Documentación
Proyecto de Grado

Facultad de Ingeniería - Universidad de la República

Coop
plug-in de Coq para Eclipse

Fecha: 29/04/2011

Versión: 1.3

Responsables: Milagros Betancor, Daniel Peláez, Manuel Torterolo

Supervisores: Daniel Calegari, Carlos Luna

Índice General

Resumen

El objetivo principal de este proyecto es la obtención de una herramienta que oficie de entorno de desarrollo para el asistente de pruebas Coq y que a su vez sea integrable con algún entorno pre existente. Se optó por usar Eclipse como entorno huésped debido a su arquitectura fácilmente extensible y su gran popularidad entre los miembros de la comunidad de desarrolladores de software. Estas propiedades son causales del gran número de plugins que existen para este entorno de desarrollo. Esto, sin lugar a dudas, suma otro punto a favor para su elección, ya que abre las puertas a la integración de esta nueva herramienta con una gran gama de plugins para Eclipse.

Para el desarrollo de esta nueva herramienta fue necesario relevar todas aquellas herramientas que cumplieran, en algún sentido, con las necesidades planteadas, con el objetivo de conocer la oferta existente y así poder confeccionar la lista de requerimientos que debería implementar la herramienta, que hemos dado a llamar Coop.

Coop no es solo una acumulación de funcionalidades soportadas por otras herramientas, sino que incorpora varias funcionalidades innovadoras como resultado de sugerencias realizadas por usuarios expertos en el uso de Coq. Entre estas funcionalidades se encuentra la ejecución independientemente de bloques específicos de código, la construcción de un árbol de derivación a medida que avanza una prueba y su exportación a diferentes formatos, la generación de un grafo de dependencias del código, la posibilidad de navegar a través del código en base a palabras clave, entre otras. Estas funcionalidades hacen de Coop un entorno innovador de desarrollo para Coq en Eclipse, amigable y con gran poder de asistencia al usuario en pro de facilitar el uso de Coq.

Palabras clave: *Coq, IDE, Eclipse, Plug-in*.

Capítulo 1

Introducción

Un Entorno de Desarrollo Integrado (Integrated Development Environment, IDE) es una herramienta que asiste a los desarrolladores de software en la visualización y manipulación del código. Eclipse [5, 27], en tanto, es uno de los IDE's más populares al día de hoy, en consecuencia se cuenta con una gran variedad de herramientas disponibles que lo extienden y con una amplia documentación e información inherente a estas y a Eclipse. Por su parte, Coq [1, 6, 7, 28] es un sistema de gestión de pruebas formales que provee de un lenguaje formal para especificar definiciones matemáticas, algoritmos ejecutables y teoremas, junto con un entorno para el desarrollo de pruebas semi-interactivas verificadas por computadora.

Coq se vería beneficiado de la existencia de un IDE específico para su uso. De hecho, hoy día existen varios IDE's, destacándose las herramientas ProverEditor[2], ProofGeneral[3, 8, 15] y CoqIDE[4]. Sin embargo, algunos de estos IDE's solo poseen funcionalidades básicas para la manipulación del código y la ejecución de las pruebas. El objetivo de este proyecto consiste en lograr un IDE, denominado Coop (gallinero, en inglés), que integre el asistente de pruebas Coq en el entorno de desarrollo Eclipse, y que aumente la capacidad de asistencia que ofrecen otros IDE's existentes.

1.1 Contexto

Este proyecto se enmarca en el trabajo de investigación del grupo COAL [18] y del grupo de Métodos Formales del InCo [19]. Entre los objetivos de estos se incluye el estudio de técnicas para el desarrollo de sistemas orientados a objetos guiados por modelos y la aplicación de métodos formales para garantizar corrección en el proceso. Asimismo, este trabajo se enmarca dentro de una línea de investigación reciente que busca integrar grupos de investigación en la región, en particular: de la Universidad de la República y de la Universidad ORT Uruguay.

El Desarrollo de Software Guiado por Modelos (Model-Driven Development [30], MDD) es un enfoque de ingeniería de software basado en el modelado de un sistema como la principal actividad del desarrollo y la construcción del mismo guiada por transformaciones de dichos modelos. Su éxito depende fuertemente de la disponibilidad de lenguajes y herramientas apropiados para realizar las transformaciones entre modelos y validar su corrección.

El proyecto de investigación "Verificación de Transformaciones de Modelos de Comportamiento Basados en UML" es un proyecto financiado por ANII (Agencia Nacional de Investigación e Innovación) cuyo objetivo es contribuir a la especificación formal de transformaciones de modelos especificados en UML, con el objetivo de hacer posible la verificación

de la corrección sintáctica y semántica de dichas transformaciones. En este contexto se está experimentando con herramientas de transformación de modelos ejecutadas en el entorno Eclipse (ATL, QVT) y el uso del asistente de pruebas Coq para la verificación de las transformaciones.

A los efectos de lograr un entorno de trabajo unificado, el objetivo principal de este proyecto de grado es la construcción de un IDE para el asistente de pruebas Coq integrable a modo de plug-in con el entorno Eclipse. En este sentido se investiga la existencia de entornos de ejecución del asistente de pruebas Coq en el entorno Eclipse, para desarrollar un entorno nuevo haciendo uso de los beneficios de la plataforma, para uso general y en particular, en un futuro no muy lejano, integrar dicho entorno con las herramientas de transformación de modelos ya existentes, utilizando el producto desarrollado por un proyecto de grado previo.

1.2 Objetivo

Una vez relevada la oferta de entornos que permitieran la integración del asistente de pruebas Coq con Eclipse y de otros independientes a él, se hace evidente la necesidad de desarrollar un nuevo entorno que cumpla con las expectativas previstas, por lo que los esfuerzos, se concentran en concebir un nuevo entorno integrable con Eclipse a modo de plug-in para Coq. Se pretende que el plug-in herede las principales características presentes en los IDE's pre-existentes, así como nuevas funcionalidades ideadas y sugeridas, tanto por el equipo de desarrollo, como por los tutores de este proyecto y por usuarios experimentados en el uso de los distintos IDE's y del asistente Coq en general [Ver anexo: *Documento de requerimientos Coop*].

Es entonces el objetivo de este proyecto integrar una herramienta con características de IDE con el asistente de pruebas Coq en el entorno Eclipse. Esto significa desarrollar un entorno nuevo haciendo uso de los beneficios de la plataforma.

1.3 Estructura del documento

El resto del documento está estructurado de la siguiente forma. En el Capítulo 2 se presentan algunos conceptos previos. En particular se explica brevemente en qué consiste Eclipse, el asistente de pruebas Coq y cómo se integran aplicaciones a él.

En el Capítulo 3 se desarrolla una breve descripción de las herramientas relevadas.

En el Capítulo 4 se introduce la herramienta construida. Se describen los requerimientos que se obtuvieron tras el análisis de las herramientas existentes y de las sugerencias de usuarios expertos en estas últimas, además de los propuestos por el equipo de desarrollo y de los tutores de este proyecto. También se presentan los requerimientos no funcionales, de los que claramente se destaca la necesidad de que Coop sea un plug-in de Eclipse.

En el Capítulo 5 se explica el diseño de la herramienta construida, cómo se configura, cómo se acopla a Eclipse, los problemas conocidos y cómo se extiende. Además, se detalla la implementación de los principales requerimientos, así como aquellos requerimientos que no fueron implementados y quedan como trabajo futuro.

Finalmente, en el Capítulo 6 se concluye con el análisis de la herramienta resultante, Coop, y su comparación con las herramientas referentes.

Adicionalmente, se presentan anexos que contienen: manual de usuario, manual de instalación, documentación técnica, relevamiento de herramientas relativas a Coq, cronograma del proyecto y documento de requerimientos.

Capítulo 2

Conceptos Preliminares

Para llevar adelante este proyecto, se realiza la lectura e investigación de varias herramientas y tecnologías fundamentales para cumplir con los objetivos planteados. En este capítulo se introducirán algunos conceptos que permitirán una mejor comprensión del resto del documento.

2.1 Coq

El asistente de pruebas Coq es una implementación del cálculo de construcciones inductivas, una lógica intuicionista de alto orden con tipos dependientes y tipos inductivos como objetos primitivos [9,10]. El usuario introduce definiciones y hace demostraciones en un estilo de deducción natural, las cuales son chequeadas mecánicamente por el sistema.

Dicho formalismo permite especificar y probar en lógica intuicionista de alto orden. Esta lógica asocia una interpretación computacional a las pruebas, la noción de veracidad de una proposición corresponde a la existencia de una prueba.

En el proceso de prueba de un teorema, Coq entra en un ciclo interactivo donde el usuario completa la demostración usando tácticas, las cuales implementan reglas de inferencia o de tipado (esquemas de prueba). El conjunto de tácticas puede ser incrementado por el usuario, a partir de un lenguaje diseñado para tal fin.

Entre las funcionalidades que incluye Coq como asistente de programación, podemos citar:

- **Definición de tipos inductivos**. Es posible definir relaciones y tipos de datos inductivos, que especifican construcciones matemáticas concretas usadas en computación, como por ejemplo: desigualdades, predicados de pertenencia y especificaciones en general, definidas por casos o inducción. Una vez definidos los tipos de datos se pueden definir funciones, recursivas o no, y especificar propiedades sobre estos tipos, como así también probar dichas propiedades, eventualmente por inducción. La teoría de tipos que implementa Coq permite programar usando tanto recursión primitiva como recursión general y por consiguiente, probar propiedades usando inducción primitiva ó inducción completa. En este último caso definiendo órdenes bien fundados.

- **Construcción y verificación de programas funcionales**. Es posible especificar sistemas y extraer/ derivar/sintetizar programas funcionales a partir de pruebas en distintos lenguajes de programación funcional, como por ejemplo Haskell. Asimismo, es posible probar la corrección de programas con respecto a una especificación dada, esto es, realizar verificación formal de programas. Esta última es una de las principales áreas de aplicación de Coq, no sólo en ámbitos estrictamente académicos.
- **Definición de tipos con objetos infinitos.** Coq permite definir tipos de datos que pueden contener objetos no bien fundados, como por ejemplo secuencias infinitas o streams, y modelar a través de estos tipos sistemas complejos, tales como: sistemas reactivos (que se comportan como una secuencia de estímulos-respuestas) y de tiempo real (sistemas reactivos cuya corrección depende de la magnitud de los retardos temporales).

2.2 Eclipse

Eclipse es un IDE bajo licencia open source. Cuenta con la gran ventaja de ser independiente de la plataforma sobre la cual se ejecuta, entre las que se encuentran: Windows, Mac y Linux. Esta característica es heredada de Java, el lenguaje con el cual fue y aún es implementado. Otra de sus grandes ventajas es su diseño versátil que brinda gran flexibilidad para ser extendido a través de una enorme variedad de plug-ins disponibles también, en su mayoría, bajo licencia Open source o a través de plug-ins desarrollados por uno mismo.

Eclipse cuenta con un diseño extensible que soporta variados lenguajes de programación y plug-ins. Si bien no fue concebido para un determinado lenguaje de programación, Java ha ocupado un lugar preponderante desde un comienzo, ya que es el lenguaje preferido por muchos de los miembros de la comunidad de desarrolladores de software.

Este IDE permite a los usuarios desarrollar proyectos de diferentes tamaños y complejidades en una variedad de lenguajes con todas las facilidades que un entorno de desarrollo debe brindar. Además Eclipse soporta la integración de plug-ins e implementa módulos con el fin de proporcionar todas sus funcionalidades en el front-end de su plataforma. Los plug-ins, entre otras cosas, permiten que Eclipse pueda interactuar con diversas aplicaciones externas. Un ejemplo claro de esto será Coop que permitirá la interacción con el motor de Coq.

2.3 ¿Qué es un Plug-in?

A la hora de implementar un plug-in para Eclipse esta pregunta surge inmediatamente. Evidentemente es necesario saber que es un plug-in y cuál es su utilidad para poder implementar uno.

Un plug-in es una pieza independiente de software, un programa en sí mismo, capaz de interactuar con otros programas para proporcionar una funcionalidad específica.

Los plug-ins cuentan con un conjunto bien definido de acciones. Su principal función es mejorar la forma en que los usuarios ven e interactúan con las distintas aplicaciones para las cuales fueron creados, ya sea implementando una interfaz más amigable, mejorando funcionalidades ya existentes, implementando nuevas funcionalidades o la combinación de estas opciones. Sin lugar a duda esto es uno de los principales objetivos de este proyecto. Es de gran utilidad contar con una herramienta que provea a los desarrolladores una sencilla interfaz para realizar pruebas de teoremas matemáticos o de alguna otra índole, como la verificación de transformaciones entre

modelos (objetivo en el que se enmarca este proyecto), en un entorno de desarrollo popular como lo es Eclipse.

Capítulo 3

Relevamiento de Herramientas

En este capítulo se hace un breve resumen de la investigación llevada a cabo sobre los IDE's existentes para Coq. El relevamiento se expone de forma completa en el anexo *Relevamiento de Herramientas Coop*. En esta sección se presenta un pantallazo de las principales características de las herramientas más relevantes que se exploraron al comienzo del proyecto, para definir las principales características del mismo. Entre las herramientas relevadas estuvieron: CoqIDE, PCoq [11, 12], ProverEditor, ProofWeb [13, 14] y ProofGeneral, entre otras.

3.1 CoqIDE (Coq Integrated Development Environment)

CoqIDE es una herramienta gráfica, que es usada como interfaz amigable para el usuario de coqtop [29]. Su principal propósito es permitir al usuario navegar hacia adelante y hacia atrás en un archivo vernacular Coq, ejecutando los comandos correspondientes o deshaciéndolos respectivamente.

Acepta los mismos comandos que coqtop. La barra de herramientas ofrece cinco comandos básicos para ejecutar el buffer.

Ofrece el comando ProofWizard para automatizar pruebas. También maneja templates.

Por último este IDE ofrece la opción de hacer consultas (*Query*).

3.2 PCoq: Interfaz Gráfica de Usuario para Coq

PCoq provee un entorno de trabajo para el probador de teoremas Coq. El mismo tiene las siguientes características:

- Desarrollado en Java.
- Interfaz gráfica: múltiples fuentes y colores son usados para mostrar las fórmulas y los comandos.
- Separación entre la interfaz y el proveedor: la interfaz gráfica y Coq corren sobre dos procesos separados. Los usuarios deben elegir sobre cuál máquina correr Coq.
- Manejo de estructuras y mecanismos de presentación: el entorno provee formas de editar las estructuras de las fórmulas y los comandos. Se pueden agregar de manera sencilla nuevas notaciones.

En comparación con CoqIDE, no ofrece muchas funcionalidades adicionales. Pero a diferencia del CoqIDE ofrece una funcionalidad interesante que consta de un parser de la interfaz gráfica que transforma las pruebas que el usuario va escribiendo a notación matemática y las despliega en otra ventana.

3.3 ProverEditor

Este editor es usado para editar interactivamente teoremas en archivos desde Eclipse. El mismo es usado con Coq, y en un futuro será usado con PVS y otros probadores de teoremas. Este editor es fácil de extender y consta de un pequeño conjunto de comandos.

Este IDE ofrece tres vistas, la del editor del archivo sobre el que se trabaja, la de Top Level que muestra las salidas de las ejecuciones, y la del Top Level que muestra la estructura de todas las variables, teoremas, lemas, etc. que se han declarado en el archivo sobre el que se trabaja.

ProverEditor ofrece los siguientes comandos:

- Avanzar en la prueba de a una línea.
- Resetear la prueba.
- Retroceder en la prueba de a una línea deshaciendo lo hecho en la misma.
- Ejecutar desde la posición actual (de la ejecución) hasta el final del archivo.
- Deshacer todo lo ejecutado hasta el momento, volviendo el cursor al comienzo del archivo.
- Cancelar la ejecución actual del archivo.
- Ctrl+K: borra toda la línea (como en Emacs).
- F3: encuentra y pinta la definición de la palabra seleccionada.

3.4 ProofWeb

ProofWeb es a la vez, un sistema para la enseñanza de lógica y para el uso de asistentes de prueba a través de la web.

Es un sistema para practicar deducciones naturales en la computadora. Está basado en el asistente de pruebas Coq, y corre dentro de varios navegadores modernos. Para usar ProofWeb el usuario no debe instalarse un software localmente, no es un plug-in, un navegador es lo único que se necesita. El usuario ProofWeb habla con un sistema Coq en el servidor.

Este IDE fue creado para ser usado con Coq, pero la interfaz puede ser usada sobre otros asistentes de prueba, y particularmente soporta el sistema Isabelle.

La interfaz de ProofWeb puede ser usada y extendida en varios proyectos.

ProofWeb tiene varias funcionalidades en común con CoqIDE:

- Avanzar en ejecución un paso
- Deshacer ejecución de un paso
- Avanzar en ejecución hasta el cursor
- Ejecutar todo hasta el final del archivo
- Deshacer toda la ejecución
- Templates
- Query

- Help

Y a diferencia de CoqIDE, ProofWeb ofrece las siguientes funcionalidades:

- Ejecutar hasta el punto.
- Display, pero no con la misma finalidad del CoqIDE. Este display es para determinar si se quieren ver o no, y de qué manera, el estado de las pruebas, ya que existen tres maneras posibles de visualizarlas. También permite mostrarlas en una ventana independiente.

3.5 Proof General

Es una herramienta genérica para asistentes de pruebas interactivos (también conocida como probador de teoremas interactivo) basado en el editor de texto Emacs [21].

Es la mejor herramienta para Emacs, ya que tiene muchas características y es una interfaz muy rica, que aprovecha la potencia y la flexibilidad de Emacs. Irónicamente, su dependencia de Emacs es uno de sus principales inconvenientes, ya que, en los últimos años, IDE's como Eclipse están reemplazando las herramientas como Emacs por los entornos estándar para el desarrollo de software.

Un script de prueba es una secuencia de comandos enviadas al asistente de prueba para construir una prueba. Proof General utiliza una técnica llamada Script Management para ayudar al usuario a escribir un script de una prueba sin tener que hacer uso de los comandos Cut-and-paste o utilizando repetidamente el "load file". También tiene una sofisticada implementación de Script Management, el cual cubre grandes desarrollos distribuidos sobre múltiples archivos.

