

UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

Editor de mapas mentales online con HTML5 y Javascripts

Realizado por

José Luis Molina Soria

Dirigido por

Juan Antonio Falgueras Cano

Departamento

Lenguajes y Ciencias de la Computación

Málaga, 18 de junio de 2014

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: _____

_____ Editor de mapas mentales online con HTML5 y Javascripts

del alumno/a D^o/D^a : José Luis Molina Soria

dirigido por D^o/D^a : Juan Antonio Falgueras Cano ,

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

1	Abstract	8
2	Motivación y objetivos	10
2.1	Motivación	10
2.2	Objetivos	12
3	Introducción	14
3.1	¿Qué es?	14
3.2	Aplicaciones y beneficios.	16
3.3	Partes de un mapa mental.	17
3.4	Elaboración.	19
4	Materiales y métodos	21
4.1	Metodología y etapas del desarrollo.	21
4.2	Casos de uso.	24
4.3	Diagramas de Clase	30
5	Implementación.	54
5.1	Javascripts.	54
5.2	Concatenación y UglifyJS	78
5.3	JsHint	80
5.4	KineticJS	80
5.5	NodeJS	88
5.6	GruntJs	92
5.7	Github	97
5.8	JSDoc	99

5.9	Mocha	100
6	Manual de usuario	105
6.1	Manual para el desarrollador	105
6.2	Manual de uso de MindMapJS	109
7	Resultados. Conclusiones	118
7.1	Resultados.	118
7.2	Conclusiones.	120
	Bibliografía	122

Índice de figuras

2.1	Esquema general de la especificación HTML5 a diciembre de 2011	11
3.1	Mapa mental de FreeMind	15
3.2	Partes de un mapa mental	17
3.3	Partes de un mapa mental considerando su contenido	18
3.4	Mapa mental de elaboración de mapas mentales	19
4.1	Versión inicial.	23
4.2	Versión inicial.	24
4.3	Casos de uso	25
4.4	Diagrama de secuencia nuevo	26
4.5	Diagrama de secuencia add	27
4.6	Diagrama de secuencia borrar	27
4.7	Diagrama de secuencia Zoom	28
4.8	Diagrama de secuencia navegación	29
4.9	Diagrama de secuencia plegar	31
4.10	Clase MM.Class	31
4.11	Diagrama de clases pubsub	32
4.12	Clase MM.PubSub	33
4.13	Diagrama de clases undo	33
4.14	Clase MM.UndoManager.ComandoHacerDeshacer	34
4.15	Secuencia de ejecución de UndoManager	35
4.16	Clase MM.UndoManager	36
4.17	Diagrama de clases MM	37
4.18	Clase MM	38
4.19	Clase MM.Arbol	39
4.20	Modulo MM.DOM	41

4.21	Modulo MM.Render	41
4.22	Diagrama de clases nodo	43
4.23	Clase MM.Mensaje	44
4.24	Clase MM.NodoSimple	45
4.25	Mapa mental con renderización de nodos simples	45
4.26	Clase MM.Globo	46
4.27	Mapa mental con renderización de nodos globo	46
4.28	Modulo MM.Color	47
4.29	Diagrama de clases aristas	47
4.30	Clase Kinetic Beizer	48
4.31	Clase MM.Arista	48
4.32	Clase MM.Rama	49
4.33	Diagrama de clases teclado	50
4.34	Modulo MM.teclado.atajos	51
4.35	Clase MM.teclado.tecla	52
4.36	Clase MM.teclado	53
4.37	Diagrama de secuencia teclado	53
5.1	Cadena prototípica de objetos	57
5.2	Secuencia de ejecución de UndoManager	71
5.3	Carga de ficheros Javascript en desarrollo	78
5.4	Carga de ficheros Javascript concatenado	79
5.5	Carga de ficheros Javascript comprimido	79
5.6	Escenario y modelo de capas KineticJS de MindMapJS	81
5.7	Ejemplo de Kinetic - uso básico de capas	82
5.8	Ejemplo de Kinetic - eventos	84
5.9	Ejemplo de Kinetic - custom shape	87
5.10	Logo NodeJS	88
5.11	Logo GruntJS	93
5.12	Mascota de Github	97
5.13	Ejemplo de código fuente documentado con JSDoc	100
5.14	Página generada por JSDoc	101
5.15	Mocha	102

5.16	Resultado de ejecutar mocha -R spec *.js	102
5.17	Resultado de ejecutar mocha -R spec array1-test.js	103
6.1	Estructura de directorios de MindMapJS	106
6.2	Resultado de ejecutar el comando 'grunt dev'	107
6.3	Resultado de ejecutar el comando 'grunt full'	107
6.4	Resultado de ejecutar el comando 'grunt hint'	108
6.5	Resultado de ejecutar el comando 'grunt test'	108
6.6	Página principal de MindMapJS	109
6.7	Menú ¿ Datos del proyecto	110
6.8	Barra de herramientas	111
6.9	Crear un nuevo mapa	111
6.10	Inserción de subideas.	112
6.11	Inserción de subideas hermana.	112
6.12	Editando una idea.	113
6.13	Mapa mental desplegado.	114
6.14	Mapa mental plegado.	114
6.15	Mapa mental con ampliado.	114
6.16	Mapa mental con reducido.	115

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 utilizó otras metodologías y paradigmas de programación como el BBD¹, 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.

¹Como sistema de pruebas y verificación del código fuente

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. HTML¹ 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 text2mindmap², MindMeister³, etc., tan versátiles o más que las propias aplicaciones nativas.

Hace ya tiempo que vio la luz los primeros borradores de HTML5⁴ (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.

¹Hypertext Markup Language

²<http://www.text2mindmap.com/>

³<http://www.mindmeister.com/es>

⁴El primer borrador de HTML5 fue publicado en 2008

HTML5 ha tomado en cuenta los defectos de su versión anterior⁵ y mejorar otras características como:

HTML5

Taxonomy & Status (December 2011)

- W3C Recommendation
- Candidate Recommendation
- Last Call
- Working Draft
- Non-W3C Specifications
- Deprecated W3C APIs

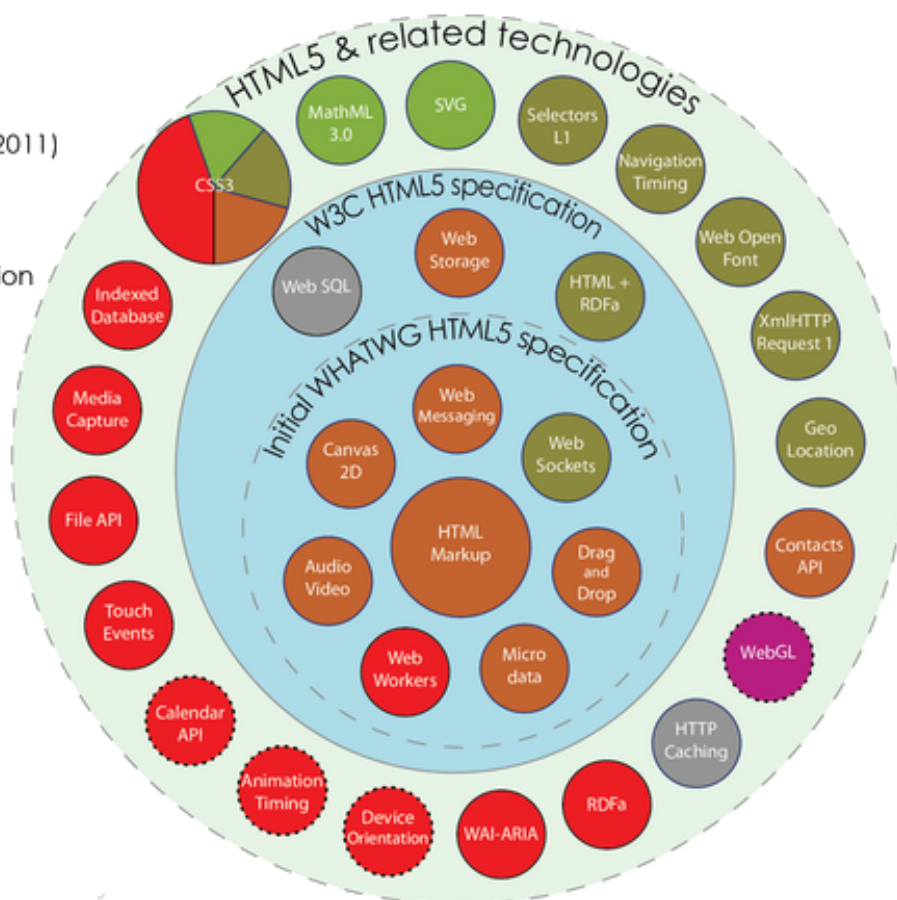


Figura 2.1: Esquema general de la especificación HTML5 a diciembre de 2011

- **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 canvas⁶.
- **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

⁵HTML 4.01

⁶En 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 Consortium⁷ (W3C) y la especificación EmacScript.

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.

⁷Web oficial de la W3C <http://www.w3.org/>

Introducción

3.1. ¿Qué es?

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

Por naturaleza, nuestro cerebro 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 irradiante¹. 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.

¹que irradia

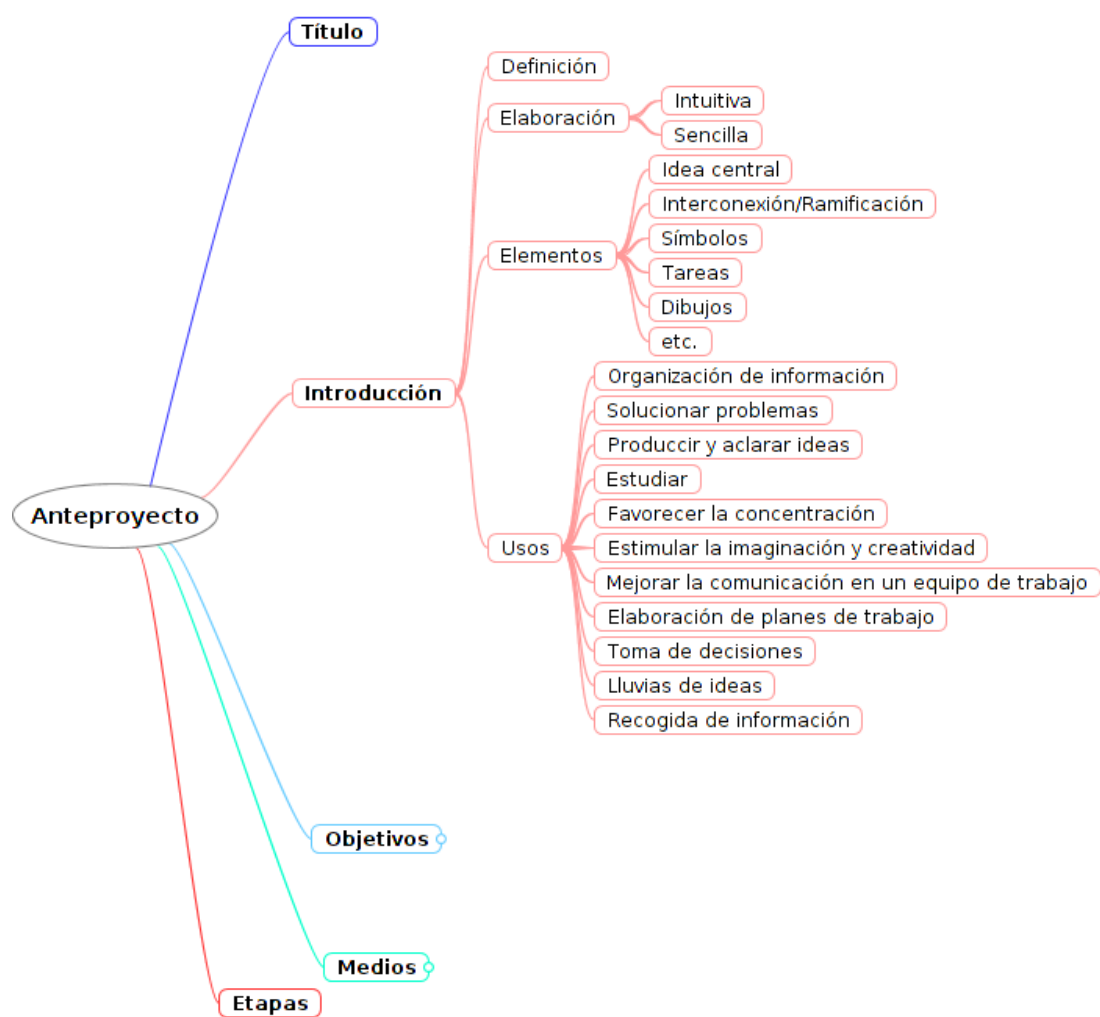


Figura 3.1: Mapa mental de FreeMind

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 web.

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 secundarias o asociada a otra idea.

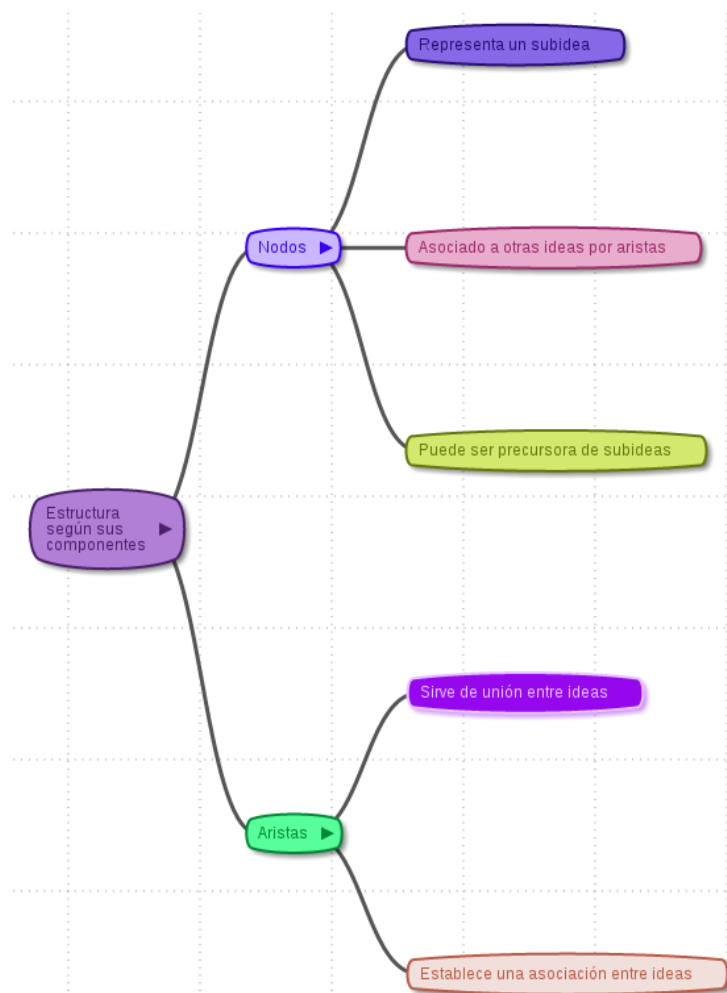


Figura 3.2: Partes de un mapa mental

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.

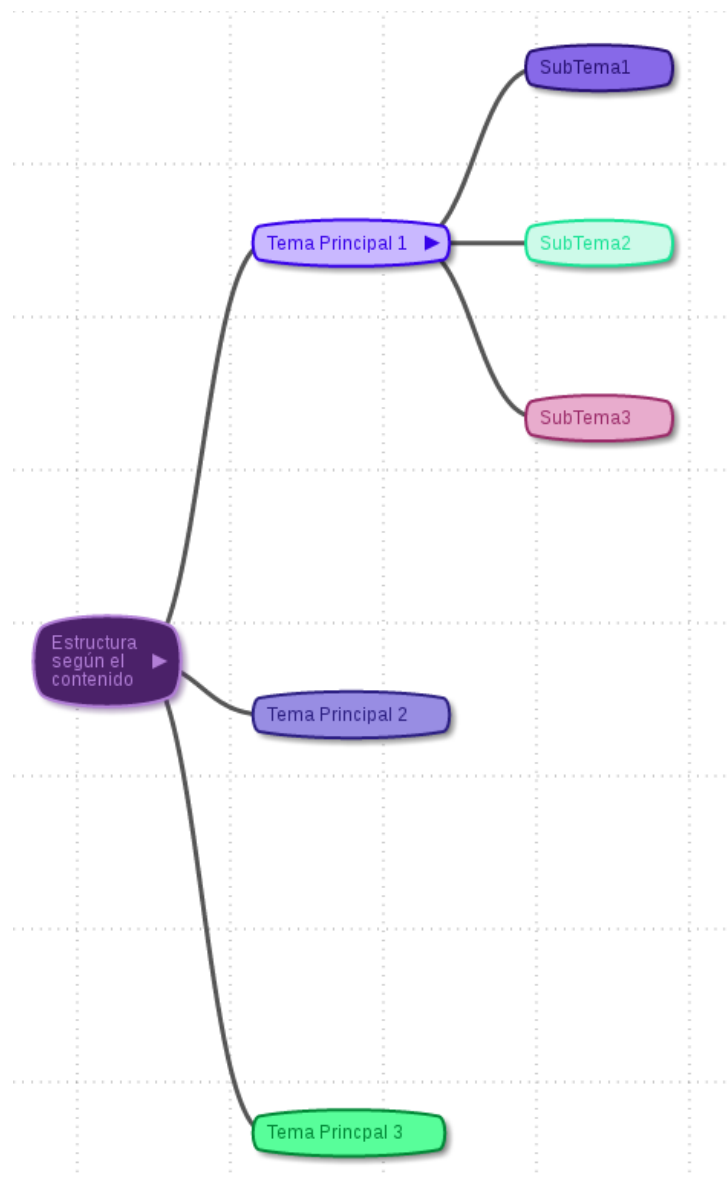


Figura 3.3: Partes de un mapa mental considerando su contenido

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.

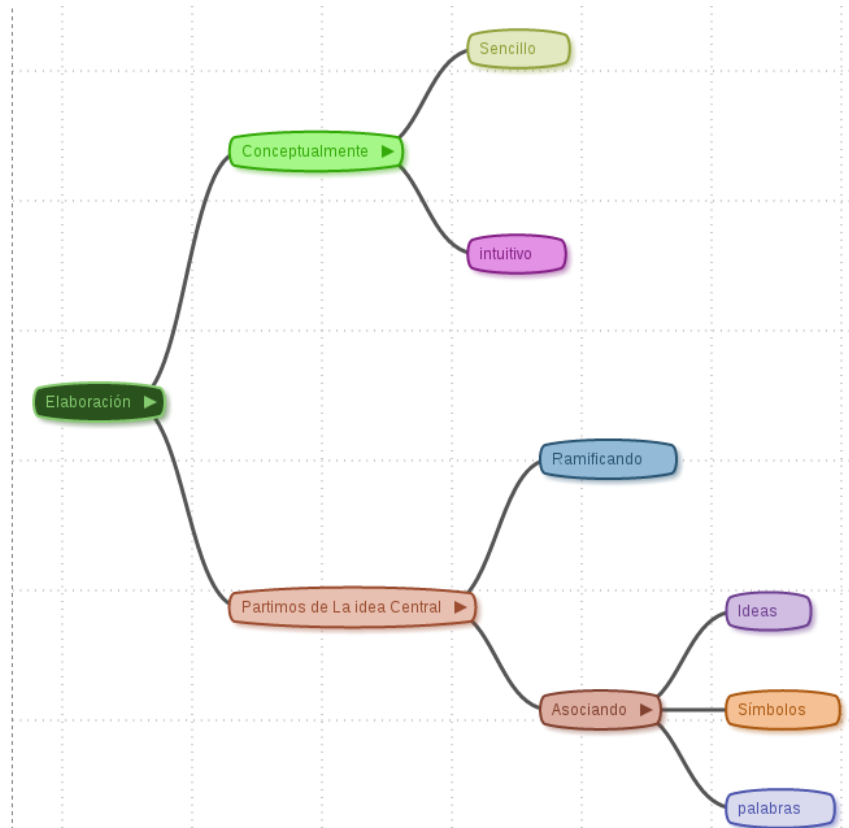


Figura 3.4: Mapa mental de elaboración de mapas mentales

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.

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, a 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 el 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 tres fases o iteraciones:

Primera iteración: 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:

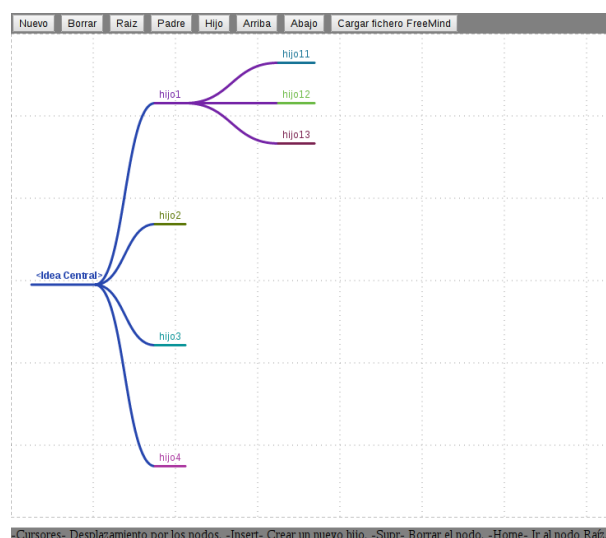


Figura 4.1: Versión inicial.

- **Librería base con soporte para herencia.** Esta librería debe tener toda la funcionalidad básica (bindings, currying, 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.

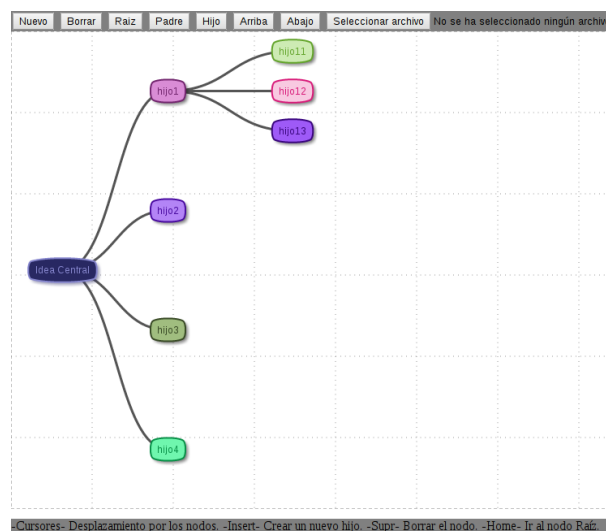


Figura 4.2: Versión inicial.

Segunda iteración: 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 iteración: 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>.

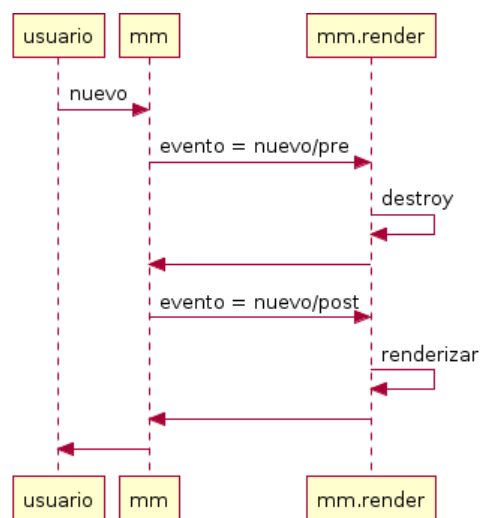


Figura 4.4: Diagrama de secuencia nuevo

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.

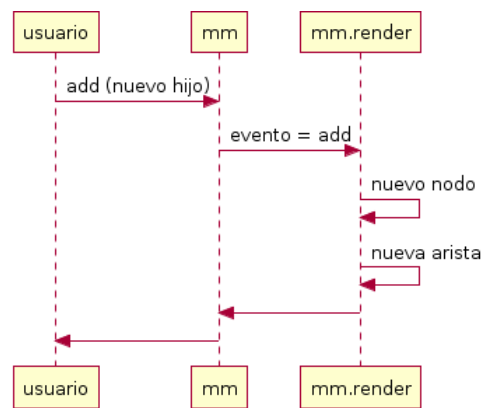


Figura 4.5: Diagrama de secuencia add

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

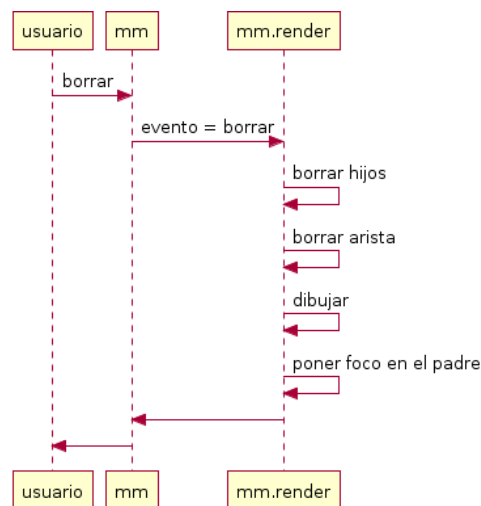


Figura 4.6: Diagrama de secuencia borrar

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 la escala (<Ctrl+0>). 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.

