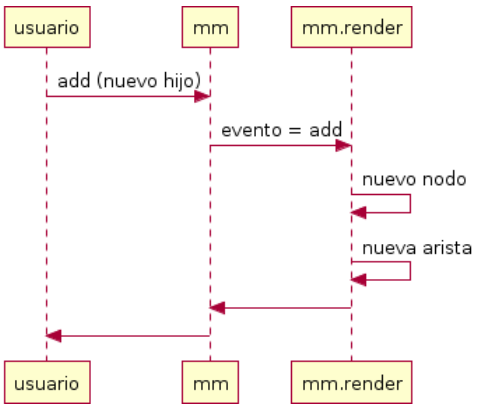

Figura 4.5: Diagrama de secuencia add

Los nodos se redistribuirán en función de los nodos restantes el mapa mental.

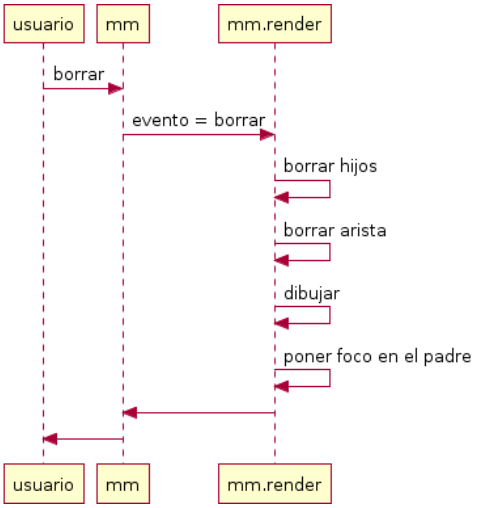

Figura 4.6: Diagrama de secuencia borrar

4.2.4. Editar idea

Toda idea será editable en cualquier momento. El usuario podrá activar el modo de edición y modificar el contenido. Se accederá al modo de edición cuando insertemos, naveguemos, o establezcamos el modo de edición.

Para entrar y salir del modo de edición se utilizarán las teclas <Enter> y <Esc> respectivamente. Una vez en modo de edición la secuencias de teclas de la aplicación se ajustarán para que <Enter> y <Tab> permita salir del modo de edición, con <Shift+Enter> se inserte un salto de línea y deshabilite el resto de atajos de teclado. El doble clic y doble touch permitirá entrar en modo de edición.

El editor deberá y ajustándose al tamaño del texto insertado.

4.2.5. Zoom

La aplicación permitirá acciones de zoom o cambio de escala a la imagen. Ampliar (<Ctrl++>), reducir (<Ctrl+->) y reiniciar (<Ctrl+0>) la escala. Con esta funcionalidad el usuario podrá ajustar las dimensiones del mapa mental a sus necesidades. La rueda del ratón es también una buena opción para realizar zoom in / out.

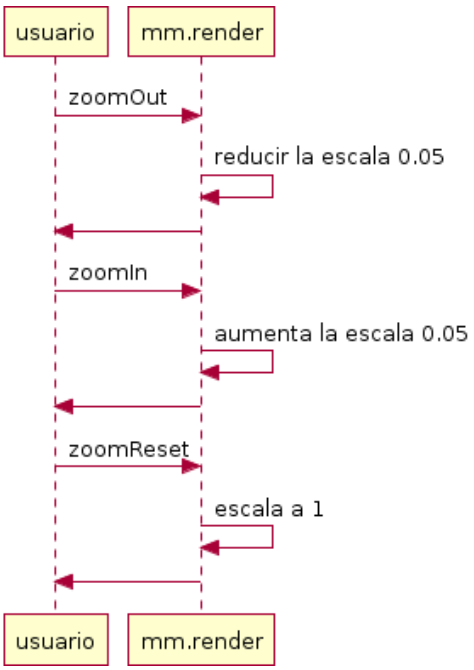

Figura 4.7: Diagrama de secuencia Zoom

4.2.6. Navegar

El usuario debe poder moverse por el mapa mental tanto por teclado como con el ratón o touch. El mapa siempre tendrá una idea activa, o focalizada, que podrá variarse mediante un clic, touch o las siguientes secuencias de tecla:

- Para ir a la **idea central** <home>
- Para ir a la **idea padre** de la idea actual <left>
- Para ir a la **idea hija** <right>

- Para ir a una **idea hermana** <up> y <down>
- Para **navegar por niveles** podemos utilizar <tab>

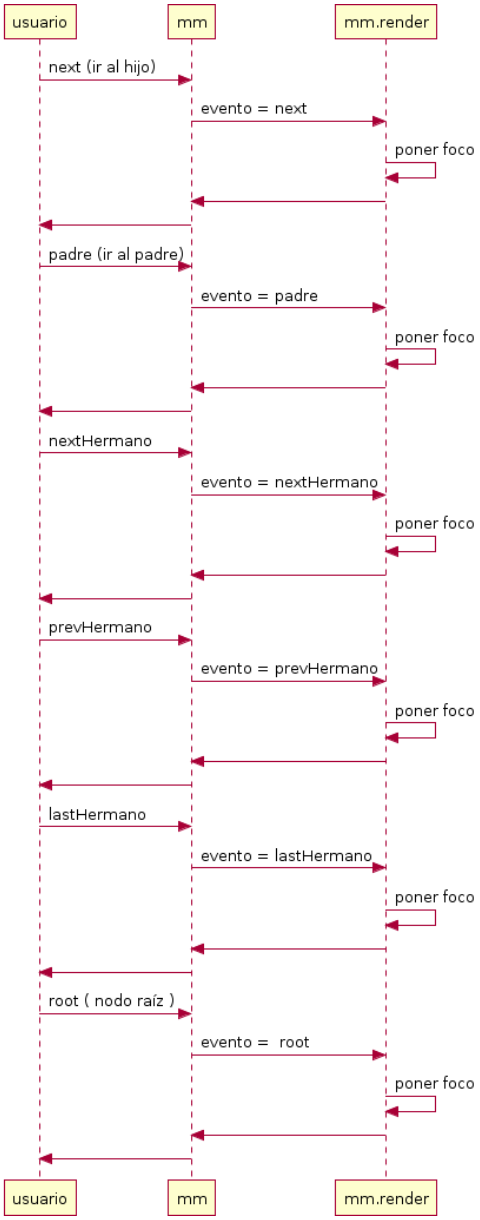

Figura 4.8: Diagrama de secuencia navegación

4.2.7. Mover idea

Con el ratón y touch podremos ajustar la posición de los nodos.

4.2.8. Mover mapa mental

Con el ratón y touch podremos desplazar el mapa.

4.2.9. Salvar/cargar mapa mental

El usuario siempre tendrá opción de salvar y cargar mapas mentales en formato FreeMind.

4.2.10. Hacer/deshacer acciones

El sistema dispondrá de opciones típicas de edición como hacer y deshacer.

4.2.11. Plegado/desplegado de ramas

Las distintas ideas se podrá plegar o desplegar para una mejor visualización. El sistema deberá ajustar las posiciones de las ideas visibles (que no estén plegada) al campo de visión siempre que sea posible.

Para mayor agilidad el programa dispondrá de una secuencia de teclado para plegar (<Shift+->) y desplegar (<Shift++>) además de botones.

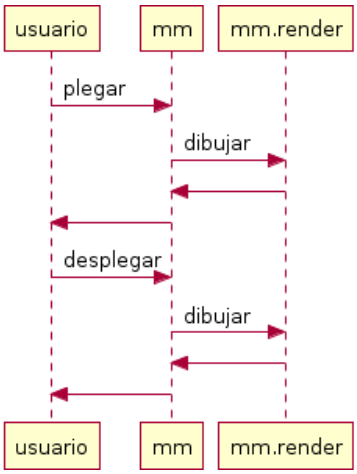

Figura 4.9: Diagrama de secuencia plegar

4.3. Diagramas de Clase

Los diagramas de clases están ordenados por importancia y bloque funcional, siguiendo una perspectiva bottom-up siempre que sea posible.

Clase MM.Class

Como centro de todo el sistema de clases implementado está el MM.Class. Una abstracción del patrón constructor que es eje de todas las clases implementadas en la aplicación. A partir de ahora, cuando hable de clase me refiero a la herencia efectuada con MM.Class.

La implementación de este objeto es fundamental ya que Javascript es un lenguaje orientado a objetos puro y libre de clases.

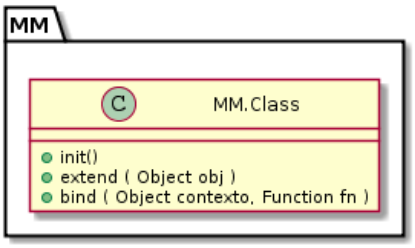

Figura 4.10: Clase MM.Class

Los principales métodos son:

- **MM.Class.extend:** método que nos permite extender sobre una clase existente.
- **MM.Class.init:** Constructor para las clases.

Cualquier método sobrescrito dispone en su clase una propiedad _super que hace referencia al método sobrescrito, de forma podamos realizar una llamada al super (o padre).

4.3.1. Diagrama de clases PubSub

Como núcleo en la comunicación, entre clases y distintos bloques funcionales, están los eventos. Para ello, se ha desarrollado la clase MM.PubSub que implementa el patrón Publicador-Suscriptor[1].

El concepto es sencillo, el objeto suscriptor se suscribe a un evento o mensaje concreto y el publicador anuncia a todos los suscriptores cuando está lista las suscripción. El símil más utilizado, y directo, es el de los suscriptores de un periódico, el cual un lector (o

[1] También conocido como patrón Observador-observable

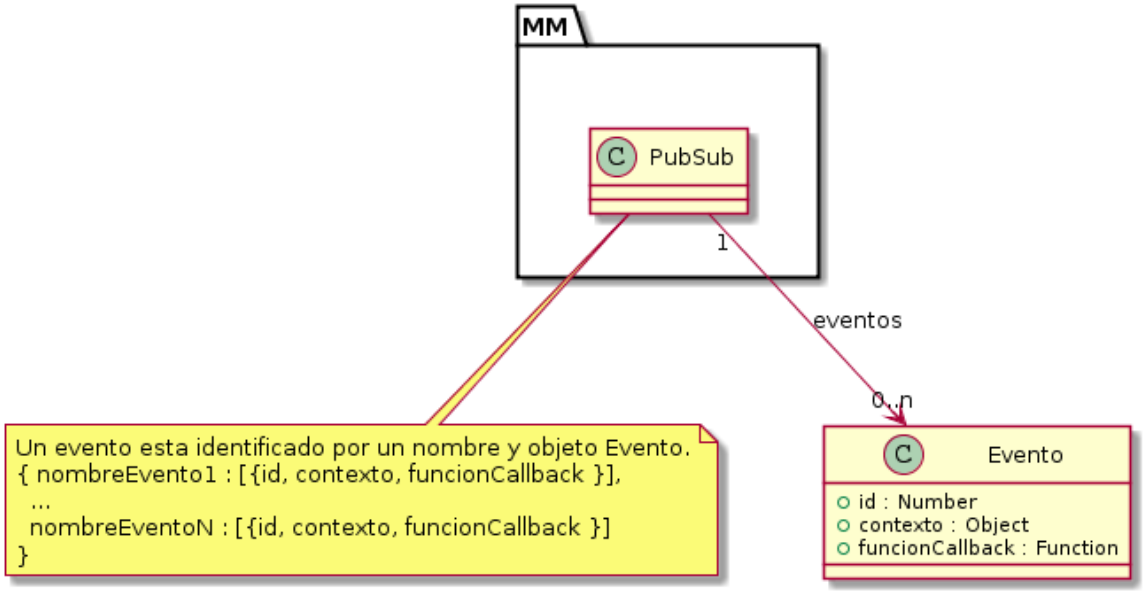

Figura 4.11: Diagrama de clases pubsub

suscriptor) paga un precio para recibir el periódico y la editorial (o publicador) le envía un ejemplar cuando lo tiene disponible.

Los eventos suscritos se registran con un nombre en una lista, que contiene un identificador de suscripción, el contexto de ejecución y la función a ejecutar en el momento de la publicación del evento.

MM.PubSub

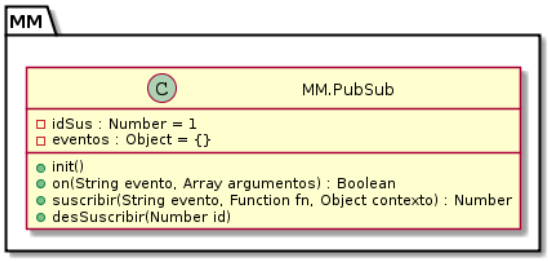

Figura 4.12: Clase MM.PubSub

Métodos:

- **MM.PubSub.suscribir:** permite a los suscriptores la suscripción a un evento o publicación.

- **MM.PubSub.desSuscribir:** permite a los suscriptores la de suscripción a un evento o publicación.
- **MM.PubSub.on:** método que permite al publicador notificar a los suscriptores la ocurrencia de un evento.

4.3.2. Diagrama de clases MM.UndoManager

Dentro de la edición, otro punto importante son las funciones de hacer y deshacer. Para ello, se ha implementado un manejador que se encarga de registrar, en una lista de comandos, los cambios realizados en el editor.