Emacs tiene muchas ventajas y está disponible en la mayoría de plataformas, incluyendo Unix, Linux y NT. A pesar de que una vez tuvo la reputación de ser difícil de aprender, las versiones modernas de Emacs son muy fáciles de usar, y da apoyo a toda la gama de actuales tecnologías de interfaz gráfica de usuario además de proporcionar mecanismos fáciles para personalizar.

Otro importante aspecto de Proof General es que es genérico, provee una interfaz uniforme y mecanismos interactivos para diferentes back-end's de asistentes de prueba. Explota las profundas semejanzas entre los sistemas ocultando algunas de sus diferencias superficiales.

Capítulo 4

La Herramienta

4.1 Descripción de Coop

Desde esta sección en adelante se presentará la herramienta construida, comenzando con una breve descripción de la misma, para seguir luego con las características deseables, extraídas en gran medida del relevamiento presentado en el capítulo anterior.

En los objetivos de este proyecto se plantean básicamente las principales características que se espera tenga el producto final. Además de ser una herramienta fácilmente extensible y configurable (a través de archivos de configuración, xml's, etc.) debe brindar al usuario el mayor conjunto posible de características disponibles en IDE's u otras herramientas que sirvan como asistentes en el uso de Coq.

Del relevamiento planteado en el capítulo cuatro se extrajo la base de requerimientos siguiendo criterios propios y otros tomados de [16] y [17]. Sin embargo, la herramienta no debe ser solo un compilado de funcionalidades pre-existente, sino contar con una gama de nuevas características que se consideran relevantes.

En las secciones venideras se presentan los requerimientos no funcionales y funcionales acompañados, más adelante, de imágenes que pretenden ilustrar algunos de ellos y permiten tener una visión más completa de como luce Coop.

4.2 Requerimientos no Funcionales

En esta sección se enumeran los requerimientos no funcionales bajo los cuales se diseñó y desarrolló Coop.

Plug-in

El sistema debe integrarse a Eclipse a modo de plug-in.

Lenguaje

El sistema debe implementarse utilizando el lenguaje de programación Java.

Idiomas

El idioma de la interfaz gráfica debe ser un parámetro de configuración, permitiendo agregar a futuro nuevos idiomas. Por el momento los idiomas soportados deben ser inglés y español.

Versión de Coq

La herramienta a construir debe ser los más independiente posible de la versión de Coq para la cual se utiliza, además de ser fácilmente adaptable a los cambios introducidos por una nueva versión.

Versión de Eclipse

La herramienta a construir debe ser los más independiente posible de la versión de Eclipse a la cual se integra, además de ser fácilmente adaptable a los cambios introducidos por una nueva versión.

4.3 Funcionalidades

En esta sección se enumeran y describen las funcionalidades que hacen de Coop un IDE innovador, con características que no están presentes en los productos disponibles del área. La lista completa de funcionalidades junto con la correspondiente descripción de todos los requerimientos incluyendo los que quedaron por fuera del alcance de este proyecto se encuentran en el anexo *Documento de Requerimientos Coop*.

Coop cuenta con una perspectiva propia dentro de Eclipse desde la cual el usuario accederá a las distintas funcionalidades que ofrece la herramienta.

La Fig. 1 muestra como se ve la perspectiva asociada a Coop dentro de Eclipse.

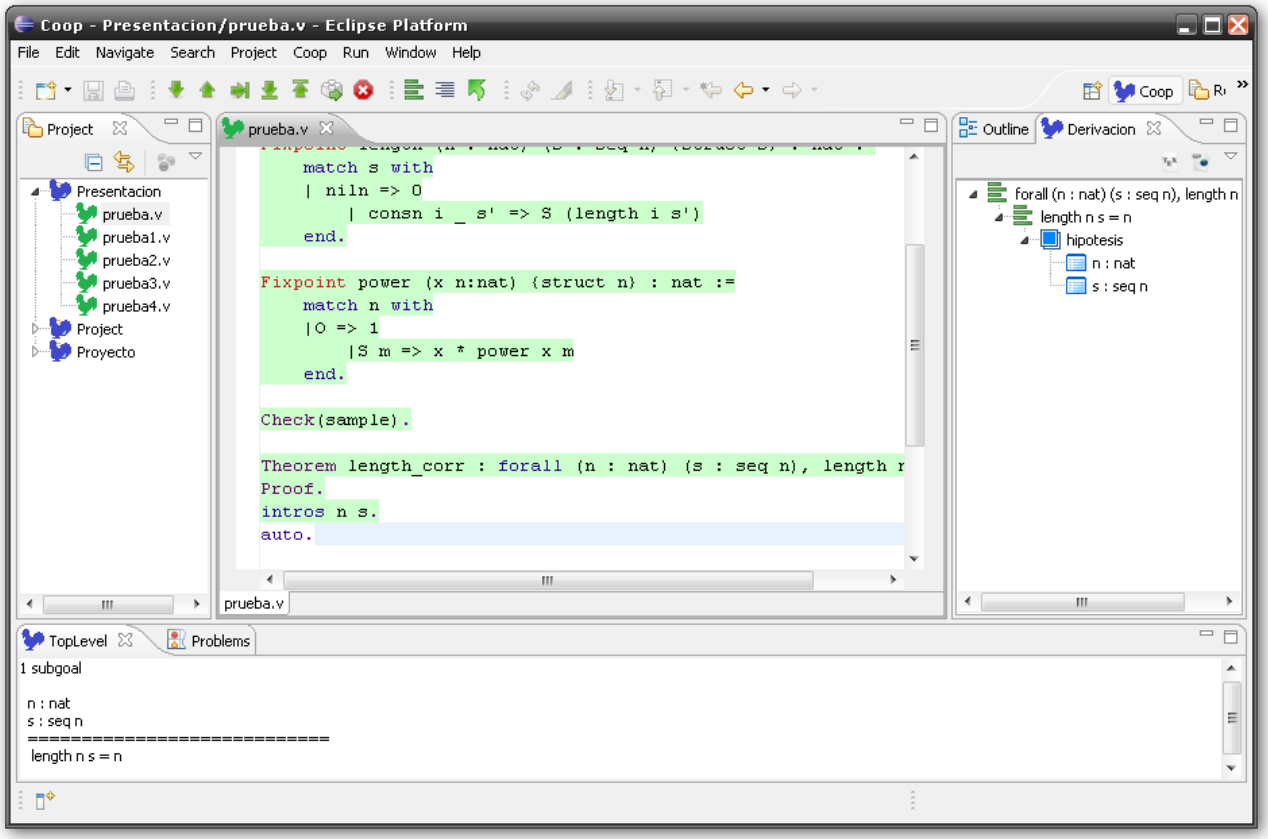

Fig. 1: Perspectiva Coop.

En la Fig. 1 podemos observar que la perspectiva cuenta con algunas vistas. Por ejemplo, en la parte inferior de la imagen se encuentra la vista del TopLevel. Esta vista es la encargada de mostrar al usuario los resultados obtenidos en las sucesivas ejecuciones realizadas sobre el código en cuestión. Sobre la izquierda se encuentra la vista de proyectos, donde se muestran los proyectos Coop que residen en el workspace del usuario. Sobre la derecha tenemos la vista de outline donde se muestran en forma de árbol las distintas estructuras definidas en el código y de derivación. Más adelante en esta sección se describen las funcionalidades ofrecidas por estas dos vistas.

Existen algunos requerimientos fundamentales con los que Coop debe contar para poder considerarse un IDE. Lógicamente el usuario tiene la opción de crear un nuevo proyecto Coq, donde una vez creado el proyecto se hará visible la perspectiva asociada a Coop. Además, el usuario puede crear un nuevo archivo Coq (.v) y de igual manera que con la creación de un proyecto Coq, inmediatamente se habilitará la perspectiva de Coop y se mostrará el nuevo archivo en el editor de archivos.

La ejecución de los archivos implica un conjunto de requerimientos o funcionalidades que debe proveer, y así lo hace, Coop. El usuario puede ejecutar instrucción por instrucción, deshacer la ejecución de la última instrucción ejecutada, ejecutar todo un archivo, deshacer la ejecución de todo un archivo, ejecutar todas las instrucciones desde la última ejecutada hasta la instrucción donde se encuentra posado el cursor, además de contar con la posibilidad, innovadora, de ejecutar el conjunto de instrucciones que se encuentran seleccionadas. Esta última funcionalidad se la denomina ejecución por bloques.

La siguiente imagen (Fig. 2) muestra como se ve el requerimiento de ejecución en Coop:

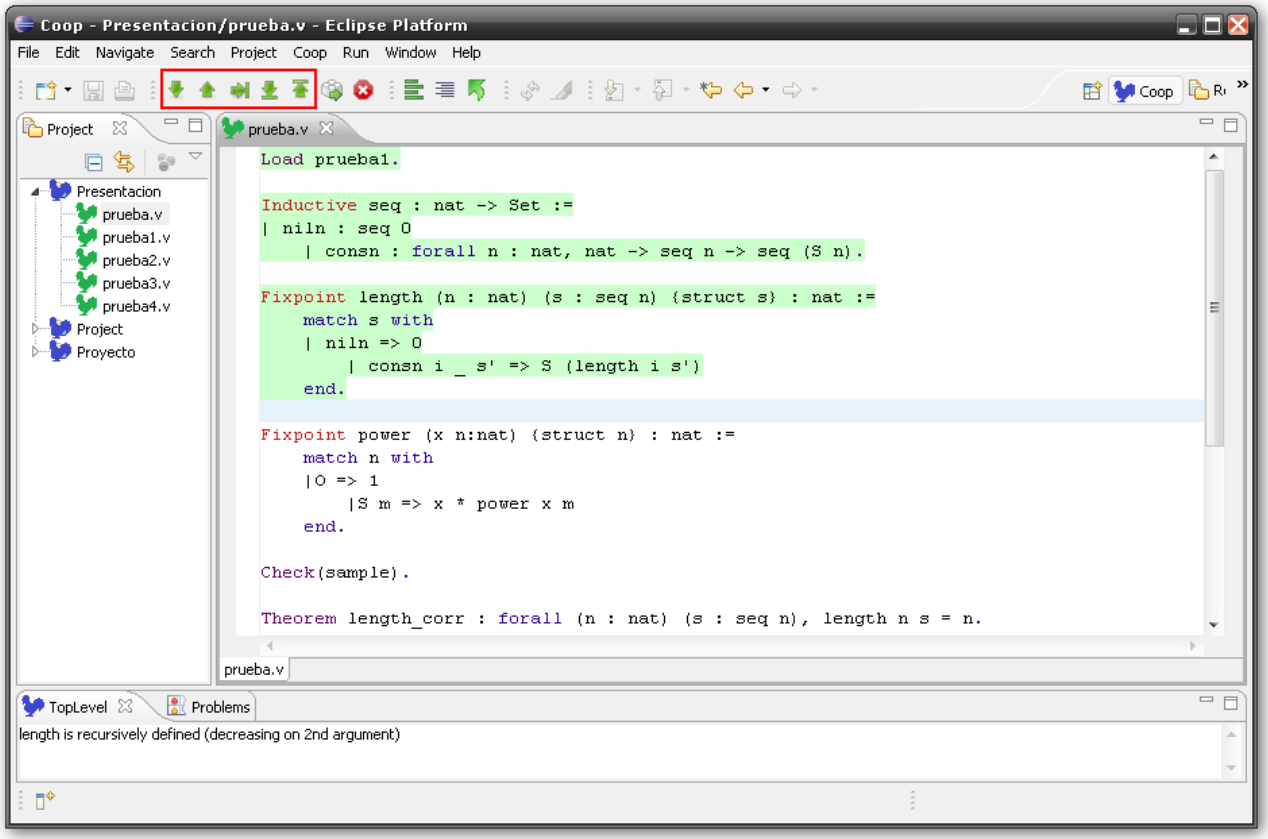

Fig. 2: Ejecución

En la ejecución hacia adelante el usuario puede ejecutar paso a paso (de a una instrucción) las instrucciones desde el comienzo del archivo Coq que se está manipulando. La ejecución hacia atrás es análoga, pero en lugar de ejecutar la siguiente línea el usuario deshace el efecto producido por la ejecución de la última instrucción. Las ejecuciones hasta el final del archivo, así como, la ejecución hasta el comienzo del archivo (o mejor dicho, deshacer todas las ejecuciones) son similares a las anteriores aunque ejecutan todas las instrucciones a partir de la última ejecutada (o deshacen todas las ejecuciones realizadas previamente).

La ejecución hasta el cursor se puede dividir en dos casos. Uno de ellos es aquel en el que el cursor se encuentra en una línea posterior a la línea de la última instrucción ejecutada, mientras que el segundo caso es el contrario, en este el cursor se encuentra en una línea anterior a la última línea ejecutada. En el primero de estos casos se ejecutan todas las instrucciones a partir de la última instrucción ejecutada, hasta la instrucción correspondiente a la línea sobre la cual se posa el cursor. En el segundo caso planteado, y a diferencia del primero, se deshace la ejecución de todas las instrucciones desde la última instrucción ejecutada hasta la instrucción correspondiente a la línea sobre la cual se posa el cursor.

Las instrucciones ejecutadas cambian de color y se deshabilita la posibilidad de su modificación al ser ejecutadas, retomando su color original, además de volver a ser pasibles de modificación, al deshacerse la ejecución.

En la ejecución por bloques el usuario puede seleccionar un bloque de código y ejecutarlo, sin necesidad de ejecutar el código anterior al bloque seleccionado. La ejecución considera la información que exista en el contexto antes de la selección del bloque, en otras palabras, se continúa la ejecución actual, ignorando el código entre la última instrucción ejecutada y el bloque de código seleccionado para la ejecución.

La siguiente imagen (Fig. 3) muestra como se ve en Coop este requerimiento:

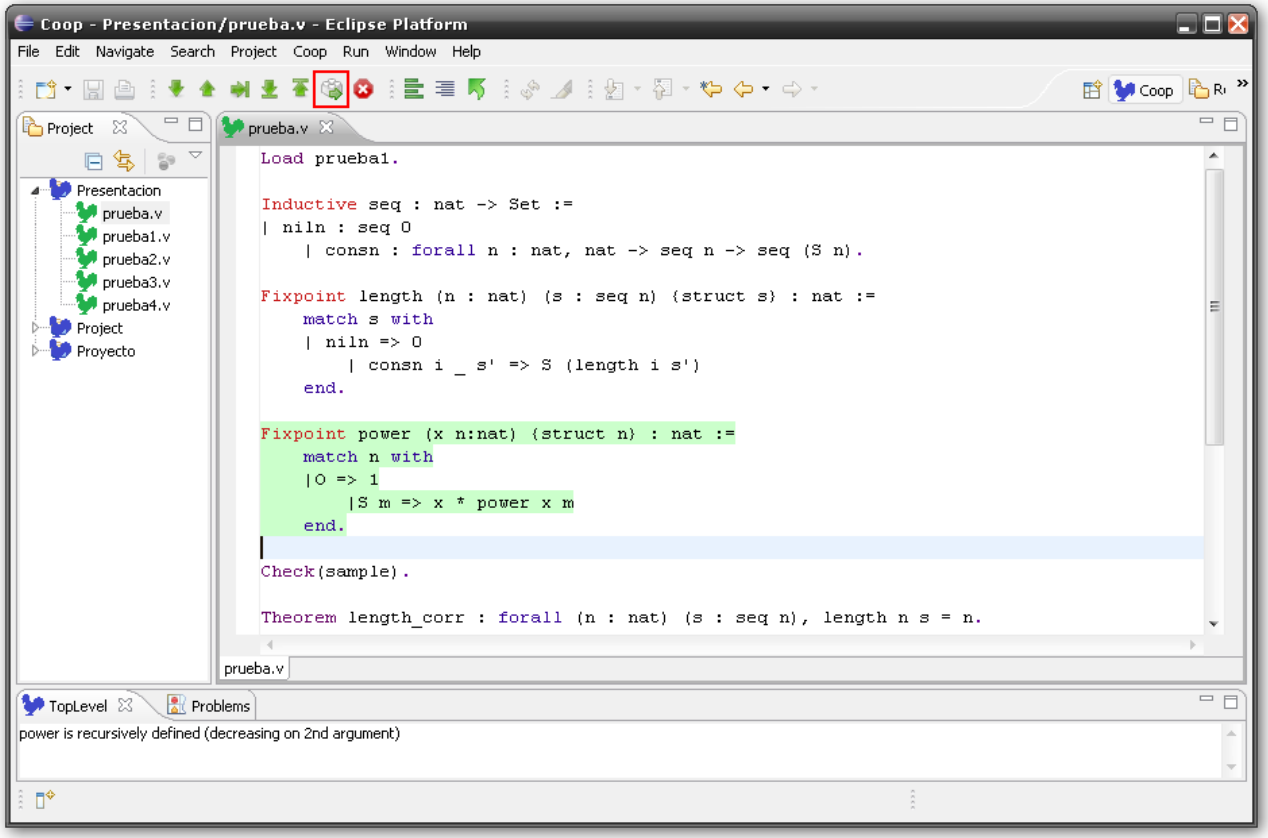

Fig. 3: Ejecución por bloques

Esta funcionalidad está disponible en la barra de herramientas de la perspectiva asociada.

Si una ejecución toma mucho tiempo, el usuario puede cancelar dicha ejecución mediante la funcionalidad interrumpir, disponible también en la barra de herramientas.

Coop, y en particular el editor, permite tener en ejecución múltiples instancias del motor de Coq al mismo tiempo, cada una asociada a un archivo Coq diferente. Una vez comenzada la ejecución de un archivo se crea una instancia del motor y se asocia a dicho archivo la vista de mensajes y el contexto correspondiente. Al seleccionar otro archivo en ejecución se cambian las vistas a la del nuevo archivo, salvando el contexto del archivo que se abandona para ser restaurado cuando retome el foco. Esta propiedad es prácticamente inexistente en otras herramientas similares.

De igual manera, al cerrar un archivo que está asociado a una instancia de un motor Coq en ejecución, se interrumpe la ejecución de dicha instancia y se avisa al usuario antes de cerrar el archivo, quien decidirá si aceptar o cancelar la acción.