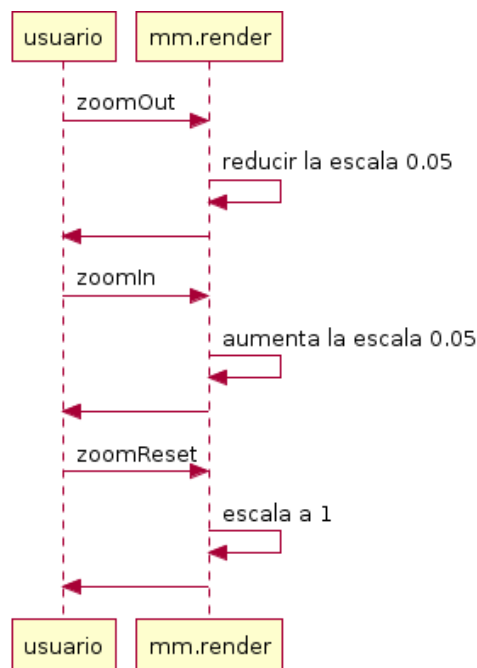


Figura 4.7: Diagrama de secuencia Zoom

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>

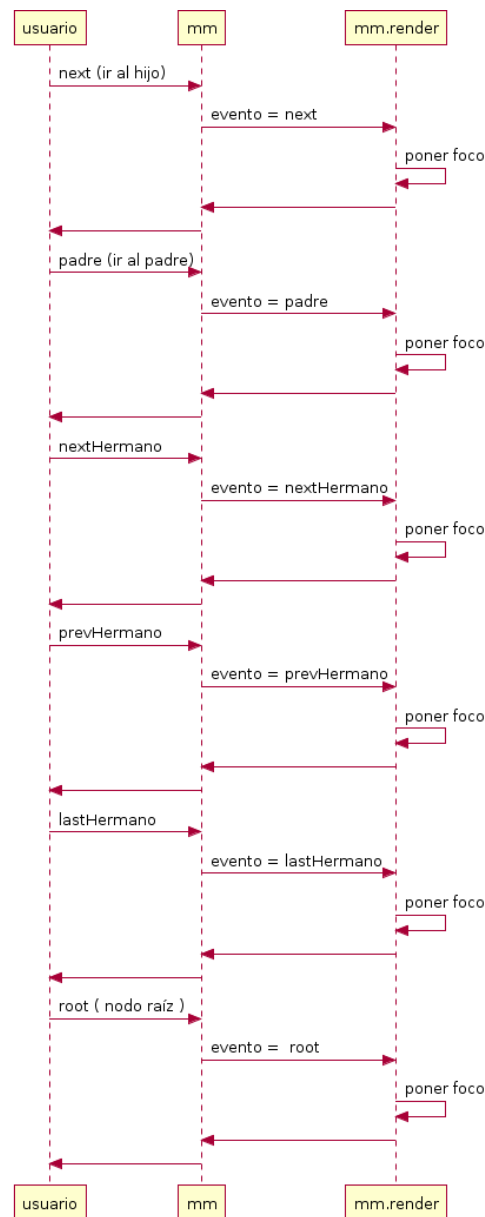


Figura 4.8: Diagrama de secuencia navegación

4.2.7. Mover idea

Con el ratón y touch podremos ajustar la posición de los nodos. Desplazando la idea por cualquier punto del marco de visión.

4.2.8. Mover mapa mental

Con el ratón y touch podremos desplazar el mapa. Ajustar la posición de todo el mapa mental para posibilitar al usuario el marco de visión deseado.

4.2.9. Salvar/cargar mapa mental

El usuario siempre tendrá opción de salvar y cargar mapas mentales en formato FreeMind. Un formato estándar para que el usuario pueda importar/exportar sus propios mapas mentales o de otros usuarios.

4.2.10. Hacer/deshacer acciones

El sistema dispondrá de opciones típicas de edición como hacer y deshacer. Tantas veces como quiera, en cualquier punto del programa.

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.

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

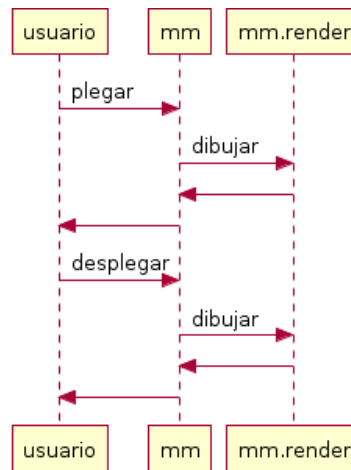


Figura 4.9: Diagrama de secuencia plegar

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 Javascripts es un lenguaje orientado a objetos puro y libre de clases.

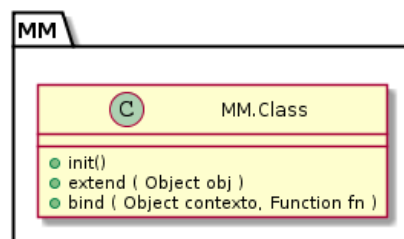


Figura 4.10: Clase MM.Class

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 de 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-Suscriptor¹.

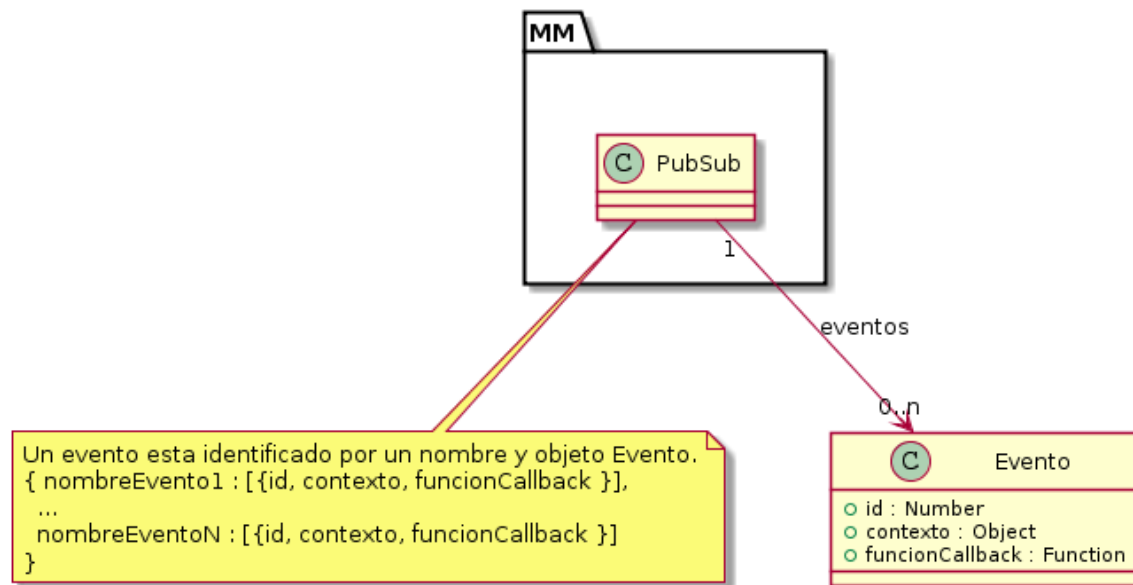


Figura 4.11: Diagrama de clases pubsub

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 la suscripción. Un símil muy utilizado, y directo, es el de los suscriptores de un periódico, en el cual un lector (o 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

Métodos:

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

¹También conocido como patrón Observador-observable

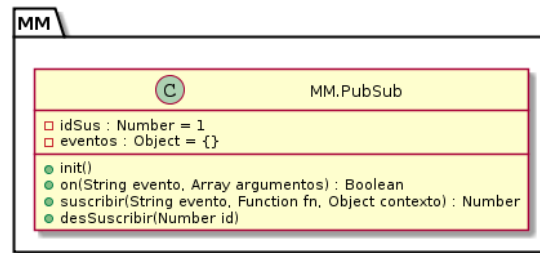


Figura 4.12: Clase MM.PubSub

- **MM.PubSub.desSuscribir:** permite a los suscriptores la desuscribirse de 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.

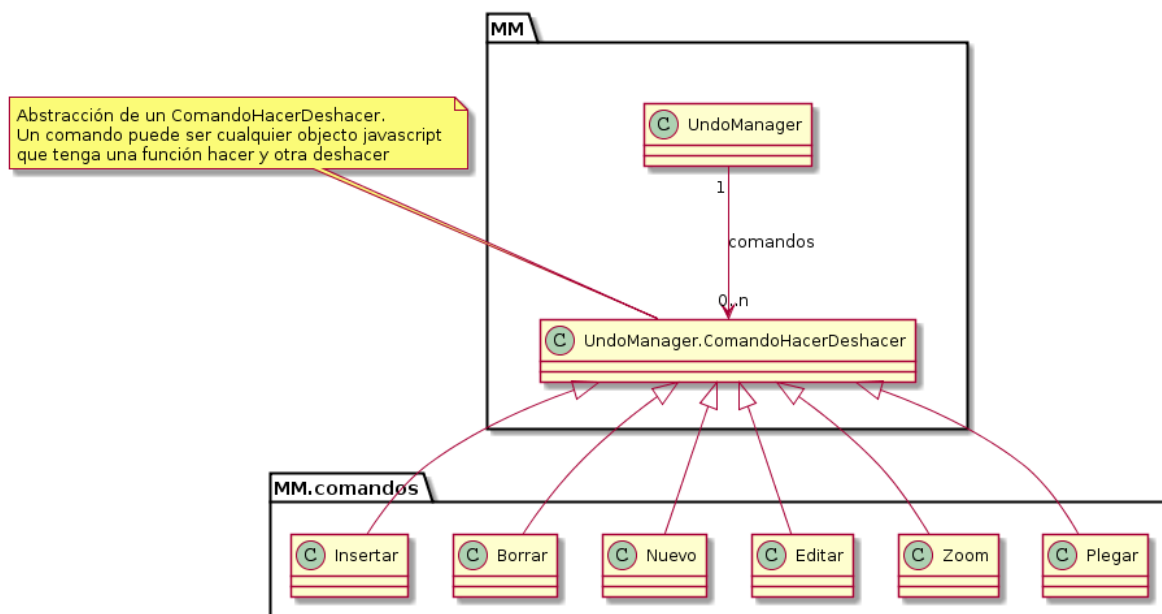


Figura 4.13: Diagrama de clases undo

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 ...

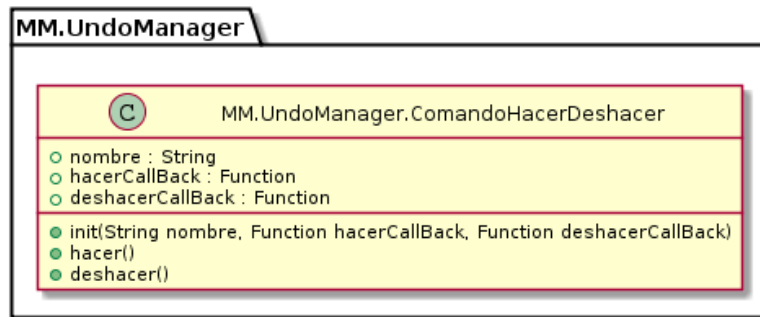


Figura 4.14: Clase MM.UndoManager.ComandoHacerDeshacer

Todo comando deberá tener un nombre e implementar los métodos hacer y deshacer. La funcionalidad del método hacer se encargará de repetir la operación ejecutada y el deshacer de revertir la.

Clase MM.UndoManager

El manejador de acciones hacer/deshacer tiene un registro de comandos ejecutados en la aplicación y un puntero² que indica el último comando ejecutado. El funcionamiento consiste en que siempre se pueda deshacer la última acción ejecutada, apuntada por el puntero actual, y sólo se pueda 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.

²Campo actual.

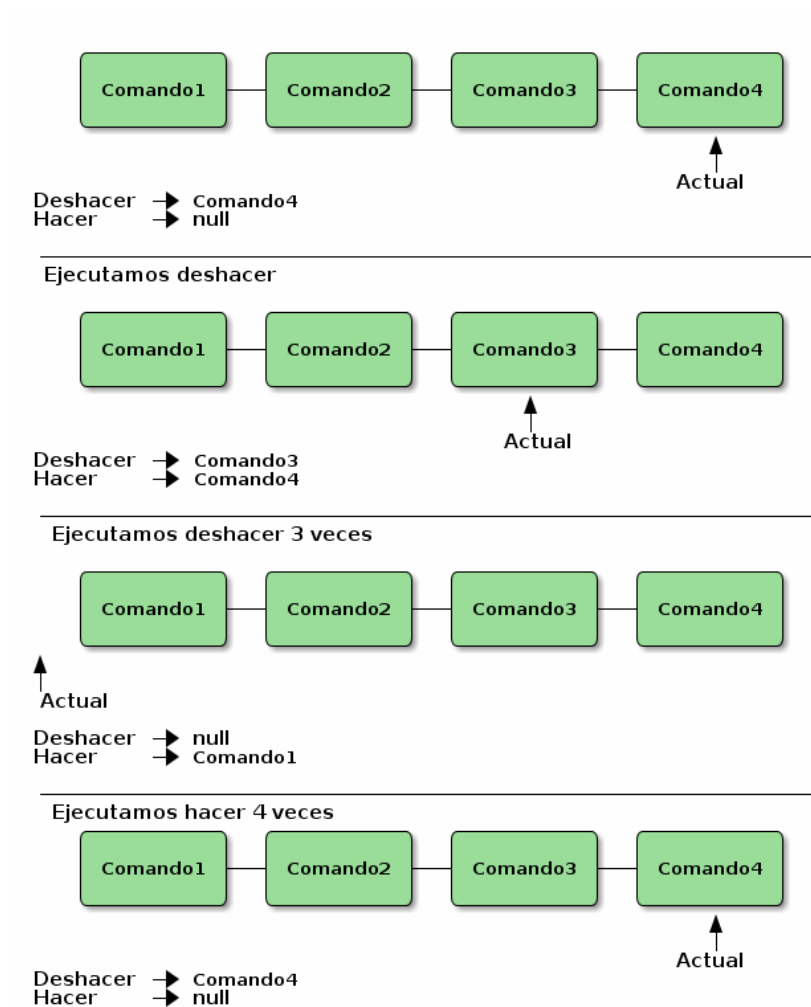


Figura 4.15: Secuencia de ejecución de UndoManager

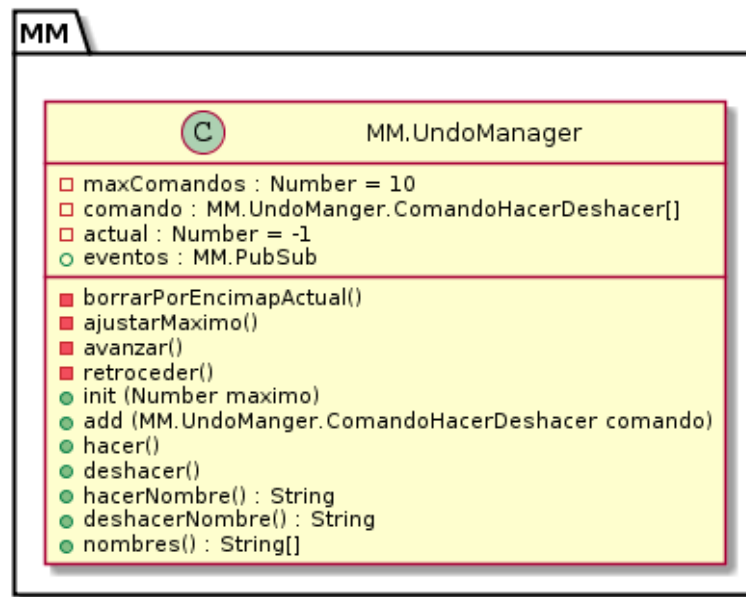


Figura 4.16: Clase MM.UndoManager

- **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.
- **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.

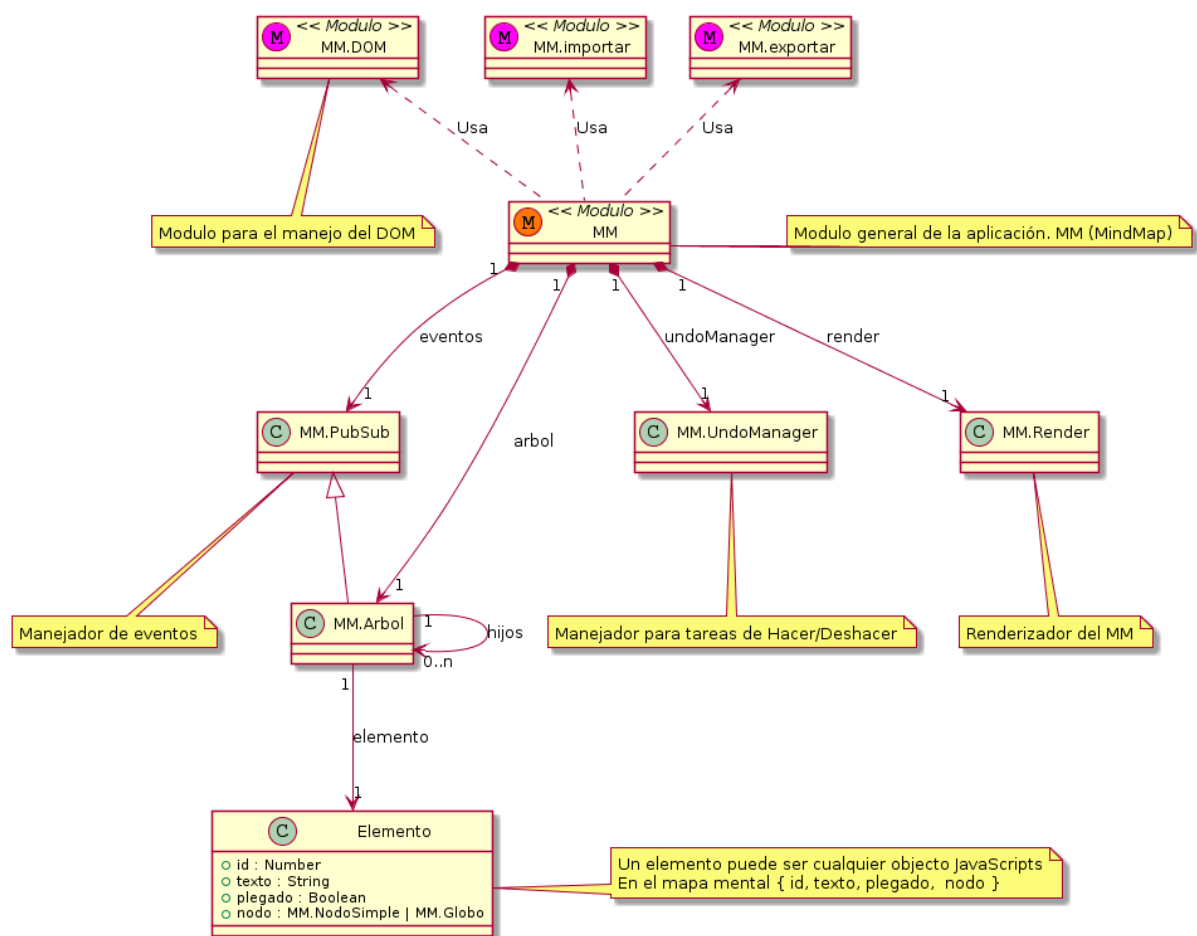


Figura 4.17: Diagrama de clases MM

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 sistema³. Así pues cuando el usuario añade un nuevo elemento al mapa mental, MM se encargará:

- Mantener la coherencia de los datos.
- 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

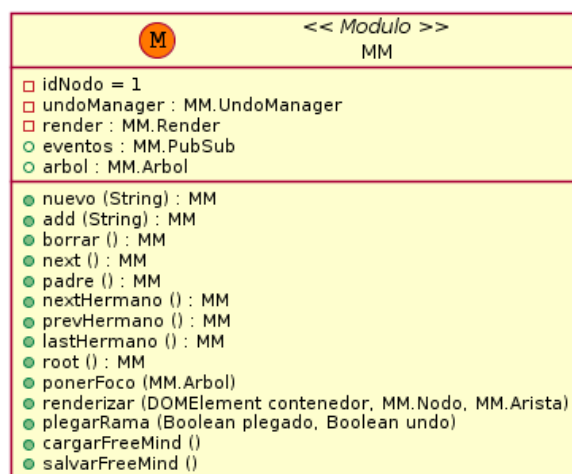


Figura 4.18: Clase MM

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.

³Por ejemplo al render o al interface de usuario

- **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

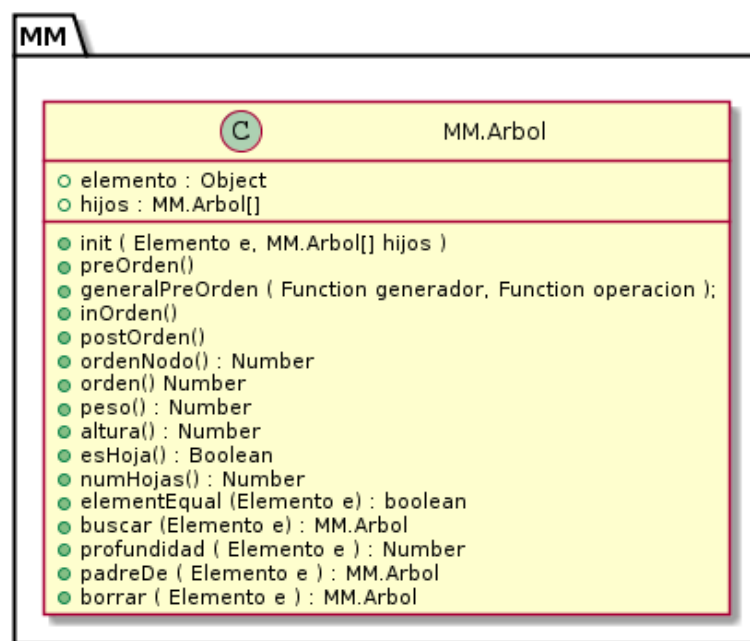


Figura 4.19: Clase MM.Arbol

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.

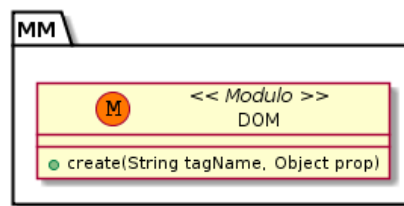


Figura 4.20: Módulo MM.DOM

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 configura o inicializa entorno a un elemento DOM, normalmente un *div*, una clase que MM.NodoSimple⁴ y una clase MM.Arista⁵. 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.

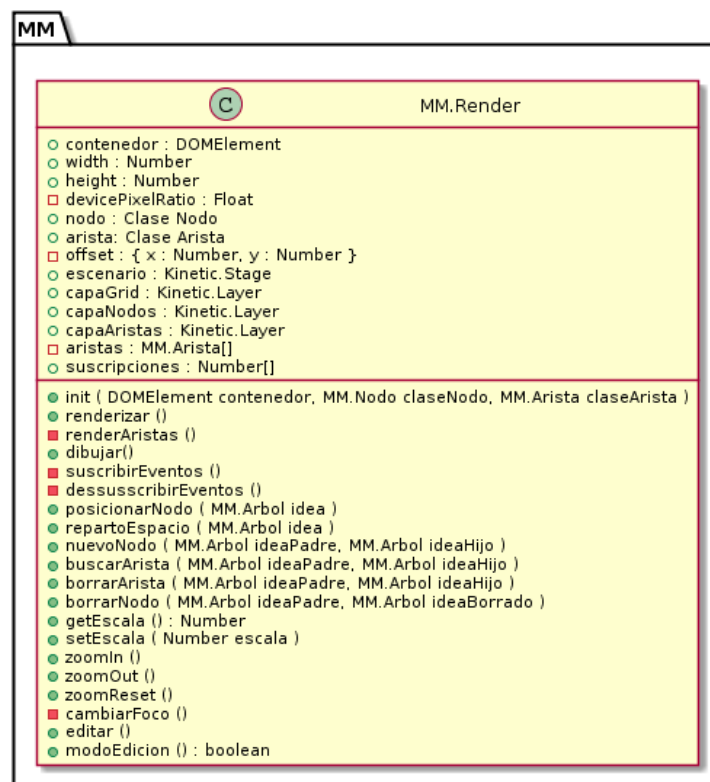


Figura 4.21: Módulo MM.Render

⁴O que herede de MM.NodoSimple como MM.Globo

⁵O que herede de MM.Arista

La clase MM.Render dispone de los siguientes métodos:

- **MM.Render.init:** constructor de la clase render. Inicializa las capas de nodos y aristas.
- **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.modaEdicion:** 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 diagrama 4.22, el MM.NodoSimple y el MM.Globo, y ambos pueden ser usados desde MM.Render. Todos los nodos existen en un escenario y en una capa dada.

