Apéndice A

Manuales

A.1. Manual de instalación

A.1.1. Prerrequisitos

Se requiere contar con:

- Windows 10, arquitectura de 64 bits. [1]
- Java (versión 8 o superior) JRE. No es necesario el JDK.
- En el caso de utilizar *tags* en `gmaster`, se debe contar con la versión 0.9.289 o superior de `gmaster` [16].
- En el caso de que se desee que para las diferencias entre versiones de archivos Excel, `gmaster` lance SpreadSheet Compare tool, se debe contar con la versión 1.0.663 o superior de `gmaster` [16].
- Los archivos Vensim a procesar deben estar codificados en el formato de espaciado propio de Windows, es decir, `CRLF`.

(TEMPORAL)Se debe tener en cuenta que si se va a utilizar Vensim, (OJO probar que funciona a partir de la 7.1), se deberá tener cuidado no manipular con la distribución DSS archivos creados con la distribución PLE, y viceversa.

[1] Para comprobar la arquitectura del sistema: `https://www.computerhope.com/issues/ch001121.htm#:~:text=Press%20and%20hold%20the%20Windows,running%20the%2064%2Dbit%20version.`

A.1.2. Descarga del software

La solución desarrollada consta de tres archivos `.jar` que deberán descargarse de:

`http://gitlab-locomotion.infor.uva.es/testing-locomotion-infrastructure/semanticmerge/-/tree/Interfaces/Executables`.

Estos archivos son:

- `AddViewNamesAndDelimiters.jar`
- `EraseViewNamesAndDelimiters.jar`
- `VensimPlugin4SemanticMerge.jar`

Los dos primeros son aplicaciones auxiliares que se explicarán más adelante, necesarias para la solución completa. El tercero es el archivo principal de la solución desarrollada para facilitar el trabajo con archivos `Vensim` en el control de versiones.

Se debe crear una carpeta para guardar estos tres archivos. En este manual asumiremos que dicha carpeta es `C:\Users\{User}\VensimPlugin4SM`, donde `{User}` debe ser sustituido por el nombre del usuario en Windows. En el caso de guardar los archivos en otra carpeta, se recomienda altamente que la ruta absoluta de dicha carpeta no contenga espacios en blanco y/o caracteres especiales.

A.1.3. Interacciones

Este proyecto no trabaja por sí solo (*standalone*) sino integrado como *plugin* de otras herramientas.

Para poder utilizarlo el usuario necesitará hacerlo a través de alguna de las siguientes aplicaciones: `SemanticMerge` [40] o `PlasticSCM` [45] o `gmaster` [34]. La aplicación elegida deberá estar instalada previamente a los siguientes pasos de la instalación del proyecto. Para una guía de instalación de alguna de estas aplicaciones debe dirigirse a las páginas indicadas en las referencias.

La primera de estas tres, `SemanticMerge`, es una herramienta independiente del control de versiones. La funcionalidad de SemanticMerge está integrada tanto en `PlasticSCM` como en `gmaster`, y no es necesario descargarla por separado si ya se va usar una de estas dos.

A.1.4. Configuración para el uso con SemanticMerge

Si el producto desarrollado se va a utilizar con `SemanticMerge`, se debe tener en cuenta que se trata de un software con licencia. Se debe adquirir o solicitar una clave de acceso, o bien, utilizar la prueba gratuita de 30 días.

Para utilizar herramientas externas utilizando puramente la aplicación de `SemanticMerge`, se debe utilizar la línea de comandos de Windows, a través de las aplicaciones de `CMD`, `Powershell` o cualquier otra aplicación externa que emule una terminal.

El comando a utilizar debe seguir la estructura que se muestra en el siguiente cuadro de texto, indicando la ruta al ejecutable de la herramienta `SemanticMerge`, la ruta a sendos archivos a comparar, la ruta a la herramienta externa desarrollada en este proyecto, y la ruta a la máquina virtual de `Java`, la cual no es más que el ejecutable de `Java`. Si se han seguido instalaciones normales, la ruta debería ser idéntica o muy similar a la que se muestra en el cuadro de texto. Se recomienda no utilizar espacios en blanco y/o caracteres especiales en las rutas que contienen los archivos necesarios para utilizar la herramienta.

```
C:\Users\{User}\AppData\Local\semanticmerge\semanticmergetool.exe
--source={absolute path}\example.mdl
--destination={absolute path}\example.mdl
--externalparser="-jar C:\Users\{User}\VensimPlugin4SM\VensimPlugin4SemanticMerge.jar"
--virtualmachine="C:\Program Files\Java\jdk-11.0.8\bin\java.exe"
```

A.1.5. Configuración para el uso con PlasticSCM

Si el producto desarrollado se va a utilizar utilizando la herramienta de `PlasticSCM` el proceso es sencillo.

Suponiendo una instalación típica, se encontrará una carpeta llamada `plastic4` en la ruta predeterminada:`C:\Users\{User}\AppData\Local\`, donde `{User}` debe ser sustituido por el nombre del usuario en Windows. En dicha carpeta se deberá crear un archivo con nombre `externalparsers.conf`, si no existía previamente. Es importante remarcar que la carpeta `AppData` está **oculta**, por lo que deberemos activar los elementos ocultos en la sección "Vista" de la barra de navegación del explorador de archivos.

Este archivo tiene la funcion de enlazar formatos no detectados de forma predefinida con sus correspondientes herramientas externas. En este caso, se debe asociar la extensión `.mdl` correspondiente a los ficheros `Vensim` a la ruta absoluta donde esté ubicada la herramienta externa, `VensimPlugin4SemanticMerge.jar`. Concretamente, el contenido de dicho fichero deberá de ser similar al siguiente cuadro de texto.

```
.mdl=java -jar C:\Users\{User}\VensimPlugin4SM\VensimPlugin4SemanticMerge.jar
```

Se recomienda no incluir espacios en blanco o caracteres especiales en las carpetas que conforman la ruta donde se encuentra alojado `VensimPlugin4SemanticMerge.jar`.

A.1.6. Configuración para el uso con `gmaster`

Actualmente `gmaster` es gratuito. En un futuro se planifica contar con una versión gratuita para uso no comercial.

