Integración de un sistema de reescritura de términos en una herramienta de desarrollo software industrial

Abel Gómez, Artur Boronat, José Á. Carsí, Isidro Ramos.

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n
46022 Valencia. España.
{agomez | aboronat | pcarsi | iramos}@dsic.upv.es

Resumen. Los métodos formales proporcionan buenas propiedades para abordar problemas de Ingeniería del Software: validación de sistemas, integración de artefactos software, etc. En este sentido, diversas han sido las aproximaciones formales para la resolución de problemas en Ingeniería de Modelos, por ejemplo, mediante teoría de grafos, o reescritura de términos. En esta última aproximación encontramos Maude: un potente sistema basado en lógica ecuacional y lógica de reescritura. A pesar de todo esto, debido a prejuicios o malas experiencias, las herramientas industriales no suelen apoyarse en estos métodos, abordando la resolución los problemas de forma *ad-hoc*. En este contexto se ha desarrollado un conjunto de herramientas que integran el sistema formal Maude en un entorno de desarrollo industrial como es Eclipse. Este artículo muestra las características de estas herramientas y las posibilidades que ofrecen al usuario y futuros desarrolladores.

1. Introducción.

En la Ingeniería del Software, los métodos formales proporcionan buenas propiedades para la resolución de problemas tales como transformación de modelos, evolución de sistemas, interoperabilidad, etc. No obstante, la aplicación de formalismos no siempre ha sido bien recibida en entornos industriales, provocando que los entornos de modelado y desarrollo software de más éxito industrial no proporcionen ningún tipo de herramienta que provea un soporte formal. En este contexto podemos encontrar, por ejemplo, la plataforma Eclipse [Eclipse,EclOv03], de código abierto, y ampliamente empleada tanto como entorno de programación (*Java*, *C++*, etc.), como de modelado (mediante el framework *EMF* [EMF]). Prueba del auge que está experimentando esta herramienta es el importante papel que juega en los productos de *IBM*, ya que su nueva familia de herramientas de desarrollo software y *CASE* están basadas en esta plataforma (i.e. *IBM Websphere* o *IBM Rational Software Architect*).

En el campo de las herramientas formales podemos encontrar *Maude* [Maude], un eficiente sistema de reescritura de términos que proporciona soporte para la definición de especificaciones algebraicas, *model–checking*, demostración de propiedades de terminación y confluencia, etc. En este trabajo se presenta la integración de este entorno formal en la plataforma Eclipse con el objetivo de proporcionar dos funcionalidades básicas: en primer lugar, facilitar una *API* al programador para emplear *Maude* desde un programa Java; y en segundo lugar, proveer al usuario una interfaz de desarrollo de programas *Maude* que supla las carencias de los deficientes entornos actuales.

El artículo se estructura de la siguiente manera: en la sección 2 se presenta una visión global de las herramientas desarrolladas. La sección 3 muestra las características de las *APIs* proporcionadas al usuario para poder emplear *Maude* de forma programática, y la sección 4 describe la interfaz diseñada para el desarrollo de programas *Maude*. La sección 5 muestra algunos trabajos relacionados, y por último se plantean las conclusiones en la sección 6.

2. Maude Development Tools: Las herramientas de desarrollo de Maude.

Eclipse proporciona un esquema de diseño modular y fácilmente extensible mediante un mecanismo de *plug-in*s. Las herramientas de desarrollo de *Maude* se componen de dos *plug-in*s que permiten la integración de Maude en Eclipse para ser utilizado desde código Java, o bien a través de interfaces gráficas.

El primero de ellos, denominado *Maude Daemon*, encapsula en un conjunto de clases Java un proceso *Maude*, proporcionando dos *APIs* diferenciadas para el control del proceso de forma interactiva o la ejecución de trabajos por lotes. De igual manera, proporciona los mecanismos para la configuración automática de *Maude*, de forma que resulten transparentes al usuario de la *API* los aspectos dependientes del sistema en que se ejecute.

El segundo *plug-in*, *Maude SimpleGUI*, es un sencillo *IDE* para el desarrollo de programas *Maude*. Entre sus componentes encontramos un editor de texto, una vista de consola que permite controlar la ejecución, así como diversos asistentes y menús que facilitan el trabajo del usuario. Un objetivo perseguido ha sido la portabilidad al mayor número de plataformas posible.

