

**UNIVERSIDAD DE MÁLAGA**

**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**

**INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN**

**Editor de mapas mentales online con HTML5 y Javascript**

**Realizado por**

**José Luis Molina Soria**

**Dirigido por**

**Juan Antonio Falgueras Cano**

**Departamento**

**Lenguajes y Ciencias de la Computación**

**Málaga, Noviembre de 2013**

**UNIVERSIDAD DE MÁLAGA**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**  
**INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN**

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

Presidente/a D<sup>o</sup>/D<sup>a</sup>. \_\_\_\_\_

Secretario/a D<sup>o</sup>/D<sup>a</sup>. \_\_\_\_\_

Vocal D<sup>o</sup>/D<sup>a</sup>. \_\_\_\_\_

para juzgar el proyecto Fin de Carrera titulado: \_\_\_\_\_

\_\_\_\_\_

del alumno/a D<sup>o</sup>/D<sup>a</sup> : \_\_\_\_\_

dirigido por D<sup>o</sup>/D<sup>a</sup> : \_\_\_\_\_ ,

y, en su caso, dirigido académicamente por D<sup>o</sup>/D<sup>a</sup> : \_\_\_\_\_

\_\_\_\_\_

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

<b>1</b>	<b>Abstract</b>	<b>8</b>
<b>2</b>	<b>Motivación y objetivos</b>	<b>10</b>
2.1	Motivación . . . . .	10
2.2	Objetivos . . . . .	12
<b>3</b>	<b>Introducción</b>	<b>14</b>
3.1	¿Qué es? . . . . .	14
3.2	Aplicaciones y beneficios. . . . .	16
3.3	Partes de un mapa mental. . . . .	17
3.4	Elaboración. . . . .	19
<b>4</b>	<b>Materiales y métodos</b>	<b>21</b>
4.1	Metodología y etapas del desarrollo. . . . .	21
4.2	Casos de uso. . . . .	24
4.3	Diagramas de Clase . . . . .	30
<b>5</b>	<b>Implementación.</b>	<b>54</b>
5.1	Javascript. . . . .	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
<b>6</b>	<b>Resultados y discusión</b>	<b>105</b>
6.1	Resultados . . . . .	105
6.2	Discusión . . . . .	105
<b>7</b>	<b>Manual de usuario</b>	<b>107</b>
7.1	Manual para el desarrollador . . . . .	107
7.2	Manual de uso de MindMapJS . . . . .	110
<b>8</b>	<b>Conclusiones</b>	<b>112</b>
	<b>Bibliografía</b>	<b>114</b>

# 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 . . . . .	30
4.10	Clase MM.Class . . . . .	31
4.11	Diagrama de clases pubsub . . . . .	32
4.12	Clase MM.PubSub . . . . .	32
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 . . . . .	46
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 . . . . .	52
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
7.1	Estructura de directorios de MindMapJS . . . . .	108
7.2	Resultado de ejecutar el comando 'grunt dev' . . . . .	109
7.3	Resultado de ejecutar el comando 'grunt full' . . . . .	109
7.4	Resultado de ejecutar el comando 'grunt hint' . . . . .	110
7.5	Resultado de ejecutar el comando 'grunt test' . . . . .	110





---

# 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 BBD<sup>1</sup>, 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.

---

<sup>1</sup>Como 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. HTML<sup>1</sup> y JavaScript son una parte importante de esta revolución, y es por ello, que decidí dar el paso y crear una aplicación que funcionará única y exclusivamente en el navegador (sin necesidad de un servidor).

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

Hace ya tiempo que vio la luz los primeros borradores de HTML5<sup>4</sup> (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.

---

<sup>1</sup>Hypertext Markup Language

<sup>2</sup><http://www.text2mindmap.com/>

<sup>3</sup><http://www.mindmeister.com/es>

<sup>4</sup>El primer borrador de HTML5 fue publicado en 2008

HTML5 ha tomado en cuenta los defectos de su versión anterior<sup>5</sup> y mejorar otras características como:

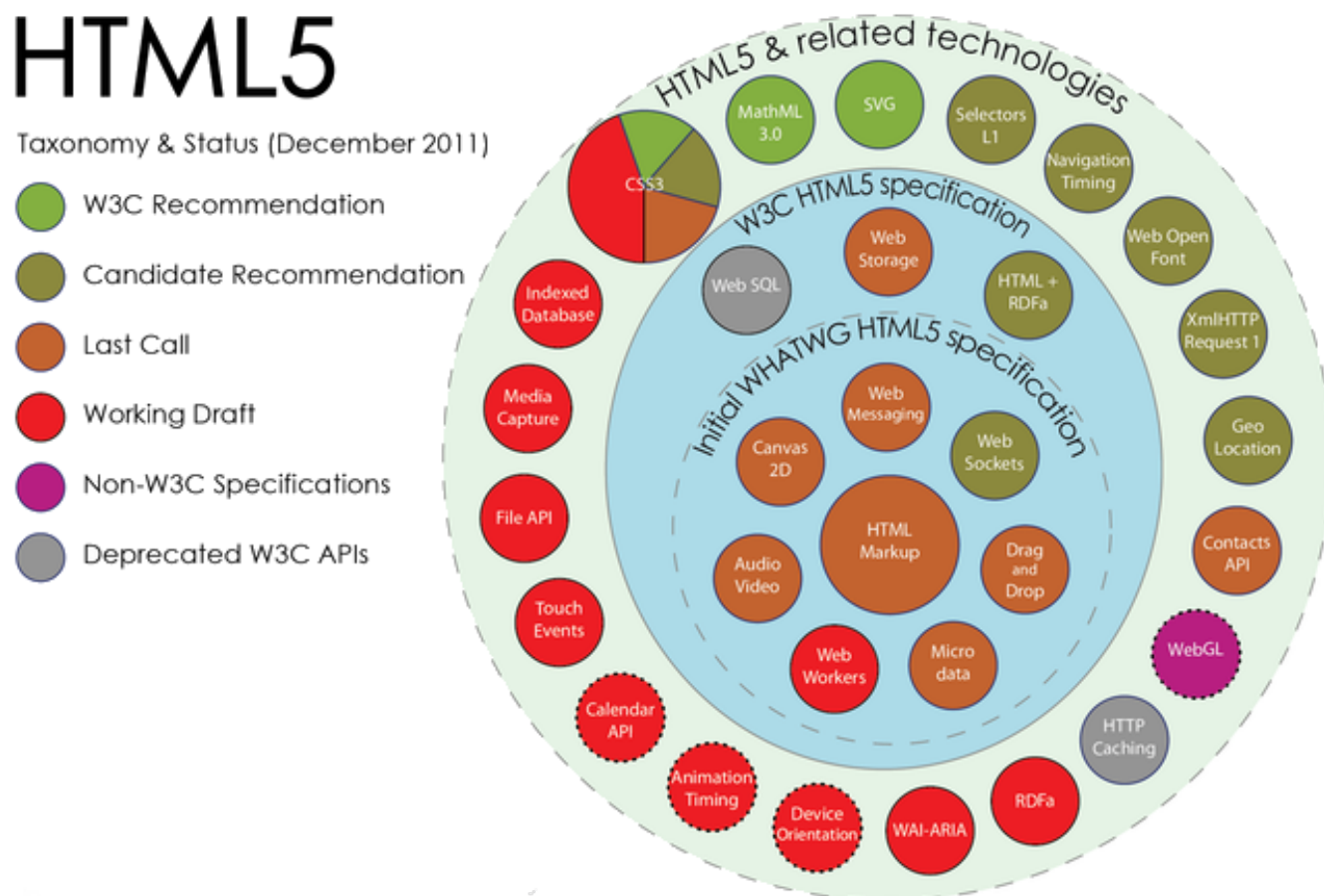


Figura 2.1: Esquema general de la especificación HTML5 a diciembre de 2011

- **Nuevas etiquetas estructurales.** Se ha incorporado un nuevo conjunto de etiquetas pensadas para definir mejor la estructura de una web, entre las más importantes están las de encabezado, barra de navegación, secciones y pie.
- **Manejo de imágenes.** En la versión anterior podía incorporar imágenes ahora, además podemos modificarlas e interactuar con ellas. También disponemos un una etiqueta y un API completo para manejo de canvas<sup>6</sup>.
- **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

<sup>5</sup>HTML 4.01

<sup>6</sup>En principio, sólo en 2D.

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

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

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

## 2.2. Objetivos

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

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

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

En todo momento, y en pos de una aplicación lo más estándar posible, se seguirá las especificaciones de la World Wide Web Consortium<sup>7</sup> (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.

---

<sup>7</sup>Web 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 irradiante<sup>1</sup>. 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.