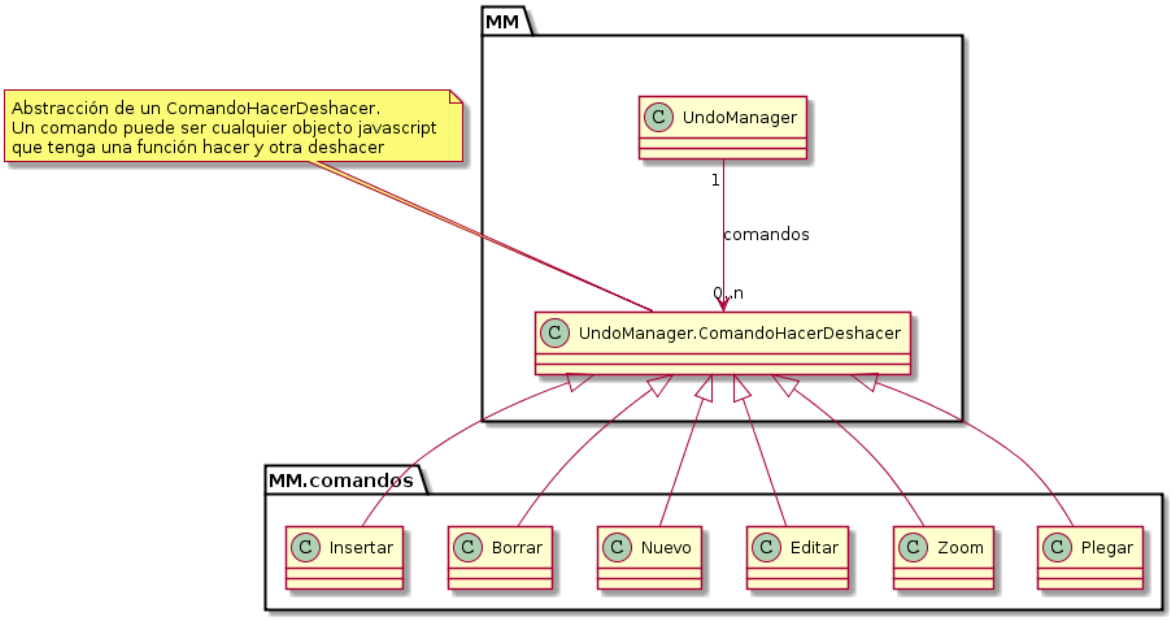

Figura 4.13: Diagrama de clases undo

Clase MM.UndoManager.ComandoHacerDeshacer

La clase MM.UndoMangerComandoHacerDeshacer es la clase base para todos los comandos para hacer y deshacer del editor de mapas mentales. De ella, como se puede observar en la figura 4.13, heredan clases para hacer y deshacer inserciones, borrados, zoom, etc ...

Todo comando deberá tener un nombre e implementar los métodos hacer y deshacer.

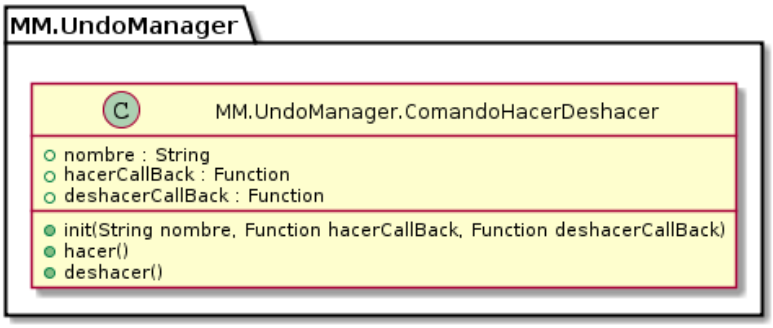

Figura 4.14: Clase MM.UndoManager.ComandoHacerDeshacer

La funcionalidad del método hacer se encargará de repetir la operación ejecutada y el deshacer de revertirla.

Clase MM.UndoManager

El manejador de acciones hacer/deshacer tiene un registro de comandos ejecutados en la aplicación y un puntero[2] que indica el último comando ejecutado. El comportamiento es que siempre se puede deshacer la última acción ejecutada, apuntada por actual, y sólo se puede hacer el comando siguiente al puntero actual.

Como puede observar en la figura 4.15 el puntero *Actual* indica que comando se puede deshacer y *Actual + 1* el comando que se puede hacer.

Los métodos:

- **MM.UndoManager.init:** al constructor se le puede indicar el máximo de la pila de ejecución.
- **MM.UndoManager.add:** añade un nuevo comando a la pila de ejecución.
- **MM.UndoManager.hacer:** ejecuta el hacer del comando que apunta actual + 1 y avanza el puntero.
- **MM.UndoManager.deshacer:** ejecuta el deshacer del comando que apunta actual y retrocede el puntero.

[2]Campo actual.

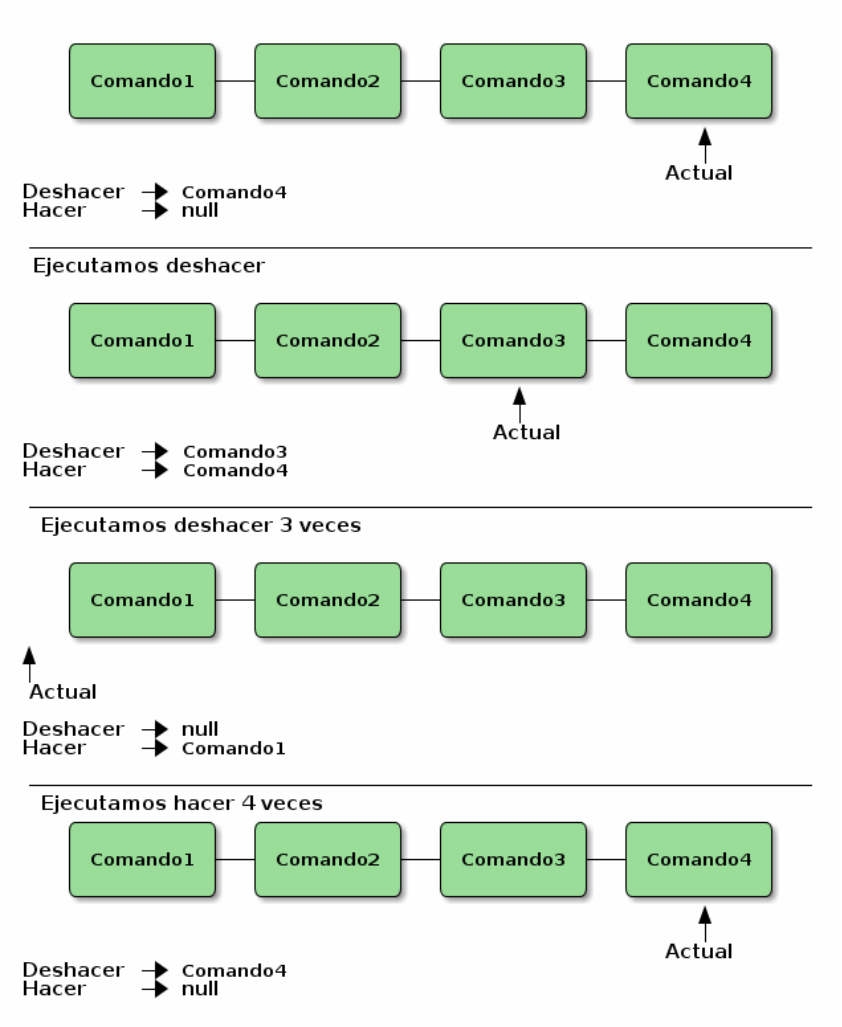

Figura 4.15: Secuencia de ejecución de UndoManager

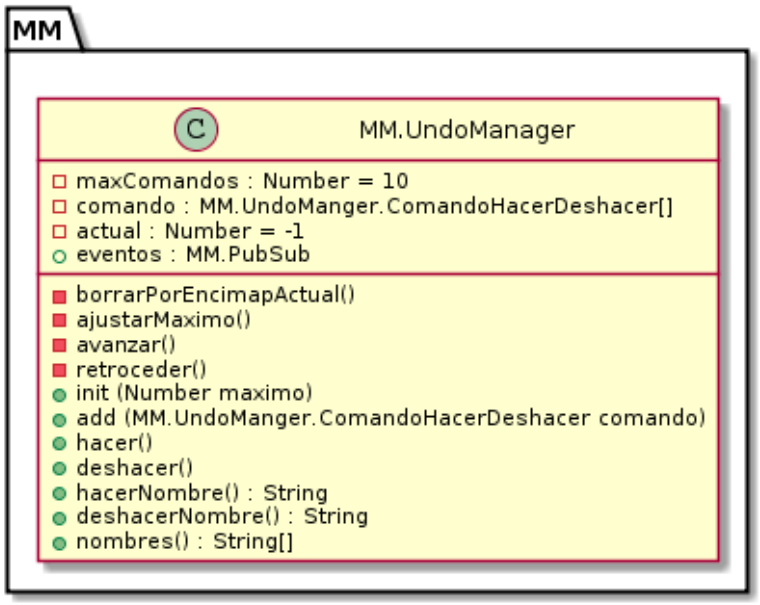

Figura 4.16: Clase MM.UndoManager

- **MM.UndoManager.hacerNombre:** devuelve el nombre del siguiente comando hacer.
- **MM.UndoManager.deshacerNombre:** devuelve el nombre del siguiente comando deshacer.
- **MM.UndoManager.nombres:** lista de comandos en la lista.

4.3.3. Diagrama de clases MM

El centro de la aplicación es sin lugar a dudas el módulo MM. El módulo MM aglutina y vertebra la ejecución del editor de mapas mentales.

Una mapa Mental (MM) tiene un árbol que representa la estructura del mapa mental y un manejador de eventos con el que podemos publicar los eventos de la aplicación para avisar a otras partes integrantes del sistema[3]. Así pues cuando el usuario añade un añade un nuevo elemento al mapa mental, MM se encargará:

- Mantener la coherencia de los datos.

[3]Por ejemplo al render o al interface de usuario

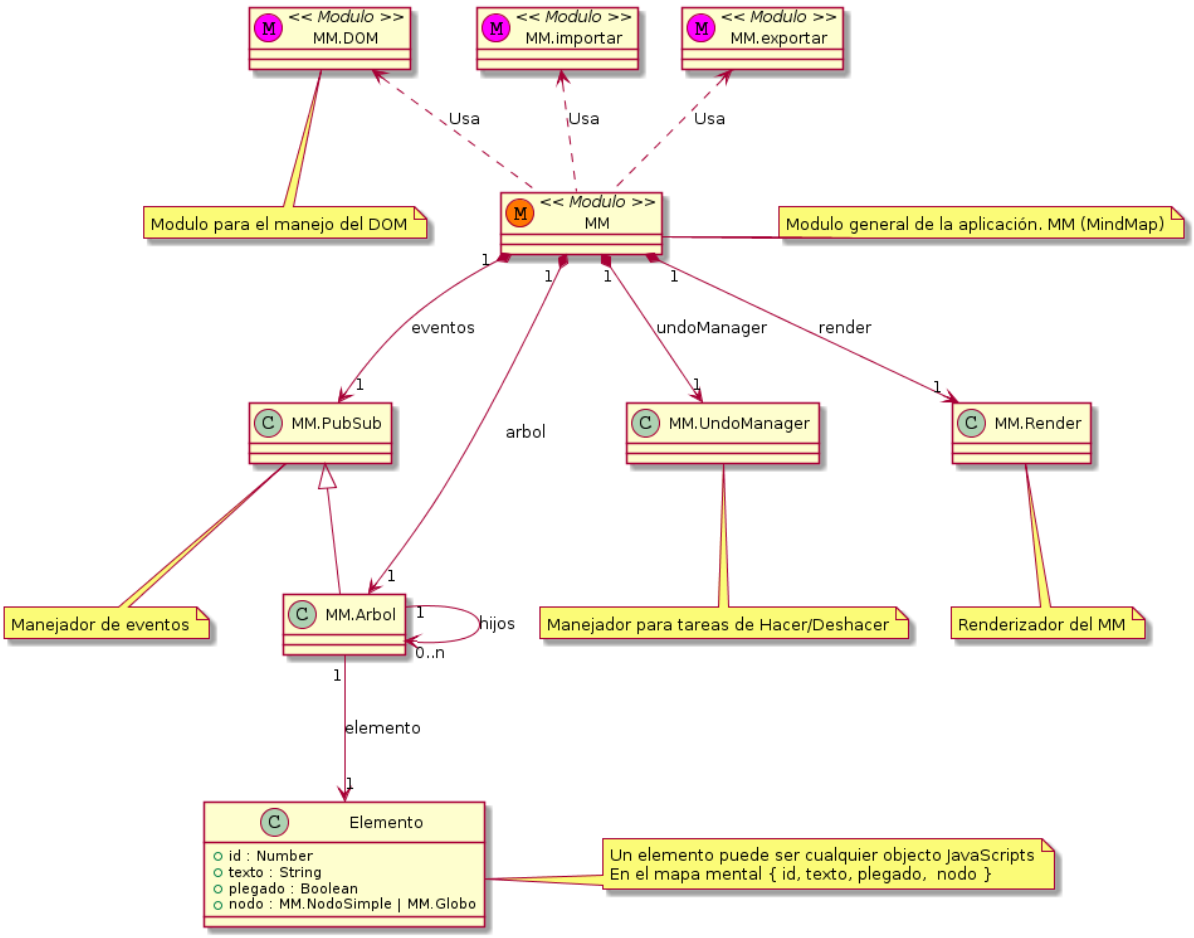

Figura 4.17: Diagrama de clases MM

- Registrar el comando ejecutado en el UndoManager
- Y avisar o publicar el evento de para indicar la operación realizada.

Cada elemento de un nodo del árbol tiene un id de nodo, un texto, un indicador de plegado y un nodo gráfico.

Módulo MM

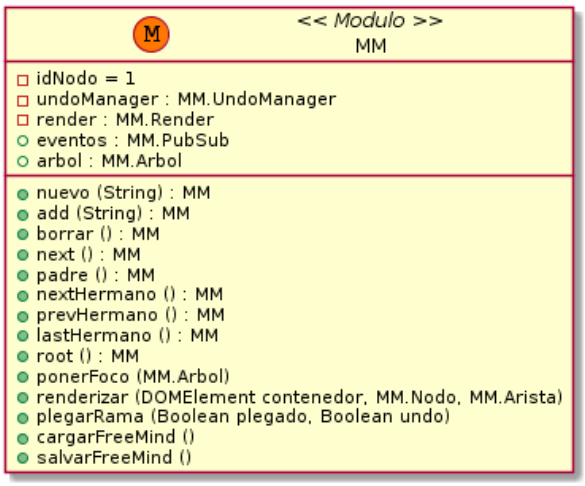

Figura 4.18: Clase MM

El módulo tiene los siguientes métodos:

- **MM.nuevo:** crea un nuevo mapa mental.
- **MM.add:** añade un nuevo nodo hijo al nodo activo.
- **MM.borrar:** borra el nodo activo.
- **MM.next:** mueve el foco al primer hijo del nodo activo.
- **MM.padre:** mueve el foco al padre del nodo activo.
- **MM.nextHermano:** mueve el foco al siguiente hermano del nodo activo.
- **MM.prevHermano:** mueve el foco al hermano anterior del nodo activo.
- **MM.lastHermano:** mueve el foco al último hermano del nodo activo.
- **MM.root:** mueve el foco al nodo raíz.

- **MM.ponerFoco:** establece el foco en nodo dado.
- **MM.renderizar:** se encarga de renderizar el mapa mental.
- **MM.plegarRama:** función para plegar y desplegar ramas.
- **MM.cargarFreeMind:** función de carga de ficheros FreeMind.
- **MM.salvarFreeMind:** se encarga de salvar el mapa mental en formato FreeMind.

