UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

Editor de mapas mentales online con HTML5 y Javascript

Realizado por

José Luis Molina Soria

Dirigido por

Juan Antonio Falgueras Cano

Departamento

Lenguajes y Ciencias de la Computación

Málaga, Noviembre de 2013

UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

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

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

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

Vocal D^o/D^a. _____

para juzgar el proyecto Fin de Carrera titulado: _____

_____

_____

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

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

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

_____

ACORDÓ POR _____ OTORGAR LA CALIFICACIÓN

DE _____

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

Málaga a ___ de _____ de 20___

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

Fdo: | Fdo: | Fdo:

Agradecimientos

A mi familia y padres por su paciencia

Tabla de contenidos

1 Abstract **7**

2 Motivación y objetivos **9**

2.1 Motivación 9

2.2 Objetivos 11

2.3 Organización de la memoria 12

3 Introducción **14**

3.1 ¿Qué es? 14

3.2 Aplicaciones y beneficios. 16

3.3 Partes de un mapa mental. 17

3.4 Elaboración. 19

4 Materiales y métodos **21**

4.1 Metodología de análisis (UML-WAE) 21

4.2 Casos de uso. 25

4.3 Diagramas de Clase 29

4.4 Diagramas de secuencia. 48

5 Implementación. **53**

5.1 Metodología y etapas del desarrollo. 53

6 Resultados y discusión **58**

6.1 Resultados 58

6.2 Discusión 58

6.3 uglify 59

6.4 jsHint 59

6.5 KineticJS 59
6.6 NodeJS 59
6.7 GruntJs 62
6.8 Github 67
6.9 JSDoc 69
6.10 Mocha 70

7 Manual de usuario **75**

8 Conclusiones **77**

Bibliografía **78**

Índice de figuras

2.1 Esquema general de la especificación HTML5 a diciembre de 2011 10

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 Esquema de diagramas 22
4.2 Estereotipos 24
4.3 Valores etiquetados 24
4.4 Estereotipos 24
4.5 Casos de uso 26
4.6 Clase MM.Class 29
4.7 Diagrama de clases pubsub 30
4.8 Clase MM.PubSub 31
4.9 Diagrama de clases undo 32
4.10 Clase MM.UndoManager.ComandoHacerDeshacer 32
4.11 Secuencia de ejecución de UndoManager 33
4.12 Clase MM.UndoManager 34
4.13 Diagrama de clases MM 35
4.14 Clase MM 36
4.15 Clase MM.Arbol 37
4.16 Modulo MM.DOM 39
4.17 Modulo MM.Render 39
4.18 Diagrama de clases nodo 41
4.19 Clase MM.Mensaje 42
4.20 Clase MM.NodoSimple 43

4.21 Clase MM.Globo 43
4.22 Modulo MM.Color 44
4.23 Diagrama de clases aristas 45
4.24 Clase Kinetic Beizer 45
4.25 Clase MM.Arista 46
4.26 Clase MM.Rama 46
4.27 Diagrama de clases teclado 47
4.28 Modulo MM.teclado.atajos 48
4.29 Clase MM.teclado.tecla 49
4.30 Clase MM.teclado 49
4.31 Diagrama de secuencia add 50
4.32 Diagrama de secuencia borrar 50
4.33 Diagrama de secuencia navegación 51

6.1 Logo NodeJS 59
6.2 Logo GruntJS 63
6.3 Mascota de Github 67
6.4 Ejemplo de código fuente documentado con JSDoc 70
6.5 Página generada por JSDoc 71
6.6 Mocha 72
6.7 Resultado de ejecutar mocha -R spec *.js 73
6.8 Resultado de ejecutar mocha -R spec array1-test.js 73

Capítulo

1

Abstract

Capítulo

2

Motivación y objetivos

En la Motivación y Objetivos se presenta la necesidad de nuestro trabajo en el contexto científico y qué objetivos se han perseguido cubrir, en qué aspectos este TFM ha profundizado o innovado. No suelen incluirse referencias bibliográficas en este apartado, que es de presentación general. Los objetivos del TFM deben quedar enumerados claramente, así como los resultados inicialmente esperados. Destacar la originalidad e innovación del trabajo.

2.1. Motivación

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

Estamos viendo como día a día aplicaciones que han sido por antonomasia nativas (editores de texto, hojas de cálculo, aplicaciones de gestión, juegos ...), están siendo

[1] Hypertext Markup Language

implementadas con tecnologías web con gran éxito. Los editores de mapas mentales también han dado ese paso existen aplicaciones como text2mindmap[2], MindMeister[3], etc., tan versátiles o más que las propias aplicaciones nativas.

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

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

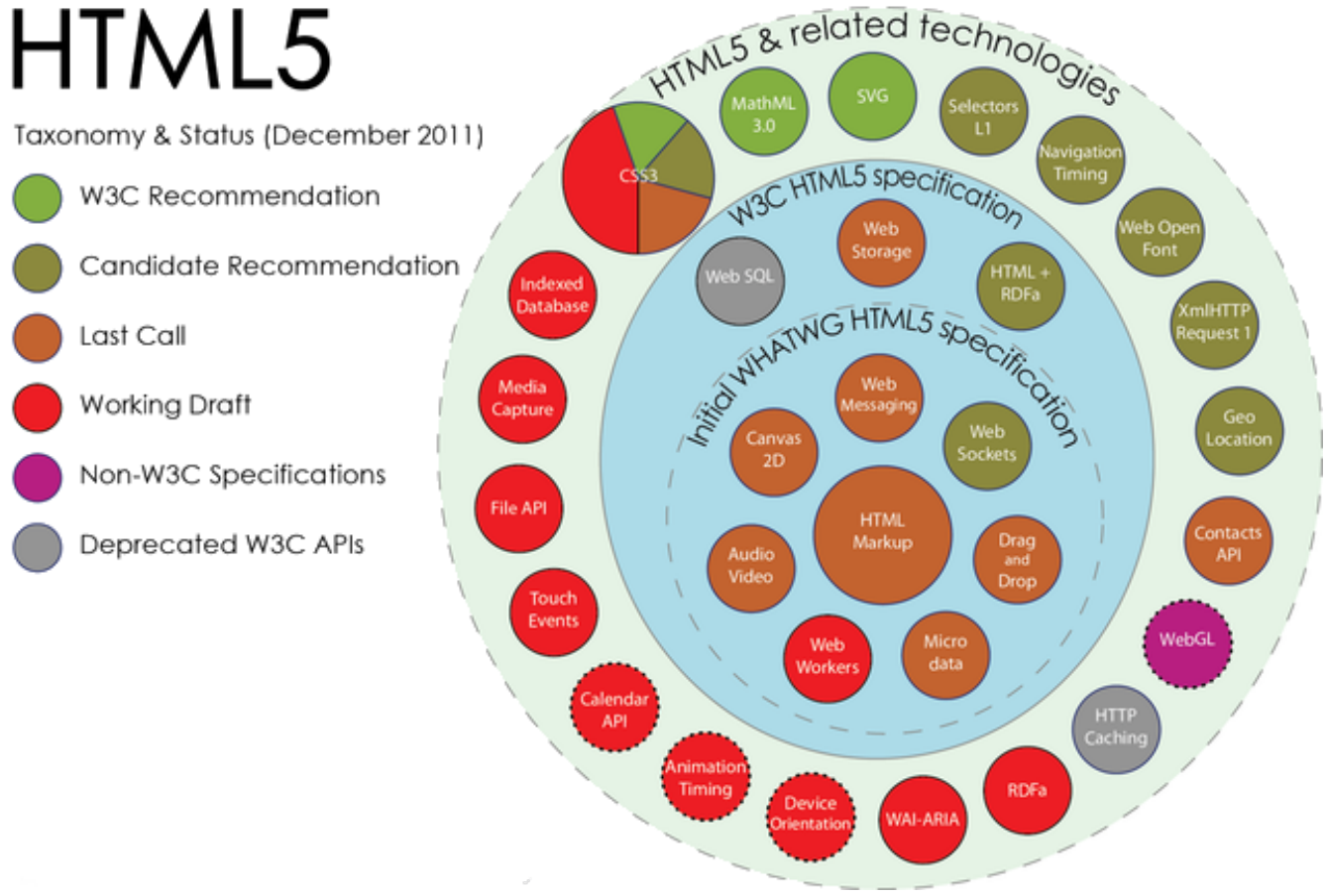


Figura 2.1: Esquema general de la especificación HTML5 a diciembre de 2011

- **Nuevas etiquetas estructurales.** Se ha incorporado un nuevo conjunto de etiquetas pensadas para definir mejor la estructura de una web, entre las más

[2]http://www.text2mindmap.com/
[3]http://www.mindmeister.com/es
[4]El primer borrador de HTML5 fue publicado en 2008
[5]HTML 4.01

importantes están las de encabezado, barra de navegación, secciones y pie.

- **Manejo de imágenes.** En la versión anterior podía incorporar imágenes ahora, además podemos modificarlas e interactuar con ellas. También disponemos un una etiqueta y un API completo para manejo de canvas[6].
- **Etiquetas de vídeo y audio.** Sin incluir flash ni aplicaciones externas podemos incorporar un reproductor de vídeo y/o audio.
- **Mejora en la semántica web.** HTML5 incluye elementos que permiten dar 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.

[6] En principio, sólo en 2D.

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

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

2.3. Organización de la memoria

El presente documento esta divido en los siguiente capítulos.

1. Introducción.

   Descripción general del proyecto y exposición de la estructura de la memoria.

2. Mapas mentales.

   Apartado para descubrir que es, historia y usos de los mapas mentales.

3. Diseño e implementación.

   En este capítulo se muestra la metodología, diseño, estudio e implementaciones para llevar a cabo la realización del proyecto.

4. Herramientas utilizadas.

   Descripción de todas y cada una de las herramientas utilizas en el desarrollo del proyecto.

5. Manual de usuarios.

   Manual para el usuario final.

6. Conclusiones.

   Capítulo para la conclusiones finales e impresiones.

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

Capítulo

3 Introducción

En la introducción al tema se presentará precisa y concisamente la situación actual de la disciplina tratado por el TFM (aquí sí que suele haber gran número de referencias bibliográficas relevantes que introduzcan y lleven a los últimos avances o estado o situación actual del tema). La descripción de la situación actual del tema debe ser breve y con referencias. Destacar, si es el caso, cualquier información relativa a los aspectos interdisciplinares o multidisciplinares que surjan y se contemplen en el trabajo aunque no se desarrollen, pero sí se referencian.

Dentro de la introducción se debe terminar especificando hasta dónde llega la investigación previa que haya guiado a la aportada.

Aclarar su relación con líneas anteriores del estado de la cuestión.

3.1. ¿Qué es?

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

Por naturaleza, nuestro celebro tiene un potencial ilimitado y que, en muchas ocasiones es desaprovechado o difícil de interpretar. Tenemos dos hemisferios el izquierdo y el

derecho, el racional y el creativo, ambos funcionan de forma separada. Los mapas mentales consiguen relacionar ambos hemisferios (racional y creativo) y lograr que funcionen conjuntamente.