Todas estas funcionalidades están disponibles en la barra de herramientas con la que cuenta la perspectiva de Coop. Otra posibilidad para ejecutar estas acciones son las teclas rápidas. Que en particular permiten:

- Ctrl+Alt+Down: Progresar un paso en la ejecución
- Ctrl+Alt+Up: Regresar un paso en la ejecución
- Ctrl+Alt+Right: Ejecutar hasta un paso antes del cursor (tanto hacia adelante como hacia atrás)
- Ctrl+Alt+Left: Cancelar la prueba en ejecución

- Ctrl+Alt+Home: Regresar la ejecución al comienzo del archivo
- Ctrl+Alt+End: Avanzar la ejecución hasta el final del archivo
- Ctrl+Alt+Break: Interrumpir la computación

Existen otras funcionalidades asociadas a la ejecución como ser las opciones de display, que permiten aumentar la información que despliega el motor de Coq al realizar diferentes acciones. Entre estas también se encuentra el outline. El editor de Coop dispone de una vista en la que se muestren, en forma de árbol (similar a ProverEditor), las estructuras presentes en el archivo Coq que está siendo manipulado, con las siguientes características:

- Los nodos del árbol están enlazados con el código correspondiente, por lo que, si el usuario navega a través del árbol, en el editor de archivos Coq se mostrará la porción de código correspondiente al nodo visitado.
- Se marca en cada caso si el nodo se corresponde con una prueba, una definición, etc.
- Los nodos incluyen además las estructuras (funciones, propiedades, definiciones, tácticas, etc.) existentes en módulos auxiliares e información sobre en qué módulo se encuentran.
- Se marcan también aquellos elementos no definidos en el contexto en el cual se está trabajando.

Este requerimiento se muestra en la siguiente imagen (Fig. 4):

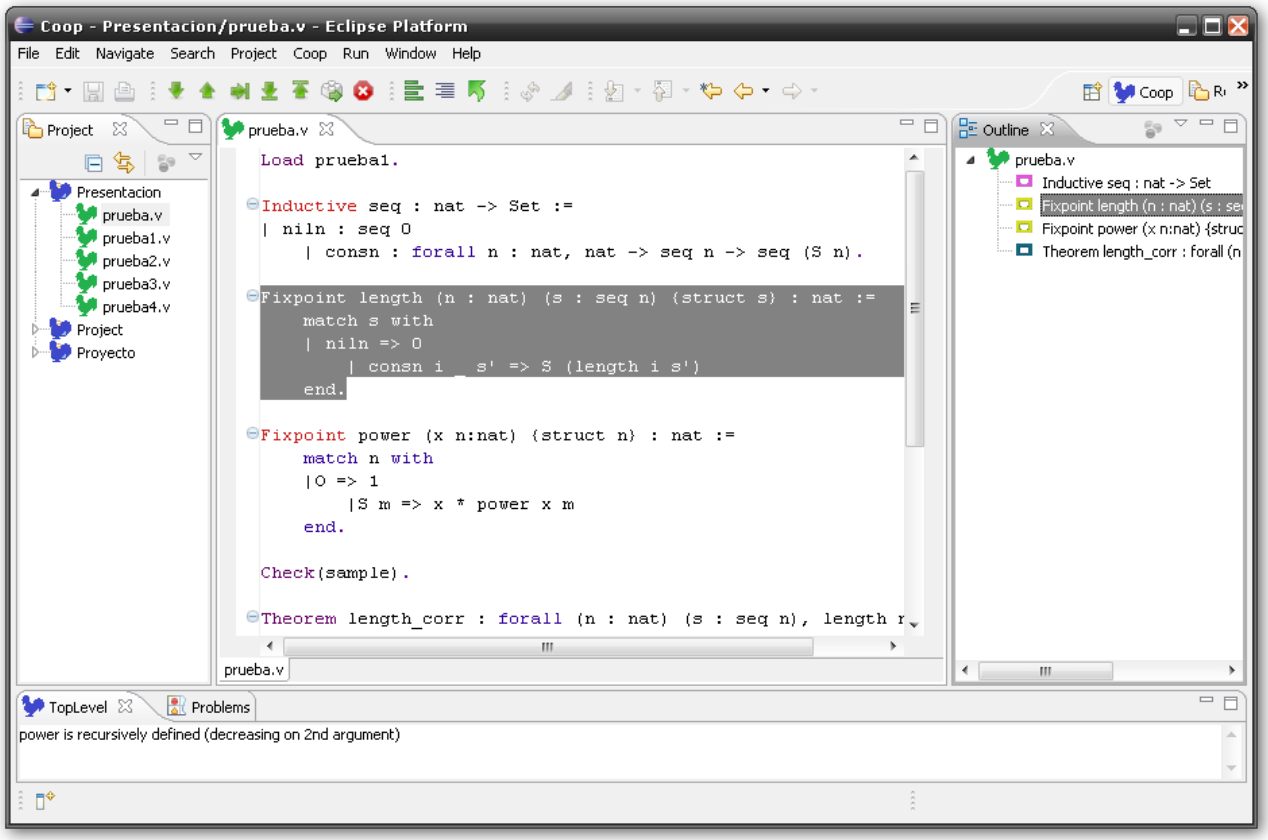

Fig. 4: Outline

Además de la vista de outline, Coop dispone de una vista en la que se muestra de forma gráfica la derivación lógica, correspondiente a la prueba que está actualmente en ejecución, en forma de árbol. Coop permite la exportación de esta de dos formas:

- A LaTeX utilizando el paquete Semantics [20].
- A XML en base a un formato a definir.

La Fig. 5 muestra como se ve en Coop una derivación lógica:

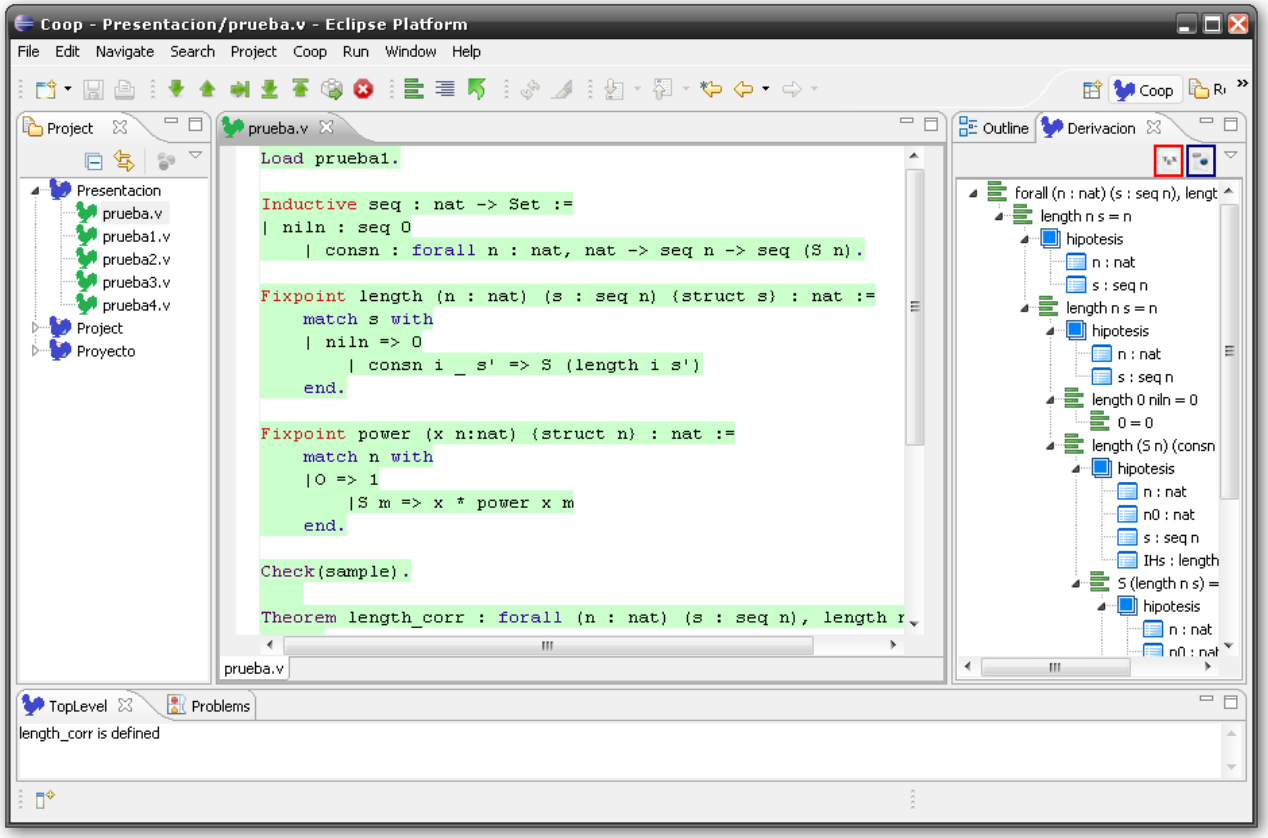

Fig. 5: Exportar derivación

Coop no solo cuenta con funcionalidades orientadas a la ejecución, existe otro conjunto de requerimientos asociados a la edición de código que pretenden facilitar la experiencia del usuario a la hora de utilizar el editor de Coop. En particular, al abrir un paréntesis se genera automáticamente el paréntesis de cierre. Además, al seleccionar un paréntesis se marca el correspondiente paréntesis de apertura/cierre.

El editor también permite simplificar el uso de comentarios de dos formas:

- Al abrir un comentario "(*" se genera automáticamente el símbolo de cierre "*)".
- Al seleccionar un bloque de código se permite comentar el bloque entero sin necesidad de agregar los símbolos a mano. De la misma forma se permite eliminar un comentario existente. Esto se puede realizar con la combinación de teclas correspondiente.

Las palabras reservadas del lenguaje Coq aparecerán coloreadas, o sea que una vez ingresada una palabra al editor de archivos Coq, esta tomará un color que la resalte del resto si pertenece al lenguaje.

Coop también permite la indentación automática y básica del código. Permite incluso configurar el tamaño de la tabulación. Esta acción se puede ejecutar tanto a través de la barra de herramientas así como a través de la combinación de teclas correspondiente.

Un archivo Coq está conformado por bloques de código, en consecuencia el sistema permite colapsar y expandir estos bloques del código Coq al igual que otros editores en Eclipse. Básicamente todo bloque definido por una estructura estándar (induction, proof, definition, etc.) o por comentarios pueden ser expandidos y colapsados para mejorar la legibilidad del código.

Otra de las principales funcionalidades destinadas a la asistencia del usuario en la edición de código es el auto completado. Mediante las teclas CTRL+SPACE se ejecuta el autocompletado de palabras reservadas o variables definidas anteriormente en el archivo. Además, el usuario dispondrá, para su selección, de plantillas de las estructuras básicas disponibles para Coq (Theorem, Lemma, etc.).

La Fig. 6 muestra como se ven las plantillas en Coop:

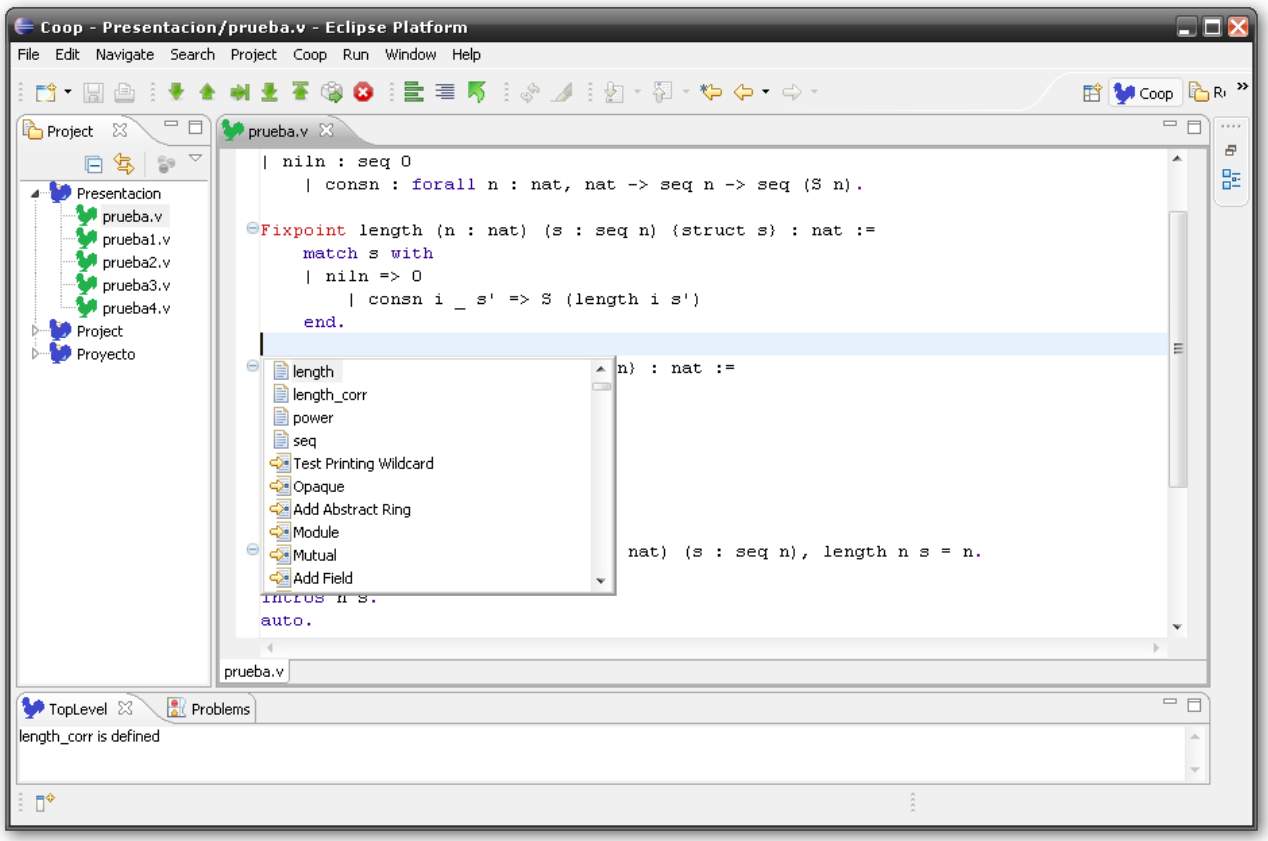

Fig. 6: Plantillas

Coop permite configurar fácilmente distintos parámetros, entre otros:

- Paleta de colores
- Ubicación del motor Coq y archivos de configuración
- Ubicación de aplicaciones de soporte
- Ubicación de documentación y manuales

En la Fig. 7 se muestra una de las páginas de configuración en Coop.

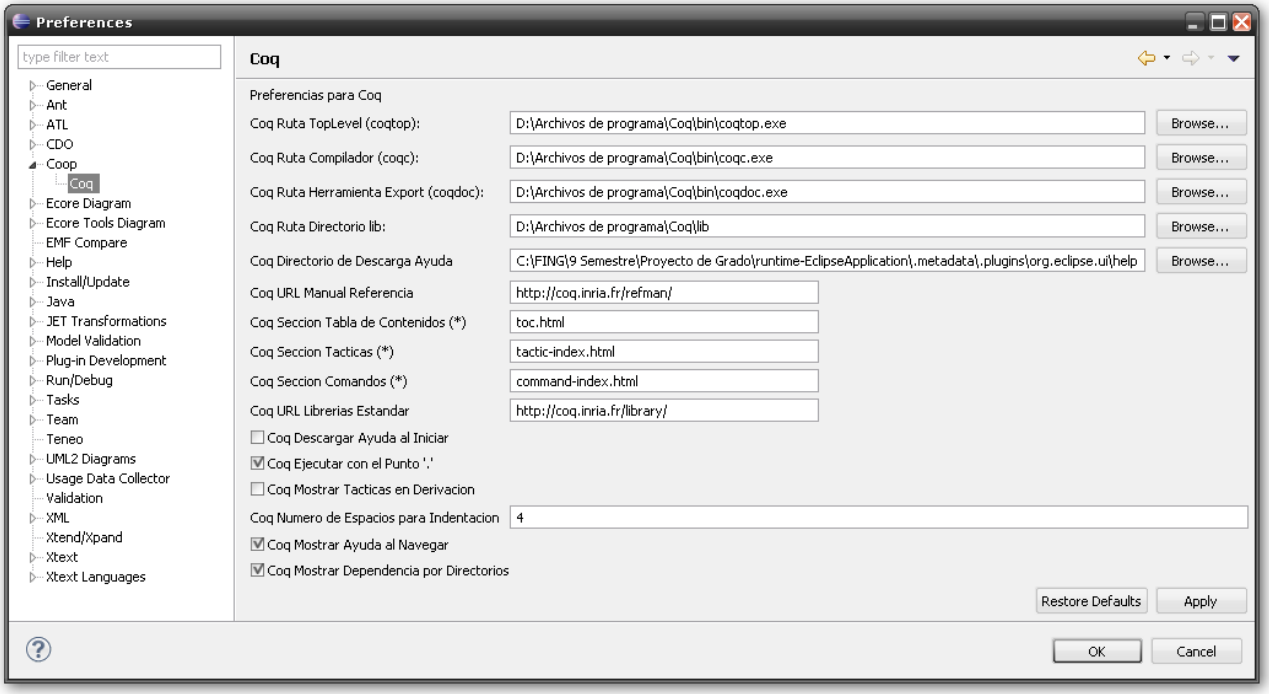

Fig. 7: Configuración

Además se permite al usuario configurar las dependencias del proyecto, simplificando la inclusión de librerías y el orden de las mismas.

Una vez configurada las rutas a los binarios de Coq, el usuario puede compilar el archivo Coq que está siendo manipulado actualmente. Adicionalmente puede exportar el proyecto Coq con el que se está trabajando, a un documento en diferentes formatos. Para esto, una vez seleccionada la opción Exportar Proyecto, se muestra una ventana donde se permite setear las diferentes opciones de configuración que se aplican al documento resultado de la exportación.