Clase MM.Arbol

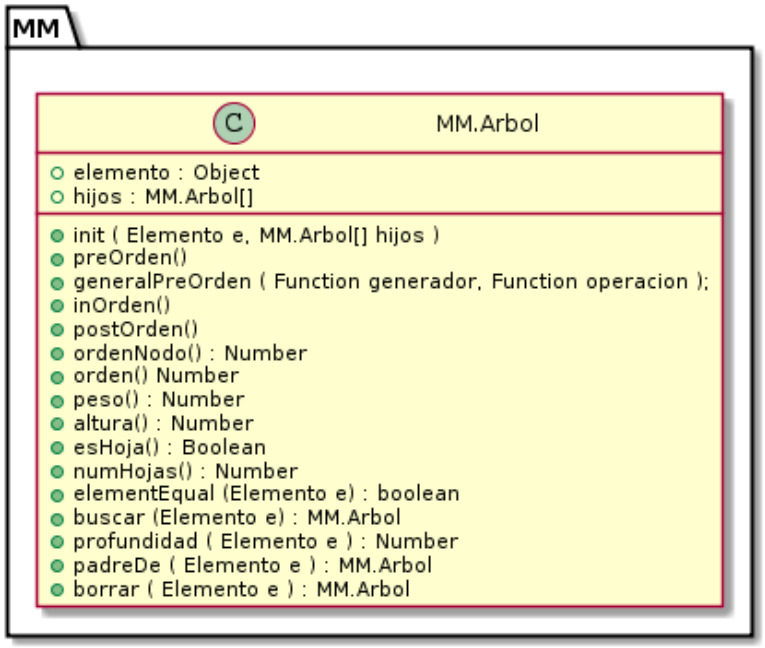

Figura 4.19: Clase MM.Arbol

La implementación MM.Arbol debe ser una implementación funcional de un árbol-n lo más general posible.

- **MM.Arbol.init:** Crea un nuevo árbol-n con un elemento raíz y array de árboles hijos.
- **MM.Arbol.preOrden:** realiza un recorrido en preorden por el árbol.
- **MM.Arbol.generalPreOrden:** recorrido en preorden, con un generador que trata el elemento actual y una operación que se encarga de operar el elemento generado

con el preorden de los nodos hijos.

- **MM.Arbol.inOrden:** realiza un recorrido inorden por los elementos del árbol.
- **MM.Arbol.postOrden:** recorre el árbol el postorden.
- **MM.Arbol.ordenNodo:** calcula el orden del nodo actual.
- **MM.Arbol.orden:** calcula el orden del árbol completo.
- **MM.Arbol.peso:** calcula el peso de un árbol.
- **MM.Arbol.altura:** altura del árbol.
- **MM.Arbol.esHoja:** indica si el nodo actual es un nodo hoja o no.
- **MM.Arbol.numHojas:** determina el número de nodos hojas del árbol.
- **MM.Arbol.elementEqual:** función de igual entre elementos de los nodos. Por defecto, es la igual estricta -==". Esta función podrá ser sobreescrita para adecuarse al tipo de elemento guardado en cada nodo.
- **MM.Arbol.buscar:** busca un elemento en el árbol. Como comparador de nodos se utiliza la función MM.Arbol.elementEqual.
- **MM.Arbol.profundidad:** determina la profundidad del árbol.
- **MM.Arbol.padreDe:** calcula el árbol padre del elemento pasado.
- **MM.Arbol.borrar:** borra un elemento del árbol.

Módulo MM.DOM

El módulo MM.DOM contendrá funciones para el manejo del DOM. Creación y borrado de elementos DOM.

Clase MM.Render

La clase MM.Render es la encargada de pintar el mapa mental y realizar los ajustes visuales necesarios para mostrar los nodos y las aristas. El renderizador se

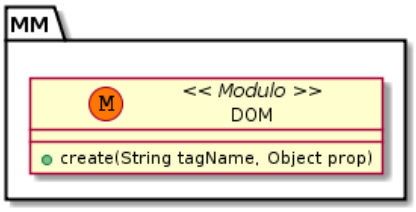

Figura 4.20: Modulo MM.DOM

configura o inicializa entorno a un elemento DOM, normalmente un *div*, una clase que MM.NodoSimple[4] y una clase MM.Arista[5]. A partir de estos datos el sistema es capaz de ir generando el mapa mental en función de los eventos producidos en el módulo MM.

Figura 4.21: Modulo MM.Render

La clase MM.Render dispone de los siguientes métodos:

- **MM.Render.init:** constructor de la clase render. Inicializas las capas del de nodos y aristas.

[4] O que herede de MM.NodoSimple como MM.Globo
[5] O que herede de MM.Arista

- **MM.Render.renderizar:** se encarga de realizar las suscripciones a eventos, dibujar y establecer los atajos de teclado.
- **MM.Render.renderizarAristas:** pinta las aristas entre los distintos nodos.
- **MM.Render.dibubar:** en función del mapa actual del módulo MM dibuja y reparte el espacio de dibujo.
- **MM.Render.suscribirEventos / dessuscribirEventos:** métodos de activar y desactivar las suscripciones a eventos del render.
- **MM.Render.get/setEscala:** establece o devuelve la escala actual.
- **MM.Render.zoomIn / zoomOut / zoomReset :** funciones de zoom, en orden, aumenta, disminuye o reinicia la escala del mapa mental.
- **MM.Render.cambiarFoco:** se encarga de resalta la idea que tiene el foco actual.
- **MM.Render.modoEdicion:** indica si la idea actual esta en modo de edición o no.
- **MM.Render.editar:** establece la idea actual en modo de edición. Mostrando el editor del nodo y activando y desactivando atajos de teclados y eventos.
- **MM.Render.nuevoNodo:** manejador de eventos para cuando se inserta una nueva idea. Se encarga de insertar la nueva idea y enlazar la idea padre con la hija mediante una arista. El sistema de establece la mejor ubicación para el nuevo elemento.
- **MM.Render.borrarNodo:** manejador de eventos para cuando se borra un idea y la correspondiente arista. Además se debe redistribuir el mapa mental en función de los nodos restantes.
- **MM.Render.buscarArista:** busca una arista entre dos ideas.
- **MM.Render.borrarArista:** borra una arista existente entre dos ideas.

4.3.4. Diagrama de clases nodo.

El nodo se encarga del pintado de una idea del mapa mental, en esencia, es un MM.Mensaje al cual se le han añadido otros elementos gráficos y funcionalidades. Existen

dos implementaciones de nodo, como se pueden ver en el diagrama4.22, el MM.NodoSimple y el MM.Globo, y ambos pueden ser usados desde MM.Render. Todos los nodos existen un escenario y en una capa dada.

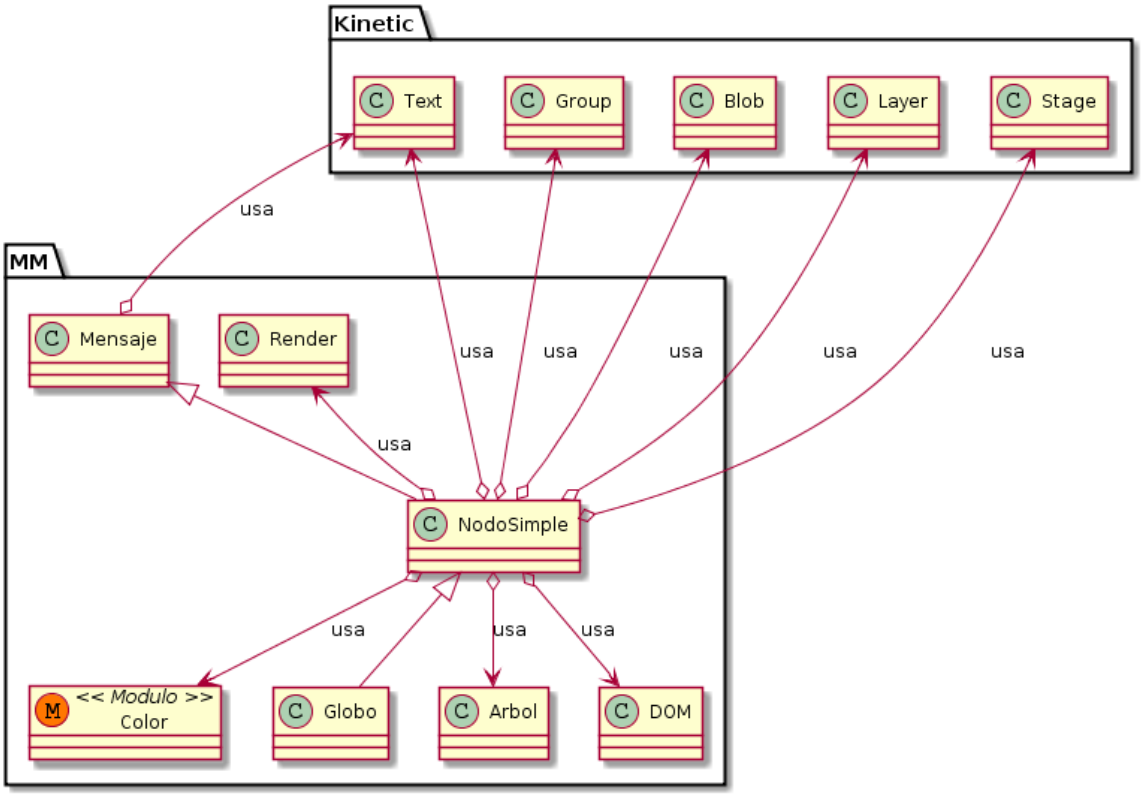

Figura 4.22: Diagrama de clases nodo

Clase MM.Mensaje.

Se trata de una simple clase que pinta un texto en una capa dada.

- **MM.Mensaje.init:** constructor de la clase. Tiene el escenario y la capa donde pintar el mensaje, además de un objeto de propiedades con la posición, texto, etc...
- **MM.Mensaje.getText/setText:** métodos para establecer y obtener el texto del mensaje.
- **MM.Mensaje.getX/setX:** métodos para establecer y obtener la posición[6] X del mensaje.

[6] en píxeles

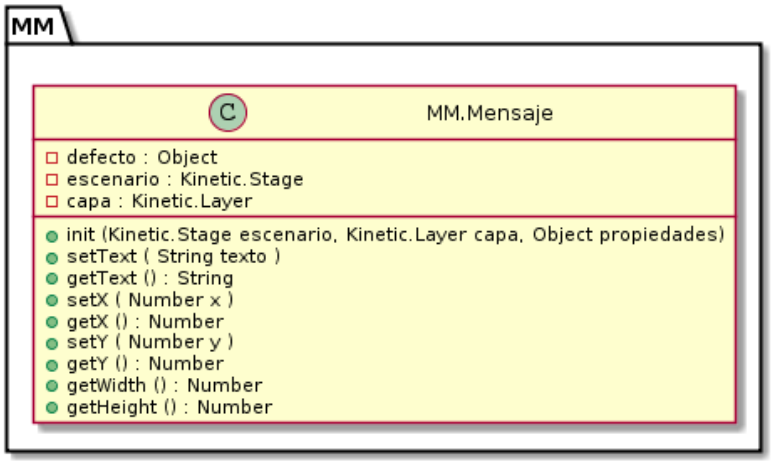

Figura 4.23: Clase MM.Mensaje

- **MM.Mensaje.getY/setY:** métodos para establecer y obtener la posición Y[7] del mensaje.
- **MM.Mensaje.getWidth:** devuelve el ancho del texto en píxeles.
- **MM.Mensaje.getHeight:** devuelve el alto del texto en píxeles.

Clase MM.NodoSimple.

Hereda de MM.Mensaje y representa un mensaje o idea subrayado. El nodo representa una idea que será renderizado y creado desde MM.Render. Su funcionalidad básica pasa por obtener el foco cuando sea la idea activa, poderse editar, ocultar cuando su rama este plegada o mostrar cuando este desplegada.

- **MM.NodoSimple.init:** constructor de la clase. Recibe el MM.Render, la idea a la que representa y un conjunto de propiedades como la posición, escala, etc ...
- **MM.ponerFoco/quitarFoco:** métodos que poner o quitan el foco en la idea que representa el nodo. Debe resaltar el nodo cuando este esté focalizado.
- **MM.Mensaje.editar/cerrarEdicion:** métodos para establecer la idea en modo edición y para cerrarlo cuando se termine la edición. Si esta en modo edición debe tener el foco.

[7] en píxeles

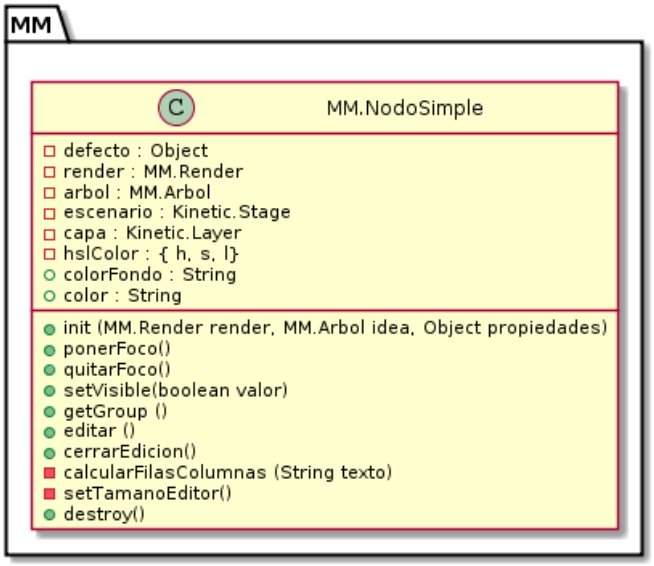

Figura 4.24: Clase MM.NodoSimple

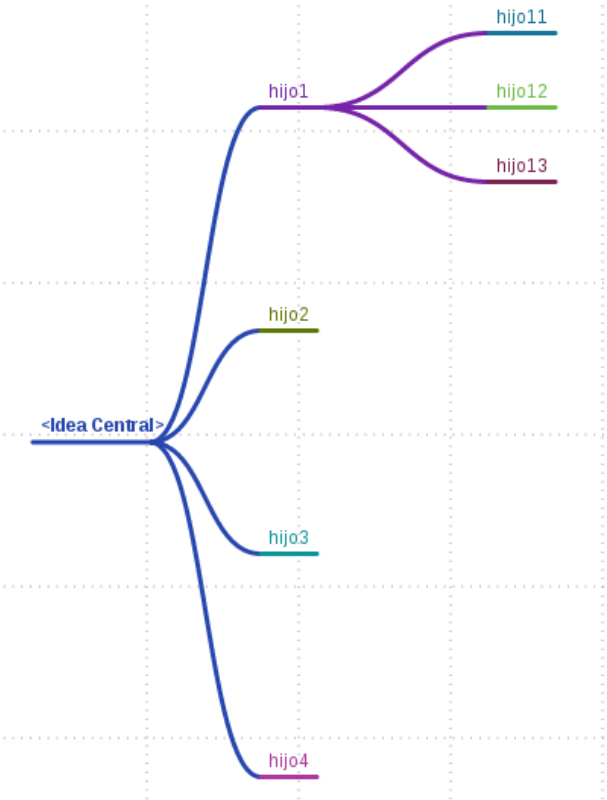

Figura 4.25: Mapa mental con renderización de nodos simples

- **MM.Mensaje.setVisible:** indica si el mensaje debe mostrarse o no.
- **MM.Mensaje.destroy:** borra y destruye el nodo.

