



ESCUELA SUPERIOR DE INGENIERA

Segundo Ciclo en Ingeniera
Informtica

XMLEye: transformador y visor genrico
de documentos estructurados

Curso 2007-2008

Antonio Garca Domnguez
Cdiz, 16 de junio de 2010



ESCUELA SUPERIOR DE INGENIERA

Segundo Ciclo en Ingeniería
Informática

XMLEye: transformador y visor genérico
de documentos estructurados

DEPARTAMENTO: Lenguajes y Sistemas Informáticos.

DIRECTORES DEL PROYECTO: Francisco Palomo Lozano e
Inmaculada Medina Buló.

AUTOR DEL PROYECTO: Antonio García Domínguez.

Cádiz, 16 de junio de 2010

Fdo.: Antonio García Domínguez

Este documento se halla bajo la licencia FDL (Free Documentation License). Segn estipula la licencia, se muestra aqu el aviso de copyright. Se ha usado la versin inglesa de la licencia, al ser la nica reconocida oficialmente por la FSF (Free Software Foundation).

Copyright ©2008 Antonio Garca Domnguez.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Índice general

Índice general	5
Índice de figuras	13
Índice de cuadros	15
1. Introduccin	17
1.1. Metalenguajes en el intercambio de informacin	17
1.2. Formatos de intercambio de informacin que no se basan en metalenguajes	19
1.3. Objetivos	20
1.4. Alcance	21
1.4.1. Limitaciones del proyecto	22
1.4.2. Licencia	22
1.5. Visin general	23
1.6. Glosario	24
1.6.1. Acrnimos	24
1.6.2. Definiciones	26
1.6.3. Otras tecnologas	31

2. Desarrollo del calendario	33
2.1. XMLEye	33
2.1.1. Primera iteracin: rediseo	33
2.1.2. Segunda iteracin: interfaz TDI	33
2.1.3. Tercera iteracin: hipervnculos entre documentos	34
2.1.4. Cuarta iteracin: descriptores de formatos de documentos	34
2.1.5. Quinta iteracin: estabilizacin	34
2.1.6. Sexta iteracin: paquete Debian	34
2.1.7. Sptima iteracin: manuales y traduccin al ingls	34
2.1.8. Octava iteracin: creacin de un wiki	35
2.2. ACL2::Procesador	35
2.2.1. Primera iteracin: rediseo	35
2.2.2. Segunda iteracin: eventos para libros	35
2.2.3. Tercera iteracin: grafos de dependencias	36
2.2.4. Cuarta iteracin: integracin con ACL2	36
2.2.5. Quinta iteracin: ejecutable monoltico y paquete Debian	36
2.2.6. Sexta iteracin: traduccin al ingls	36
2.2.7. Sptima iteracin: actualizacin para ACL2 v3.3	36
2.3. YAXML::Reverse	37
2.3.1. Primera iteracin: Procesamiento de YAML 1.0	37
2.3.2. Segunda iteracin: reorganizacin y traduccin	37
2.3.3. Tercera iteracin: ejecutable monoltico y paquete Debian	37
2.3.4. Cuarta iteracin: YAML 1.1/JSON y otras mejoras	37
2.4. Diagrama Gantt	37
2.5. Porcentajes de esfuerzo	38

3. Descripción general del proyecto	43
3.1. Perspectiva del producto	43
3.1.1. Entorno de los productos	43
3.1.2. Interfaces software y hardware	43
3.1.3. Interfaz de usuario	44
3.2. Funciones	44
3.2.1. XMLEye	44
3.2.2. ACL2::Procesador	44
3.2.3. YAXML::Reverse	45
3.3. Características del usuario	45
3.4. Restricciones generales	47
3.4.1. Control de versiones	47
3.4.2. Lenguajes de programación	48
3.4.3. Sistemas operativos y hardware	49
3.4.4. Bibliotecas y módulos usados	50
3.5. Requisitos para futuras versiones	53
3.5.1. XMLEye	53
3.5.2. ACL2::Procesador	54
3.5.3. YAXML::Reverse	54
4. Desarrollo del proyecto	55
4.1. Proceso	55
4.1.1. Origen	55
4.1.2. Características	56
4.2. Herramientas de modelado usadas	61
4.2.1. BOUML	61
4.2.2. UMLet	61
4.3. Requisitos	61
4.3.1. Interfaces externas	61
4.3.2. Funcionales	62

4.3.3. Atributos del sistema software	65
4.4. Analisis del sistema	66
4.4.1. Historias de usuario	66
4.4.2. Casos de uso	71
4.4.3. Modelo conceptual de datos del dominio de ACL2	80
4.4.4. Modelo conceptual de datos del dominio de YAML	85
4.5. Diseo del sistema	89
4.5.1. Arquitectura del sistema	89
4.5.2. Capa de filtrado: ACL2::Procesador	96
4.5.3. Capa de filtrado: YAXML::Reverse	108
4.5.4. Capa de aplicacin	114
4.5.5. Capa de presentacin	122
4.6. Implementacin	130
4.6.1. Perl	130
4.6.2. Java	132
4.6.3. Hojas XSLT	133
4.7. Pruebas y validacin	135
4.7.1. Pruebas en XP	135
4.7.2. Plan de pruebas	136
4.7.3. Diseo de pruebas	136
4.7.4. Especificacin de los casos de prueba	138
5. Resumen	157
5.1. Pruebas continuas	157
5.2. Diseo iterativo y dirigido por pruebas	158
5.3. Transformaciones declarativas dirigidas por datos	158
5.4. Mejora como producto software	159