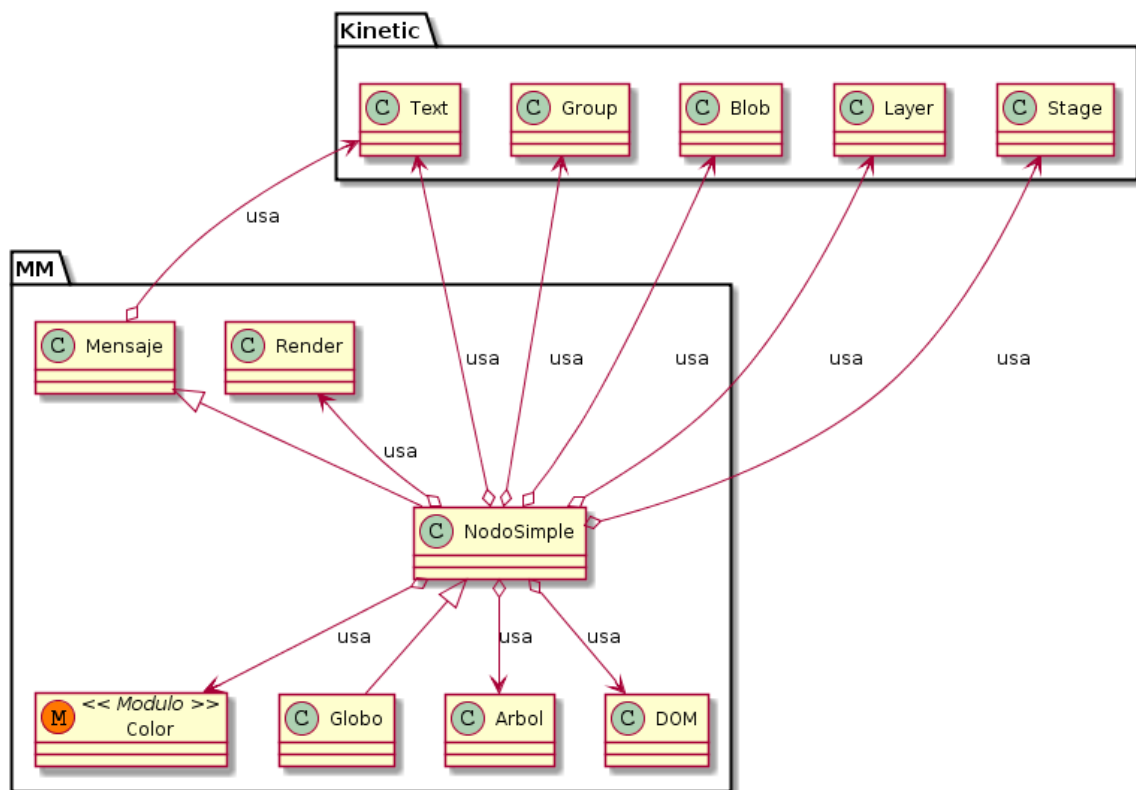


Figura 4.22: Diagrama de clases nodo

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.

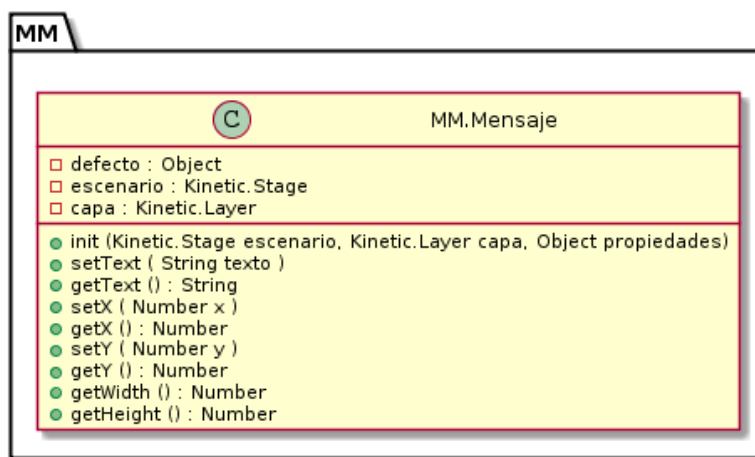


Figura 4.23: Clase MM.Mensaje

- **MM.Mensaje.getX/setX**: métodos para establecer y obtener la posición⁶ X del mensaje.
- **MM.Mensaje.getY/setY**: métodos para establecer y obtener la posición Y⁷ 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 subrayada. El nodo representa una idea que será renderizada y creada 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.

⁶en píxeles

⁷en píxeles

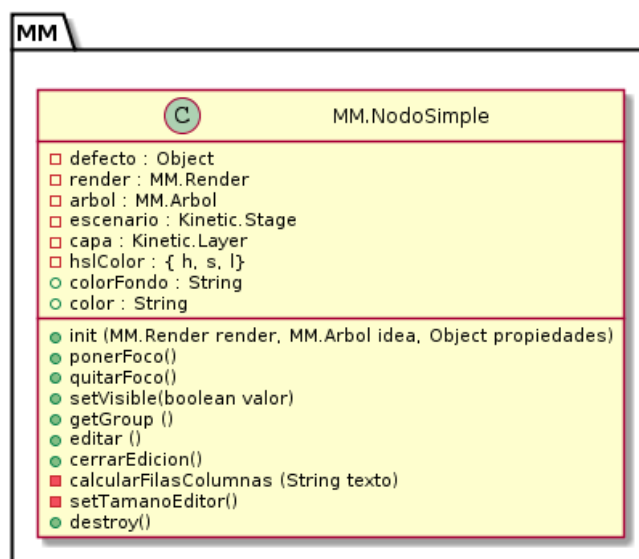


Figura 4.24: Clase MM.NodoSimple

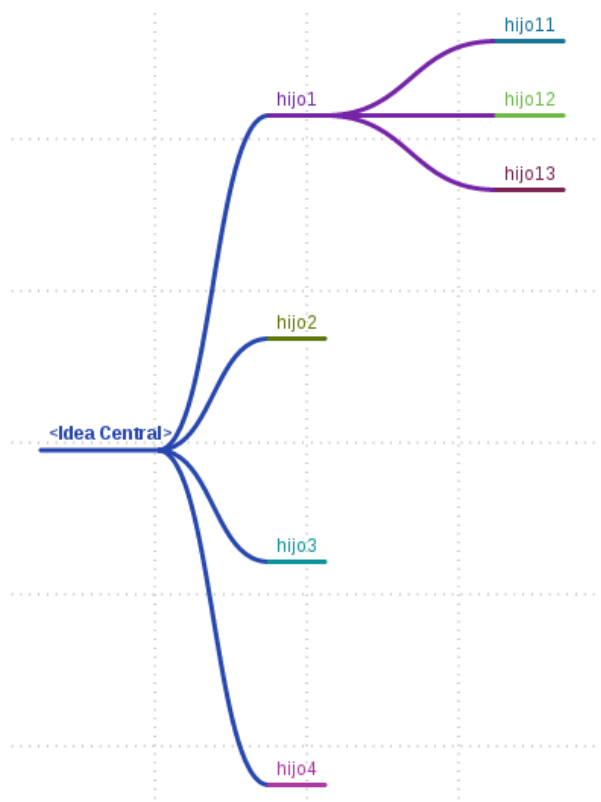


Figura 4.25: Mapa mental con renderización de nodos simples

- **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.
- **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.

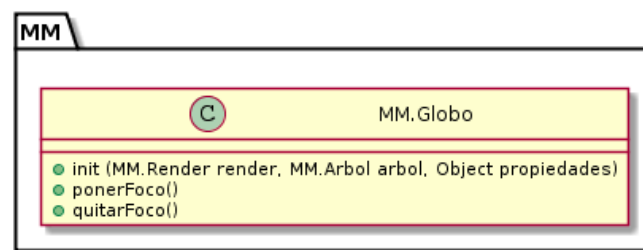


Figura 4.26: Clase MM.Globo



Figura 4.27: Mapa mental con renderización de nodos globo

Módulo **MM.Color**.

Módulo con funcionalidades de color. Permite generar distintas representaciones de color⁸ y realizar conversiones sobre los distintos tipos.

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 que deben unir. Las aristas,

⁸HSL, RGB y HUE

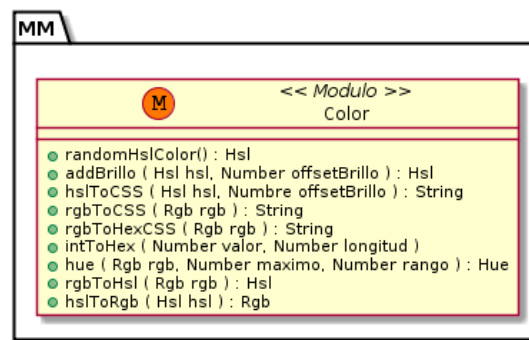


Figura 4.28: Modulo MM.Color

han sido implementadas con una curva Beizer. El diagrama de clases de aristas podemos ver lo en la figura 4.29.

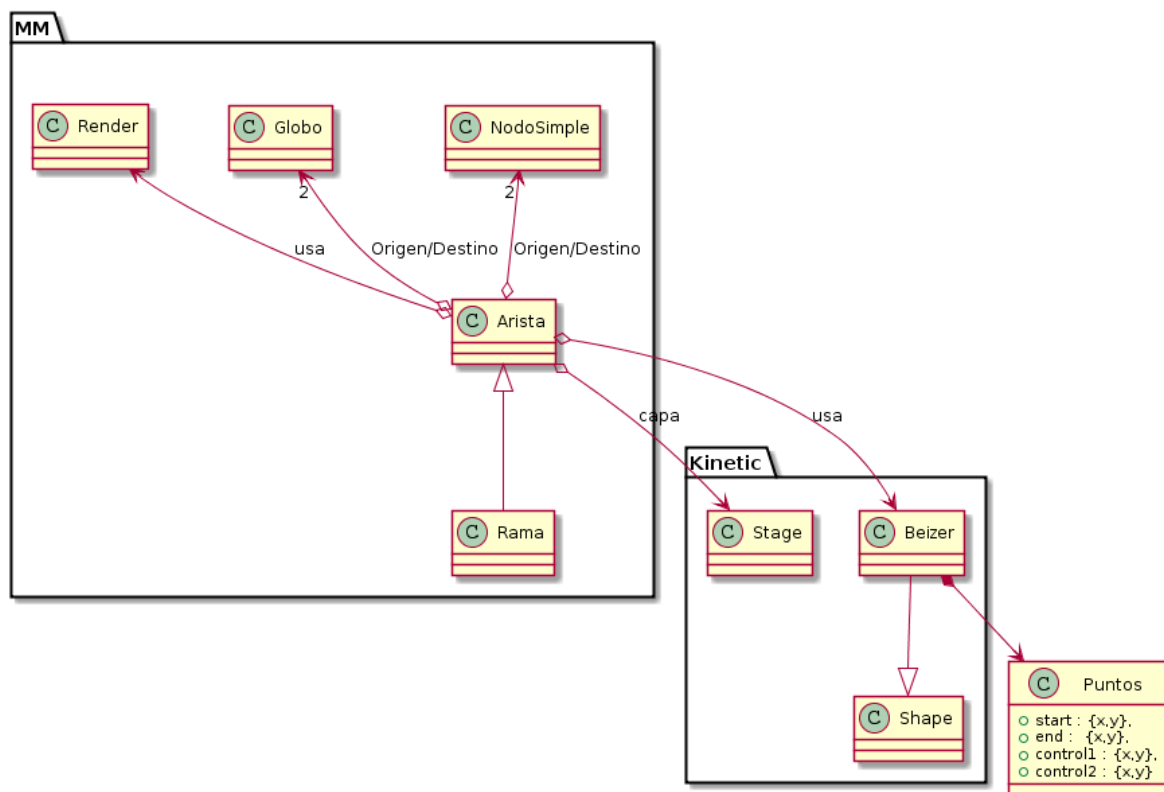


Figura 4.29: Diagrama de clases aristas

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.

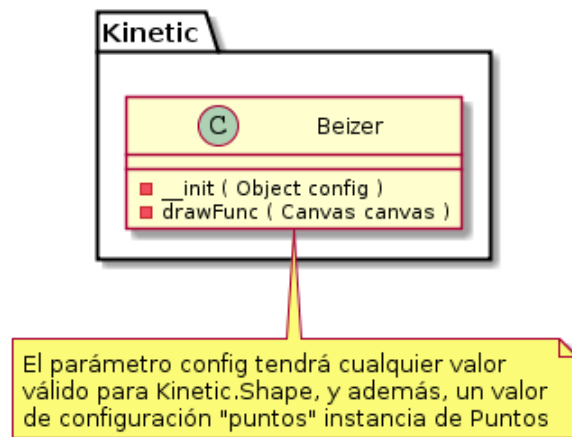


Figura 4.30: Clase Kinetic Beizer

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.

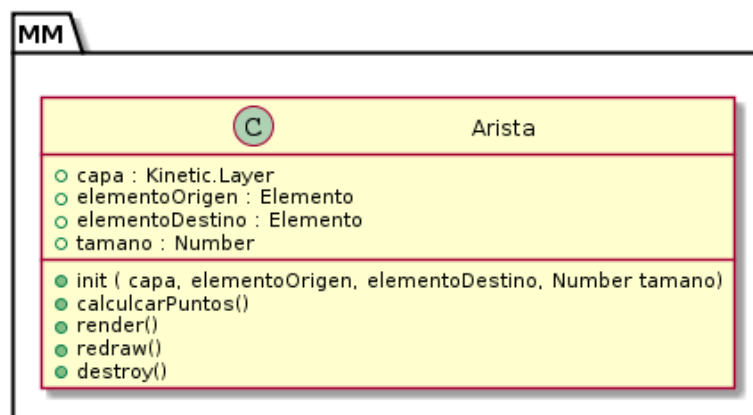


Figura 4.31: Clase MM.Arista

- **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**.

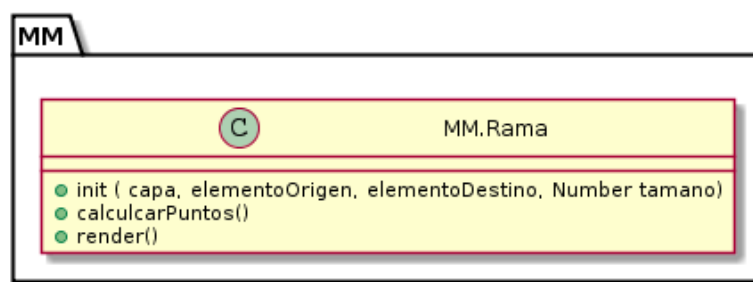


Figura 4.32: Clase **MM.Rama**

4.3.7. Diagrama de clases de teclado

Para una mejor experiencia de usuario se ha implementado un complejo manejador de teclados para procesar secuencias de 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 web no han sido pensadas para un uso intensivo de teclado.

⁹En los sistemas Mac.

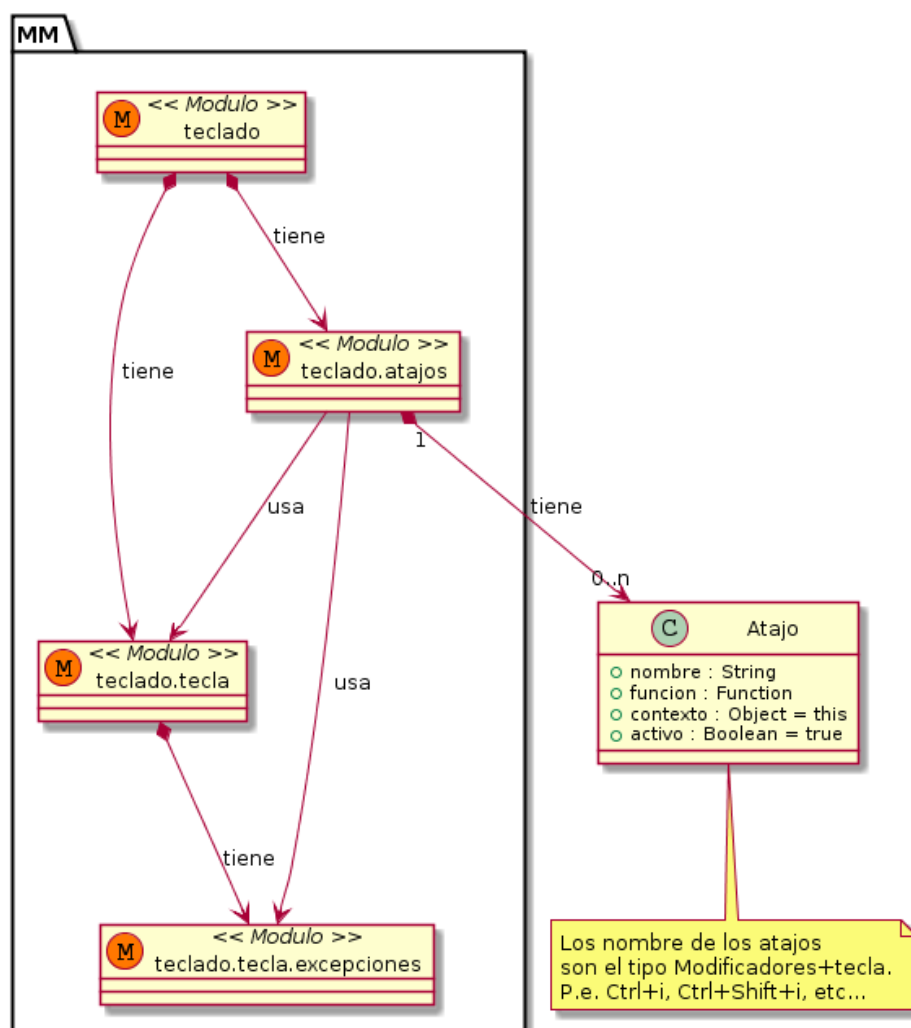


Figura 4.33: Diagrama de clases teclado

Módulo MM.teclado.atajos

Un atajo esta compuesto por un nombre¹⁰, 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 módulo de atajos registrar los atajos de teclados del sistema.

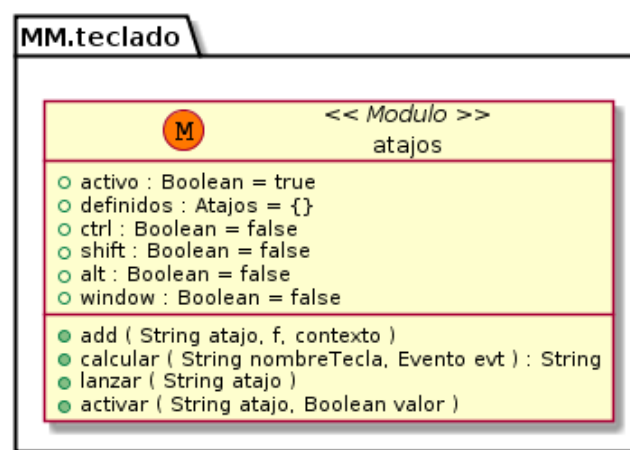


Figura 4.34: Modulo MM.teclado.atajos

- **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.

- **MM.teclado.tecla.nombre:** calcula el nombre de una tecla en función de su código.

¹⁰Ctrl+i

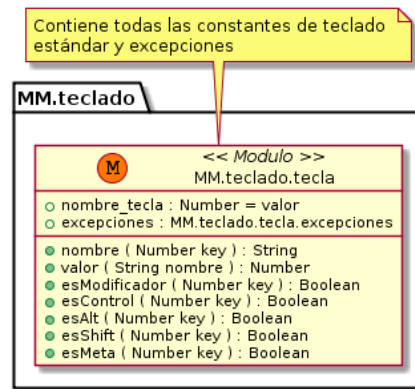


Figura 4.35: Clase MM.teclado.tecla

- **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.

El sistema de control de teclado se encarga de recoger todos los eventos¹¹ 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.

¹¹Todos los eventos KeyDown del navegador.

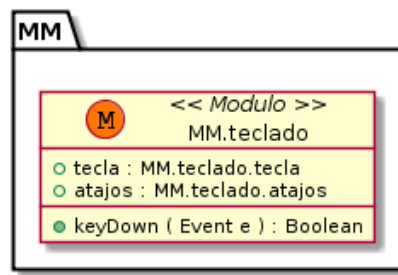


Figura 4.36: Clase MM.teclado

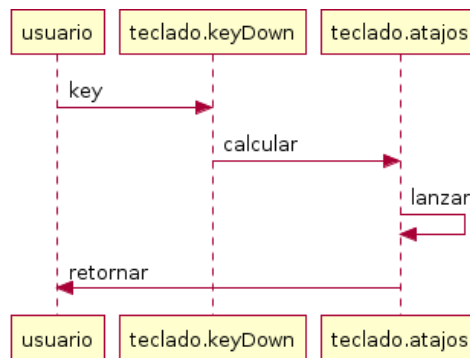


Figura 4.37: Diagrama de secuencia teclado

Implementación.

5.1. Javascripts.

5.1.1. Qué es.

Mozilla, los herederos directos de Netscape, definen Javascripts como:^{1 2}

Javascripts 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 Javascripts 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

¹MDN (Mozilla Developer Network)

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

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 objetos³. 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 = objeto['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 };
```

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

```
1 function map(f, xs) {  
2   var result = new Array();  
3   for (var i = 0; i < xs.length; i++)  
4     result.push(f.apply(null, [xs[i]]));  
5   return result;  
6 }
```

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

```
1 var Persona = function () {  
2   this.nombre = 'Sin nombre';  
3 };  
4
```

³Salvo los valores destacados null y undefined, el resto de valores Javascripts son objetos

```
5 Persona.prototype.setNombre = function () {  
6     this.nombre;  
7 };  
8  
9 var pepe = new Persona(); // pepe es una instancia de Persona  
10 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 Javascripts⁴ nadie podía intuir la importancia que adquiriría años después.

Internet aun estaba en pañales, navegar era lento⁵ 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 surgió 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, Microsoft lanzó JScript en su Internet Explorer 3. JScript no era más que una copia de Javascripts al que le cambiaron el nombre para evitar problemas legales. De esta forma comienzan las divergencias entre las distintas versiones de Javascripts, en esencia todas parten del mismo lenguaje y estándar, 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 evitara la guerra de tecnologías, Netscape decidió dar el paso, y en 1997 puso a disposición de ECMA⁶ la especificación de Javascripts1.1. ECMA creó el comité TC39 del cual surgió el primer estándar que se denominó ECMA-262⁷, o más popularmente, ECMAScript.

⁴ Javascripts fue un nombre por conveniencia legal. Originalmente se llamaba LiveScript

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

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>

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 su tarea.

5.1.3. Prototipos

Como se ha comentado con anterioridad Javascripts no utiliza el mecanismos de clases⁸ para implementar la herencia, para ello hace uso de los prototipos. Los prototipos son un paradigma de programación orientada a objetos en la cual una instanciación de objetos, se lleva a cabo mediante la clonación de otros objetos.

Los objetos en cualquier lenguaje son un conjunto de propiedades y métodos. Pues bien, Javascripts 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 prototipo⁹. 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).

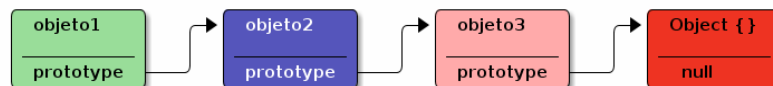


Figura 5.1: Cadena prototípica de objetos

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 prototípica de objeto --> Object.prototype --> null
2 > Object.getPrototypeOf(o);
3   Object {}
4 > Object.getPrototypeOf(Object.getPrototypeOf(o));
5   null
6
7 \\ cadena prototípica de una array --> Array.prototype --> Object.prototype --> null
8 > var array = [1,2];
9 > Object.getPrototypeOf(array);
10  []
11 > Object.getPrototypeOf(Object.getPrototypeOf(array));
12  Object {}
13 > Object.getPrototypeOf(Object.getPrototypeOf(Object.getPrototypeOf(array)));
14  null
  
```

⁸Paradigma sin clases

⁹prototype

Herencia de propiedades y métodos

Un objeto Javascripts es un conjunto de propiedades¹⁰ 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".

```

1 > var a = {
2   contador: 0,
3   contar : function () {
4     console.log('Contador ' + this.contador++);
5   }
6 };
7 > a.contar();
8   Contador 0
9 > a.contar();
10  Contador 1
11
12 > var b = Object.create(a); // Crea un objeto "b" que hereda de "a"
13 > b.contador; // es una copia exacta de "a"
14   2
15 > b.contar();
16   Contador 2
17 > a.contar(); // una copia no el mismo
18   Contador 2
19
20 // la cadena de prototipos de los objetos:
21 // 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`.