Entre las principales opciones se encuentran:

- Tipo: Formato del documento exportado (formatos disponibles: HTML, PDF y LaTex).
- Encabezado/Pie de Archivo: Seleccionar archivos para establecer como encabezado y/o pie del documento exportado.
- Índice/Tabla de Contenidos: Omitir la generación de Índice (por defecto se genera) o bien generar tabla de contenidos.
- Codificación: Establecer la codificación del documento generado UTF-8 o ISO-8859-1 (también el charset para HTML).
- Archivo de Salida: Por defecto se crea un archivo (con formato seleccionado) por cada archivo fuente en el proyecto. Esta opción permite seleccionar un único archivo de salida.

La Fig. 8 muestra como se ve en Coop la ventana de exportación:

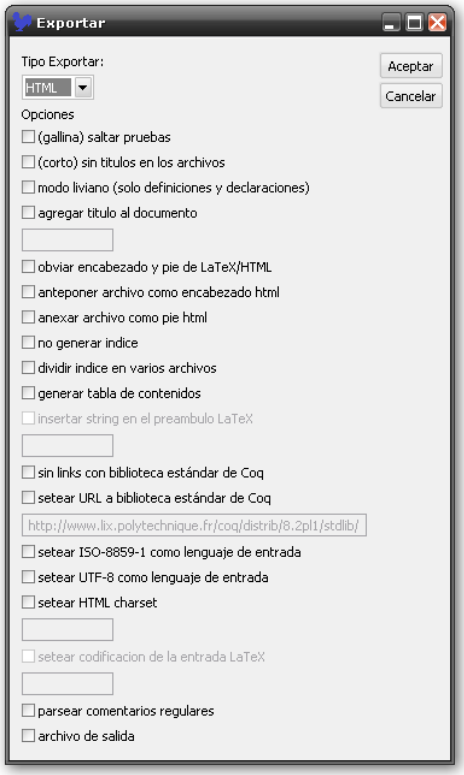

Fig. 8: Export

Para realizar la exportación del proyecto en un documento se utiliza la herramienta coqdoc.

Otra de las funcionalidades heredada del relevamiento de herramientas es la realización de Query.

Este requerimiento se visualiza en Coop como lo muestra la Fig. 9:

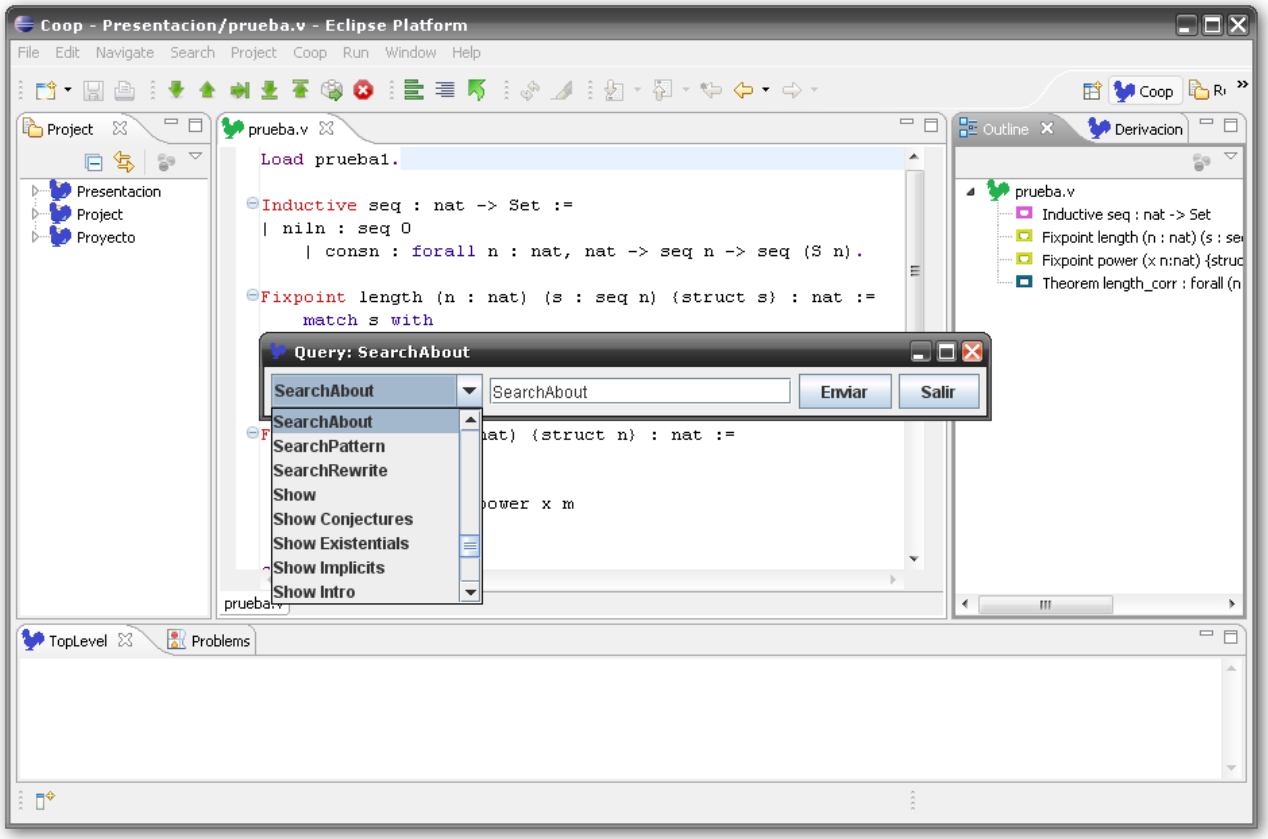

Fig. 9: Query

Esta funcionalidad brinda la posibilidad de poder realizar consultas o subconsultas en forma independiente al código de la prueba que se está ejecutando, o sea sin alterarlo. Un ejemplo de esta funcionalidad es realizar un "Check" de una expresión ya definida, que no es de interés para la prueba actual. Coop cuenta con un conjunto predefinido de consultas para facilitar el uso, tal como lo hace CoqIDE.

En lo referente a la ayuda, el usuario cuenta con un menú de ayuda con información de tácticas, construcciones, links al manual de Coq, etc. Este requerimiento permite además enlazar la web de Coq desde el editor. Además el usuario puede consultar en forma dinámica la ayuda para una determinada palabra o comando con solo seleccionarlo en el código.

La Fig. 10 ilustra como se ve este requerimiento en Coop.

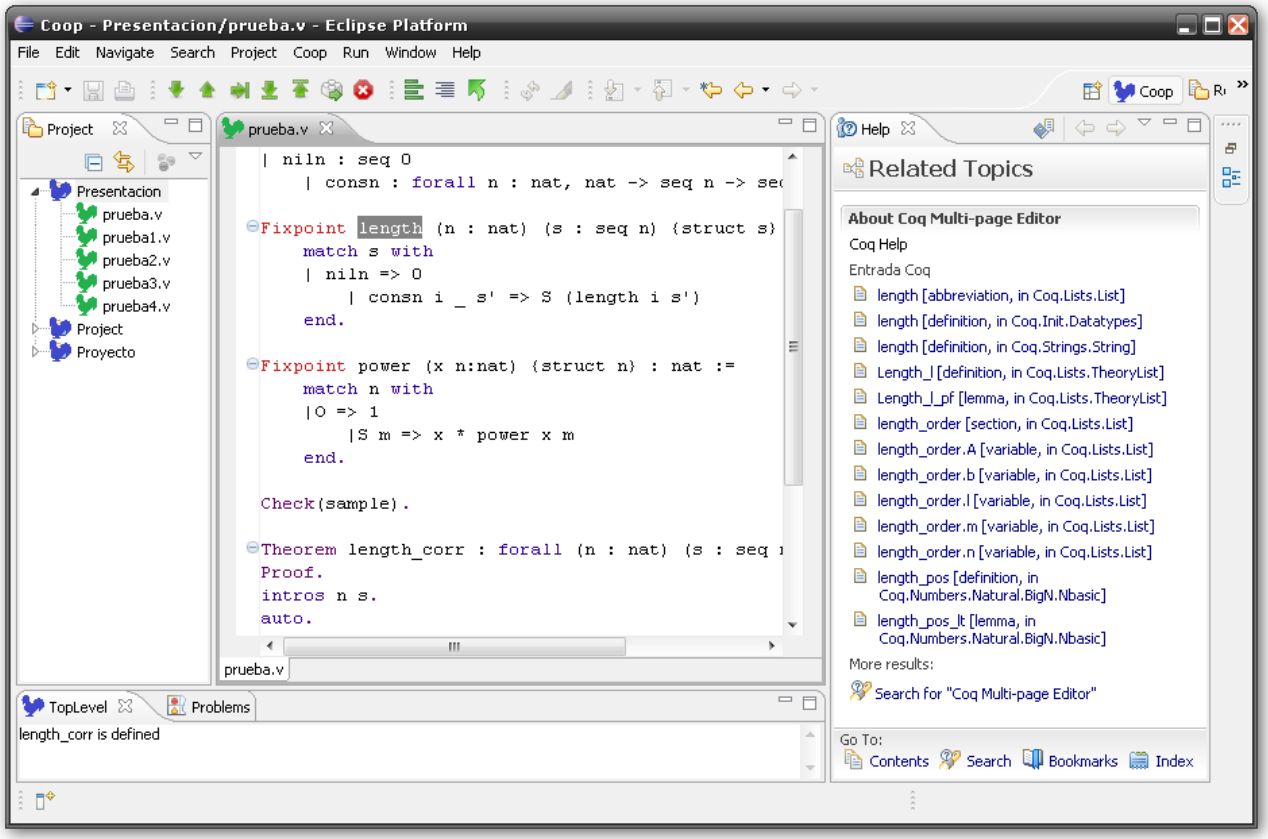

Fig. 10: Ayuda dinámica

Las últimas dos funcionalidades presentadas en esta capítulo son la navegación a definiciones y el grafo de dependencias. En el primer caso, las funciones o propiedades que son utilizadas en determinado lugar del código y que se encuentren definidas en el contexto del proyecto (en los archivos definidos o en los importados) son navegables al hacer CTRL+N sobre ellas, con lo que se irá a su definición. Mientras que en la vista del grafo de dependencias el usuario puede acceder a una representación gráfica de las dependencias de los archivos del proyecto seleccionado.

En la Fig. 11 se muestra el grafo de dependencias de un proyecto en Coop.

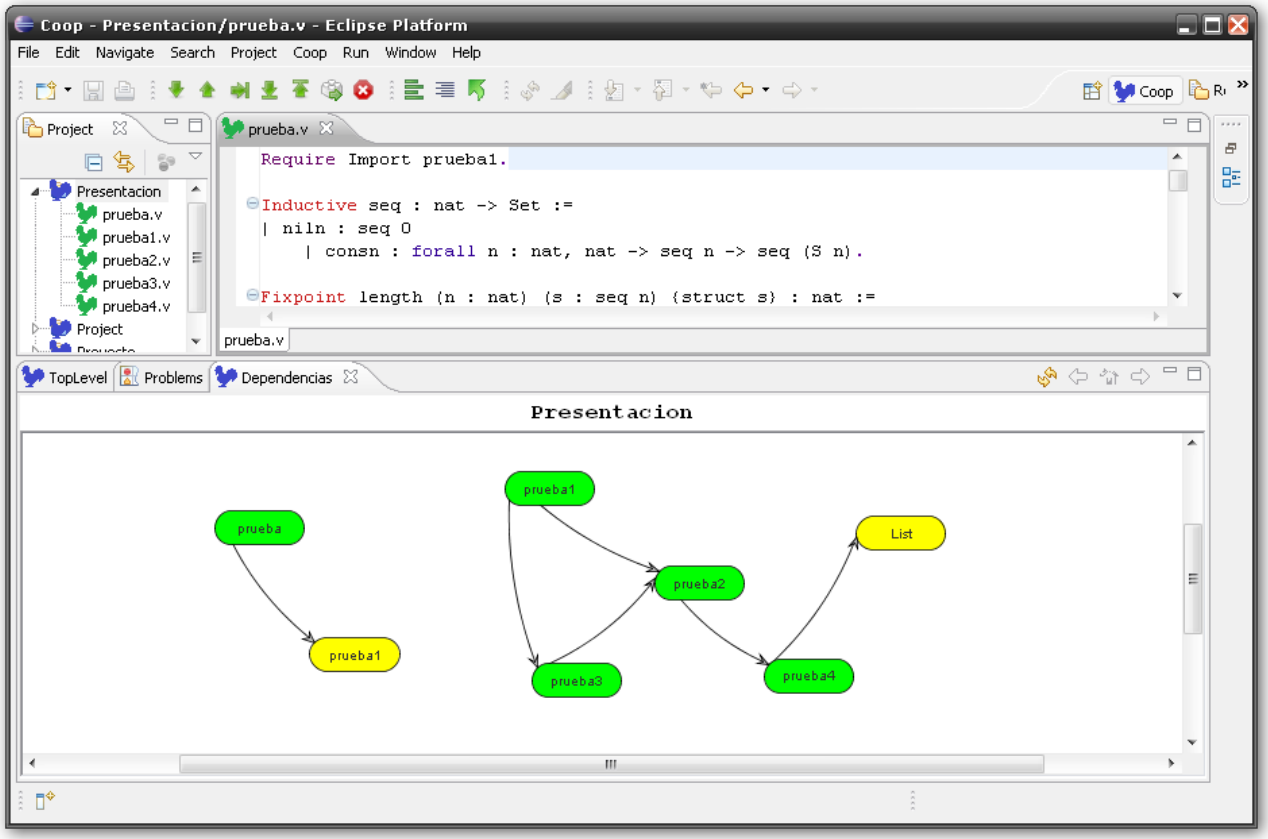

Fig. 11: Grafo de dependencias

Se recomienda consultar los anexos: *Documento de Requerimientos Coop* y *Manual de Usuario Coop* para obtener una visión más completa de las funcionalidades ofrecidas por Coop.

Capítulo 5

Diseño y Arquitectura

En este capítulo veremos los principales fundamentos del diseño de la herramienta así como la manera en que ésta se acopla con Eclipse. También se desarrollan los principales medios de configuración y extensión de Coop. Finalizamos este capítulo con las implementaciones, o mejor dicho, los conceptos centrales detrás de las implementaciones más interesantes de la herramienta. Las descripciones van acompañadas de diagramas ilustrativos que permiten apreciar de mejor manera las relaciones entre las clases participantes de las distintas soluciones presentadas.

5.1 Fundamentos del Diseño de la herramienta

La arquitectura modular y extensible de Eclipse permite integrar, a través de un plug-in, el asistente de pruebas Coq en un ambiente de desarrollo popular y libre, que cuenta con extensiones disponibles para el manejo de modelos y transformaciones, lo cual se enmarca en las necesidades que llevaron a la concepción de este proyecto.

El plug-in Coop extiende una amplia variedad de puntos de extensión (extension-points) de Eclipse y cumple con el rol de intermediario entre el usuario y el motor Coq, ofreciendo una experiencia de usuario más amigable.

La Fig. 12 ilustra la arquitectura básica de Coop.

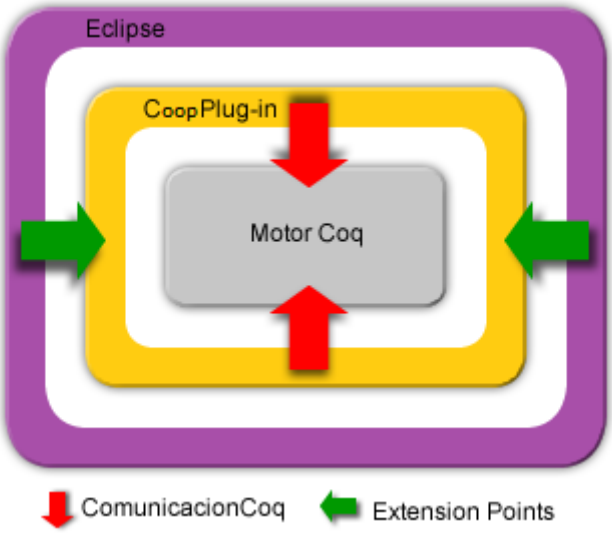

Fig. 12: Arquitectura de Coop

5.2 Configuración de la herramienta

Todo plug-in de Eclipse cuenta con un archivo de configuración, llamado plugin.xml, el cual contiene la configuración de los puntos extendidos por el plug-in. Básicamente, entre otras cosas, se asocia una clase con un punto de extensión. Esto se traduce en forzar que determinada clase implemente o extienda una interfaz o clase de Eclipse.

Un ejemplo del contenido del archivo plugin.xml es el que se presenta a continuación en la Fig. 13.

```
<extension point="org.eclipse.ui.views">
      <category
            name="IDE Coq"
            id="Coq">
      </category>
      <view
            name="TopLevel"
            icon="icons/coqB.gif"
            category="Coq"
            class="coq.views.TopLevel"
            id="coq.views.TopLevel">
      </view>
</extension>
```

Fig. 13: Extracto de plugin.xml

Aquí se puede observar un ejemplo de configuración de una vista, donde se define la categoría en la cual aparecerá la vista dentro del menú Window -> Show View -> Other..., la vista propiamente dicha con su nombre, el ícono, la categoría a la cual pertenece (en este caso la definida anteriormente), el identificador y la clase que implementa, la vista la cual deberá extender la clase `ViewPart`.

Otro ejemplo interesante para presentar es el de la configuración de las acciones para que sean ejecutables a través de una combinación de teclas y que aparezcan tanto en la barra de herramientas, como en el menú de la herramienta.