6. Conclusiones	161
6.1. Valoracin	161
6.2. Mejoras y ampliaciones	162
6.2.1. Funcionalidad	162
6.2.2. Diseo	162
6.3. Otros aspectos de inters	163
7. Manual del usuario de XMLEye y conversores asociados	165
7.1. Instalacin de XMLEye	165
7.1.1. Requisitos previos	165
7.1.2. Instalacin desde distribuciones precompiladas	166
7.1.3. Instalacin desde paquetes Debian	168
7.1.4. Compilacin del cdigo fuente	170
7.2. Instalacin de conversores asociados	171
7.2.1. ACL2::Procesador: convertidor de demostraciones de ACL2 . . .	171
7.2.2. YAXML::Reverse: conversor de documentos YAML a XML	176
7.2.3. Instrucciones genricas	178
7.3. Uso y configuracin	179
7.3.1. Apertura y edicin de documentos	180
7.3.2. Navegacin por los documentos	180
7.3.3. Navegacin por la vista del nodo actual	182
7.3.4. Personalizacin de la presentacin	182
7.3.5. Personalizacin de los formatos aceptados	183

8. Manual del desarrollador	185
8.1. Cmo abrir nuevos formatos con XMLEye	185
8.1.1. Introduccin	185
8.1.2. Creacin de un convertidor	185
8.1.3. Creacin de un descriptor de formato	187
8.2. Cmo aadir nuevas visualizaciones de documentos y elementos	189
8.2.1. Introduccin	189
8.2.2. Estructura de los repositorios de hojas de usuario	190
8.2.3. Localizacin de una hoja de usuario	191
8.2.4. Herencia a partir de una hoja base	192
8.2.5. Ejemplo de hoja de preprocesado: xml	193
8.2.6. Ejemplo de hoja de visualizacin: xml	197
A. Gua de desarrollo de paquetes Debian	201
A.1. Introduccin	201
A.1.1. ¿Qu es un paquete Debian?	201
A.1.2. ¿Por qu se desarrollan paquetes?	201
A.1.3. ¿Quin desarrolla paquetes?	202
A.1.4. Notas acerca de esta gua	203
A.2. Preparativos	203
A.2.1. Cmo instalar las herramientas bsicas	204
A.2.2. Cmo conseguir paquetes limpios	205
A.2.3. Cmo simplificar la distribucin	206
A.2.4. Control de versiones	208
A.2.5. Autenticacin	210
A.2.6. Distribucin a travs de Internet	212
A.3. Creacin y mantenimiento del paquete	215
A.3.1. Adaptaciones previas al uso de Subversion	215
A.3.2. Preparacin de una primera versin	228
A.3.3. Actualizacin del paquete a una nueva versin del programa	234

A.4. Otros aspectos de inters	238
A.4.1. Integracin de repositorio propio con la jaula <i>chroot</i>	238
A.4.2. Sincronizacin de un repositorio en Internet con un repositorio local	239
A.4.3. Adaptacin de aplicaciones Java	239
A.4.4. Actualizacin del escritorio	241
A.4.5. Generacin automtica de paquetes	245
A.4.6. Otros formatos de paquete	247
B. GNU Free Documentation License	251
1. APPLICABILITY AND DEFINITIONS	251
2. VERBATIM COPYING	253
3. COPYING IN QUANTITY	253
4. MODIFICATIONS	254
5. COMBINING DOCUMENTS	256
6. COLLECTIONS OF DOCUMENTS	256
7. AGGREGATION WITH INDEPENDENT WORKS	257
8. TRANSLATION	257
9. TERMINATION	257
10. FUTURE REVISIONS OF THIS LICENSE	258

Índice de figuras

2.1. Diagrama Gantt de iteraciones	39
4.1. Prototipo de la ventana principal	63
4.2. Prototipo del dialogo de bsqueda	63
4.3. Prototipo del dialogo de gestin de descriptores de formatos de documentos	64
4.4. Prototipo del dialogo de informacin del programa	64
4.5. Diagrama de casos de uso	72
4.6. Diagrama de clases conceptuales general de ACL2	82
4.7. Diagrama de clases conceptuales de rdenes de ACL2	83
4.8. Diagrama de clases conceptuales de procesos de ACL2	84
4.9. Secuencia de procesamiento de YAML	87
4.10. Diagrama de clases conceptuales para YAML	88
4.11. Diagrama arquitectnico del sistema	91
4.12. Diagrama de componentes para visualizacin	94
4.13. Diagrama de clases de <code>ACL2::Procesador</code> (general)	98
4.14. Diagrama de secuencia general de filtrado	99
4.15. Diagrama de secuencia de filtrado de una <i>Demostracin</i>	100
4.16. Diagrama de clases de rdenes de <code>ACL2::Procesador</code>	104
4.17. Diagrama de clases de Procesos de <code>ACL2::Procesador</code>	106
4.18. Diagrama de secuencia de filtrado de <code>defthm</code> (anlisis)	109
4.19. Diagrama de secuencia de filtrado de <code>defthm</code> (generacin de salida) . . .	110
4.20. Representacin de grafos mediante rboles XML	112
4.21. Orculo de pruebas para <code>YAXML::Reverse</code>	113
4.22. Diagrama de clases de diseo para modelado de documentos	116