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

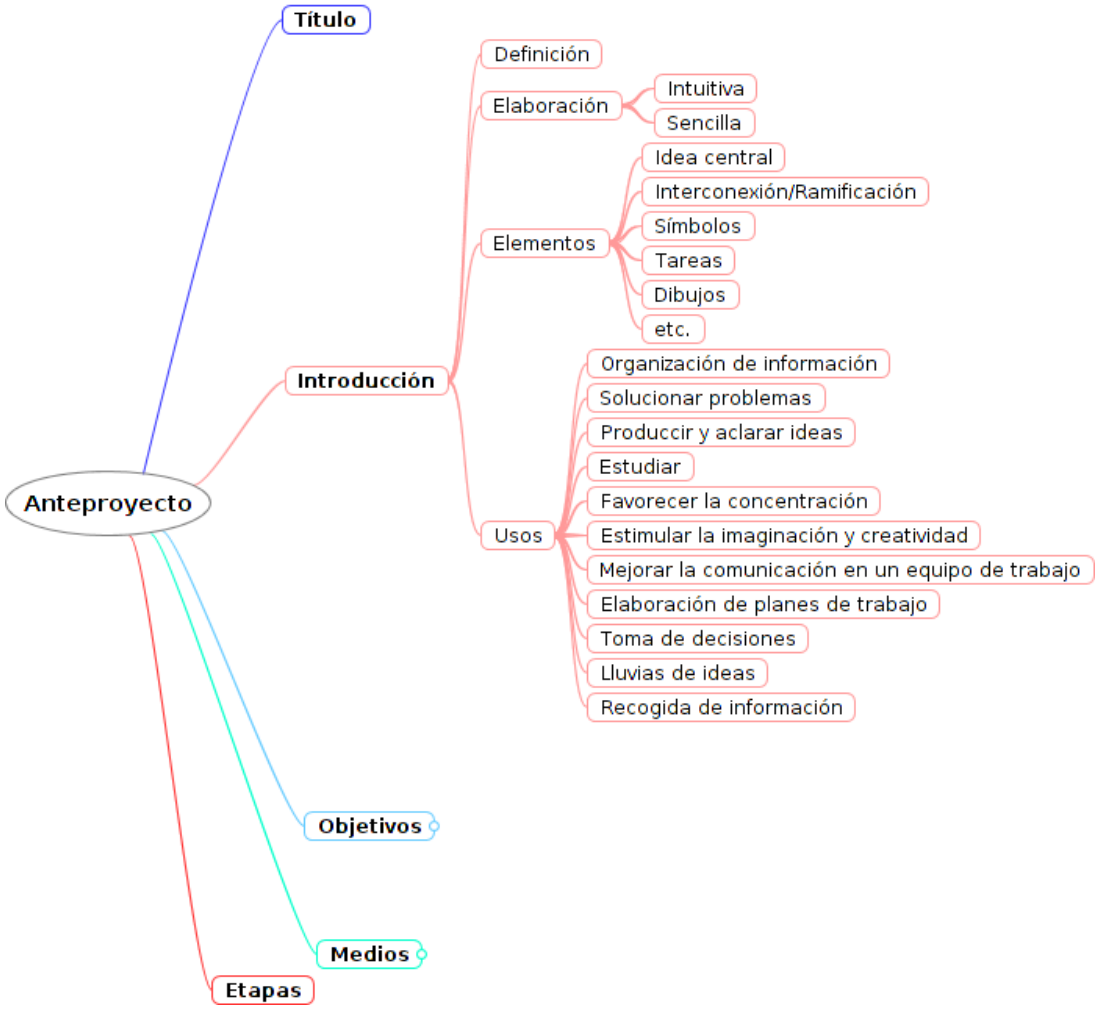


Figura 3.1: Mapa mental de FreeMind

[1] que irradia

3.2. Aplicaciones y beneficios.

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

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

3.3. Partes de un mapa mental.

3.3.1. Según su estructura.

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

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

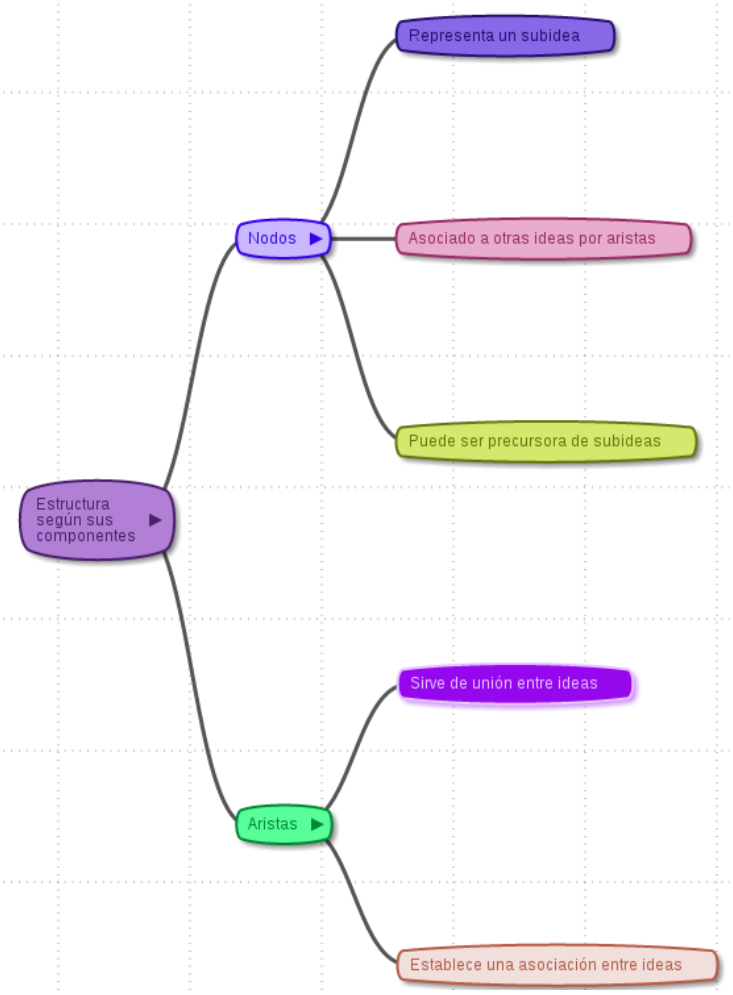


Figura 3.2: Partes de un mapa mental

3.3.2. Por contenido.

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

1. Idea central

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

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

4. Las ramas forman una estructura nodal conectada.

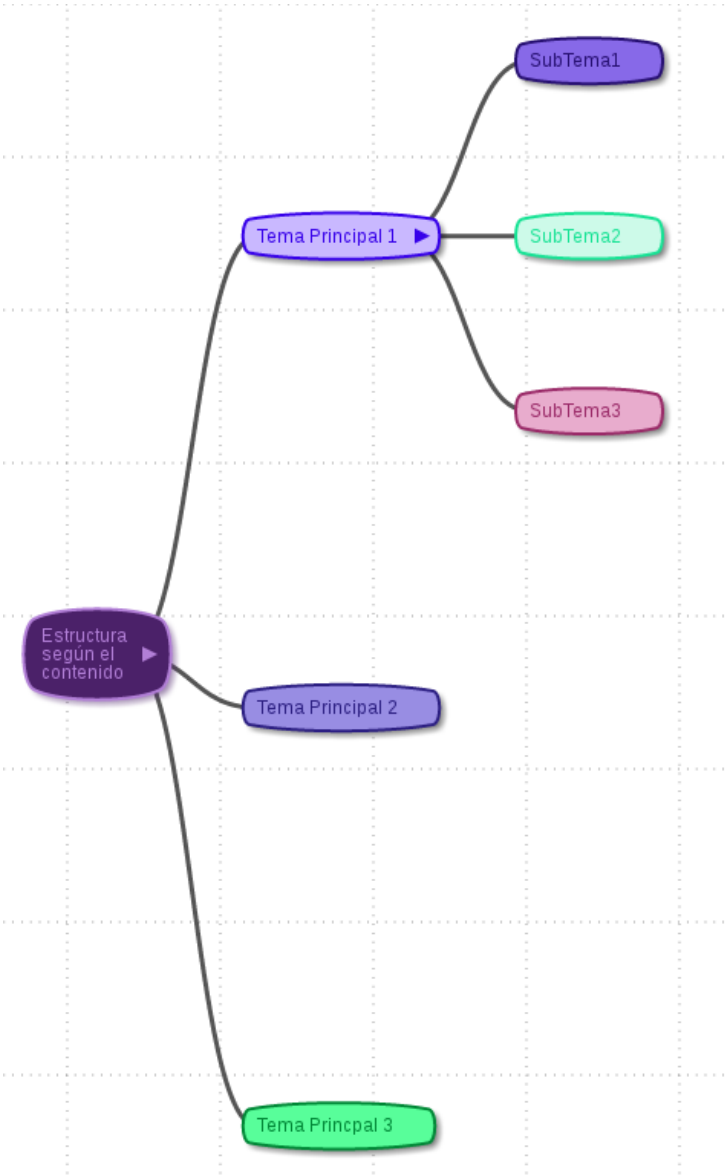


Figura 3.3: Partes de un mapa mental considerando su contenido

3.4. Elaboración.

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

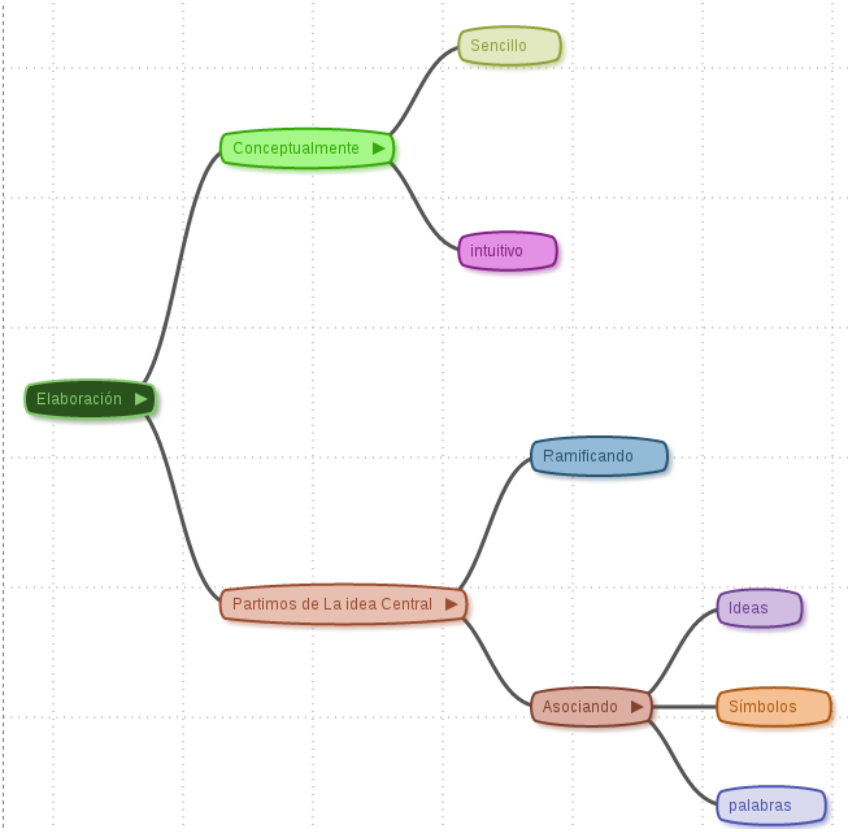


Figura 3.4: Mapa mental de elaboración de mapas mentales

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

Capítulo

4 Materiales y métodos

En Materiales y métodos se describen los pasos seguidos, las herramientas usadas, también el tiempo y orden del desarrollo. Exponer lo adecuado al trabajo de la metodología empleada así como los métodos en sí para su posible reproducibilidad.

4.1. Metodología de análisis (UML-WAE)

Como metodología de análisis utilizaré el estándar de facto UML.

4.1.1. UML

UML o Lenguaje de Modelado Unificado (Unified Modeling Language) fue creado para unificar las distintas técnicas que existían. Se trata de un lenguaje, principalmente, visual que nos permite visualizar, especificar, construir y documentar un sistema. Como lenguaje de modelado nos da las herramientas para especificar los métodos y/o procesos del sistema.

UML consta principalmente de dos puntos de vista. Un punto de vista estructural o estático en el cual se plasman estructuras estáticas compuesta por objetos, atributos, operaciones y relaciones. El otro punto de vista es dinámico o de comportamiento que

permite establecer y/o especificar las colaboraciones entre las distintas partes del sistema. Este último punto de vista, además nos proporciona estados internos de nuestro sistema.

Diagramas

La metodología UML nos dota de un conjunto de diagramas para modelar.

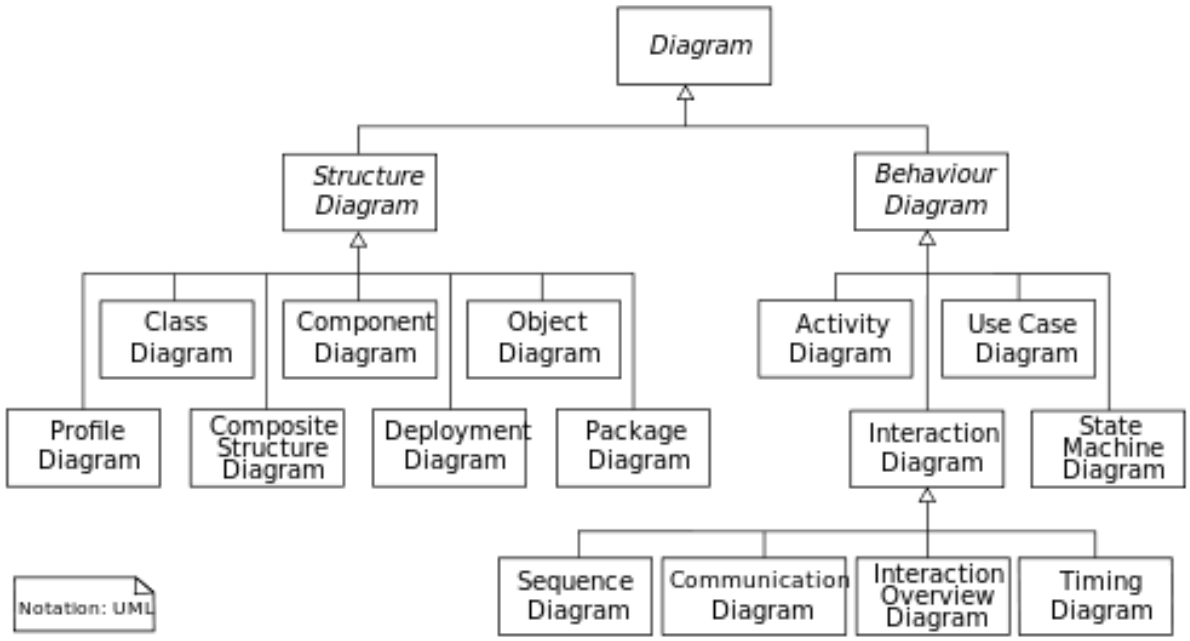


Figura 4.1: Esquema de diagramas

El conjunto de diagramas se divide en dos grandes grupos:

- Diagramas de estructura. Se corresponde con el punto de vista estático. Entre los diagramas más destacados están:
  - Diagrama de clases
  - Diagrama de componentes
  - Diagrama de paquete
- Diagramas de comportamiento. Se corresponde con el punto de vista dinámico. Entre los más utilizados están:
  - Diagrama de casos de usos
  - Diagrama de actividad

- Diagrama de secuencia

UML 2 es extensible. Los mecanismo para personalizar y extender el UML son los perfiles y los estereotipos. UML-WAE utiliza los dos.