Si suponemos una instalación típica de `gmaster`, se encontrará una carpeta `gmaster`, en la ruta predeterminada: `C:\Users\{User}\AppData\Local\`, donde `{User}` debe ser sustituido por el nombre del usuario en Windows. Como se ha explicado antes, se debe tener precaución, pues la carpeta `AppData` está **oculta**.

Dentro de esa carpeta se encontrará una denominada `\config`. Se debe crear en dicha carpeta un archivo con nombre `externalparsers.conf`, si no existía ya anteriormente. Este archivo tiene la función de enlazar formatos no detectados de forma predefinida con sus correspondientes herramientas externas para `SemanticMerge`.

En nuestro caso, debemos asociar la extensión `.mdl` a la ruta absoluta del *plugin* para `Vensim`. Concretamente, el contenido del archivo `externalparsers.conf` será similar al siguiente bloque de texto.

```
.mdl=java -jar C:\Users\{User}\VensimPlugin4SM\VensimPlugin4SemanticMerge.jar
```

Se recomienda no incluir espacios en blanco o caracteres especiales en las carpetas que conforman la ruta donde se encuentra alojado `VensimPlugin4SemanticMerge.jar`.

A.2. Manual de usuario

Una vez descargados todos los archivos y configurada la integración con `SemanticMerge` o `PlasticSCM` o `gmaster` se puede comenzar a trabajar con la herramienta.

Este manual se centrará en describir el uso integrado en `gmaster`. El proceso hace uso de los tres programas indicados en el Manual de instalación (el Adder, el Eraser y el plugin de Vensim para SemanticMerge). En las subsecciones siguientes se explica cada programa en profundidad.

En este punto se explican de manera general los dos flujos de trabajo más habituales al usar estas herramientas. Un punto a destacar es la recomendación de utilizar ficheros `Vensim` que **no** contengan caracteres especiales y/o espacios en blanco en su nombre, pues la herramienta `SemanticMerge` puede ocasionar problemas en estos casos.

El primer flujo de trabajo describe la situación en la que se desean fusionar dos ramas (merge de una rama a otra).

En la Figura A.1 se muestra gráficamente este flujo de trabajo, suponiendo que se cuente con una rama *master* y una rama llamada *a-branch*. Si el usuario se encuentra en el punto en el que desea fusionar los cambios realizados en *a-branch* con la rama *master*, deberá:

1. Hacer *checkout* de la rama *master*.
2. Aplicar `ViewNamesAndDelimiterAdder.jar` al archivo `.mdl` (ver subsección A.2.1).
3. Hacer *commit* de los cambios realizados por el Adder al `.mdl`. El mensaje de commit podría ser "Prepare to merge". Opcionalmente, aplicar tag (semantic) al *commit* realizado.
4. Hacer *checkout* de la rama *a-branch*.
5. Realizar en esta rama los pasos (2) y (3) de este flujo de trabajo.
6. Realizar el *merge* de *a-branch* a *master* (siguiendo las indicaciones de cómo hacer *merge* en `gmaster`).
7. Para afianzar el merge se realiza un *commit*. Opcionalmente, aplicar tag (semantic) al *commit* del *merge*.
8. Aplicar `ViewNamesAndDelimiterEraser.jar` al archivo `.mdl` (ver subsección A.2.3).
9. Hacer *commit* de los cambios realizados por el Eraser al `.mdl`. El mensaje de *commit* podría ser "Ready to work with `Vensim`".
10. Hacer *push* de los cambios en *master* al repositorio remoto.

El segundo flujo de trabajo habitual describe la situación en la que se quiere comprobar los cambios entre versiones o *commits* en una misma rama.

En la Figura A.2 se muestra gráficamente este flujo de trabajo, suponiendo que se está trabajando en una rama y se desea poder utilizar `SemanticMerge` para ayudar a visualizar las diferencias entre las versiones de un archivo `.mdl`.

La propuesta de flujo de trabajo consiste en que cada modificación realizada a un archivo `.mdl` con `Vensim` de la que se quiera hacer seguimiento en el control de versiones, llevará:

1. Hacer *commit* con los cambios.
2. Aplicar `ViewNamesAndDelimiterAdder.jar` al archivo `.mdl` (ver subsección A.2.1).
3. Hacer *commit* de los cambios realizados por el Adder al `.mdl`. El mensaje de *commit* podría ser la concatenación del mensaje del *commit* anterior con "Prepared for SemanticDiff". Opcionalmente, aplicar tag (semantic) al *commit* realizado.
4. Aplicar `ViewNamesAndDelimiterEraser.jar` al archivo `.mdl` (ver subsección A.2.3).
5. Trabajar con el archivo `.mdl` de la forma habitual con Vensim hasta obtener una nueva versión de la que se quiera hacer seguimiento en el control de versiones.

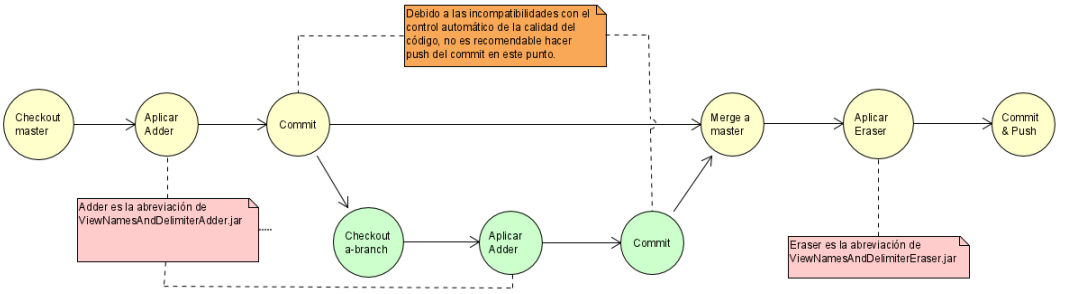

Figura A.1: Diagrama del flujo de trabajo con la herramienta en la fusión de ramas.