3. Maude Daemon.

En primer lugar, este *plug-in* proporciona una *API* para controlar el proceso *Maude* durante la ejecución de un programa de *Java*; de forma que se pueda configurar *Maude*, iniciarlo, enviar comandos, obtener su respuesta, y detener el proceso (de forma normal, o forzosa, si el proceso no responde).

En segundo lugar se ofrece un soporte gráfico (integrado en las preferencias de Eclipse) para configurar *Maude*. Esto es, especificar la ruta del ejecutable y el fichero que define *Full Maude*, modo de ejecución, etc. Cabe mencionar que toda actividad realizada por el proceso *Maude*, es almacenada en un archivo de registro, para así poder observar a posteriori el comportamiento del sistema en caso de fallos.

Dada la implementación actual de *Maude*, sólo es posible ejecutar este sistema sobre equipos con diversas variantes de sistemas *UNIX*. Igualmente, para el control del proceso *Maude* desde el *plug-in Maude Daemon* son necesarios los programas «bash» y «kill», propios también de sistemas *UNIX*. No obstante, basándonos en otros trabajos

realizados, como por ejemplo *Maude Workstation* [Muñoz04], se ha conseguido ejecutar *Maude* sobre un sistema *Windows* haciendo uso del entorno *Cygwin* [*Cygwin*].

3.1. Estructura del *plug-in*: dos *APIs* diferenciadas.

La Figura 1 muestra la estructura de paquetes. El paquete «*core*» (*es.upv.dsic.issi.moment.MaudeDaemon.core*) es el paquete principal. En él se definen las clases que permitirán crear e interactuar con el proceso *Maude*. En el paquete «*Maude*», se definen las interfaces que implementarán otras clases que intervienen con el proceso «*Maude*», y en el paquete «*Maude.internal*», las clases que implementan estas interfaces. El paquete «*parser*» es generado a partir de una gramática por el *plug-in ANTLR* [ANTLR], y nos sirve de ayuda para analizar los comandos de *Full Maude*. Por último, los paquetes «*ui*», y «*ui.preferences*», agrupan las clases que implementan la interfaz de usuario del *plug-in*.

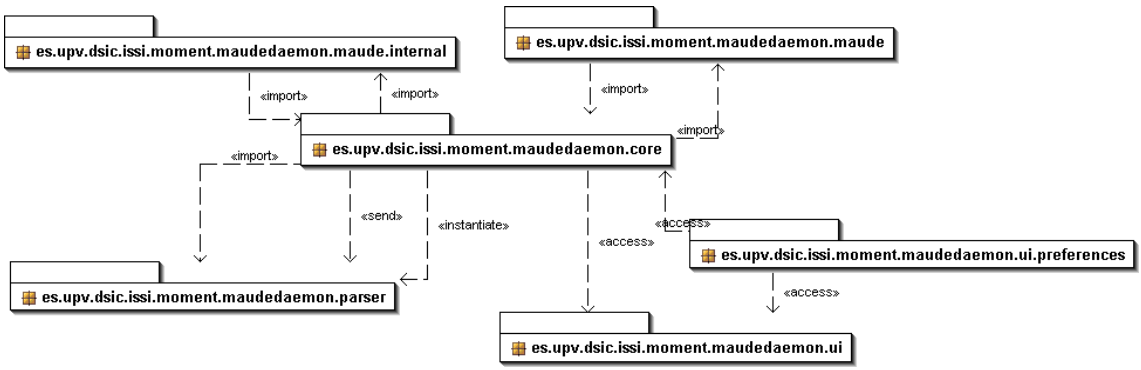

Figura 1. Diagrama de paquetes de Maude Daemon.

A continuación se muestra el diagrama de clases *UML* de las principales clases del *plug-in* que permiten controlar la ejecución de un proceso *Maude* y componen la *API* que se proporciona al programa.