UML-WAE

Hoy en día, la mayoría de las aplicaciones se desarrollan entorno a la web, es decir, son aquellas aplicaciones que en su arquitectura y elementos principales están el navegador y el protocolo HTTP. Fue Jim Conallen, en 1988, quien definió una extensión del estándar UML para aplicaciones webs. Esta extensión pasó a llamarse WAE (Web Application Extension) y se trata de la convención o extensión más aceptada por lo que podríamos a hablar de un estándar dentro de las distintas extensiones webs de UML.

En un principio las aplicaciones webs se basaban en un conjunto bien definidos de tecnologías. Con un servidor web, un cliente web capaz de comunicarse, vía protocolo HTTP, y de renderizar HTML. Con el tiempo se fueron sumando otras tecnologías como CSS, JavaScripts, XML, Ajax, Comet, webGL, etc ... Es fácil comprobar, que el desarrollo de una aplicación web, en principio sencilla se ha complicado con el paso del tiempo. De hecho, el propio HTML en principio sencillo y liviano se ha vuelto amplio y pesado. En el siguiente diagrama, podemos ver de forma esquemática un compendio del conjunto de tecnologías que se incorporan o que están en proceso de incorporación al HTML5.

Es evidente que tal conjunto de tecnologías y especificaciones llevan al desarrollador web a un sobre esfuerzo para estar al día. Pero también abre un sin fin de posibilidades que, ya hoy en día, podemos observar en la Internet.

WAE lo que nos ofrece, a los desarrolladores, es la posibilidad de modelar elementos como HTML, CSS, JavaScript, páginas de servidor, etc ... Para ello, hizo uso de los mecanismos de extensión del UML:

- Estereotipo: permite extender el vocabulario del UML, con el fin de crear nuevo elementos del modelo ya existente. Se representa de la entre ≪ estereotipo≫.

Figura 4.2: Estereotipos

- Valores etiquetados: se trata de un conjunto de pares clave-valor que puede ir asociado a un elemento del modelo. Por ejemplo, memoria="4Gb", nucleos=2. En la versión 1.3, los "valores etiquetados" quedaron obsoletos A partir de UML2.0 se descartaron los "valores etiquetados", en favor de los atributos. Por lo que, hoy en día se utilizan atributos para estos menesteres.

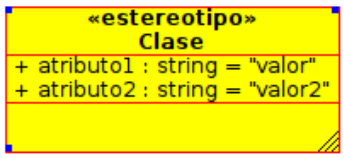


Figura 4.3: Valores etiquetados

- Restricciones: se trata de una etiqueta. Su misión realizar asertos y/o especificar restricciones a una relación entre objetos del modelo. Por ejemplo, A incluye B.

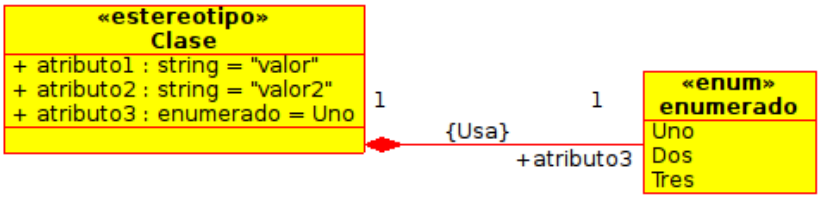


Figura 4.4: Estereotipos

Principales extensiones definidas por UML-WAE

Las principales extensiones que podemos encontrar dentro de la extensión UML-WAE son:

- ≪ Server page ≫: representá una instancia de páginas de servidor. Estas páginas se caracterizan por tener scripts y/o código que se ejecutan en el servidor. (.php, .asp, .jsp).

- ≪ Client page ≫: entidad que representa una una instancia de página que se renderiza en el cliente. Normalmente HTML y JavaScript.
- ≪ Form ≫: representa a la entidad de formulario de un documento HTML. Se trata de una colección de campos de formularios HTML.
- ≪ FrameSet ≫: representa el conjunto de marcos o frames que tiene una página.
- ≪ JavaScript ≫: entidad que se refiere a una clase o entidad JavaScript.
- ≪ Script Library ≫: componente de estereotipo Script Library representa una librería de funciones y/o rutinas que pueden ser incluidas en una página web.
- ≪ Web Page ≫: componente de estereotipo Web page.
- ≪ Submit ≫: una asociación de estereotipo submit son asociaciones de formularios HTML a otras páginas de clientes o servidor.
- ≪ Build ≫: asociación utilizada para representar la salida de una página de servidor en una instancia de página de cliente.
- ≪ Redirect ≫: sirve para especificar redirecciones realizada realizadas por cualquier página.
- ≪ Link ≫: representa un ancho o enlace de una página de cliente a otra.

4.2. Casos de uso.

En la figura 4.5 podemos ver de forma general el diagrama de casos de usos.

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.

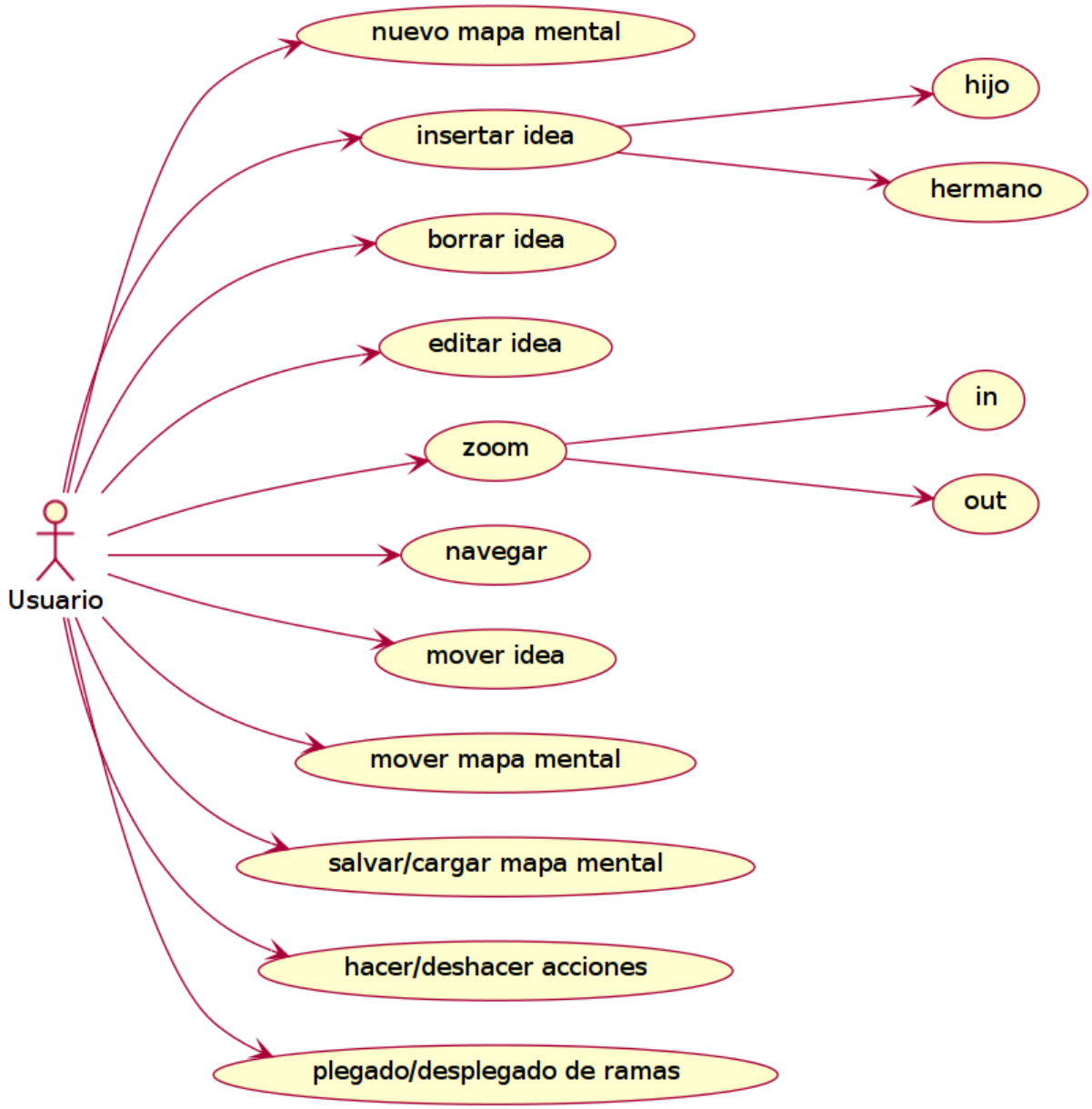


Figura 4.5: Casos de uso

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

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

4.2.4. Editar idea

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

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

El editor deberá y ajustándose al tamaño del texto insertado.

4.2.5. Zoom

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

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>

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.

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.

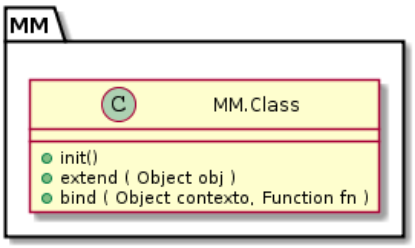


Figura 4.6: Clase MM.Class

Los principales métodos son:

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

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

4.3.1. Diagrama de clases PubSub

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

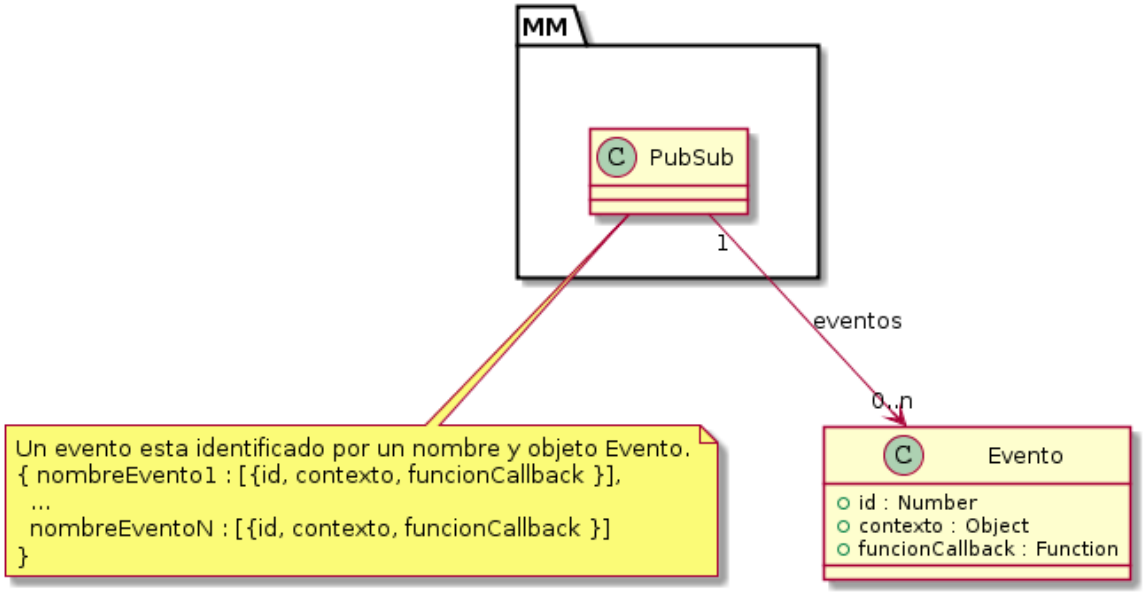


Figura 4.7: Diagrama de clases pubsub

El concepto es sencillo, el objeto suscriptor se suscribe a un evento o mensaje concreto y el publicador anuncia a todos los suscriptores cuando está lista las suscripción. El símil más utilizado, y directo, es el de los suscriptores de un periódico, el cual un lector (o suscriptor) paga un precio para recibir el periódico y la editorial (o publicador) le envía un ejemplar cuando lo tiene disponible.