---

<sup>1</sup>que irradia



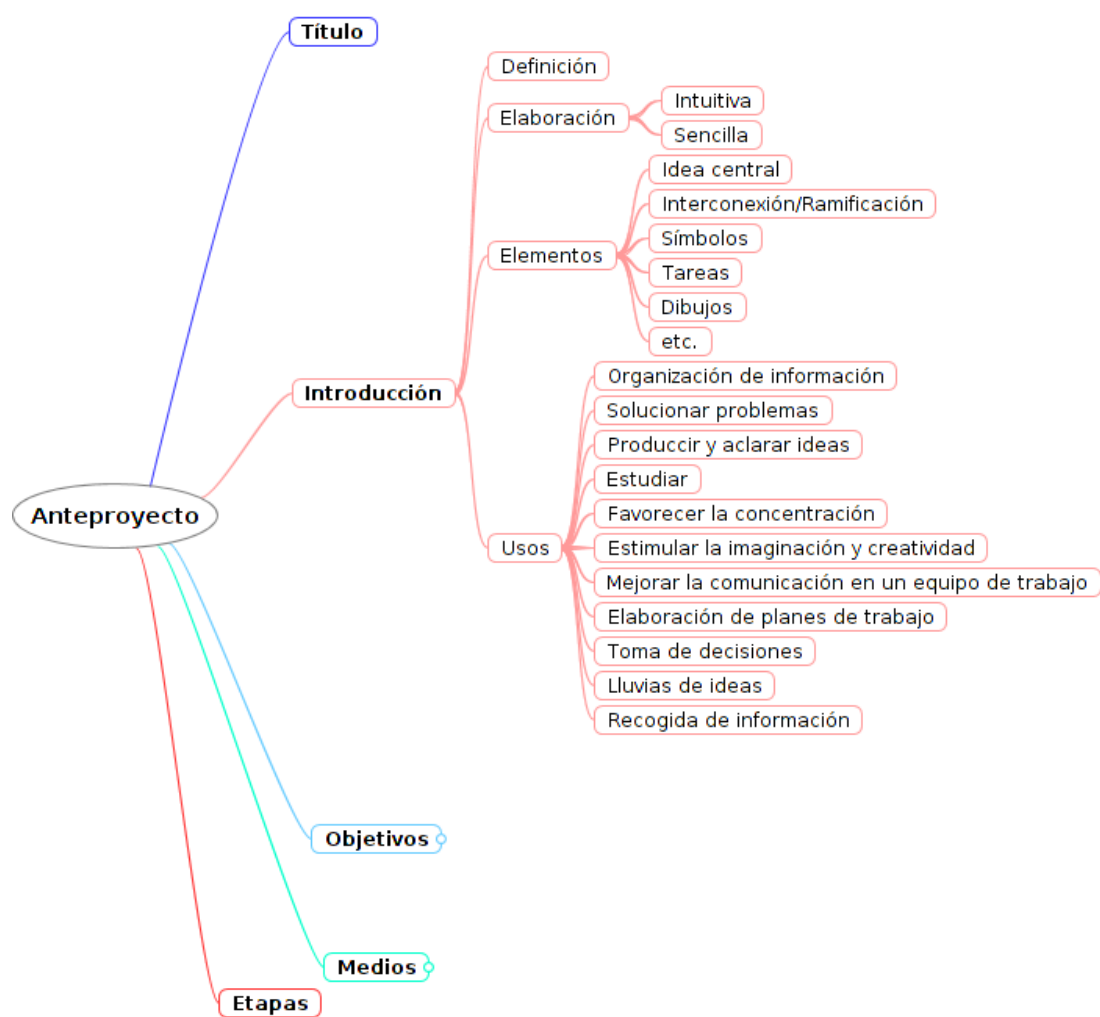


Figura 3.1: Mapa mental de FreeMind

### 3.2. Aplicaciones y beneficios.

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

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

### 3.3. Partes de un mapa mental.

#### 3.3.1. Según su estructura.

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

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

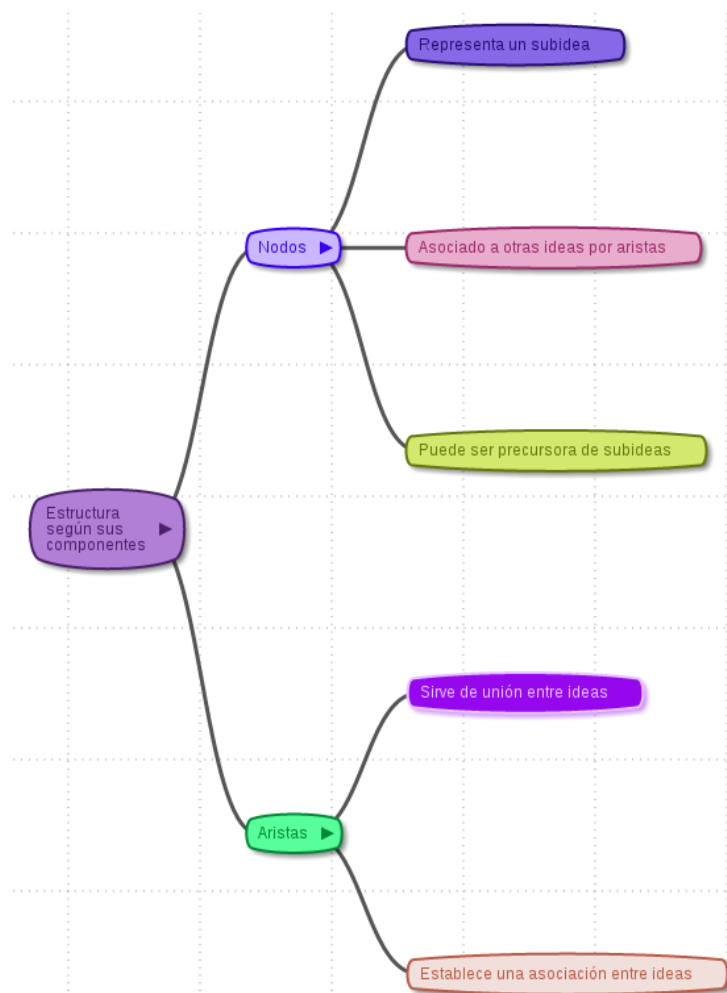


Figura 3.2: Partes de un mapa mental

#### 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.

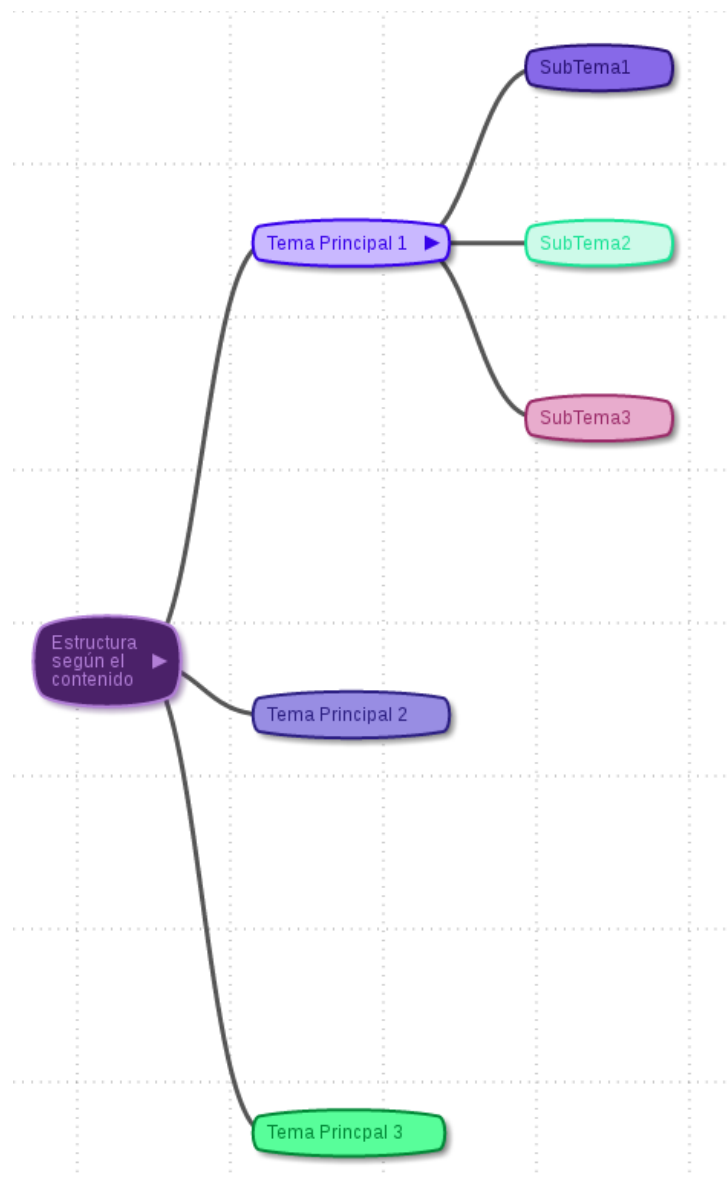


Figura 3.3: Partes de un mapa mental considerando su contenido

### 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.

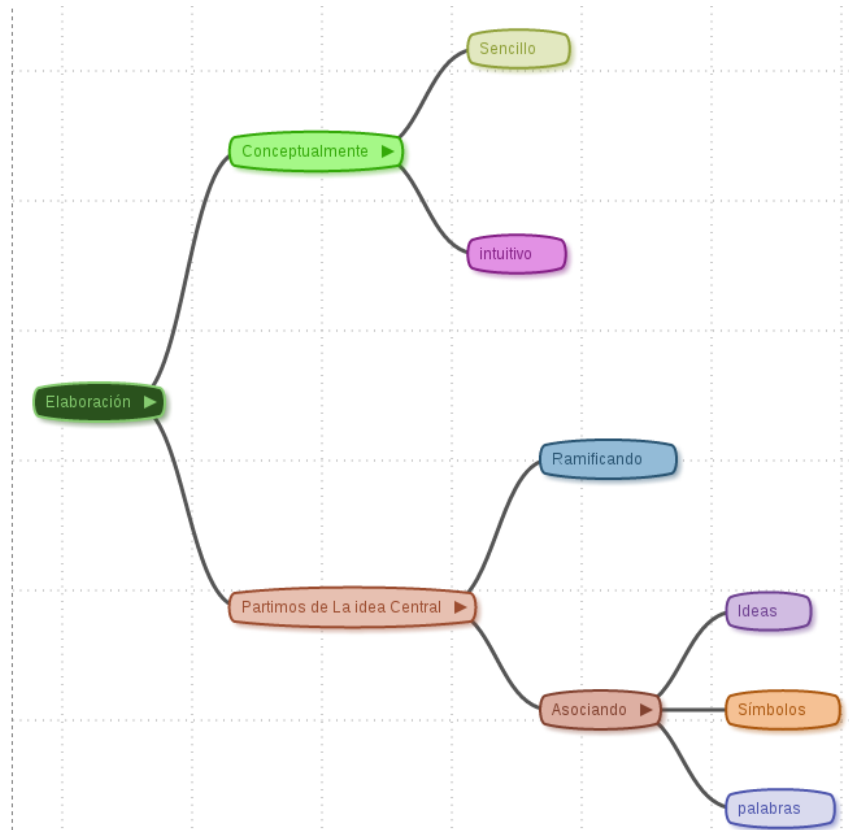


Figura 3.4: Mapa mental de elaboración de mapas mentales

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:

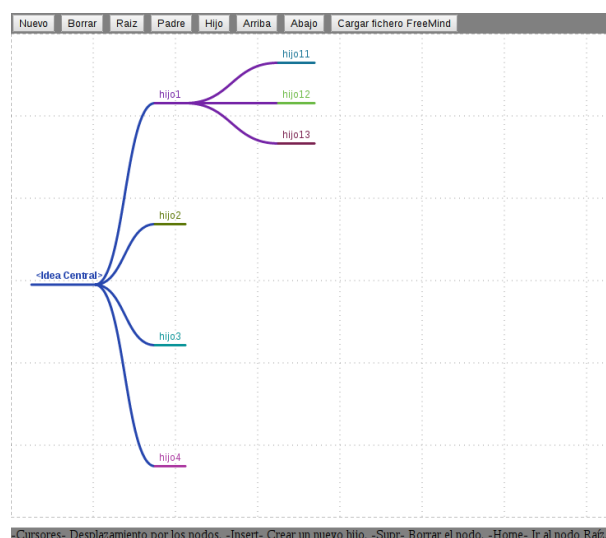


Figura 4.1: Versión inicial.

- **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.

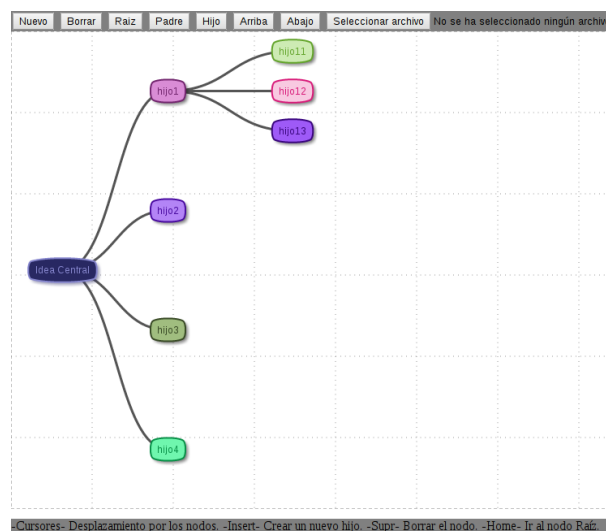


Figura 4.2: Versión inicial.

**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>.

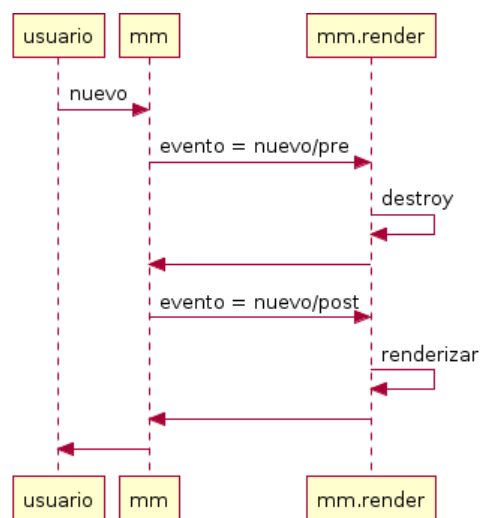


Figura 4.4: Diagrama de secuencia nuevo

### 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.

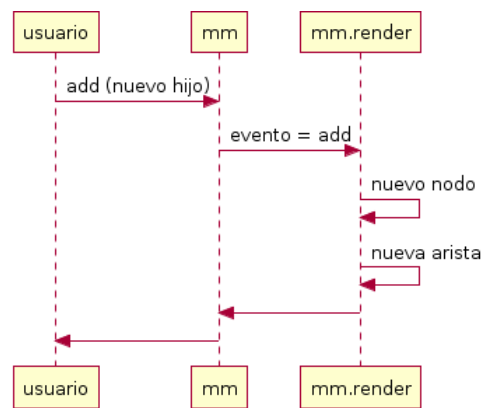


Figura 4.5: Diagrama de secuencia add

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

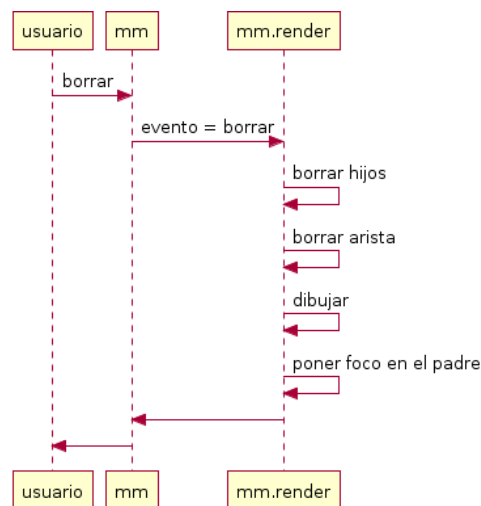


Figura 4.6: Diagrama de secuencia borrar

#### 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.

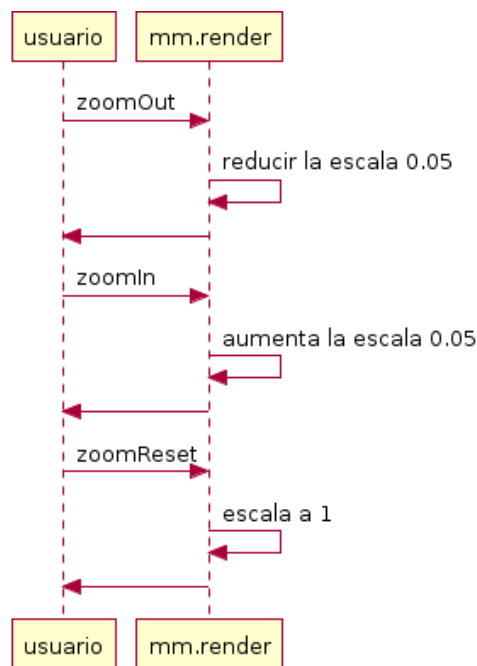


Figura 4.7: Diagrama de secuencia Zoom

#### 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>

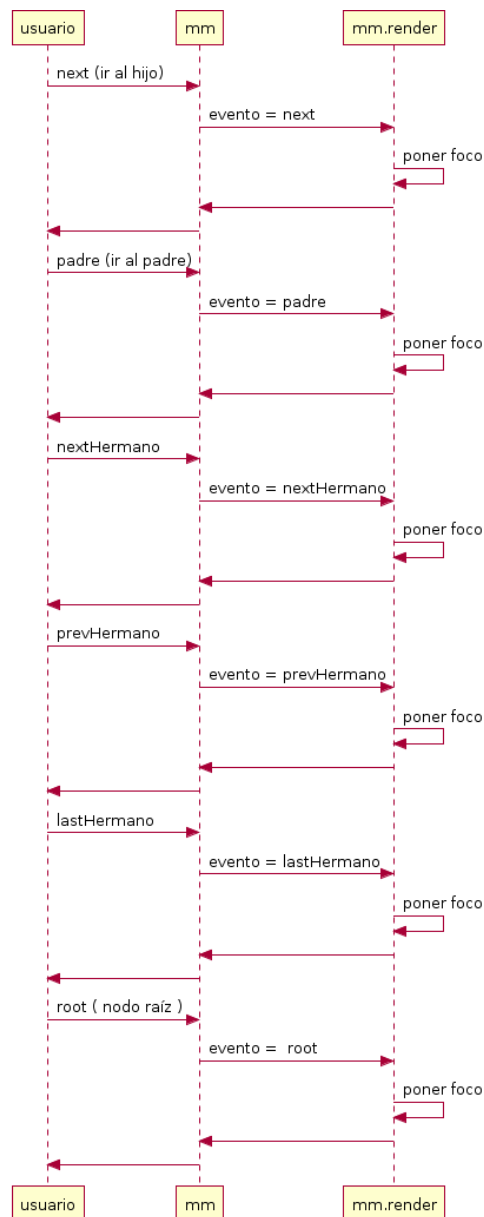


Figura 4.8: Diagrama de secuencia navegación

#### 4.2.7. Mover idea

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

#### 4.2.8. Mover mapa mental

Con el ratón y touch podremos desplazar el mapa.