NDICE DE FIGURAS

4.23. Diagrama de clases de diseo para las hojas de usuario	119
4.24. Diagrama de diseo de clases de descriptores de formatos	120
4.25. Diagrama de clases de diseo de gestin de preferencias	122
4.26. Diagrama de secuencia de gestin de preferencias	123
4.27. Diagrama de clases de diseo de los modelos de presentacin	125
4.28. Diagrama de clases de diseo de bsquedas	127
4.29. Diagrama de clases de manejo de rboles	129
4.30. Diagrama de clases de rdenes	134
A.1. Pestaa de Software de Terceros	214
A.2. Pestaa de Autenticacin	215

Índice de cuadros

2.1. Cuadro de fechas y duraciones de las iteraciones	40
2.2. Porcentajes de esfuerzo: XMLEye	40
2.3. Porcentajes de esfuerzo: ACL2::Procesador	41
2.4. Porcentajes de esfuerzo: YAXML::Reverse	41
4.1. Prueba de aceptacin para “Visualizar XML genrico”	148
4.2. Prueba de aceptacin para “Hacer a XMLEye independiente de ACL2” . .	149
4.3. Prueba de aceptacin para “Visualizar y enlazar varios documentos entre s.”	150
4.4. Prueba de aceptacin para “Historial de documentos recientes”	151
4.5. Prueba de aceptacin para “Integrar editor”	152
4.6. Prueba de aceptacin para “Actualizar automticamente”	153
4.7. Prueba de aceptacin para “Guardar preferencias”	153
4.8. Prueba de aceptacin para “Integrar XMLEye y ACL2::Procesador” . .	154
4.9. Prueba de aceptacin para “Hojas summary y reverse”	154
4.10. Prueba de aceptacin para “Ver usos de una meta”	155
4.11. Prueba de aceptacin para “Integrar XMLEye y YAXML::Reverse” . . .	155
7.1. Accesos de teclado para navegacin por el rbol	181
7.2. Accesos de teclado para navegacin por la vista del nodo actual	182

1. Introduccin

1.1. Metalenguajes en el intercambio de informacin

Comunicar dos o ms aplicaciones entre s (en mquinas o espacios de memoria distintos, o incluso la misma aplicacin en ejecuciones separadas en el tiempo) significa, a grandes rasgos, intercambiar secuencias de bits que siguen una serie de reglas que les dan estructura y significado, es decir, que pertenecen a un *lenguaje*.

Evidentemente, en ambos extremos de la comunicacin debe de existir la lgica necesaria para recuperar esa estructura y significado a partir de los bits: los analizadores *lxicos* reconocen los lexemas que constituyen los elementos bsicos y los analizadores *sintcticos* y *semnticos* se ocupan de crear estructuras de datos ms elaboradas a partir de ellos.

Originalmente, tanto la gramtica con las reglas que describan la sintaxis de la mayora de los documentos, como el contenido que deban tener, eran propios de cada lenguaje. Esto originaba muchas complicaciones imprevistas al tener que integrar varias aplicaciones, posiblemente desarrolladas por diferentes organizaciones, y para entornos hardware y software distintos.

Las razones de este enfoque *ad hoc* eran las limitaciones en memoria y capacidad de proceso del hardware de la poca: primaba la eficiencia en recursos computacionales sobre los costes de desarrollo, as que los lenguajes usados eran un fiel reflejo (a veces demasiado) de las representaciones internas de los datos. Con el tiempo, el avance en la tecnologa alter este equilibrio para la gran mayora de aplicaciones que con el auge de Internet necesitaban ms que nunca garantizar el intercambio fiable y perdurable de la informacin con una mejor gestin de errores y comprobaciones ms estrictas.

Esto impuls la popularidad de los *metalenguajes*, que definan las reglas no ya de uno, sino de familias completas de lenguajes. Los desarrolladores podan partir de una base de reglas y herramientas ya existentes para definir y explotar sus lenguajes de intercambio de informacin.

Las familias de lenguajes definidas por los metalenguajes pueden tener diversos usos, no excluyentes entre s:

1. Introduccin

- Lenguajes de *marcado*, que emplean etiquetas para anotar el contenido de los documentos con informacin acerca del significado de sus elementos. Son lenguajes descriptivos, ms que procedimentales. Por ejemplo, en lugar de indicar que se imprima una serie de cadenas separadas por saltos de lnea de 16 puntos, al estilo de PostScript, indicaramos que dicho grupo de cadenas forman un prrafo.

Algunos de los primeros metalenguajes de marcado fueron GML (Generalized Markup Language) [?] (1973) y su sucesor, SGML (Standard Generalized Markup Language), estandarizado como ISO 8879:1986. HTML (Hyper Text Markup Language), sobre el que se apoya hoy en da toda la World Wide Web, es una derivacin de SGML.

Posteriormente, se emple SGML para definir XML (eXtensible Markup Language) [?], el metalenguaje por excelencia hoy en da. Sacrifica gran parte de la flexibilidad de SGML para simplificar su manipulacin automtica. Adems de las ventajas de los metalenguajes y de los lenguajes de marcado, XML permite definir vocabularios fcilmente extensibles por terceras partes que acomoden nueva informacin antes no prevista.

Uno de los ejemplos ms notables del uso de XML es el formato para documentos ofimticos ODF (Open Document Format), publicado como el estndar ISO 26300:2006, e implementado en diversos paquetes ofimticos, como OpenOffice, KOffice o Google Docs.

- Lenguajes de *serializacin* y *deserializacin* de estructuras de datos de memoria a un flujo de bytes y viceversa, respectivamente. Esto permite enviar dichas estructuras a travs de una conexin de red o guardarlas en ficheros, por ejemplo.