[1] También conocido como patrón Observador-observable

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

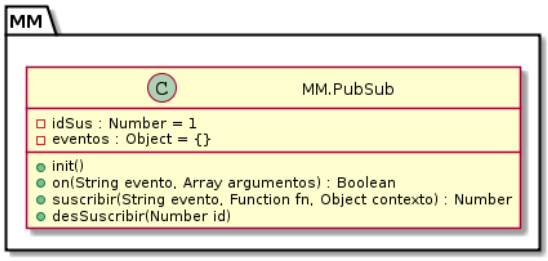


Figura 4.8: Clase MM.PubSub

Métodos:

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

4.3.2. Diagrama de clases MM.UndoManager

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

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

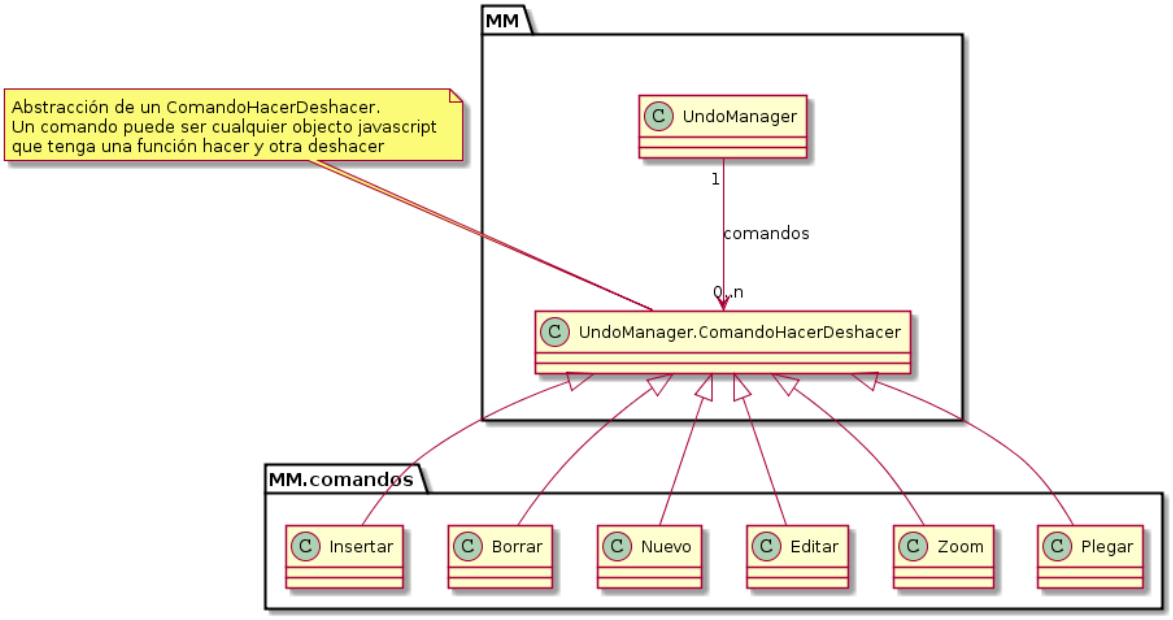


Figura 4.9: Diagrama de clases undo

observar en la figura 4.9, heredan clases para hacer y deshacer inserciones, borrados, zoom, etc ...

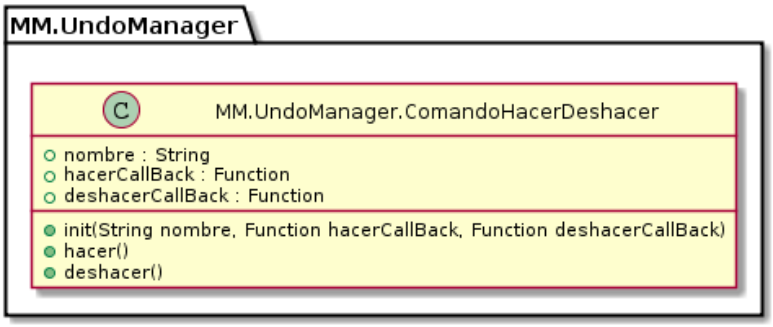


Figura 4.10: Clase MM.UndoManager.ComandoHacerDeshacer

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

Clase MM.UndoManager

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

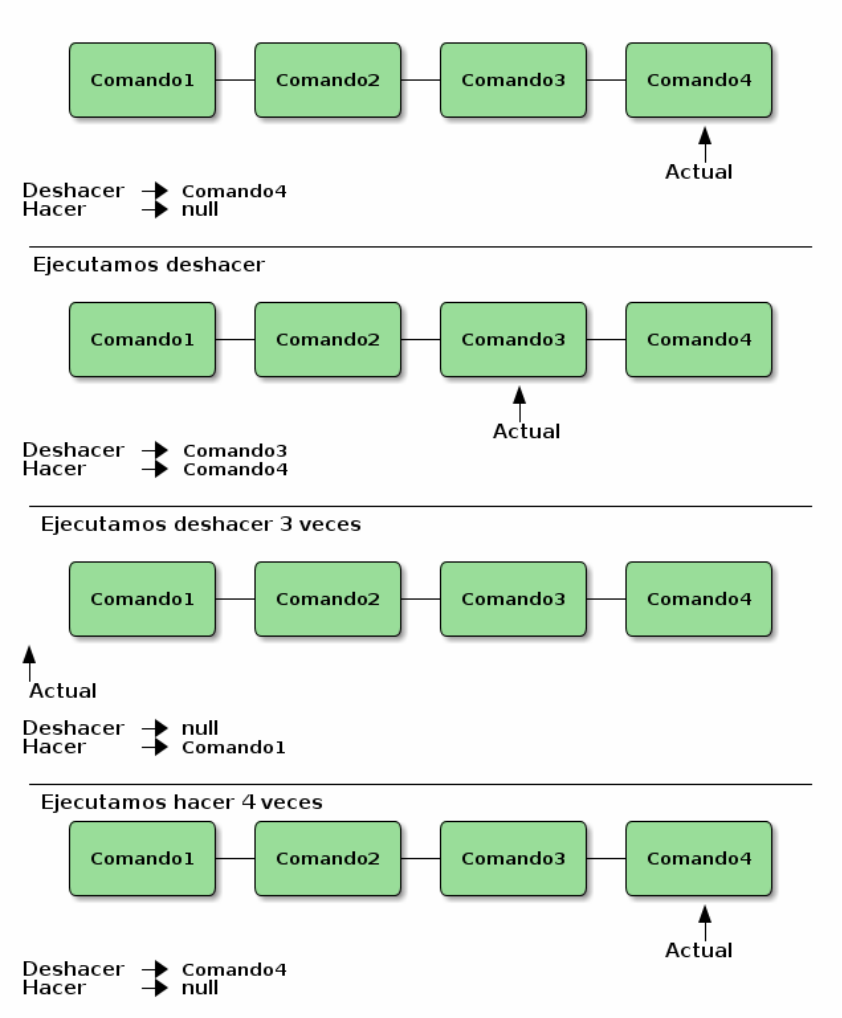


Figura 4.11: Secuencia de ejecución de UndoManager

Como puede observar en la figura 4.11 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

[2] Campo actual.

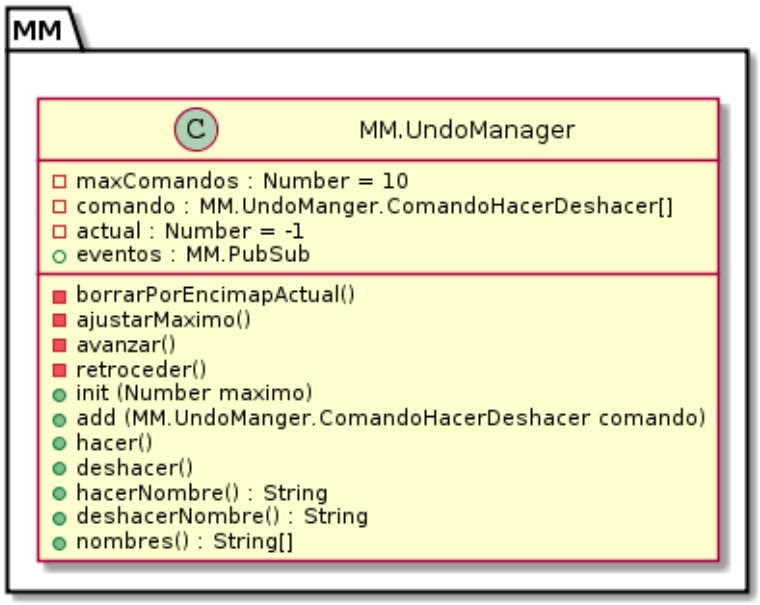


Figura 4.12: Clase MM.UndoManager

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

4.3.3. Diagrama de clases MM

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

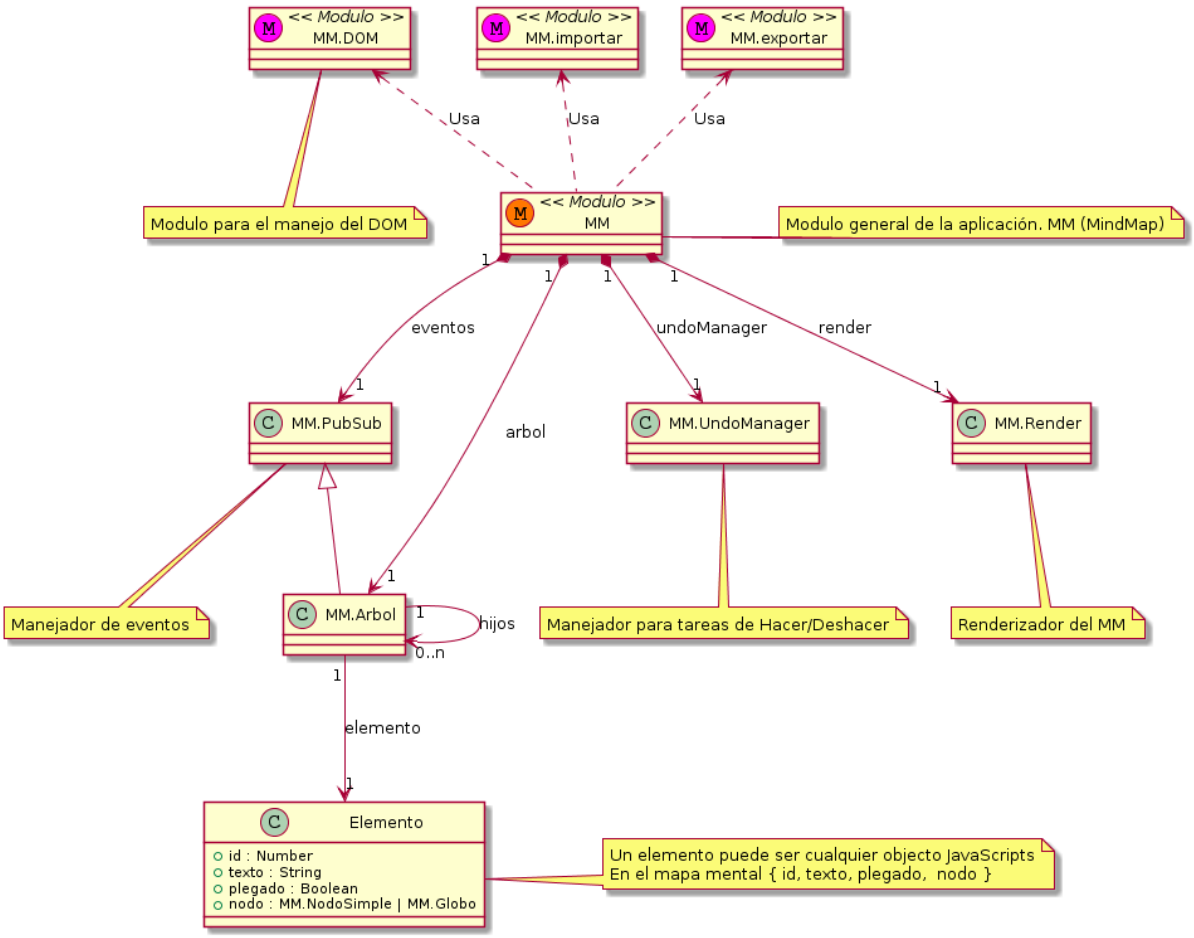


Figura 4.13: Diagrama de clases MM

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

- Mantener la coherencia de los datos.
- 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

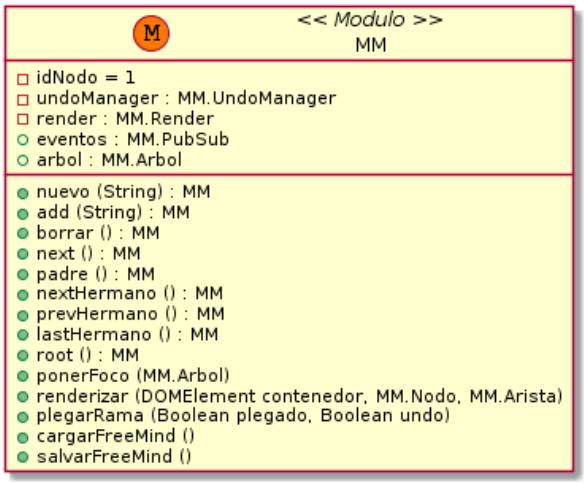


Figura 4.14: Clase MM

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.