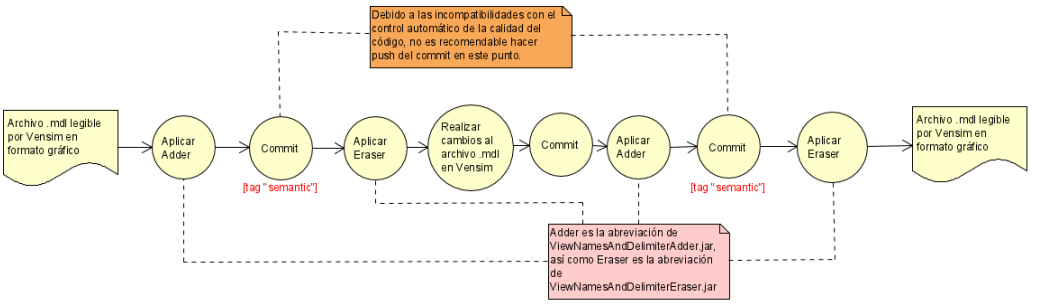

Figura A.2: Diagrama de flujo trabajo con la herramienta en la diferenciación de commits.

6. Realizar los pasos (1), (2), (3) y (4) de este flujo de trabajo.

Cuando se desea utilizar `SemanticMerge` para visualizar cambios entre versiones siempre debe hacerse entre los *commits* intermedios (los indicados en la Figura con el tag "semantic").

Así la propuesta consiste en hacer que lo que sería un *commit* en un flujo normal de trabajo con `git` se convierte en dos *commits*, uno después de trabajar con Vensim y uno después del Adder. Después de este (el llamado intermedio o con tag "semantic"), se debe dejar el archivo listo para trabajar nuevamente con Vensim aplicando el Eraser.

Es importante resaltar que no se debe hacer *push* de los *commits* intermedios (los indicados en la Figura con el tag "semantic"), pues esto podría causar problemas con el control de calidad para los archivos `Vensim` que se puede lanzar automáticamente al hacer *push* al repositorio remoto.

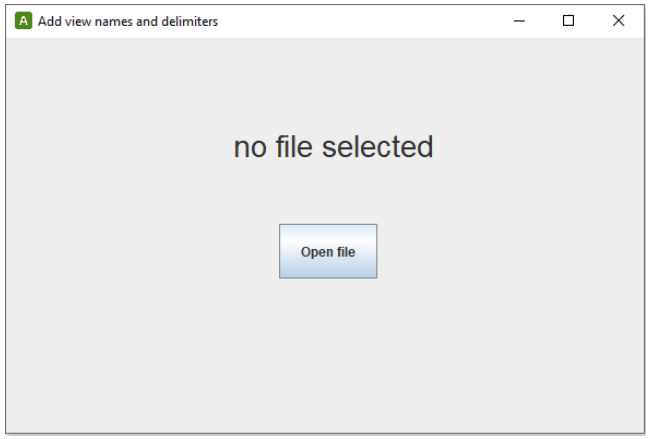

Figura A.3: Pantalla principal de la interfaz de adición de nombres de las vistas y delimitadores.

A.2.1. El programa auxiliar AddViewNamesAndDelimiters

`AddViewNamesAndDelimiters.jar` modifica los archivos `Vensim` que queremos utilizar, añadiendo los nombres de las vistas en la descripción de las ecuaciones y unos delimitadores de diferentes partes del archivo `.mdl`. Estos delimitadores son necesarios para que la herramienta externa `VensimPlugin4SemanticMerge` sea capaz de leer los archivos `Vensim` y diferenciar sus partes. De esta forma, a la hora de resolver un conflicto en una fusión de ramas será mucho más fácil identificar las causas del mismo y decidir la versión que debe resultar de dicha fusión (*merge*). Es importante señalar que una vez aplicado este programa, `Vensim` no será capaz de leer el archivo resultante de forma gráfica. Para revertir este efecto se utiliza el programa `EraseViewNamesAndDelimiters.jar`, que se explica más adelante (ver subsección A.2.3).

Antes de modificar el archivo, el programa genera automáticamente un archivo (con extensión `.2mdl`) a modo de *backup* del archivo antes de ser procesado. El archivo una vez modificado, conservará tanto el nombre como la extensión originales.

La pantalla inicial de `AddViewNamesAndDelimiters`, tal y como se aprecia en la Figura A.3, consta únicamente de un botón. Dicho botón, `Open file`, como su propio nombre indica, sirve para abrir el archivo a modificar. Sabremos que el archivo se ha cargado cuando la interfaz se modifique tal y como se muestra en la Figura A.4.

En caso de abrir un archivo cuyo formato no es `.mdl` o cualquier otro error, se notificará al usuario a través de un mensaje con un llamativo color rojo.

En caso de seleccionarse un archivo al que ya se la han añadido los nombres de las vistas y los delimitadores, también se indicará al usuario con un mensaje.

Una vez cargado el archivo, se debe pulsar el botón de la izquierda, `Process`, el cual procederá a modificar el archivo y a conservar el *backup* del original con la extensión `.2mdl`. Cuando haya finalizado el proceso, se notificará al usuario con un mensaje en color verde.

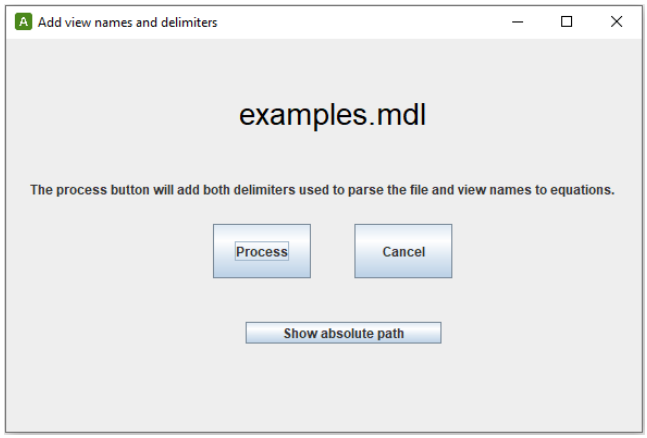

Figura A.4: Pantalla secundaria de la interfaz de adición de nombres de las vistas y delimitadores.