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

```

1 > var Persona = function (nombre) {
2   this.nombre = nombre;
3 };
4
5 > Persona.prototype = {
6   saluda : function () {
7     console.log('Hola soy ' + this.nombre);
8   }
9 };
10
11 > var pepe = new Persona ("pepe");
12 > pepe.saluda()
13   Hola soy pepe
14
15 /* prototipo de pepe --> Persona.prototype --> Object.prototype --> null */
16
17 > var juan = new Persona ("juan");
18 > juan.saluda()
19   Hola soy juan

```

¹⁰Los métodos son propiedades que referencia a una función

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.

```
1 > var Empleado = function (nombre, puesto) {
2   this.nombre = nombre;
3   this.puesto = puesto;
4 };
5 > Empleado.prototype = new Persona('');
6 // sobreescribimos saluda
7 > Empleado.prototype.saluda = function () {
8   console.log('Hola soy ' + this.nombre + ' ' + this.puesto );
9 };
10
11 > var pepe = new Empleado ('pepe', 'programador');
12 > pepe.saluda();
13   Hola soy pepe programador
14 > pepe instanceof Empleado
15   true
16 > pepe instanceof Persona
17   true
```

5.1.4. Ámbito de variable (Scope)

El ámbito de una variable¹¹ 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 Javascripts, nos referimos a nivel de función. Es decir, que las variables declaradas dentro de la función serán accesibles sólo dentro de la propia función.

```
1 var vbleGlobal = 'Soy una variable global';
2
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 Javascripts. Su simplicidad encierra gran potencia y flexibilidad que han aprovechado multitud de librerías¹².

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

¹¹scope

¹²Por citar algunas de las más populares: JQuery, Dojo y Undercore, entre otros

- 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 módulo¹³.

```

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 publica
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**.
- Al quedar encapsulado bajo un espacio de nombre nos lleva a mantener un **contexto global limpio**. Sólo necesitamos de una variable global¹⁴.
- 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

```

¹³Módulo propuesto por Douglas Crockford

¹⁴Nos referimos al propio módulo

¹⁵Utilizado MM (MindMap) por comodidad.

```

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 módulo MM 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 (
18        *                               undo/redo)
19       * @memberof MM
20       * @inner
21       */
22      mm.undoManager = new MM.UndoManager(10);
23
24      /**
25       * @prop {MM.PubSub} eventos Gestor de eventos del Mapa mental
26       * @memberof MM
27       * @inner
28       */
29      mm.eventos = new MM.PubSub();
30
31      /**
32       * @desc Sobreescritura del método "equal" del MM.Arbol. La comparación se realiza a
33       *       nivel de identificador.
34       * @method elementEqual
35       * @memberof MM
36       * @inner
37       */
38      MM.Arbol.prototype.elementEqual = function ( id ) {
39          return id === this.elemento.id;
40      };
41
42      /**
43       * @desc Genera un nuevo Mapa mental. Eliminar el Mapa mental existente hasta el
44       *       momento.
45       *       Resetea el contador de nodos.
46       * @param {String} ideaCentral Texto de la idea central. Por cefecto 'Idea Central'
47       * @method nuevo
48       * @memberof MM
49       * @instance
50       */
51      mm.nuevo = function ( ideaCentral ) {
52          if ( this.arbol ) {
53              this.ponerFoco ( this.arbol );
54
55              for ( var i = 0; i < this.arbol.hijos.length; i ) {
56                  this.next();
57                  this.borrar();
58              }
59
60              this.eventos.on ( 'nuevo/pre' );

```

```

61
62     idNodos = 1;
63
64     /**
65     * @prop {MM.Arbol} arbol Arbol-enario 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
81   *   el foco).
82   *   Todos los nodos del árbol tiene como elemento un id, texto y un nodo (
83   *   instancia de
84   *   MM.NodoSimple o MM.Globo. Es Chainable, esto nos permite realizar
85   *   operaciones encadenadas.
86   *   Por ejemplo, MM.add('Abuelo').add('Padre').add('Hijo').add('Nieto');
87   * @param {string} texto Texto del nuevo nodo. Valor por defecto "Nuevo".
88   * @return {MM} Al ser Chainable devuelve this (MM).
89   * @method add
90   * @memberof MM
91   * @instance
92   */
93   mm.add = function ( texto ) {
94     texto = texto || "Nueva idea";
95     var nuevo = new MM.Arbol ( { id: idNodos++, texto: texto, plegado: false, nodo:
96       null } );
97     this.foco.hijos.push ( nuevo );
98     this.undoManager.add(new MM.comandos.Insertar(this.foco.elemento.id, nuevo.
99       elemento.id, texto));
100     this.eventos.on ( 'add', this.foco, nuevo );
101     nuevo = null;
102   }.chain();
103
104   /**
105   * @desc Borra el nodo que tiene el foco. Implementael patrón Chainable.
106   * @return {MM} Al ser Chainable devuelve this (MM).
107   * @method borrar
108   * @memberof MM
109   * @instance
110   */
111   mm.borrar = function () {
112     if ( this.arbol === this.foco ) {
113       this.nuevo();
114       return;
115     }
116     var borrar = this.foco;
117     this.padre();
118     this.arbol.borrar ( borrar.elemento.id );
119     this.undoManager.add(new MM.comandos.Borrar(this.foco, borrar));
120     this.eventos.on ( 'borrar', this.foco, borrar );
121     borrar = null;
122   }.chain();
123
124   /**
125   * @desc Cambia el foco a primer hijo del nodo que tiene actualmente el foco.
126   * @return {MM} Al ser Chainable devuelve this (MM).
127   * @method next
128   * @memberof MM
129   * @instance
130   */
131   mm.next = function () {
132     if ( this.foco.ordenNodo() !== 0 ) {
133       this.eventos.on ( 'next', this.foco, this.foco.hijos[0] );
134       this.ponerFoco ( this.foco.hijos[0] );
135     }
136   }.chain();
137
138   /**
139   * @desc Cambia el foco al padre del nodo activo.
140   * @return {MM} Al ser Chainable devuelve this (MM).
141   * @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.
196                     length - 1] );
197                 this.ponerFoco ( padre.hijos[padre.hijos.length - 1] );
198             } else {
199                 this.eventos.on ( 'prevHermano', this.foco, padre.hijos[i - 1] );
200                 this.ponerFoco ( padre.hijos[i - 1] );
201             }
202             return;
203         }
204     }
205     padre = null;
206 }.chain();
207
208 /**
209 * @desc Cambia el foco al último hermano
210 * @return {MM} Al ser Chainable devuelve this (MM).
211 * @method lastHermano
212 * @memberof MM
213 * @instance
214 */
215 mm.lastHermano = function () {
216     var padre = this.arbol.padreDe ( this.foco.elemento.id );
217
218     if ( padre === null ) { return; }

```

```

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
263     *         el escenario al contenedor.
264     *         Una vez llamada a esta función queda establecido el valor de la propiedad MM
265     *         .render.
266     * @param {Element} contenedor Elemento del árbol DOM que contendrá
267     *         el Mapa mental.
268     * @param {MM.NodoSimple|MM.Globo} claseNodo Clase de renderizado de nodo
269     * @param {MM.Arista|MM.Rama} claseArista Clase de renderizado de aristas
270     * @method renderizar
271     * @memberof MM
272     * @instance
273     */
274     mm.renderizar = function ( contenedor, claseNodo, claseArista ) {
275         mm.render = new MM.Render ( contenedor, claseNodo, claseArista );
276         mm.render.renderizar();
277     };
278
279     /**
280     * @desc Marca el nodo actual (foco) como plegado, si no estaba plegado o como
281     *         desplegado si estaba plegado.
282     * @param {Boolean} plegado Si es true fuerza el plegado y si es false el desplegado
283     * @method plegadoRama
284     * @memberof MM
285     * @instance
286     */
287     mm.plegarRama = function (plegado, undo) {
288         // - PLEGADO: Se pliega toda la herencia del nodo.
289         // - DESPLEGADO: Se despliega sólo el nodo en cuestión.
290
291         plegado = plegado || !this.foco.elemento.plegado;
292         this.foco.elemento.plegado = plegado;
293         var plegar = function (a) {
294             a.hijos.forEach(function (h) {
295                 h.elemento.plegado = true;
296                 plegar(h);
297             });
298         };
299         var desplegar = function (a) {
300             var aPlegado = a.elemento.plegado;

```

```

298         a.hijos.forEach(function (h) {
299             h.elemento.plegado = ( !aPlegado && h.esHoja() )?false:h.elemento.plegado
300             ;
301             desplegar(h);
302         });
303         aPlegado = null;
304     };
305     if ( plegado ) {
306         plegar(this.foco);
307     } else {
308         desplegar(this.foco);
309     }
310     this.render.dibujar(MM.arbol);
311     if ( !undo ) {
312         this.undoManager.add(new MM.comandos.Plegar(this.foco, plegado));
313     }
314 };
315
316 /**
317  * @desc Abre un cuadro de dialogo para seleccionar el fichero FreeMind que deseamos
318  *        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.desuscribirEventos();
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 return mm;
347 }(MM);

```

Como se puede observar se ha utilizado incorporado una pequeña variación con respecto al módulo propuesto por Douglas Crockford. Se trata de un módulo 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

Como ya hemos podido comprobar Javascripts es un lenguaje sin clases, pero podemos simular, más o menos, el comportamiento de clase. Para el proyecto he implementado un patrón de extensión que nos permite tanto heredar como implementar la sobreescritura de funciones (métodos).

```

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
26   *   clase padre.
27   */
28  MM.Class.extend = function(prop) {
29      var _super = this.prototype || MM.Class.prototype; // prototype de la clase padre
30
31      function F() {}
32      F.prototype = _super;
33      var proto = new F();
34      var wrapperMetodo = function(name, fn) { // asociamos las funciones al nuevo contexto
35          return function() {
36              var tmp = this._super;

```

```

36     this._super = _super[name];           // función super => podemos hacer this.
37     _super(argumentos)
38     var ret = fn.apply(this, arguments); // ejecutamos el método en el contexto
39     de la nueva instancia                // restauramos el _super
40     return ret;
41 };
42
43 // recorremos el objeto que nos han pasado como parámetro...
44 for (var name in prop) {
45     // Si estamos sobrescribiendo 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 sobrescribimos 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 }

```

Como se puede observar el código implementado en `MM.Class.extend`, se trata de una función que devuelve otra función constructora, a la cual se le ha añadido, manipulado el prototipo a partir de un objeto (`prop`) del que deseamos heredar o extender. Las propiedades del objeto padre, simplemente se copia pero las funciones (métodos) del objetos padre son envueltos en un closure, para poder disponer de sobrescritura de métodos.

La función `init` es el constructor de nuestra clase y se llamará cuando se instancie el objeto.

```

1 var Persona = MM.Class.extend({
2     init: function(nombre) {
3         this.nombre = nombre;
4     },
5     bailar: function() {
6         return "Si";
7     },
8     piernas: 2
9 });
10
11 var Hombre = Persona.extend({
12     init: function(nombre) {
13         this._super(nombre);

```

```

14   },
15   bailar: function() {
16     return this._super() + ", pero es torpe";
17   },
18
19   piernas: 2,
20
21   sexo: 'Hombre'
22 });
23
24 > var pepe = new Hombre('pepe'); // nueva instancia de hombre
25 > pepe.nombre; // pepe
26 > pepe.bailar(); // Si, pero es torpe
27 > pepe instanceof Hombre; // true

```

Se puede observar en el ejemplo, como la implementación de MM.Class nos permite crear una jerarquía de clases. Además de un constructor y sobreescritura de funciones o métodos. En otras palabras, nos proporciona el mecanismo básico para sistemas más complejos.

5.1.7. Patrón Chainable.

Popularizado por JQuery, el patrón Chainable, o encadenado, nos permite encadenar llamadas de forma que las aplicaciones pueden quedar más legibles y compactas. En MindMapJS se ha utilizado siempre que se ha podido sobre todo en el módulo MM.

```

1 // ejemplo de Chainable en el MindMapJS
2 MM.nuevo('Como usar MindMapJS').add('Teclado').add('Raton').add('Tablet');
3 // Crea un nuevo mapa mental con tres nodos

```

El patrón Chain, devuelve siempre el propio objeto del contexto de ejecución. De forma, que siempre podemos seguir realizando llamadas encadenadas.

```

1 /**
2  * @file chain.js añade el patrón chainable al sistema
3  * @author José Luis Molina Soria
4  * @version 20130224
5  */
6
7 /**
8  * @desc Implementación del patrón Chainable, mediante la extensión del prototipo de la
9  * función
10  * @return {function} función extendida
11  */
12 Function.prototype.chain = function() {
13   var self = this;
14   return function() {
15     var ret = self.apply(this, arguments);
16     return ret === undefined ? this : ret;
17   };
18 };

```

En la implementación realizada se puede observar como se ha modificado el prototipo del propio objeto Function. De esta forma siempre podemos hacer nuestras funciones chain sin necesidad de acordarnos de devolver "this".

```

1 // ejemplo uso del patron
2 var mod = function () {
3   return {

```

```

4   saludar: function(){ console.log(" hola ") }.chain();
5   };
6 }();
7 > mod.saludar().saludar().saludar() // " hola  hola  hola  "

```

5.1.8. Patrón publicador/suscriptor.

Una de las grandes virtudes de NodeJS es el manejo de eventos para vertebrar distintos módulos o clases. MindMapJS también tiene un sistema de manejo de eventos. Más concretamente un patrón publicador/suscriptor.

```

1  /**
2   * @file pubsub.js Implementación del patrón Publish/Subscribe
3   * @author José Luis Molina Soria
4   * @version 20130227
5   */
6
7  if ( typeof module !== 'undefined' ) {
8      var MM = require('./MindMapJS.js');
9      MM.Class = require('./klass.js');
10 }
11
12 /**
13  * @class MM.PubSub
14  * @classdesc Implementación del patrón Publish/Subscribe
15  * @constructor MM.PubSub
16  */
17 MM.PubSub = MM.Class.extend(/** @lends MM.PubSub.prototype */{
18
19     eventos : {},
20
21     idSus : 1,
22
23     init : function () {
24         this.eventos = {};
25         this.idSus = 1;
26     },
27
28     /**
29      * @desc Realiza la notificación a los suscriptores de que se a producido
30      * una publicación o evento.
31      * @param evento {string} nombre del evento o publicación a notificar
32      * @param args {[*]} argumentos para la función callback
33      * @return {boolean} Si el evento no es un nombre valido retorna false en
34      * otro caso retorna true
35      */
36     on : function( evento ) {
37         if (!this.eventos[evento]) {
38             return false;
39         }
40         var args = Array.prototype.slice.call(arguments, 1);
41         this.eventos[evento].forEach(function (evt){
42             evt.fuccion.apply(evt.contexto, args);
43         });
44         args = null;
45
46         return true;
47     },
48
49     /**
50      * @desc Pemite la suscripción a una publicación o evento. Donde el parametro func es
51      * la función a ejecutar en el caso de que se produzca la notificación y contexto el
52      * contexto de ejecución para la función callback
53      * @param evento {string} nombre del evento o publicación en la que deseamos
54      * suscribarnos
55      * @param func {function} función callback
56      * @param contexto {object} contexto de ejecución de la función callback
57      * @return {null|number} null en caso de fallo o *idSus* el identificador de
58      * suscripción
59      */
60     suscribir : function( evento, func, contexto ) {
61         if ( !evento || !func ) {

```

```

60         return null;
61     }
62
63     if (!this.eventos[evento]) {
64         this.eventos[evento] = [];
65     }
66
67     contexto = contexto || this;
68     this.eventos[evento].push({ id : this.idSus, contexto: contexto, funcion: func })
69     ;
70     return this.idSus++;
71 },
72 /**
73  * @desc realiza una dessuscripción a un evento o notificación
74  * @param id {number} identificador de suscripción
75  * @return {null|number} null si no se ha podido realizar la dessuscripción
76  */
77 desSuscribir : function (id) {
78     for (var evento in this.eventos) {
79         if ( this.eventos[evento] ) {
80             for (var i = 0, len = this.eventos[evento].length; i < len; i++) {
81                 if (this.eventos[evento][i].id === id) {
82                     this.eventos[evento].splice(i, 1);
83                     return id;
84                 }
85             }
86         }
87     }
88     return null;
89 }
90
91 });
92
93 if ( typeof module !== 'undefined' ) {
94     module.exports = MM.PubSub;
95 }

```

En realidad, se trata de un patrón bastante simple, pero las posibilidades que nos proporcionan y limpieza son muy, muy importantes. Se trata de un registro de eventos y funciones callbacks a ejecutar cuando el evento publicador lo requiera.

MindMapJS ha realizado un uso intensivo del patrón. De echo, ha permitido implementar complejos mecanismos e interacciones entre el módulo MM y el render del árbol. El siguiente código muestra como es usado en la función add crea una nueva idea en el mapa mental, y como suscrito al evento, podrá reaccionar a la creación de una nueva idea.

```

1  // MM eleva el evento 'add'
2  mm.add = function ( texto ) {
3      texto = texto || 'Nueva idea';
4      var nuevo = new MM.Arbol ( { id: idNodos++, texto: texto, plegado: false, nodo: null
5      } );
6      this.foco.hijos.push ( nuevo );
7      this.undoManager.add(new MM.comandos.Insertar(this.foco.elemento.id, nuevo.elemento.
8      id, texto) );
9      this.eventos.on ( 'add', this.foco, nuevo );
10     nuevo = null;
11     }.chain();
12
13 // El render esta suscrito al evento para poder reaccionar a los
14 // cambios del mapa mental.
15 render.prototype.suscribirEventos = function ( ) {
16     this.desuscribirEventos(); // evitamos dobles suscripciones
17     var sus = this.suscripciones;
18     var e = MM.eventos;
19     sus.push ( e.suscribir('ponerFoco', cambiarFoco) );
20     sus.push ( e.suscribir('add', this.nuevoNodo, this) );

```



```

19  sus.push ( e.suscribir('borrar', this.borrarNodo, this) );
20  sus.push ( e.suscribir('nuevo/pre', function () {
21      MM.arbol.elemento.nodo.destroy();
22  }) );
23  sus.push ( e.suscribir('nuevo/post', function () {
24      this.renderizar();
25  }, this) );
26  this.contenedor.addEventListener("mousewheel", handlerWheel, false);
27  this.contenedor.addEventListener("DOMMouseScroll", handlerWheel, false);
28  sus = e = null;
29  };

```

Las funciones callbacks asociadas a un evento se apilan y se ejecutan en orden de registro.

Así pues, puede existir distintos puntos dentro de la aplicación donde tratar el evento.

5.1.9. UndoManager.

El comportamiento de esta clase ya fue explicado con su diagrama de clase y en la figura 5.2 en este apartado vamos a destripar un poco más su comportamiento.

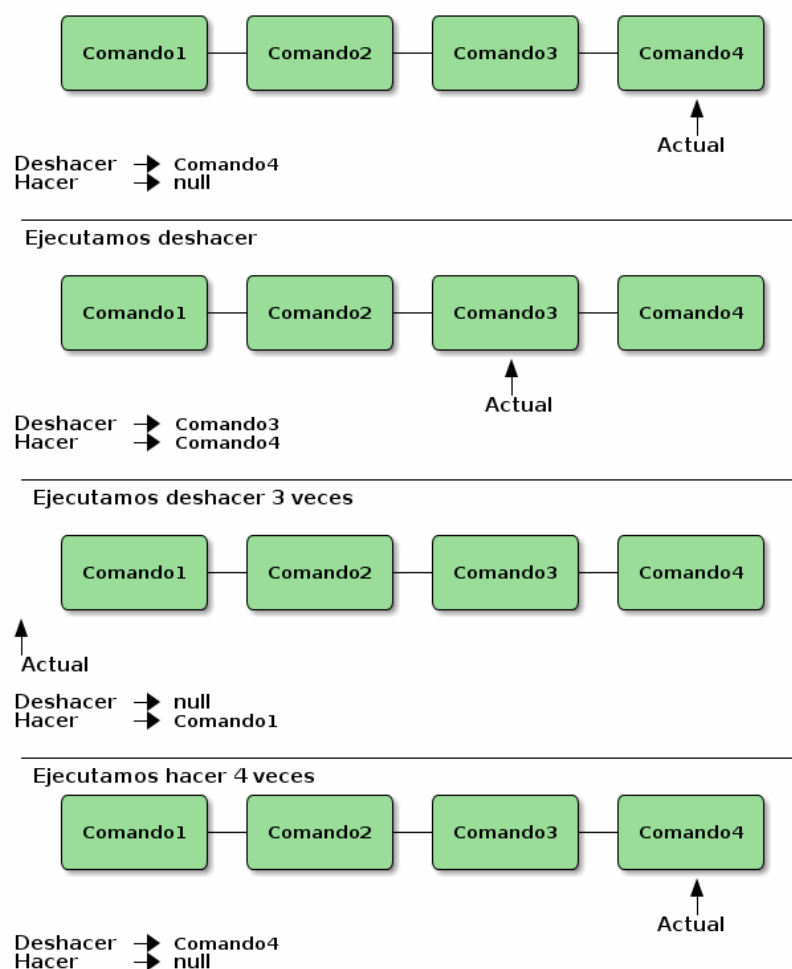


Figura 5.2: Secuencia de ejecución de UndoManager

La idea primigenia es posibilitar al usuario final de la aplicación la opción de deshacer y rehacer las acciones ejecutadas en el MindMapJS. UndoManager mantiene en un array los comandos realizados y un puntero (actual) que apunta a la última comando realizado. También disponemos de un limite de comandos(maxComandos) apilados en el historial.

El interfaz de la clase MM.UndoManager nos permite añadir nuevos comandos al historial, hacer y deshacer, conocer el estado del historial y el nombre del siguiente comando hacer o deshacer. También se le ha incorporado un manejador de eventos para que, los usuarios de la clase puedan saber en todo momento los cambios sufridos en el historial.