#### 4.2.9. Salvar/cargar mapa mental

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

#### 4.2.10. Hacer/deshacer acciones

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

#### 4.2.11. Plegado/desplegado de ramas

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

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

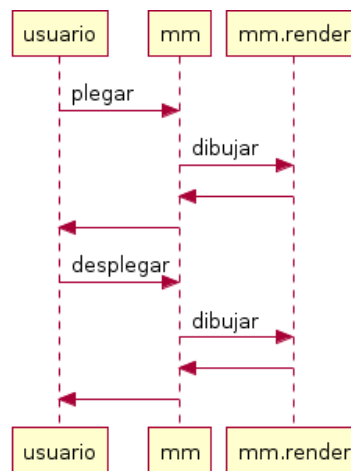


Figura 4.9: Diagrama de secuencia plegar

### 4.3. Diagramas de Clase

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



### Clase MM.Class

Como centro de todo el sistema de clases implementado está el MM.Class. Una abstracción del patrón constructor que es eje de todas las clases implementadas en la aplicación. A partir de ahora, cuando hable de clase me refiero a la herencia efectuada con MM.Class. La implementación de este objeto es fundamental ya que Javascript es un lenguaje orientado a objetos puro y libre de clases.

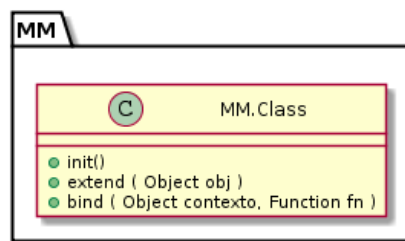


Figura 4.10: Clase MM.Class

Los principales métodos son:

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

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

#### 4.3.1. Diagrama de clases PubSub

Como núcleo 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-Suscriptor<sup>1</sup>.

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

<sup>1</sup>También conocido como patrón Observador-observable

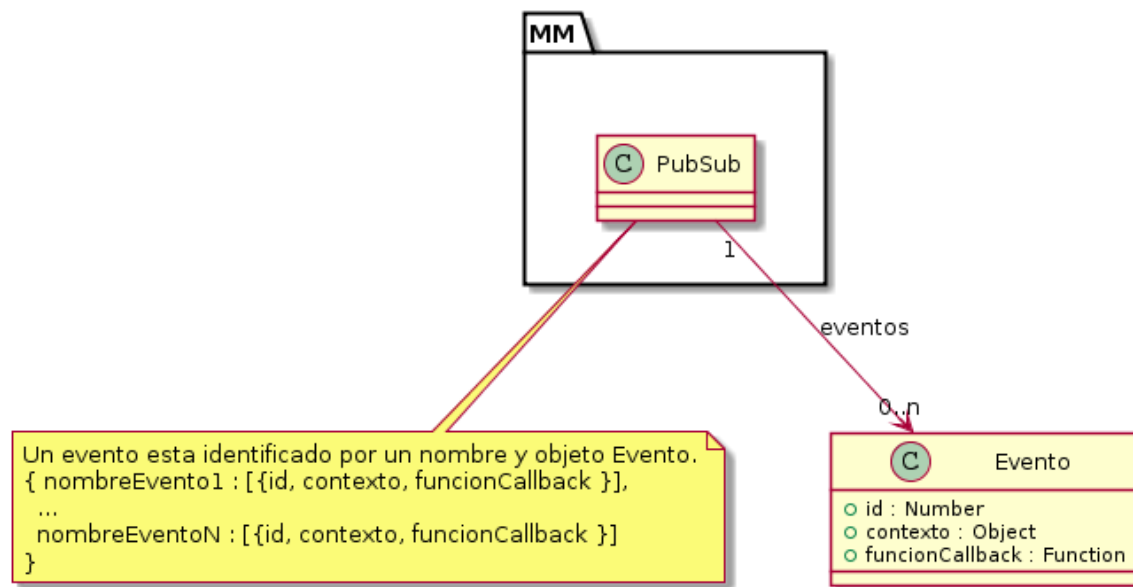


Figura 4.11: Diagrama de clases pubsub

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

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

#### MM.PubSub

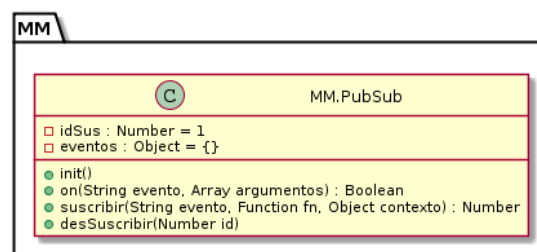


Figura 4.12: Clase MM.PubSub

Métodos:

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

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

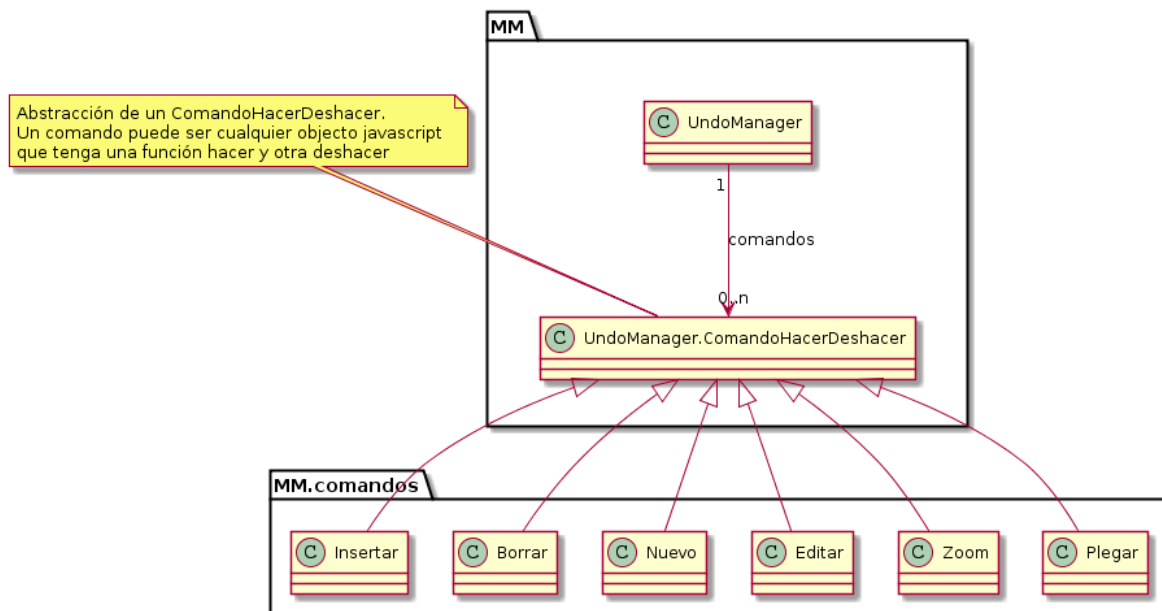


Figura 4.13: Diagrama de clases undo

#### 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 ...

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

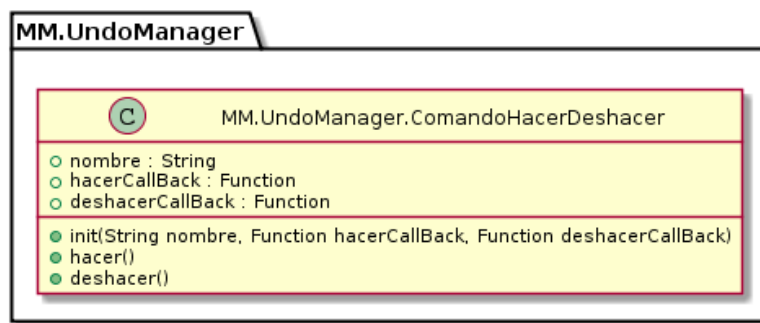


Figura 4.14: Clase MM.UndoManager.ComandoHacerDeshacer

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

#### Clase MM.UndoManager

El manejador de acciones hacer/deshacer tiene un registro de comandos ejecutados en la aplicación y un puntero<sup>2</sup> 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.
- **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.

<sup>2</sup>Campo actual.

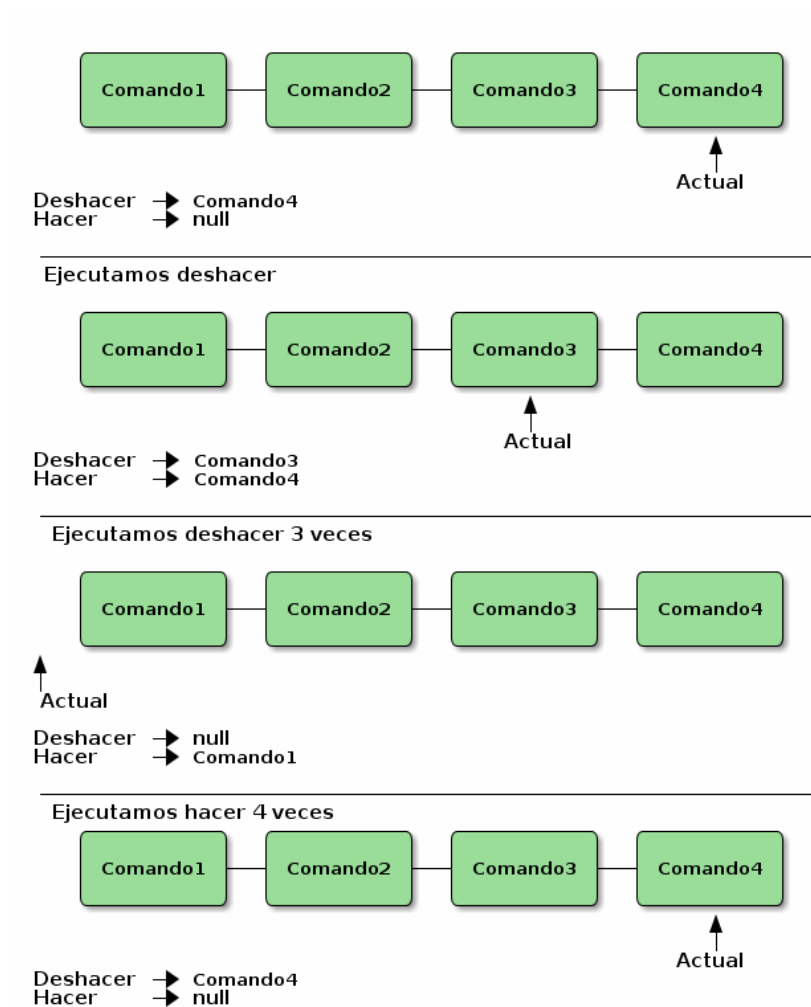


Figura 4.15: Secuencia de ejecución de UndoManager

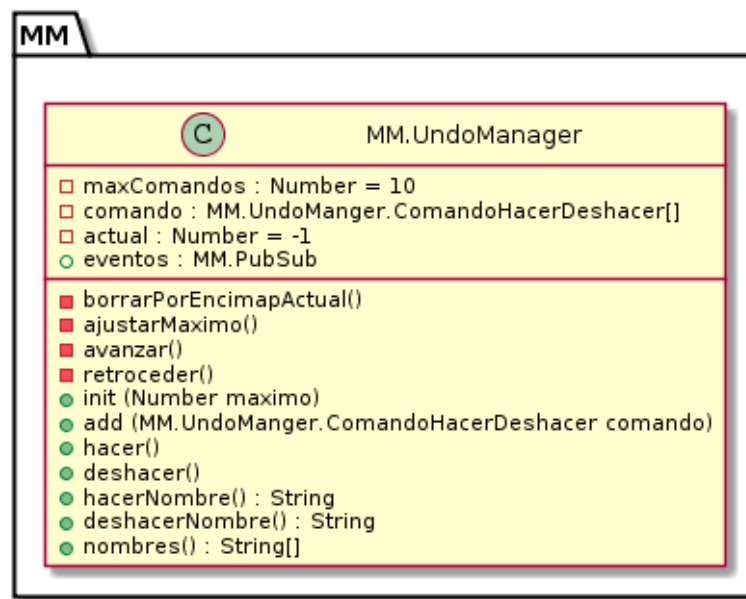


Figura 4.16: Clase MM.UndoManager

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

#### 4.3.3. Diagrama de clases MM

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

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

- Mantener la coherencia de los datos.