Clase MM.Globo.

Se trata de un nodo más elaborado. Representa al texto de la idea incluido en un globo.

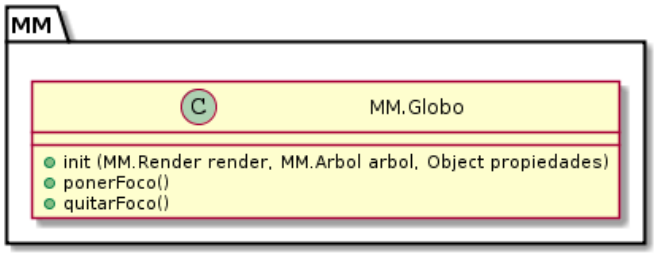

Figura 4.26: Clase MM.Globo

Figura 4.27: Mapa mental con renderización de nodos globo

Módulo MM.Color.

Módulo con funcionalidades de color. Permite generar distintos representaciones de color[8] y realizar conversiones sobre los distintos tipos.

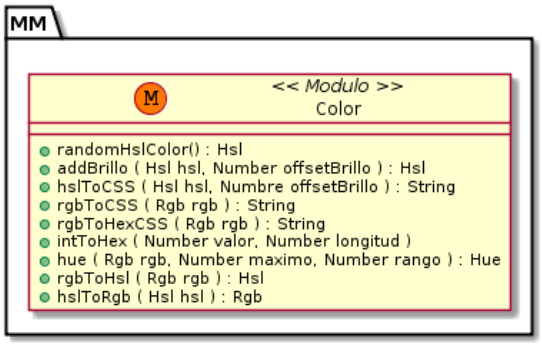

Figura 4.28: Modulo MM.Color

[8] HSL, RGB y HUE

4.3.5. Diagrama de clases de aristas.

Una arista representa la línea de unión entre dos ideas o nodos. Existen dos tipos de aristas MM.Arista y MM.Rama, ambas tienen dos nodos a los deben unir. Las aristas, han sido implementadas con una curva Beizer. El diagrama de clases de aristas podemos ver lo en la figura 4.29.

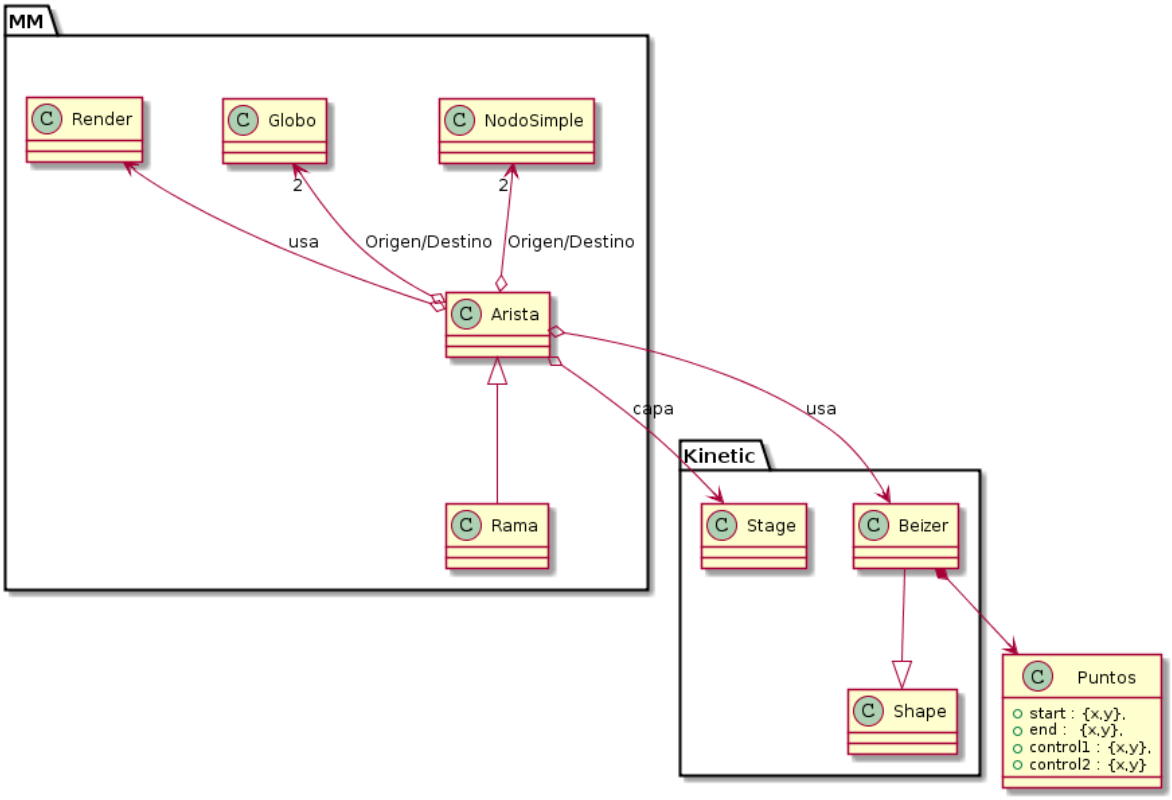

Figura 4.29: Diagrama de clases aristas

4.3.6. Clase Kinetic.Beizer.

Extensión realizada en la librería KineticJS. Una curva Beizer está representada por cuatro puntos inicio, fin y dos puntos de control que determinan la curvatura. En el constructor debe recibir un objeto con los puntos de inicio, fin y de control. Esta clase se encarga de pintar en un canvas la curva en cuestión.

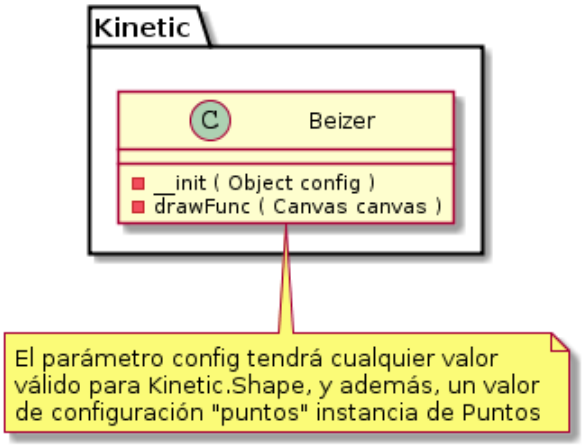

Figura 4.30: Clase Kinetic Beizer

Clase MM.Arista.

Una arista recibe dos ideas y un tamaño (o grosor de línea). Esta clase en cuestión se encarga de unir dos ideas mediante una curva beizer, y de mantenerlos unidos a pesar de los cambios.

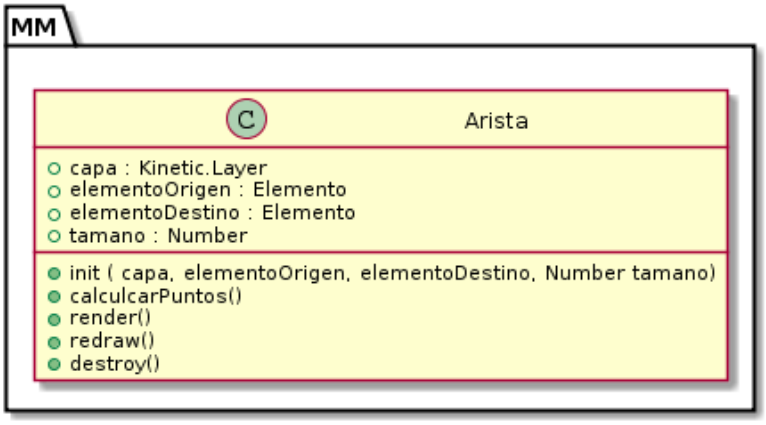

Figura 4.31: Clase MM.Arista

- **MM.Arista.init:** constructor de la clase.
- **MM.Arista.calcularPuntos:** calcula los puntos para dibujar la curva.
- **MM.Arista.render:** dibuja la curva beizer.
- **MM.Arista.rendraw:** redibuja la curva beizer para adaptarse a los cambios

producidos en su entorno.

- **MM.Arista.destroy:** borra y destruye la arista.

Clase MM.Rama.

Se trata de otro tipo de arista pensada para unir dos nodos de tipo MM.NodoSimple.

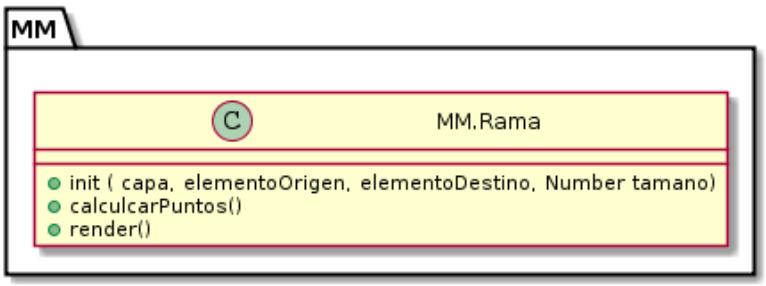

Figura 4.32: Clase MM.Rama

4.3.7. Diagrama de clases de teclado

Para una mejor experiencia de usuario se ha implementado un complejo manejador de teclados para procesar teclas del tipo *Modificadores+tecla*. El manejo de teclado en el mundo web puede complicarse bastante ya que dependen del navegador y el sistema operativo, ya no sólo por que pueden existir o no teclas como *Meta*[9] o *Windows*, si no por que existen teclas como + que tienen distinto keycode en un Firefox, Chrome y Safari.

También hay que tener en cuenta que las aplicaciones webs no han sido pensadas para un uso intensivo de teclado.

Módulo MM.teclado.atajos

Un atajo esta compuesto por un nombre[10], una función que será ejecutada cuando se detecte la pulsación de la secuencia de teclas. Un atajo puede estar activado o desactivado, es decir, que se ejecutará cuando se detecte el atajo de teclado o no.

El modulo de atajos registrar los atajos de teclados del sistema.

[9] En los sistemas Mac.
[10] Ctrl+i

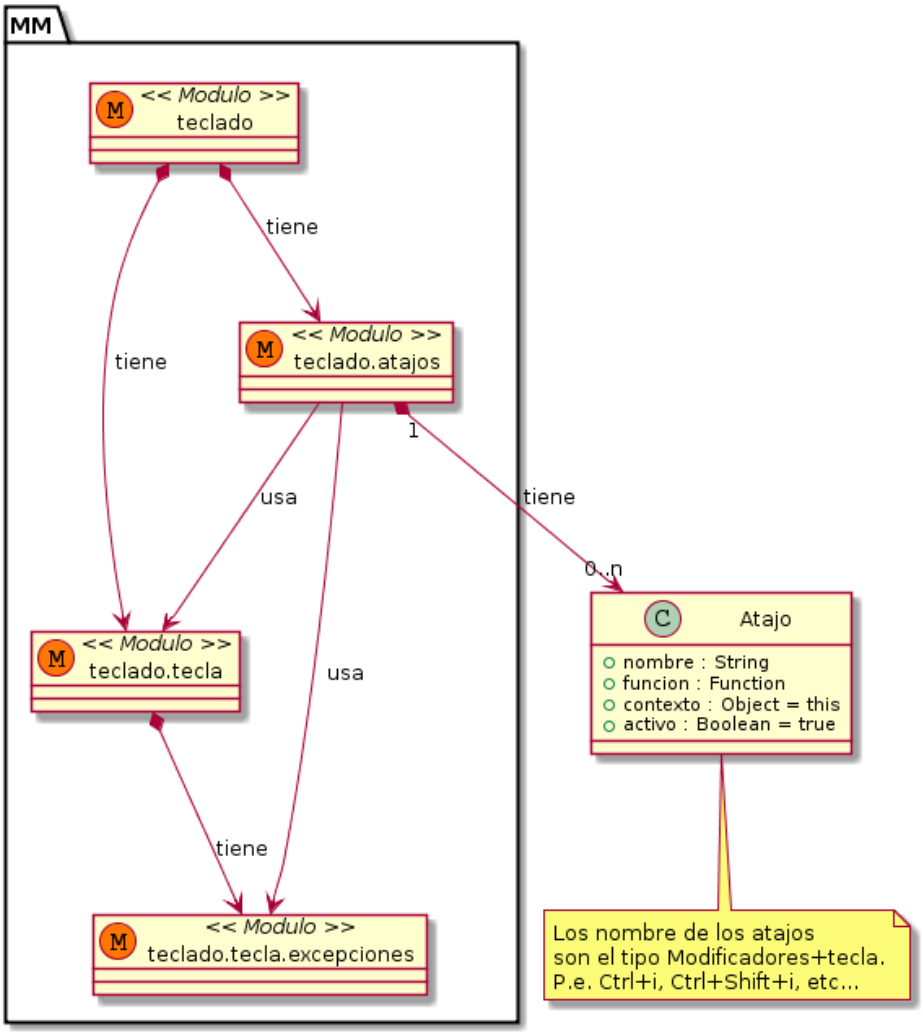

Figura 4.33: Diagrama de clases teclado

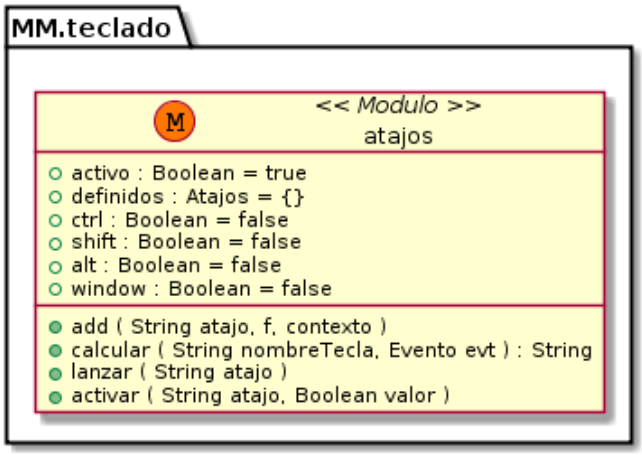

Figura 4.34: Modulo MM.teclado.atajos

- **MM.teclado.atajos.add:** añade un nuevo atajo de teclado al sistema.
- **MM.teclado.atajos.calcular:** calcula el atajo de teclado producido.
- **MM.teclado.atajos.lanzar:** lanza un atajo de teclado, es decir, la función asociada.
- **MM.teclado.atajos.activar:** activa o desactiva un atajo de teclado

Módulo MM.teclado.tecla

Se trata de un conjunto de constantes de códigos de teclados. También incluye las posibles excepciones y discordancias que se producen entre los distintos navegadores y sistemas operativos.