```

1  /**
2   * @file undoManager.js Implementación de un gestor de comandos hacer y deshacer
3   * @author José Luis Molina Soria
4   * @version 20130620
5   */
6
7  if ( typeof module !== 'undefined' ) {
8      var MM = require('./MindMapJS.js');
9      MM.Class = require('./klass.js');
10     MM.PubSub = require('./pubsub.js');
11 }
12
13 /**
14  * @class MM.UndoManager
15  * @classdesc Gestor de comandos undo (hacer y deshacer).
16  * @constructor
17  * @param maximo {integer} El máximo de comando en buffer. Por defecto, 10.
18  */
19 MM.UndoManager = MM.Class.extend(function() {
20     /**
21      * @prop {Array} Comando del tipo Hacer / Deshacer
22      * @memberof MM.UndoManager
23      * @inner
24      */
25     var comandos = []; // la lista de comandos
26
27     /**
28      * @prop {integer} Tamaño máximo del buffer
29      * @memberof MM.UndoManager
30      * @inner
31      */
32     var maxComandos = 10; // número máximo de comandos en cola
33
34     /**
35      * @prop {integer} Indice del comando actual
36      * @memberof MM.UndoManager
37      * @inner
38      */
39     var actual = -1; // índice comando actual
40
41
42     var eventos = new MM.PubSub();
43
44     var init = function ( maximo ) {
45         maxComandos = maximo || 10;
46     };
47
48     /**
49      * @desc Añade un nuevo comando a la pila de comandos. Si el tamaño del buffer
50      *       sobrepasa el
51      *       máximo fijado, entonces elimina el comando más antiguo. Si existiesen
52      *       comandos por
53      *       encima del actual, estos serán eliminados.
54      * @param {MM.UndoManager.ComandoHacerDeshacer} Comando a añadir al buffer.
55      * @memberof MM.UndoManager
56      * @instance
57      */
58     var add = function (comando) {
59         borrarPorEncimaActual();
60     };
61
62     /**
63      * @desc Elimina el comando más antiguo del buffer. Si el tamaño del buffer
64      *       es menor que el máximo fijado, no se elimina nada.
65      * @memberof MM.UndoManager
66      * @instance
67      */
68     var borrarPorEncimaActual = function () {
69         if (comandos.length > maxComandos) {
70             comandos.splice(0, 1);
71         }
72     };
73
74     /**
75      * @desc Hace el comando actual. Si el comando actual es 'Hacer', se ejecuta el
76      *       comando. Si es 'Deshacer', se elimina el comando actual del buffer y se
77      *       ejecuta el comando anterior.
78      * @memberof MM.UndoManager
79      * @instance
80      */
81     var hacer = function () {
82         if (comandos[actual].tipo === 'Hacer') {
83             comandos[actual].hacer();
84         } else {
85             borrarPorEncimaActual();
86             comandos[actual].hacer();
87         }
88     };
89
90     /**
91      * @desc Deshace el comando actual. Si el comando actual es 'Deshacer', se
92      *       elimina el comando actual del buffer y se ejecuta el comando anterior.
93      *       Si es 'Hacer', se ejecuta el comando actual.
94      * @memberof MM.UndoManager
95      * @instance
96      */
97     var deshacer = function () {
98         if (comandos[actual].tipo === 'Deshacer') {
99             borrarPorEncimaActual();
100            comandos[actual].deshacer();
101        } else {
102            comandos[actual].deshacer();
103        }
104    };
105
106     /**
107      * @desc Devuelve el estado del historial.
108      * @memberof MM.UndoManager
109      * @instance
110      */
111     var estado = function () {
112         return {
113             comandos: comandos,
114             actual: actual,
115             maxComandos: maxComandos
116         };
117     };
118
119     /**
120      * @desc Devuelve el nombre del siguiente comando.
121      * @memberof MM.UndoManager
122      * @instance
123      */
124     var siguienteComando = function () {
125         if (comandos[actual].tipo === 'Hacer') {
126             return 'Deshacer';
127         } else {
128             return 'Hacer';
129         }
130     };
131
132     /**
133      * @desc Devuelve el nombre del comando anterior.
134      * @memberof MM.UndoManager
135      * @instance
136      */
137     var anteriorComando = function () {
138         if (comandos[actual].tipo === 'Hacer') {
139             return 'Deshacer';
140         } else {
141             return 'Hacer';
142         }
143     };
144
145     /**
146      * @desc Devuelve el nombre del comando actual.
147      * @memberof MM.UndoManager
148      * @instance
149      */
150     var comandoActual = function () {
151         return comandos[actual].tipo;
152     };
153
154     /**
155      * @desc Devuelve el nombre del comando anterior.
156      * @memberof MM.UndoManager
157      * @instance
158      */
159     var comandoAnterior = function () {
160         return comandos[actual].tipo;
161     };
162
163     /**
164      * @desc Devuelve el nombre del comando actual.
165      * @memberof MM.UndoManager
166      * @instance
167      */
168     var comandoActual = function () {
169         return comandos[actual].tipo;
170     };
171
172     /**
173      * @desc Devuelve el nombre del comando anterior.
174      * @memberof MM.UndoManager
175      * @instance
176      */
177     var comandoAnterior = function () {
178         return comandos[actual].tipo;
179     };
180
181     /**
182      * @desc Devuelve el nombre del comando actual.
183      * @memberof MM.UndoManager
184      * @instance
185      */
186     var comandoActual = function () {
187         return comandos[actual].tipo;
188     };
189
190     /**
191      * @desc Devuelve el nombre del comando anterior.
192      * @memberof MM.UndoManager
193      * @instance
194      */
195     var comandoAnterior = function () {
196         return comandos[actual].tipo;
197     };
198
199     /**
200      * @desc Devuelve el nombre del comando actual.
201      * @memberof MM.UndoManager
202      * @instance
203      */
204     var comandoActual = function () {
205         return comandos[actual].tipo;
206     };
207
208     /**
209      * @desc Devuelve el nombre del comando anterior.
210      * @memberof MM.UndoManager
211      * @instance
212      */
213     var comandoAnterior = function () {
214         return comandos[actual].tipo;
215     };
216
217     /**
218      * @desc Devuelve el nombre del comando actual.
219      * @memberof MM.UndoManager
220      * @instance
221      */
222     var comandoActual = function () {
223         return comandos[actual].tipo;
224     };
225
226     /**
227      * @desc Devuelve el nombre del comando anterior.
228      * @memberof MM.UndoManager
229      * @instance
230      */
231     var comandoAnterior = function () {
232         return comandos[actual].tipo;
233     };
234
235     /**
236      * @desc Devuelve el nombre del comando actual.
237      * @memberof MM.UndoManager
238      * @instance
239      */
240     var comandoActual = function () {
241         return comandos[actual].tipo;
242     };
243
244     /**
245      * @desc Devuelve el nombre del comando anterior.
246      * @memberof MM.UndoManager
247      * @instance
248      */
249     var comandoAnterior = function () {
250         return comandos[actual].tipo;
251     };
252
253     /**
254      * @desc Devuelve el nombre del comando actual.
255      * @memberof MM.UndoManager
256      * @instance
257      */
258     var comandoActual = function () {
259         return comandos[actual].tipo;
260     };
261
262     /**
263      * @desc Devuelve el nombre del comando anterior.
264      * @memberof MM.UndoManager
265      * @instance
266      */
267     var comandoAnterior = function () {
268         return comandos[actual].tipo;
269     };
270
271     /**
272      * @desc Devuelve el nombre del comando actual.
273      * @memberof MM.UndoManager
274      * @instance
275      */
276     var comandoActual = function () {
277         return comandos[actual].tipo;
278     };
279
280     /**
281      * @desc Devuelve el nombre del comando anterior.
282      * @memberof MM.UndoManager
283      * @instance
284      */
285     var comandoAnterior = function () {
286         return comandos[actual].tipo;
287     };
288
289     /**
290      * @desc Devuelve el nombre del comando actual.
291      * @memberof MM.UndoManager
292      * @instance
293      */
294     var comandoActual = function () {
295         return comandos[actual].tipo;
296     };
297
298     /**
299      * @desc Devuelve el nombre del comando anterior.
300      * @memberof MM.UndoManager
301      * @instance
302      */
303     var comandoAnterior = function () {
304         return comandos[actual].tipo;
305     };
306
307     /**
308      * @desc Devuelve el nombre del comando actual.
309      * @memberof MM.UndoManager
310      * @instance
311      */
312     var comandoActual = function () {
313         return comandos[actual].tipo;
314     };
315
316     /**
317      * @desc Devuelve el nombre del comando anterior.
318      * @memberof MM.UndoManager
319      * @instance
320      */
321     var comandoAnterior = function () {
322         return comandos[actual].tipo;
323     };
324
325     /**
326      * @desc Devuelve el nombre del comando actual.
327      * @memberof MM.UndoManager
328      * @instance
329      */
330     var comandoActual = function () {
331         return comandos[actual].tipo;
332     };
333
334     /**
335      * @desc Devuelve el nombre del comando anterior.
336      * @memberof MM.UndoManager
337      * @instance
338      */
339     var comandoAnterior = function () {
340         return comandos[actual].tipo;
341     };
342
343     /**
344      * @desc Devuelve el nombre del comando actual.
345      * @memberof MM.UndoManager
346      * @instance
347      */
348     var comandoActual = function () {
349         return comandos[actual].tipo;
350     };
351
352     /**
353      * @desc Devuelve el nombre del comando anterior.
354      * @memberof MM.UndoManager
355      * @instance
356      */
357     var comandoAnterior = function () {
358         return comandos[actual].tipo;
359     };
360
361     /**
362      * @desc Devuelve el nombre del comando actual.
363      * @memberof MM.UndoManager
364      * @instance
365      */
366     var comandoActual = function () {
367         return comandos[actual].tipo;
368     };
369
370     /**
371      * @desc Devuelve el nombre del comando anterior.
372      * @memberof MM.UndoManager
373      * @instance
374      */
375     var comandoAnterior = function () {
376         return comandos[actual].tipo;
377     };
378
379     /**
380      * @desc Devuelve el nombre del comando actual.
381      * @memberof MM.UndoManager
382      * @instance
383      */
384     var comandoActual = function () {
385         return comandos[actual].tipo;
386     };
387
388     /**
389      * @desc Devuelve el nombre del comando anterior.
390      * @memberof MM.UndoManager
391      * @instance
392      */
393     var comandoAnterior = function () {
394         return comandos[actual].tipo;
395     };
396
397     /**
398      * @desc Devuelve el nombre del comando actual.
399      * @memberof MM.UndoManager
400      * @instance
401      */
402     var comandoActual = function () {
403         return comandos[actual].tipo;
404     };
405
406     /**
407      * @desc Devuelve el nombre del comando anterior.
408      * @memberof MM.UndoManager
409      * @instance
410      */
411     var comandoAnterior = function () {
412         return comandos[actual].tipo;
413     };
414
415     /**
416      * @desc Devuelve el nombre del comando actual.
417      * @memberof MM.UndoManager
418      * @instance
419      */
420     var comandoActual = function () {
421         return comandos[actual].tipo;
422     };
423
424     /**
425      * @desc Devuelve el nombre del comando anterior.
426      * @memberof MM.UndoManager
427      * @instance
428      */
429     var comandoAnterior = function () {
430         return comandos[actual].tipo;
431     };
432
433     /**
434      * @desc Devuelve el nombre del comando actual.
435      * @memberof MM.UndoManager
436      * @instance
437      */
438     var comandoActual = function () {
439         return comandos[actual].tipo;
440     };
441
442     /**
443      * @desc Devuelve el nombre del comando anterior.
444      * @memberof MM.UndoManager
445      * @instance
446      */
447     var comandoAnterior = function () {
448         return comandos[actual].tipo;
449     };
450
451     /**
452      * @desc Devuelve el nombre del comando actual.
453      * @memberof MM.UndoManager
454      * @instance
455      */
456     var comandoActual = function () {
457         return comandos[actual].tipo;
458     };
459
460     /**
461      * @desc Devuelve el nombre del comando anterior.
462      * @memberof MM.UndoManager
463      * @instance
464      */
465     var comandoAnterior = function () {
466         return comandos[actual].tipo;
467     };
468
469     /**
470      * @desc Devuelve el nombre del comando actual.
471      * @memberof MM.UndoManager
472      * @instance
473      */
474     var comandoActual = function () {
475         return comandos[actual].tipo;
476     };
477
478     /**
479      * @desc Devuelve el nombre del comando anterior.
480      * @memberof MM.UndoManager
481      * @instance
482      */
483     var comandoAnterior = function () {
484         return comandos[actual].tipo;
485     };
486
487     /**
488      * @desc Devuelve el nombre del comando actual.
489      * @memberof MM.UndoManager
490      * @instance
491      */
492     var comandoActual = function () {
493         return comandos[actual].tipo;
494     };
495
496     /**
497      * @desc Devuelve el nombre del comando anterior.
498      * @memberof MM.UndoManager
499      * @instance
500      */
501     var comandoAnterior = function () {
502         return comandos[actual].tipo;
503     };
504
505     /**
506      * @desc Devuelve el nombre del comando actual.
507      * @memberof MM.UndoManager
508      * @instance
509      */
510     var comandoActual = function () {
511         return comandos[actual].tipo;
512     };
513
514     /**
515      * @desc Devuelve el nombre del comando anterior.
516      * @memberof MM.UndoManager
517      * @instance
518      */
519     var comandoAnterior = function () {
520         return comandos[actual].tipo;
521     };
522
523     /**
524      * @desc Devuelve el nombre del comando actual.
525      * @memberof MM.UndoManager
526      * @instance
527      */
528     var comandoActual = function () {
529         return comandos[actual].tipo;
530     };
531
532     /**
533      * @desc Devuelve el nombre del comando anterior.
534      * @memberof MM.UndoManager
535      * @instance
536      */
537     var comandoAnterior = function () {
538         return comandos[actual].tipo;
539     };
540
541     /**
542      * @desc Devuelve el nombre del comando actual.
543      * @memberof MM.UndoManager
544      * @instance
545      */
546     var comandoActual = function () {
547         return comandos[actual].tipo;
548     };
549
550     /**
551      * @desc Devuelve el nombre del comando anterior.
552      * @memberof MM.UndoManager
553      * @instance
554      */
555     var comandoAnterior = function () {
556         return comandos[actual].tipo;
557     };
558
559     /**
560      * @desc Devuelve el nombre del comando actual.
561      * @memberof MM.UndoManager
562      * @instance
563      */
564     var comandoActual = function () {
565         return comandos[actual].tipo;
566     };
567
568     /**
569      * @desc Devuelve el nombre del comando anterior.
570      * @memberof MM.UndoManager
571      * @instance
572      */
573     var comandoAnterior = function () {
574         return comandos[actual].tipo;
575     };
576
577     /**
578      * @desc Devuelve el nombre del comando actual.
579      * @memberof MM.UndoManager
580      * @instance
581      */
582     var comandoActual = function () {
583         return comandos[actual].tipo;
584     };
585
586     /**
587      * @desc Devuelve el nombre del comando anterior.
588      * @memberof MM.UndoManager
589      * @instance
590      */
591     var comandoAnterior = function () {
592         return comandos[actual].tipo;
593     };
594
595     /**
596      * @desc Devuelve el nombre del comando actual.
597      * @memberof MM.UndoManager
598      * @instance
599      */
600     var comandoActual = function () {
601         return comandos[actual].tipo;
602     };
603
604     /**
605      * @desc Devuelve el nombre del comando anterior.
606      * @memberof MM.UndoManager
607      * @instance
608      */
609     var comandoAnterior = function () {
610         return comandos[actual].tipo;
611     };
612
613     /**
614      * @desc Devuelve el nombre del comando actual.
615      * @memberof MM.UndoManager
616      * @instance
617      */
618     var comandoActual = function () {
619         return comandos[actual].tipo;
620     };
621
622     /**
623      * @desc Devuelve el nombre del comando anterior.
624      * @memberof MM.UndoManager
625      * @instance
626      */
627     var comandoAnterior = function () {
628         return comandos[actual].tipo;
629     };
630
631     /**
632      * @desc Devuelve el nombre del comando actual.
633      * @memberof MM.UndoManager
634      * @instance
635      */
636     var comandoActual = function () {
637         return comandos[actual].tipo;
638     };
639
640     /**
641      * @desc Devuelve el nombre del comando anterior.
642      * @memberof MM.UndoManager
643      * @instance
644      */
645     var comandoAnterior = function () {
646         return comandos[actual].tipo;
647     };
648
649     /**
650      * @desc Devuelve el nombre del comando actual.
651      * @memberof MM.UndoManager
652      * @instance
653      */
654     var comandoActual = function () {
655         return comandos[actual].tipo;
656     };
657
658     /**
659      * @desc Devuelve el nombre del comando anterior.
660      * @memberof MM.UndoManager
661      * @instance
662      */
663     var comandoAnterior = function () {
664         return comandos[actual].tipo;
665     };
666
667     /**
668      * @desc Devuelve el nombre del comando actual.
669      * @memberof MM.UndoManager
670      * @instance
671      */
672     var comandoActual = function () {
673         return comandos[actual].tipo;
674     };
675
676     /**
677      * @desc Devuelve el nombre del comando anterior.
678      * @memberof MM.UndoManager
679      * @instance
680      */
681     var comandoAnterior = function () {
682         return comandos[actual].tipo;
683     };
684
685     /**
686      * @desc Devuelve el nombre del comando actual.
687      * @memberof MM.UndoManager
688      * @instance
689      */
690     var comandoActual = function () {
691         return comandos[actual].tipo;
692     };
693
694     /**
695      * @desc Devuelve el nombre del comando anterior.
696      * @memberof MM.UndoManager
697      * @instance
698      */
699     var comandoAnterior = function () {
700         return comandos[actual].tipo;
701     };
702
703     /**
704      * @desc Devuelve el nombre del comando actual.
705      * @memberof MM.UndoManager
706      * @instance
707      */
708     var comandoActual = function () {
709         return comandos[actual].tipo;
710     };
711
712     /**
713      * @desc Devuelve el nombre del comando anterior.
714      * @memberof MM.UndoManager
715      * @instance
716      */
717     var comandoAnterior = function () {
718         return comandos[actual].tipo;
719     };
720
721     /**
722      * @desc Devuelve el nombre del comando actual.
723      * @memberof MM.UndoManager
724      * @instance
725      */
726     var comandoActual = function () {
727         return comandos[actual].tipo;
728     };
729
730     /**
731      * @desc Devuelve el nombre del comando anterior.
732      * @memberof MM.UndoManager
733      * @instance
734      */
735     var comandoAnterior = function () {
736         return comandos[actual].tipo;
737     };
738
739     /**
740      * @desc Devuelve el nombre del comando actual.
741      * @memberof MM.UndoManager
742      * @instance
743      */
744     var comandoActual = function () {
745         return comandos[actual].tipo;
746     };
747
748     /**
749      * @desc Devuelve el nombre del comando anterior.
750      * @memberof MM.UndoManager
751      * @instance
752      */
753     var comandoAnterior = function () {
754         return comandos[actual].tipo;
755     };
756
757     /**
758      * @desc Devuelve el nombre del comando actual.
759      * @memberof MM.UndoManager
760      * @instance
761      */
762     var comandoActual = function () {
763         return comandos[actual].tipo;
764     };
765
766     /**
767      * @desc Devuelve el nombre del comando anterior.
768      * @memberof MM.UndoManager
769      * @instance
770      */
771     var comandoAnterior = function () {
772         return comandos[actual].tipo;
773     };
774
775     /**
776      * @desc Devuelve el nombre del comando actual.
777      * @memberof MM.UndoManager
778      * @instance
779      */
780     var comandoActual = function () {
781         return comandos[actual].tipo;
782     };
783
784     /**
785      * @desc Devuelve el nombre del comando anterior.
786      * @memberof MM.UndoManager
787      * @instance
788      */
789     var comandoAnterior = function () {
790         return comandos[actual].tipo;
791     };
792
793     /**
794      * @desc Devuelve el nombre del comando actual.
795      * @memberof MM.UndoManager
796      * @instance
797      */
798     var comandoActual = function () {
799         return comandos[actual].tipo;
800     };
801
802     /**
803      * @desc Devuelve el nombre del comando anterior.
804      * @memberof MM.UndoManager
805      * @instance
806      */
807     var comandoAnterior = function () {
808         return comandos[actual].tipo;
809     };
810
811     /**
812      * @desc Devuelve el nombre del comando actual.
813      * @memberof MM.UndoManager
814      * @instance
815      */
816     var comandoActual = function () {
817         return comandos[actual].tipo;
818     };
819
820     /**
821      * @desc Devuelve el nombre del comando anterior.
822      * @memberof MM.UndoManager
823      * @instance
824      */
825     var comandoAnterior = function () {
826         return comandos[actual].tipo;
827     };
828
829     /**
830      * @desc Devuelve el nombre del comando actual.
831      * @memberof MM.UndoManager
832      * @instance
833      */
834     var comandoActual = function () {
835         return comandos[actual].tipo;
836     };
837
838     /**
839      * @desc Devuelve el nombre del comando anterior.
840      * @memberof MM.UndoManager
841      * @instance
842      */
843     var comandoAnterior = function () {
844         return comandos[actual].tipo;
845     };
846
847     /**
848      * @desc Devuelve el nombre del comando actual.
849      * @memberof MM.UndoManager
850      * @instance
851      */
852     var comandoActual = function () {
853         return comandos[actual].tipo;
854     };
855
856     /**
857      * @desc Devuelve el nombre del comando anterior.
858      * @memberof MM.UndoManager
859      * @instance
860      */
861     var comandoAnterior = function () {
862         return comandos[actual].tipo;
863     };
864
865     /**
866      * @desc Devuelve el nombre del comando actual.
867      * @memberof MM.UndoManager
868      * @instance
869      */
870     var comandoActual = function () {
871         return comandos[actual].tipo;
872     };
873
874     /**
875      * @desc Devuelve el nombre del comando anterior.
876      * @memberof MM.UndoManager
877      * @instance
878      */
879     var comandoAnterior = function () {
880         return comandos[actual].tipo;
881     };
882
883     /**
884      * @desc Devuelve el nombre del comando actual.
885      * @memberof MM.UndoManager
886      * @instance
887      */
888     var comandoActual = function () {
889         return comandos[actual].tipo;
890     };
891
892     /**
893      * @desc Devuelve el nombre del comando anterior.
894      * @memberof MM.UndoManager
895      * @instance
896      */
897     var comandoAnterior = function () {
898         return comandos[actual].tipo;
899     };
900
901     /**
902      * @desc Devuelve el nombre del comando actual.
903      * @memberof MM.UndoManager
904      * @instance
905      */
906     var comandoActual = function () {
907         return comandos[actual].tipo;
908     };
909
910     /**
911      * @desc Devuelve el nombre del comando anterior.
912      * @memberof MM.UndoManager
913      * @instance
914      */
915     var comandoAnterior = function () {
916         return comandos[actual].tipo;
917     };
918
919     /**
920      * @desc Devuelve el nombre del comando actual.
921      * @memberof MM.UndoManager
922      * @instance
923      */
924     var comandoActual = function () {
925         return comandos[actual].tipo;
926     };
927
928     /**
929      * @desc Devuelve el nombre del comando anterior.
930      * @memberof MM.UndoManager
931      * @instance
932      */
933     var comandoAnterior = function () {
934         return comandos[actual].tipo;
935     };
936
937     /**
938      * @desc Devuelve el nombre del comando actual.
939      * @memberof MM.UndoManager
940      * @instance
941      */
942     var comandoActual = function () {
943         return comandos[actual].tipo;
944     };
945
946     /**
947      * @desc Devuelve el nombre del comando anterior.
948      * @memberof MM.UndoManager
949      * @instance
950      */
951     var comandoAnterior = function () {
952         return comandos[actual].tipo;
953     };
954
955     /**
956      * @desc Devuelve el nombre del comando actual.
957      * @memberof MM.UndoManager
958      * @instance
959      */
960     var comandoActual = function () {
961         return comandos[actual].tipo;
962     };
963
964     /**
965      * @desc Devuelve el nombre del comando anterior.
966      * @memberof MM.UndoManager
967      * @instance
968      */
969     var comandoAnterior = function () {
970         return comandos[actual].tipo;
971     };
972
973     /**
974      * @desc Devuelve el nombre del comando actual.
975      * @memberof MM.UndoManager
976      * @instance
977      */
978     var comandoActual = function () {
979         return comandos[actual].tipo;
980     };
981
982     /**
983      * @desc Devuelve el nombre del comando anterior.
984      * @memberof MM.UndoManager
985      * @instance
986      */
987     var comandoAnterior = function () {
988         return comandos[actual].tipo;
989     };
990
991     /**
992      * @desc Devuelve el nombre del comando actual.
993      * @memberof MM.UndoManager
994      * @instance
995      */
996     var comandoActual = function () {
997         return comandos[actual].tipo;
1000    };
1001
1002    /**
1003     * @desc Devuelve el nombre del comando anterior.
1004     * @memberof MM.UndoManager
1005     * @instance
1006     */
1007    var comandoAnterior = function () {
1008        return comandos[actual].tipo;
1009    };
1010
1011    /**
1012     * @desc Devuelve el nombre del comando actual.
1013     * @memberof MM.UndoManager
1014     * @instance
1015     */
1016    var comandoActual = function () {
1017        return comandos[actual].tipo;
1018    };
1019
1020    /**
1021     * @desc Devuelve el nombre del comando anterior.
1022     * @memberof MM.UndoManager
1023     * @instance
1024     */
1025    var comandoAnterior = function () {
1026        return comandos[actual].tipo;
1027    };
1028
1029    /**
1030     * @desc Devuelve el nombre del comando actual.
1031     * @memberof MM.UndoManager
1032     * @instance
1033     */
1034    var comandoActual = function () {
1035        return comandos[actual].tipo;
1036    };
1037
1038    /**
1039     * @desc Devuelve el nombre del comando anterior.
1040     * @memberof MM.UndoManager
1041     * @instance
1042     */
1043    var comandoAnterior = function () {
1044        return comandos[actual].tipo;
1045    };
1046
1047    /**
1048     * @desc Devuelve el nombre del comando actual.
1049     * @memberof MM.UndoManager
1050     * @instance
1051     */
1052    var comandoActual = function () {
1053        return comandos[actual].tipo;
1054    };
1055
1056    /**
1057     * @desc Devuelve el nombre del comando anterior.
1058     * @memberof MM.UndoManager
1059     * @instance
1060     */
1061    var comandoAnterior = function () {
1062        return comandos[actual].tipo;
1063    };
1064
1065    /**
1066     * @desc Devuelve el nombre del comando actual.
1067     * @memberof MM.UndoManager
1068     * @instance
1069     */
1070    var comandoActual = function () {
1071        return comandos[actual].tipo;
1072    };
1073
1074    /**
1075     * @desc Devuelve el nombre del comando anterior.
1076     * @memberof MM.UndoManager
1077     * @instance
1078     */
1079    var comandoAnterior = function () {
1080        return comandos[actual].tipo;
1081    };
1082
1083    /**
1084     * @desc Devuelve el nombre del comando actual.
1085     * @memberof MM.UndoManager
1086     * @instance
1087     */
1088    var comandoActual = function () {
1089        return comandos[actual].tipo;
1090    };
1091
1092    /**
1093     * @desc Devuelve el nombre del comando anterior.
1094     * @memberof MM.UndoManager
1095     * @instance
1096     */
1097    var comandoAnterior = function () {
1098        return comandos[actual].tipo;
1099    };
1100
1101    /**
1102     * @desc Devuelve el nombre del comando actual.
1103     * @memberof MM.UndoManager
1104     * @instance
1105     */
1106    var comandoActual = function () {
1107        return comandos[actual].tipo;
1108    };
1109
1110    /**
1111     * @desc Devuelve el nombre del comando anterior.
1112     * @memberof MM.UndoManager
1113     * @instance
1114     */
1115    var comandoAnterior = function () {
1116        return comandos[actual].tipo;
1117    };
1118
1119    /**
1120     * @desc Devuelve el nombre del comando actual.
1121     * @memberof MM.UndoManager
1122     * @instance
1123     */
1124    var comandoActual = function () {
1125        return comandos[actual].tipo;
1126    };
1127
1128    /**
1129     * @desc Devuelve el nombre del comando anterior.
1130     * @memberof MM.UndoManager
1131     * @instance
1132     */
1133    var comandoAnterior = function () {
1134        return comandos[actual].tipo;
1135    };
1136
1137    /**
1138     * @desc Devuelve el nombre del comando actual.
1139     * @memberof MM.UndoManager
1140     * @instance
1141     */
1142    var comandoActual = function () {
1143        return comandos[actual].tipo;
1144    };
1145
1146    /**
1147     * @desc Devuelve el nombre del comando anterior.
1148     * @memberof MM.UndoManager
1149     * @instance
1150     */
1151    var comandoAnterior = function () {
1152        return comandos[actual].tipo;
1153    };
1154
1155    /**
1156     * @desc Devuelve el nombre del comando actual.
1157     * @memberof MM.UndoManager
1158     * @instance
1159     */
1160    var comandoActual = function () {
1161        return comandos[actual].tipo;
1162    };
1163
1164    /**
1165     * @desc Devuelve el nombre del comando anterior.
1166     * @memberof MM.UndoManager
1167     * @instance
1168     */
1169    var comandoAnterior = function () {
1170        return comandos[actual].tipo;
1171    };
1172
1173    /**
1174     * @desc Devuelve el nombre del comando actual.
1175     * @memberof MM.UndoManager
1176     * @instance
1177     */
1178    var comandoActual = function () {
1179        return comandos[actual].tipo;
1180    };
1181
1182    /**
1183     * @desc Devuelve el nombre del comando anterior.
1184     * @memberof MM.UndoManager
1185     * @instance
1186     */
1187    var comandoAnterior = function () {
1188        return comandos[actual].tipo;
1189    };
1190
1191    /**
1192     * @desc Devuelve el nombre del comando actual.
1193     * @memberof MM.UndoManager
1194     * @instance
1195     */
1196    var comandoActual = function () {
1197        return comandos[actual].tipo;
1198    };
1199
1200    /**
1201     * @desc Devuelve el nombre del comando anterior.
1202     * @memberof MM.UndoManager
1203     * @instance
1204     */
1205    var comandoAnterior = function () {
1206        return comandos[actual].tipo;
1207    };
1208
1209    /**
1210     * @desc Devuelve el nombre del comando actual.
1211     * @memberof MM.UndoManager
1212     * @instance
1213     */
1214    var comandoActual = function () {
1215        return comandos[actual].tipo;
1216    };
1217
1218    /**
1219     * @desc Devuelve el nombre del comando anterior.
1220     * @memberof MM.UndoManager
1221     * @instance
1222     */
1223    var comandoAnterior = function () {
1224        return comandos[actual].tipo;
1225    };
1226
1227    /**
1228     * @desc Devuelve el nombre del comando actual.
1229     * @memberof MM.UndoManager
1230     * @instance
1231     */
1232    var comandoActual = function () {
1233        return comandos[actual].tipo;
1234    };
1235
1236    /**
1237     * @desc Devuelve el nombre del comando anterior.
1238     * @memberof MM.UndoManager
1239     * @instance
1240     */
1241    var comandoAnterior = function () {
1242        return comandos[actual].tipo;
1243    };
1244
1245    /**
1246     * @desc Devuelve el nombre del comando actual.
1247     * @memberof MM.UndoManager
1248     * @instance
1249     */
1250    var comandoActual = function () {
1251        return comandos[actual].tipo;
1252    };
1253
1254    /**
1255     * @desc Devuelve el nombre del comando anterior.
1256     * @memberof MM.UndoManager
1257     * @instance
1258     */
1259    var comandoAnterior = function () {
1260        return comandos[
```