<sup>3</sup>Por ejemplo al render o al interface de usuario

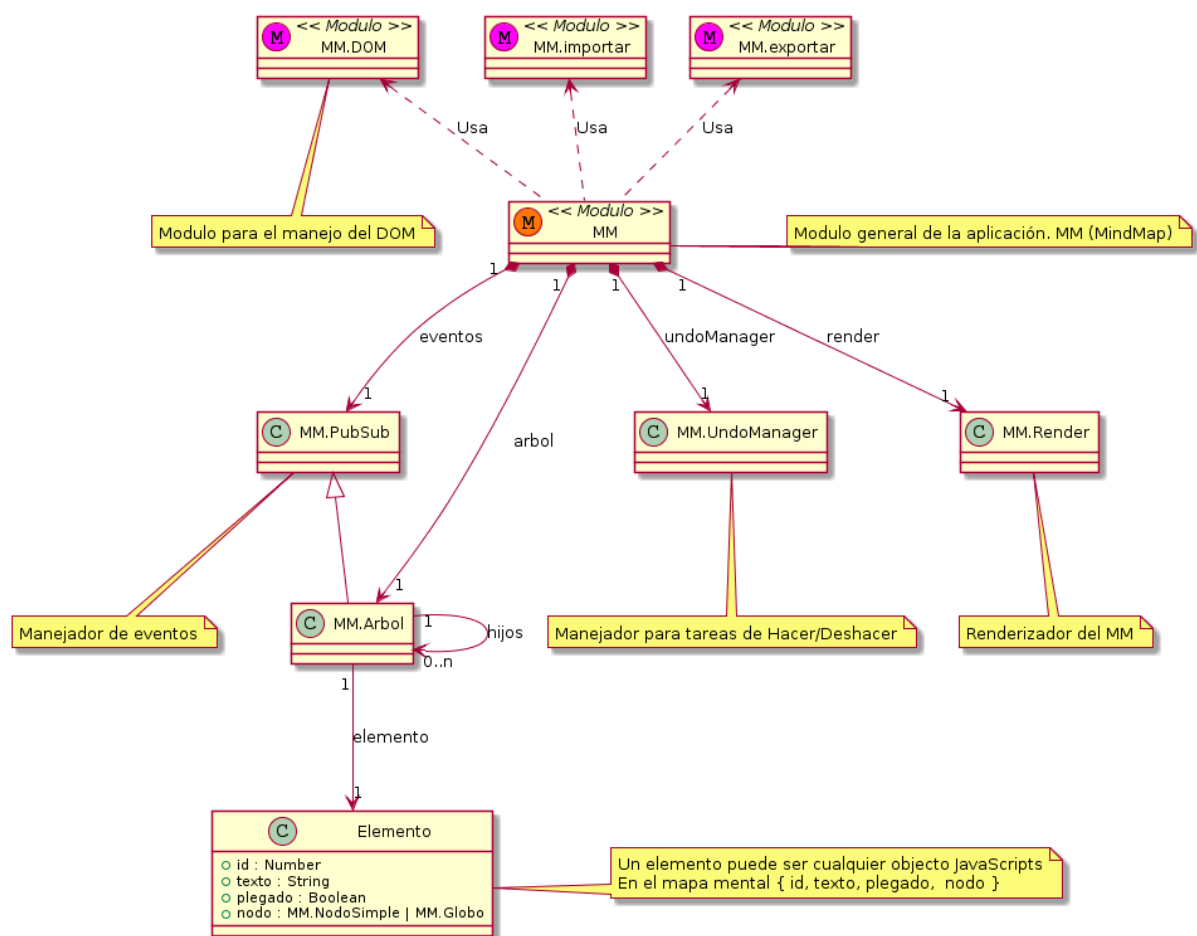


Figura 4.17: Diagrama de clases MM

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

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

#### Módulo MM

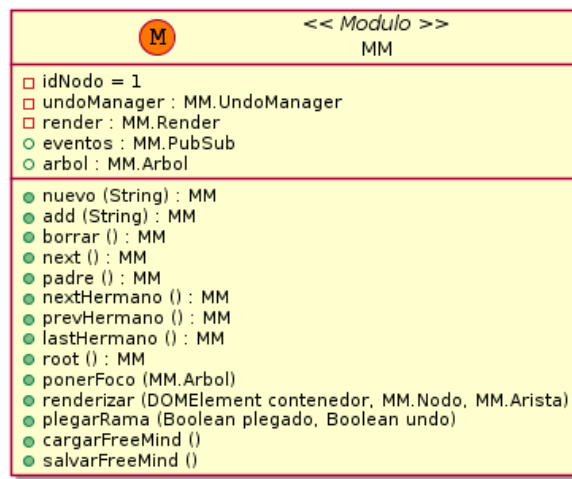


Figura 4.18: Clase MM

El módulo tiene los siguientes métodos:

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



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

#### Clase MM.Arbol

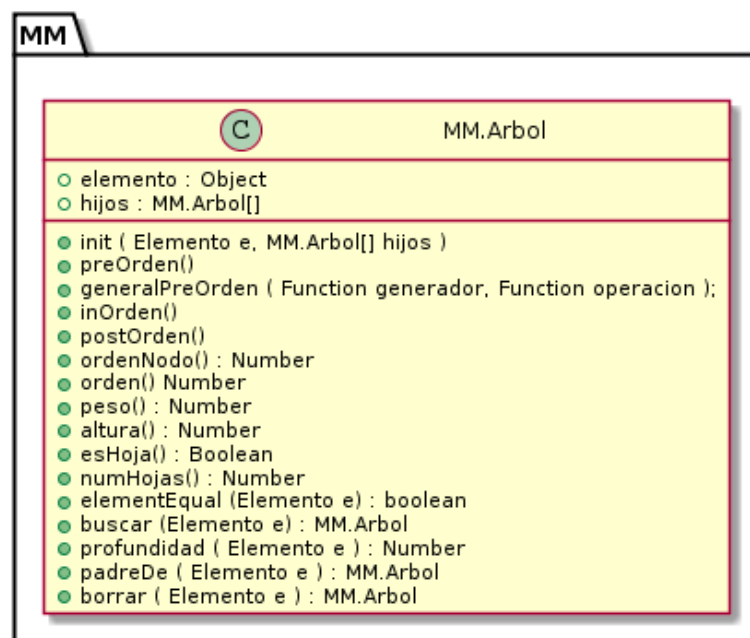


Figura 4.19: Clase MM.Arbol

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

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

con el preorden de los nodos hijos.

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

#### Módulo MM.DOM

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

#### Clase MM.Render

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

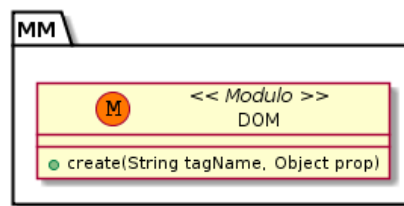


Figura 4.20: Modulo MM.DOM

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



Figura 4.21: Modulo MM.Render

La clase MM.Render dispone de los siguientes métodos:

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

<sup>4</sup>O que herede de MM.NodoSimple como MM.Globo

<sup>5</sup>O que herede de MM.Arista

- **MM.Render.renderizar:** se encarga de realizar las suscripciones a eventos, dibujar y establecer los atajos de teclado.
- **MM.Render.renderizarAristas:** pinta las aristas entre los distintos nodos.
- **MM.Render.dibubar:** en función del mapa actual del módulo MM dibuja y reparte el espacio de dibujo.
- **MM.Render.suscribirEventos / dessuscribirEventos:** métodos de activar y desactivar las suscripciones a eventos del render.
- **MM.Render.get/setEscala:** establece o devuelve la escala actual.
- **MM.Render.zoomIn / zoomOut / zoomReset :** funciones de zoom, en orden, aumenta, disminuye o reinicia la escala del mapa mental.
- **MM.Render.cambiarFoco:** se encarga de resalta la idea que tiene el foco actual.
- **MM.Render.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.

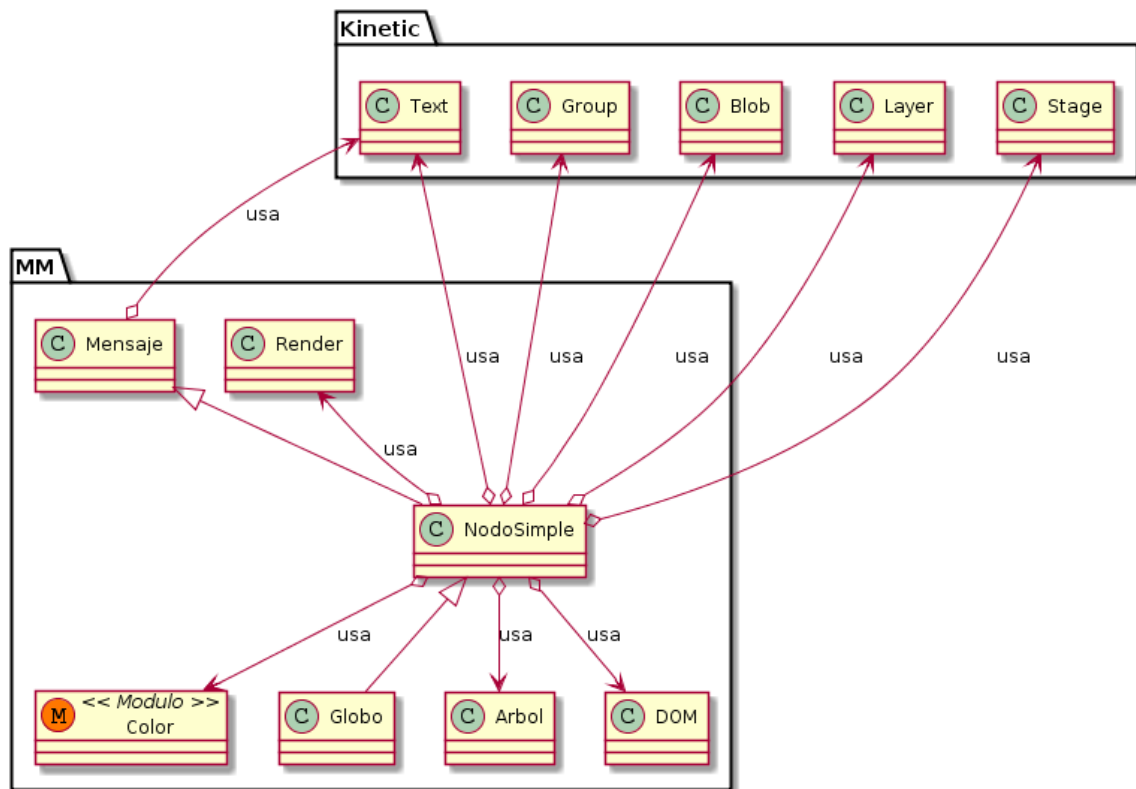


Figura 4.22: Diagrama de clases nodo

#### 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.
- **MM.Mensaje.getX/setX:** métodos para establecer y obtener la posición<sup>6</sup> X del mensaje.

<sup>6</sup>en píxeles

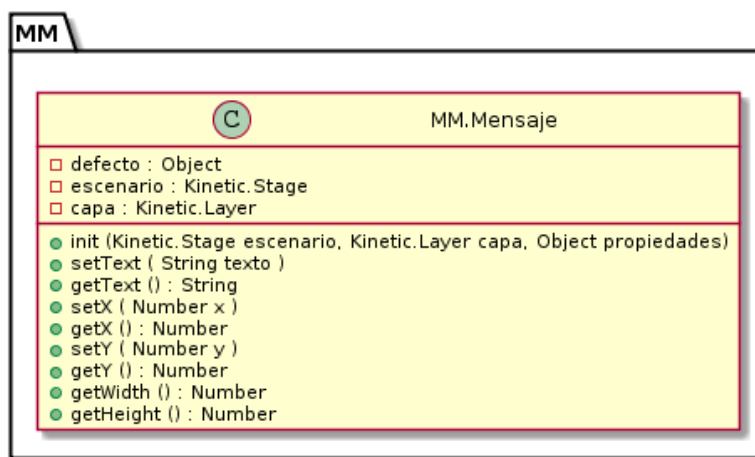


Figura 4.23: Clase MM.Mensaje

- **MM.Mensaje.getY/setY**: métodos para establecer y obtener la posición Y<sup>7</sup> 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 ponen o quitan el foco en la idea que representa el nodo. Debe resaltar el nodo cuando este esté focalizado.
- **MM.Mensaje.editar/cerrarEdicion**: métodos para establecer la idea en modo edición y para cerrarlo cuando se termine la edición. Si esta en modo edición debe tener el foco.

<sup>7</sup>en píxeles

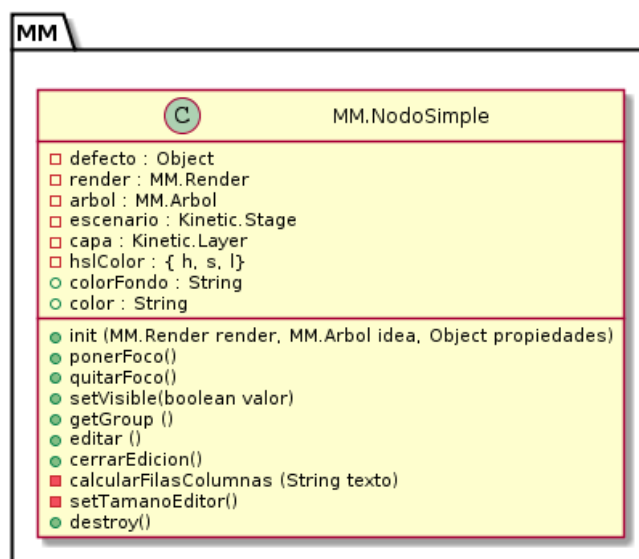


Figura 4.24: Clase MM.NodoSimple

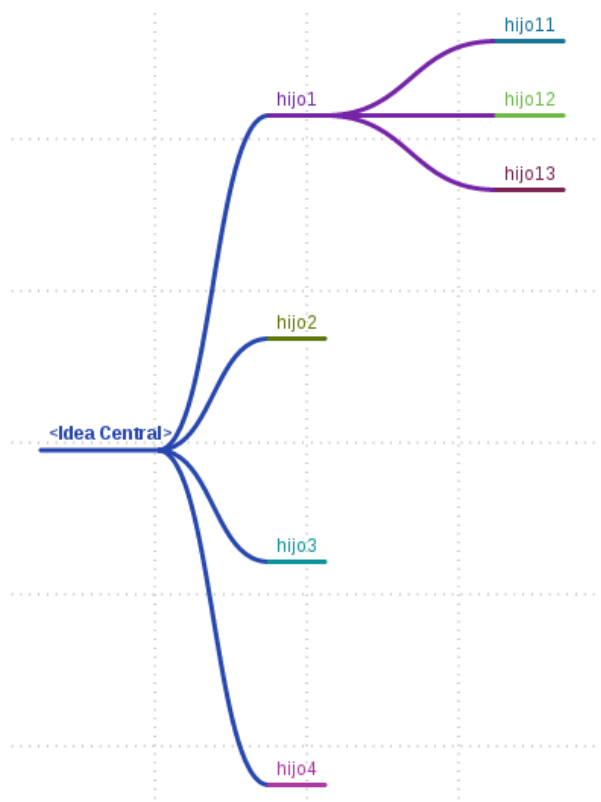


Figura 4.25: Mapa mental con renderización de nodos simples

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

Clase **MM.Globo**.

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

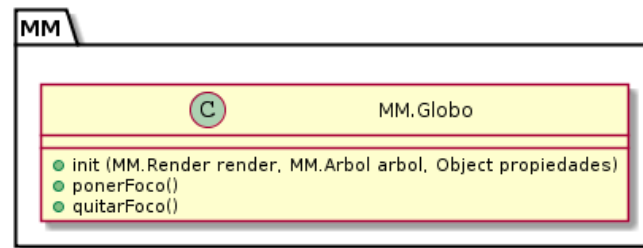


Figura 4.26: Clase **MM.Globo**



Figura 4.27: Mapa mental con renderización de nodos globo

Módulo **MM.Color**.