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

- **MM.padre:** mueve el foco al padre del nodo activo.
- **MM.nextHermano:** mueve el foco al siguiente hermano del nodo activo.
- **MM.prevHermano:** mueve el foco al hermano anterior del nodo activo.
- **MM.lastHermano:** mueve el foco al último hermano del nodo activo.
- **MM.root:** mueve el foco al nodo raíz.
- **MM.ponerFoco:** establece el foco en nodo dado.
- **MM.renderizar:** se encarga de renderizar el mapa mental.
- **MM.plegarRama:** función para plegar y desplegar ramas.
- **MM.cargarFreeMind:** función de carga de ficheros FreeMind.
- **MM.salvarFreeMind:** se encarga de salvar el mapa mental en formato FreeMind.

Clase MM.Arbol

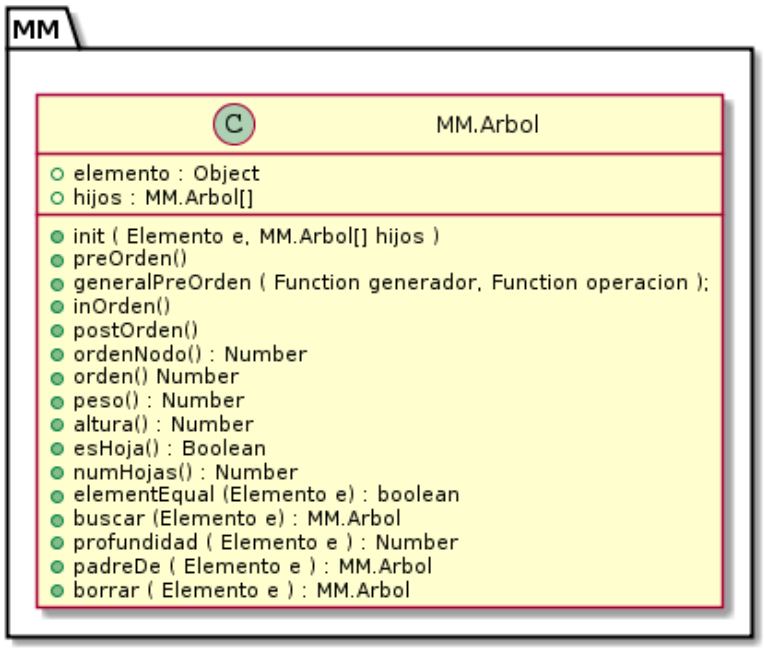


Figura 4.15: Clase MM.Arbol

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

- **MM.Arbol.init:** Crea un nuevo árbol-n con un elemento raíz y array de árboles hijos.
- **MM.Arbol.preOrden:** realiza un recorrido en preorden por el árbol.
- **MM.Arbol.generalPreOrden:** recorrido en preorden, con un generador que trata el elemento actual y una operación que se encarga de operar el elemento generado con el preorden de los nodos hijos.
- **MM.Arbol.inOrden:** realiza un recorrido inorden por los elementos del árbol.
- **MM.Arbol.postOrden:** recorre el árbol el postorden.
- **MM.Arbol.ordenNodo:** calcula el orden del nodo actual.
- **MM.Arbol.orden:** calcula el orden del árbol completo.
- **MM.Arbol.peso:** calcula el peso de un árbol.
- **MM.Arbol.altura:** altura del árbol.
- **MM.Arbol.esHoja:** indica si el nodo actual es un nodo hoja o no.
- **MM.Arbol.numHojas:** determina el número de nodos hojas del árbol.
- **MM.Arbol.elementEqual:** función de igual entre elementos de los nodos. Por defecto, es la igual estricta -==". Esta función podrá ser sobreescrita para adecuarse al tipo de elemento guardado en cada nodo.
- **MM.Arbol.buscar:** busca un elemento en el árbol. Como comparador de nodos se utiliza la función MM.Arbol.elementEqual.
- **MM.Arbol.profundidad:** determina la profundidad del árbol.
- **MM.Arbol.padreDe:** calcula el árbol padre del elemento pasado.
- **MM.Arbol.borrar:** borra un elemento del árbol.

Módulo MM.DOM

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

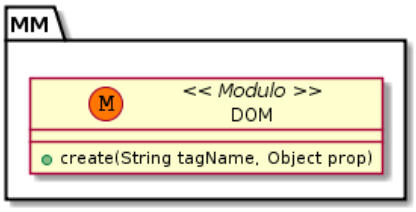


Figura 4.16: Modulo MM.DOM

Clase MM.Render

La clase MM.Render es la encargada de pintar el mapa mental y realizar los ajustes visuales necesarios para mostrar los nodos y las aristas. El renderizador se configura o inicializa entorno a un elemento DOM, normalmente un *div*, una clase que MM.NodoSimple[4] y una clase MM.Arista[5]. A partir de estos datos el sistema es capaz de ir generando el mapa mental en función de los eventos producidos en el módulo MM.

Figura 4.17: Modulo MM.Render

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

La clase MM.Render dispone de los siguientes métodos:

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

4.3.4. Diagrama de clases nodo.

El nodo se encarga del pintado de una idea del mapa mental, en esencia, es un MM.Mensaje al cual se le han añadido otros elementos gráficos y funcionalidades. Existen dos implementaciones de nodo, como se pueden ver en el diagrama4.18, el MM.NodoSimple y el MM.Globo, y ambos pueden ser usados desde MM.Render. Todos los nodos existen un escenario y en una capa dada.

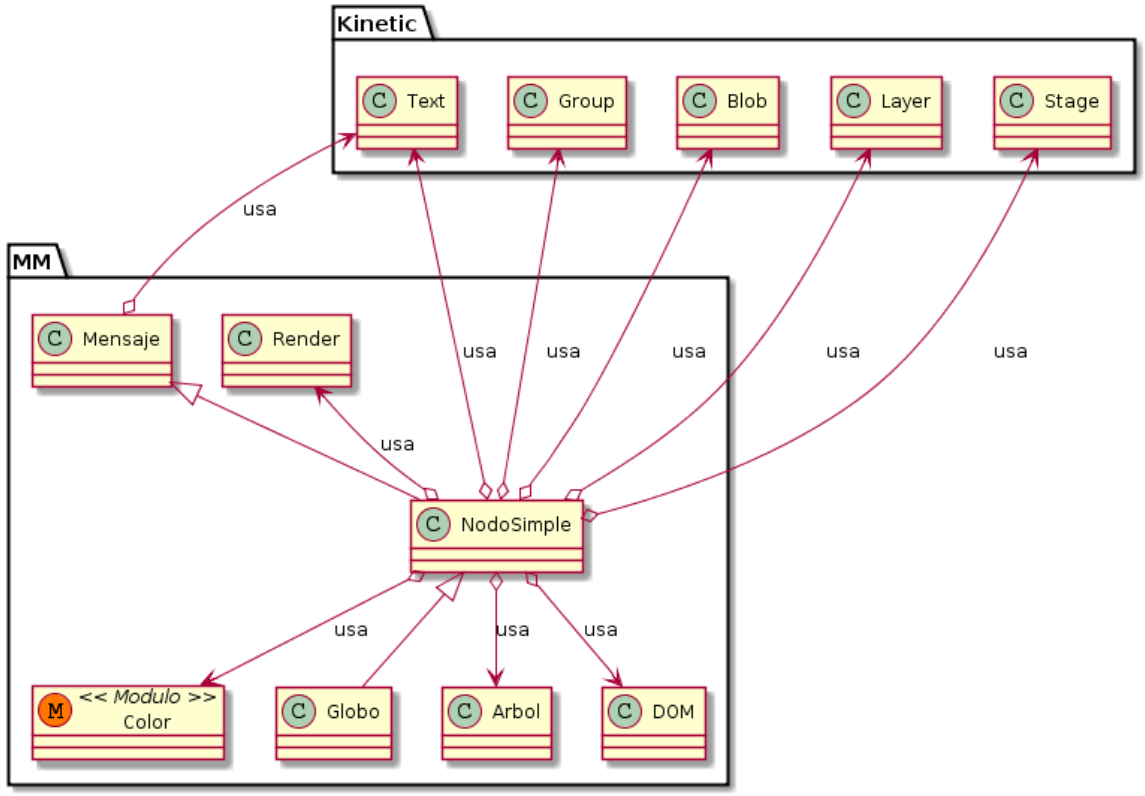


Figura 4.18: Diagrama de clases nodo

Clase MM.Mensaje.

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

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

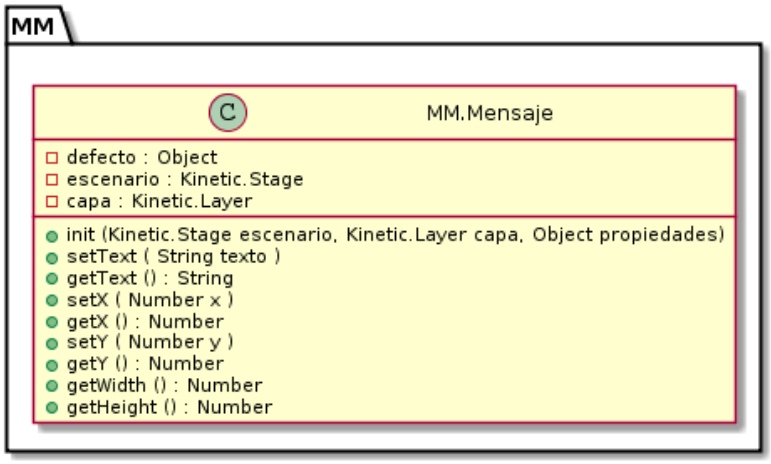


Figura 4.19: Clase MM.Mensaje

- **MM.Mensaje.getX/setX:** métodos para establecer y obtener la posición[6] X del mensaje.
- **MM.Mensaje.getY/setY:** métodos para establecer y obtener la posición Y[7] del mensaje.
- **MM.Mensaje.getWidth:** devuelve el ancho del texto en píxeles.
- **MM.Mensaje.getHeight:** devuelve el alto del texto en píxeles.

Clase MM.NodoSimple.

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

- **MM.NodoSimple.init:** constructor de la clase. Recibe el MM.Render, la idea a la que representa y un conjunto de propiedades como la posición, escala, etc ...
- **MM.ponerFoco/quitarFoco:** métodos que poner o quitan el foco en la idea que representa el nodo. Debe resaltar el nodo cuando este esté focalizado.

[6] en píxeles
[7] en píxeles

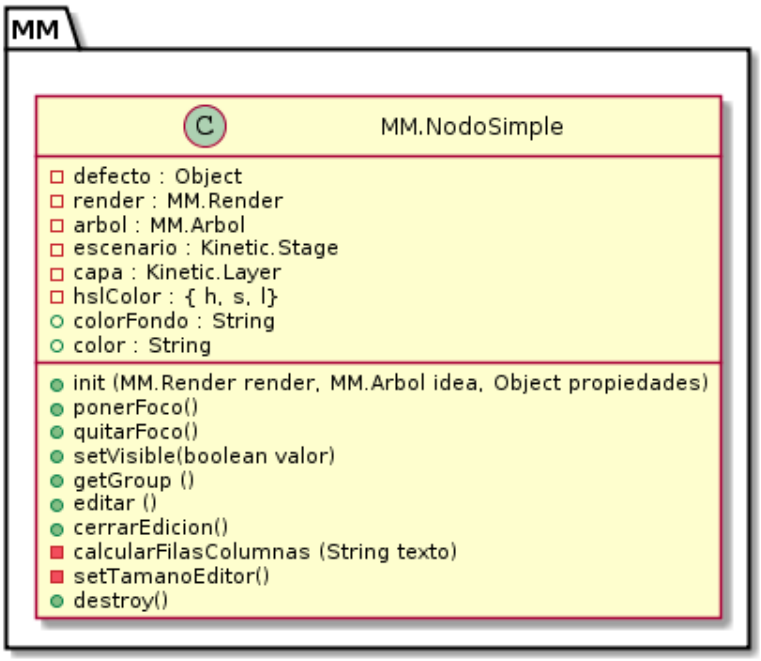


Figura 4.20: Clase MM.NodoSimple

- **MM.Mensaje.editar/cerrarEdicion:** métodos para establecer la idea en modo edición y para cerrarlo cuando se termine la edición. Si esta en modo edición debe tener el foco.
- **MM.Mensaje.setVisible:** indica si el mensaje debe mostrarse o no.
- **MM.Mensaje.destroy:** borra y destruye el nodo.