Dicha configuración se realiza de la siguiente manera (Fig. 14):

```
<extension point="org.eclipse.ui.commands">
      <command
            name="Coq comentar"
            categoryId="coq.commands.category"
            id="coq.commands.comentar">
      </command>
</extension>
<extension point="org.eclipse.ui.handlers">
      <handler
            commandId="coq.commands.comentar"
            class="coq.actions.CoqActionComentar">
      </handler>
</extension>
<extension point="org.eclipse.ui.bindings">
      <key
            commandId="coq.commands.comentar"
            contextId="org.eclipse.ui.contexts.window"
            sequence="M1+7"
            schemeId="org.eclipse.ui.defaultAcceleratorConfiguration">
      </key>
</extension>
<extension point="org.eclipse.ui.actionSets">
      <actionSet
            label="IDE Coq"
            visible="true"
            id="Coq.actionSet">
            <menu
                  label="IDE Coq"
                  id="MenuCoq">
            </menu>
            <action
                  label="Comentar"
                  icon="icons/comment.ico"
                  class="coq.actions.CoqActionComentar"
                  tooltip="Comentar codigo seleccionado"
                  toolbarPath="sampleGroup"
                  menubarPath="MenuCoq"
                  id="coq.actions.CoqActionComentar">
            </action>
      </actionSet>
</extension>
```

Fig. 14: Configuración de extension points

Aquí se puede observar cómo se define un actionSet (conjunto de acciones) con una acción con etiqueta "Comentar", se especifica su ícono, la clase que la implementa (indicando también los paquetes a los que pertenece), el texto que se desplegará cuando el usuario pose el ratón sobre el botón asociado a la acción (tooltip), el grupo al que pertenece dentro de la barra de herramientas (toolbar), el camino (path) dentro del menú de la herramienta y el identificador de la acción.

La misma clase que se asocia a esta acción es la que se vincula al handler, que se define más arriba. Este handler se asocia con un comando a través del identificador de este último. Esto mismo se hace con el "key", en el cual se define el atributo sequence que indica la secuencia de teclas con la cual se dispara el comando y a través de él el handler que invocará a la misma clase que atenderá los eventos disparados tanto desde la barra de herramientas, como los disparados desde el menú. De esta manera nos aseguramos el mismo comportamiento para los tres disparadores posibles para esta acción.

Como se puede observar en la Fig. 15 son muchos los puntos en los que Coop extiende a Eclipse. A continuación se enumeran los principales:

```
org.eclipse.ui.newWizards
org.eclipse.ui.commands
org.eclipse.ui.handlers
org.eclipse.ui.bindings
org.eclipse.ui.actionSets
org.eclipse.ui.editors
org.eclipse.ui.perspectives
org.eclipse.ui.popupMenus
org.eclipse.ui.views
org.eclipse.ui.perspectiveExtensions
org.eclipse.core.resources.builders
org.eclipse.core.resources.natures
org.eclipse.ui.ide.projectNatureImages
org.eclipse.core.resources.markers
org.eclipse.ui.preferencePages
org.eclipse.help.toc
org.eclipse.help.contexts
org.eclipse.help.base.luceneSearchParticipants
org.eclipse.help.contentProducer
```

Fig. 15: Puntos de extensión

Con el fin de esclarecer el funcionamiento básico interno y el mecanismo utilizado para comunicar Coop con el motor Coq se presenta el siguiente diagrama (Fig. 16):

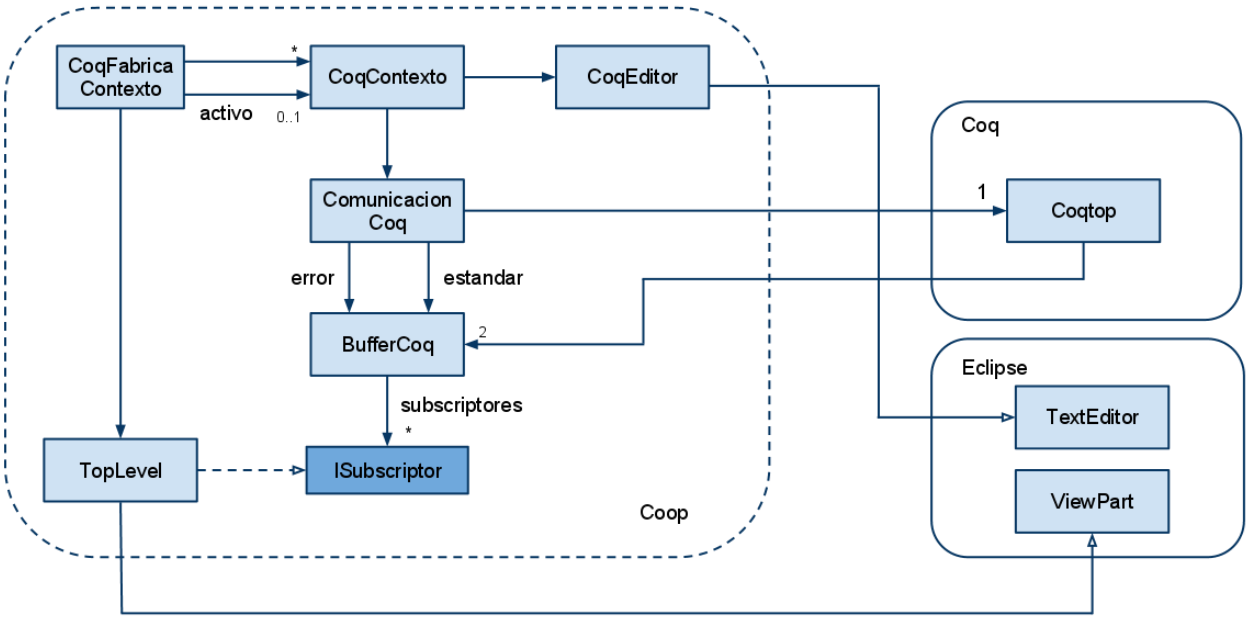

Fig. 16: Diagrama de clases: comunicación Coop-Coq-Eclipse

Como podemos ver en el diagrama, Coop tiene un conjunto de clases que posibilita la comunicación con el motor Coq. Este conjunto es conformado principalmente por la clase `CoqContexto`. Todas las instancias de esta clase son manejadas por una única instancia de la clase `CoqFabricaContexto`. Cada archivo abierto en Eclipse tendrá asociado una instancia de `CoqContexto`. A su vez, cada una de estas instancias dispondrá de instancias de `CoqEditor` y de `ComunicacionCoq`, entre otras, siendo la primera una extensión de la clase `TextEditor` de Eclipse, mientras que la segunda es la responsable de posibilitar la comunicación con Coq.

`ComunicacionCoq` genera un proceso correspondiente al binario coqtop.exe (en un ambiente Windows) al cual le envía directamente todos los comandos que la herramienta requiera para ejecutar el archivo asociado al contexto (`CoqContexto`) "dueño" de la instancia de `ComunicacionCoq`, al que se le llama contexto activo.

Para la recepción de la salida, `ComunicacionCoq` crea un par de instancias de la clase `BufferCoq`, las cuales se encargan de hacer polling (de forma asincrónica con respecto al resto de la aplicación, se extiende la clase `Thread` con tal fin) sobre su respectivo buffer (el de error o salida estándar). Cada instancia de `BufferCoq` cuenta con una colección de `ISubscriptor`'s y de los métodos necesarios para dar de alta y baja de esta colección a toda clase que implemente la interfaz `ISubscriptor`. Una vez que `BufferCoq` recibe un mensaje por parte del motor Coq, este notifica (envía el mensaje recibido) a todos los miembros de la colección de `ISubscriptor`'s como es el caso de la clase `TopLevel`, que además de extender una vista de Eclipse implementa la Interfaz `ISubscriptor`, con el fin de mostrar al usuario los resultados de la ejecución de sus pruebas a través de Coop.

Hasta aquí hemos visto cómo se extienden los distintos puntos de Eclipse, ahora veremos los archivos configurables con los que cuenta la herramienta. Estos son:

- messages_español.properties
- messages_english.properties
- language.xml
- templates.xml

Tanto el archivo messages_español.properties como messages_english.properties contiene todos los strings que son visibles para el usuario en el idioma correspondiente. De estos archivos la herramienta se nutrirá según la configuración de idioma que seleccione el usuario (ver anexo *Manual de Usuario Coop*).

Un breve ejemplo de estos archivos se aprecia en la Fig. 17.

messages_español.properties

```
CoqFileWizard_1=Nuevo archivo Coq
CoqFileWizard_2=Crear un nuevo archivo para editar en Coop.
CoquitoAction_0=Salvar cambios
CoquitoAction_1=\u00BFDesea guardar los cambios?°
```

messages_english.properties

```
CoqFileWizard_1=New File
CoqFileWizard_2=Create a new file for editing in Coop.
CoquitoAction_0=Save changes
CoquitoAction_1=\u00BFSave changes?
```

Fig. 17: Configuración de idiomas

Acabamos de ver cómo obtiene la herramienta los strings a mostrar, dependiendo del idioma seleccionado por el usuario, pero aún no sabemos cómo se configuran los idiomas que tendrá a su disposición. Aquí es donde entra el archivo languages.xml, el cual contiene los idiomas que el usuario podrá seleccionar (por defecto español e inglés). Vale destacar que el nombre que se le dé en el archivo languages.xml al idioma debe coincidir con la sección resaltada en el nombre del correspondiente archivo messages_nombreIdioma.properties. Además, y como resulta lógico, este último archivo debe existir para proporcionar los strings correspondientes.

Un ejemplo de este archivo es el que sigue (en Fig. 18):

```
<languageConf>
      <languages>
            <language>
                  <name>english</name>
            </language>
            <language>
                  <name>español</name>
            </language>
            </languages>
</languageConf>
```

Fig. 18: languages.xml

Por último, queda ver el archivo templates.xml que es el que contiene las tácticas y templates o plantillas que serán sugeridas al usuario al presionar CTRL+espacio en el editor de la herramienta.

A continuación se ilustra el contenido del archivo (Fig. 19):

```
<templates>
      <generales>
            <general>
                  <titulo>
                        Theorem
                  </titulo>
                  <cuerpo>
                        Theorem new_theorem : .
                        Proof.

                        Save.
                  </cuerpo>
            </general>
      </generales>
      <tacticas>
            <tactica>
                  <titulo>
                        Intro
                  </titulo>
                  <cuerpo>
                        intro
                  </cuerpo>
            </tactica>
      </tacticas>
</templates>
```

Fig. 19: templates.xml

5.3 Solución de los principales requerimientos

En esta sección se describen las soluciones a los problemas inherentes a los principales requerimientos.

5.3.1 **Ejecución** (adelante, atrás, fin de archivo, comienzo de archivo y hasta cursor)

La ejecución de alguna de estas funcionalidades comienza con la acción del usuario, ya sea pulsando el botón correspondiente en la barra de herramientas, tecleando los atajos por teclas o a través del menú de la herramienta, e involucra la ejecución propiamente dicha de la(s) instrucción(es) y del pintado correspondiente.

Una vez disparada la funcionalidad el control del flujo cae sobre la clase heredera de `CoqActionPadre` correspondiente. Dependiendo de la funcionalidad estas podrían ser: `CoqAction`, `CoqActionBack`, `CoqActionStart`, `CoqActionEnd` y `CoqActionCursor`. Básicamente todas estas clases tienen el mismo comportamiento y no hacen mucho más que delegar responsabilidad al contexto correspondiente, comprobando previamente que el archivo no ha sido editado desde la última vez que fue salvado. Como se vio previamente existe una instancia de la clase `CoqContexto` por archivo abierto y una sola de estas instancias se corresponderá con el contexto activo, que no es más que el contexto asociado al archivo que tiene el foco en el editor de la herramienta. En otras palabras, el contexto activo es aquel asociado al archivo que se está ejecutando o editando al momento de disparar la acción. Al pasar el control de la ejecución al contexto activo se obtiene la siguiente instrucción a ejecutar (aquí se expone la ejecución de un

paso hacia adelante por simplicidad (entendiendo la gran similitud conceptual con la implementación de las otras funcionalidades) la que no necesariamente se debe corresponder con la siguiente línea, ya que una instrucción puede, y en muchos casos será así, ocupar más de una línea. Es por esto que la operación que retorna la siguiente instrucción a ejecutar, nextLine (también implementada por la clase `CoqContexto`), implementa un relevamiento de todas las líneas entre la última línea de la instrucción ejecutada previamente y el carácter '.', que es el que indica el fin de una instrucción Coq.

Al disponer ahora de la instrucción a ejecutar, se obtiene la última instrucción ejecutada, de una lista (que se usa como pila), también guardada por el contexto, donde se guarda el número de las instrucciones ejecutadas, con la intención de volver atrás la ejecución en caso de error. Aunque la verdadera motivación de esta lista es la ejecución por bloques, como se verá más adelante en esta sección. En este punto disponemos de la instrucción o comando a ejecutar y del punto de retorno en caso de error, pero para determinar si ha ocurrido un error en la ejecución se dispara un hilo (clase `HiloHayError`, implementada dentro de `CoqContexto`) que correrá paralelamente a la ejecución del nuevo comando escuchando la salida estándar (se inscribirá en la lista de subscriptores de la instancia de `ComunicacionCoq` como se vio previamente). En caso de detectar un error seteará la variable a consultar por parte del hilo principal con la intención de determinar el éxito o no de la última ejecución. En caso de haber fallado, como se mencionó previamente, se volverá atrás en la ejecución pintando de rojo la instrucción que provocó la falla.

Para ejecutar efectivamente la instrucción se corre el método enviar de la clase `ComunicacionCoq`, cuyo funcionamiento fue expuesto previamente.

Hasta aquí lo relevante a la ejecución, pero resta el pintado de todas las líneas que componen la instrucción recién ejecutada. Como también se vio previamente, la clase `CoqContexto` tiene asociada la instancia `CoqEditor`, y a esta será a la que se le delegue la responsabilidad de pintar estas líneas a través del método pintarLíneas que entre otras cosas se encargará de setear el límite (hasta donde pintar) en la clase `CoqConfiguration`. Esta a su vez hará lo mismo pero con su instancia de `CoqScanner`, que en su método nextToken evalúa, según el límite que se le ha seteado, si debe pintar el fondo de un token (o palabra) o no.

La Fig. 20 ilustra las asociaciones entre las distintas clases aquí involucradas:

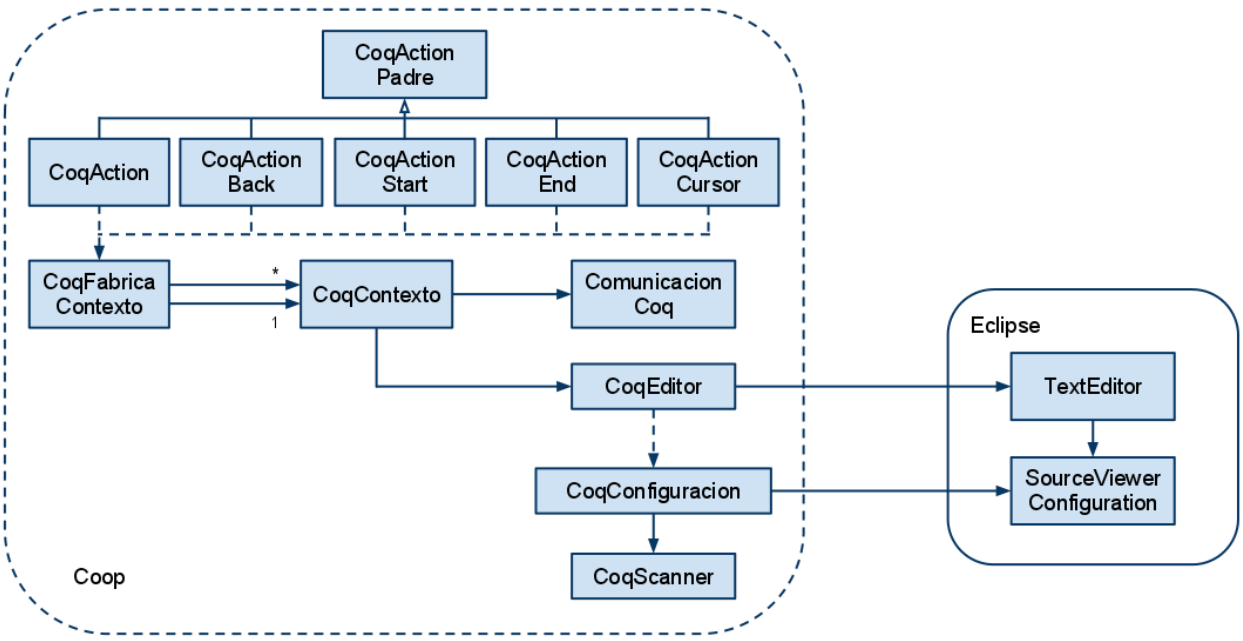

Fig. 20: Diagrama de clases para la ejecución

Otra de las cosas interesantes a remarcar es que además de pintar, se debe impedir la modificación de las líneas involucradas con las instrucciones ejecutadas. Para esto se adoptó una estrategia que según la posición del cursor, en un momento dado, modifica la condición de editable o no de todo el documento. En otras palabras, si el cursor se encuentra en texto ejecutado, el estado de todo el documento pasa a ser no editable. En caso contrario se mantiene editable, pero como lógicamente el cursor sólo puede estar en una posición en un momento dado, esto da la sensación de que sólo se puede editar el texto no ejecutado aún.

5.3.2 Ejecución por bloques

Este requerimiento se resolvió implementando una pila, en la clase `CoqContexto`, en la que se guardan los números de instrucciones ejecutadas hasta el momento, quitándole el tope a la pila cada vez que se deshace una ejecución. De este modo se pueden ejecutar bloques de instrucciones y a la hora de deshacer una ejecución saber cuál fue la última línea ejecutada, lo que permite saltar los "huecos" dejados al ejecutar bloques disjuntos.

La siguiente imagen (Fig. 21) muestra el comportamiento de la pila luego de deshacer la ejecución de las últimas cuatro instrucciones ejecutadas.