Módulo con funcionalidades de color. Permite generar distintas representaciones de color<sup>8</sup> y realizar conversiones sobre los distintos tipos.

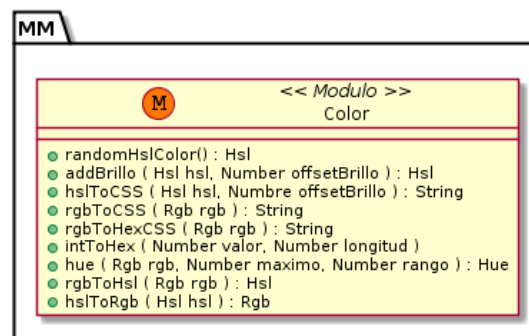


Figura 4.28: Módulo **MM.Color**

<sup>8</sup>HSL, RGB y HUE



#### 4.3.5. Diagrama de clases de aristas.

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

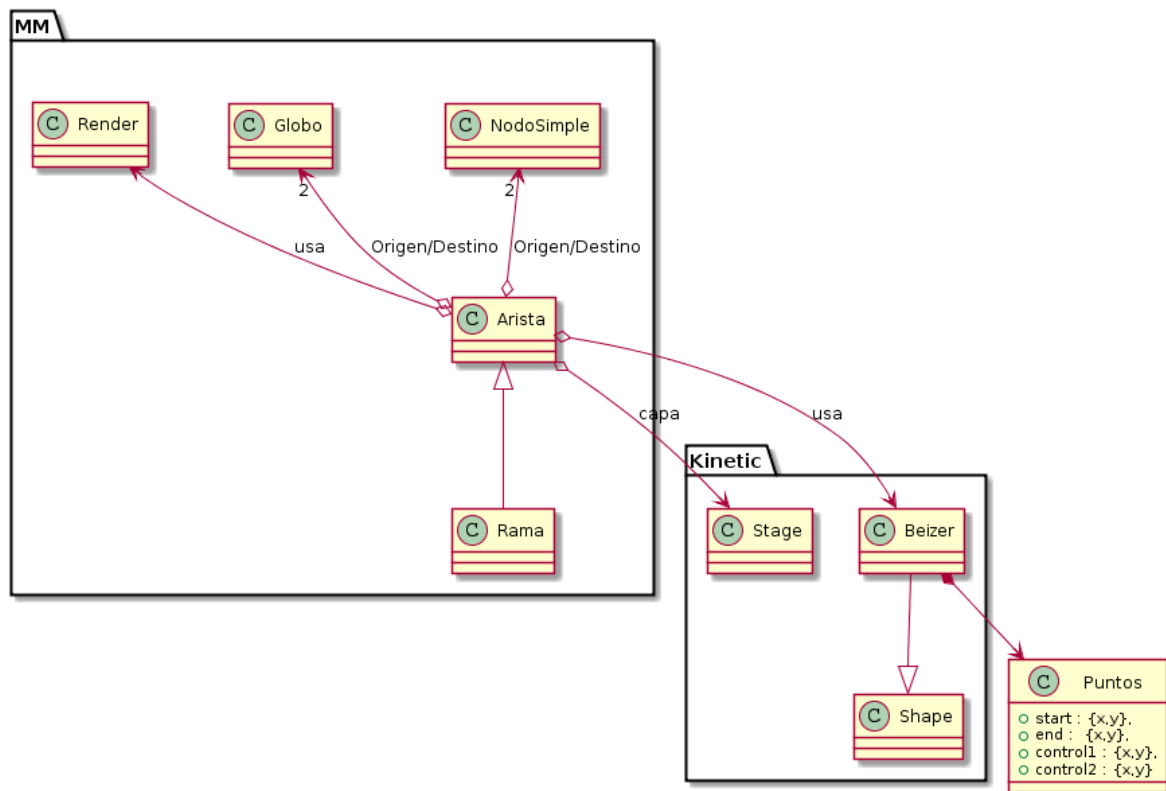


Figura 4.29: Diagrama de clases aristas

#### 4.3.6. Clase Kinetic.Beizer.

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

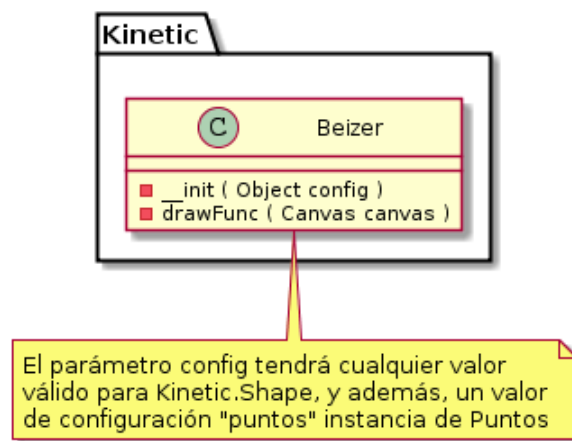


Figura 4.30: Clase Kinetic Beizer

**Clase MM.Arista.**

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

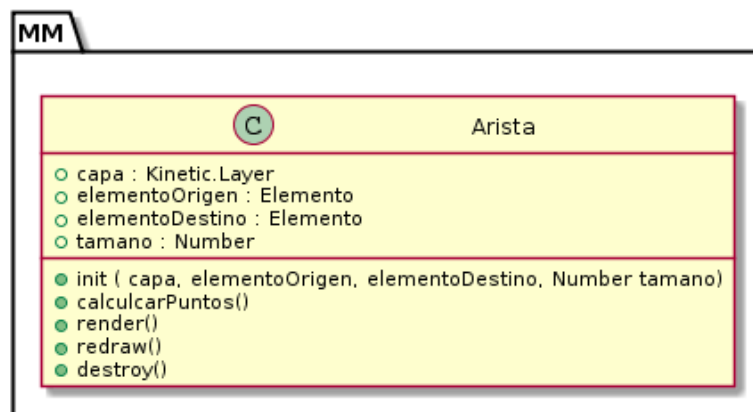


Figura 4.31: Clase MM.Arista

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

producidos en su entorno.

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

#### Clase MM.Rama.

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

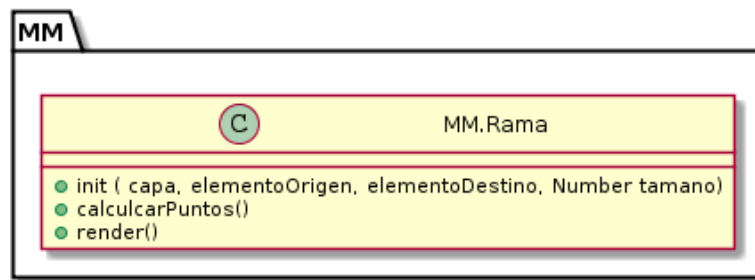


Figura 4.32: Clase MM.Rama

#### 4.3.7. Diagrama de clases de teclado

Para una mejor experiencia de usuario se ha implementado un complejo manejador de teclados para procesar 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*<sup>9</sup> o *Windows*, si no por que existen teclas como *+* que tienen distinto keycode en un Firefox, Chrome y Safari.

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

#### Módulo MM.teclado.atajos

Un atajo esta compuesto por un nombre<sup>10</sup>, 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.

<sup>9</sup>En los sistemas Mac.

<sup>10</sup>Ctrl+i

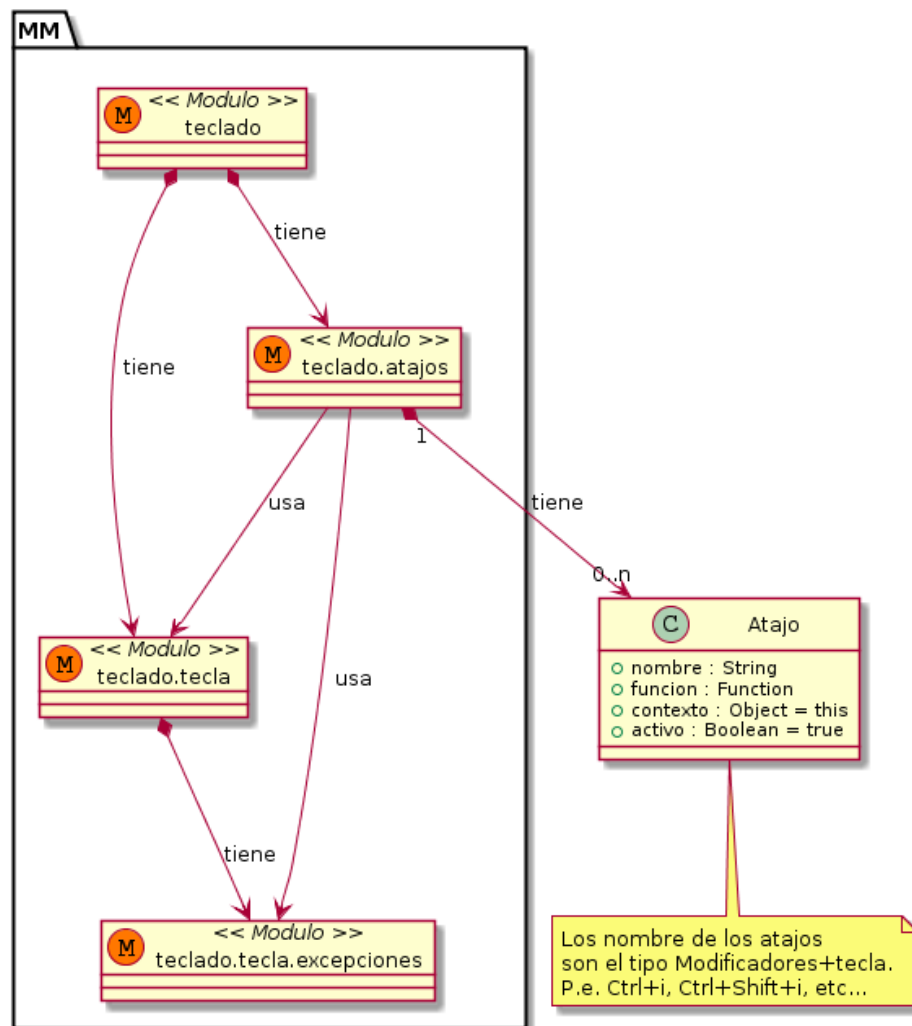


Figura 4.33: Diagrama de clases teclado

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

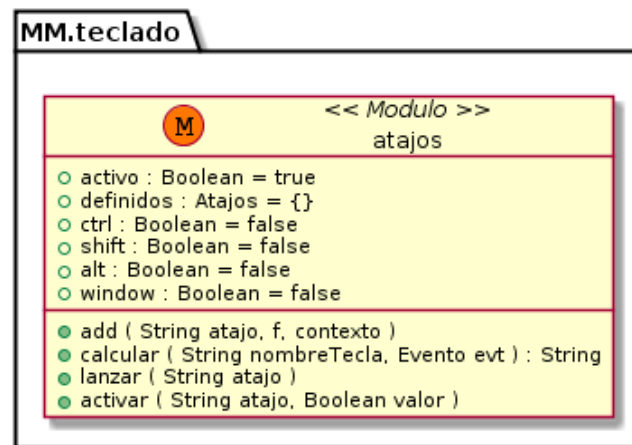


Figura 4.34: Módulo MM.teclado.atajos

- **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.
- **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>.

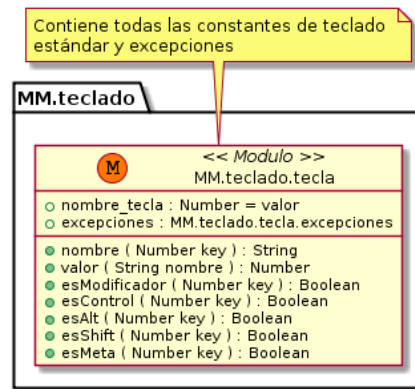


Figura 4.35: Clase MM.teclado.tecla

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

#### Módulo MM.teclado

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

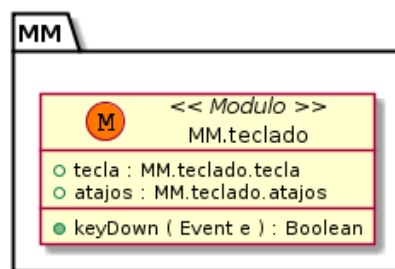


Figura 4.36: Clase MM.teclado

El sistema de control de teclado se encarga de recoger todos los eventos<sup>11</sup> 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.

<sup>11</sup>Todos los eventos KeyDown del navegador.

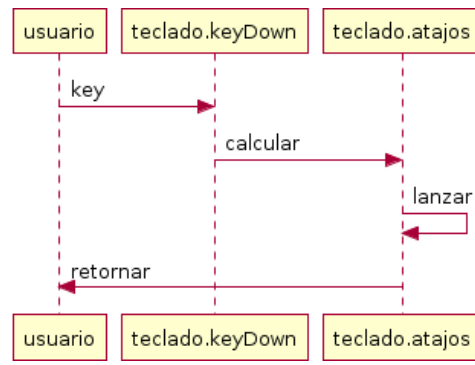


Figura 4.37: Diagrama de secuencia teclado

---

# Implementación.

## 5.1. Javascript.

### 5.1.1. Qué es.

Mozilla, los herederos directos de Netscape, definen javascript como:<sup>1 2</sup>

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

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

De C ha heredado que sea un lenguaje imperativo y estructurado con distinción entre sentencias y expresiones. A diferencia de C y Java el ámbito de las variables no son a

---

<sup>1</sup>MDN (Mozilla Developer Network)

<sup>2</sup>[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/JavaScript\\_Overview](https://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 objetos<sup>3</sup>. 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 Javascript 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 }
```

Javascript 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
```

<sup>3</sup>Salvo 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 Javascript<sup>4</sup> nadie podía intuir la importancia que adquiriría años después.