Aunque XML se usa ampliamente para este fin, hoy en da YAML (YAML Ain't a Markup Language) est reuniendo cada vez ms adeptos, por su sintaxis ms concisa y fcil de leer y su facilidad de uso. JSON (JavaScript Object Notation), un subconjunto de YAML 1.1 ms fcil de analizar pero menos legible, est cobrando fuerza en la comunidad de desarrolladores Web.

Algunos ejemplos del uso de YAML y JSON incluyen los volcados de bases de datos del entorno de desarrollo web Django, los descriptores de mdulos Perl y los marcadores del navegador web Mozilla Firefox en sus versiones 3.0 y posteriores.

Sin embargo, muchos de estos metalenguajes no fueron diseados para ser legibles por personas. Este es el caso de aquellos basados en XML o JSON, por ejemplo. Otras veces, incluso para metalenguajes ms legibles como YAML, los documentos son simplemente demasiado complejos o grandes.

El nico tipo de herramienta que puede resolver estos problemas es un visor, que rena informacin agregada en un formato fcil de comprender. Por otro lado, la creacin de un visor especfico para un lenguaje determinado implica un esfuerzo considerable, que slo se halla justificado si dispone de la «masa crtica» necesaria. Especializar un visor

genrico ya existente supone el mejor enfoque para los formatos con comunidades de usuarios ms pequeas.

1.2. Formatos de intercambio de informacin que no se basan en metalenguajes

Los problemas de legibilidad antes mencionados no son especficos de los formatos basados en metalenguajes. Las grandes capacidades de procesamiento automatizado que nos otorgan las computadoras nos permiten tratar problemas mucho ms grandes, pero con cada vez mayor frecuencia nos vemos impedidos a la hora de analizar los propios resultados.

Tambin aqu, la solucin usual para formatos lo bastante populares es crear visores especficos para ellos. En muchos casos, el documento original se convierte primero a un formato basado en un metalenguaje, para reutilizar la amplia gama de documentacin, herramientas y bibliotecas existentes. Esto no debera suponer una prdida de informacin: XML, por ejemplo, permite construir lenguajes arbitrariamente complejos. Otras veces, es la propia aplicacin la que pasa a basar el formato de su salida en un metalenguaje, como en el caso del sistema de demostraciones automatizadas Mizar [?, ?], que pas a utilizar XML.

Con la infraestructura bsica establecida, se puede pasar directamente a crear el visor. Si ya existiera un visor genrico basado en el metalenguaje usado, podramos evitar tambin este paso: slo habra que especializarlo para nuestro formato.

La reduccin de las barreras necesarias para crear un visor nos permitira concentrarnos ms en el anlisis del formato origen y en la definicin del formato final de presentacin, sin tener que molestarnos en los detalles de implementacin necesarios para mantener un historial de documentos, habilitar la navegacin por hipervnculos y otros aspectos de bajo nivel.

As, podramos manejar formatos considerados ms difciles de procesar, como aquellos que carecen de gramticas explcitamente definidas. Tal es el caso de muchos resultados generados de manera automtica y dirigidos al usuario final: en este caso, la gramtica est definida por el propio cdigo que la produce. Convertir la salida producida a un metalenguaje puede resultar ms eficiente que reescribir el programa original si no se ha hecho una separacin entre el procesamiento y la presentacin del resultado desde un primer momento.

Uno de estos formatos con gramticas implcitas es el de las demostraciones producidas por el sistema ACL2 (A Computational Logic for Applicative Common Lisp). ACL2 es un sistema de razonamiento automatizado desarrollado en la Universidad de Texas en Austin y publicado bajo la licencia GNU GPL. Utiliza un subconjunto funcional (*aplicativo* o sin efectos colaterales, es decir, sin bucles, ni asignaciones, ni variables globales,

1. Introduccin

etc.) de Lisp para realizar demostraciones formales de propiedades de sistemas software y hardware complejos.

En [?] se muestran algunos ejemplos de demostraciones sobre sistemas complejos realizadas mediante ACL2. Entre ellos, se menciona la verificacin de que cierto microcdigo de un chip DSP (Digital Signal Processing) de Motorola realmente implementaba ciertos algoritmos conocidos, y que lleg a los 84 megabytes de texto puro. No es slo eso: la salida de ACL2 se realiza en una mezcla de lenguaje natural y expresiones Lisp que puede hacerse muy difcil y tediosa de manejar en demostraciones de inters prctico.

Puede darse tambin el caso de que la demostracin se halle dividida a lo largo de varios ficheros, y la traza de una demostracin fallida requiera as la inspeccin de varios ficheros enlazados entre s.

Definiendo un visor para ACL2, se podra navegar de forma mucho ms sencilla y cmoda, y partiendo de un marco genrico, podemos ahorrarnos la mayor parte del trabajo implicado en ello. Adems, seguir los sutiles cambios producidos en el formato de la salida de ACL2 entre versiones se hara mucho ms sencillo, al no estar atado el visor a esos detalles.

1.3. Objetivos

Este proyecto toma como punto de partida el Proyecto Fin de Carrera “Post-procesador y Visor de Demostraciones del Sistema ACL2”, en el cual se defini una conversin preliminar de un subconjunto de las demostraciones sin uso de libros de ACL2 a un lenguaje basado en XML, y un visor especfico para este formato.