```

59     comandos.push(comando);
60     actual = comandos.length - 1;
61     ajustarMaximo();
62     eventos.on('add');
63     eventos.on('cambio');
64 };
65
66 var borrarPorEncimaActual = function () {
67     if ( actual !== -1 && actual < comandos.length - 1 ){
68         comandos = comandos.slice(0,actual+1);
69     }
70 };
71
72 var ajustarMaximo = function () {
73     if ( actual === maxComandos ){
74         comandos.shift();
75         actual--;
76     }
77 };
78
79 /**
80  * @desc Ejecuta el comando hacer correspondiente, según el comando actual. También
81  *     hace avanzar
82  *     el puntero actual. El comando que se ejecuta o (hace) es el siguiente al
83  *     comando actual.
84  *     Si el comando actual es último no hay comando hacer, o no hay que hacer nada
85  *
86  * @memberof MM.UndoManager
87  * @instance
88  */
89 var hacer = function () {
90     if ( comandos[actual+1] ) {
91         comandos[actual+1].hacer();
92         avanzar();
93         eventos.on('hacer');
94         eventos.on('cambio');
95     }
96 };
97
98 /**
99  * @desc Ejecuta el comando deshacer correspondiente, según el comando actual.
100  *     También hace
101  *     retroceder el puntero actual.
102  * @memberof MM.UndoManager
103  * @instance
104  */
105 var deshacer = function () {
106     if ( actual !== -1 ) {
107         comandos[actual].deshacer();
108         retroceder();
109         eventos.on('deshacer');
110         eventos.on('cambio');
111     }
112 };
113
114 var avanzar = function () {
115     if (actual < comandos.length - 1) {
116         actual++;
117         eventos.on('avanzar');
118         eventos.on('cambio');
119     }
120 };
121
122 var retroceder = function () {
123     if (actual >= 0) {
124         actual--;
125         eventos.on('retroceder');
126         eventos.on('cambio');
127     }
128 };
129
130 /**
131  * @desc Calcula el nombre del comando a Hacer según la situación actual.
132  * @return {String} nombre del comando hacer.
133  * @memberof MM.UndoManager
134  * @instance
135  */
136 var hacerNombre = function () {
137     if ( comandos[actual+1] ) {
138         return comandos[actual+1].nombre;
139     }
140     return null;

```

```

137     };
138
139     /**
140     * @desc Calcula el nombre del comando a deshacer según la situación actual.
141     * @return {String} nombre del comando deshacer.
142     * @memberof MM.UndoManager
143     * @instance
144     */
145     var deshacerNombre = function () {
146         if ( actual !== -1 ) {
147             return comandos[actual].nombre;
148         }
149         return null;
150     };
151
152
153     /**
154     * @desc Genera un array con los nombres de los comandos
155     * @return {Array} Array con los nombres de los comandos
156     * @memberof MM.UndoManager
157     * @instance
158     */
159     var nombres = function () {
160         return comandos.map(function (c) { return c.nombre; });
161     };
162
163     return {
164         init : init,
165         nombres : nombres,
166         hacerNombre : hacerNombre,
167         deshacerNombre: deshacerNombre,
168         /**
169         * @desc Indica el índice actual dentro de la lista de comandos.
170         * @return {Integer} índice actual
171         * @memberof MM.UndoManager
172         * @instance
173         */
174         actual : function () { return actual; },
175         add : add,
176         hacer : hacer,
177         deshacer : deshacer,
178         /**
179         * @prop {MM.PubSub} eventos Gestor de eventos del undoManager
180         * @memberof MM.UndoManager
181         * @instance
182         */
183         eventos : eventos
184     };
185 }());
186
187 /**
188 * @class MM.UndoManager.ComandoHacerDeshacer
189 * @classdesc Clase base para el comportamiento de una comando hacer/deshacer (undo/redo)
190 *
191 * @constructor
192 * @param {string} nombre Nombre del comando
193 * @param {function} hacerCallBack Función a ejecutar en el hacer.
194 * @param {function} deshacerCallBack Función a ejecutar en el deshacer
195 */
196 MM.UndoManager.ComandoHacerDeshacer = MM.Class.extend(
197 /** @lends MM.UndoManager.ComandoHacerDeshacer.prototype */{
198     init: function (nombre, hacerCallBack, deshacerCallBack) {
199         this.nombre = nombre;
200         this.hacerCallBack = hacerCallBack;
201         this.deshacerCallBack = deshacerCallBack;
202     },
203     /**
204     * @desc Ejecuta el comando hacer
205     * @memberof MM.UndoManager.ComandoHacerDeshacer
206     * @instance
207     */
208     hacer : function () {
209         this.hacerCallBack();
210     },
211     /**
212     * @desc Ejecuta el comando deshacer
213     * @memberof MM.UndoManager.ComandoHacerDeshacer
214     * @instance
215     */
216     deshacer : function () {

```

```

218         this.deshacerCallBack();
219     }
220 });
221
222
223 if ( typeof module !== 'undefined' ) {
224     module.exports.UndoManager = MM.UndoManager;
225 }

```

La clase base de todos los comandos hacer y deshacer es MM.UndoManager.ComandoHacerDeshacer.

Esta clase implementa un patrón comando con una variante obvia, que en realidad tiene registra dos comandos. Uno para hacer y otro para deshacer.

```

1  /**
2   * @file undoManager.js Implementación de un gestor de comandos hacer y deshacer
3   * @author José Luis Molina Soria
4   * @version 20130620
5   */
6
7  if ( typeof module !== 'undefined' ) {
8      var MM = require('./MindMapJS.js');
9      MM.Class = require('./klass.js');
10     MM.PubSub = require('./pubsub.js');
11 }
12
13 /**
14  * @class MM.UndoManager
15  * @classdesc Gestor de comandos undo (hacer y deshacer).
16  * @constructor
17  * @param maximo {integer} El máximo de comando en buffer. Por defecto, 10.
18  */
19 MM.UndoManager = MM.Class.extend(function() {
20     /**
21      * @prop {Array} Comando del tipo Hacer / Deshacer
22      * @memberof MM.UndoManager
23      * @inner
24      */
25     var comandos = []; // la lista de comandos
26
27     /**
28      * @prop {integer} Tamaño máximo del buffer
29      * @memberof MM.UndoManager
30      * @inner
31      */
32     var maxComandos = 10; // número máximo de comandos en cola
33
34     /**
35      * @prop {integer} Índice del comando actual
36      * @memberof MM.UndoManager
37      * @inner
38      */
39     var actual = -1; // índice comando actual
40
41     var eventos = new MM.PubSub();
42
43     var init = function ( maximo ) {
44         maxComandos = maximo || 10;
45     };
46
47     /**
48      * @desc Añade un nuevo comando a la pila de comandos. Si el tamaño del buffer
49      *       sobrepasa el
50      *       máximo fijado, entonces elimina el comando más antiguo. Si existiesen
51      *       comandos por
52      *       encima del actual, estos serán eliminados.
53      * @param {MM.UndoManager.ComandoHacerDeshacer} Comando a añadir al buffer.
54      * @memberof MM.UndoManager
55      * @instance
56      */
57     var add = function (comando) {
58         borrarPorEncimaActual();
59         comandos.push(comando);
60         actual = comandos.length - 1;
61         ajustarMaximo();
62         eventos.on('add');

```

```

63     eventos.on('cambio');
64 };
65
66 var borrarPorEncimaActual = function () {
67     if ( actual !== -1 && actual < comandos.length -1 ){
68         comandos = comandos.slice(0,actual+1);
69     }
70 };
71
72 var ajustarMaximo = function () {
73     if ( actual === maxComandos ){
74         comandos.shift();
75         actual--;
76     }
77 };
78
79 /**
80  * @desc Ejecuta el comando hacer correspondiente, según el comando actual. También
81  *     hace avanzar
82  *     el puntero actual. El comando que se ejecuta o (hace) es el siguiente al
83  *     comando actual.
84  *     Si el comando actual es último no hay comando hacer, o no hay que hacer nada
85  *
86  * @memberof MM.UndoManager
87  * @instance
88  */
89 var hacer = function () {
90     if ( comandos[actual+1] ) {
91         comandos[actual+1].hacer();
92         avanzar();
93         eventos.on('hacer');
94         eventos.on('cambio');
95     }
96 };
97
98 /**
99  * @desc Ejecuta el comando deshacer correspondiente, según el comando actual.
100  *     También hace
101  *     retroceder el puntero actual.
102  * @memberof MM.UndoManager
103  * @instance
104  */
105 var deshacer = function () {
106     if ( actual !== -1 ) {
107         comandos[actual].deshacer();
108         retroceder();
109         eventos.on('deshacer');
110         eventos.on('cambio');
111     }
112 };
113
114 var avanzar = function () {
115     if (actual < comandos.length - 1) {
116         actual++;
117         eventos.on('avanzar');
118         eventos.on('cambio');
119     }
120 };
121
122 var retroceder = function () {
123     if (actual >= 0) {
124         actual--;
125         eventos.on('retroceder');
126         eventos.on('cambio');
127     }
128 };
129
130 /**
131  * @desc Calcula el nombre del comando a Hacer según la situación actual.
132  * @return {String} nombre del comando hacer.
133  * @memberof MM.UndoManager
134  * @instance
135  */
136 var hacerNombre = function () {
137     if ( comandos[actual+1] ) {
138         return comandos[actual+1].nombre;
139     }
140     return null;
141 };
142
143 /**
144  * @desc Calcula el nombre del comando a deshacer según la situación actual.

```

```

141 * @return {String} nombre del comando deshacer.
142 * @memberof MM.UndoManager
143 * @instance
144 */
145 var deshacerNombre = function () {
146   if ( actual !== -1 ) {
147     return comandos[actual].nombre;
148   }
149   return null;
150 };
151
152
153 /**
154 * @desc Genera un array con los nombres de los comandos
155 * @return {Array} Array con los nombres de los comandos
156 * @memberof MM.UndoManager
157 * @instance
158 */
159 var nombres = function () {
160   return comandos.map(function (c) { return c.nombre; });
161 };
162
163 return {
164   init : init,
165   nombres : nombres,
166   hacerNombre : hacerNombre,
167   deshacerNombre: deshacerNombre,
168   /**
169    * @desc Indica el indice actual dentro de la lista de comandos.
170    * @return {Integer} indice actual
171    * @memberof MM.UndoManager
172    * @instance
173    */
174   actual : function () { return actual; },
175   add : add,
176   hacer : hacer,
177   deshacer : deshacer,
178   /**
179    * @prop {MM.PubSub} eventos Gestor de eventos del undoManager
180    * @memberof MM.UndoManager
181    * @instance
182    */
183   eventos : eventos
184 };
185 }());
186
187 /**
188 * @class MM.UndoManager.ComandoHacerDeshacer
189 * @classdesc Clase base para el comportamiento de una comando hacer/deshacer (undo/redo)
190 *
191 * @constructor
192 * @param {string} nombre Nombre del comando
193 * @param {function} hacerCallBack Función a ejecutar en el hacer.
194 * @param {function} deshacerCallBack Función a ejecutar en el deshacer
195 */
196 MM.UndoManager.ComandoHacerDeshacer = MM.Class.extend(
197 /** @lends MM.UndoManager.ComandoHacerDeshacer.prototype */{
198   init: function (nombre, hacerCallBack, deshacerCallBack) {
199     this.nombre = nombre;
200     this.hacerCallBack = hacerCallBack;
201     this.deshacerCallBack = deshacerCallBack;
202   },
203   /**
204    * @desc Ejecuta el comando hacer
205    * @memberof MM.UndoManager.ComandoHacerDeshacer
206    * @instance
207    */
208   hacer : function () {
209     this.hacerCallBack();
210   },
211   /**
212    * @desc Ejecuta el comando deshacer
213    * @memberof MM.UndoManager.ComandoHacerDeshacer
214    * @instance
215    */
216   deshacer : function () {
217     this.deshacerCallBack();
218   }
219 });
220
221

```

```

222
223 if ( typeof module !== 'undefined' ) {
224     module.exports.UndoManager = MM.UndoManager;
225 }

```

5.2. Concatenación y UglifyJS

Es conveniente la unificación y compresión de código Javascripts. Con ello, no sólo reducimos el número de peticiones¹⁶, desde el navegador al servidor, sino que optimizamos los tiempos de carga y respuesta del navegador. Este aspecto es siempre deseable, evitando al usuario final esperas indeseadas.

Se ha utilizado la herramienta UglifyJS¹⁷ en MindMapJS, no sólo por tratarse de una librería estándar dentro de herramientas de ofuscación y reducción de código Javascript, sino por los beneficios que nos aporta.

Es sencillo, comprobar que se reduce considerablemente el tiempo, si se concatenan los ficheros Javascript. Por cada fichero Javascript el navegador realiza una petición Get como se puede comprobar en la imagen 5.3.

Name	Path	Method	Status	Type	Initiator	Size Content	Time Latency	Timeline
MindMapJS.js	/MindMapJS/src	GET	200 OK	application...	MM-dev.html:16 Parser	1.0 KB 807 B	71 ms 70 ms	
properties.js	/MindMapJS/src	GET	200 OK	application...	MM-dev.html:17 Parser	963 B 735 B	71 ms 70 ms	
chain.js	/MindMapJS/src	GET	200 OK	application...	MM-dev.html:18 Parser	676 B 449 B	72 ms 71 ms	
processable.js	/MindMapJS/src	GET	200 OK	application...	MM-dev.html:19 Parser	1.1 KB 918 B	73 ms 72 ms	
klass.js	/MindMapJS/src	GET	200 OK	application...	MM-dev.html:20 Parser	2.6 KB 2.4 KB	76 ms 74 ms	
pubsub.js	/MindMapJS/src	GET	200 OK	application...	MM-dev.html:21 Parser	3.1 KB 2.9 KB	76 ms 75 ms	75 ms → 1 ms
arbol-n.js		GET	200 OK		MM-dev.html:22	7.0 KB	78 ms	

Figura 5.3: Carga de ficheros Javascript en desarrollo

Cada solicitud de fichero, por parte del navegador, tiene una latencia de red y un tiempo de carga por parte de navegador que puede provocar, dependiendo de las condiciones y velocidad de la red utilizada, esperas en por parte de usuario final. En la siguiente captura 5.4, veremos como al concatenar reducimos el número de peticiones realizadas al servidor

¹⁶Más concretamente peticiones http get

¹⁷En combinación con GruntJS

y por lo tanto, el tiempo de carga¹⁸.

Name	Path	Method	Status	Type	Initiator	Size Content	Time Latency	Timeline
kinetic-v4.6.0.min.js	/MindMapJS/lib	GET	200 OK	application/javascript	MM-concat.html:14 Parser	93.2 KB	76 ms	
MindMapJS-v0.1.2.js	/MindMapJS/dist	GET	200 OK	application/javascript	MM-concat.html:15 Parser	111 KB	85 ms	
kickstart-buttons.css	/MindMapJS/css/kickstart	GET	200 OK	text/css	MM-concat.html:11 Parser	2.8 KB	10 ms	
kickstart-forms.css	/MindMapJS/css/kickstart	GET	200 OK	text/css	MM-concat.html:11 Parser	1.8 KB	14 ms	
kickstart-menus.css	/MindMapJS/css/kickstart	GET	200 OK	text/css	MM-concat.html:11 Parser	1.9 KB	16 ms	
kickstart-grid.css	/MindMapJS/css/kickstart	GET	200 OK	text/css	MM-concat.html:11 Parser	1.3 KB	19 ms	
jquery.fancybox-1.3.4.css		GET	200 OK	text/css	MM-concat.html:11 Parser	2.0 KB	21 ms	

Figura 5.4: Carga de ficheros Javascript concatenado

Si a su vez comprimimos y reducimos al máximo el tamaño del fichero con UglifyJS, podemos comprobar, en la figura 5.5, como el tiempo de carga de la librería se reduce considerablemente.

Name	Path	Method	Status	Type	Initiator	Size Content	Time Latency	Timeline
d-6IYpIOFocCackzwwXSOD8E0i7KZn-EPh...	themes.googleusercontent.com/static/fonts	GET	200 OK	font/woff	MM.html:9 Parser	(from cache)	0 ms	
jquery.min.js	ajax.googleapis.com/ajax/libs/jquery/1.9.1	GET	200 OK	text/javascript	MM.html:11 Parser	(from cache)	12 ms	
kickstart.js	/MindMapJS/lib	GET	200 OK	application/javascript	MM.html:12 Parser	(from cache)	12 ms	
kinetic-v4.6.0.min.js	/MindMapJS/lib	GET	200 OK	application/javascript	MM.html:14 Parser	(from cache)	12 ms	
MindMapJS-v0.1.2.min.js	/MindMapJS/dist	GET	200 OK	application/javascript	MM.html:15 Parser	40.7 KB	14 ms	
jquery.min.map	ajax.googleapis.com/ajax/libs/jquery/1.9.1	GET	200 OK	application/javascript	Other	(from cache)	8 ms	

Figura 5.5: Carga de ficheros Javascript comprimido

Más concretamente, podemos comprobar que el fichero se ha reducido desde los 111 Kbytes a 40.7 Kbytes. Es decir, se ha reducido el tamaño del fichero más de 35 % y el tiempo de carga a tan sólo 4 ms.

Como se puede comprobar es deseable la concatenación y reducción del código en toda aplicación web. Ya que el tiempo de respuesta y la experiencia de usuario mejoran al evitar tiempos muertos en la carga. Sobre todo en sistemas donde el ancho de banda es reducido.

¹⁸Más concretamente a sólo 16ms en un fichero un único fichero de 111Kb.

5.3. JsHint

JsHint es otra herramienta que no debe faltar en ningún desarrollo web. En MindMapJS se ha utilizado continuamente en los periodos de desarrollo. JsHint nos permite comprobar la validez y calidad de nuestro código Javascript, analizando del código fuente y mostrándonos los puntos en los que puede mejorarse desde el punto de vista de las buenas prácticas y código limpio. Evitando pues, errores o posibles errores de interpretación del código fuente.

He utilizado JsHint por que se trata de un estándar utilizado por multitud de desarrollares web y que tiene una gran aceptación por parte de los grandes la programación web. Salvo quizás Douglas Crockford, autor de la herramienta JSLint.

JSLint es un validador de código muy exhaustivo y da muchos falsos positivos. Además tiene muchos detractores, que alegan que los criterios evaluados son bastante subjetivos, sigue los criterios impuestos por Douglas Crockford¹⁹. Por todo ello, algunos desarrolladores crearon un fork llamado JSHint con el objeto de mejorar las mediciones que eran bastante arbitrarias en JSLint.

5.4. KineticJS

KineticJS es un framework gráfico sobre el canvas de HTML5, permitiendo anidamiento de nodos, capas, filtros, animaciones y manejo de eventos de forma muy sencilla.

La librería KineticJS tiene en lo alto de la jerarquía de Objetos al escenario que puede tener una o más capas. Cada capa se renderiza en dos canvas, uno para contener los elementos gráficos y otro oculto para agilizar y aclarar la detección de eventos. Cada capa puede contener cualquier elemento gráfico.

5.4.1. ¿Por qué usar KineticJS?

Entre otras virtudes, lo que más me atrajo a la hora de utilizar KineticJS es:

¹⁹su creador

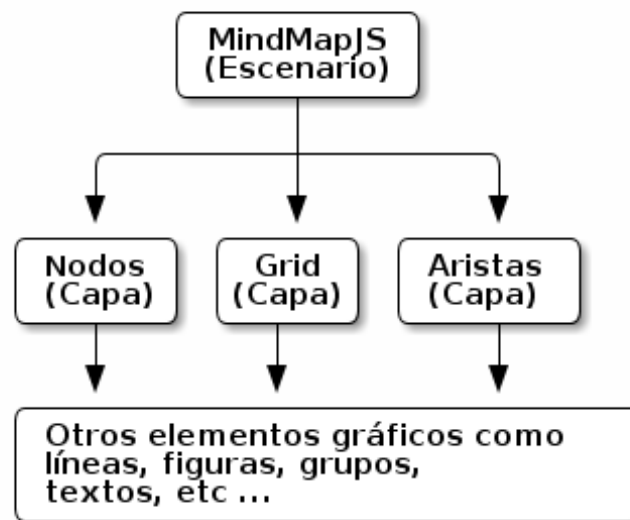


Figura 5.6: Escenario y modelo de capas KineticJS de MindMapJS

- **Su rendimiento.** Presenta un muy buen rendimiento gráfico.
- **Soporte para capas.** Manejo sencillo de capas permitiendo sobreponer capas.
- **API clara y extensible.** Presenta un código claro, orientado a objetos y eventos. Resulta sumamente sencillo extender la librería como se puede ver en el apartado 'Extendiendo KineticJS'.
- **Manejo de eventos.** No sólo a nivel de canvas. También a nivel de figura. Soporta multitud de eventos no sólo de escritorio sino también de dispositivos táctiles.
- **Continuo desarrollo.** El desarrollo y avance de la librería es notable y tiene una comunidad detrás proporcionando ideas, soluciones, y verificando el desarrollo de la librería.

5.4.2. Algunos ejemplos sencillos.

Para todos los ejemplos sobre KineticJS a partir de ahora vamos a utilizar el mismo modelo de página HTML5. En ella, incorporaremos la librería de KineticJS y un contenedor para nuestro lienzo. Una etiqueta div que posteriormente utilizaremos para indicarle a KineticJS donde tiene que pintar.