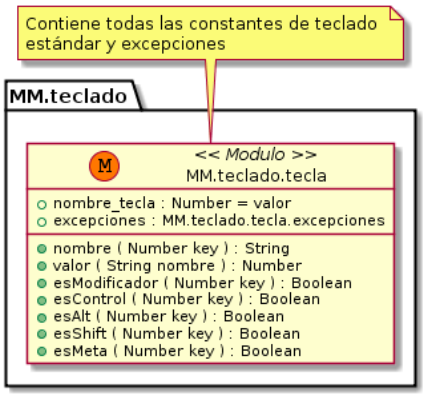

Figura 4.35: Clase MM.teclado.tecla

- **MM.teclado.tecla.nombre:** calcula el nombre de una tecla en función de su código.
- **MM.teclado.tecla.valor:** nos devuelve el código de tecla en función del nombre.
- **MM.teclado.tecla.esModificador:** indica si un código de teclado es un modificador.
- **MM.teclado.tecla.esControl:** indica si el código de teclado se corresponde con la tecla <Control>.
- **MM.teclado.tecla.esAlt:** indica si el código de teclado se corresponde con la tecla <Alt>.

- **MM.teclado.tecla.esShift:** indica si el código de teclado se corresponde con la tecla <Shift>.
- **MM.teclado.tecla.esMeta:** indica si el código de teclado se corresponde con la tecla <Meta>.

Módulo MM.teclado

Módulo encargado de mantener el registro de atajos y manejar los eventos de teclado.

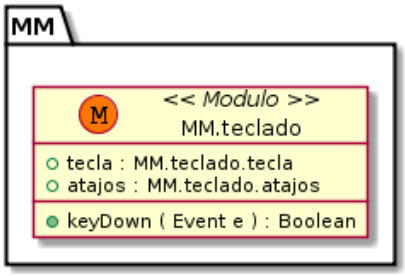

Figura 4.36: Clase MM.teclado

El sistema de control de teclado se encarga de recoger todos los eventos[11] de pulsación de teclas y revisar y calcular si se trata de un atajo registrado en el sistema y lanzar la función asociada a dicho atajo.

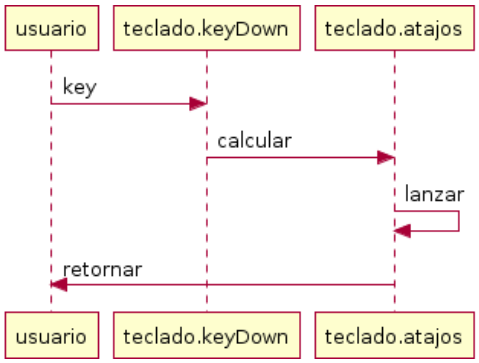

Figura 4.37: Diagrama de secuencia teclado

[11] Todos los eventos KeyDown del navegador.

Capítulo

5 Implementación.

qué es JavaScript,

qué son los prototipos,

cómo es posible hacer clases con JQuery,

en qué contexto y qué herramientas típicamente has usado, cómo te has movido para hacer el desarrollo... y ese tipo de cosas, estaría bien.

Quizás también presentar KineticJS con algún ejemplo básico, y el tema de la interactividad de esos canvas. Antes puedes también añadir a la descripción general que haces de HTML5 el tema de Canvas, cómo aparece, por qué hacía falta, limitaciones, defectos...

5.1. Javascript.

5.1.1. Qué es.

Mozilla, los herederos directos de Netscape, definen javascript como:[1] [2]

[1] MDN (Mozilla Developer Network)

[2] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/JavaScript_Overview

> JavaScript is a cross-platform, object-oriented scripting language. JavaScript is a small, lightweight language; it is not useful as a standalone language, but is designed for easy embedding in other products and applications, such as web browsers. Inside a host environment, JavaScript can be connected to the objects of its environment to provide programmatic control over them.

Definición que desde mi punto de vista se queda corta. Ya que Javascript es un lenguaje de scripting (que debe ser interpretado), imperativo, estructurado, orientado a objeto sin clases, débilmente tipado, dinámico, funcional y basado en prototipos.

De C ha heredado que sea un lenguaje imperativo y estructurado con distinción entre sentencias y expresiones. A diferencia de C y Java el ámbito de las variables no son a nivel de bloque sino de función, es decir, que una variable definida dentro de una sentencia puede ser utilizada fuera de dicha sentencia, ya que el ámbito no lo define la sentencia sino la función que la contiene.

```
1 var f = function (valor) {
2   if ( valor ) {
3     var resultado = 'Si';
4   }
5   return resultado;
6 };
```

Sin lugar a dudas se trata de un lenguaje orientado a objeto. Los arrays, números, funciones, cadenas, casi en su totalidad el lenguaje son objetos[3]. Los objetos son array asociativos al cual podemos acceder a través de la notación objeto.campo o como si de un array se tratará.

```
1 var f = function () {
2   var objeto = {
3     campo0 : 0,
4     campo1 : 'una cadena'
5   };
6
7   var valorCampo0 = objeto.campo0;
8   valorCampo0 = objecto['campo0'];
9 };
```

Es débilmente tipado, el tipo no va asociado a la variable si al valor que contiene. Por lo que podemos crear variables y asignarle un valor numérico y posteriormente una cadena.

```
1 var f = function () {
2   var numero = 1;
3   numero++;
4   numero = '1';
5 };
```

[3]Salvo los valores destacados null y undefined, el resto de valores javascripts son objetos

Se trata de un lenguaje funcional en el que una función es un objeto de primera clase. Esto significa que Javascript soporta el paso de funciones como argumentos a otras funciones, funciones que devuelve funciones, variables que almacenan funciones, creación de funciones anónimas, etc ...

```
function map(f, xs) {
  var result = new Array();
  for (var i = 0; i < xs.length; i++)
    result.push(f.apply(null, [xs[i]]));
  return result;
}
```

Javascript no utiliza el mecanismos de clases para implementar la herencia, para ello hace uso de los prototipos.

```
var Persona = function (){
  this.nombre = 'Sin nombre';
};

Persona.prototype.setNombre = function () {
  this.nombre;
};

var pepe = new Persona(); // pepe es una instancia de Persona
pepe.setNombre('Pepe');   // y contiene una copia del prototipo de Persona.
```

5.1.2. Un poco de historia.

Cuando en 1996, el navegador Netscape introdujo su primer interprete de Javascript[4] nadie podía intuir la importancia que adquiriría años después.

Internet aun estaba en pañales, navegar era lento[5] y los ordenadores personales poco potentes. En el mejor de los casos, el usuario tenía que esperar durante largo tiempo para poder interactuar con la web solicitada. Las páginas comenzaban a ser más complejas, y la navegación más lenta, de ello surguió la necesidad de un lenguaje de programación que se ejecutará en el navegador del cliente. De esta forma, si el usuario introducía un valor incorrecto, en un formulario, no tendría que esperar a la respuesta del servidor, el mismo cliente podría dar una respuesta más rápida, indicando los errores existentes.

Netscape Navigator 3.0 incorporó la primera versión del lenguaje, como ya se había comentado, y al mismo tiempo, o al poco, Microsofot lanzo JScript en su Internet Explorer 3. JScript no erá más que una copia de Javascript al que le cambiaron el nombre para evitar problemas legales. De esta forma comienzan las divergencias entre las distintas

[4] Javascript fue un nombre por conveniencia legal. Originalmente se llamaba LiveScript

[5] La velocidad máxima de los modems de usuario era 28.8Kbps

versiones de Javascript, en esencia todas parten del mismo lenguaje y estandar, pero cada una aportaba sus mejoras provocando diferencias entre ellas.

La guerra entre las distintas versiones estaba servida. Todos deseaban que su versión fuera la aceptada por la comunidad y se popularizará. Bien intentando estandarizar su versión, o buscando que se evitará la guerra de tecnologías, Netscape decidió dar el paso, y en 1997 puso a disposición de ECMA[6] la especificación de Javascript1.1. ECMA creo el comité TC39 del cual surgío el primer estándar que se denominó ECMA-262[7], o más popularmente, ECMAScript.

Durante mucho tiempo el estándar ECMAScript no fue el aceptado por todos los navegadores, ni que decir tiene que el más reacio al cambio fue el Internet Explorer de Microsoft. Es ahora, donde Microsoft a dado su brazo a torcer y poco a poco tiende al estándar ECMAScript facilitando al los desarrolladores la tarea.

5.1.3. Prototipos

Como se ha comentado con anterioridad Javascript no utiliza el mecanismos de clases[8] para implementar la herencia, para ello hace uso de los prototipos. Los prototipos son un paradigma de programación orientada a objetos en la cual instanciación de objetos se hace mediante la clonación de otros objetos.

Los objetos en cualquier lenguaje son un conjunto de propiedades y métodos. Pues bien, Javascript carece de métodos en su lugar existen propiedades que apuntan a funciones que hacen las veces de métodos. Además, cada objeto tiene un enlace interno a otro objeto llamado prototipo[9]. El prototipo de un objeto puede ser, o bien otro objeto, o bien el valor null. Es lo que se llama cadena de prototipos o cadena prototípica (ver figura5.1).

Cómo se puede observar toda cadena de prototipo acaba con el prototipo de Object, cuyo prototipo es null. Veamos algunos ejemplo de cadenas prototípicas.

```
1 > var objeto = { a : 1 }; \\ cadena prototÃpica de objeto --> Object.prototype --> null
2 > Object.getPrototypeOf(o);
3   Object {}
```

[6] European Computer Manufacturers Association. Web oficial http://www.ecma-international.org/

[7] Se puede encontrar la versión 5.1 en http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf

[8] Paradigma sin clases

[9] prototype

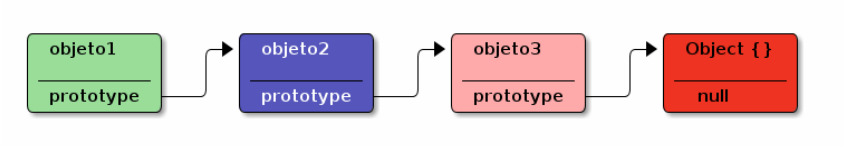

Figura 5.1: Cadena prototípica de objetos

```
> Object.getPrototypeOf(Object.getPrototypeOf(o));
  null

\\ cadena prototÃpica de una array --> Array.prototype --> Object.prototype --> null
> var array = [1,2];
> Object.getPrototypeOf(array);
  []
> Object.getPrototypeOf(Object.getPrototypeOf(array));
  Object {}
> Object.getPrototypeOf(Object.getPrototypeOf(Object.getPrototypeOf(array)));
  null
```

Herencia de propiedades y métodos

Un objeto javascript es un conjunto de propiedades[10] que en el momento de la herencia se copian en el nuevo objeto o objeto hijo. Así pues, en el siguiente ejemplo podemos observar como se heredan las propiedades y los "métodos".

```
> var a = {
  contador: 0,
  contar : function () {
    console.log('Contador ' + this.contador++);
  }
};
> a.contar();
  Contador 0
> a.contar();
  Contador 1

> var b = Object.create(a); // Crea un objeto "b" que hereda de "a"
> b.contador; // es una copia exacta de "a"
  2
> b.contar();
  Contador 2
> a.contar();  // una copia no el mismo
  Contador 2

// la cadena de prototipos de los objectos:
// b.prototype --> a.prototype --> Object.prototype --> null
```

También hay que tener especial cuidado con la palabra reservada this que siempre apunta al objeto que está heredando y no al prototipo.

Constructor, propiedad prototype y herencia

Todos los objetos poseen un único constructor. Un constructor es sólo una función que ha sido llamada con la palabra reservada new.

[10] Los métodos son propiedades que referencia a una función

Todo constructor tiene una propiedad prototype con la cual podemos definir el prototipo de todos los objetos creados con dicho constructor.

```
> var Persona = function (nombre) {
  this.nombre = nombre;
};

> Persona.prototype = {
  saluda : function () {
    console.log('Hola soy ' + this.nombre);
  }
};

> var pepe = new Persona ("pepe");
> pepe.saluda()
  Hola soy pepe

/* prototipo de pepe --> Persona.prototype --> Object.prototype --> null */

> var juan = new Persona ("juan");
> juan.saluda()
  Hola soy juan
```

En el siguiente, ejemplo se ilustra como se puede implementar la herencia en base a un constructor y las propiedades. Para ello, vamos a basarnos en el ejemplo anterior.

```
> var Empleado = function (nombre, puesto) {
  this.nombre = nombre;
  this.puesto = puesto;
};
> Empleado.prototype = new Persona('');
// sobreescribimos saluda
> Empleado.prototype.saluda = function () {
  console.log('Hola soy ' + this.nombre + ' ' + this.puesto );
};

> var pepe = new Empleado ('pepe', 'programador');
> pepe.saluda();
  Hola soy pepe programador
> pepe instanceof Empleado
  true
> pepe instanceof Persona
  true
```

5.1.4. Ámbito de variable (Scope)

El ámbito de una variable[11] es la zona del programa donde es accesible la variable. En JavaScript existen dos ámbitos: local y global.

En el ámbito global, las variables son accesibles desde cualquier punto del programa. Salvo si existe una variable con él mismo nombre en el ámbito local.