Internet aun estaba en pañales, navegar era lento<sup>5</sup> 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 Javascript al que le cambiaron el nombre para evitar problemas legales. De esta forma comienzan las divergencias entre las distintas versiones de Javascript, en esencia todas parten del mismo lenguaje y estandar, pero cada una aportaba sus mejoras provocando diferencias entre ellas.

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

<sup>4</sup> Javascript fue un nombre por conveniencia legal. Originalmente se llamaba LiveScript

<sup>5</sup> La velocidad máxima de los modems de usuario era 28.8Kbps

<sup>6</sup> European Computer Manufacturers Association. Web oficial <http://www.ecma-international.org/>

<sup>7</sup> 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 Javascript no utiliza el mecanismos de clases<sup>8</sup> 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, Javascript carece de métodos en su lugar existen propiedades que apuntan a funciones que hacen las veces de métodos. Además, cada objeto tiene un enlace interno a otro objeto llamado prototipo<sup>9</sup>. 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).

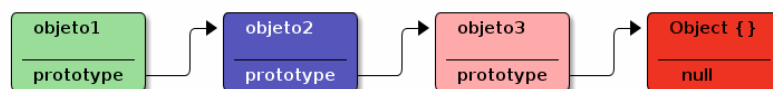


Figura 5.1: Cadena prototípica de objetos

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
  
```

<sup>8</sup>Paradigma sin clases

<sup>9</sup>prototype

## Herencia de propiedades y métodos

Un objeto javascript es un conjunto de propiedades<sup>10</sup> 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
```

<sup>10</sup>Los 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 variable<sup>11</sup> es la zona del programa donde es accesible la variable. En JavaScript existen dos ámbitos: local y global.

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

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

```
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 Javascript. Su simplicidad encierra gran potencia y flexibilidad que han aprovechado multitud de librerías<sup>12</sup>.

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

---

<sup>11</sup>scope

<sup>12</sup>Por 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 módulo<sup>13</sup>.

```

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 global<sup>14</sup>.
- Concepto simple y fácilmente extensible.

En MindMapJS no es una excepción, se ha utilizado un espacio de nombres MM.<sup>15</sup>

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

```

<sup>13</sup>Módulo propuesto por Douglas Crockford

<sup>14</sup>Nos referimos al propio módulo

<sup>15</sup>Utilizado 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 Javascript 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;
37