```

1 <!DOCTYPE HTML>
2 <html>
3   <head>
4     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
5     <style>
6       body {
7         margin: 0px;
8         padding: 0px;
9       }
10      canvas {
11        border: 1px solid blue;
12      }
13    </style>
14    <script src="../../lib/kinetic-v4.4.3.min.js"></script>
15    <script src="1_eventos.js"></script>
16  </head>
17  <body>
18    <div id="container" style="border: 1px solid red"></div>
19  </body>
20 </html>

```

5.4.3. Creando escenarios y capas.

En este ejemplo vamos a crear un escenario y le incorporaremos una capa donde se dibujará un texto. En la figura 5.7 podemos comprobar el resultado final.

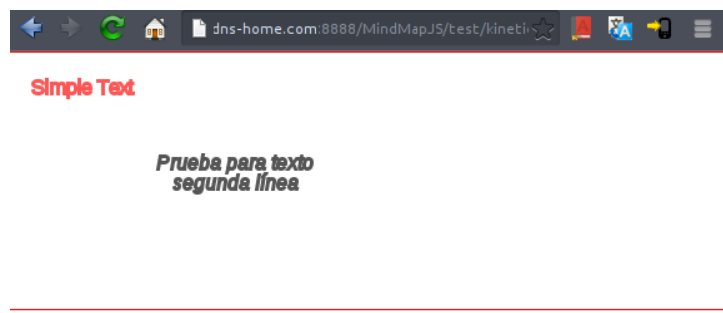


Figura 5.7: Ejemplo de Kinetic - uso básico de capas

```

1 function load() {
2   var simpleText = new Kinetic.Text({
3     x: 20,
4     y: 20,
5     text: 'Simple Text',
6     stroke: '#f55',
7     textWidth: 2,
8     fontSize: 16,
9     fontFamily: 'helvetica'
10  });
11
12  var complexText = new Kinetic.Text({
13    x: 100,
14    y: 60,
15    stroke: '#555',
16    strokeWidth: 2,
17    text: 'Prueba para texto\nsegunda línea',
18    fontSize: 16,
19    fontFamily: 'helvetica',
20    width: 'auto',
21    padding: 20,
22    align: 'center',
23    fontStyle: 'italic',
24    shadow: {
25      color: 'black',
26      blur: 10,
27      offset: [10, 10],
28      opacity: 0.4
29    }

```

```
30     cornerRadius: 10
31   });
32
33
34   var stage = new Kinetic.Stage({
35     container: 'container',
36     width: 578,
37     height: 200
38   });
39
40   var layer = new Kinetic.Layer();
41   layer.add(simpleText);
42   layer.add(complexText);
43   stage.add(layer);
44 }
```

El ejemplo muestra en la línea 34 como se crea un escenario (Kinetic.Stage) indicando el contenedor dentro nuestro árbol DOM y su dimensiones. A continuación (en la línea 40) creamos una capa (Kinetic.Layer) donde dibujaremos los elementos gráficos que deseamos pintar. Más concretamente se ha creado dos elementos Kinetic.Text y se han incorporado a la capa.

Se trata de un concepto muy utilizado en editores gráficos, por nombrar algunos, como Gimp y PhotoShop. Es exactamente el mismo concepto. En MindMapJS se ha utilizado concretamente tres capas superpuestas en el siguiente orden:

- **Capa grid:** muestra la rejilla de fondo.
- **Capa aristas:** contiene todas las aristas que interconecta las ideas.
- **Capa nodos:** contiene las ideas que representan el mapa mental.

5.4.4. Manejo de eventos.

Como se ha expresado con anterioridad tiene un buen sistema para manejo de eventos tanto de escritorio como touch. En el siguiente ejemplo 5.8 podemos ver como manejar eventos sobre objetos con KineticJS.

```
1  var mensaje = new Kinetic.Text({
2    x: 5,
3    y: 5,
4    text: 'mensaje ...',
5    fontSize: 14,
6    fontFamily: 'helvetica',
7    strokeWidth: 1,
8    textFill: '#555'
9  });
10
11  var mensaje2 = new Kinetic.Text({
12    x: 5,
13    y: 18,
14    text: 'Nodo posición x: 100, y: 60',
15    fontSize: 14,
```

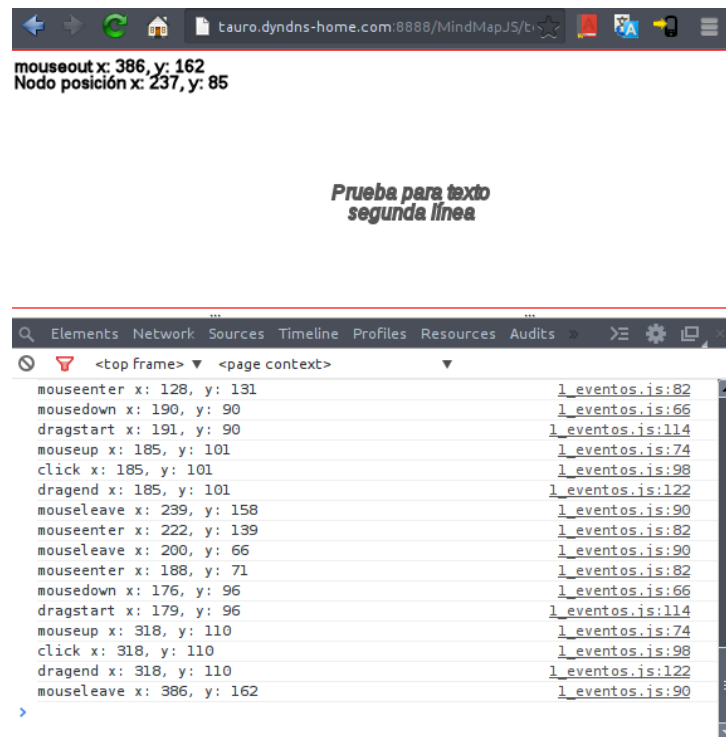


Figura 5.8: Ejemplo de Kinetic - eventos

```

16     fontFamily: 'helvetica',
17     strokeWidth : 1,
18     textFill: '#555'
19 });
20
21 var nodo = new Kinetic.Text({
22     x: 100,
23     y: 60,
24     stroke: '#555',
25     strokeWidth: 2,
26     fill: '#ddd',
27     text: 'Prueba para texto\nsegunda línea',
28     fontSize: 16,
29     fontFamily: 'helvetica',
30     textFill: '#555',
31     width: 'auto',
32     padding: 20,
33     align: 'center',
34     fontStyle: 'italic',
35     shadow: {
36         color: 'black',
37         blur: 10,
38         offset: [5, 5],
39         opacity: 0.3
40     },
41     cornerRadius: 10,
42     draggable : true
43 });
44
45 nodo.on('mouseout', function() {
46     var mousePos = stage.getMousePosition();
47     var x = mousePos.x;
48     var y = mousePos.y;
49     var mensaje = 'x: ' + x + ', y: ' + y;
50     mostrarMensaje('mouseout ' + mensaje);
51 });
52
53 nodo.on('mousemove', function() {
54     var mousePos = stage.getMousePosition();
55     var x = mousePos.x;
56     var y = mousePos.y;
57     var mensaje = 'x: ' + x + ', y: ' + y;
58     mostrarMensaje('mousemove ' + mensaje);

```

```

59 });
60
61 nodo.on('mousedown', function() {
62     var mousePos = stage.getMousePosition();
63     var x = mousePos.x;
64     var y = mousePos.y;
65     var mensaje = 'x: ' + x + ', y: ' + y;
66     console.log('mousedown ' + mensaje);
67 });
68
69 nodo.on('mouseup', function() {
70     var mousePos = stage.getMousePosition();
71     var x = mousePos.x;
72     var y = mousePos.y;
73     var mensaje = 'x: ' + x + ', y: ' + y;
74     console.log('mouseup ' + mensaje);
75 });
76
77 nodo.on('mouseenter', function() {
78     var mousePos = stage.getMousePosition();
79     var x = mousePos.x;
80     var y = mousePos.y;
81     var mensaje = 'x: ' + x + ', y: ' + y;
82     console.log('mouseenter ' + mensaje);
83 });
84
85 nodo.on('mouseleave', function() {
86     var mousePos = stage.getMousePosition();
87     var x = mousePos.x;
88     var y = mousePos.y;
89     var mensaje = 'x: ' + x + ', y: ' + y;
90     console.log('mouseleave ' + mensaje);
91 });
92
93 nodo.on('click', function() {
94     var mousePos = stage.getMousePosition();
95     var x = mousePos.x;
96     var y = mousePos.y;
97     var mensaje = 'x: ' + x + ', y: ' + y;
98     console.log('click ' + mensaje);
99 });
100
101 nodo.on('dblclick', function() {
102     var mousePos = stage.getMousePosition();
103     var x = mousePos.x;
104     var y = mousePos.y;
105     var mensaje = 'x: ' + x + ', y: ' + y;
106     console.log('dblclick ' + mensaje);
107 });
108
109 nodo.on('dragstart', function() {
110     var mousePos = stage.getMousePosition();
111     var x = mousePos.x;
112     var y = mousePos.y;
113     var mensaje = 'x: ' + x + ', y: ' + y;
114     console.log('dragstart ' + mensaje);
115 });
116
117 nodo.on('dragend', function() {
118     var mousePos = stage.getMousePosition();
119     var x = mousePos.x;
120     var y = mousePos.y;
121     var mensaje = 'x: ' + x + ', y: ' + y;
122     console.log('dragend ' + mensaje);
123     mostrarMensaje2('Nodo posición x: ' + nodo.getX() + ', y: ' + nodo.getY() );
124 });
125
126 var layer;
127 var stage;
128
129 window.onload = function load() {
130     stage = new Kinetic.Stage({
131         container: 'container',
132         width: 578,
133         height: 200
134     });
135
136     layer = new Kinetic.Layer();
137     layer.add(mensaje);
138     layer.add(mensaje2);
139     layer.add(nodo);
140     stage.add(layer);

```

```
141 }
142
143 function mostrarMensaje (texto) {
144     mensaje.setText(texto);
145     layer.draw();
146 }
147
148 function mostrarMensaje2 (texto) {
149     mensaje2.setText(texto);
150     layer.draw();
151 }
```

Podemos ver que sigue un modelo de eventos estilo JQuery en el cual nos suscribimos a un evento mediante la fórmula: ²⁰

```
1 ElementoGrafico.on('nombreEvento', function () {
2     // Manejo del evento.
3 });
```

Con este modelo, cualquier programador Javascript familiarizado con JQuery entiende como se comporta y sabe de inmediato como utilizarlo.

Existen otro muchos ejemplos que podemos ver y manipular en la página tutorial de KineticJS²¹.

5.4.5. Extendiendo KineticJS.

Existen dos formas de crear y extender KineticJS. La primera es mediante la creación de 'custom shape', se trata de un mecanismo para poder crear figuras que nosotros deseemos pintando directamente en el canvas. La segunda es implementación de una clase y extender su comportamiento dotando a nuestro nuevo elemento gráfico todas la virtudes de cualquier figura KineticJS.

La creación de 'custom shape' se basa en la implementación de una función (sceneFunc) que será utilizada a la hora de dibujar la figura. Esta función recibe un canvas sobre el que podemos dibujar directamente.

```
1 var stage = new Kinetic.Stage({
2     container: 'container',
3     width: 578,
4     height: 200
5 });
6 var layer = new Kinetic.Layer();
7
8 var triangle = new Kinetic.Shape({
9     sceneFunc: function(context) {
10         context.beginPath();
11         context.moveTo(200, 50);
12         context.lineTo(420, 80);
13         context.quadraticCurveTo(300, 100, 260, 170);
```

²⁰También implementada en MindMapJS

²¹Tutoriales en <http://www.html5canvastutorials.com/kineticjs/html5-canvas-kineticjs-shape-tutorial/>

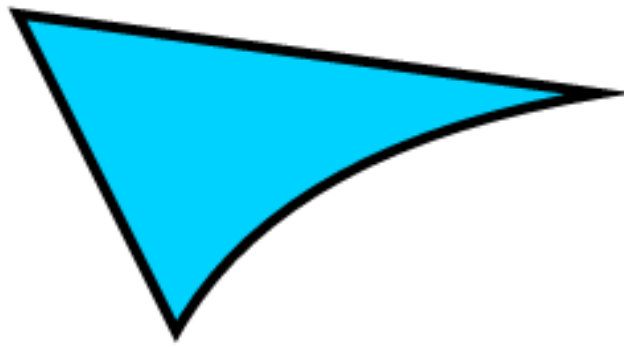


Figura 5.9: Ejemplo de Kinetic - custom shape

```

14     context.closePath();
15     context.fillStrokeShape(this);
16 },
17 fill: '#00D2FF',
18 stroke: 'black',
19 strokeWidth: 4
20 });
21
22 layer.add(triangle);
23 stage.add(layer);

```

Esta segunda forma de extensión tiene la ventaja de que nos permite crear una clase que podremos reutilizar cada vez que necesitemos. Este mecanismo ha sido el utilizado para crear las aristas de MindMapJS implementando, como podemos comprobar en el siguiente código, una curva beizer para estos propósitos. Se trata, en definitiva, de crear una nueva función constructora y definir su prototipo en el cual no puede faltar la función `init` y `drawFunc`.

```

1 (function() {
2     Kinetic.Beizer = function(config) {
3         this.___init(config);
4     };
5
6     Kinetic.Beizer.prototype = {
7         // Tiene que recibir un Objeto, puntos = { start : {x,y}, end: {x,y}, control1 :
8             {x,y}, control2 : {x,y} }
9         ___init: function(config) {
10             Kinetic.Shape.call(this, config);
11             this.className = 'Beizer';
12         },
13         drawFunc: function(canvas) {
14             console.log('beizer');
15             var context = canvas.getContext(),
16                 puntos = this.attrs.puntos;
17
18             context.beginPath();
19             context.moveTo(puntos.start.x, puntos.start.y);
20             context.bezierCurveTo ( puntos.control1.x, puntos.control1.y,
21                                     puntos.control2.x, puntos.control2.y,
22                                     puntos.end.x, puntos.end.y );
23             canvas.stroke(this);
24         }
25     };
26     Kinetic.Util.extend(Kinetic.Beizer, Kinetic.Shape);
27
28     // add getters setters

```

```
29 Kinetic.Factory.addGetterSetter(Kinetic.Beizer, 'puntos', 0);  
30 }());
```

5.5. NodeJS

Basado en la máquina virtual Javascripts V8 de Google, NodeJS²² a supuesto una revolución en el mundo de la programación Javascripts, 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 REPL²³ para pruebas y depuración interactiva hasta un gestor de paquetes y librerías NPM²⁴ (Node Packaged Modules).



Figura 5.10: 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 sola 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.

²²La web oficial de NodeJS es nodejs.org

²³Patrón Read-Eval-Print Loop

²⁴La web oficial de NPM es npmjs.org

5.5.1. 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 GitHub²⁵.

Lo primero que tenemos que 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
```

5.5.2. Instalación del NPM

Como ya se ha comentado antes NPM²⁶ 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
```

5.5.3. 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.
```

²⁵Repositorio de NodeJS <https://github.com/joyent/node>

²⁶Node 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 preguntas 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 ²⁷ para que lo puedan utilizar todas las librerías del sistema.

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

De forma local²⁸, 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

²⁷Con el modificador -g

²⁸Sin 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.

```
1 {  
2   "name": "hola",  
3   "version": "0.0.0",  
4   "description": "Hola mundo",  
5   "main": "index.js",  
6   "scripts": {  
7     "test": "echo \"Error: no test specified\" && exit 1"  
8   },  
9   "keywords": [  
10    "hola",  
11    "mundo"  
12  ],  
13   "author": "",  
14   "license": "BSD-2-Clause",  
15   "dependencies": {  
16     "colors": "~0.6.2"  
17  },  
18   "devDependencies": {  
19     "grunt-cli": "~0.1.9"  
20  }  
21 }
```

Desinstalación de paquetes

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

5.5.4. ¿Por qué utilizar NodeJS y NPM?

Sin lugar a dudas se trata de herramientas potentes que proporcionan al desarrollador un buen punto de partida para cualquier desarrollo Javascripts. Lo que me decidió por NodeJS, desde un principio, fue:

- Disponer de una **consola** que me permita probar algunas partes del desarrollo sin necesidad de un navegador. Agilizando así el desarrollo de algunas librerías.
- Tener disponible un **sistema de paquetes como NPM** que me permite integrar multitud de librerías para el desarrollo, de forma sencilla y eficaz.
- La amplia aceptación de esta herramienta y la multitud de **librerías** implementadas para la plataforma. Entre ellas, están todas la herramientas utilizadas en Mind-MapJS. Librerías para gestión de tareas²⁹, pruebas³⁰ y verificación de código³¹ entre otras muchas.

²⁹GruntJS ha sido la elección utilizada como gestor de tareas

³⁰En el caso de las pruebas unitarias opté por MochaJS

³¹JsHint

5.6. 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 Javascripts, 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.

5.6.1. 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 Javascripts.
- **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.

5.6.2. Instalación

La instalación de GruntJS no tiene complicación, ya que, al tratarse de una aplicación Node y estar publicado en NPM sólo necesitamos como prerequisite 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
```

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 incluirá 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 }
```



Figura 5.11: Logo GruntJS

5.6.3. Creando el Gruntfile

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

```
1 module.exports = function(grunt) {  
2  
3   grunt.registerTask('default', 'Tarea Hola Mundo', function() {  
4     grunt.log.write('Hola Mundo!').ok();  
5   });  
6  
7 };
```

Como se puede observar se trata de un modulo Node, que será llamado por grunt cuando lo ejecutemos. En el ejemplo, le hemos registrado 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.

5.6.4. Gruntfile.js de MindMapJS

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

```

1 module.exports = function(grunt) {
2   var config = {
3     pkg: grunt.file.readJSON('package.json'),
4
5     concat: {
6       options: {
7         separator: ';',
8       },
9       source: {
10        src: ['src/*.js'],
11        dest: 'dist/<%= pkg.name %>-v<%= pkg.version %>.js'
12      }
13    },
14
15    replace: {
16      dev: {
17        options: {
18          variables: {
19            version: '<%= pkg.version %>',
20            date: '<%= grunt.template.today("yyyy-mm-dd") %>',
21          },
22          prefix: '@@'
23        },
24
25        files: [{
26          src: ['dist/<%= pkg.name %>-v<%= pkg.version %>.js'],
27          dest: 'dist/<%= pkg.name %>-v<%= pkg.version %>.js'
28        }],
29      },
30      prod: {
31        options: {
32          variables: {
33            version: '<%= pkg.version %>',
34          },
35          prefix: '@@'
36        },
37        files: [{
38          src: ['dist/<%= pkg.name %>-v<%= pkg.version %>.min.js'],
39          dest: 'dist/<%= pkg.name %>-v<%= pkg.version %>.min.js'
40        }],
41      }
42    },
43
44    uglify: {
45      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
51            .version %>.js'
52        }
53      },
54    },
55
56    clean: {
57      build: ['dist/*']
58    },
59
60    jshint: {
61      options: {
62        laxbreak: true,
63        curly: true,
64        eqnull: true,
65        eqeqeq: true,
66        undef: true,
67        browser: true,
68        // immed: true,
69        latedef: true,
70        newcap: true,
71        noarg: true,
72        sub: true,
73        boss: true,
74        globals: {
75          console: true,
76          window: true,
77          module: true,
78          MM: true,
79          Kinetic: true,
80          require: true,
81          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 funciona :( 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 realizar 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 tarea de validación de código JSHint.
- **jsdoc:** genera la documentación del API.

5.6.5. ¿Por qué GruntJs?

En cualquier desarrollo surgen multitud de tareas repetitivas que pueden llegar a ralentizar la elaboración de cualquier aplicación.

GruntJs está empaquetado en el sistema NPM, por lo que, es se puede instalar con un sencillo comando NPM. Pero su fuerza radica en que, se programan las tareas en Javascripts de forma sencilla y clara. Permitiendo al programador la posibilidad de extenderlo mediante un sistema de plugins.

Esta siendo ampliamente utilizado por JQuery, Modernizr, Bootstrap y WordPress Build Process. Por nombrar algunos de los más destacados.

5.7. Github

Todo proyecto que se precie debe estar sustentado con sistema de control de versiones, en nuestro caso ha sido Git³². Más concretamente se trata de un sistema distribuido de control de código fuente o SCM³³ creado por Linus Torvalds, a partir, de su propia experiencia en el desarrollo de los kernels de Linux.

Github³⁴ es una plataforma online pensada para el desarrollo colaborativo de proyectos, utilizando para ello Git. Github nos permite almacenar de forma pública³⁵ nuestro código fuente, promoviendo el trabajo colaborativo entre profesionales. Así pues, otro profesional ajeno al proyecto puede solicitar cambios sugerir mejoras o reportar bugs.

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



Figura 5.12: Mascota de Github

- **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 proyecto 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

³²Web oficial de Git es git-scm.com

³³SCM (Source Code Management)

³⁴La web de Github es github.com

³⁵Github permite crear proyectos privados con cuentas de pago

concretas. Un fork nos permite colaborar con el proyecto original mediante 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.

5.7.1. 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 markdown³⁶ en el cual hay que introducir un descripción del proyecto. Este fichero se visualizará en la 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
```

³⁶<http://es.wikipedia.org/wiki/Markdown>

5.7.2. 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/usuarioOriginal/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 rama³⁷, realizar los cambios y subirlos³⁸ a nuestro fork de Github. Desde Github procede realizar la revisión de los cambios y pulsar sobre la opción de 'create a pull request for this comparison'.

5.8. JSDoc

Tan importante como el código es la documentación del mismo, JSDoc³⁹ es una herramienta inspirada en Javadoc⁴⁰ pero pensada para Javascripts.

Mediante una conjunto de etiquetas (`@class`, `@function`, etc) introducidas como comentarios del código fuente, se generará la documentación en formato HTML⁴¹. 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 5.13 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 5.14 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

³⁷Operaciones a realizar: branch y checkout

³⁸Operaciones a realizar: commit y push

³⁹Url del proyecto `https://github.com/jsdoc3/jsdoc`

⁴⁰`http://es.wikipedia.org/wiki/Javadoc`

⁴¹Por 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 5.13: Ejemplo de código fuente documentado con JSDoc

funcional, de la documentación de nuestro código fuente.

5.9. Mocha

Mocha⁴² es un framework Javascripts 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"⁴³.

⁴²Página oficial de Mocha: <http://visionmedia.github.io/mocha/>

⁴³En la cabecera de su web podemos leer "mocha simple, flexible, fun"

Class: PubSub**MM. PubSub***Implementación del patrón Publish/Subscribe***new PubSub()**Source: [pubsub.js](#), line 12**Methods****desSuscribir(id) → {null|number}**

realiza una dessuscripción a un evento o notificación

Parameters:

Name	Type	Description
id	number	identificador de suscripción

Source: [pubsub.js](#), line 77**Returns:**

null si no se ha podido realizar la dessuscripción

Type

null | number

on(evento, args) → {boolean}**Index****Classes**

[Arbol](#)
[Arista](#)
[Borde](#)
[Class](#)
[Globo](#)
[Grid](#)
[FreeMind](#)
[XML](#)
[Mensaje](#)
[NodoSimple](#)
[PubSub](#)
[Rama](#)
[Render](#)
[UndoManager](#)
[ComandoHacerDeshacer](#)

Namespaces
[MM](#)
[DOM](#)
[exportar](#)
[importar](#)
[Properties](#)
[teclado](#)

Figura 5.14: Página generada por JSDoc

5.9.1. 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 interfaz 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.

5.9.2. 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 });

```



Para empezar, debe realizar importar la librería de afirmaciones (assertions) que vamos a utilizar en el presente ejemplo se ha utilizado `assert`⁴⁴.

Figura 5.15: Mocha

Las líneas 2 y 3 describen a su vez, un módulo y submódulo de ejecución. En nuestro caso el módulo lo hemos llamado `.Array` el submódulo de ejecución `indexOf()`. 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 5.16.

```

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

1 test complete (3 ms)

```

Figura 5.16: 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 });

```

⁴⁴En MindMapJS se ha utilizado Should y Chai


```
8
9
10 describe('Array', function(){
11   describe('#indexOf()', function(){
12     it('debe retornar -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 retornar 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.

A terminal window with a dark background and light green text. It shows the output of a Mocha test run. The output is organized into a tree structure: 'Array' is the root, followed by '#indexOf()', which contains two sub-items: '✓ debe retornar -1 si el valor no esta presente' and '✓ debe retornar 1 si pedimos al primer valor'. At the bottom, it says '2 tests complete (4 ms)'.

Figura 5.17: 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.

Manual de usuario

6.1. Manual para el desarrollador

El usuario final de MindMapJS puede tanto un programador, como un usuario final. Para los primeros se ha elaborado el siguiente documento, buscando facilitar su integración en otros proyectos.

6.1.1. Dependencias

Para poder poner en marcha el proyecto debes tener instalado en tu sistema:

- **Git**
- **NodeJS**
- **NPM** en el caso de que no venga integrado con NodeJS. En las últimas versiones NPM ya viene integrado en el instalador de NodeJS.

6.1.2. Clonar el MindMapJS

primero que debemos realizar para poder obtener el código fuente es clonar el proyecto. Para ello, supongo que se tiene ya instalado el Git. Para clonar el proyecto MindMapJS

ejecutamos el siguiente comando.

```
1 $ git clone https://github.com/joseluismolinasoria/MindMapJS.git
```

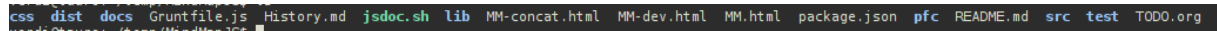


Figura 6.1: Estructura de directorios de MindMapJS

En la estructura del proyecto 6.1 podemos observar como disponemos de los siguientes directorios:

- Directorio **css** con los ficheros de estilo de la página/s HTML.
- Directorio **dist** dónde se generará las versiones de distribución de la librería Javascripts de MindMapJS.
- Directorio **docs** con la documentación, en formato HTML, del API de MindMapJS.
- Directorio **lib** con librerías externas al proyecto como KineticJS.
- Directorio **pfc** este documento.
- Directorio **src** con todo el código fuente Javascripts del proyecto.
- Directorio **test** con los fuentes de los test unitarios de MindMapJS.
- Y en general ficheros de configuración y páginas HTML de demos, como un fichero TODO en formato Org-mode.

6.1.3. Resolver dependencias

El siguiente paso es tener en cuenta es la resolución de dependencias. Para ello, haremos uso de NPM. Para este paso debes tener NodeJS y NPM. El siguiente comando nos descargará las librerías y herramientas necesarias para el desarrollo.