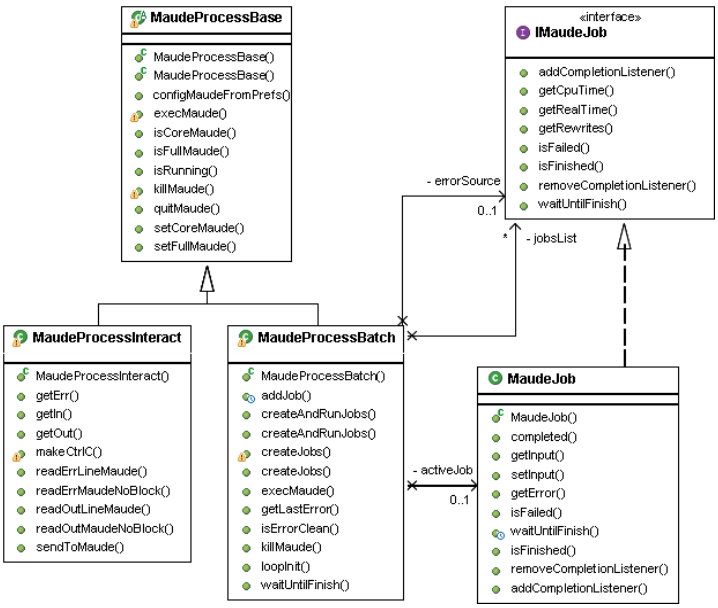

Figura 2. Diagrama de las clases que implementan la API ofrecida al programador.

Como se puede observar en la Figura 2, existe una clase base, denominada *MaudeProcessBase*, que implementa todo el mecanismo de creación de un proceso *Maude.* Sus especializaciones definen los métodos para proporcionar las diferentes *APIs* para el programador, una para usar de forma interactiva y otra como un proceso por lotes. *MaudeJob* es la clase que encapsula cada uno de los «lotes» de comandos *Maude* que se desean ejecutar.

Las dos posibles formas de interactuar sobre un proceso *Maude* son incompatibles entre sí, y es por ello que se impide que se pueda interactuar con un mismo proceso *Maude* con ambas *APIs* de forma simultánea.

3.2. Creación de un proceso *Maude*.

El mecanismo de creación de un proceso *Maude* está basado en la implementación de la plataforma *Maude Workstation* [Muñoz04]. Para lanzar un proceso externo al programa en ejecución, del que se dispone de un fichero ejecutable en disco, se hace uso del método *exec(String cmd)* de la clase *java.lang.Runtime*, donde «*cmd*» es el comando externo a ejecutar. Esto devuelve una instancia de *java.lang.Process*, que es la que permitirá controlar el proceso que hemos lanzado.

En particular, los métodos que nos resultarán más relevantes de ella son *getInputStream()*, *getOutputStream()* y *getErrorStream()*. Estos métodos devuelven los *Streams* correspondientes a la entrada estándar del proceso, a la salida estándar, y a la de error, respectivamente.

La lectura y escritura de forma adecuada de estos *Streams* es la que permite enviar comandos a *Maude*, y obtener sus respuestas o posibles errores. El código Java que nos permite ejecutar un programa y controlarlo es, básicamente, el siguiente:

```
Process MaudeProcess = runTime.exec(commandString);

BufferedWriter buffStdin = new BufferedWriter(
      new OutputStreamWriter(MaudeProcess.getOutputStream()));

BufferedReader buffStdout = new BufferedReader(
      new InputStreamReader(MaudeProcess.getInputStream()));

BufferedReader buffStderr = new BufferedReader(
      new InputStreamReader(MaudeProcess.getErrorStream()));
```

Cabe mencionar que el comando a ejecutar no es directamente Maude, sino un shell «*bash*» (en Windows se empleará el de *Cygwin*) que ejecutará Maude: «`bash -c echo $$; exec 'pathMaude' -interactive`». Esto permite gestionar las interrupciones, que permiten activar el modo de depuración de Maude (ejecución paso a paso de las reglas de reescritura) o matar directamente el proceso en su caso.

3.3. Funcionamiento de las *APIs*.

A continuación se detalla la funcionalidad de las dos *APIs* que proporciona el *plug-in*:

3.3.1. *MaudeProcessInteract*.

Esta *API* encapsula mediante métodos las lecturas y escrituras sobre los *buffers* del proceso *Maude* para simplificar la comunicación con el proceso *Maude*. Sus principales métodos son:

```
public void sendToMaude(String txt)

public String readOutLineMaude()

public String readErrLineMaude()

public String readOutMaudeNoBlock(int maxLen)

public String readErrMaudeNoBlock(int maxLen)
```

El primero de ellos, nos sirve para mandar un texto a *Maude* y que se ejecute. Los cuatro siguientes corresponden a los métodos de lectura de la respuesta de *Maude*, con sus variantes bloqueantes y no bloqueantes. Cada vez que se invoca a cualquiera de estos cuatro últimos métodos, la cadena devuelta es también escrita a su vez en el fichero de registro de ejecución de *Maude*.