Clase MM.Globo.

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

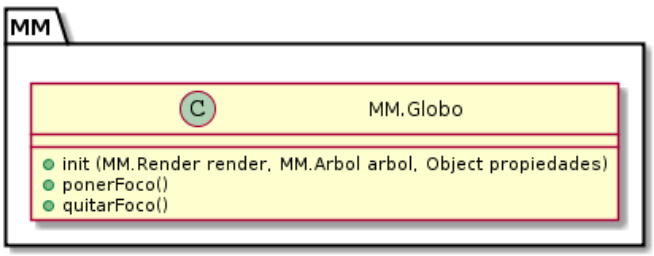


Figura 4.21: Clase MM.Globo

Módulo MM.Color.

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

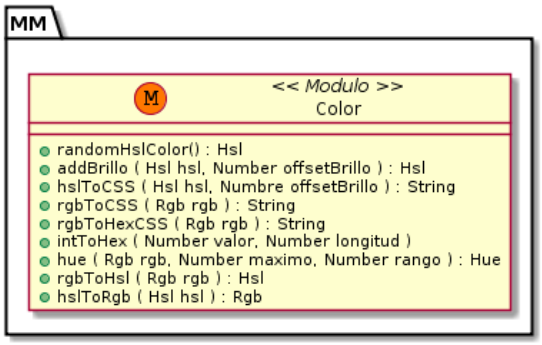


Figura 4.22: Modulo MM.Color

4.3.5. Diagrama de clases de aristas.

Una arista representa la línea de unión entre dos ideas o nodos. Existen dos tipos de aristas MM.Arista y MM.Rama, ambas tienen dos nodos a los deben unir. Las aristas, han sido implementadas con una curva Beizer. El diagrama de clases de aristas podemos ver lo en la figura 4.23.

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.

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.

[8] HSL, RGB y HUE

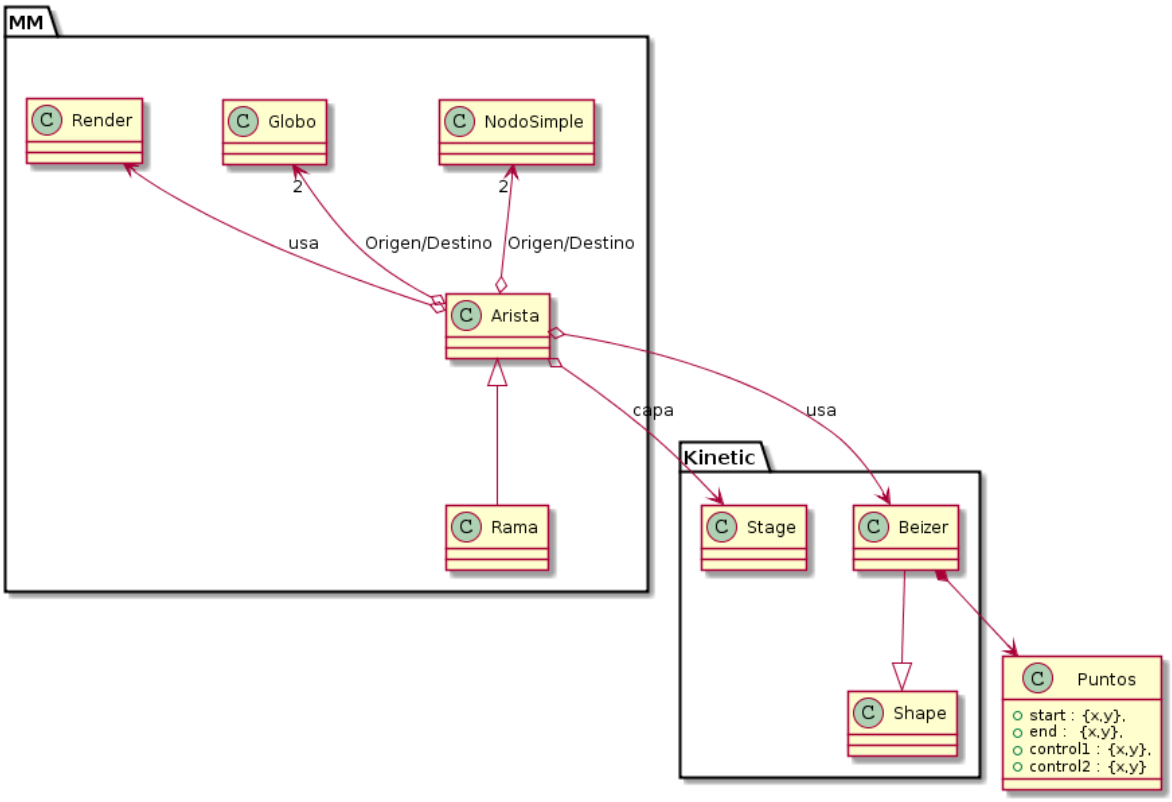


Figura 4.23: Diagrama de clases aristas

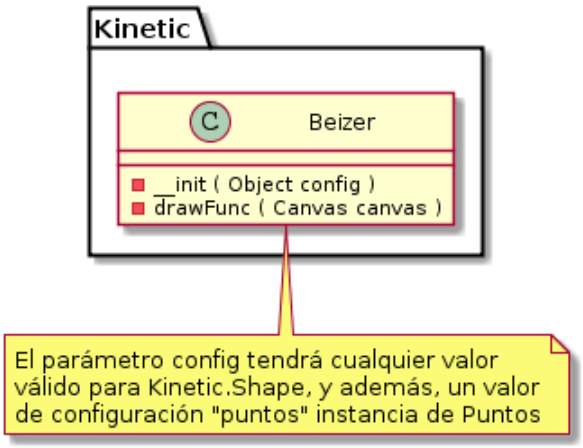


Figura 4.24: Clase Kinetic Beizer

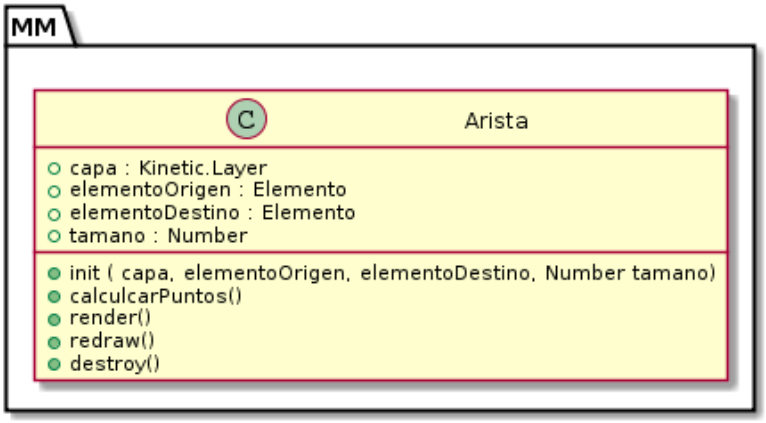


Figura 4.25: Clase MM.Arista

- **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.

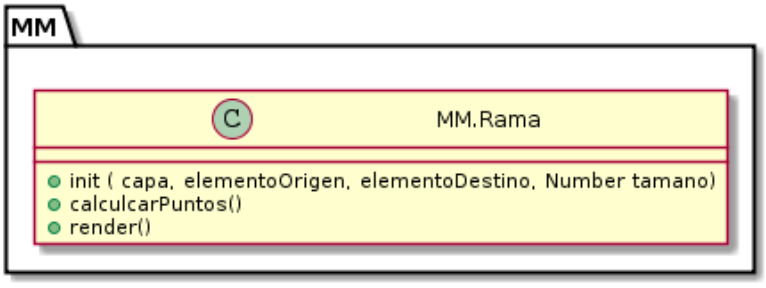


Figura 4.26: Clase MM.Rama

4.3.7. Diagrama de clases de teclado

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

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

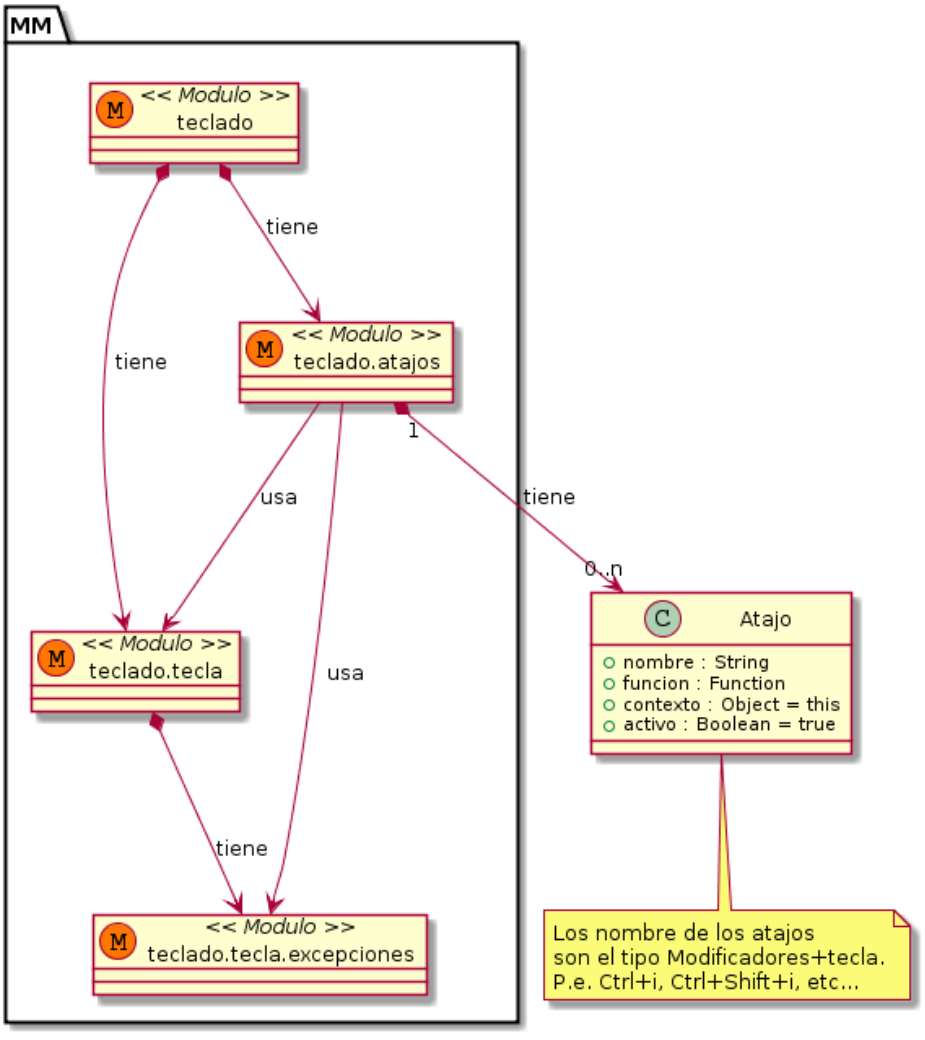


Figura 4.27: Diagrama de clases teclado

[9] En los sistemas Mac.

Módulo MM.teclado.atajos

Un atajo esta compuesto por un nombre[10], una función que será ejecutada

El modulo de atajos se encarga de registrar las secuencias de teclado y lanzarlos cuando sea necesario.

pueden desactivarse y activarse

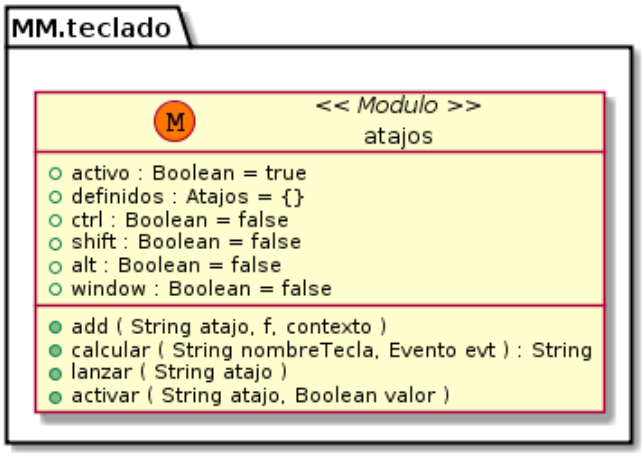


Figura 4.28: Modulo MM.teclado.atajos

- **MM.teclado.atajo** 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.