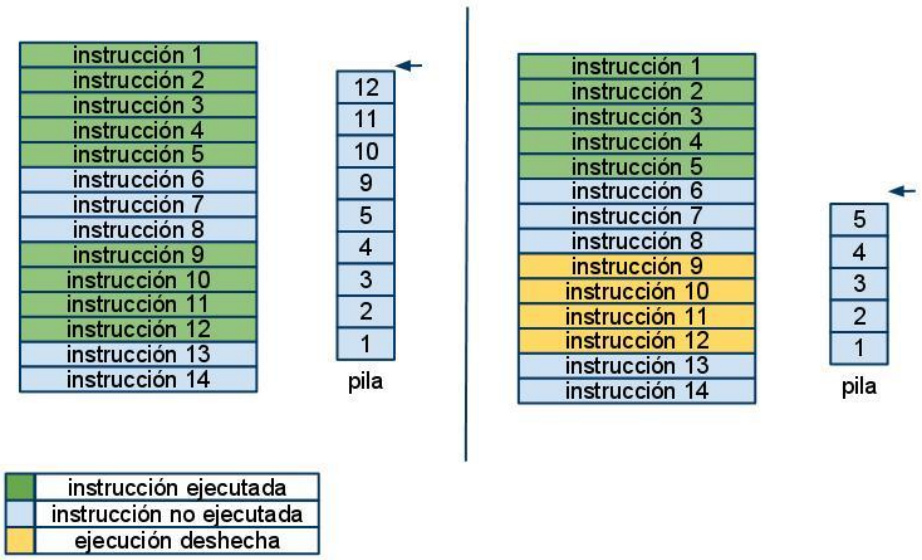

Fig. 21: Pila en ejecución por bloques

5.3.3 Compilar

El proceso de compilación de un archivo vernacular (.v) se realiza utilizando la herramienta coqc (presente en la distribución de Coq). De este modo, para realizar esta acción se utiliza la clase `ProcessBuilder` que permite ejecutar un comando del sistema operativo. Para este caso entonces se ejecuta la herramienta coqc recibiendo como parámetro el archivo vernacular que se quiere compilar. Se muestra como resultado el obtenido de ejecutar el comando, independientemente si el resultado fue exitoso o no. El archivo resultado de la compilación (archivo complidado con extensión .vo) se genera en el mismo directorio donde existe el archivo .v.

Notar que a el comando se le pasa como parámetro el archivo de entrada (nombre del archivo vernacular) y el loadpath definido en el proyecto. El comando que se ejecuta es el siguiente:

comando: *coqc path_file.v -I <loadpath1> -I <loadlpath2> .. -<loadpathN>*

Donde:

path_file.v: Ruta absoluta del archivo vernacular a compilar.

<loadpathi>: Ruta del directorio cargado en el i-ésimo lugar en el "loadpath" del proyecto.

5.3.4 Teclas rápidas (short keys)

Para lograr que ciertas funcionalidades sean ejecutables simplemente tecleando una combinación determinada de teclas se hace que las clases que implementan estas funcionalidades extiendan la clase `CoqActionPadre`, la cual, a su vez, extiende a `AbstractHandler`. Esto permite asociar, justamente, una combinación de teclas a la acción manejada por la clase en cuestión. La forma de configurar esta asociación es a través del archivo plugin.xml, como se describe más arriba.

5.3.5 Múltiples Coq

La clase `CoqMultiPageEditor` es la clase que implementa el punto de extensión org.eclipse.ui.editors (extiende a `MultiPageEditorPart`). Esto implica que Eclipse creará cada vez que se abra un archivo de extensión "v" (esto también se especifica en el archivo plugin.xml) una instancia de `CoqMultiPageEditor` y ejecutará su método createPages. Será en este, entonces, que se le pedirá a la fábrica de contextos (`CoqFabricaContexto`) que cree un nuevo contexto y lo setee como contexto activo. Esto es porque siempre cuando se abre un archivo Eclipse le da el foco, y el contexto activo es justamente el contexto asociado al archivo que se está ejecutando o editando en un momento dado. De esta manera, la fábrica guarda todos los contextos asociados a los archivos abiertos y como indirectamente estos están asociados a una instancia del motor Coq se podrán ejecutar las pruebas inherentes a cada archivo de forma independiente más allá que se cambie el foco de un archivo a otro en reiteradas veces.

El siguiente diagrama (Fig. 22) ilustra las relaciones recién planteadas entre las clases de Coop.

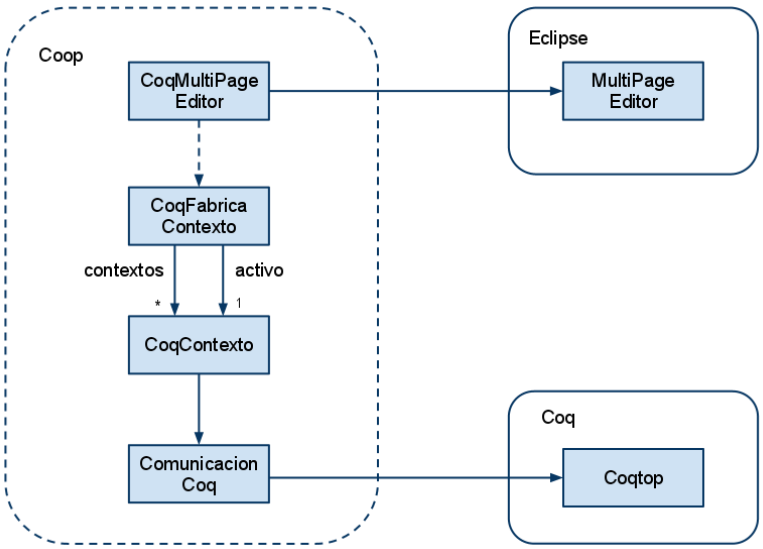

Fig. 22: Diagrama de clases para múltiples Coq

5.3.6 Configuración

Para que el usuario pueda realizar las distintas configuraciones desde el propio Eclipse y en la ventana de preferencias, se extiende el punto org.eclipse.ui.preferencePages con la clase `PreferencePage`. Esta clase es la implementación de la página de preferencias principal, la que se ve apenas el usuario se posa, o selecciona, la categoría Coop en la ventana de preferencias de Eclipse. Aquí se puede configurar el idioma de Coop y prácticamente todos los colores que usa el editor de archivos de Coop.

Al iniciarse el plug-in, o sea, al llamarse, por parte de Eclipse, al método start de la clase `Activator` se setean los valores por defecto de la página implementada por `PreferencePage` pasándole como parámetro la instancia de la clase, perteneciente a Eclipse, `PreferenceStore`. Esta clase es la encargada de persistir los cambios en la configuración hechos por el usuario, luego se crea una instancia de la clase `CoqPreferenceNode` que también recibe la instancia de `PreferenceStore`. Esta clase implementa el nuevo nodo para Coop en la ventana de preferencias o configuración de Eclipse y en el constructor, esta clase, crea una instancia de `CoqPreferencePage`, que es la implementación de la página de configuración dentro del nodo bajo el nombre Coq. El nuevo nodo se agrega al preference manager de Eclipse usando el método `PlatformUI.getWorkbench().getPreferenceManager().addTo("Coq.page", CoqPreferenceNode)`.

El acceso a los valores configurados por parte del resto de la herramienta se realiza a través de la clase `Activator`, que cuenta con geters para cada valor configurable. Aquí el `Activator` accederá a la página de preferencias que corresponda para retornar el valor requerido.

El siguiente diagrama (Fig. 23) ilustra las asociaciones entre clases que intervienen aquí.

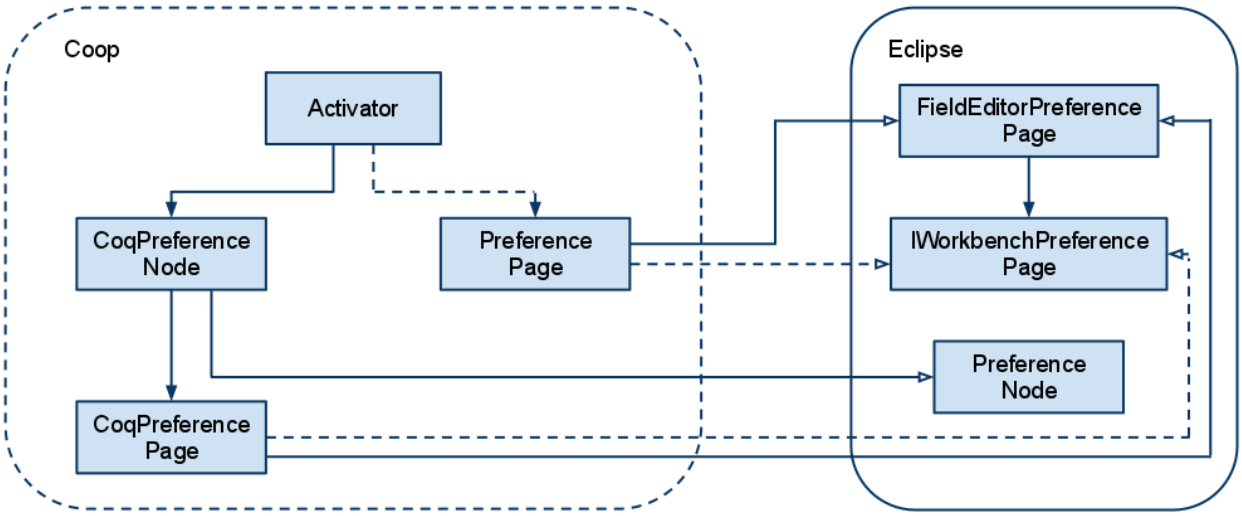

Fig. 23: Diagrama de clases para configuración

Nota: Esta implementación se basó en la homóloga del ProverEditor.

5.3.7 Outline

Eclipse, cuando el usuario abre el outline en una perspectiva determinada, llama al método getAdapter del `TextEditor` correspondiente, pidiendo por una instancia que implemente la interfaz `IContentOutlinePage`. Entonces el primer paso para implementar el outline es sobrescribir el método getAdapter en la clase `CoqEditor` que extiende a `TextEditor` y retornar ante la llamada recién descripta una instancia de la clase `BasicContentOutline`, que es

la implementación del outline en Coop. Esta clase está asociada con `TreeViewer`, que permite la implementación de una vista en forma de árbol. Este viewer requiere para generar el contenido a mostrar en el árbol una implementación de `ITreeContentProvider` que proveerá el contenido y una implementación de `ILabelProvider` que brinda las etiquetas a mostrar. Estas dos interfaces son implementadas en Coop por `TypeContentProvider` y por `TypeLabelProvider` respectivamente, y trabajarán sobre el árbol modelado por la clase `ProverType`.

La Fig. 24 muestra la interacción de las clases participantes en esta implementación.

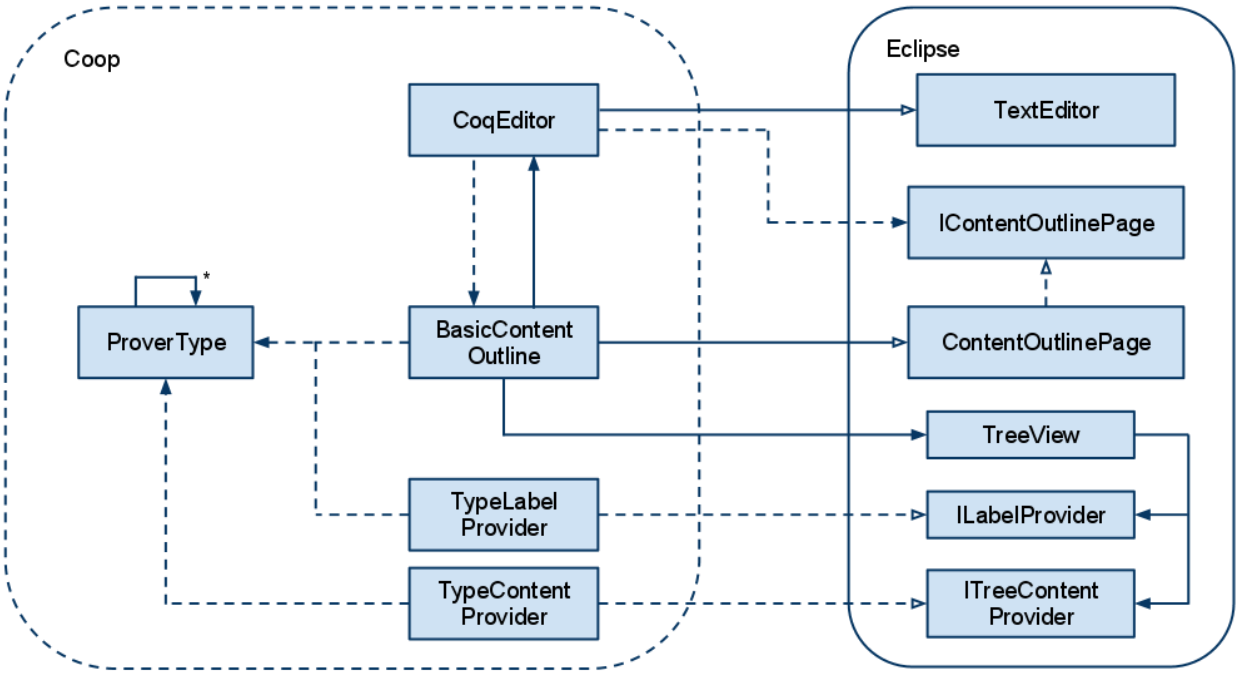

Fig. 24: Diagrama de clases para outline

Nota: Esta implementación se basó fuertemente en la homóloga del ProverEditor.

5.3.8 Exportación básica

Al igual que para la acción de compilar, para realizar la acción de exportar se utiliza una herramienta propia de Coq, siendo esta herramienta coqdoc. Como bien se dijo en los requerimientos al seleccionar la opción de exportar se muestra una ventana de "configuración" que permite seleccionar diferentes opciones. Estas opciones que se seleccionan se traducen en parámetros que recibe la herramienta coqdoc cuando es invocada. Como para el caso anterior se utiliza la clase `ProcessBuilder` para realizar la acción propiamente dicha. El comando que se ejecuta es el siguiente:

comando: *coqdoc --coqlib_path pathLib --Tipo --files-from fileFiles* <*opciones*>

Dónde:

pathLib: Ruta del directorio lib (presente con la distribución de Coq) debe configurarse en las preferences CoqIDE de Eclipse.

Tipo: Formato en el que se exporta el proyecto.

fileFiles: Archivo generado automáticamente que contiene la ruta absoluta de todos los archivos vernacular del proyecto (el archivo se genera con el nombre ".file" en el directorio del proyecto).

Opciones: lista de opciones seleccionadas en la ventana exportar.

Por más detalles del comando coqdoc se recomienda ejecutar el comando "coqdoc --help", donde se describen cada una de las opciones.

5.3.9 Query

La clase `CoqActionQueries`, heredera de `CoqActionPadre`, es la invocada cuando el usuario acciona la funcionalidad Query. Como ya se vio, esto puede ocurrir mediante la opción que se encuentra en el menú de la herramienta o mediante el atajo por teclas correspondiente, motivo por el cual esta clase extiende a `CoqActionPadre`.

`CoqActionQueries` lo único que hace es ejecutar un hilo cuyo código reside en el mismo archivo bajo la clase `OtroHilo`, que extiende `Thread` por ser un hilo. Este hilo simplemente despliega una ventana, `DialogoModal` extensión de `JFrame` (Notar que está implementado con Swing y no con SWT como el resto de Eclipse, esto se hizo así como prueba de concepto), esta ventana permite al usuario seleccionar o ingresar un comando a consultar. La consulta se hará sobre el contexto activo, ejecutando el comando seleccionado, el cual se obtiene a través de la fábrica de contextos `CoqFabricaContexto`.

5.3.10 Plantillas o Templates

Como se vio previamente, la herramienta dispone de un archivo, templates.xml, que dispone de todas las plantillas que se le presentarán al usuario una vez que presione la combinación ctrl + espacio.

La clase `Activator`, que extiende a `AbstractUIPlugin`, es la que controla el ciclo de vida del plug-in. Al iniciarse Eclipse se cargará el plug-in correspondiente a la herramienta Coop. Para esto Eclipse invocará al método start del `Activator`. Es aquí cuando a través del método privado cargarTemplates se parsea el archivo templates.xml generando la lista de plantillas, las cuales se setean en variables dispuestas para tal fin en la clase `CoqCompletionProcessor` que implementa la interfaz `IContentAssistProcessor`. Esta clase provee a Eclipse la lista de "opciones" a sugerir al usuario al invocarse al método computeCompletionProposals. Ahora bien, ¿Cómo sabe Eclipse que debe usar esta clase? Pues, la implementación de `TextEditor` correspondiente a Coop, `CoqEditor`, en su constructor crea y guarda una instancia de `CoqConfiguration` que es una extensión de `SourceViewerConfiguration` y es esta configuración la que proporciona a Eclipse, con el método getContentAssistant, una implementación de `IContentAssistant` (`ContentAssistant`) que está asociada con una instancia de `CoqCompletionProcessor`, cerrando así el círculo.

La siguiente imagen (Fig. 25) pretende ilustrar todas estas relaciones entre clases.

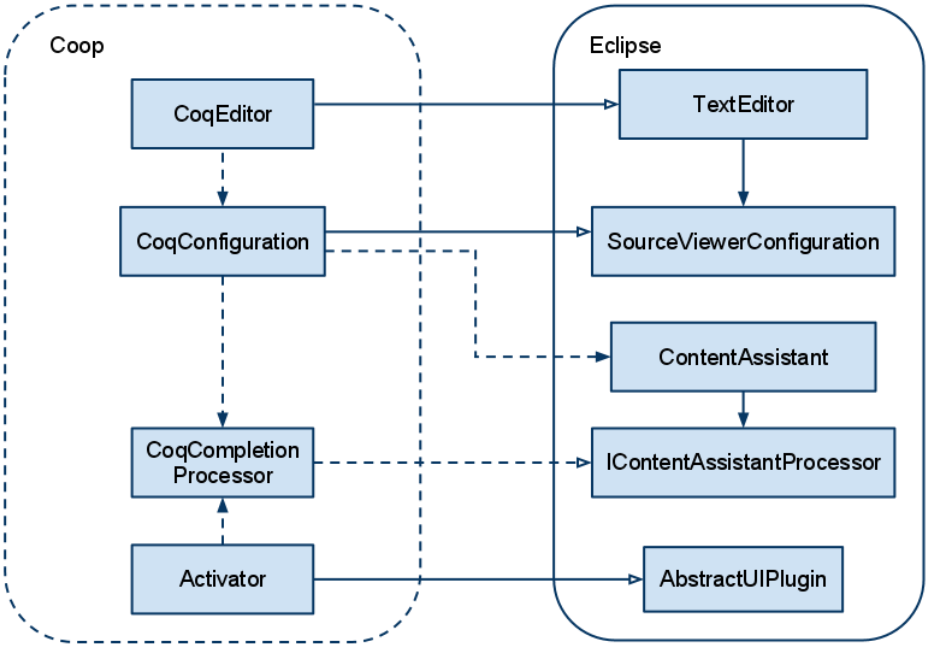

Fig. 25: Diagrama de clases para plantillas

5.3.11 Navegación a definiciones

La clase `CoqNavegarAction` heredera de `CoqActionPadre` es la encargada de manejar la acción disparada por el usuario vía atajo por teclas, menú de la herramienta o botón en la barra de herramientas (por eso extiende a `CoqActionPadre`).