En caso de haber seleccionado un archivo que no era el deseado, se podrá hacer uso del botón de `Cancel` para volver a la pantalla inicial (Figura A.3).

Para permitir al usuario que pueda comprobar la ruta completa del archivo que ha seleccionado, se cuenta con el botón `Show absolute path`. Al pulsar la primera vez en dicho botón, se mostrará dicha ruta. Pulsando de nuevo en el mismo botón, se recuperará la vista del nombre del archivo sin ruta.

Finalmente, y con el objetivo de hacer la experiencia de usuario más clara y concisa, se explicará en la parte inferior del nombre del archivo seleccionado qué efecto tiene procesar el archivo con este programa.

Las dos interfaces gráficas de los programas `AddViewNamesAndDelimiters` y `EraseViewNamesAndDelimiters` son prácticamente idénticas, por lo que se debe comprobar el nombre de la etiqueta del programa.

A.2.2. El plugin de Vensim para SemanticMerge dentro de gmaster

Una vez el(los) archivo(s) `.mdl` deseado(s) son modificados añadiendo los nombres de las vistas en la descripción de las ecuaciones y los delimitadores de las partes de un `.mdl`, se deberán asentar sus cambios en el repositorio mediante un *commit* (ver flujos de trabajo explicados en las Figuras A.1 y A.2). Es importante modificar los archivos con el "Adder", puesto que sin ello `gmaster` no será capaz de reconocer los archivos **semánticamente**, sino únicamente como texto plano. Es decir, se conservará la capacidad de lectura convencional de la herramienta, pero no se aplicarán las mejoras que aporta el *plugin* para `SemanticMerge`.

Ya sea en el flujo descrito para realizar fusión de ramas (*merge*), o en el descrito para permitir la visualización de diferencias entre *commits* en la misma rama, *gmaster* detectará que la extensión del archivo es `.mdl` y utilizará la herramienta externa `VensimPlugin4SemanticMerge`

para realizar el análisis semántico. La herramienta nos mostrará las diferencias entre los dos archivos, proporcionándonos información sobre qué significa cada línea modificada.

La Figura A.5 muestra las diferentes partes de la interfaz gráfica de `gmaster`.

En la primera sección señalada con un recuadro rojo y el número 1, podemos observar la primera nueva característica que añade la herramienta. En esta sección se puede observar el recuento de cambios entre dos versiones de un mismo archivo. Como se puede ver en la figura, los cambios se distinguen en cinco apartados principales: adiciones, modificaciones, eliminaciones, movimientos y renombres. Dentro de cada una de estas categorías de cambios, a su vez podemos distinguir el tipo del elemento modificado como puede ser una ecuación y una vista, así como su nombre.

En la segunda sección señalada con un recuadro rojo y el número 2, se observan las dos versiones del archivo que se está analizando, mostrando en la parte izquierda la versión más antigua y en la derecha la actual (en el caso de un *merge* se mostraría la versión de la rama de origen y la versión de la rama de destino). Se observan franjas coloreadas que relacionan los cambios visualmente entre ambas versiones del archivo. Al lado de cada cambio aparece una inicial, la cual indicará la categoría de modificación de entre las cinco posibles explicadas anteriormente.

El recuadro en color azul etiquetado con 2.1 (en la parte de abajo de la sección 2), sirve para cambiar entre el modo convencional de diferencias entre versiones con texto plano, y el modo semántico propio de `SemanticMerge`.

El segundo recuadro en color azul etiquetado con 2.2, permite ver los cambios de una forma más compacta y visual, utilizando únicamente los grafismos asociados a las cinco categorías posibles de modificaciones. En la Figura A.6 se muestra el resultado de seleccionar esa forma de visualizar las diferencias.

Por último, en la tercera sección se aprecian los archivos modificados en la versión del proyecto o *commit* seleccionado.

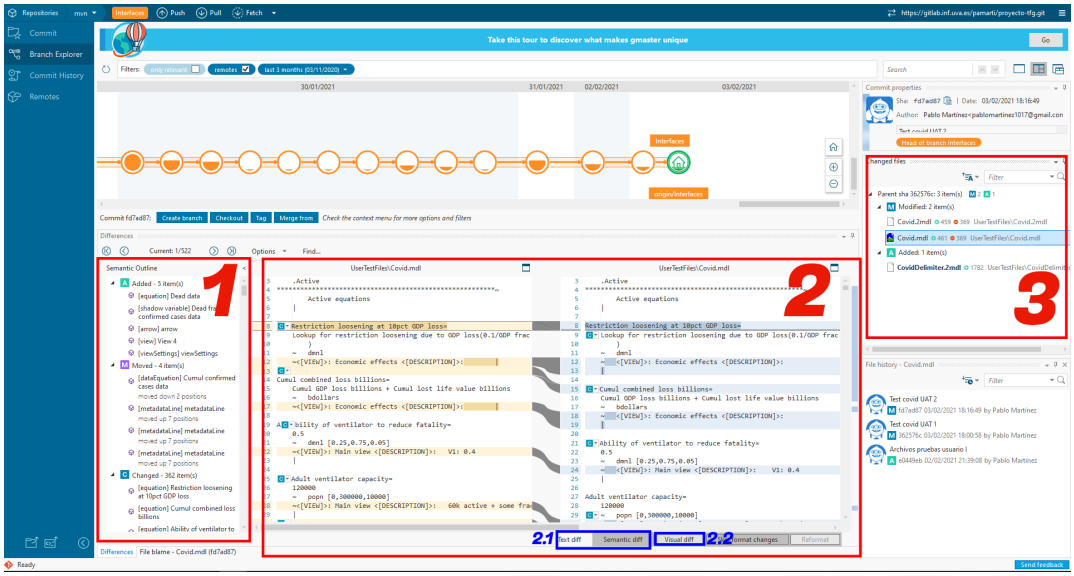

Figura A.5: Interfaz de gmaster y sus correspondientes partes.