```

```

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   salutar: function(){ console.log(" hola ") }.chain();
5   };
6 }();
7 > mod.salutar().salutar().salutar() // " 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      * suscribimos
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.

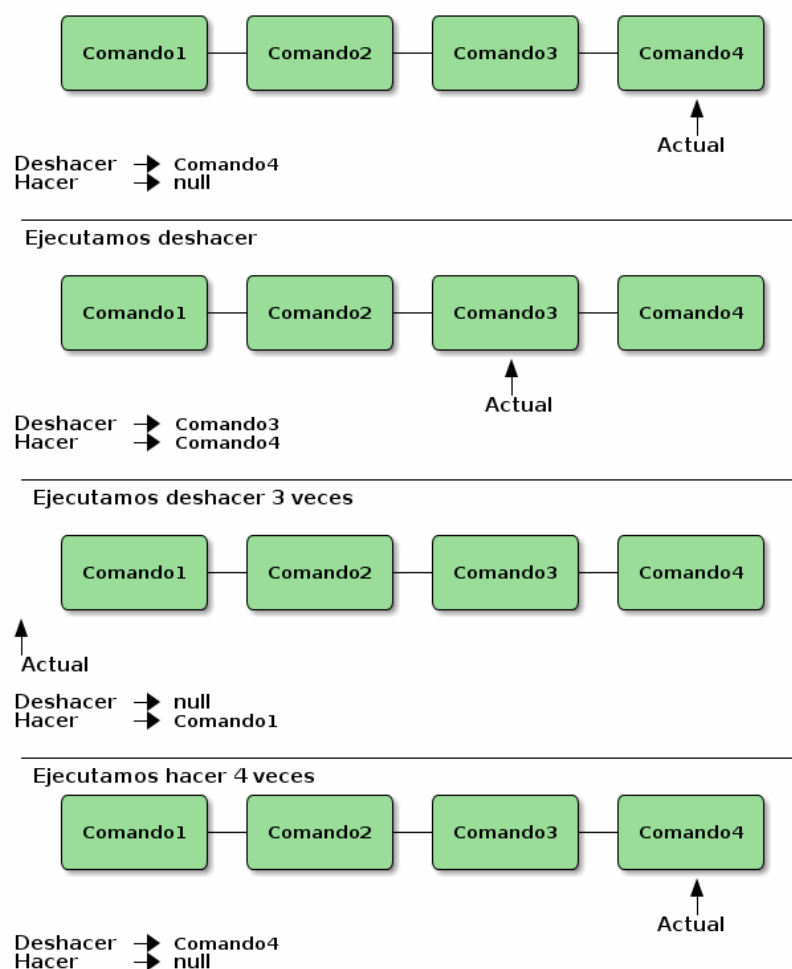


Figura 5.2: Secuencia de ejecución de UndoManager

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 de la pila de comandos. Si el tamaño del buffer
64      *       es menor que el máximo fijado, entonces no se elimina nada. Si el tamaño del buffer
65      *       es mayor o igual al máximo fijado, entonces se elimina el comando más antiguo.
66      * @memberof MM.UndoManager
67      * @instance
68      */
69     var borrarPorEncimaActual = function () {
70         if (comandos.length > maxComandos) {
71             comandos.splice(0, 1);
72         }
73     };
74
75     /**
76      * @desc Hace el comando actual. Si el comando actual es de tipo Hacer, entonces se ejecuta el comando. Si el comando actual es de tipo Deshacer, entonces se deshace el comando.
77      * @memberof MM.UndoManager
78      * @instance
79      */
80     var hacer = function () {
81         if (comandos[actual].tipo === 'Hacer') {
82             comandos[actual].hacer();
83         } else {
84             comandos[actual].deshacer();
85         }
86     };
87
88     /**
89      * @desc Deshace el comando actual. Si el comando actual es de tipo Hacer, entonces se deshace el comando. Si el comando actual es de tipo Deshacer, entonces se hace el comando.
90      * @memberof MM.UndoManager
91      * @instance
92      */
93     var deshacer = function () {
94         if (comandos[actual].tipo === 'Deshacer') {
95             comandos[actual].hacer();
96         } else {
97             comandos[actual].deshacer();
98         }
99     };
100
101     /**
102      * @desc Obtiene el estado del historial.
103      * @memberof MM.UndoManager
104      * @instance
105      */
106     var estado = function () {
107         return {
108             comandos: comandos,
109             actual: actual,
110             maxComandos: maxComandos
111         };
112     };
113
114     /**
115      * @desc Obtiene el nombre del siguiente comando hacer o deshacer.
116      * @memberof MM.UndoManager
117      * @instance
118      */
119     var siguienteComando = function () {
120         return comandos[actual + 1].nombre;
121     };
122
123     /**
124      * @desc Obtiene el nombre del comando actual.
125      * @memberof MM.UndoManager
126      * @instance
127      */
128     var nombreComandoActual = function () {
129         return comandos[actual].nombre;
130     };
131
132     /**
133      * @desc Obtiene el tipo del comando actual.
134      * @memberof MM.UndoManager
135      * @instance
136      */
137     var tipoComandoActual = function () {
138         return comandos[actual].tipo;
139     };
140
141     /**
142      * @desc Obtiene el índice del comando actual.
143      * @memberof MM.UndoManager
144      * @instance
145      */
146     var indiceComandoActual = function () {
147         return actual;
148     };
149
150     /**
151      * @desc Obtiene el tamaño máximo del buffer.
152      * @memberof MM.UndoManager
153      * @instance
154      */
155     var tamañoMaximoBuffer = function () {
156         return maxComandos;
157     };
158
159     /**
160      * @desc Obtiene el tamaño actual del buffer.
161      * @memberof MM.UndoManager
162      * @instance
163      */
164     var tamañoActualBuffer = function () {
165         return comandos.length;
166     };
167
168     /**
169      * @desc Obtiene el nombre del comando anterior al actual.
170      * @memberof MM.UndoManager
171      * @instance
172      */
173     var nombreComandoAnterior = function () {
174         return comandos[actual - 1].nombre;
175     };
176
177     /**
178      * @desc Obtiene el tipo del comando anterior al actual.
179      * @memberof MM.UndoManager
180      * @instance
181      */
182     var tipoComandoAnterior = function () {
183         return comandos[actual - 1].tipo;
184     };
185
186     /**
187      * @desc Obtiene el índice del comando anterior al actual.
188      * @memberof MM.UndoManager
189      * @instance
190      */
191     var indiceComandoAnterior = function () {
192         return actual - 1;
193     };
194
195     /**
196      * @desc Obtiene el nombre del comando siguiente al actual.
197      * @memberof MM.UndoManager
198      * @instance
199      */
200     var nombreComandoSiguiente = function () {
201         return comandos[actual + 1].nombre;
202     };
203
204     /**
205      * @desc Obtiene el tipo del comando siguiente al actual.
206      * @memberof MM.UndoManager
207      * @instance
208      */
209     var tipoComandoSiguiente = function () {
210         return comandos[actual + 1].tipo;
211     };
212
213     /**
214      * @desc Obtiene el índice del comando siguiente al actual.
215      * @memberof MM.UndoManager
216      * @instance
217      */
218     var indiceComandoSiguiente = function () {
219         return actual + 1;
220     };
221
222     /**
223      * @desc Obtiene el nombre del comando anterior al anterior al actual.
224      * @memberof MM.UndoManager
225      * @instance
226      */
227     var nombreComandoAnteriorAnterior = function () {
228         return comandos[actual - 2].nombre;
229     };
230
231     /**
232      * @desc Obtiene el tipo del comando anterior al anterior al actual.
233      * @memberof MM.UndoManager
234      * @instance
235      */
236     var tipoComandoAnteriorAnterior = function () {
237         return comandos[actual - 2].tipo;
238     };
239
240     /**
241      * @desc Obtiene el índice del comando anterior al anterior al actual.
242      * @memberof MM.UndoManager
243      * @instance
244      */
245     var indiceComandoAnteriorAnterior = function () {
246         return actual - 2;
247     };
248
249     /**
250      * @desc Obtiene el nombre del comando siguiente al siguiente al actual.
251      * @memberof MM.UndoManager
252      * @instance
253      */
254     var nombreComandoSiguienteSiguiente = function () {
255         return comandos[actual + 2].nombre;
256     };
257
258     /**
259      * @desc Obtiene el tipo del comando siguiente al siguiente al actual.
260      * @memberof MM.UndoManager
261      * @instance
262      */
263     var tipoComandoSiguienteSiguiente = function () {
264         return comandos[actual + 2].tipo;
265     };
266
267     /**
268      * @desc Obtiene el índice del comando siguiente al siguiente al actual.
269      * @memberof MM.UndoManager
270      * @instance
271      */
272     var indiceComandoSiguienteSiguiente = function () {
273         return actual + 2;
274     };
275
276     /**
277      * @desc Obtiene el nombre del comando anterior al anterior al anterior al actual.
278      * @memberof MM.UndoManager
279      * @instance
280      */
281     var nombreComandoAnteriorAnteriorAnterior = function () {
282         return comandos[actual - 3].nombre;
283     };
284
285     /**
286      * @desc Obtiene el tipo del comando anterior al anterior al anterior al actual.
287      * @memberof MM.UndoManager
288      * @instance
289      */
290     var tipoComandoAnteriorAnteriorAnterior = function () {
291         return comandos[actual - 3].tipo;
292     };
293
294     /**
295      * @desc Obtiene el índice del comando anterior al anterior al anterior al actual.
296      * @memberof MM.UndoManager
297      * @instance
298      */
299     var indiceComandoAnteriorAnteriorAnterior = function () {
300         return actual - 3;
301     };
302
303     /**
304      * @desc Obtiene el nombre del comando siguiente al siguiente al siguiente al actual.
305      * @memberof MM.UndoManager
306      * @instance
307      */
308     var nombreComandoSiguienteSiguienteSiguiente = function () {
309         return comandos[actual + 3].nombre;
310     };
311
312     /**
313      * @desc Obtiene el tipo del comando siguiente al siguiente al siguiente al actual.
314      * @memberof MM.UndoManager
315      * @instance
316      */
317     var tipoComandoSiguienteSiguienteSiguiente = function () {
318         return comandos[actual + 3].tipo;
319     };
320
321     /**
322      * @desc Obtiene el índice del comando siguiente al siguiente al siguiente al actual.
323      * @memberof MM.UndoManager
324      * @instance
325      */
326     var indiceComandoSiguienteSiguienteSiguiente = function () {
327         return actual + 3;
328     };
329
330     /**
331      * @desc Obtiene el nombre del comando anterior al anterior al anterior al anterior al actual.
332      * @memberof MM.UndoManager
333      * @instance
334      */
335     var nombreComandoAnteriorAnteriorAnteriorAnterior = function () {
336         return comandos[actual - 4].nombre;
337     };
338
339     /**
340      * @desc Obtiene el tipo del comando anterior al anterior al anterior al anterior al actual.
341      * @memberof MM.UndoManager
342      * @instance
343      */
344     var tipoComandoAnteriorAnteriorAnteriorAnterior = function () {
345         return comandos[actual - 4].tipo;
346     };
347
348     /**
349      * @desc Obtiene el índice del comando anterior al anterior al anterior al anterior al actual.
350      * @memberof MM.UndoManager
351      * @instance
352      */
353     var indiceComandoAnteriorAnteriorAnteriorAnterior = function () {
354         return actual - 4;
355     };
356
357     /**
358      * @desc Obtiene el nombre del comando siguiente al siguiente al siguiente al siguiente al actual.
359      * @memberof MM.UndoManager
360      * @instance
361      */
362     var nombreComandoSiguienteSiguienteSiguienteSiguiente = function () {
363         return comandos[actual + 4].nombre;
364     };
365
366     /**
367      * @desc Obtiene el tipo del comando siguiente al siguiente al siguiente al siguiente al actual.
368      * @memberof MM.UndoManager
369      * @instance
370      */
371     var tipoComandoSiguienteSiguienteSiguienteSiguiente = function () {
372         return comandos[actual + 4].tipo;
373     };
374
375     /**
376      * @desc Obtiene el índice del comando siguiente al siguiente al siguiente al siguiente al actual.
377      * @memberof MM.UndoManager
378      * @instance
379      */
380     var indiceComandoSiguienteSiguienteSiguienteSiguiente = function () {
381         return actual + 4;
382     };
383
384     /**
385      * @desc Obtiene el nombre del comando anterior al anterior al anterior al anterior al anterior al actual.
386      * @memberof MM.UndoManager
387      * @instance
388      */
389     var nombreComandoAnteriorAnteriorAnteriorAnteriorAnterior = function () {
390         return comandos[actual - 5].nombre;
391     };
392
393     /**
394      * @desc Obtiene el tipo del comando anterior al anterior al anterior al anterior al anterior al actual.
395      * @memberof MM.UndoManager
396      * @instance
397      */
398     var tipoComandoAnteriorAnteriorAnteriorAnteriorAnterior = function () {
399         return comandos[actual - 5].tipo;
400     };
401
402     /**
403      * @desc Obtiene el índice del comando anterior al anterior al anterior al anterior al anterior al actual.
404      * @memberof MM.UndoManager
405      * @instance
406      */
407     var indiceComandoAnteriorAnteriorAnteriorAnteriorAnterior = function () {
408         return actual - 5;
409     };
410
411     /**
412      * @desc Obtiene el nombre del comando siguiente al siguiente al siguiente al siguiente al siguiente al actual.
413      * @memberof MM.UndoManager
414      * @instance
415      */
416     var nombreComandoSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
417         return comandos[actual + 5].nombre;
418     };
419
420     /**
421      * @desc Obtiene el tipo del comando siguiente al siguiente al siguiente al siguiente al siguiente al actual.
422      * @memberof MM.UndoManager
423      * @instance
424      */
425     var tipoComandoSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
426         return comandos[actual + 5].tipo;
427     };
428
429     /**
430      * @desc Obtiene el índice del comando siguiente al siguiente al siguiente al siguiente al siguiente al actual.
431      * @memberof MM.UndoManager
432      * @instance
433      */
434     var indiceComandoSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
435         return actual + 5;
436     };
437
438     /**
439      * @desc Obtiene el nombre del comando anterior al anterior al anterior al anterior al anterior al anterior al actual.
440      * @memberof MM.UndoManager
441      * @instance
442      */
443     var nombreComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
444         return comandos[actual - 6].nombre;
445     };
446
447     /**
448      * @desc Obtiene el tipo del comando anterior al anterior al anterior al anterior al anterior al anterior al actual.
449      * @memberof MM.UndoManager
450      * @instance
451      */
452     var tipoComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
453         return comandos[actual - 6].tipo;
454     };
455
456     /**
457      * @desc Obtiene el índice del comando anterior al anterior al anterior al anterior al anterior al anterior al actual.
458      * @memberof MM.UndoManager
459      * @instance
460      */
461     var indiceComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
462         return actual - 6;
463     };
464
465     /**
466      * @desc Obtiene el nombre del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
467      * @memberof MM.UndoManager
468      * @instance
469      */
470     var nombreComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
471         return comandos[actual + 6].nombre;
472     };
473
474     /**
475      * @desc Obtiene el tipo del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
476      * @memberof MM.UndoManager
477      * @instance
478      */
479     var tipoComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
480         return comandos[actual + 6].tipo;
481     };
482
483     /**
484      * @desc Obtiene el índice del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
485      * @memberof MM.UndoManager
486      * @instance
487      */
488     var indiceComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
489         return actual + 6;
490     };
491
492     /**
493      * @desc Obtiene el nombre del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
494      * @memberof MM.UndoManager
495      * @instance
496      */
497     var nombreComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
498         return comandos[actual - 7].nombre;
499     };
500
501     /**
502      * @desc Obtiene el tipo del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
503      * @memberof MM.UndoManager
504      * @instance
505      */
506     var tipoComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
507         return comandos[actual - 7].tipo;
508     };
509
510     /**
511      * @desc Obtiene el índice del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
512      * @memberof MM.UndoManager
513      * @instance
514      */
515     var indiceComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
516         return actual - 7;
517     };
518
519     /**
520      * @desc Obtiene el nombre del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
521      * @memberof MM.UndoManager
522      * @instance
523      */
524     var nombreComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
525         return comandos[actual + 7].nombre;
526     };
527
528     /**
529      * @desc Obtiene el tipo del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
530      * @memberof MM.UndoManager
531      * @instance
532      */
533     var tipoComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
534         return comandos[actual + 7].tipo;
535     };
536
537     /**
538      * @desc Obtiene el índice del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
539      * @memberof MM.UndoManager
540      * @instance
541      */
542     var indiceComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
543         return actual + 7;
544     };
545
546     /**
547      * @desc Obtiene el nombre del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
548      * @memberof MM.UndoManager
549      * @instance
550      */
551     var nombreComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
552         return comandos[actual - 8].nombre;
553     };
554
555     /**
556      * @desc Obtiene el tipo del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
557      * @memberof MM.UndoManager
558      * @instance
559      */
560     var tipoComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
561         return comandos[actual - 8].tipo;
562     };
563
564     /**
565      * @desc Obtiene el índice del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
566      * @memberof MM.UndoManager
567      * @instance
568      */
569     var indiceComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
570         return actual - 8;
571     };
572
573     /**
574      * @desc Obtiene el nombre del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
575      * @memberof MM.UndoManager
576      * @instance
577      */
578     var nombreComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
579         return comandos[actual + 8].nombre;
580     };
581
582     /**
583      * @desc Obtiene el tipo del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
584      * @memberof MM.UndoManager
585      * @instance
586      */
587     var tipoComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
588         return comandos[actual + 8].tipo;
589     };
590
591     /**
592      * @desc Obtiene el índice del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
593      * @memberof MM.UndoManager
594      * @instance
595      */
596     var indiceComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
597         return actual + 8;
598     };
599
600     /**
601      * @desc Obtiene el nombre del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
602      * @memberof MM.UndoManager
603      * @instance
604      */
605     var nombreComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
606         return comandos[actual - 9].nombre;
607     };
608
609     /**
610      * @desc Obtiene el tipo del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
611      * @memberof MM.UndoManager
612      * @instance
613      */
614     var tipoComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
615         return comandos[actual - 9].tipo;
616     };
617
618     /**
619      * @desc Obtiene el índice del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
620      * @memberof MM.UndoManager
621      * @instance
622      */
623     var indiceComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
624         return actual - 9;
625     };
626
627     /**
628      * @desc Obtiene el nombre del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
629      * @memberof MM.UndoManager
630      * @instance
631      */
632     var nombreComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
633         return comandos[actual + 9].nombre;
634     };
635
636     /**
637      * @desc Obtiene el tipo del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
638      * @memberof MM.UndoManager
639      * @instance
640      */
641     var tipoComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
642         return comandos[actual + 9].tipo;
643     };
644
645     /**
646      * @desc Obtiene el índice del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
647      * @memberof MM.UndoManager
648      * @instance
649      */
650     var indiceComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
651         return actual + 9;
652     };
653
654     /**
655      * @desc Obtiene el nombre del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
656      * @memberof MM.UndoManager
657      * @instance
658      */
659     var nombreComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
660         return comandos[actual - 10].nombre;
661     };
662
663     /**
664      * @desc Obtiene el tipo del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
665      * @memberof MM.UndoManager
666      * @instance
667      */
668     var tipoComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
669         return comandos[actual - 10].tipo;
670     };
671
672     /**
673      * @desc Obtiene el índice del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
674      * @memberof MM.UndoManager
675      * @instance
676      */
677     var indiceComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
678         return actual - 10;
679     };
680
681     /**
682      * @desc Obtiene el nombre del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
683      * @memberof MM.UndoManager
684      * @instance
685      */
686     var nombreComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
687         return comandos[actual + 10].nombre;
688     };
689
690     /**
691      * @desc Obtiene el tipo del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
692      * @memberof MM.UndoManager
693      * @instance
694      */
695     var tipoComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
696         return comandos[actual + 10].tipo;
697     };
698
699     /**
700      * @desc Obtiene el índice del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
701      * @memberof MM.UndoManager
702      * @instance
703      */
704     var indiceComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
705         return actual + 10;
706     };
707
708     /**
709      * @desc Obtiene el nombre del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
710      * @memberof MM.UndoManager
711      * @instance
712      */
713     var nombreComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
714         return comandos[actual - 11].nombre;
715     };
716
717     /**
718      * @desc Obtiene el tipo del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
719      * @memberof MM.UndoManager
720      * @instance
721      */
722     var tipoComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
723         return comandos[actual - 11].tipo;
724     };
725
726     /**
727      * @desc Obtiene el índice del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
728      * @memberof MM.UndoManager
729      * @instance
730      */
731     var indiceComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
732         return actual - 11;
733     };
734
735     /**
736      * @desc Obtiene el nombre del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
737      * @memberof MM.UndoManager
738      * @instance
739      */
740     var nombreComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
741         return comandos[actual + 11].nombre;
742     };
743
744     /**
745      * @desc Obtiene el tipo del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
746      * @memberof MM.UndoManager
747      * @instance
748      */
749     var tipoComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
750         return comandos[actual + 11].tipo;
751     };
752
753     /**
754      * @desc Obtiene el índice del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
755      * @memberof MM.UndoManager
756      * @instance
757      */
758     var indiceComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
759         return actual + 11;
760     };
761
762     /**
763      * @desc Obtiene el nombre del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
764      * @memberof MM.UndoManager
765      * @instance
766      */
767     var nombreComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
768         return comandos[actual - 12].nombre;
769     };
770
771     /**
772      * @desc Obtiene el tipo del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
773      * @memberof MM.UndoManager
774      * @instance
775      */
776     var tipoComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
777         return comandos[actual - 12].tipo;
778     };
779
780     /**
781      * @desc Obtiene el índice del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
782      * @memberof MM.UndoManager
783      * @instance
784      */
785     var indiceComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
786         return actual - 12;
787     };
788
789     /**
790      * @desc Obtiene el nombre del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
791      * @memberof MM.UndoManager
792      * @instance
793      */
794     var nombreComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
795         return comandos[actual + 12].nombre;
796     };
797
798     /**
799      * @desc Obtiene el tipo del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
800      * @memberof MM.UndoManager
801      * @instance
802      */
803     var tipoComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
804         return comandos[actual + 12].tipo;
805     };
806
807     /**
808      * @desc Obtiene el índice del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
809      * @memberof MM.UndoManager
810      * @instance
811      */
812     var indiceComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
813         return actual + 12;
814     };
815
816     /**
817      * @desc Obtiene el nombre del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
818      * @memberof MM.UndoManager
819      * @instance
820      */
821     var nombreComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
822         return comandos[actual - 13].nombre;
823     };
824
825     /**
826      * @desc Obtiene el tipo del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
827      * @memberof MM.UndoManager
828      * @instance
829      */
830     var tipoComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
831         return comandos[actual - 13].tipo;
832     };
833
834     /**
835      * @desc Obtiene el índice del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
836      * @memberof MM.UndoManager
837      * @instance
838      */
839     var indiceComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
840         return actual - 13;
841     };
842
843     /**
844      * @desc Obtiene el nombre del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
845      * @memberof MM.UndoManager
846      * @instance
847      */
848     var nombreComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
849         return comandos[actual + 13].nombre;
850     };
851
852     /**
853      * @desc Obtiene el tipo del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
854      * @memberof MM.UndoManager
855      * @instance
856      */
857     var tipoComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
858         return comandos[actual + 13].tipo;
859     };
860
861     /**
862      * @desc Obtiene el índice del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
863      * @memberof MM.UndoManager
864      * @instance
865      */
866     var indiceComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
867         return actual + 13;
868     };
869
870     /**
871      * @desc Obtiene el nombre del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
872      * @memberof MM.UndoManager
873      * @instance
874      */
875     var nombreComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
876         return comandos[actual - 14].nombre;
877     };
878
879     /**
880      * @desc Obtiene el tipo del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
881      * @memberof MM.UndoManager
882      * @instance
883      */
884     var tipoComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
885         return comandos[actual - 14].tipo;
886     };
887
888     /**
889      * @desc Obtiene el índice del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
890      * @memberof MM.UndoManager
891      * @instance
892      */
893     var indiceComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
894         return actual - 14;
895     };
896
897     /**
898      * @desc Obtiene el nombre del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
899      * @memberof MM.UndoManager
900      * @instance
901      */
902     var nombreComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
903         return comandos[actual + 14].nombre;
904     };
905
906     /**
907      * @desc Obtiene el tipo del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
908      * @memberof MM.UndoManager
909      * @instance
910      */
911     var tipoComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
912         return comandos[actual + 14].tipo;
913     };
914
915     /**
916      * @desc Obtiene el índice del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
917      * @memberof MM.UndoManager
918      * @instance
919      */
920     var indiceComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
921         return actual + 14;
922     };
923
924     /**
925      * @desc Obtiene el nombre del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
926      * @memberof MM.UndoManager
927      * @instance
928      */
929     var nombreComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
930         return comandos[actual - 15].nombre;
931     };
932
933     /**
934      * @desc Obtiene el tipo del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
935      * @memberof MM.UndoManager
936      * @instance
937      */
938     var tipoComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
939         return comandos[actual - 15].tipo;
940     };
941
942     /**
943      * @desc Obtiene el índice del comando anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al anterior al actual.
944      * @memberof MM.UndoManager
945      * @instance
946      */
947     var indiceComandoAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnteriorAnterior = function () {
948         return actual - 15;
949     };
950
951     /**
952      * @desc Obtiene el nombre del comando siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al siguiente al actual.
953      * @memberof MM.UndoManager
954      * @instance
955      */
956     var nombreComandoSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguienteSiguiente = function () {
957         return comandos[actual + 15].nombre;
958     };
959
960     /**
961      * @desc Obtiene el tipo del comando siguiente al siguiente al siguiente al siguiente al siguiente
```

```

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.UndoMangaer.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 existiensen
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 Javascript. Con ello, no sólo reducimos el número de peticiones<sup>16</sup>, 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 UglifyJS<sup>17</sup> 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

<sup>16</sup>Más concretamente peticiones http get

<sup>17</sup>En combinación con GruntJS



y por lo tanto, el tiempo de carga<sup>18</sup>.

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.

<sup>18</sup>Más 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 Crockford<sup>19</sup>. 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:

---

<sup>19</sup>su creador

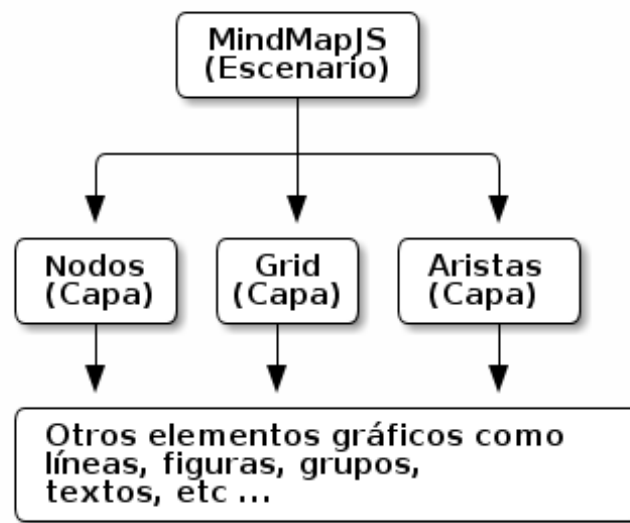


Figura 5.6: Escenario y modelo de capas KineticJS de MindMapJS

- **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.

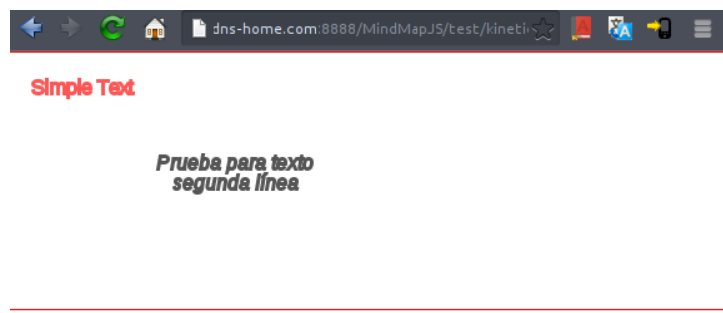


Figura 5.7: Ejemplo de Kinetic - uso básico de capas