Para ejecutar un comando, así pues, bastará con realizar un *sendToMaude(<COMANDO>)*, y obtener su respuesta con una o sucesivas llamadas a *readOutMaudeNoBlock(<NUM_CHARS>)*, o cualquier otra variante del método de

lectura. Cabe destacar que los métodos no bloqueantes pueden devolver una cadena vacía si *Maude* todavía no ha procesado por completo el comando enviado.

Ésta es una *API* de bajo nivel destinada a la implementación de consolas virtuales de *Maude*, o para que el programador se diseñe su propia *API* de acceso a *Maude*. Por ello se debe ser cuidadoso en su uso, especialmente con los métodos bloqueantes, ya que se puede provocar de forma no deseada la suspensión del proceso *Maude*. Para invocar comandos desde un programa Java en *Maude*, se ha diseñado la *API* de procesado por lotes que no requiere de estas precauciones.

3.3.2. *MaudeProcessBatch*.

El esquema de funcionamiento de esta API es muy simple: todo comando o comandos que se deseen ejecutar en *Maude* son empaquetado en un trabajo (*MaudeJob*) que se envía al proceso *Maude*, que los mantiene en una cola mientras esperan su ejecución. Un trabajo encapsula todos los datos que son relevantes para dicho conjunto de órdenes de *Maude*. Estos son, las órdenes a ejecutar y si el trabajo ha sido procesado ya o no. En caso de que las órdenes ya se hayan ejecutado, almacena también la respuesta de *Maude*, si el comando ha fallado o el número de reescrituras y tiempo de ejecución, si es aplicable.

Este método de interactuar con *Maude* es mucho más sencillo con el objetivo de utilizarlo en un programa Java. La clase *MaudeProcessBatch* incluye los siguientes métodos, que nos permiten, a partir de un *stream* o una cadena de texto, crear uno o varios trabajos:

```
public List<IMaudeJob> createAndRunJobs (InputStream input)
           throws ParseException

public List<IMaudeJob> createAndRunJobs (String input) throws ParseException

public List<IMaudeJob> createJobs (String input) throws ParseException

public List<IMaudeJob> createJobs (InputStream input) throws ParseException

synchronized public void addJob(IMaudeJob job)
```

Los dos primeros crean los trabajos y los añaden directamente en la cola de ejecución de *Maude*. Los dos siguientes métodos únicamente crean los correspondientes trabajos, y los devuelven en una lista. Cuando se desee, pueden ser también añadidos a la cola de ejecución de *Maude* usando el último método proporcionado.

El funcionamiento del proceso *Maude* es asíncrono por defecto, es decir, una vez mandados los trabajos a *Maude*, la ejecución del programa que hace uso de la *API* puede continuar normalmente sin quedarse detenida. Sin embargo, ya que puede resultar útil suspender la ejecución del programa hasta que se obtiene un determinado resultado, se proporcionan sendos métodos para esperar la finalización de los trabajos: tanto la clase *MaudeProcessBatch* como *MaudeJob* proporcionan un método llamado *waitUntilFinish()*. El primer método, perteneciente a *MaudeProcessBatch*, suspende la

ejecución hasta que todos los trabajos de la cola han sido procesados; el segundo, únicamente suspende el programa que lo invoca hasta que el trabajo al que pertenece el método termina.

La gestión de estos trabajos la realizan dos hilos de ejecución (*threads*) encargados de controlar la entrada, salida y salida de error del proceso. Los trabajos son procesados en el orden de llegada a la cola y problema por tanto a tratar es el clásico de productores y consumidores: el primer hilo actúa como productor de comandos *Maude* y el segundo como consumidor de las salidas estándar y error de *Maude* evitando el bloqueo del proceso *Maude*.

El principal problema que encierra la ejecución de los trabajos en *Maude* es la capacidad de poder determinar cuando un trabajo se ha terminado de procesar y se está empezando a procesar el siguiente.

Para código *Full Maude* es una tarea relativamente sencilla, ya que toda orden en *Full Maude* está encerrada entre paréntesis «(», «)». Gracias a esto se puede descomponer fácilmente un conjunto de órdenes en una lista de trabajos simples, por lo que la obtención de una respuesta supone que se ha procesado un trabajo.