Cuando hablamos de ámbito local, en Javascript, nos referimos a nivel de función. Es decir, que las variables declaradas dentro de la función serán accesibles sólo dentro de la propia función.

```
var vbleGlobal = 'Soy una variable global';

```

[11] scope

```
3  function fn (){
4    var vbleLocal = "Soy una variable local";
5    // vbleGlobal === 'Soy una variable global'
6  }
7
8  // vbleLocal === undefined
```

5.1.5. Patrón módulo

Popularizado por Douglas Crockford el patrón módulo es sin lugar a dudas el más utilizado dentro del mundo de Javascript. Su simplicidad encierra gran potencia y flexibilidad que han aprovechado multitud de librerías[12].

Para definir un módulo nos basamos principalmente en dos conceptos fundamentales:

- El **ámbito local**, nos va a permitir crear funciones y variables locales al módulo, es decir, privadas a nuestro módulo.
- Y en una **función auto-ejecutable** que retorna un objeto con el interfaz pública del módulo.

El siguiente módulo muestra un ejemplo básico de módulo[13].

```
1  // El espacio de nombres en este ejemplo es la variable "modulo"
2  var modulo = function () {
3    // variables privadas
4    var p1, p2;
5
6    // funciones privadas
7    function privado() {
8    }
9
10   // Interfaz pÃºblica
11   return {
12     variablePublica : null,
13     funcionPublica: function () {
14     }
15   }
16 }();
```

Entre sus virtudes más destacadas están:

- Encapsulamiento bajo un **espacio de nombres**. Evitando colisiones de nombres con otras librerías.
- Permite y propicia una mejor organización del código permitiendo o facilitando la **reutilización**.

[12] Por citar algunas de las más populares: JQuery, Dojo y Undercore, entre otros
[13] Módulo propuesto pro Douglas Crockford

- Al quedar encapsulado bajo un espacio de nombre nos lleva a mantener un **contexto global limpio**. Sólo necesitamos de una variable global[14].
- Concepto simple y fácilmente extensible.

En MindMapJS no es una excepción, se ha utilizado un espacio de nombres MM.[15] Simplemente esto:

```
1  /**
2   * @file MindMapJS.js Definición del espacio de nombres de la aplicación MM
3   * @author José Luis Molina Soria
4   * @version @@version
5   * @date    @@date
6   */
7
8  /**
9   * Espacio de nombres de la aplicación MindMapJS. Reducido a MM por comodidad
10  * @namespace MM
11  * @property {MM.Class}        Class      - Sistema de clases para MM
12  * @property {MM.Arbol}        Arbol      - Constructor de Árboles enarios.
13  * @property {MM.Properties}   Properties - Extensión para manejo de propiedades
14  * @property {MM.DOM}          DOM        - Funciones para manejo del DOM
15  * @property {MM.PubSub}       PubSub     - Patrón Publish/Subscribe
16  * @property {MM.teclado}      teclado    - Gestión y manejo de eventos de teclado
17  */
18 var MM = {};
19
20 if ( typeof module !== 'undefined' ) {
21     module.exports = MM;
22 }
```

El modulo MM es tiene el interfaz de uso para la aplicación y sobre el que gira todo comportamiento.

```
1  /**
2   * @File mm.js Implementación del MM
3   * @author José Luis Molina Soria
4   * @version 20130520
5   */
6  MM = function (mm) {
7
8      /**
9       * @prop {number} idNodos Identificador de nodos. Cada vez que se crea un nodo se
10      *                        le asigna un nuevo identificador
11      * @memberof MM
12      * @inner
13      */
14     var idNodos = 1;
15
16     /**
17      * @prop {MM.UndoManager} undoManager es el manejador de acciones hacer/deshacer (
            undo/redo)
18      * @memberof MM
19      * @inner
20      */
21     mm.undoManager = new MM.UndoManager(10);
22
23     /**
24      * @prop {MM.PubSub} eventos Gestor de eventos del Mapa mental
25      * @memberof MM
26      * @inner
27      */
28     mm.eventos = new MM.PubSub();
29
30     /**
31      * @desc Sobreescritura del método "equal" del MM.Arbol. La comparación se realiza a
32      *       nivel de identificador.
```

[14] Nos referimos al propio módulo

[15] Utilizado MM (MindMap) por comodidad.

```
33      * @method elementEqual
34      * @memberof MM
35      * @inner
36      */
37     MM.Arbol.prototype.elementEqual = function ( id ) {
38         return id === this.elemento.id;
39     };
40
41     /**
42      * @desc Genera un nuevo Mapa mental. Eliminar el Mapa mental existente hasta el
            momento.
43      *       Resetea el contador de nodos.
44      * @param {String} ideaCentral Texto de la idea central. Por cefecto 'Idea Central'
45      * @method nuevo
46      * @memberof MM
47      * @instance
48      */
49     mm.nuevo = function ( ideaCentral ) {
50         if ( this.arbol ) {
51
52             this.ponerFoco ( this.arbol );
53
54             for ( var i = 0; i < this.arbol.hijos.length; i ) {
55                 this.next();
56                 this.borrar();
57             }
58
59             this.eventos.on ( 'nuevo/pre' );
60         }
61
62         idNodos = 1;
63
64         /**
65          * @prop {MM.Arbol} arbol Arbol-eneario que representa al Mapa mental.
66          * @memberof MM
67          * @inner
68          */
69         this.arbol = this.foco = new MM.Arbol(
70             { id: idNodos++,
71               texto: ideaCentral || 'Idea Central',
72               plegado: false,
73               nodo: null }
74         );
75         this.ponerFoco ( this.arbol );
76         this.eventos.on ( 'nuevo/post' );
77     }.chain();
78
79     /**
80      * @desc Añade un nodo al Mapa mental. Se añade un hijo al elemento activo (que tiene
            el foco).
81      *       Todos los nodos del árbol tiene como elemento un id, texto y un nodo (
            instancia de
82      *       MM.NodoSimple o MM.Globo. Es Chainable, esto nos permite realizar
            operaciones encadenadas.
83      *       Por ejemplo, MM.add('Abuelo').add('Padre').add('Hijo').add('Nieto');
84      * @param {string} texto Texto del nuevo nodo. Valor por defecto "Nuevo".
85      * @return {MM} Al ser Chainable devuelve this (MM).
86      * @method add
87      * @memberof MM
88      * @instance
89      */
90     mm.add = function ( texto ) {
91         texto = texto || "Nueva idea";
92         var nuevo = new MM.Arbol ( { id: idNodos++, texto: texto, plegado: false,  nodo:
                null } );
93         this.foco.hijos.push ( nuevo );
94         this.undoManager.add(new MM.comandos.Insertar(this.foco.elemento.id, nuevo.
                elemento.id, texto) );
95         this.eventos.on ( 'add', this.foco, nuevo );
96         nuevo = null;
97     }.chain();
98
99     /**
100     * @desc Borra el nodo que tiene el foco. Implementael patrón Chainable.
101     * @return {MM} Al ser Chainable devuelve this (MM).
102     * @method borrar
103     * @memberof MM
104     * @instance
105     */
106    mm.borrar = function () {
107        if ( this.arbol === this.foco ) {
108            this.nuevo();
```

```
109             return;
110         }
111 
112         var borrar = this.foco;
113         this.padre();
114         this.arbol.borrar ( borrar.elemento.id );
115         this.undoManager.add(new MM.comandos.Borrar(this.foco, borrar));
116         this.eventos.on ( 'borrar', this.foco, borrar );
117         borrar = null;
118     }.chain();
119 
120     /**
121      * @desc Cambia el foco a primer hijo del nodo que tiene actualmente el foco.
122      * @return {MM} Al ser Chainable devuelve this (MM).
123      * @method next
124      * @memberof MM
125      * @instance
126      */
127     mm.next = function () {
128         if ( this.foco.ordenNodo() !== 0 ) {
129             this.eventos.on ( 'next', this.foco, this.foco.hijos[0] );
130             this.ponerFoco ( this.foco.hijos[0] );
131         }
132     }.chain();
133 
134     /**
135      * @desc Cambia el foco al padre del nodo activo.
136      * @return {MM} Al ser Chainable devuelve this (MM).
137      * @method padre
138      * @memberof MM
139      * @instance
140      */
141     mm.padre = function () {
142         if ( !this.foco ) { return; }
143         var padre = this.arbol.padreDe ( this.foco.elemento.id );
144         if ( padre !== null ) {
145             this.eventos.on ( 'padre', this.foco, padre );
146             this.ponerFoco ( padre );
147         }
148         padre = null;
149     }.chain();
150 
151     /**
152      * @desc Cambia el foco al siguiente hermano del nodo actual. Si llega al último
153      *       siguiente hermano se entiende que es el primero
154      * @return {MM} Al ser Chainable devuelve this (MM).
155      * @method nextHermano
156      * @memberof MM
157      * @instance
158      */
159     mm.nextHermano = function () {
160         var padre = this.arbol.padreDe ( this.foco.elemento.id );
161 
162         if ( padre === null ) { return; }
163 
164         for ( var i = 0; i < padre.hijos.length; i++ ) {
165             if ( padre.hijos[i].elementEqual ( this.foco.elemento.id ) ) {
166                 if ( i === padre.hijos.length - 1 ) {
167                     this.eventos.on ( 'nextHermano', this.foco, padre.hijos[0] );
168                     this.ponerFoco ( padre.hijos[0] );
169                 } else {
170                     this.eventos.on ( 'nextHermano', this.foco, padre.hijos[i + 1] );
171                     this.ponerFoco ( padre.hijos[i + 1] );
172                 }
173                 break;
174             }
175         }
176         padre = null;
177     }.chain();
178 
179     /**
180      * @desc Cambia el foco al hermano anterior del nodo actual. Si llega al primero
181      *       en la siguiente llamada pasará al último de los hermanos.
182      * @return {MM} Al ser Chainable devuelve this (MM).
183      * @method prevHermano
184      * @memberof MM
185      * @instance
186      */
187     mm.prevHermano = function () {
188         var padre = this.arbol.padreDe ( this.foco.elemento.id );
189 
190         if ( padre === null ) { return; }
```

```
191
192         for ( var i = 0; i < padre.hijos.length; i++ ) {
193             if ( padre.hijos[i].elementEqual ( this.foco.elemento.id ) ) {
194                 if ( i === 0 ) {
195                     this.eventos.on ( 'prevHermano', this.foco, padre.hijos[padre.hijos.
                            length - 1] );
196                     this.ponerFoco ( padre.hijos[padre.hijos.length - 1] );
197                 } else {
198                     this.eventos.on ( 'prevHermano', this.foco, padre.hijos[i - 1] );
199                     this.ponerFoco ( padre.hijos[i - 1] );
200                 }
201                 return;
202             }
203         }
204         padre = null;
205     }.chain();
206
207     /**
208      * @desc Cambia el foco al último hermano
209      * @return {MM} Al ser Chainable devuelve this (MM).
210      * @method lastHermano
211      * @memberof MM
212      * @instance
213      */
214     mm.lastHermano = function () {
215         var padre = this.arbol.padreDe ( this.foco.elemento.id );
216
217         if ( padre === null ) { return; }
218
219         if ( padre.hijos.length >= 1 ) {
220             this.ponerFoco ( padre.hijos[padre.hijos.length - 1] );
221         }
222         padre = null;
223     }.chain();
224
225
226     /**
227      * @desc Pasa el foco al elemento raiz (Idea central).
228      * @return {MM} Al ser Chainable devuelve this (MM).
229      * @method root
230      * @memberof MM
231      * @instance
232      */
233     mm.root = function () {
234         this.eventos.on ( 'root', this.foco, this.arbol );
235         this.ponerFoco ( this.arbol );
236     }.chain();
237
238
239     /**
240      * @desc Pone el foco en nodo (subárbol) dado.
241      * @param {MM.Arbol} arbol Subárbol (nodo) donde poner el foco.
242      * @method ponerFoco
243      * @memberof MM
244      * @instance
245      */
246     mm.ponerFoco = function ( arbol ) {
247         this.eventos.on ( 'ponerFoco', this.foco, arbol );
248         this.foco = arbol;
249     };
250
251     mm.nuevo( "Idea Central" );
252
253     /**
254      * @prop {MM.Render} render Instancia de MM.Render. El valor por defecto es null
255      *                          y se crea en el momento de renderizar el árbol.
256      * @memberof MM
257      * @inner
258      */
259     mm.render = null;
260
261     /**
262      * @desc Realiza el renderizado del Mapa mental. El renderizado se realiza ajustando
            el escenario al contenedor.
263      *       Una vez llamada a esta función queda establecido el valor de la propiedad MM
            .render.
264      * @param {Element}                contenedor  Elemento del árbol DOM que contendrá
            el Mapa mental.
265      * @param {MM.NodoSimple|MM.Globo} claseNodo   Clase de renderizado de nodo
266      * @param {MM.Arista|MM.Rama}      claseArista Clase de renderizado de aristas
267      * @method renderizar
268      * @memberof MM
```

```
269         * @instance
270         */
271        mm.renderizar = function ( contenedor, claseNodo, claseArista ) {
272            mm.render = new MM.Render ( contenedor, claseNodo, claseArista );
273            mm.render.renderizar();
274        };
275
276        /**
277         * @desc Marca el nodo actual (foco) como plegado, si no establa plegado o como
278         *       desplegado si estaba plegado.
279         * @param {Boolean} plegado Si es true fuerza el plegado y si es false el desplegado
280         * @method plegadoRama
281         * @memberof MM
282         * @instance
283         */
284        mm.plegarRama = function (plegado, undo) {
285            //   - PLEGADO:      Se pliega toda la herencia del nodo.
286            //   - DESPLEGADO:   Se despliega sólo el nodo en cuestión.
287
288            plegado = plegado || !this.foco.elemento.plegado;
289            this.foco.elemento.plegado = plegado;
290            var plegar = function (a) {
291                a.hijos.forEach(function (h) {
292                    h.elemento.plegado = true;
293                    plegar(h);
294                });
295            };
296            var desplegar = function (a) {
297                var aPlegado = a.elemento.plegado;
298                a.hijos.forEach(function (h) {
299                    h.elemento.plegado = ( !aPlegado && h.esHoja() )?false:h.elemento.plegado
                        ;
300                    desplegar(h);
301                });
302                aPlegado = null;
303            };
304
305            if ( plegado ) {
306                plegar(this.foco);
307            } else {
308                desplegar(this.foco);
309            }
310            this.render.dibujar();
311            if ( !undo ) {
312                this.undoManager.add(new MM.comandos.Plegar(this.foco, plegado));
313            }
314        };
315
316
317        /**
318         * @desc Abre un cuadro de dialogo para seleccionar el fichero FreeMind que deseamos
              abrir.
319         *       Lo carga y redendiza el nuevo Mapa mental una vez terminado la carga.
320         * @method cargarFreeMind
321         * @memberof MM
322         * @instance
323         */
324        mm.cargarFreeMind = function () {
325            var importer = new MM.importar.FreeMind();
326
327            var susR = MM.importar.evento.suscribir("freeMind/raiz", function () {
328                MM.render.desuscribrirEventos();
329            });
330            var susP = MM.importar.evento.suscribir("freeMind/procesado", function () {
331                MM.render.renderizar();
332            });
333
334            var input = MM.DOM.create('input', {
335                'type' : 'file',
336                'id'   : 'ficheros'
337            });
338            input.addEventListener("change", function(evt) {
339                if ( input.files.length !== 0 ) {
340                    importer.cargar(input.files[0]);
341                }
342            }, false);
343            input.click();
344
345        };
346
347        return mm;
348    }(MM);
```

Como se puede observar se ha utilizado incorporado una pequeña variación con respecto al módulo propuesto por Douglas Crockford. Se trata de un modulo extensible. Para ello, el espacio de nombre del módulo debe estar previamente definido y posteriormente se le pasa a la función auto-ejecutable para que extienda su interfaz. Un esquema de este módulo es:

```
1  // El espacio de nombres en este ejemplo es la variable "modulo"
2  var modulo = {};
3
4  modulo = function (m) {
5    // variables privadas
6    var p1, p2;
7
8    // funciones privadas
9    function privado() {
10   }
11
12   m.variablePublica = null;
13
14   m.funcionPublica = function () {};
15
16   return m;
17 }(modulo);
18
19 modulo = function (m) { // extesion del modulo
20   // variables privadas solo accesibles en la extension
21   var p1_ext, p2_ext;
22
23   // funciones privadas
24   function privado_ext() {
25   }
26
27   m.variablePublica_ext = null;
28
29   m.funcionPublica_ext = function () {};
30
31   return m;
32 }(modulo);
```

Quedando el interfaz de nuestro módulo como la conjunción de los métodos y variables públicas definida en cada una de la extensiones.