Los objetivos a cumplir son los siguientes:

- Generalizar el diseo del visor a un visor genrico de documentos basado en XML para su uso en un amplio rango de formatos, tanto basados como no basados en metalenguajes. Se necesitan:
 - Integracin personalizable con los editores y los convertidores a XML de los mltiples formatos utilizados.
 - Apertura de mltiples documentos en una interfaz en pestaas o TDI (Tabbed Document Interface), con la posibilidad de mantener distintas configuraciones de visualizacin en cada documento abierto.
 - Recorrido de enlaces entre mltiples documentos, pudiendo seleccionar el nodo exacto a visualizar en el documento enlazado.
 - Retirada de toda lgica especfica a ACL2 del visor, movindola al antiguo post-procesador.

- Mejorar la calidad global del producto a travs de:
 - Mejorar la cantidad y calidad de la documentacin, situandola en una plataforma colaborativa.
 - Facilitar la instalacin del visor y sus extensiones.
 - Traducir la interfaz del visor al ingls.
 - Publicar el cdigo fuente y otros ficheros asociados a travs de una forja.
 - Ampliar el conjunto de casos de prueba de todos los productos.
- Refinar el antiguo post-procesador de ACL2 y sus hojas de estilo, ejecutando las siguientes acciones:
 - Integracin de la lgica especfica a ACL2 antes contenida en el visor.
 - Manejo de demostraciones que certifiquen libros y reutilicen sus definiciones a mltiples niveles, teniendo en cuenta el grafo completo de dependencias y evitando enviar a ACL2 el mismo fichero varias veces.
- Crear una nueva extensin para procesar documentos basados en YAML y JSON, con sus hojas de estilo:
 - Conversin de documentos YAML 1.1 y JSON a XML, garantizando que no se producen prdidas ni cambios de la informacin original.
 - Manejo de anclas y alias de YAML, con la debida creacin de vnculos en el XML resultante.

1.4. Alcance

El proyecto se compone de tres productos:

- El producto principal es XMLEye, un visor genrico de documentos basados en XML, integrable con extensiones externas que convierten formatos no basados en XML y con hojas de estilo XSLT (eXtensible Stylesheet Language Transformations) que definen visualizaciones especializadas para ciertos formatos.
- `ACL2::Procesador`, un conversor integrable en XMLEye que toma la salida de ACL2 en lenguaje humano y su fuente Lisp y procesa ambos para obtener un fichero XML sin prdida de informacin que otros programas puedan manejar fcilmente. Incluye sus propias hojas de estilo especializadas, tambin integrables en XMLEye.
- `YAXML::Reverse`, un conversor a XML de documentos YAML 1.1 y JSON. Tambin incluye sus propias hojas de estilo.

1. Introduccin

Estos productos dispondrn de su propia documentacin para usuarios y desarrolladores. Se habilitar un *wiki* y un espacio en una forja para permitir la futura colaboracin de nuevos participantes.

1.4.1. Limitaciones del proyecto

nicamente se utilizarn visualizaciones en HTML mediante una interfaz Swing. Generar rboles estticos de ficheros XHTML (eXtensible Hyper Text Markup Language) o utilizar interfaces Swing dinmicas mediante JavaFX se dejan como trabajo futuro.

Por limitaciones de tiempo, `ACL2::Procesador` se restringe a un subconjunto de todos los eventos de ACL2, tratando demostraciones que, aunque complejas, no hacen uso de todo el potencial de ACL2. Su uso ser, pues, eminentemente didctico.

Se ignoran en la actualidad los mensajes de la implementacin de Lisp sobre la que se ejecuta ACL2, como los mensajes del recolector de basura, ya que existe actualmente una gran variedad de ellas y no ser factible ni til aadir lgica para cada una. Por supuesto, s se han considerado los mensajes de error durante la demostracin propios de ACL2.

No se crearn hojas de estilo para lenguajes derivados de YAML o JSON para este proyecto. En futuras versiones se crearn hojas de estilo para ficheros de marcadores de Firefox y volcados de bases de datos de pruebas de Django.

1.4.2. Licencia

Se decidi publicar `ACL2::Procesador`, `YAXML::Reverse` y `XMLEye` como software libre bajo la licencia GPL (General Public License) en sus versiones 2 o posteriores. Las licencias de las principales bibliotecas, mdulos Perl y dems herramientas utilizadas en este Proyecto son:

- Perl: la versin original dirigida a sistemas UNIX puede usarse bajo o bien la licencia denominada “Artstica” (disponible bajo <http://www.perl.com/language/misc/Artistic.html>), o bajo la GNU (GNU is Not Unix) GPL en sus versiones 1 o posteriores.
- La distribucin de Perl para Windows usada, Strawberry Perl, y la mayora de los mdulos Perl usados comparten el esquema de licenciado de Perl. Algunos mdulos varan ligeramente, como en el caso de `XML::Writer`, que no impone restriccin alguna en su uso comercial o no comercial, o `XML::Validate`, que nicamente admite la licencia GPL.