En caso de Core Maude la descomposición no es trivial por lo que un único trabajo puede componerse de múltiples órdenes. Una solución a este problema de detectar el final de un trabajo, es realizar la ejecución de un determinado comando para el que podamos reconocer su respuesta, y que no produzca efectos colaterales en la ejecución de las órdenes del usuario. Se ha optado en este caso, ejecutar un orden trivial: una reducción de una expresión lógica.

Así pues, todo trabajo, al ser creado, genera una expresión lógica aleatoria y lo almacena como su *identificador*. Igualmente, se añade a su lista de órdenes a ejecutar la orden «`red ID_TRABAJO .`». De esta forma, y sabiendo que todo trabajo terminará con una reducción de una expresión lógica determinada, el hilo encargado de leer de la salida estándar de *Maude* podrá identificar el final de la ejecución de éste.

Una vez se ha terminado la ejecución de un trabajo, se marca como finalizado, y se procede a la ejecución de los métodos de post-procesado: éstos limpian el resultado devuelto por *Maude* eliminando información innecesaria, y comprobando posibles errores o advertencias.

3.3.3. Las preferencias de Maude.

La configuración de *Maude* dependerá de en qué sistema nos encontremos, y donde se encuentren los ejecutables de *Maude*. Mediante una ventana de preferencias podemos establecer parámetros como la ruta donde se encuentran los ficheros, el modo de ejecución de Maude, la ubicación del registro de ejecución o la plataforma sobre la que se ejecuta el entorno. Esto permite, entre otras cosas, que el usuario la *API* pueda obviar los mecanismos de configuración de *Maude*, siendo un proceso automático. No obstante, se proporcionan los métodos correspondientes para configurarlo de forma manual si así lo desea. La Figura 3 la ventana principal de preferencias de Maude.

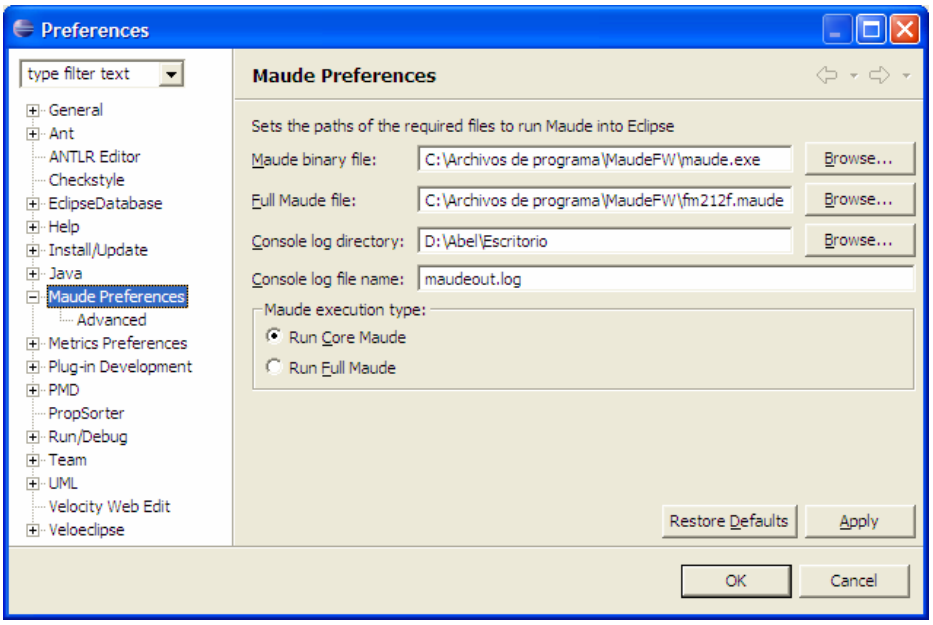

Figura 3. Ventana principal de preferencias de Maude.

También se proporciona una página de propiedades avanzadas donde se puede seleccionar el sistema operativo en que nos encontramos para así ajustar el comando necesario para ejecutar *Maude* según la plataforma. Este valor se ajusta automáticamente para equipos Windows 9X, Windows NT/XP, Mac OS X y Linux; aunque para sistemas no contemplados se puede indicar de forma manual el comando conveniente.