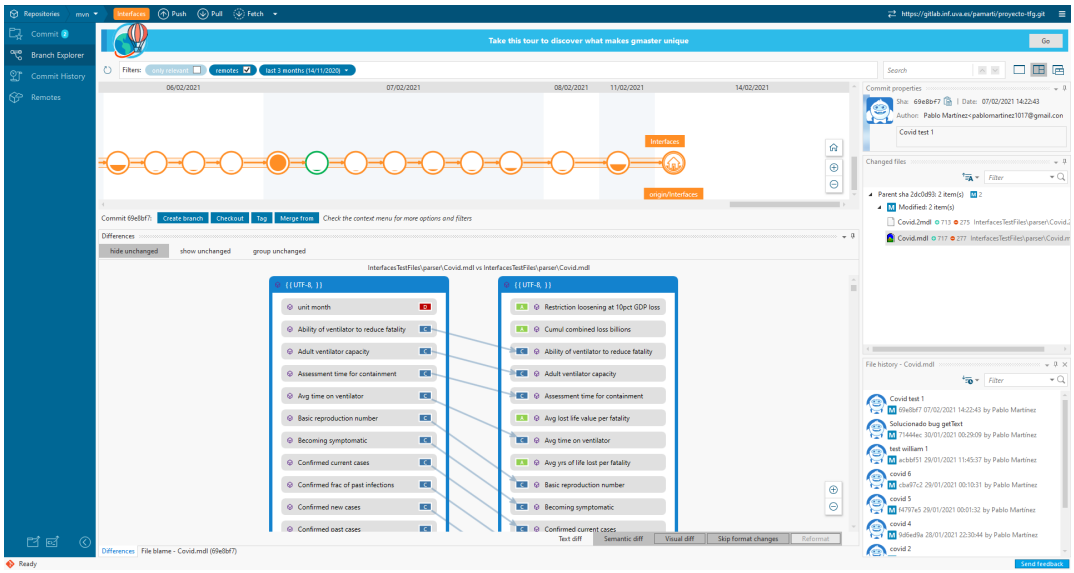

Figura A.6: Visualización de diferencias entre versiones de un .mdl basada en grafismos.

A.2.3. El programa auxiliar EraseViewNamesAndDelimiters

El programa `EraseViewNamesAndDelimiters.jar` modifica los archivos `Vensim` a los que se les haya añadido nombres de vistas en la descripción de las ecuaciones, y delimitadores de las partes de un `.mdl` (resultado de modificar un archivo `.mdl` con el programa

`AddViewNamesAndDelimiters`), eliminando todo lo que se le haya añadido extra.

Según el flujo de trabajo de fusión de ramas, una vez tengamos la rama fusionada y hecho el *commit* de *merge*, debemos utilizar `EraseViewNamesAndDelimiters` y de esta forma, los archivos vuelven a ser visibles y modificables con Vensim. De no hacerlo, no se podrá abrir correctamente el archivo con la interfaz gráfica de `Vensim`. Esto mismo sucede si el flujo de trabajo es el que se utilizaría para ayudar a visualizar las diferencias entre dos versiones de un `.mdl` en la misma rama, hay que volver siempre a un `.mdl` legible por Vensim para poder continuar con el trabajo de programación de los modelos.

La interfaz se comporta igual que la explicada con el primer programa auxiliar (`AddViewNamesAndDelimiters`). Se recuerda que son prácticamente idénticas, por lo que se debe comprobar el nombre de la etiqueta del programa. La explicación textual del efecto del botón *Process* también pretende ayudar en esta diferenciación.

`EraseViewNamesAndDelimiters` también se encarga de generar automáticamente un archivo (con extensión `.2mdl`) a modo de *backup* antes de realizar modificaciones. Pero en este caso lo llamará añadiendo la palabra Delimiter al nombre del archivo original (`NombreDeArchivoDelimiter.2mdl`) para no reemplazar el *backup* creado con `AddViewNamesAndDelimiters`. El archivo una vez modificado, conservará tanto el nombre como la extensión originales.

Las Figuras A.7 y A.8 muestran las pantallas de `EraseViewNamesAndDelimiters`.

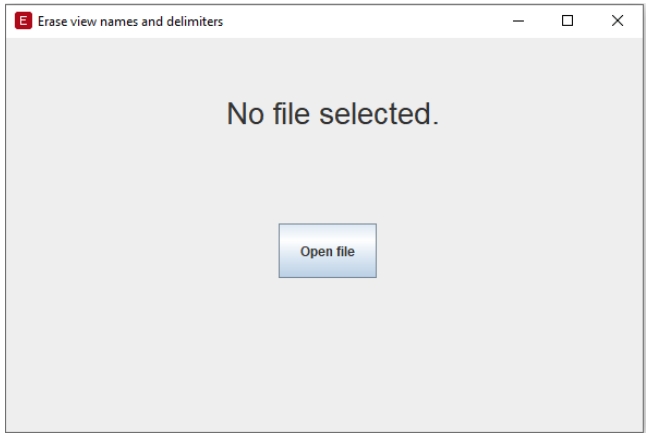

Figura A.7: Pantalla principal de la interfaz de eliminación de delimitadores.

Es muy útil en el repositorio `git` añadir el archivo `.gitignore` indicando que los archivos de *backup* no se tengan en cuenta en el control de versiones. El siguiente cuadro de texto muestra el contenido de `.gitignore` típico para un proyecto de programación de modelos con Vensim:

```
# Be careful and do not name an important file with temporal file extension
*.tmp
```

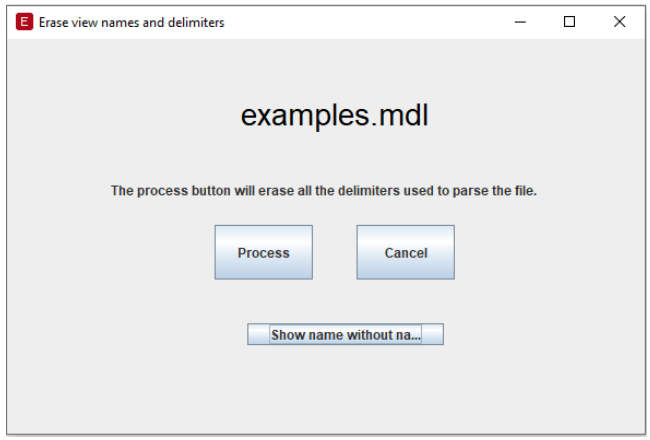

Figura A.8: Pantalla secundaria de la interfaz de eliminación de delimitadores.

```
# Word temporary
~$*.doc*

# Word Auto Backup File
Backup of *.doc*

# Excel temporary
~$*.xls*

# Excel Backup File
*.xlk

# Vensim mdl Backup File
*.2mdl

# Vensim simulation
*.vdf
```

A.2.4. Parámetros de las líneas de definición de variables en las vistas.