- A travs de algunos de los mdulos Perl, se emplean de forma indirecta las bibliotecas `libxml2`, `libxslt` y `libyaml`. Todas se hallan bajo la licencia MIT (Massachusetts Institute of Technology), prcticamente equivalente a la licencia BSD (Berkeley Software Distribution) revisada, en la que se retira la necesidad de incluir el aviso del uso de dicho componente en todo el material publicitario del producto final. Al no ser una licencia *copyleft*, los usuarios no se hallan obligados a devolver a la comunidad los cambios que hagan sobre las bibliotecas.
- OpenJDK 6.0: GPL, con la excepcin Classpath (vase <http://www.gnu.org/software/classpath/license.html>), que permite enlazar mdulos independientes bajo cualquier licencia con mdulos escritos bajo la GPL versin 2, sin que la GPL se extienda a dichos mdulos. De esta forma, puede usarse para desarrollar aplicaciones comerciales, por ejemplo, y no supone un impedimento para su adopcin. En ltima instancia, es muy similar a la licencia LGPL (Lesser General Public License).

Un pequeo porcentaje (menor al 5 %) de OpenJDK sigue estando bajo licencias propietarias: Sun est ahora trabajando en sustituirlas por componentes equivalentes.

- IcedTea 6.0: GPL con la excepcin Classpath. Reemplaza ese pequeo porcentaje propietario de OpenJDK y aade un proceso de compilacin completamente basado en herramientas libres.
- Apache Ant: APL (Apache Public License) 2.0. Esta licencia tampoco es *copyleft*, y es compatible con la GPL en sus versiones 3 o superiores.
- ACL2: GNU GPL, combinando a su vez componentes bajo la GNU GPL (por ejemplo, la biblioteca `readline`) y bajo la GNU LGPL (como el entorno Lisp GCL (GNU Common Lisp)).

1.5. Visin general

Tras una revisin del calendario seguido, detallaremos a lo largo del resto de la memoria el proceso de anlisis, diseo, codificacin y pruebas que se sigui al realizar el proyecto.

En particular, se describir el estado del proyecto en su ltima iteracin, que tiene la arquitectura y el diseo definitivos.

Los manuales de usuario y de instalacin se incluyen tras un resumen de los aspectos ms destacables de proyecto y las conclusiones. En dicho manual, se halla un apartado dirigido a usuarios avanzados que deseen definir sus propias hojas XSLT.

1.6. Glosario

1.6.1. Acrnimos

ACL2 A Computational Logic for Applicative Common Lisp

AMD Advanced Micro Devices

API Application Programming Interface

APL Apache Public License

ASCII American Standard Code for Information Interchange

BNF Backus-Naur Form

BOM Byte Order Mark

BSD Berkeley Software Distribution

C3 Chrysler Comprehensive Compensation System

CASE Computer Assisted Software Engineering

CPAN Comprehensive Perl Archive Network

CPU Central Processing Unit

CSS Cascading Style Sheet(s)

CVS Concurrent Versions System

DOM Document Object Model

DSP Digital Signal Processing

DTD Document Type Definition

EAFP Easier to Ask for Forgiveness than Permission

FDL Free Documentation License

FSF Free Software Foundation

GCL GNU Common Lisp

GIMP GNU Image Manipulation Program

GML Generalized Markup Language

GNU GNU is Not Unix

GPL General Public License

GTK+ GIMP Toolkit

GUI Graphical User Interface

HTML Hyper Text Markup Language

IBM International Business Machines

IDE Integrated Development Environment

J2SE Java 2 Standard Edition

JAXP Java API for XML Parsing

JDBC Java DataBase Connectivity

JDK Java Development Kit

JIT Just In Time

JRE Java Runtime Environment

JSON JavaScript Object Notation

JVM Java Virtual Machine

LGPL Lesser General Public License

LTS Long Time Support

MIT Massachusetts Institute of Technology

MVC Model View Controller

MVP Model View Presenter

ODF Open Document Format

PAR Perl ARchive Toolkit

PDF Portable Document Format

POD Plain Old Documentation

POSIX Portable Operating System Interface, UniX based

Perl Practical Extraction and Report Language

SAX Simple API for XML

SGML Standard Generalized Markup Language

SVG Structured Vector Graphics

SWT Standard Widget Toolkit

TDI Tabbed Document Interface

UML Unified Modelling Language

1. Introduccin

URI Uniform Resource Identifier

URL Uniform Resource Locator

UTF Universal Transformation Format

W3C World Wide Web Consortium

WWW World Wide Web

XHTML eXtensible Hyper Text Markup Language

XML eXtensible Markup Language

XP eXtreme Programming

XSL-FO eXtensible Stylesheet Language Formatting Objects

XSLT eXtensible Stylesheet Language Transformations

YAML YAML Ain't a Markup Language

YAXML YAML XML binding

1.6.2. Definiciones

Definiciones generales

Analizador lxico Programa o mdulo que se ocupa de reconocer los smbolos terminales de un lenguaje dentro de las cadenas de texto recibidas por el analizador sintctico.

Analizador sintctico Programa o mdulo que se encarga de agrupar las cadenas de smbolos terminales proporcionados por el analizador lxico en smbolos no terminales. Pueden reflejar as restricciones de mayor nivel de abstraccin sobre las cadenas vlidas del lenguaje. Normalmente le sigue un analizador semntico.

Analizador semntico Programa o mdulo cuyo cometido es realizar cualquier procesamiento adicional especfico para el lenguaje del texto procesado, y que el analizador sintctico no sea capaz de implementar a travs de su gramtica. Excepto por los lenguajes ms simples, una gramtica estndar no puede representar todas sus restricciones.

Deserializacin Proceso inverso a la serializacin. En l se recupera una copia con el mismo contenido semntico que la estructura de datos original, a partir de la secuencia de bytes resultante de la serializacin.