En primer lugar, esta clase, verificará si en la configuración de la herramienta el usuario ha configurado que se despliegue, al navegar, la ayuda por palabra (ver solución más adelante), que en este caso mostrará los temas relativos a la palabra seleccionada para realizar la navegación a su definición. En caso afirmativo se abrirá la vista correspondiente a la ayuda dinámica. Luego de esto, simplemente delega al contexto activo toda la responsabilidad.

El contexto resolverá el problema en forma recursiva. Cada contexto realiza, al abrirse un archivo, un parseo del archivo que tiene asociado y entre otras cosas guarda los nombres de las variables declaradas y las dependencias en el archivo. La lista de variables se usa para determinar si la variable buscada para navegar se encuentra o no declarada en el archivo. Si la variable se encuentra definida en el archivo se procede a cambiar el foco al archivo donde reside la declaración y a seleccionar la línea donde, justamente, se encuentra la declaración. En caso contrario, se abre de a uno cada archivo del que depende el asociado al contexto activo con la intención de que se generen los contextos correspondientes (y en consecuencia la lista de definiciones y dependencias de este nuevo archivo) y así poder realizar las llamadas recursivas necesarias hasta encontrar la declaración buscada.

5.3.12 Árbol de derivaciones

Para implementar el árbol de derivación se implementó la clase `CoqDerivacion` que extiende a `ViewPart` y es un punto de extensión, en particular extiende una vista.

Previamente se vio que a la hora de ejecutar una instrucción el contexto lanza un hilo, implementado por la clase `HiloHayError`, que se subscribe para recibir la salida producida por el comando recién ejecutado y así determinar si ha ocurrido o no un error. Pero esta no es su única

función, también notifica la salida producida a la derivación activa (instancia de la clase `CoqDerivacion`), que no es más que la derivación asociada al contexto activo, con esta información `CoqDerivacion` genera el árbol a mostrar.

Como la derivación se despliega frente al usuario en forma de árbol, la clase `CoqDerivacion` usa una instancia de `TreeViewer` que al igual que en el outline (visto más arriba) necesita de una implementación de `ITreeContentProvider` y de `ILabelProvider`. Este rol lo asumen las clases `CoqDerivacionContentProvider` y `CoqDerivacionLabelProvider`, respectivamente. Estas clases trabajan sobre la representación en forma de árbol de la derivación generada por `CoqDerivacion`, que está dada por la clase abstracta `CoqItemDerivacion`, que es extendida por las clases: `CoqNodoDerivacion` (también abstracta) y `CoqHojaDerivacion`. Mientras que `CoqNodoDerivacion` es extendida por: `CoqDerivacionPrueba` y `CoqDerivacionHipotesis`.

A continuación se presenta una ilustración (Fig. 26) con la intención de aclarar las relaciones entre las clases involucradas.

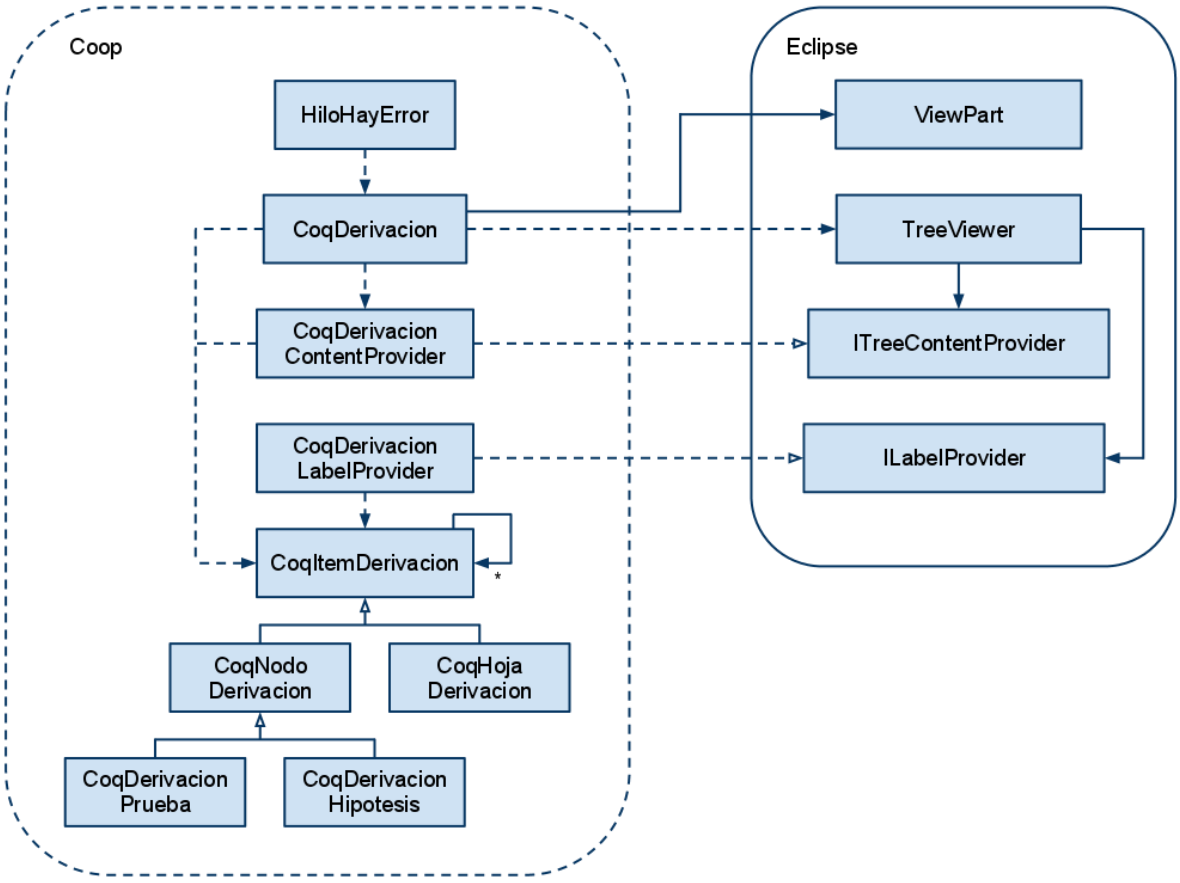

Fig. 26: Diagrama de clases para árbol de derivación

5.3.13 Exportar derivación

La implementación de este requerimiento consiste en recorrer el árbol modelado por la clase `CoqItemDerivacion` y sus correspondientes herederos generando, dependiendo de qué estructura se trata, si es una prueba, una hipótesis, una hoja, etc. El código, LaTex (para lo que se requiere tener instalado el paquete Semantics [10]) o XML correspondiente, sujeto al tipo de

exportación que seleccionó el usuario. La selección del tipo de exportación se ha de realizar mediante un par de botones incorporados a la vista implementada por `CoqDerivacion` que es la clase encargada de implementar este requerimiento en las rutinas manejadoras de los eventos asociados a los botones recién mencionados.

5.3.14 Ayuda

Este requerimiento refiere a la ayuda estática; es decir, la que se accede en el propio Eclipse mediante el menú Help->Help Contents. La implementación del mismo consiste básicamente en definir archivos de configuración y xml con la información, ya que este es otro punto de extensión que brinda Eclipse. Al igual que para los otros casos este punto de extensión se define en el archivo plugin.xml con la declaración (Fig. 27):

```
<extension point="org.eclipse.help.toc">
      <tocProvider class="coq.help.TocProvider"/>
</extension>
```

Fig. 27: Configuración de ayuda en plugin.xml

La tag "tocProvider" determina la clase que se utiliza para proveer el archivo toc.xml (se implementó la clase `TocProvider` que extiende de `AbstractTocProvider` y la clase `CoqTocFile` que extiende de `TocFiles`, esta última se utiliza para "representar" el archivo toc). Este archivo es el que Eclipse utiliza para mostrar la ayuda. El archivo toc.xml (ver Fig. 28) es un archivo que se genera automáticamente (ver generación de ayuda) y contiene la estructura de la ayuda. El formato del archivo es el siguiente:

```
<toc label="Table of Contents">
      <topic
            href="Reference-Manual001.html"
            label="Introduction">
            <topic
                  href="Reference-Manual001.html#toc1"
                  label="How to read this book">
            <topic/>
      </topic>
</toc>
```

Fig. 28: toc.xml

Como se puede observar el archivo consta de dos tipos de tags, la tag toc y la tag topic. La tag toc es la raíz del xml y no tiene efectos de visualización en el árbol de ayuda. Por otra parte la tag topic se utiliza para construir el árbol de ayuda (notar que existe tag topic como hijo de otra tag topic), el atributo label se muestra como nombre del ítem en el árbol y el atributo href refiere al archivo html que se muestra como consecuencia de seleccionar el ítem.

5.3.15 Ayuda por palabra clave

A diferencia del caso anterior la ayuda en este caso es dinámica, es decir, que la ayuda que se muestra depende del contexto, puntualmente de la "palabra" que esté seleccionada. Esta ayuda también se implementa utilizando un punto de extensión de Eclipse, el punto de extensión org.eclipse.help.contexts. La definición en el archivo plugin.xml es la siguiente (Fig. 29):

```
<extension point="org.eclipse.help.contexts">
      <contexts
            file="context.xml"
            plugin="Coq">
      </contexts>
</extension>
```

Fig. 29: Configuración de ayuda dinámica en plugin.xml

Como se puede observar, en este cuadro se hace referencia al archivo context.xml. En este archivo se define el contenido de la ayuda. Para el caso de Coop la ayuda es variable por lo que el archivo context.xml está vacío en lo referente a información de ayuda (sólo contiene el "Encabezado" que se muestra en la ayuda). Para poder mostrar la información adecuada en la vista de ayuda (recordar que para este caso lo que se muestra son las entradas en la documentación de Coq que refiere a la palabra seleccionada) se implementaron tres clases. A continuación se describe cada una de estas clases:

- `SelectionHelpResource`: Implementa la interfase `IHelpResource`. Esta clase representa el elemento que se muestra en la vista de ayuda, por lo que contiene la label que se muestra y la url al archivo html a mostrar como contenido.
- `CoqContextProvider`: Implementa la interfase `IContextProvider`. Esta clase devuelve el contexto, devuelve una instancia de la clase `SelectionContext`. Esta última es quien provee la información con la ayuda.
- `SelectionContext`: Implementa la interfase `IContext2`, como se dijo esta clase se devuelve como contexto, contiene la información con la ayuda propiamente dicha. Cuando se instancia se carga un array de `SelectionHelpResource` con las entradas de ayuda relevantes al texto seleccionado. Lo que se visualiza como ayuda son los elementos del array.

Notar que para buscar las entradas de ayuda asociadas a la palabra seleccionada se buscan en el archivo help.xml (ver generación de ayuda)

5.3.16 Generación de Ayuda

Si bien este no es un requerimiento es un comportamiento interno del plug-in que está vinculado directamente con la ayuda, tanto estática como dinámica. Lo que se muestra como ayuda estática y dinámica es básicamente la documentación de Coq que se obtiene del sitio web de Coq. Para poder mostrarla como ayuda es necesario en primer instancia descargar el contenido y luego cargar los archivos de ayuda necesarios (toc.xml y help.xml). A continuación se describe cada proceso:

- Descargar Contenido: Si bien se podría acceder a la documentación de Coq de forma online, se decidió descargar el contenido para que la ayuda esté disponible en el plug-in incluso cuando no se esté conectado a Internet. Para esto se utilizó la librería "HTML Parser" [23], puntualmente se accede a cada página HTML referenciada en la documentación de Coq (se accede a la página principal y se navega recursivamente por todos los links) y se descarga el contenido de forma local (luego se referenciará a esta página descargada).

- Generación Toc.xml: Una vez descargada la ayuda se genera el archivo Toc.xml (archivo utilizado por la ayuda estática). Para esto se lee la página índice de la documentación de Coq (http://coq.inria.fr/refman/toc.html) y se genera el xml con cada una de las entradas que figuran en dicha página (las entradas se diferencian por las tag HTML). Puntualmente se reconoce la tag div con id "content" y para cada tag table (en el div content) se genera una tag topic en el archivo toc.xml (el atributo label y href del topic se obtienen también de la tag table), luego se reconoce la tag ul (se asume que la tag ul persigue a una tag table) y para cada tag li (en la ul anterior) se genera también una tag topic (hija de la topic anterior) con la información asociada a dicha tag li.
- Generación Help.xml: Luego de generar el archivo Toc.xml se genera el archivo help.xml. Este archivo se lee cada vez que se inicia el plug-in Coq y su contenido queda cargado para utilizarse como ayuda dinámica. Al igual que para el caso anterior se accede a la documentación de Coq y utilizando la librería "HTML Parser" se reconocen las tags adecuadas. Para cada tag se genera una entrada en el archivo help.xml con la información de la ayuda.

Este proceso se realiza la primera vez que se inicia Eclipse (la primera vez que se inicia el plug-in Coq) o bien si se setea la opción "Actualizar ayuda desde URL" en preferencias.

5.3.17 Grafo de dependencias

Este requerimiento permite visualizar de forma gráfica mediante un grafo, las dependencias existentes en un proyecto Coq. El concepto de dependencia refiere al hecho de que un archivo fuente (vernacular) dependa de: otro archivo fuente, un archivo compilado (extensión vo), o bien una librería básica de Coq.

La implementación del requerimiento se divide en dos partes bien diferentes. La primera, donde se obtienen las dependencias para cada archivo fuente (vernacular), y la segunda donde a partir de las dependencias anteriores se "dibuja" el grafo. A continuación se detalla cada una de las partes antes mencionadas:

- Obtención de Dependencias: Para obtener las dependencias que presenta cada archivo fuente (vernacular) se identifican las declaraciones de dependencias que se definen explícitamente en el propio archivo. Es de importancia mencionar que cada vez que un archivo fuente es "abierto" se realiza un parseo del mismo para obtener las instrucciones Coq definidas y eliminar los comentarios entre otras acciones. Con este parseo se identifican también las declaraciones de dependencias definidas. Estas dependencias son almacenadas por el contexto. Por lo que para resolver esta primera tarea (obtención de dependencias) se realizan las siguientes acciones:
- Se obtienen todos los archivos fuentes (vernacular) existentes bajo los directorios definidos en el loadPath del proyecto Coq.
- Cada archivo obtenido de la parte anterior se abre y se obtienen las dependencias que el mismo presenta (recordar que al abrir el archivo se realiza el parseo donde, entre otra acciones, se obtienen las dependencias).

De este modo se obtiene entonces, para cada archivo fuente, la lista de archivos fuentes, compilados o librerías de que depende (puntualmente se realiza un mapeo archivo → [lista dependencias]).

Notar que al realizar el parseo se identifica el tipo de dependencia, dependiendo de la instrucción Coq que se utiliza la instrucción Load determina que se depende de un archivo fuente; y la instrucción Require determina que se depende de un archivo compilado. Este tipo se utilizará para realizar distinciones a la hora de dibujar el grafo.

Si el usuario seleccionó que desea visualizar el grafo de dependencias sin discriminar por directorio, la tarea de obtención de dependencias finaliza. De lo contrario (en las preferencias de Coop se seleccionó mostrar el grafo de dependencias por directorio) se continúa con las siguientes acciones:

- De todos los archivos vernacular obtenidos se consideran solo los que se encuentran bajo el directorio en que se está trabajando (el directorio del proyecto se considera el directorio origen).
- Se resuelven las dependencias entre los directorios hijos (definidos en el loadpath) al directorio en que se está trabajando. Se entiende por dependencias entre el directorio A y B que existe algún archivo en A (o subdirectorios de A) que dependa de algún archivo en B (o subdirectorios de B).

Como resultado se tienen entonces las dependencias de los archivos bajo el directorio en el que se está trabajando y las dependencias entre directorios hijos del anterior (al igual que para el caso anterior se tiene un mapea archivo/directorio → [lista dependencias]).

Notar que el concepto de directorio en el que se está trabajando existe por el hecho de que al seleccionar la opción de mostrar el grafo por directorio, es posible (y es la idea) navegar por los directorios y subdirectorios.

Dibujar Grafo: Una vez obtenidas las dependencias para cada archivo/directorio se dibuja el grafo propiamente dicho. Para esto se utiliza el framework Jung [15]. No es de interés en este documento detallar el uso de dicho framework, por lo que solo se mencionarán las características que se consideran relevantes.

Para dibujar el grafo se utiliza la clase `VisualizationViewer` del framework arriba mencionado, esta clase extiende de la clase `JPanel` de Swing por lo que básicamente la visualización del grafo se realiza embebiendo Swing en el IDE Eclipse. Sobre esta clase (`VisualizationViewer`) es donde se define los elementos del grafo (nodos y aristas), la conflagración de los mismos (color, tamaño, etc.) y el comportamiento o acciones que se podrán realizar sobre estos elementos. Para representar los nodos en el grafo (el framework permite utilizar cualquier clase como nodo) se implementó la clase "Vertex" que contiene la información relevante a cada nodo:

- Nombre: Nombre del archivo que representa el nodo.
- Path: Ruta absoluta del archivo que representa el nodo.
- Tipo: Tipo de archivo que representa el nodo (Archivo fuente, Dependencia a Archivo Fuente, Dependencia a Archivo Compilado o Dependencia a Librería).

De este modo para cada para archivo → [lista dependencias] obtenido de la etapa anterior se crea un nodo que representa al archivo y se crea también un nodo para cada archivo dependencias. Dependiendo del tipo de nodo (archivo, dependencia fuente, dependencia compilado y dependencia librería) se setea el color de nodo para diferenciar. Del mismo modo dependiendo del tipo de nodo se setean diferentes acciones para el evento "doble clic" sobre el nodo, puntualmente; si el nodo representa un archivo fuente o una dependencia a otro archivo del proyecto.