```

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,
```

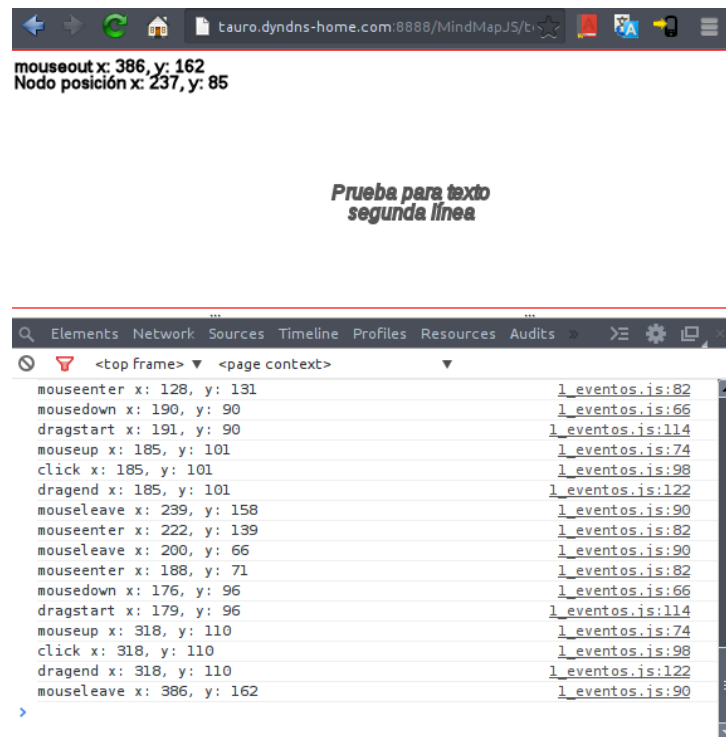


Figura 5.8: Ejemplo de Kinetic - eventos

```

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: <sup>20</sup>

```
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 KineticJS<sup>21</sup>.

#### 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);
```

<sup>20</sup>También implementada en MindMapJS

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



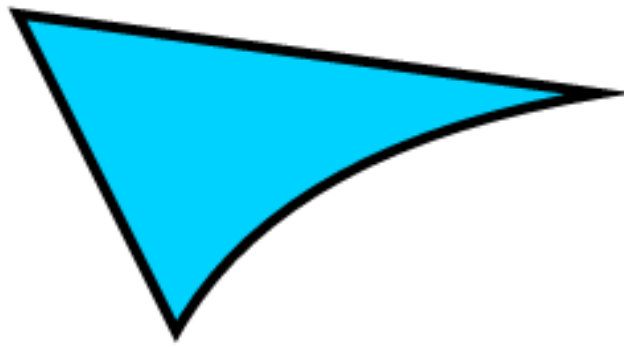


Figura 5.9: Ejemplo de Kineti - custom shape

```

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 JavaScript V8 de Google, NodeJS<sup>22</sup> a supuesto una revolución en el mundo de la programación JavaScript, dando un salto de gigante desde el lado del cliente al servidor. Este enorme evolución, y de manos de V8, ha provocado la creación de un entorno de programación completo, en el cual se aglutina desde un REPL<sup>23</sup> para pruebas y depuración interactiva hasta un gestor de paquetes y librerías NPM<sup>24</sup> (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 sólo 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.

<sup>22</sup>La web oficial de NodeJS es [nodejs.org](http://nodejs.org)

<sup>23</sup>Patrón Read-Eval-Print Loop

<sup>24</sup>La web oficial de NPM es [npmjs.org](http://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 GitHub<sup>25</sup>.

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 NPM<sup>26</sup> 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.
```

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

<sup>26</sup>Node Packaged Module (NPM) web oficial [npmjs.org](https://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 <sup>27</sup> para que lo puedan utilizar todas las librerías del sistema.

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

De forma local<sup>28</sup>, 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

---

<sup>27</sup>Con el modificador -g

<sup>28</sup>Sin el modificador -g

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

```
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 tareas<sup>29</sup>, pruebas<sup>30</sup> y verificación de código<sup>31</sup> entre otras muchas.

---

<sup>29</sup>GruntJS ha sido la elección utilizada como gestor de tareas

<sup>30</sup>En el caso de las pruebas unitarias opté por MochaJS

<sup>31</sup>JsHint

## 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 Javascript, evitándonos dentro de lo posible la realización de tareas repetitivas. Con un simple archivo de configuración nos permite realizar tareas tan diversas como minificar código, lanzar la suite de tests, etc.

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

### 5.6.2. 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/**/*.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 reliazar la verificación y validación de buenas prácticas establecidas en JavaScripts.

- **grunt-jsdoc:** compilar los comentarios JSDocs para generarla documentación HTML del API.
- **grunt-mocha-test:** tarea que lanza la suite de tests unitarios del proyecto.

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

- **dev:** que concatena el código fuente y realiza los reemplazo como fechas, versión, etc ...
- **full:** además de realizar las tareas propias de la tarea 'dev', minimiza y realiza los reemplazos de producción.
- **test:** lanza la suite de test
- **hint:** lanza la 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 Git<sup>32</sup>. Más concretamente se trata de un sistema distribuido de control de código fuente o SCM<sup>33</sup> creado por Linus Torvalds, a partir, de su propia experiencia en el desarrollo de los kernels de Linux.

Github<sup>34</sup> es una plataforma online pensada para el desarrollo colaborativo de proyectos, utilizando para ello Git. Github nos permite almacenar de forma pública<sup>35</sup> 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

---

<sup>32</sup>Web oficial de Git es [git-scm.com](http://git-scm.com)

<sup>33</sup>SCM (Source Code Management)

<sup>34</sup>La web de Github es [github.com](http://github.com)

<sup>35</sup>Github 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 markdown<sup>36</sup> 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
```

---

<sup>36</sup><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 rama<sup>37</sup>, realizar los cambios y subirlos<sup>38</sup> 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, JSDoc<sup>39</sup> es una herramienta inspirada en Javadoc<sup>40</sup> pero pensada para Javascript.

Mediante una conjunto de etiquetas (@class, @function, etc) introducidas como comentarios del código fuente, se generará la documentación en formato HTML<sup>41</sup>. 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

---

<sup>37</sup>Operaciones a realizar: branch y checkout

<sup>38</sup>Operaciones a realizar: commit y push

<sup>39</sup>Url del proyecto <https://github.com/jsdoc3/jsdoc>

<sup>40</sup><http://es.wikipedia.org/wiki/Javadoc>

<sup>41</sup>Por lo general, se genera HTML pero permite otros formatos como RTF.

```

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

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

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

    eventos : {},

    idSus : 1,

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

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

```

Figura 5.13: Ejemplo de código fuente documentado con JSDoc

funcional, de la documentación de nuestro código fuente.

## 5.9. Mocha

Mocha<sup>42</sup> es un framework Javascript para realizar pruebas unitarias. Sus creadores lo definen como:

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

Y sinceramente, creo que la definición no puede ser más acertada. Permite crear test con relativa facilidad y con una sintaxis clara y concisa. De puedo decir, que es "simple, flexible y divertido"<sup>43</sup>.

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

<sup>43</sup>En 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
<a href="#">id</a>	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 interface en el navegador y verificar su correcto funcionamiento.
- **Informes.** Tiene opciones para generar informes en varios formatos dependiendo de las necesidades.
- **Uso de cualquier librería de afirmaciones(Assertions).** Existe principalmente cuatro librerías que pueden ser utilizadas con Mocha Chai, Should, Expect y Better-Assert.
- **Soporte para test síncronos y asíncronos.** Permite abarcar todas las necesidades de nuestro código.

### 5.9.2. 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`<sup>44</sup>.

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 });

```

<sup>44</sup>En 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 structured as follows: 'Array' on one line, '#indexOf()' on the next, followed by two lines of test results, each preceded by a checkmark: '✓ 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.



---

# Resultados y discusión

## 6.1. Resultados

## 6.2. Discusión



---

# Manual de usuario

## 7.1. 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.

### 7.1.1. 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.

### 7.1.2. 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
```

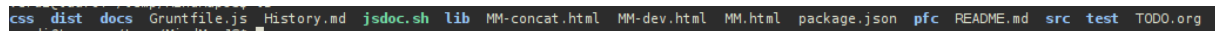


Figura 7.1: Estructura de directorios de MindMapJS

En la estructura del proyecto 7.1 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.

### 7.1.3. 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 7.2: Resultado de ejecutar el comando 'grunt dev'

#### 7.1.4. 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.

#### 7.1.5. 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.
```

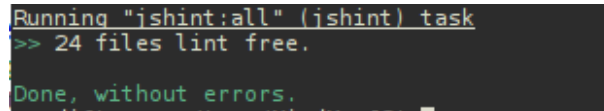
Figura 7.3: Resultado de ejecutar el comando 'grunt full'

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.

#### 7.1.6. 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.

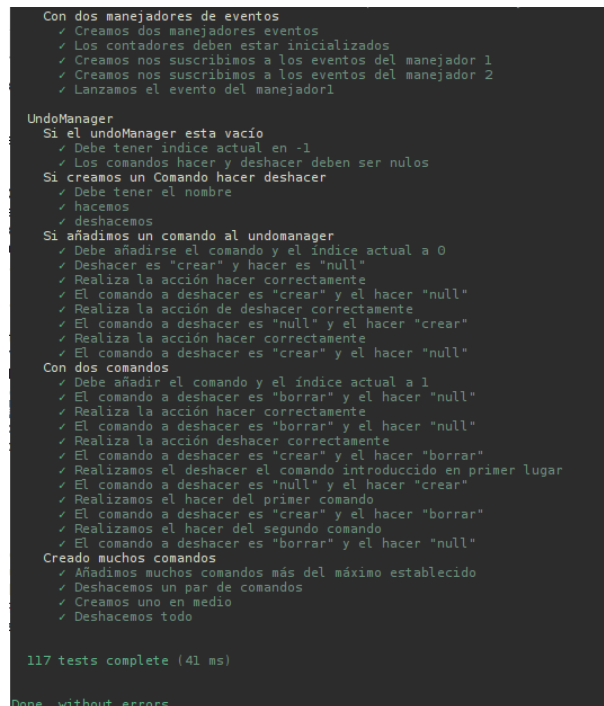
Done, without errors.
```

Figura 7.4: Resultado de ejecutar el comando 'grunt hint'

### 7.1.7. 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 7.5: Resultado de ejecutar el comando 'grunt test'

### 7.1.8. Generar la documentación del API

Si es necesario podemos generar API en formato JSDoc.

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

## 7.2. Manual de uso de MindMapJS





---

## Conclusiones

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

Ventajas del uso de MM.Class y desventajas. Consumo de memoria ya que se duplica los objetos vs mejor rendimiento por no tener que buscar en las propiedades y metodos en las cadenas de prototipos.

Si bien a menudo se considera uno de los puntos débiles de JavaScript, el modelo de herencia prototipado es de hecho más poderoso que el modelo clásico. Por ejemplo, es bastante trivial construir un modelo clásico a partir del modelo prototipado, mientras que al contrario es una tarea mucho más difícil.

Búsqueda de propiedades

Cuando se accede a las propiedades de un objeto, JavaScript recorre la cadena de prototipo hacia arriba hasta encontrar la propiedad con el nombre solicitado.

Cuando se llega al final de la cadena - concretamente `Object.prototype` - y aún no se ha encontrado la propiedad especificada, se retornará un valor `undefined` en su lugar.



---

## Bibliografía