4. Maude SimpleGUI.

Este *plug-in* proporciona una interfaz de trabajo simple y sencilla de manejar pero facilitando toda la funcionalidad básica necesaria. Sus componentes más importantes son un editor de texto (cuyo contenido puede ejecutarse directamente en una instancia de *Maude*), un conjunto de asistentes integrados con los que Eclipse proporciona y una vista de consola que permite controlar la ejecución del proceso Maude.

4.1. Arquitectura del *plug-in Maude SimpleGUI*.

La estructura del *plug-in* está organizada en seis paquetes, agrupando las clases según su funcionalidad. El paquete principal es *es.upv.dsic.issi.moment. MaudeSimpleGUI*. En él se encuentran los datos que serán accedidos por todos los demás paquetes, principalmente, la instancia en ejecución de *Maude*. Igualmente, en este paquete se encuentra definida la clase *Message*, donde se encapsulan todos los posibles mensajes que muestre el *plug-in* al usuario. El resto de paquetes se corresponden con cada uno de los componentes que se relacionan a continuación:

4.1.1. Componentes no visuales.

Maude SimpleGUI es un entorno de desarrollo simple para la creación, prueba y ejecución de programas en *Maude*. Su núcleo básico está formado por el editor, y la consola de *Maude*, que representan respectivamente, la entrada y la salida de *Maude*.

El funcionamiento del *plug-in* es sencillo: cuando éste se inicia se crea una instancia de la clase *Maude* (*es.upv.dsic.issi.moment.MaudeSimpleGUI.core.Maude*), que será el proceso *Maude* asociado a cualquier instancia abierta del editor. La clase *Maude*, a su vez, creará un objeto *MaudeProcessInteract*, perteneciente al *plug-in Maude Daemon* (que debe estar también instalado en Eclipse).

Esta clase –*Maude*–, es la encargada de invocar los métodos del proceso *Maude*, así como de lanzar el hilo de ejecución dirigido a leer las respuestas de *Maude* y mostrarlas por la consola. Igualmente, proporciona una serie de métodos, que serán los utilizados en el *plug-in* para interactuar con *Maude*, evitando invocar los propios de la clase *MaudeProcessInteract*.

4.1.2. La consola de Maude.

La consola de *Maude* es el elemento central del *plug-in* y nos permite controlar la ejecución y ver la respuesta devuelta por *Maude*. Se ha implementado como una vista independiente y puede activarse mediante un acceso directo de la barra de herramientas del editor, o mediante el menú de vistas de Eclipse. Consta de cuatro partes fundamentales: una barra de herramientas, una barra de estado, el panel de la consola y la línea de introducción de comandos breves (Figura 4).

Figura 4. Vista de la consola de Maude.

La barra de herramientas, permite realizar las diferentes acciones sobre el proceso, así como otras operaciones concernientes a la consola. Entendemos por controlar *Maude* el poder iniciar un proceso *Maude* (4) —tal y como esté especificado en las preferencias

del *Maude Daemon*—, detenerlo (5), matarlo (6), reiniciar el loop de Full *Maude* (7), o activar el modo traza (8). En cuanto a las opciones del panel de consola, encontramos el borrado de la consola (1), el control de autopaginado (2) o la ocultación de los comandos enviados (3).

La barra de estado muestra información acerca del proceso (preparado, parado, en ejecución…) así como el tiempo real que se ha tardado en ejecutar el último conjunto de operaciones enviado a *Maude*. Por otra parte, el panel de la consola muestra los mensajes en tres colores diferentes: negro, verde y rojo. El *prompt* se imprime en color verde, las respuestas por la salida estándar en negro, y los mensajes de error o alertas (salida de error), en color rojo. También se han implementado las funciones de copiado, cortado, pegado y búsqueda convencionales.

Los último, la línea de introducción de comandos breves nos permite enviar comandos escribiéndolos directamente en ella, tal y como se hace de forma habitual en *Maude* desde una consola. Proporciona un historial de todos los comandos que se han enviado a *Maude* que se puede navegar mediante las teclas «↑» y «↓».

4.1.3. Los asistentes.

Se han implementado dos asistentes, integrados en el menú para crear un nuevo documento de Eclipse, que nos permiten generar un fichero vacío con extensión «*.maude*» o «*.fm*». Una vez creados se abrirán con el editor de código *Maude* automáticamente.