Aunque muchos de los parámetros de las líneas asociadas a las variables en la parte de definición de las vistas son triviales, es importante poder conocer qué representan cada uno de los campos de dichas líneas.

Podemos diferenciar dos claras estructuras diferentes [51]. Los campos de las variables marcados con **(*)** se han considerado no triviales e importantes a tener en cuenta a la hora de resolver un conflicto, ya que contienen datos importantes en la estructura de la vista y no son simples parámetros que configuran aspectos triviales del diseño como pueden ser el color o la forma de una variable:

- **Variables**. Esta estructura representa las variables, incluyendo sus variaciones como las variables sombra. También se incluye en este grupo a las válvulas o *valves* y a los comentarios. Se componen de:
  - **1**. n → Código numérico que indica el tipo de variable **(*)**.
  - **2**. id → Código numérico ascendente que indica la posición respecto a otras variables en que se introdujo la variable a la vista **(*)**.
  - **3**. name → Nombre de la variable. Si fuese un '0', el nombre de la variable aparecería aislado en la línea siguiente **(*)**.
  - **4,5**. x, y → Posición de la variable en el plano **(*)**.
  - **6,7**. w, h → Ancho y alto de la variable en el plano **(*)**.
  - **8**. sh → Forma alrededor de la palabra y características relativas. Estas se disponen en un conjunto de bits que es representado en forma decimal.
  - **9**. bits → Conjunto de bits que indica si pueden entrar y/o salir flechas de la variable **(*)**.
  - **10.** hid → Indica si la variable está oculta. Un valor distinto de cero indica el nivel de oculta en que se encuentra la variable. **(*)**
  - **11**. hasf → Indica si la variable tiene una fuente de texto especial.
  - **12**. tpos → Indica la posición del texto respecto a la forma que encierra la variable.
  - **13**. bw → Indica el ancho del borde de la caja o forma que rodea a la variable.
  - **14**. nav1 → Indica el número de vista al que puede redireccionar la variable.
  - **15**. nav2 → En el caso de que haya más de 255 vistas, el número de vista pasa a calcularse con: $nav1 + 256 * nav2$.
  - **16**. box → Color del borde de la caja o forma que rodea a la variable.
  - **17**. fill → Color de relleno de la caja o forma que rodea a la variable.
  - **18**. font → Tipografía de la variable.
  - **19-24**. Constantes numéricas desconocidas.
  - **25.** visualInfo → Campo utilizado para anexar el nombre de la variable en caso de estar definido en la línea siguiente.

- **Flechas**. Esta estructura representa las uniones y relaciones entre el resto de variables. Se componen de:
  - **1**. n → Código numérico que indica el tipo de variable. En las flechas o *arrows* siempre será 1 **(*)**.
  - **2**. id → Código numérico ascendente que indica la posición respecto a otras variables en que se introdujo la variable a la vista **(*)**.
  - **3,4**. from, to → Ids de las variables de donde sale la flecha y a donde llega respectivamente **(*)**.
  - **5**. shape → Forma de la flecha (línea recta, curvada...etc).
  - **6**. hid → Indica si la flecha está oculta. Un valor distinto de cero indica el nivel de oculta en que se encuentra la flecha **(*)**.
  - **7**. pol → Indica la polaridad de la flecha.
  - **8**. thickness → Indica el grosor de la flecha. Un valor de más de 20 unidades indica que la flecha utiliza dos líneas paralelas.
  - **9**. hasf → Indica cambios en el color y fuente que la vista trae como predeterminados.
  - **10**. dtype → Indica el delay de la flecha.
  - **11**. res → Valor reservado, debería ser 0.
  - **12**. color → Color de la flecha o "$-1--1--1$".
  - **13**. font → Fuente o color de la flecha. Este campo puede permanecer vacío.
  - **14,15**. nc, pointlist → El primer valor representa el número de puntos intermedios que tiene la flecha. El segundo valor es la vista de las coordenadas de dichos puntos intermedios **(*)**.

En el caso de el apartado catalogado como "metadatos", esta sección apenas varía y su tamaño es muy reducido en comparación del tamaño total del fichero `.mdl`. No parece existir documentación sobre esta última parte, pero son líneas referentes a la configuración del archivo y similares. Debido a su pequeño tamaño y que la gramática permite separar línea por línea este apartado, se considera que no debería generar problemas de comprensión.

A.2.5. Advertencias sobre posibles cambios que puede realizar el programa Vensim en los ficheros.

Se listan a continuación una serie de advertencias acerca del funcionamiento de las herramientas de `SemanticMerge` y `gMaster` sobre cómo pueden llegar a afectar al formato textual de los archivos `.mdl`:

- En la parte donde se listan las variables asociadas a una vista, uno de los parámetros de cada variable es su orden de aparición en el texto. Por ello, ciertas acciones como eliminar una variable o moverla de vista, modificarán todas las líneas que representan a variables en esa vista con un orden de aparición mayor.