5.1.6. Implementación de MM.Class con prototipos

* ya se ha hablado del él * implementación de clase * Constructor por defecto init * sobreescritura de métodos. * Se puede extender

```
1  /**
2   * @file klass.js Implementación de Classes
3   * @author José Luis Molina Soria
4   * @version 20130224
5   */
6
7  if ( typeof module !== 'undefined' ) {
8      var MM = require('./MindMapJS.js');
9  }
10
11 /**
12  * @class MM.Class
13  * @classdesc Clase base.
14  * @constructor MM.Class
15  */
16
17 MM.Class = function (){
```

```
18      this.init = function () {};
19  };
20
21
22  /**
23   * @desc Función que nos permite extender sobre una clase existente
24   * @param {object} prop Clase que deseamos extender.
25   * @return {Class} una nueva clase. Clase hija hereda los métodos y propiedades de la
         clase padre.
26   */
27  MM.Class.extend = function(prop) {
28      var _super = this.prototype || MM.Class.prototype; // prototype de la clase padre
29
30      function F() {}
31      F.prototype = _super;
32      var proto = new F();
33      var wrapperMetodo = function(name, fn) { // asociamos las funciones al nuevo contexto
34          return function() {
35              var tmp = this._super;                  // guardamos _super
36              this._super = _super[name];             // función super => podemos hacer this.
                    _super(argumentos)
37              var ret = fn.apply(this, arguments); // ejecutamos el método en el contexto
                    de la nueva instancia
38              this._super = tmp;                      // restauramos el _super
39              return ret;
40          };
41      };
42
43      // recorremos el objeto que nos han pasado como parámetro...
44      for (var name in prop) {
45          // Si estamos sobreescribiendo un método de la clase padre.
46          if (typeof prop[name] === "function" && typeof _super[name] === "function") {
47              proto[name] = wrapperMetodo(name, prop[name]);
48          } else { // no sobreescribimos métodos ni p
49              proto[name] = prop[name];
50          }
51      }
52
53      function Klass() {
54          if (this.init) {
55              this.init.apply(this, arguments);
56    }
57      }
58
59      Klass.prototype = proto;
60      Klass.prototype.constructor = Klass;
61      Klass.extend = this.extend;
62
63      return Klass;
64  };
65
66  /**
67   * @desc Permite especificar un contexto concreto a una función dada
68   * @param {object} ctx Contexto en que desea asociar a la función
69   * @param {function} fn Función a la que le vamos a realizar el bind
70   * @return {function} nueva función asociada al contexto dado.
71   */
72  MM.Class.bind = function (ctx, fn) {
73      return function() {
74          return fn.apply(ctx, arguments);
75      };
76  };
77
78  if ( typeof module !== 'undefined' ) {
79      module.exports = MM.Class;
80  }
```

explicar la implementación del clases realizada en la aplicación incluir el klass.js y los test realizados para su verificación.

ambito privado y público

5.1.7. KineticJS

5.2. NodeJS

Basado en la máquina virtual JavaScript V8 de Google, NodeJS[16] a supuesto una revolución en el mundo de la programación JavaScript, dando un salto de gigante desde el lado del cliente al servidor. Este enorme evolución, y de manos de V8, ha provocado la creación de un entorno de programación completo, en el cual se aglutina desde un REPL[17] para pruebas y depuración interactiva hasta un gestor de paquetes y librerías NPM[18] (Node Packaged Modules).

Figura 5.2: Logo NodeJS

NodeJS nos permite crear aplicaciones de red escalables, alcanzando un alto rendimiento utilizando entrada/salida no bloqueante y un bucle de eventos en una sóla hebra. Es decir, que NodeJS se programa sobre un sólo hijo de ejecución y en el caso de que necesite operaciones de entrada/salida, creará una segunda hebra para evitar su bloqueo. En teoría NodeJS puede mantener tantas conexiones simultaneas abiertas como descriptores de fichero soporte el sistema operativo (en UNIX aproximadamente 65.000), en la realidad son bastantes menos (se calcula que entre 20.000 y 25.000).

Como ya se ha mencionado, y debido a que su arquitectura es usar un único hilo, sólo puede unas una CPU. Es el principal inconveniente que presenta la arquitectura de NodeJS.

Sus principales objetivos son:

- Escribir aplicaciones eficientes en entrada y salida con un lenguaje dinámico.
- Soporte a miles de conexiones.
- Evitar las complicaciones de la programación paralela (Concurrencia vs paralelismo).
- Aplicaciones basadas en eventos y callbacks.

[16] La web oficial de NodeJS es nodejs.org
[17] Patrón Read-Eval-Print Loop
[18] La web oficial de NPM es npmjs.org

5.2.1. Instalación de NodeJS

Existen varias formas de instalar NodeJS, por ejemplo, utilizando los repositorios del sistema operativo o instaladores. En mi caso, he utilizado la compilación del código fuente que esta alojado en GitHub[19].

Lo primero que tenemos debemos hacer es clonar el proyecto.

```
$ git clone git://github.com/joyent/node.git
$ cd node
```

Una vez tengamos la copia del código fuente realizaremos un checkout de una versión estable.

```
$ git branch vXXXX Nombre
$ git checkout Nombre
```

Ahora, ya estamos en disposición de compilar el fuente de la versión estable.

```
$ ./configure --prefix=/usr/local
$ sudo make install
```

5.2.2. Instalación del NPM

Como ya se ha comentado antes NPM[20] es el gestor de paquetes de NodeJS. En la versiones actuales ya viene instalado, pero eso no fue siempre así. También se puede optar por instalarse de sin NodeJS. Para ello, ejecutaremos el siguiente comando:

```
$ curl https://npmjs.org/install.sh | sh
```

5.2.3. Uso básico de NPM

Iniciar un proyecto nuevo

A continuación se muestra la secuencia de comandos necesaria para crear un proyecto.

```
$ mkdir hola
$ cd hola
$ npm init
npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sane defaults.

See ‘npm help json‘ for definitive documentation on these fields
and exactly what they do.

Use ‘npm install <pkg> --save‘ afterwards to install a package and
save it as a dependency in the package.json file.
```

[19] Repositorio de NodeJS https://github.com/joyent/node
[20] Node Packaged Module (NPM) web oficial npmjs.org

```
Press ^C at any time to quit.
name: (hola)
version: (0.0.0)
git repository:
author:
license: (BSD-2-Clause)
About to write to /tmp/hola/package.json:

{
  "name": "hola",
  "version": "0.0.0",
  "description": "Hola mundo",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "hola",
    "mundo"
  ],
  "author": "",
  "license": "BSD-2-Clause"
}

Is this ok? (yes)
```

El comando **npm init** comenzará a realizarnos sobre los datos del proyecto como nombre, versión, etc. Una vez terminado, tendremos nuestro fichero de configuración (package.json) preparado.

Buscar paquetes y obtener información

El primer comando nos permite buscar paquetes interesantes o útiles a nuestro proyecto, y el segundo, para obtener una descripción más exhaustiva del mismo.

```
$ npm search <palabra>:
$ npm info <paquete>
```

Instalación de paquetes

Existen varias formas para instalar un paquete y/o librería.

De forma global [21] para que lo puedan utilizar todas las librerías del sistema.

```
$ npm install <paquete> -g
```

De forma local[22], es decir, sólo se podrá utilizar el proyecto actual.

```
$ npm install <package name>
```

También existen dos modificadores muy interesantes *–save* para que se incluya (en el fichero package.json) la librería o paquete como dependencia del proyecto. Y el otro

[21] Con el modificador -g
[22] Sin el modificador -g

modificador es *–save-dev* para que la dependencia sea de desarrollo. Así quedaría un fichero package.json después de haber incluido un paquete (colors) como dependencia y otro (grunt-cli) como dependencia de desarrollo.

```
{
  "name": "hola",
  "version": "0.0.0",
  "description": "Hola mundo",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "hola",
    "mundo"
  ],
  "author": "",
  "license": "BSD-2-Clause",
  "dependencies": {
    "colors": "~0.6.2"
  },
  "devDependencies": {
    "grunt-cli": "~0.1.9"
  }
}
```

Desinstalación de paquetes

Las instrucciones son la misma salvo por que el comando *install* se sustituye por *uninstall*.

5.3. GruntJs

Se trata de una aplicación Node que está empaquetada y disponible en NPM. GruntJS es una herramienta versátil para la automatización de tareas mediante Javascript, evitándonos dentro de lo posible la realización de tareas repetitivas. Con un simple archivo de configuración nos permite realizar tareas tan diversas como minificar código, lanzar la suite de tests, etc.

5.3.1. Características

- **Acceso a archivos:** No tenemos que preocuparnos del acceso a archivos, sólo tratarlos.
- **Automatización de tareas y conjunto de tareas:** Podemos automatizar pequeñas tareas o mediante un conjunto de ellas automatizar tareas más complejas como la comprensión de una librería Javascript.

- **Fácil instalación:** Esta en NPM, la instalación es simplemente un npm install.
- **Plugins comunitarios:** Existe un gran comunidad detrás creando plugins, que podemos utilizar utilizando NPM.
- **Multi-plataforma:** Al ser una librería Node nos permite utilizarlo en cualquier plataforma que soporte Node.

5.3.2. Instalación

La instalación de GruntJS no tiene complicación, ya que, al tratarse de una apliación Node y estar publicado en NPM sólo necesitamos como prerrequisito tener instalado Node y NPM.

Lo primero es instalar el cliente de forma global con el comando:

```
$ npm install grunt-cli -g
```

Y una vez instalado el cliente, en nuestro proyecto debemos ejecutar:

```
$ npm install grunt --save-dev
```

Figura 5.3: Logo GruntJS

Ya tenemos agregado GruntJS a nuestro proyecto. Con los –save-dev le indicamos al NPM que lo añada a las dependencias del proyecto para desarrollo. Así inlcuirá las líneas pertinentes en nuestro fichero package.json.

```
1  {
2    "name": "nombre",
3    "version": "0.0.1",
4    "dependencies": {
5
6    },
7    "devDependencies": {
8      "grunt": "~0.4.1"
9    }
10 }
```

5.3.3. Creando el Gruntfile

En el fichero Gruntfile.js será donde definamos las tareas que deseamos en nuestro proyecto. El esquema de fichero es:

```
module.exports = function(grunt) {

  grunt.registerTask('default', 'Tarea Hola Mundo', function() {
    grunt.log.write('Hola Mundo!').ok();
  });

};
```

Como se puede observar se trata de un modulo Node, que será llamado por grunt cuando lo ejecutemos. En el ejemplo, le hemos regristrado una tarea por defecto que imprime "Hola Mundo!". Ahora sólo tenemos que ejecutar el comando grunt para ver el resultado de nuestra tarea.

GruntJS tiene un conjunto básico de plugins, nombrados grunt-contrib-XXXX, empaquetados en NPM y que podemos instalar fácilmente.