4.1.4. El editor.

El editor es el elemento que nos permite escribir programas en *Maude* así como interactuar con él, enviándole éstos programas. Consta de una ventana con un editor de texto convencional, una barra de herramientas, y un menú en la barra de menús.

El editor proporciona capacidades de coloreando sintaxis, además de mantener completamente operativas las teclas de acceso rápido de los editores de Eclipse para facilitar la escritura de programas. La interfaz que se proporciona al usuario para interactuar con el proceso *Maude* es la habitual, mediante menús, barras de herramientas y atajos de teclado. Principalmente nos permite el envío de texto a *Maude*, ya sea todo el contenido del editor o únicamente la selección. Igualmente proporciona un acceso directo para mostrar la consola de *Maude*.

4.1.5. Envío directo de ficheros.

El envío directo de ficheros es una opción útil cuando se desea mandar numerosos ficheros a la vez a *Maude*. Bastará con seleccionarlos de la vista de carpetas de Eclipse, y pulsando el botón derecho del ratón sobre ellos, seleccionar «*Send to Maude*», del menú «*Maude*». Los archivos se enviarán a *Maude* en orden alfabético, por lo que una

buena opción es prefijarlos con un número, marcando el orden en que deben ser cargados.

5. Trabajos relacionados.

El entorno de programación que se proporciona desde el proyecto *Maude* para su lenguaje es sin duda su principal debilidad, ya que se reduce a un terminal donde se escribirán los comandos.

Se han realizado diversos esfuerzos para mejorar la productividad en el desarrollo de programas en *Maude*, por ejemplo, existen módulos para integrar el uso de *Maude* en XEmacs [Brü05], con la principal desventaja de que no todo el mundo está familiarizado con sistemas Linux y más aún, con un completo conocimiento de XEmacs.

Otras aproximaciones han pretendido proporcionar un entorno de desarrollo para *Maude* mucho más sencillo, y orientado a ejecutarse en sistemas de ventanas. Una de ellas es *Maude Workstation* [Muñoz04].

Maude Workstation guarda ciertas similitudes con las «Herramientas de Desarrollo de *Maude*» (*Maude Development Tools*). Ambos entornos son portables dado que se han implementado en Java, y se aprovechan de la posibilidad de ejecutar *Maude* sobre Windows gracias a Cygwin. De igual forma, ambos siguen la misma filosofía de funcionamiento lanzando el proceso en Java, y capturando los *buffers* de entrada y salida.

A pesar de estas similitudes, también existen numerosas diferencias. En primer lugar, uno de los objetivos de *Maude SimpleGUI* era proporcionar un conjunto de herramientas mínimo y simple que permitieran trabajar de forma cómoda con Maude. Es por ello no proporciona ciertas funciones que sí lo hace *Maude Workstation*, como la integración de opciones de *debug* en la interfaz de usuario, o el mantenimiento de una base de datos en memoria de los módulos cargados y su jerarquía de dependencias.

Por otra parte, como contrapartida, las *Herramientas de Desarrollo de Maude* proporcionan un diseño modular, que diferencia en dos *plug-in*s independientes la *API* al programador, y el editor del usuario, permitiendo una mayor mantenibilidad y extensibilidad.

También, al margen de cuestiones de reutilización del código, la principal ventaja obtenida con el editor *Maude SimpleGUI* es su eficiencia. Realizando diversas pruebas[1], se ha comprobado que el envío de un único fichero de *Maude* (de aproximadamente 45KB) con la especificación de diversos módulos para cargar, tarda aproximadamente 15 segundos con *Maude Workstation*, siento el tiempo de carga inapreciable (menor que un segundo), con el *plug-in* desarrollado para Eclipse.

Por otra parte, las *Maude Development Tools*¸ son mucho más flexibles y sencillas de instalar que *Maude Workstation*. En el contexto de este trabajo se ha desarrollado un

[1] En un Pentium 4 Hyper-Threading / 3GHz / 1 GB de memoria RAM.

instalador —disponible en [MOMENT]— que no requiere de ningún software adicional, y permite configurar un sistema *Windows* para ejecutar *Maude* en segundos.

Por último cabe comentar la notable robustez de las *Herramientas de Desarrollo de Maude.* En este sentido, se aprovecha la estabilidad de Eclipse, ya que se reutilizan componentes que el entorno nos proporciona, así como la experiencia obtenida del estudio de la implementación de *Maude Workstation*.