4.4. Diagramas de secuencia.

[10] Ctrl+i

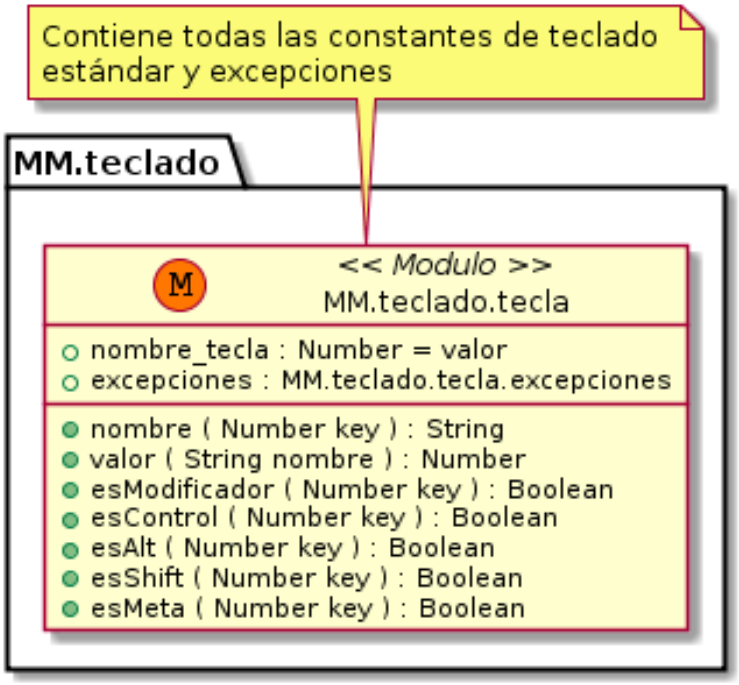


Figura 4.29: Clase MM.teclado.tecla

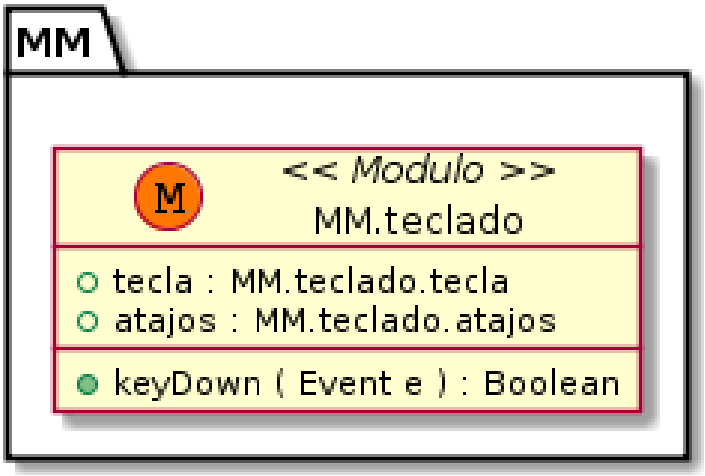


Figura 4.30: Clase MM.teclado

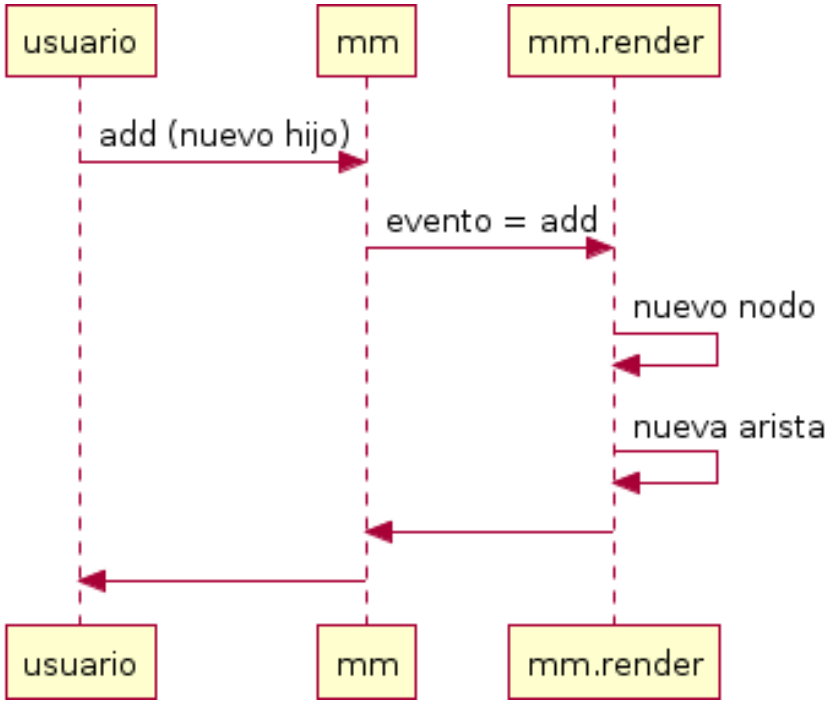


Figura 4.31: Diagrama de secuencia add

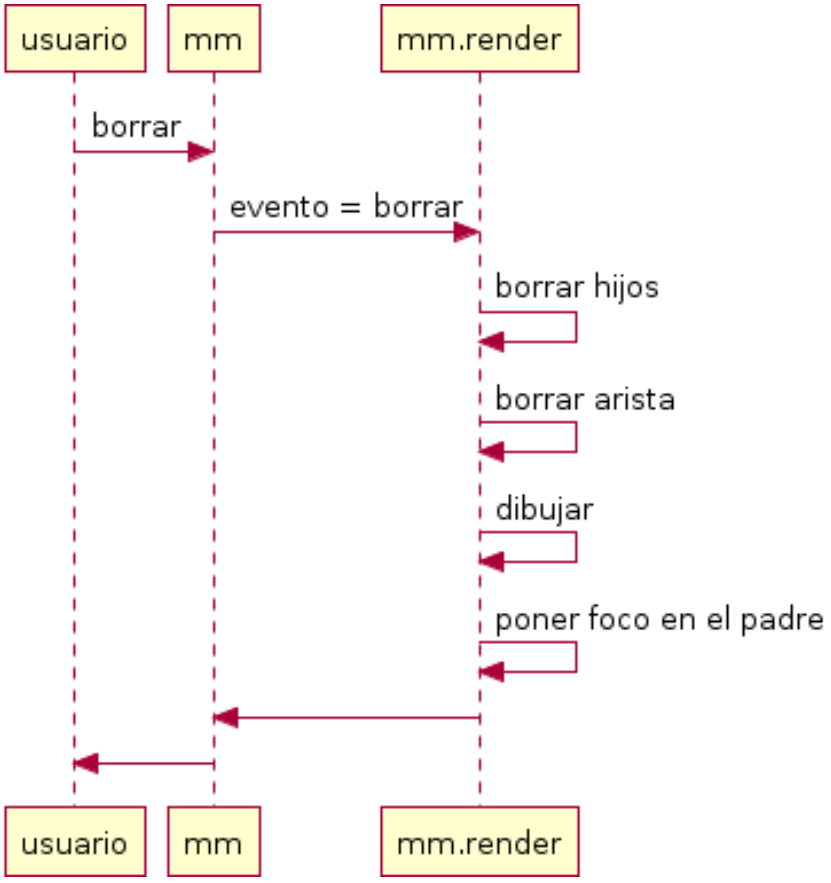
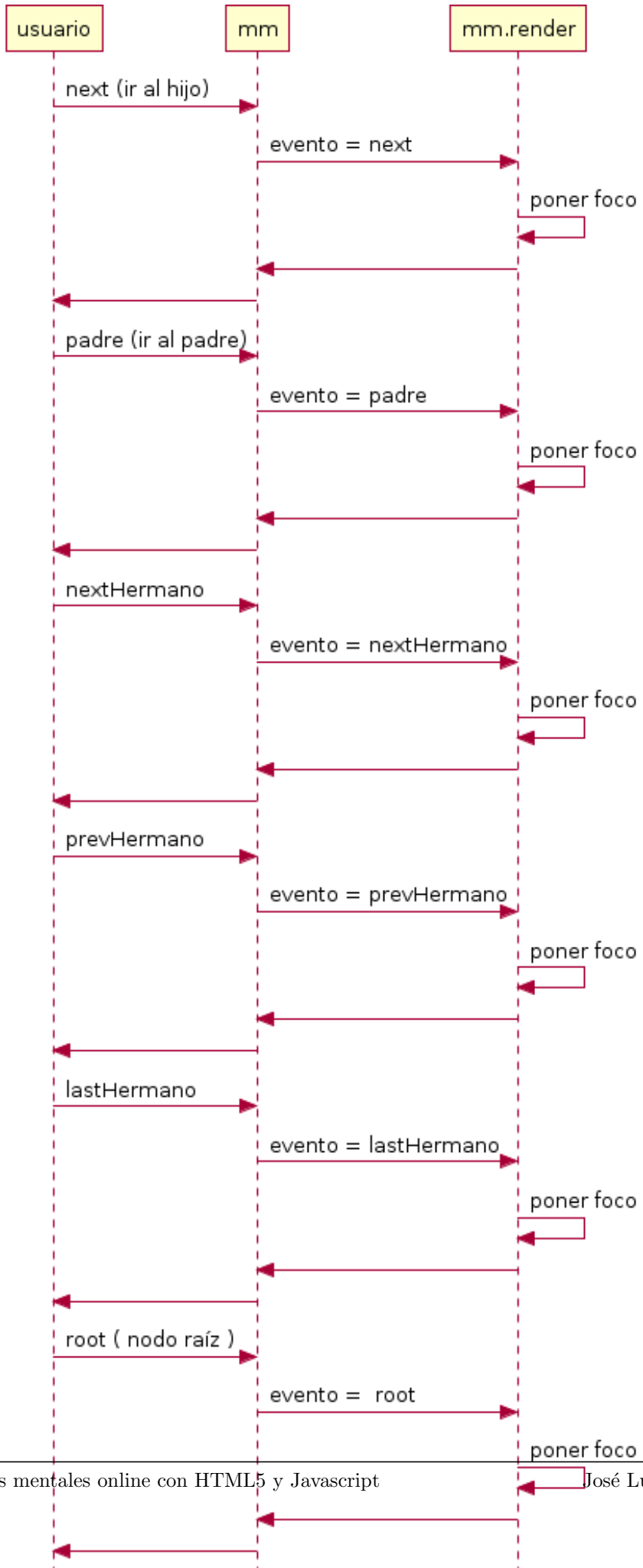


Figura 4.32: Diagrama de secuencia borrar


usuario
mm
mm.render
next (ir al hijo)
evento = next
poner foco
padre (ir al padre)
evento = padre
poner foco
nextHermano
evento = nextHermano
poner foco
prevHermano
evento = prevHermano
poner foco
lastHermano
evento = lastHermano
poner foco
root (nodo raíz)
evento = root
poner foco

Capítulo

5 Implementación.

La implementación describe los detalles más difíciles, las soluciones elegidas, y presenta esquemas y también detalles relevantes de programas, algoritmos y métodos de trabajo y tareas de campo desarrollados.

5.1. Metodología y etapas del desarrollo.

5.1.1. Metodología de desarrollo ágil.

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

- Individuos e interacciones sobre procesos y herramientas
- Software funcionando sobre documentación extensiva

- Colaboración con el cliente sobre negociación contractual
- Respuesta ante el cambio sobre seguir un plan

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

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

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

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

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

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

- Documentación.

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

5.1.2. Etapas del desarrollo.

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

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

- **Librería base con soporte para herencia.** Esta librería debe tener toda la funcionalidad básica (bindings, curryings, etc) y debe estar muy optimizada ya que el perfecto funcionamiento de la aplicación dependerá en buena medida de ella.
- **Librería para manejo de árboles n-arios.**
- **Librería para el manejo de ficheros.** Será la encargada de manejar ficheros, a partir de ella, realizaremos las clases de exportación e importación de mapas mentales de la aplicación.
- **Librería gráfica.** Ciñéndonos al contexto 2D, necesitamos un wrapper sobre la librerías propias del canvas. Esta librería, nos debe permitir pintar, cada uno de los elementos de nuestro árbol. Además de configurar, atributos visuales tales como color del trazo, relleno, etc. No se descarta el uso de alguna librería estándar. Para ello, se realizará una pruebas de concepto sobre ellas.

- **Librería para el manejo de eventos del canvas.** El canvas debe reaccionar tanto al teclado, ratón y touch. El canvas viene desprovisto de eventos sobre los elementos pintados en él y es aquí donde entra en juego esta librería.
- Por último, las librerías propias del mapa y **prototipo** o primera versión.