Forja Sitio web dedicado a hospedar proyectos de desarrollo de software. Normalmente, las forjas son gratuitas para proyectos de software libre y/o código abierto, pero también existen forjas [?] para entornos comerciales. Los servicios proporcionados varían mucho entre forja y forja, pero como mínimo suelen tener espacio en algún sistema de control de versiones, un sistema de control de incidencias y un área en la que alojar ficheros para su descarga por los usuarios.

Lenguaje Subconjunto del conjunto potencia de los caracteres de un alfabeto. Un lenguaje puede definirse a partir de una gramática que especifique restricciones en la sintaxis de las cadenas consideradas válidas. En el caso de los lenguajes artificiales, esta gramática viene definida explícitamente a través de algún mecanismo formal.

Lenguaje de marcado Lenguaje artificial que decora un texto sencillo con anotaciones acerca de la estructura, el significado y/o el formato de sus partes. Estas anotaciones son de tipo declarativo, describiendo qué características tiene cada región del texto, y no cómo se consiguen.

Algunos ejemplos muy conocidos incluyen SGML, HTML, XML, \TeX o \LaTeX .

Lenguaje de serialización Lenguaje artificial que representa estructuras de datos a partir de una serie de símbolos terminales organizados según una serie de reglas. Idealmente, un lenguaje de serialización debería ser conciso, fácilmente legible y eficiente tanto al serializar como deserializar.

Metalinguaje En el ámbito en que nos ocupa, un metalinguaje consiste en una serie de reglas que proporcionan la sintaxis y semántica, dentro de un modelo de datos, como un árbol o un grafo, por ejemplo, de una familia completa de lenguajes. No llega a detallar el contenido exacto de los documentos, quedando por concretar en cada una de sus instancias. Estas que se hallan en un nivel más bajo de abstracción, y ya son lenguajes completamente definidos.

Serialización Proceso en el que se traduce una estructura de datos en memoria a una secuencia de bytes que representan su contenido. Esta puede posteriormente enviarse a través de una conexión de red, guardarse en un fichero, etc.

Símbolo no terminal Símbolo de una gramática que puede ser expandido a una sucesión de símbolos terminales y/o no terminales.

Para ilustrar mejor el concepto, supongamos que tenemos la gramática $G = (N, \Sigma, P, S)$ con los símbolos terminales $\Sigma = \{a, b\}$, los símbolos no terminales $N = \{E\}$, el símbolo inicial $S = E$, y el conjunto de reglas P de producción en BNF (Backus-Naur Form) siguiente:

$$E \rightarrow Eb$$

$$E \rightarrow a$$

1. Introduccin

Podramos hacer la siguiente expansin para obtener la secuencia de smbolos terminales *abb*:

$$\begin{aligned} E &\Rightarrow Eb \\ &\Rightarrow Ebb \\ &\Rightarrow abb \end{aligned}$$

Smbolo terminal Smbolo de una gramtica que representa una secuencia de caracteres de un alfabeto con un significado coherente. Algunos ejemplos incluyen “divisin” (con la cadena “/”, por ejemplo) o “entero” (“200”). El significado de un smbolo terminal puede ser el mismo para todas sus ocurrencias (como en “divisin”), o puede variar en cada instancia (como en el caso de “entero”).

Wiki Sitio web cuyas pginas pueden ser editadas a travs del propio navegador web, distribuyendo su mantenimiento a una comunidad completa de usuarios. Existe una gran variedad: desde *wikis* de empresa slo accesibles por una parte de los empleados, hasta *wikis* de acceso y modificacin pblicos, como la conocida Wikipedia. El primer wiki fue WikiWikiWeb [?].

Definiciones en el dominio de YAML, JSON y YAXML

Alias Identificador nico establecido sobre un nodo del documento, para posterior referencia mediante anclas.

Ancla Referencia a un alias definido en otra parte del documento. Evita tener que repetir contenido idntico en varias partes del documento en la salida serializada.

Documento Estructura de datos nativa al lenguaje de programacin utilizado, consistente en un nico *nodo* raz.

Se obtiene a partir del flujo tras los eventos de *anlisis*, la *composicin* del grafo a partir de estos, y la *construccin* de la estructura nativa de datos a partir del grafo.

En direccin inversa, una estructura de datos nativa se *representa* por un grafo dirigido y con raz de nodos, se convierte a un rbol ordenado, que se recorre para generar eventos de *serializacin* y finalmente *presentar* la estructura original en forma de una sucesin de bytes.

Flujo Sucesin de bytes que cumple las reglas correspondientes de produccin de la gramtica de YAML, y en la cual se pueden hallar presentados cero, uno o ms *documentos* YAML, separados por delimitadores. El primer delimitador es opcional.

Opcionalmente, puede comenzar por un BOM (Byte Order Mark) de UTF (Universal Transformation Format) indicando si la entrada se halla codificada en UTF-8, UTF-16 Little Endian, o UTF-16 Big Endian. Por defecto, se asume que la entrada se halla codificada en UTF-8.

Nodo Cualquier elemento con entidad propia del rbol del documento. Puede tener otros nodos anidados en su interior.

Escalar Categora de todo nodo con contenido atmico opaco consistente en una serie de cero o ms caracteres Unicode. El significado de dichos caracteres se concreta a partir de su etiqueta.

Esquema Combinacin de un conjunto de etiquetas y un mecanismo para etiquetar nodos por defecto. A travs de estos esquemas, como el JSON Schema o el Core Schema, podemos darles interpretaciones estndar a los nodos secuencia, nodos vector asociativo o nodos escalar.