- Ciertas acciones provocan que se inserten o eliminen líneas de metadatos en el final del archivo.
- Al mover variables entre vistas, la definición de la variable en la parte de ecuaciones aparecerá como cambio, sin embargo, en la parte de definición en las vistas aparecerá como eliminación en la vista anterior e inserción en la nueva.
- En ciertas ocasiones, `Vensim` transforma la definición de una ecuación de forma extendida (en tres líneas) a forma compactada (una única línea) y viceversa.
- Borrar una vista sin borrar previamente las variables de dicha vista hace que las variables no se eliminen y queden como variables vacías en el formato textual del archivo.
- En ciertas ocasiones, `Vensim` aumenta el número de parámetros de una variable, generalmente con parámetros con valor cero.
- En ciertas ocasiones, mover variables modifica ciertos elementos que `Vensim` denomina válvulas.
- Eliminar ciertas partes de un modelo que hagan que el modelo no sea correcto respecto al funcionamiento de `Vensim` puede acarrear problemas en la definición de algunas variables.
- En el caso de las variables numéricas como comentarios o *valves*, el campo del nombre lleva un valor numérico asignado que según la documentación oficial, debe de ser ignorado [50]. Este valor cambia tras realizar ciertas operaciones como adiciones o eliminaciones de elementos.

A.2.6. Advertencias sobre el nombrado de variables.

Existen ciertas formas de nombrar a las variables que hacen que el programa falle, consecuencia de una mala formación del árbol descriptor del fichero en el archivo `YAML` o ciertos caracteres especiales no soportados por el sistema de codificación del proyecto, `UTF-8`. Los nombrados de variable que causan o pueden llegar a causar problemas son los siguientes:

- Utilizar el carácter ':' seguido de un espacio en blanco.
- Acabar el nombre de la variable en uno o varios espacios en blanco.
- Utilizar comillas, salvo en los casos en los que las comillas envuelvan todo el nombre de la variable. Se recomienda utilizar comillas dobles.
- Utilizar caracteres especiales no soportados.

A.2.7. Enlaces de interés.

En caso de querer conocer más acerca del funcionamiento específico de `SemanticMerge`, `PlasticSCM` y/o `gMaster` o cómo ambas herramientas incorporan herramientas o *parsers* externos, se recomienda consultar [24], [34], [36], [37], [40] y [44] en la bibliografía del proyecto.

Apéndice B

Resumen de enlaces adicionales

Los enlaces útiles de interés en este Trabajo Fin de Grado son:

- Repositorio del proyecto: `https://gitlab.inf.uva.es/pamarti/proyecto-tfg`.

Estructura del repositorio (carpeta de la memoria)

Bibliografía

[1] andrew1234. Java swing jfilechooser. https://www.geeksforgeeks.org/java-swing-jfilechooser/, 2018. Accesed: 2021-19-01.

[2] Apache. Maven. https://maven.apache.org/, 2020. Accesed: 2020-9-23.

[3] Baeldung. Java with antlr. https://www.baeldung.com/java-antlr, 2020. Accesed: 2020-9-15.

[4] Daniel Bazaco. Final vensim grammar in antlr 4, developed for this project. https://gitlab.inf.uva.es/danbaza/tfg-sonarvensim/-/blob/master/src/main/antlr/Model.g4, 2019. Accesed: 2020-9-15.

[5] Rocket Chat. Rocket chat. https://rocket.chat/, 2020. Accesed: 2020-9-23.

[6] CoronA. Antlr 4.5 - mismatched input 'x' expecting 'x'. https://stackoverflow.com/questions/29777778/antlr-4-5-mismatched-input-x-expecting-x, 2015. Accesed: 2020-9-18.

[7] Departamento de Informática. Archivos cloud vensim. https://cloud.infor.uva.es/index.php/s/3r85R4oavbe3ivk, 2020. Accesed: 2020-9-30.

[8] Universidad de Valladolid. Guía docente de la signatura. trabado de fin de grado (mención ingeniería de software). https://www.inf.uva.es/wp-content/uploads/2016/06/G46976.pdf, 2020. Accesed: 2020-10-2.

[9] Universidad de Valladolid. Preguntas frecuentes. https://www.uva.es/export/sites/uva/2.docencia/2.02.mastersoficiales/2.02.13.preguntasfrecuentes/index.html, 2020. Accesed: 2020-10-2.

[10] eclass. ¿cuáles son los eventos de scrum? https://blog.eclass.com/cuales-son-los-eventos-de-scrum-conocelos-aqui-0, 2020. Accesed: 2021-22-01.

[11] Norberto Figuerola. Riesgos: Plan de mitigación vs plan de contingencia vs fallback plan. https://articulospm.files.wordpress.com/2015/06/riesgos-plan-mitigacion-vs-plan-contingencia-vs-fallback-plan.pdf, 2015. Accesed: 2020-9-26.

[12] Tobi G. Java special characters in regex. https://stackoverflow.com/questions/14134558/list-of-all-special-characters-that-need-to-be-escaped-in-a-regex, 2019. Accesed: 2020-11-11.

[13] Research Gate. Locomotion h2020. https://www.locomotion-h2020.eu/, 2020. Accesed: 2020-9-16.

[14] Research Gate. Medeas. https://www.medeas.eu/#home, 2020. Accesed: 2020-9-16.

[15] GitLab.org. Gitlab. https://gitlab.com/gitlab-org, 2020. Accesed: 2020-9-23.

[16] gMasterRealeaseNotes. Release notes. https://gmaster.io/releasenotes/0.9.471.0/-6, 2021. Accesed: 2021-11-02.

[17] Object Management Group. Unified modeling language. https://www.uml.org/, 2021. Accesed: 2021-11-02.

[18] James P. Houghton. test-models. https://github.com/SDXorg/test-models, 2020. Accesed: 2020-9-30.

[19] Joel Francia Huambachano. ¿qué es scrum? https://www.scrum.org/resources/blog/que-es-scrum, 2017. Accesed: 2021-22-01.

[20] Change Vision Inc. Astah professional. https://astah.net/products/astah-professional/, 2021. Accesed: 2021-11-02.

[21] La información. ¿cuál es el coste real de tener un trabajador para una empresa? https://www.lainformacion.com/practicopedia/cual-es-el-coste-real-de-un-trabajador-para-una-empresa/6491464/#:~:text=Cotizaciones%20a%20la%20Seguridad%20Social,26.300%20euros%20a%20la%20empresa., 2019. Accesed: 2020-9-26.

[22] Jitsi.org. Jitsi meet. https://meet.jit.si/, 2020. Accesed: 2020-9-23.