```
1 $ npm install
```

Ahora ya tenemos el proyecto operativo para realizar nuestras mejoras.

```
Running "clean:build" (clean) task
Cleaning "dist/MindMapJS-v0.1.2.js"...OK
Cleaning "dist/MindMapJS-v0.1.2.min.js"...OK

Running "concat:source" (concat) task
File "dist/MindMapJS-v0.1.2.js" created.

Running "replace:dev" (replace) task
Replace dist/MindMapJS-v0.1.2.js -> dist/MindMapJS-v0.1.2.js

Done, without errors.
```

Figura 6.2: Resultado de ejecutar el comando 'grunt dev'

6.1.4. Construir la versión de desarrollo

Para obtener una versión de desarrollo de la librería utilizaremos el siguiente comando.

```
1 $ grunt dev
```

Con el obtendremos una concatenación de todos los módulos y clases de MindMapJS. El fichero final es **dist/MindMapJS-vXXX.js** donde XXX es la versión actual del proyecto.

6.1.5. Construir la versión de producción

Para obtener una versión de producción de MindMapJS utilizaremos el siguiente comando.

```
1 $ grunt full
```

```
Running "clean:build" (clean) task
Cleaning "dist/MindMapJS-v0.1.2.js"...OK
Cleaning "dist/MindMapJS-v0.1.2.min.js"...OK

Running "concat:source" (concat) task
File "dist/MindMapJS-v0.1.2.js" created.

Running "replace:dev" (replace) task
Replace dist/MindMapJS-v0.1.2.js -> dist/MindMapJS-v0.1.2.js

Running "uglify:build" (uglify) task
File "dist/MindMapJS-v0.1.2.min.js" created.

Running "replace:prod" (replace) task

Done, without errors.
```

Figura 6.3: Resultado de ejecutar el comando 'grunt full'

Con el obtendremos una concatenación de todos los módulos y clases de MindMapJS. El fichero final es **dist/MindMapJS-vXXX.min.js** donde XXX es la versión actual del proyecto.

6.1.6. JSHint. Verificar el código.

Para comprobar la validez del código fuente con JSHint, realizaremos:

```
1 $ grunt hint
```

```
Running "jshint:all" (jshint) task
>> 24 files lint free.

Done, without errors.
```

Figura 6.4: Resultado de ejecutar el comando 'grunt hint'

6.1.7. Tests.

El siguiente comando lanza la batería de pruebas del proyecto.

```
1 $ grunt test
```

```
Con dos manejadores de eventos
✓ Creamos dos manejadores eventos
✓ Los contadores deben estar inicializados
✓ Creamos nos suscribimos a los eventos del manejador 1
✓ Creamos nos suscribimos a los eventos del manejador 2
✓ Lanzamos el evento del manejador1

UndoManager
Si el undoManager esta vacío
✓ Debe tener indice actual en -1
✓ Los comandos hacer y deshacer deben ser nulos
Si creamos un Comando hacer deshacer
✓ Debe tener el nombre
✓ hacemos
✓ deshacemos
Si añadimos un comando al undomanager
✓ Debe añadirse el comando y el indice actual a 0
✓ Deshacer es "crear" y hacer es "null"
✓ Realiza la acción hacer correctamente
✓ El comando a deshacer es "crear" y el hacer "null"
✓ Realiza la acción de deshacer correctamente
✓ El comando a deshacer es "null" y el hacer "crear"
✓ Realiza la acción hacer correctamente
✓ El comando a deshacer es "crear" y el hacer "null"
Con dos comandos
✓ Debe añadir el comando y el índice actual a 1
✓ El comando a deshacer es "borrar" y el hacer "null"
✓ Realiza la acción hacer correctamente
✓ El comando a deshacer es "borrar" y el hacer "null"
✓ Realiza la acción deshacer correctamente
✓ El comando a deshacer es "crear" y el hacer "borrar"
✓ Realizamos el deshacer el comando introducido en primer lugar
✓ El comando a deshacer es "null" y el hacer "crear"
✓ Realizamos el hacer del primer comando
✓ El comando a deshacer es "crear" y el hacer "borrar"
✓ Realizamos el hacer del segundo comando
✓ El comando a deshacer es "borrar" y el hacer "null"
Creado muchos comandos
✓ Añadimos muchos comandos más del máximo establecido
✓ Deshacemos un par de comandos
✓ Creamos uno en medio
✓ Deshacemos todo

117 tests complete (41 ms)

Done, without errors.
```

Figura 6.5: Resultado de ejecutar el comando 'grunt test'

6.1.8. Generar la documentación del API

Si es necesario podemos generar API en formato JSDoc.

```
1 $ ./jsdocs.sh
```

6.2. Manual de uso de MindMapJS

MindMapJS es un editor de mapas mentales online que le permite crear mapas mentales de forma fácil y rápida. Se trata de una página web con la cual no necesitas de instalaciones, ni complicadas configuraciones. Siempre disponible.

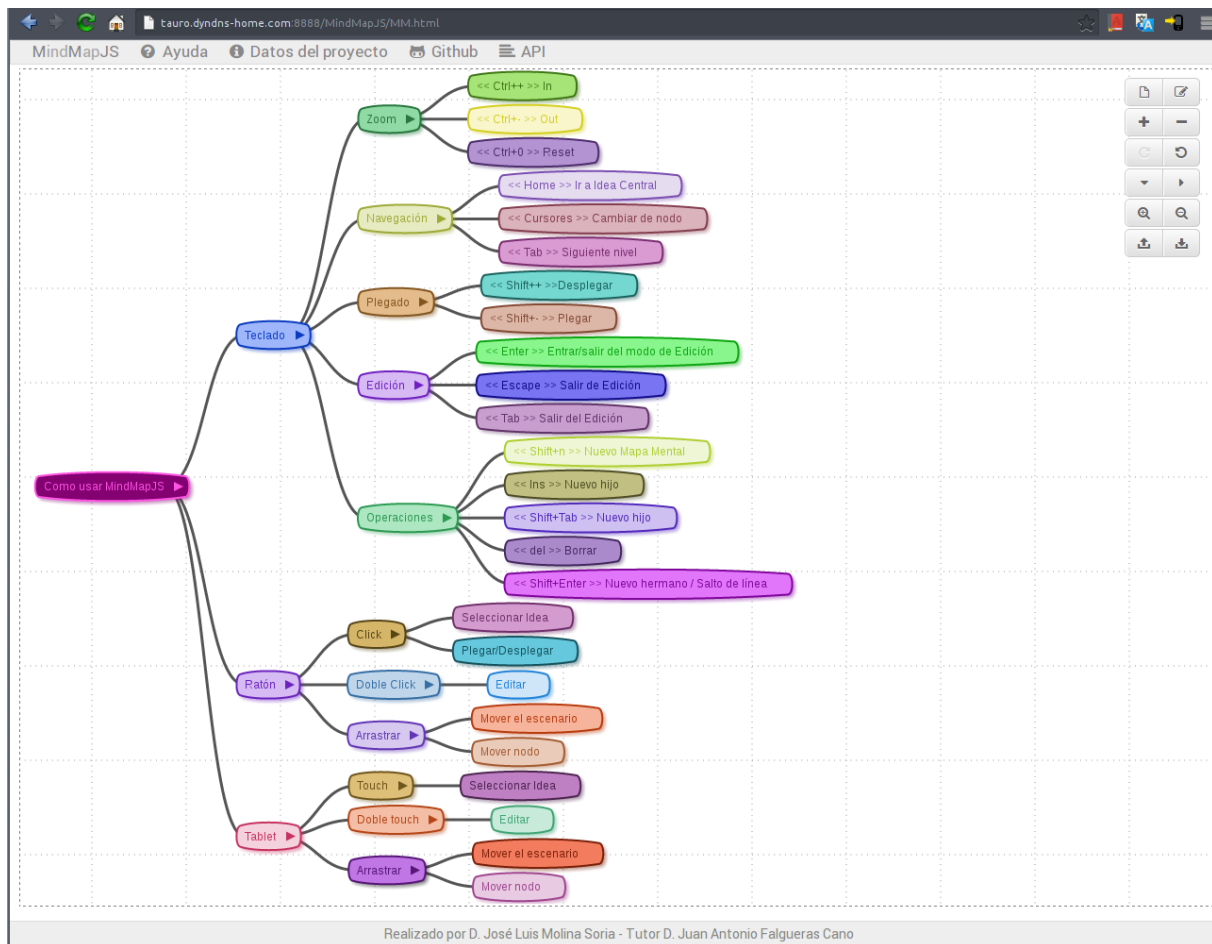


Figura 6.6: Página principal de MindMapJS

6.2.1. Estructura de la página principal.

En la página de inicio de MindMapJS (ver figura 6.6) podemos diferenciar claramente tres partes.

- **Cabecera o menú superior.** En cual disponemos de algunos enlaces para despegar información sobre el proyecto (ver figura 6.7).
- **Pie.**

- **Cuerpo.** Contenedor para el editor de mapas mentales.
- **Barra de herramientas.** Con la funcionalidad básica para manejar interactuar con el Mapa mental.



Figura 6.7: Menú ¿ Datos del proyecto

Al cargar la página (se puede ver en la figura 6.6) nos muestra un mapa mental con las formas de uso habituales. Mediante teclado, ratón o touch.

6.2.2. Barra de herramientas.

En la barra de herramientas disponemos de las siguientes opciones (imagen 6.8), de izquierda a derecha y de arriba a bajo son:

- Nuevo: crea un nuevo mapa mental
- Editar: establece el modo de edición para la idea activa.
- Crear: crea una nueva idea.
- Borrar: borra la idea activa.
- Hacer: para volver a rehacer acciones deshechas.
- Deshacer: deshace la última acción realizada.
- Plegar: pliega la subideas de la idea activa.
- Desplegar: despliega una idea plegada.
- Zoom in: amplia la escala del mapa mental.

- Zoom out: reduce la escala del mapa mental.
- Cargar: realiza la carga de un fichero freeMind.
- Salvar: salva el mapa mental actual en un fichero freeMind.



Figura 6.8: Barra de herramientas

6.2.3. ¿Cómo crear un nuevo Mapa Mental?

Para iniciar o crear un nuevo mapa mental podemos optar por hacer clic en el botón de la barra de herramientas, o bien pulsar la secuencia de teclas Shift+n. Una vez pulsado el botón para crear un nuevo mapa mental se borrará el mapa actual y preparará el editor para un nuevo mapa mental a partir de una nueva idea central (ver figura 6.9).

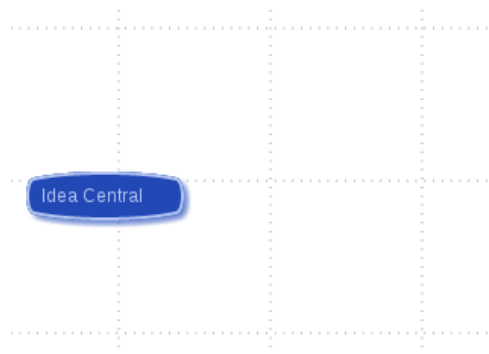


Figura 6.9: Crear un nuevo mapa

6.2.4. Insertar nuevas subideas.

Siempre podemos insertar una subidea a partir de otra ideal principal (ver figura 6.10) con el botón de la barra de herramientas, o pulsando la tecla ins. Siempre que generemos una nueva idea el editor pasara al modo de edición para que se pueda cambiar el contenido (ver figura 6.12).

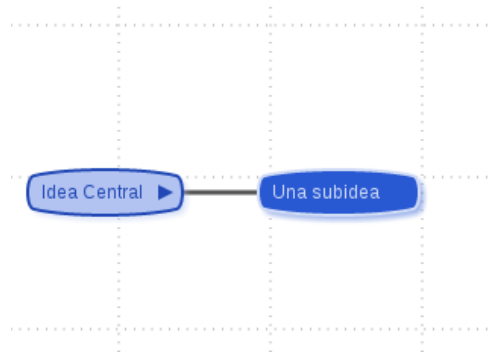


Figura 6.10: Inserción de subideas.

Si pulsamos Shift+Enter generaremos un idea hermana a la actual (ver figura 6.11), es decir, que depende de la misma idea principal.

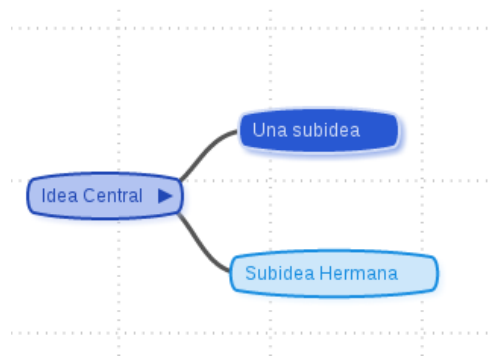


Figura 6.11: Inserción de subideas hermana.

6.2.5. Editar idea.

Para entrar en modo de edición de la idea actual podemos pulsar Enter, doble clic o pulsar el botón correspondiente en la barra de herramientas. Siempre podremos salir pulsando la tecla Escape, pulsando en otro punto de la pantalla con el ratón, o bien, volviendo a pulsar el botón de edición.

6.2.6. Borrar idea.

Una vez seleccionada la idea que deseamos borrar pulsaremos el botón de borrar o la tecla Supr. El proceso de borrado elimina todas las subideas de la idea borrada.

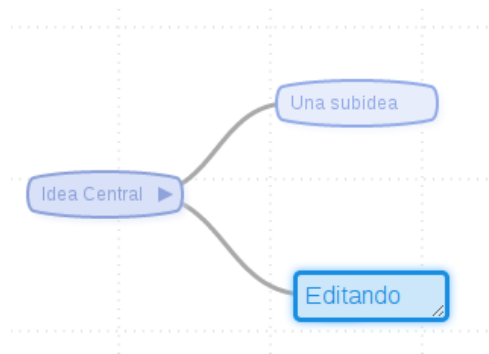


Figura 6.12: Editando una idea.

6.2.7. Navegar por el mapa.

Como podrá observar siempre existe una idea seleccionada (tiene el cursor). Para poder movernos por las distintas ideas siempre podremos seleccionar una idea con el clic del ratón o tocando con el dedo en un dispositivo táctil.

También disponemos de los cursores del teclado para desplazarnos por el mapa mental. La tecla home para ir a la idea principal y el tabulador para desplazarnos entre los distintos niveles.

6.2.8. Plegar/desplegar.

Por defecto, el editor MindMapJS, crear o muestra las ideas de forma desplegada (como se puede ver en la siguiente figura 6.13). Si una idea tiene asociada una o más subideas, esta idea principal mostrará un pequeño triangulo como indicador des/plegado. Si el triangulo apunta hacia los hijos indica que la idea esta desplegada. Si el triangulo a punta hacia abajo indica que la idea esta plegada (ver imagen 6.14).

Para plegar o desplegar una idea, tan sólo debemos realizar un clic sobre el indicador de plegado (triangulo) o mediante el uso de la barra de herramienta. También se dispones de opciones de teclados Shift+- para plegar y Shift++ para desplegar.

6.2.9. Zoom. Ampliar y reducir la imagen.

Para ampliar o reducir el mapa mental como siempre disponemos de distintas opciones.

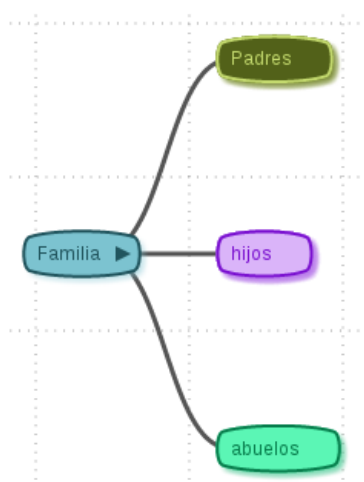


Figura 6.13: Mapa mental desplegado.



Figura 6.14: Mapa mental plegado.

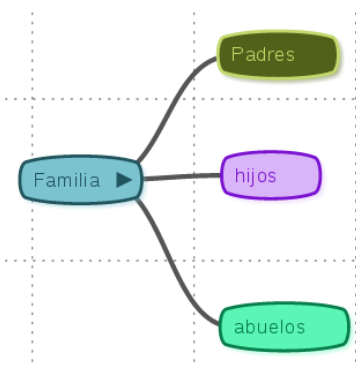


Figura 6.15: Mapa mental con ampliado.

Se puede utilizar los botones de la barra de herramientas (icono de la lupa). La secuencias de teclas: Ctrl++ para ampliar (imagen 6.16); Ctrl+- para reducir (imagen 6.15) y Ctrl+0 para reiniciar la escala de la imagen.

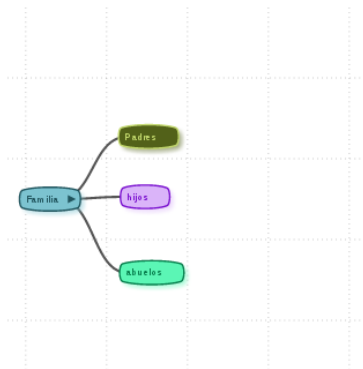


Figura 6.16: Mapa mental con reducido.

También dispones de la rueda de ratón para cambiar el tamaño o escala de nuestro mapa mental.

6.2.10. Listado de secuencias de teclado. Keystrokes.

Secuencia de teclado	Acción
Ctrl++	Zoom in
Ctrl+-	Zoom out
Ctrl+0	Resetear el zoom
home	Ir a idea central
cursores	Para moverse por las ideas del mapa mental
Tab	Para moverse al siguiente nivel del mapa
Tab	Crear nueva idea si no hay siguiente nivel.
Tab	Desplegar un nodo plegado.
Enter	Pone la idea actual en modo edición.
Escape	Sale del modo de edición.
Shift++	Desplegar un nodo plegado.
Shift+-	Plegar un nodo plegado.
Shift+n	Nuevo mapa mental.
ins	Nueva idea hija.
Shift+Tab	Nueva idea hija.
del	Borra la idea actual.
Shift+Enter	Crea una nueva idea hermana.
Shift+Enter	Inserta un salto de línea en modo de edición.

Resultados. Conclusiones

7.1. Resultados.

Cómo resultado del desarrollo de MindMapJS se ha obtenido una aplicación cross browser, capaz de funcionar completamente en los principales navegadores del mercado¹. Se ha verificado su funcionamiento en sistemas Linux, Windows, Mac Os, iOS y Android. Esto ha sido posible gracias a que se ha seguido los estándares de la W3C² y Emacs³. Ampliamente soportados en casi todos los navegadores.

Se ha trabajado en muchos casos con borradores y propuestas de especificaciones que en algunos casos no han sido implementadas por los navegadores, o han sido parcialmente implementadas. Un claro ejemplo de este tipo de problemas lo podemos ver en la especificación de File API. Según la especificación es posible escribir un fichero⁴ en el cliente, pero no siempre ha sido posible. Por ejemplo, en todos los navegadores, en los que se ha probado, se cargan de forma satisfactoria los ficheros, pero la escritura a dado problemas y salvo Internet Explorer y Safari, el resto de los navegadores me han permitido la descarga del fichero. En el caso de Internet Explorer directamente no es soportado, pero

¹Internet Explorer 10 y 11, Google Chrome, FireFox, Opera y Safari

²Sobre HTML5

³Sobre Javascripts más concretamente la especificación EmacsScript 5.1

⁴Con estrictas directivas de seguridad

en el caso de Safari se ha publicado declaraciones indicando que no se ha implementado ni se implementará por problemas de seguridad.

Para solventar el problema de la carga y descarga de fichero se puede optar por utilizar servicios de terceros⁵.

Se aplicado un diseño basado en patrones que ha permitido que el editor de mapas mentales sea fácilmente extensible, por cualquier desarrollador con conocimientos en JavaScript. Siempre se puede extender las clases de MM.Nodo o MM.Artistas y utilizarlas.

Un usuario puede incorporar un mapa mental en poco más de 20 líneas de código⁶. Y con un poco más de esfuerzo incorporar los distintos eventos a botones o nuevas secuencias de teclas. O bien incorporar un div contenedor donde desea el Mapa mental y las librerías JavaScript. Este aspecto ha sido buscado conscientemente para facilitar usuario poder incorporarlo.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>::MindMapJS::</title>
5     <meta charset="UTF-8">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
7     <link rel="stylesheet" type="text/css" href="css/MM.css" media="all" />
8     <script src="lib/kinetic-v4.6.0.min.js"></script>
9     <script src="dist/MindMapJS-v0.1.2.min.js"></script>
10    <script type="text/javascript">
11      window.onload = function () {
12        MM.nuevo('Mapa Mental');
13        MM.renderizar('contenedorEditor');
14      };
15    </script>
16  </head>
17  <body>
18    <div id="contenedorEditor"></div>
19  </body>
20 </html>
```

Como informático, le doy mucha importancia al hecho de no tener que utilizar el ratón, por ello, he dado mucha importancia a la usabilidad del teclado y su configuración.

Entre los problemas que presenta la aplicación está la falta de flexibilidad en uso de sistemas táctiles. Necesita mejorar la experiencia de usuario en este tipo de dispositivos. Además se ha detectado en algunos sistemas Android de baja gama que no presenta un buen rendimiento debido a que KineticJS redibuja en función de los FPS del dispositivo.

⁵P.e. DropBox, Drive, Copy, etc

⁶La demo apenas supera las 150 líneas de código

7.2. Conclusiones.

HTML5 y Javascripts proporcionan un conjunto de herramientas que mejoran con creces a la versión anterior. Un conjunto de API que permiten dar rienda suelta a la imaginación del programador y que tiene grandes expectativas de futuro. Aun así, los navegadores y las políticas de empresa ponen cortapisas a dichas mejoras. Un ejemplo claro es la implementación del File API, dónde no sólo hay diferencias de implementación entre los navegadores sino, que existen navegadores que no pretenden implementarlo⁷.

MindMapJS es sin duda mejorable. Entre algunas mejoras estaría la posibilidad de:

- Creación de enlaces. Incluir tanto enlaces a otros nodos como a recursos externos.
- Inserción de imágenes. Añadir imágenes en las ideas sería una característica deseable que permitirá al usuario enriquecer sus ideas.
- Notas sobre ideas. Incorporar notas sobre las ideas. Un texto enriquecido probablemente en un formato html o markdown.
- Mapas múltiples. Dar la posibilidad al usuario de crear varios mapas mentales en el mismo editor.
- Cargar y guardar mapas en la nube. Otra característica que podría mejorar sensiblemente el uso, por parte del usuario final, es el hecho de incorporar la posibilidad de cargar/guardar sus mapas mentales en servicios de ficheros como Drive, Dropbox, Copy, etc...
- Edición cooperativa. Implementación de un servicio, que permita la modificación simultanea de varios usuario sobre un mapa mental. Esta característica no ha sido implementada por ningún editor de mapas mentales hasta el momento y podría ser utilizado como herramienta de brainstorming.
- Mejora en dispositivos táctiles. El uso de MindMapJS es, hasta el momento, tosco. Necesita de mejorar en este aspecto.

⁷Es el caso de Safari.

A pesar de los problemas y dificultades superadas, no puedo por menos, que considerar la experiencia positiva y divertida. Un reto que me ha llevado a innovar, adaptarme y probar multitud de tecnologías. Que me ha llevado por intrincados y tortuosos caminos. Y sobre todo que me ha permitido ampliar mis conocimientos sobre la materia.



Bibliografía

- [1] David Flanagan. JavaScript: The Definitive Guide (Definitive Guides). O'Reilly. 6^a Edición. 2011.
- [2] Douglas Crockford. JavaScript: The Good Parts: Working with the Shallow Grain of JavaScript. O'Reilly, Yahoo press. 2008.
- [3] Stoyan Stefanov. JavaScript Patterns. O'Reilly, Yahoo press. 2010.
- [4] Addy Osmani. Learning JavaScript Design Patterns. O'Reilly. 2012.
- [5] Nicholas C. Zakas. High Performance JavaScript (Build Faster Web Application Interfaces). O'Reilly, Yahoo press. 2010.
- [6] Nicholas C. Zakas. Maintainable JavaScript. O'Reilly. 2012.
- [7] Alex MacCaw. JavaScript Web Applications. O'Reilly. 2011.
- [8] EMAC. Standard ECMA-262. ECMAScript® Language Specification. Edición 5.1. 2011. <http://www.ecma-international.org/ecma-262/5.1/>. PDF <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>
- [9] W3C. Especificación de HTML5. <http://www.w3.org/TR/html5/>

- [10] W3C. Especificación de File API. <http://www.w3.org/TR/FileAPI/>
- [11] W3C. Especificación de Canvas. <http://www.w3.org/TR/2dcontext/>
- [12] KineticJS. <http://kineticjs.com/>
- [13] GruntJS. <http://gruntjs.com/>
- [14] NPM. <https://www.npmjs.org/>
- [15] NodeJS. <http://nodejs.org/>
- [16] Mocha. <http://visionmedia.github.io/mocha/>
- [17] JsDoc. <http://usejsdoc.org/>