6. Conclusiones.

En este trabajo se han implementado los mecanismos necesarios para integrar una herramienta formal en un entorno industrial con éxito, proporcionando un mecanismo de instalación, configuración y uso sencillo para el usuario, facilitando el acercamiento al uso de sistemas formales en Ingeniería del Software.

Prueba de ello es que se ha conseguido proporcionar un sistema de instalación automático de *Maude* sobre un sistema Windows, siendo un proceso para el usuario extremadamente simple, —algo impensable hasta el momento—. Esto, junto al *plug-in* de Eclipse *Maude SimpleGUI* permite disponer en pocos segundos de un entorno de desarrollo de programas en *Maude* robusto y fácil de usar en cualquier sistema operativo como se ha comprobado en sistemas Windows, Linux y Mac OS X. Este entorno, por otra parte es altamente eficiente comparado con las escasas soluciones que se han proporcionado al respecto, proporcionando una amigable interfaz gráfica.

Por último, la *API* proporcionada al programador es sencilla de utilizar, potente, y totalmente funcional. Prueba de esto es el uso exitoso que de ella hace la herramienta MOMENT [MOMENT]. MOMENT es una herramienta de Gestión de Modelos [Ber00] que utiliza el entorno *Maude* desde Eclipse Modeling Framework [EMF] gracias a las «*Herramientas de Desarrollo de Maude*» [BoI05]. En MOMENT se definen algebraicamente un conjunto de operadores que permiten tratar con modelos. Estos modelos se especifican como conjuntos de elementos de forma independiente del metamodelo, de manera que los operadores pueden acceder a los elementos sin conocer la representación de un modelo.

MOMENT por tanto, se sirve de la *API* descrita en este trabajo para implementar los puentes tecnológicos [Ibo05] entre *Maude* y *EMF* permitiendo que su interfaz esté integrada en EMF, de manera que el formalismo de especificaciones algebraicas quede totalmente transparente al usuario.

Agradecimientos.

A Francisco Durán, por facilitarnos la documentación y la implementación de *Maude Workstation*, en el que nos hemos basado para ejecutar *Maude* sobre *Windows*.

Referencias

[ANTLR] ANTLR plug-in for Eclipse. http://antlreclipse.sourceforge.net/

[Ber00] Bernstein, P.A., Levy, A.Y., Pottinger, R.A., «A Vision for Management of Complex Models». Microsoft Research Technical Report MSR-TR-2000-53. Junio 2000. SIGMOD'00. Diciembre 2000.

[BoCR05] Boronat, A., Carsí, J.Á., Ramos, I. «Automatic Reengineering in MDA Using Rewriting Logic as Transformation Engine». IEEE Computer Society Press. 9th European Conference on Software Maintenance and Reengineering. Manchester, Reino Unido. 2005.

[BoI05] Boronat A., Iborra J., Carsí J. Á., Ramos I., Gómez A. «Del método formal a la aplicación industrial en Gestión de Modelos: Maude aplicado a Eclipse Modeling Framework». JISBD'05. Septiembre 2005.

[Brü05] Brünnler, K. Software Section. Abril 2005. http://www.iam.unibe.ch/~kai/

[Cygwin] RedHat, Inc. «Cygwin Information and Installation». http://www.cygwin.com.

[EclOv03] Object Technology International, Inc. «Eclipse Platform Technical Overview», Febrero 2003. http://www.eclipse.org/whitepapers/eclipse-overview.pdf

[Eclipse] Eclipse Project. http://www.eclipse.org

[EMF] Eclipse Tools. EMF home. http://www.eclipse.org/emf/

[Ibo05] Iborra, J. «Prototipo de integración de una herramienta de Gestión de Modelos». Proyecto final de carrera. Universidad Politécnica de Valencia. Septiembre 2005.

[Maude] Department of Computer Science, University of Illinois, Urbana-Champaign. The Maude System. http://maude.cs.uiuc.edu/.

[MOMENT] MOMENT Website. «MOMENT: A framework for MOdel manageMENT.». Junio 2005. http://moment.dsic.upv.es:8080

[Muñoz04] Muñoz A., «Maude Workstation: Entorno de programación para el lenguaje de especificación algebraica Maude». Proyecto final de carrera. Universidad de Málaga. Julio 2004.