[23] Bart Kiers. If/else statements in antlr using listeners. https://stackoverflow.com/questions/15610183/if-else-statements-in-antlr-using-listeners, 2013. Accesed: 2020-9-15.

[24] Sergio L. Using external parsers with gmaster. http://blog.gmaster.io/2018/03/using-external-parsers-with-gmaster.html, 2018. Accesed: 2020-14-11.

[25] Pablo Santos Luaces. charposition. https://github.com/PlasticSCM/charposition, 2016. Accesed: 2020-31-10.

[26] Pablo Martínez López. Gramática en antlr4 para vensim. https://gitlab.inf.uva.es/pamarti/proyecto-tfg/-/blob/master/src/main/antlr4/es/uva/inf/grammar/Grammar.g4, 2020. Accesed: 2020-9-29.

[27] Pablo Martínez López. Simplejavaparser-semanticmerge. https://github.com/HylianPablo/SimpleJavaParser-SemanticMerge, 2020. Accesed: 2020-9-29.

[28] McGraw-Hill. Business dynamics. http://www.mhhe.com/business/opsci/sterman/models.mhtml, 2001. Accesed: 2020-9-30.

[29] Microsoft. Visual studio code. https://code.visualstudio.com/, 2020. Accesed: 2020-9-23.

[30] migraciones y seguridad social Ministerio de trabajo. Boletín oficial del estado. iii. otras disposiciones. https://www.boe.es/boe/dias/2019/06/03/pdfs/BOE-A-2019-8222.pdf, 2019. Accesed: 2020-9-26.

[31] MiryamGSM. External parser sample. https://github.com/PlasticSCM/external-parser-sample/tree/master-SCM20098, 2017. Accesed: 2020-8-28.

[32] picodotdev. Cómo ejecutar un proceso del sistema con java. https://picodotdev.github.io/blog-bitix/2016/03/como-ejecutar-un-proceso-del-sistema-con-java/, 2016. Accesed: 2020-16-12.

[33] PlasticSCM. Semanticmerge pricing. https://users.semanticmerge.com/Checkout, 2020. Accesed: 2020-9-26.

[34] Plastic SCM. gmaster - git gui. https://gmaster.io/, 2021. Accesed: 2021-21-01.

[35] Shodor. Vensim models. http://shodor.org/talks/ncsi/vensim/, 2005. Accesed: 2020-9-19.

[36] Códice Software. Custom languages in semantic version control. http://blog.plasticscm.com/2015/09/custom-languages-in-semantic-version.html, 2015. Accesed: 2021-15-02.

[37] Códice Software. External parsers. https://semanticmerge.com/documentation/external-parsers/external-parsers-guide, 2016. Accesed: 2020-9-19.

[38] Códice Software. Advanced version control - pocket guide. https://www.plasticscm.com/documentation/advanced-version-control-guide#two-way-merge, 2020. Accesed: 2020-9-17.

[39] Códice Software. Cygnus solutions. https://www.cygwin.com/, 2020. Accesed: 2021-24-02.

[40] Códice Software. Semanticmerge 2.0. https://semanticmerge.com/, 2020. Accesed: 2020-7-11.

[41] Códice Software. Semanticmerge features. http://www.semanticmerge.com/features, 2020. Accesed: 2020-9-17.

[42] Códice Software. Semanticmerge intro guide. https://semanticmerge.com/documentation/intro-guide/semanticmerge-intro-guide, 2020. Accesed: 2020-9-17.

[43] Códice Software. Xdiff and xmerge. https://www.plasticscm.com/features/xmerge, 2020. Accesed: 2020-9-17.

[44] Códice Software. Plasticscm. https://www.plasticscm.com/?utm_source=plasticscm-blog&utm_medium=blog-info&utm_content=whoweare, 2021. Accesed: 2021-15-02.

[45] Códice Software. Plasticscm - the distributed version control for big projects. https://www.plasticscm.com/, 2021. Accesed: 2021-2-10.

[46] Sonarqube. Code quality and code security. https://www.sonarqube.org/, 2021. Accesed: 2021-27-02.

[47] Saumitra Srivastav. Antlr4 - visitor vs listener pattern. https://saumitra.me/blog/antlr4-visitor-vs-listener-pattern/, 2017. Accesed: 2020-9-18.

[48] Johannes Link Matthias Merdes Marc Philipp Juliette de Rancourt Christian Stein Stefan Bechtold, Sam Brannen. Junit 5 user guide. https://junit.org/junit5/docs/current/user-guide/, 2020. Accesed: 2020-23-12.

[49] Ventana Systems. Defined and shadow variables. https://www.vensim.com/documentation/22890.htm, 2020. Accesed: 2020-10-6.

[50] Ventana Systems. Sketch information. https://www.vensim.com/documentation/ref_sketch_format.html, 2020. Accesed: 2021-2-21.

[51] Ventana Systems. Sketch objects. https://www.vensim.com/documentation/index.html?23275.htm, 2020. Accesed: 2020-10-3.

[52] Ventana Systems. Variable types. https://www.vensim.com/documentation/ref_variable_types.htm, 2020. Accesed: 2020-9-16.

[53] Ventana Systems. Vensim. https://vensim.com/, 2020. Accesed: 2020-23-12.

[54] Ej Technologies. Java profiler - jprofiler. https://www.ej-technologies.com/products/jprofiler/overview.html, 2021. Accesed: 2021-24-02.

[55] Linus Torvalds. Git. https://git-scm.com/, 2020. Accesed: 2020-9-23.

[56] TutorialsPoint. Compiler design - symbol table. https://www.tutorialspoint.com/compiler_design/compiler_design_symbol_table.htm#:~:text=Symbol%20table%20is%20an%20important,synthesis%20parts%20of%20a%20compiler., 2020. Accesed: 2020-10-7.

[57] Wangdq. How to display antlr tree gui. https://stackoverflow.com/questions/23809005/how-to-display-antlr-tree-gui, 2014. Accesed: 2020-9-24.

[58] Wikipedia. Swing (biblioteca gráfica). https://es.wikipedia.org/wiki/Swing_(biblioteca_gr%C3%A1fica), 2021. Accesed: 2021-20-01.