5.3.4. Gruntfile.js de MindMapJS

El fichero de configuración de GruntJS utilizado para el proyecto es :

```
module.exports = function(grunt) {
    var config = {
  pkg: grunt.file.readJSON('package.json'),

  concat: {
      options: {
    separator: ';'
      },
      source: {
    src:  ['src/*.js'],
    dest: 'dist/<%= pkg.name %>-v<%= pkg.version %>.js'
      }
  },

  replace: {
      dev: {
    options: {
        variables: {
      version: '<%= pkg.version %>',
      date: '<%= grunt.template.today("yyyy-mm-dd") %>'
        },
        prefix: '@@'
    },

    files: [{
        src: ['dist/<%= pkg.name %>-v<%= pkg.version %>.js'],
        dest: 'dist/<%= pkg.name %>-v<%= pkg.version %>.js'
    }]
      },
      prod: {
    options: {
        variables: {
      version: '<%= pkg.version %>'
        },
        prefix: '@@'
    },
    files: [{
        src: ['dist/<%= pkg.name %>-v<%= pkg.version %>.min.js'],
        dest: 'dist/<%= pkg.name %>-v<%= pkg.version %>.min.js'
    }]
      }
  },

  uglify: {
      options: {
```

```
46      banner: '/*! <%= pkg.name %> v<%= pkg.version %> <%= grunt.template.today("yyyy-mm-dd
            ") %> Por José Luis Molina Soria */\n'
47        },
48        build: {
49      files: {
50          'dist/<%= pkg.name %>-v<%= pkg.version %>.min.js': 'dist/<%= pkg.name %>-v<%= pkg
                .version %>.js'
51      }
52        }
53    },
54
55    clean: {
56        build: ['dist/*']
57    },
58
59    jshint: {
60        options: {
61      laxbreak: true,
62      curly: true,
63      eqnull: true,
64      eqeqeq: true,
65      undef: true,
66      browser: true,
67  //    immed: true,
68      latedef: true,
69      newcap: true,
70      noarg: true,
71      sub: true,
72      boss: true,
73      globals: {
74          console: true,
75          window : true,
76          module: true,
77          MM : true,
78          Kinetic : true,
79          require : true,
80          ActiveXObject : true,
81          FileReader : true,
82          DOMParser : true,
83          Blob : true,
84          alert : true
85      }
86        },
87        all : ['src/*.js']
88    },
89
90    jsdoc : {
91              dist : {
92      src: ['src/*.js'],
93      options: {
94                      destination: 'docs/jsdocs/'
95      }
96              }
97    },
98
99    mochaTest: {
100       test: {
101     options: {
102         reporter: 'spec',
103         require: 'should'
104     },
105     src: ['test/**/*-test.js']
106       }
107   }
108
109     };
110
111     grunt.initConfig(config);
112
113     // Load plugins
114     grunt.loadNpmTasks('grunt-contrib-concat');
115     grunt.loadNpmTasks('grunt-replace');
116     grunt.loadNpmTasks('grunt-contrib-uglify');
117     grunt.loadNpmTasks('grunt-contrib-clean');
118     grunt.loadNpmTasks('grunt-contrib-jshint');
119     grunt.loadNpmTasks('grunt-jsdoc');
120     grunt.loadNpmTasks('grunt-mocha-test');
121
122     // Tasks
123     grunt.registerTask('dev', ['clean', 'concat:source', 'replace:dev']);
124     grunt.registerTask('full', ['clean', 'concat:source', 'replace:dev', 'uglify', '
            replace:prod']);
```

```
125     grunt.registerTask('test', ['mochaTest']);
126     grunt.registerTask('hint', ['jshint']);
127     grunt.registerTask('jsdoc', ['jsdoc']); // no funca :(( utilizar el script "jsdoc.sh"
128 };
```

Como se puede comprobar se han incorporados distintos plugins:

- **grunt-contrib-concat:** permite concatenar un conjunto de ficheros en nuestro caso los ficheros JavaScripts.
- **grunt-replace:** plugins para realizar operaciones de reemplazo dentro de un conjunto de ficheros.
- **grunt-contrib-uglify:** para comprimir y/o minimizar el código JavaScripts.
- **grunt-contrib-clean:** borrar un conjunto de ficheros o el contenido de un directorio.
- **grunt-contrib-jshint:** permite reliazar la verificación y validación de buenas prácticas establecidas en JavaScripts.
- **grunt-jsdoc:** compilar los comentarios JSDocs para generarla documentación HTML del API.
- **grunt-mocha-test:** tarea que lanza la suite de tests unitarios del proyecto.

Con estos plugins se han cubierto todas las necesidades de automatización de tareas del proyecto. Las tareas implementadas son:

- **dev:** que concatena el código fuente y realiza los reemplazo como fechas, versión, etc ...
- **full:** además de realizar las tareas propias de la tarea 'dev', minimiza y realiza los reemplazos de producción.
- **test:** lanza la suite de test
- **hint:** lanza la terea de validación de código JSHint.
- **jsdoc:** genera la documentación del API.

5.4. Github

Todo proyecto que se precie debe estar sustentado con sistema de control de versiones, en nuestro caso ha sido Git[23]. Más concretamente se trata de un sistema distribuido de control de código fuente o SCM[24] creado por Linus Torvalds, a partir, de su propia experiencia en el desarrollo de los kernels de Linux.

Github[25] es una plataforma online pensada para el desarrollo colaborativo de proyectos, utilizando para ello Git. Github nos permite almacenar de forma pública[26] nuestro código fuente, promoviendo el trabajo colaborativo entre profesionales. Así pues, otro profesional ajeno al proyecto puede solicitar cambios sugerir mejoras o reportar bugs.

Figura 5.4: Mascota de Github

De las características mas resaltables de Github para el control de versiones, podemos enumerar las siguientes:

- **Wiki para el proyecto**, con el principal propósito de documentar nuestro proyecto Github nos proporciona una Wiki.
- **Gráficas**, tiene un conjunto de gráficas detalladas para determinar el avance del proyecto y el progreso de cada colaborador del proyecto.
- **Página web del proyecto**, para presentar nuestro poryecto y/o repositorio

Como sistemas de colaboración entre programadores tenemos el:

- **Fork**, con un fork podemos clonar un repositorio para realizar cambios que necesitemos, de forma que podamos adaptar el proyecto a nuestras necesidades

[23] Web oficial de Git es git-scm.com
[24] SCM (Source Code Management)
[25] La web de Github es github.com
[26] Github permite crear proyectos privados con cuentas de pago

concretas. Un fork nos permite colaborar con el proyecto original mediantes los pull requests.

- **Pull requests**, una vez realizados los cambios, y si lo vemos oportuno, podemos reportar las variaciones al proyecto original mediante un pull request. El pull request pueden ser cambios, mejoras en la funcionalidad, y/o correcciones, que deberá aprobar él/los programadores del proyecto original.

5.4.1. Crear el repositorio

Previo a la creación del repositorio debemos crearnos una cuenta de usuario en Github. una vez realizado, sólo debemos pulsar la opción de "new repository". Ahora, ya tenemos repositorio pero debemos dotarlo de contenido, y para ello, y desde una consola local realizaremos:

- Creamos el directorio del proyecto.

```
$ mkdir ~/proyecto
$ cd proyecto
```

- Iniciamos el repositorio git

```
$ git init
```

- Creamos el fichero README.md. Se trata de un fichero con formato markdown[27] en el cual hay que introducir un descripción del proyecto. Este fichero se visualizará en le página principal del repositorio.

- Añadimos y confirmamos los cambios.

```
$ git add .
$ git commit -m 'primer commit'
```

- Cambiamos el remote origin a la ruta de nuestro repositorio.

```
$ git remote add origin https://github.com/usuario/proyecto.git
```

- subimos los cambios al repositorio

```
$ git push origin master
```

[27] http://es.wikipedia.org/wiki/Markdown

5.4.2. Fork/Pull request

Crear un fork de un proyecto utilizando Github es trivial. Tan sólo hay que ir al proyecto en cuestión y pulsar el botón de fork. Github crea una copia del proyecto de forma que si el proyecto original tiene la url https://github.com/usuariOriginal/proyecto.git y la copia tendrá la url https://github.com/usuario/proyecto.git. Ahora ya estamos en disposición de trabajar clonando el repositorio:

```
$ git clone https://github.com/usuario/proyecto.git
```

Ya tenemos el repositorio listo para su uso. Si deseamos colaborar con el proyecto original debemos crear una rama[28], realizar los cambios y subirlos[29] a nuestro fork de Github. Desde Github procede realizar la revisión de los cambios y pulsar sobre la opción de çreate a pull request for this comparison".

5.5. JSDoc

Tan importante como el código es la documentación del mismo, JSDoc[30] es una herramienta inspirada en Javadoc[31] pero pensada para Javascript.

Mediante una conjunto de etiquetas (@class, @function, etc) introducidas como comentarios del código fuente, se generará la documentación en formato HTML[32]. Todos los desarrolladores que alguna vez hemos programado en Java y generado documentación de nuestro código, en JavaDoc, estamos familiarizados con el mecanismo de etiquetas, por lo que resulta muy intuitivo la elaboración de la documentación.

En la figura 5.5 se puede ver un ejemplo de uso de las etiquetas en el código fuente. En concreto de una clase PubSub del propio proyecto MindMapJS. Se puede observar claramente, como se usan etiquetas como @author, @versión, @constrcutor, @class, etc ...

En la figura 5.6 tenemos el resultado de compilar el código fuente con JSDoc. El resultado es un HTML que podemos retocar y configurar, permitiendo tener una Wiki, vistosa y

[28] Operaciones a realizar: branch y checkout
[29] Operaciones a realizar: commit y push
[30] Url del proyecto https://github.com/jsdoc3/jsdoc
[31] http://es.wikipedia.org/wiki/Javadoc
[32] Por lo general, se genera HTML pero permite otros formatos como RTF.

```
/**
 * @file pubsub.js Implementación del patrón Publish/Subscribe
 * @author José Luis Molina Soria
 * @version 20130227
 */

if ( typeof module !== 'undefined' ) {
    var MM = require('./MindMapJS.js');
    MM.Class = require('./klass.js');
}

/**
 * @class MM.PubSub
 * @classdesc Implementación del patrón Publish/Subscribe
 * @constructor MM.PubSub
 */
MM.PubSub = MM.Class.extend(/** @lends MM.PubSub.prototype */{

    eventos : {},

    idSus : 1,

    init : function () {
        this.eventos = {};
        this.idSus = 1;
    },

    /**
     * @desc Realiza la notificación a los suscriptores de que se a producido
     * una publicación o evento.
     * @param evento {string}    nombre del evento o publicación a notificar
     * @param args   {*}         argumentos para la función callback
     * @return {boolean} Si el evento no es un nombre valido retorna false en
     * otro caso retorna true
     */
    on : function( evento ) {
        if (!this.eventos[evento]) {
            return false;
```

Figura 5.5: Ejemplo de código fuente documentado con JSDoc

funcional, de la documentación de nuestro cógido fuente.

5.6. Mocha

Mocha[33] es un framework Javascript para realizar pruebas unitarias. Sus creadores lo definen como:

> *Mocha is a feature-rich JavaScript test framework running on node.js and the browser, making asynchronous testing simple and fun. Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases. Hosted on GitHub.*

Y sinceramente, creo que la definición no puede ser más acertada. Permite crear test con relativa facilidad y con una sintaxis clara y concisa. De puedo decir, que es "simple, flexible y divertido"[34].

[33] Página oficial de Mocha: http://visionmedia.github.io/mocha/

[34] En la cabecera de su web podemos leer "mocha simple, flexible, fun"

Figura 5.6: Página generada por JSDoc

5.6.1. Características

Entre sus características más destacables están:

- **Soporte para NodeJs.** No sólo soporta el uso con NodeJS sino que esta empaquetado para NPM, por lo que, la instalación y la puesta en marcha resulta muy, muy sencilla. También existen plugins para utilizarlo con GruntJs.
- **Soporte para diferentes navegadores.** Por lo que, podemos probar nuestro interface en el navegador y verificar su correcto funcionamiento.
- **Informes.** Tiene opciones para generar informes en varios formatos dependiendo de las necesidades.
- **Uso de cualquier librería de afirmaciones(Assertions).** Existe principalmente cuatro librerías que pueden ser utilizadas con Mocha Chai, Should, Expect y Better-Assert.
- **Soporte para test síncronos y asíncronos.** Permite abarcar todas las necesidades de nuestro código.

5.6.2. Ejemplo

He aquí un simple ejemplo de uso de mocha.

```
1 var assert = require("assert");
2 describe('Array', function(){
3   describe('#indexOf()', function(){
4     it('debe retorna -1 si el valor no esta presente', function(){
5       assert.equal(-1, [1,2,3].indexOf(5));
6       assert.equal(-1, [1,2,3].indexOf(0));
7     });
8   });
9 });
```

Figura 5.7: Mocha

Para empezar, debe realizar importar la librería de afirmaciones (assertions) que vamos a utilizar en el presente ejemplo se ha utilizado assert[35].

Las líneas 2 y 3 describen a su vez, un módulo y submódulo de ejecución. En nuestro caso el módulo lo hemos llamado .ªrrayz el submódulo de ejecución ïndexOf()". Conviene ser descriptivos en los nombres de los módulos y submódulos.

La línea 4 describe una prueba unitaria a la que le asociamos una función anónima con las afirmaciones necesaria.

Una vez escrita la prueba unitaria ejecutamos el siguiente comando "mocha -R *.js" obtenemos el resultado que podemos observar en la figura 5.8.

```
Array
  #indexOf()
    ✓ debe retorna -1 si el valor no esta presente


1 test complete (3 ms)
```

Figura 5.8: Resultado de ejecutar mocha -R spec *.js

Existe una función especial, que podemos observar en la línea 5 del siguiente código. Esta función especial es "beforeEach", que tiene la misión de ejecutarse antes de cada test unitario. Nuestro ejemplo podemos ver que la hemos utilizado para inicializar variables.

```
1 var assert = require("assert");
2
3 var a;
4
5 beforeEach(function(){
6    a = [1,2,3];
7 });
```

[35] En MindMapJS se ha utilizado Should y Chai

```
8
9
10 describe('Array', function(){
11   describe('#indexOf()', function(){
12     it('debe retorna -1 si el valor no esta presente', function(){
13       assert.equal(-1, a.indexOf(5));
14       assert.equal(-1, a.indexOf(0));
15     });
16     it('debe retorna 1 si pedimos al primer valor', function(){
17       assert.equal(0, a.indexOf(1));
18     });
19   });
20 });
```

El resultado de ejecutar nuestro nuevo test es.

```
Array
  #indexOf()
    ✓ debe retorna -1 si el valor no esta presente
    ✓ debe retorna 1 si pedimos al primer valor


2 tests complete (4 ms)
```

Figura 5.9: Resultado de ejecutar mocha -R spec array1-test.js

Como se puede deducir, de los ejemplos anteriores, Mocha nos proporciona los mecanismos básicos para poder realizar test unitarios fáciles, rápidos y sobre todo sencillos.

Capítulo

6 Resultados y discusión

6.1. Resultados

En los Resultados (del trabajo) se deben analizar críticamente las características, bondades, limitaciones y defectos de lo implementado y/o de las tareas que se han seguido. Se pueden poner ejemplos de aplicación a distintos casos.

6.2. Discusión

En la Discusión se pueden justificar las limitaciones, compararlas con las de trabajos anteriores en el tema y analizar los productos obtenidos de la aplicación de nuestro trabajo.

6.3. uglify

6.4. jsHint

6.5. KineticJS

Capítulo

7

Manual de usuario

Capítulo

8

Conclusiones

Finalmente se presentarán breves Conclusiones a las que haya llegado el autor, así como posibles sugerencias y futuros desarrollos del tema tratado, indicando expresamente cuáles son las partes totalmente originales del trabajo, mayores esfuerzos, expectativas, interés suscitado personalmente y sus posibilidades en la comunidad científica.

Bibliografía