Segunda fase: una vez implementadas todas las librerías necesarias y una primera versión (inoperativa), nos encontramos en disposición de ir elaborando la aplicación. En esta fase, desde mi punto de vista, el modelo iterativo incremental se adapta a la perfección al proceso de refinamiento de la aplicación. Cada iteración con llevará la implementación de la funcionalidad, en la que nos enfoquemos, y su posterior evaluación y prueba.

Capítulo

6 Resultados y discusión

6.1. Resultados

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

6.2. Discusión

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

6.3. uglify

6.4. jsHint

6.5. KineticJS

6.6. NodeJS

Basado en la máquina virtual JavaScript V8 de Google, NodeJS[1] 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[2] para pruebas y depuración interactiva hasta un gestor de paquetes y librerías NPM[3] (Node Packaged Modules).

Figura 6.1: Logo NodeJS

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

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

Sus principales objetivos son:

- Escribir aplicaciones eficientes en entrada y salida con un lenguaje dinámico.

[1] La web oficial de NodeJS es nodejs.org
[2] Patrón Read-Eval-Print Loop
[3] La web oficial de NPM es npmjs.org

- Soporte a miles de conexiones.
- Evitar las complicaciones de la programación paralela (Concurrencia vs paralelismo).
- Aplicaciones basadas en eventos y callbacks.

6.6.1. 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[4].

Lo primero que tenemos debemos hacer es clonar el proyecto.

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

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

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

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

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

6.6.2. Instalación del NPM

Como ya se ha comentado antes NPM[5] 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
```

6.6.3. Uso básico de NPM

Iniciar un proyecto nuevo

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

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

```
$ 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.

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

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

Is this ok? (yes)
```

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

Buscar paquetes y obtener información

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

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

Instalación de paquetes

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

De forma global [6] para que lo puedan utilizar todas las librerías del sistema.

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

[6] Con el modificador -g

De forma local[7], 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 modificador es *–save-dev* para que la dependencia sea de desarrollo. Así quedaría un fichero package.json después de haber incluido un paquete (colors) como dependencia y otro (grunt-cli) como dependencia de desarrollo.

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

Desinstalación de paquetes

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

6.7. 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.

[7] Sin el modificador -g

6.7.1. 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.

6.7.2. Instalación

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

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

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

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

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

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

Figura 6.2: Logo GruntJS

```
{
  "name": "nombre",
  "version": "0.0.1",
  "dependencies": {

  },
  "devDependencies": {
    "grunt": "~0.4.1"
  }
}
```

6.7.3. Creando el Gruntfile

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

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

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

};
```

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

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

6.7.4. Gruntfile.js de MindMapJS

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

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

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

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

```
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 JosÃ© Luis Molina Soria */\n'
47        },
48        build: {
49      files: {
50          'dist/<%= pkg.name %>-v<%= pkg.version %>.min.js': 'dist/<%= pkg.name %>-v<%= pkg
              .version %>.js'
51      }
52        }
53    },
54
55    clean: {
56        build: ['dist/*']
57    },
58
59    jshint: {
60        options: {
61      laxbreak: true,
62      curly: true,
63      eqnull: true,
64      eqeqeq: true,
65      undef: true,
66      browser: true,
67  //    immed: true,
68      latedef: true,
69      newcap: true,
70      noarg: true,
71      sub: true,
72      boss: true,
73      globals: {
74          console: true,
75          window : true,
76          module: true,
77          MM : true,
78          Kinetic : true,
79          require : true,
80          ActiveXObject : true,
81          FileReader : true,
82          DOMParser : true,
83          Blob : true,
84          alert : true
85      }
86        },
87        all : ['src/*.js']
88    },
89
90    jsdoc : {
91              dist : {
92      src: ['src/*.js'],
93      options: {
94                      destination: 'docs/jsdocs/'
95      }
96              }
97    },
98
99    mochaTest: {
100       test: {
101     options: {
```

```
102         reporter: 'spec',
103         require: 'should'
104     },
105     src: ['test/**/*-test.js']
106       }
107   }
108
109     };
110
111     grunt.initConfig(config);
112
113     // Load plugins
114     grunt.loadNpmTasks('grunt-contrib-concat');
115     grunt.loadNpmTasks('grunt-replace');
116     grunt.loadNpmTasks('grunt-contrib-uglify');
117     grunt.loadNpmTasks('grunt-contrib-clean');
118     grunt.loadNpmTasks('grunt-contrib-jshint');
119     grunt.loadNpmTasks('grunt-jsdoc');
120     grunt.loadNpmTasks('grunt-mocha-test');
121
122     // Tasks
123     grunt.registerTask('dev', ['clean', 'concat:source', 'replace:dev']);
124     grunt.registerTask('full', ['clean', 'concat:source', 'replace:dev', 'uglify', '
            replace:prod']);
125     grunt.registerTask('test', ['mochaTest']);
126     grunt.registerTask('hint', ['jshint']);
127     grunt.registerTask('jsdoc', ['jsdoc']); // no funca :(( utilizar el script "jsdoc.sh"
128 };
```

Como se puede comprobar se han incorporados distintos plugins:

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

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

- **dev:** que concatena el código fuente y realiza los reemplazo como fechas, versión, etc ...

- **full:** además de realizar las tareas propias de la tarea 'dev', minimiza y realiza los reemplazos de producción.
- **test:** lanza la suite de test
- **hint:** lanza la terea de validación de código JSHint.
- **jsdoc:** genera la documentación del API.

6.8. Github

Todo proyecto que se precie debe estar sustentado con sistema de control de versiones, en nuestro caso ha sido Git[8]. Más concretamente se trata de un sistema distribuido de control de código fuente o SCM[9] creado por Linus Torvalds, a partir, de su propia experiencia en el desarrollo de los kernels de Linux.

Github[10] es una plataforma online pensada para el desarrollo colaborativo de proyectos, utilizando para ello Git. Github nos permite almacenar de forma pública[11] nuestro código fuente, promoviendo el trabajo colaborativo entre profesionales. Así pues, otro profesional ajeno al proyecto puede solicitar cambios sugerir mejoras o reportar bugs.

Figura 6.3: Mascota de Github

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

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

[8] Web oficial de Git es git-scm.com
[9] SCM (Source Code Management)
[10] La web de Github es github.com
[11] Github permite crear proyectos privados con cuentas de pago

- **Página web del proyecto**, para presentar nuestro poryecto y/o repositorio

Como sistemas de colaboración entre programadores tenemos el:

- **Fork**, con un fork podemos clonar un repositorio para realizar cambios que necesitemos, de forma que podamos adaptar el proyecto a nuestras necesidades concretas. Un fork nos permite colaborar con el proyecto original mediantes los pull requests.
- **Pull requests**, una vez realizados los cambios, y si lo vemos oportuno, podemos reportar las variaciones al proyecto original mediante un pull request. El pull request pueden ser cambios, mejoras en la funcionalidad, y/o correcciones, que deberá aprobar él/los programadores del proyecto original.

6.8.1. 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[12] en el cual hay que introducir un descripción del proyecto. Este fichero se visualizará en le página principal del repositorio.
- Añadimos y confirmamos los cambios.

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

```
$ 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
```

6.8.2. Fork/Pull request

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

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

Ya tenemos el repositorio listo para su uso. Si deseamos colaborar con el proyecto original debemos crear una rama[13], realizar los cambios y subirlos[14] a nuestro fork de Github. Desde Github procede realizar la revisión de los cambios y pulsar sobre la opción de çreate a pull request for this comparison".

6.9. JSDoc

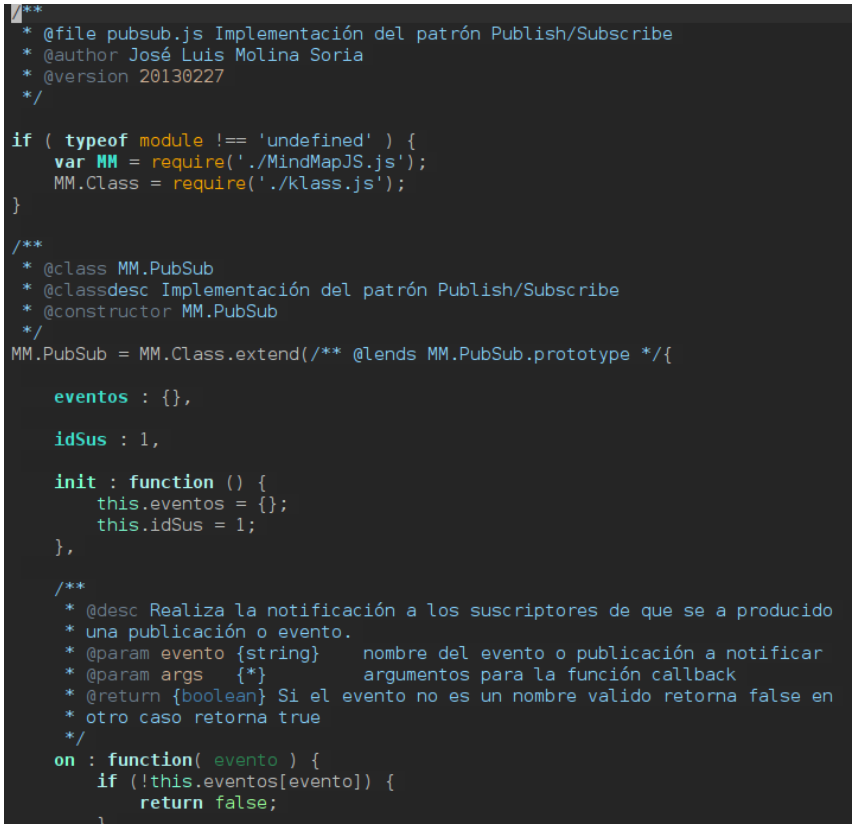
Tan importante como el código es la documentación del mismo, JSDoc[15] es una herramienta inspirada en Javadoc[16] pero pensada para Javascript.

Mediante una conjunto de etiquetas (@class, @function, etc) introducidas como comentarios del código fuente, se generará la documentación en formato HTML[17]. Todos los desarrolladores que alguna vez hemos programado en Java y generado documentación

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

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 6.4 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 ...



```
/**
 * @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 6.4: Ejemplo de código fuente documentado con JSDoc

En la figura 6.5 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 funcional, de la documentación de nuestro cógido fuente.

6.10. Mocha

Mocha[18] es un framework Javascript para realizar pruebas unitarias. Sus creadores lo definen como:

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

Figura 6.5: Página generada por JSDoc

> *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"[19].

6.10.1. 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.

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

- **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.

6.10.2. Ejemplo

He aquí un simple ejemplo de uso de mocha.

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

Figura 6.6: Mocha

Para empezar, debe realizar importar la librería de afirmaciones (assertions) que vamos a utilizar en el presente ejemplo se ha utilizado assert[20].

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

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

Una vez escrita la prueba unitaria ejecutamos el siguiente comando "mocha -R *.js" obtenemos el resultado que podemos observar en la figura 6.7.

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.

[20] En MindMapJS se ha utilizado Should y Chai

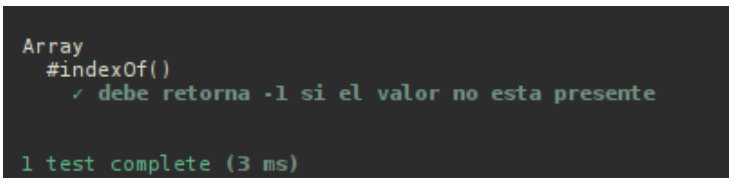


Figura 6.7: Resultado de ejecutar mocha -R spec *.js

```
1  var assert = require("assert");
2
3  var a;
4
5  beforeEach(function(){
6     a = [1,2,3];
7  });
8
9
10 describe('Array', function(){
11   describe('#indexOf()', function(){
12     it('debe retorna -1 si el valor no esta presente', function(){
13       assert.equal(-1, a.indexOf(5));
14       assert.equal(-1, a.indexOf(0));
15     });
16     it('debe retorna 1 si pedimos al primer valor', function(){
17       assert.equal(0, a.indexOf(1));
18     });
19   });
20 });
```

El resultado de ejecutar nuestro nuevo test es.

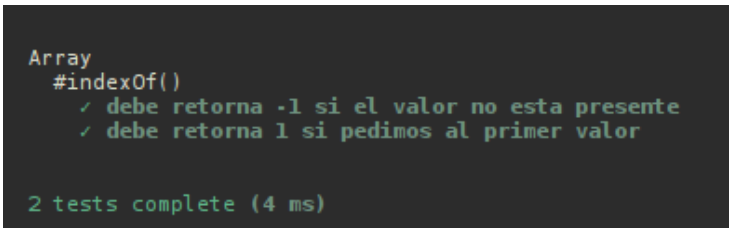


Figura 6.8: Resultado de ejecutar mocha -R spec array1-test.js

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

Capítulo

7

Manual de usuario

Capítulo

8

Conclusiones

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

Bibliografía