Notar que si el usuario selecciona en las preferencias de Coop "mostrar el grafo de dependencias por directorio", se "crean" dos `VisualizationViewer`, uno donde solo se muestran archivos y el segundo donde solo se muestran directorios. La "configuración" de ambas clases es similar.

Ahora bien, para poder visualizar el grafo como una vista de Eclipse se implementó la clase `CoqDependenceViewer` que extiende de la clase `ViewPart` (punto de extensión). En esta clase se "embebe" (si bien no directamente) el/los `JPanel VisualizationViewer`. Además de este componente se agregan también una label que muestra el nombre del directorio de trabajo y los botones de navegación; esta funcionalidad aplica si está seleccionada la opción "mostrar el grafo de dependencias por directorio", para poder ir adelante y atrás según se navegue por los directorios. Para poder implementar esta navegación en la clase `CoqDependenceViewer` se manejan dos listas para mantener el histórico de las navegaciones (hacia adelante y atrás). En estas listas se "guarda/n" el/los `VisualizationViewer` correspondiente/s al grafo parcial para el directorio de trabajo así como la label asociada al mismo. Al navegar entonces se carga el "grafo" de dicho histórico, por lo que no se vuelve a "regenerar".

También se agrega un botón para modificar el layout asociado al grafo. El framework consta con varios layout que permite visualizar el grafo de diferente forma.

A continuación se muestra un diagrama (Fig. 30) con las clases involucradas en este requerimiento:

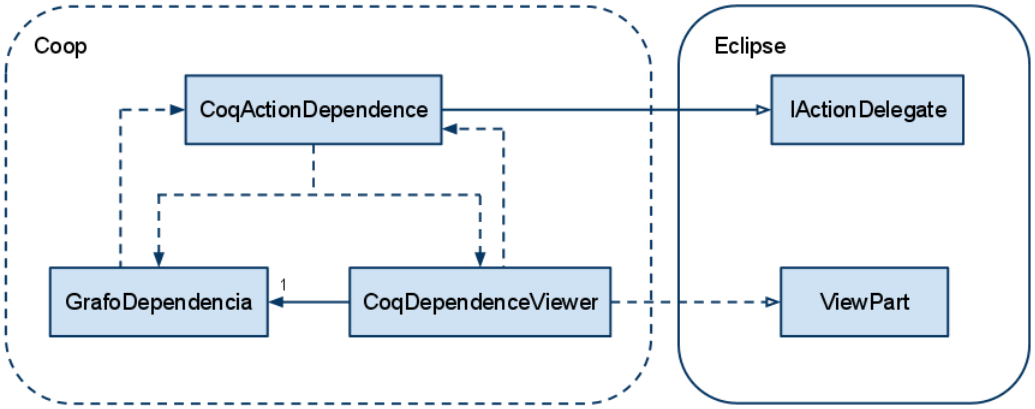

Fig. 30: Diagrama de clases para grafo de dependencias

Capítulo 6

Conclusiones y Trabajo Futuro

6.1 Conclusiones

En este proyecto se implementó un IDE para el asistente de pruebas Coq integrable al entorno de desarrollo Eclipse, con una gran variedad de funcionalidades. Dentro de las funcionalidades que brinda Coop al usuario, se encuentran las principales ofrecidas por la mayoría de las herramientas disponibles actualmente. Asimismo, brinda una buena cantidad de funcionalidades innovadoras que hacen de Coop un entorno de desarrollo para Coq en Eclipse mucho más amigable y con potencial para el usuario en pro de facilitar el uso de Coq.

Habiendo recorrido los requerimientos y las soluciones ofrecidas por Coop, resta comparar esta nueva herramienta con sus similares disponibles en la actualidad. Para la comparación, teniendo en cuenta las funcionalidades implementadas por Coop, se confeccionó un cuadro (Tabla 1) en el que se aprecia qué funcionalidades implementa cada una de las herramientas relevadas.

Dado que los requerimientos de Coop fueron extraídos, en una primera instancia, del relevamiento de asistentes Coq pre-existentes, es completamente lógico que al menos en los requerimientos básicos el cuadro comparativo muestre que varias de las herramientas, además de Coop, implementan estas funcionalidades. Sin embargo, al observar las funcionalidades derivadas de las variadas propuestas provenientes de los tutores, de los propios desarrolladores, así como de usuarios con mayor experiencia en el uso de este tipo de herramientas, se nota una disparidad interesante que evidencia la innovación, en algunos aspectos, que representa Coop.

En la tabla Tabla 1 se realiza un análisis comparativo de las herramientas:

Funcionalidad\IDE	CoqIDE	Proof General	Proof Web	Prover Editor	PCoq	Coop
Avanzar Paso	✓	✓	✓	✓	✓	✓
Retroceder Paso	✓	✓	✓	✓	✓	✓
Ejecutar hasta Cursor	✓	✓	✓	✗	✗	✓
Ejecutar hasta el Final	✓	✓	✓	✓	✗	✓
Deshacer Ejecución	✓	✓	✓	✓	✗	✓
Ejecutar bloque	✗	✗	✗	✗	✗	✓
Ejecutar con el Punto	✗	✗	✓	✗	✗	✓
Interrumpir	✓	✓	✓	✓	✗	✓
Display	✓	✗	✓	✗	✓	✓
Query	✓	✗	✓	✗	✗	✓
Sintax Highlighting	✓	✓	✗	✓	✗	✓
Template	✓	✓	✓	✗	✗	✓
Autocompletar	✓	✗	✗	✗	✗	✓
Indentar	✗	✗	✗	✗	✗	✓
Comentar	✗	✗	✗	✗	✗	✓
Parentización	✗	✗	✗	✗	✗	✓
Navegar	✗	✗	✗	✗	✗	✓
Proof Wizard	✓	✗	✗	✗	✗	✗
Exportación Básica	✓	✗	✗	✗	✗	✓
Generación de Makefile	✗	✗	✗	✗	✗	✓
Bifurcación	✗	✗	✗	✗	✗	✓
Proof by Pointing	✗	✓	✗	✗	✓	✗
Vista TopLevel	✓	✓	✗	✓	✓	✓
Vista Outline	✗	✗	✗	✓	✗	✓
Visualizar y Exportar Derivación	✗	✗	✗	✗	✗	✓
Visualización de Dependencias	✗	✗	✗	✗	✗	✓
Ayuda Estática	✗	✓	✗	✗	✗	✓
Ayuda Dinámica	✗	✗	✗	✗	✗	✓
Manejo de Varios Archivos	✗	✓	✗	✗	✓	✓
Edición y Pintado	✓	✓	✗	✓	✓	✓
Visualización de Símbolos	✗	✓	✗	✗	✓	✗
Expandir y Contraer Bloques	✗	✗	✗	✗	✗	✓

Tabla 1: Comparación de herramientas

Esta tabla destaca la gran variedad de funcionalidades que ofrece Coop y que ningún otro IDE hasta el momento ofrece. Consideramos que estas son las funcionalidades más potentes de Coop, y por esto, junto con muchas otras características entre las que no se puede obviar las heredadas de Eclipse, notamos que este plug-in es una herramienta mucho más amigable al usuario y útil que muchas de las que se encuentran disponibles.

En lo relativo al desempeño de la herramienta, ésta ha mostrado tiempos de respuesta muy aceptables, sobre todo en comparación con el CoqIDE. No obstante en la próxima subsección se menciona una excepción.

Se realizaron algunas pruebas sobre Coop y CoqIDE interesantes de mencionar. Por ejemplo, se ha podido probar con proyectos Coq de gran porte, es decir, con una buena cantidad de archivos y con una gran cantidad de instrucciones por archivo (del orden de las diez mil líneas), teniendo, aún así, un buen desempeño incluso presentando mejores tiempos de respuesta que CoqIDE.

Además es bueno destacar que el desarrollo de Coop fue realizado para la última versión disponible de Coq (la 8.2) al momento de comenzar el desarrollo. Posteriormente, cerca de la finalización del proyecto, se liberó una nueva versión del asistente de pruebas (la 8.3). Esto ameritó la realización de pruebas con ánimo de establecer el comportamiento de la herramienta frente a esta nueva versión. Afortunadamente Coop respondió de muy buena manera sin tener que realizar cambios significantes, ni en la implementación, ni en la configuración.

Algo similar sucedió con la versión de Eclipse utilizada, que en un comienzo fue Galileo 3.5, no obstante en la actualidad se encuentra disponible la versión Helios 3.6. También se probó Coop con esta nueva versión obteniendo un comportamiento satisfactorio.

6.2 Problemas Conocidos

En esta sección se describen los problemas conocidos de Coop.

Uno de los problemas más notorios de la herramienta Coop está vinculado a la performance a la hora de deshacer la ejecución de una instrucción, más concretamente, en la implementación del requerimiento paso a paso hacia atrás cuando se ha ejecutado una gran cantidad de instrucciones.

Existe otro defecto, o más bien una restricción en la estructura de los archivos que se ejecutan con la herramienta. Sabido es que una instrucción puede ocupar varias líneas de texto o bien puede haber varias instrucciones en una misma línea. Coop soporta perfectamente instrucciones que abarquen varias líneas, pero no presenta un comportamiento bien definido a la hora de ejecutar archivos donde en una misma línea se presenta más de una instrucción. Si bien el resultado de la ejecución es correcto, la presentación gráfica asociada presenta algunos problemas. Éstos se deben tener presentes a la hora de confeccionar un archivo si se pretende un correcto funcionamiento de la herramienta en lo concerniente a las funcionalidades relativas a la ejecución.

6.3 Trabajo Futuro

No todos los requerimientos relevados y sugeridos entraron en el alcance del proyecto, por lo que un subconjunto de estos queda como trabajo a futuro.

Algunos de los requerimientos que aquí se presentan son enteramente nuevos, mientras que otros son extensiones de algunos ya implementados, como es el caso de las teclas rápidas,

donde el usuario podrá configurar las combinaciones de teclas a utilizar a través de la interfaz de configuración correspondiente a la herramienta en Eclipse.

En la configuración el editor debería permitir, además de lo que ya permite, cargar scripts de arranque coqrc e incluso permitir configurar fácilmente estos scripts.

En la indentación se debe permitir configurarla a partir de plantillas que indiquen la manera de indentar cada construcción de manera independiente. Además de permitir que al presionar la tecla Tab por primera vez en una línea se indente en la misma línea que la línea anterior (independientemente del tamaño de tabulación) y que al presionar la combinación M-Tab se decremente la indentación.

En el árbol de derivaciones se podría permitir realizar el UNDO y REDO desde diferentes puntos de la derivación (considerar múltiples caminos) sin necesidad de rehacer la prueba a mano.

Otra funcionalidad deseable es la detección de dependencias. Es decir, al utilizar alguna función o táctica que no esté definida en el contexto, se la debería buscar en las librerías e incluir la dependencia automáticamente, o sugerir cuál de las dependencias incluir.

El chequeo sintáctico de errores es otro de los requerimientos que quedaron por fuera del alcance del proyecto. Este consiste en realizar el chequeo sintáctico de errores en código al momento en que se va escribiendo el código sin tener que esperar a la ejecución del mismo para saber si hay errores sintácticos. Similar, de algún modo, a este tenemos la compilación con recorrida por los errores que, básicamente, consiste en permitir al usuario compilar el buffer (archivo) y avanzar de un error a otro, paso a paso.

Por otra parte, el proof wizard podría permitir probar tácticas automáticas básicas para resolver una prueba. Se podría además permitir configurar una serie de tácticas para aplicar automáticamente.

Cada táctica Coq se puede aplicar a un objetivo de prueba que tiene cierta forma. Por ejemplo SPLIT se aplica si se tiene un AND de dos expresiones y el efecto es que se generan dos objetivos de prueba independientes (uno para cada expresión). Una idea es evaluar de antemano la forma de una expresión y poder asistir al desarrollador de las pruebas sugiriéndole qué tácticas podría llegar a aplicar, algo muy similar al proof by pointing encontrado en otras herramientas.

Cambiar el objetivo de una prueba, es otro de los requerimientos para trabajo a futuro. Este requerimiento consiste en seleccionar con el mouse un objetivo de prueba diferente del que está activo y pasar directamente a la demostración del segundo (se utiliza la táctica focus).

Otra funcionalidad deseable sería poder realizar drag and drop de una parte de una demostración para demostrar otra rama en caso de que los objetivos de prueba sean iguales. Junto con eso, se podría detectar automáticamente cuando el objetivo de prueba ya fue probado antes y que utilice la demostración original para ello.

Por último consideramos integrar el lenguaje de definición de tácticas de manera natural en la herramienta para facilitar la definición al verificador. Dejar disponible una lista rápida de tácticas definidas por el usuario de común aplicación en el contexto de un proyecto.

Además de considerar estos requerimientos es de interés destacar que Coop será utilizado como IDE para Coq en el curso TPPSF del INCO.

Referencias

[1] **INRIA**. The Coq Proof Assistant. [En línea] [Citado el: 16 de 04 de 2011.] http://coq.inria.fr/.

[2] —. ProverEditor v.0.1.0. [En línea] 27 de 09 de 2010. [Citado el: 16 de 04 de 2011.] http://provereditor.gforge.inria.fr/.

[3] **Aspinall, David**. Proof General. [En línea] University of Edinburgh, 9 de 03 de 2011. [Citado el: 16 de 04 de 2011.]http://proofgeneral.inf.ed.ac.uk/.

[4] **The Coq Development team**. Chapter 15- Coq Integrated Development Environment. The Coq Proof Assistant Reference Manual - Version 8.2. : INRIA, 2006.

[5] **Eclipse Foundation, Inc**. Eclipse - The Eclipse Foundation open source community website. [En línea] [Citado el: 16 de 04 de 2011.] http://www.eclipse.org/.

[6] **The Coq Development team**. The Coq Proof Assistant Reference Manual - Version 8.3. s.l. : INRIA, 2006.

[7] Coq - Wikipedia, la enciclopedia libre. [En línea] 04 de 11 de 2010. [Citado el: 16 de 04 de 2011.]http://es.wikipedia.org/wiki/Coq.

[8] **Aspinall, David**. Proof General user manual. [En línea] 11 de 10 de 2010. [Citado el: 16 de 04 de 2011.]http://proofgeneral.inf.ed.ac.uk/userman.

[9] **Paulin-Mohring, C**. Inductive definitions in the system Coq - rules and properties. s.l. : LNCS 664, 1993.

[10] **Coquand, T y Huet, G**. Information and Computation 76(2/3). The calculus of constructions. 1998.

[11] **Bertot, Yves**. Pcoq: a beginner's tutorial. [En línea] 01 de 02 de 2000. [Citado el: 16 de 04 de 2011.] http://www-sop.inria.fr/lemme/pcoq/docs/begin.html.

[12] —. Pcoq : A java-based user-interface for Coq. [En línea] 11 de 02 de 2003. [Citado el: 16 de 04 de 2011.]http://www-sop.inria.fr/lemme/pcoq/.

[13] **Kaliszyk, Cezary, y otros**. Deduction using the ProofWeb system. Radboud University Nijmegen and the Free University Amsterdam

[14] ProofWeb. [En línea] [Citado el: 16 de 04 de 2011.] http://prover.cs.ru.nl/.

[15] **Daniel Winterstein, David Aspinall and Christoph Lüth**. Proof General / Eclipse: A Generic Interface for Interactive Proof. University of Edinburgh, U.K. and Universität Bremen, Germany, 2005.

[16] **Gast, Holger**. Engineering the Prover Interface. Wilhelm-Schickard Institut für Informatik, 2010.

[17] **Mulhern, Anne, Fischer, Charles y Liblit, Ben**. Tool Support for Proof Engineering. Computer Sciences Department - University of Wisconsin-Madison - Madison, WI USA, 2006.

[18] **InCo**. COAL [En línea] [Citado el: 20 de 04 de 2011] http://www.fing.edu.uy/inco/grupos/coal

[19] —. The Formal Methods Group [En línea] [Citado el: 20 de 04 de 2011] http://www.fing.edu.uy/inco/grupos/mf/

[20] **Peter Møller Neergaard, Arne John Glenstrup**. The semantic Packaged - June 27, 2005

[21] Emacs - Wikipedia, la enciclopedia libre. [En línea] [Citado el: 20 de 04 de 2011.] http://es.wikipedia.org/wiki/Emacs

[22] **Sourceforge**. JUNG - Java Universal Network/Graph Framework [En línea] [Citado el: 20 de 04 de 2011] http://jung.sourceforge.net/

[23] —. HTML Parser [En línea] [Citado el: 20 de 04 de 2011] http://htmlparser.sourceforge.net/

[24] **Prashant Deva**. Folding in Eclipse Text Editors - March 11, 2005 [En línea] [Citado el: 20 de 04 de 2011.] http://www.eclipse.org/articles/Article-Folding-in-Eclipse-Text-Editors/folding.html

[25] **Vim.** Coq indent: Indentation for coq files [En línea] [Citado el: 20 de 04 de 2011.] http://www.vim.org/scripts/script.php?script_id=2079

[26] —. Coq syntax: Syntax colouring for coq files [En línea] [Citado el: 20 de 04 de 2011.] http://www.vim.org/scripts/script.php?script_id=2063

[27] **Eclipse**. Eclipse documentation - Current Release [En línea] [Citado el: 20 de 04 de 2011.] http://help.eclipse.org

[28] **Carlos Luna, Gustavo Betarte**. Taller de Producción de Programas sin Fallas [En línea] [Citado el: 20 de 04 de 2011.] http://www.fing.edu.uy/inco/grupos/mf/TPPSF/

[29] **The Coq Development team**. Chapter 13 - The Coq commands. The Coq Proof Assistant Reference Manual - Version 8.2. : INRIA, 2006.

[30] **France, R. and Rumpe, B.** Model-driven Development of Complex Software: A Research Roadmap. In Proceedings of the 29th Int. Conference on Software Engineering (ICSE) (Minneapolis, USA, 2007). IEEE.