Por ejemplo, conseguimos que los escalares con el contenido “true” se les d la etiqueta “tag:yaml.org,2002:bool”, indicando que se trata de un valor de verdad booleano.

Etiqueta Todo nodo tiene una etiqueta, que indica el tipo de informacin que representan. Por ejemplo, los enteros tienen la etiqueta “int”, y los nmeros de coma flotante “float”.

Secuencia Categora de los nodos que contienen cero o ms nodos, numerables segn su posicin relativa entre los dems.

Vector asociativo Categora de todo nodo que contiene un conjunto de cero o ms pares clave-valor, en donde las claves han de ser nicas. Tanto las claves como los valores pueden ser nodos arbitrariamente complejos y de cualquier tipo.

Definiciones en el dominio de ACL2

tomo Lisp Mnima unidad sintctica de toda expresin Lisp. Puede ser un smbolo, un nmero, un carcter o una cadena.

Clausura lambda Par formado por un bloque de cdigo a ejecutar y las ligaduras a variables usadas en dicho cdigo locales al bloque en que fue definida la clausura, que persisten tras la ejecucin de dicho bloque.

El concepto, creado por Alonzo Church en su λ -clculo, ha sido llevado a diversos lenguajes de programacin con mayor o menor exactitud: Scheme (el primer lenguaje en implementarlo completamente [?, ?, ?, ?]), Perl y sus subrutinas annimas, Java y sus clases annimas, etc.

Dependencia inversa Se dice de la relacin entre un evento y aquellos eventos que lo mencionan en su demostracin.

1. Introduccin

Evento Orden de ACL2 que modifica su mundo lgico en alguna medida.

Libro Conjunto de funciones y teoremas que expresan propiedades sobre ellas y que forman una teora computacional en ACL2.

Meta Unidad mnima de toda demostracin de ACL2, a la que le corresponde una etiqueta nica dentro de la demostracin, una conjetura y una serie de acciones que conducen a su demostracin, refutacin, simplificacin o divisin en otras metas.

Una meta puede contener submetas, generando un rbol de demostracin de manera recursiva. Por supuesto, existe una meta raz en toda demostracin de ACL2, que recibe la etiqueta “Goal”.

Mundo lgico A nivel abstracto, el *mundo lgico* o simplemente *mundo* de ACL2 es la base de datos en la que basa sus demostraciones, y que podemos extender con nuevas reglas, funciones y otros elementos.

A nivel de implementacin, un *mundo* no es ms que una lista propia de Lisp de tripletas propiedad-smbolo-valor. Podemos crear nuestros propios mundos si as lo deseamos y utilizarlos con las funciones `getprop` y `setprop`. El mundo de ACL2 se halla bajo el nombre `'current-acl2-world`, pero no puede ser modificado excepto por un evento.

Paquete Lisp Nombre de espacios que califica a un identificador, reduciendo el riesgo de una colisin. Por defecto, en ACL2 se usa el paquete ACL2, pero podemos definir nuestros propios paquetes y cambiar entre ellos libremente.

Par Lisp Estructura de datos ordenada que engloba a dos elementos.

El formato bsico es $(a . b)$, donde a o b pueden ser tomos o pares Lisp. As, para crear una lista con los elementos 1, 2 y 3, escribiremos $(1 . (2 . (3 . nil)))$, o en una notacin simplificada, $(1\ 2\ 3)$. Listas como sta donde el ltimo elemento es *nil* se conocen como *listas propias*.

Proyecto Grafo acclico dirigido con raz en un fichero Lisp que describe una demostracin a realizar mediante ACL2, utilizando una serie de eventos definidos en varios libros, ya se hallen integrados en la distribucin de ACL2 o hayan sido creados por el usuario.

Un proyecto depende de una serie de *subproyectos*, cuyas races son los nodos directamente adyacentes al nodo raz del proyecto principal.

S-expresin Su nombre es una abreviatura de “expresin simblica”, y consiste en un tomo Lisp o en un par Lisp. Los eventos de ACL2, por ejemplo, estn formados por S-expresiones, utilizando notacin prefija para la aplicacin de funciones y operadores.

1.6.3. Otras tecnologas

XSLT

XSLT (vase [?]) se trata de un lenguaje XML estandarizado por el W3C (World Wide Web Consortium) que define hojas de estilo capaces de transformar una entrada XML a otros formatos.

El formato de salida no ha de ser necesariamente XML tambien: se podra generar texto arbitrario, o un documento HTML.

El uso de hojas XSLT permite definir las transformaciones de forma declarativa, simplificando mucho la tarea de transformacin frente a otros mtodos, como el uso del API DOM (Document Object Model), o el procesado mediante SAX (Simple API for XML).

Permite que la visualizacin del documento XML en XMLEye se halle dirigida por datos y no por cdigo, y as sea extensible y personalizable por el usuario sin necesidad de recompilacin.

XPath

XPath (vase [?]) es un estndar relacionado con XML del W3C que adems de poderse usar independientemente, es parte fundamental de otras tecnologas XML, como XSLT, o XQuery, un lenguaje especializado para consultas sobre datos XML de cualquier fuente (incluso una base de datos).

Su propsito es permitir identificar fcilmente conjuntos de nodos de un documento XML, siguiendo un modelo de datos similar al del estndar DOM, con ligeras diferencias.

La sintaxis de una expresin XPath es declarativa, usando condiciones que deben cumplir los nodos resultado de dicha expresin. Se pueden filtrar nodos en funcin de su tipo, nombre (en caso de ser elemento), atributos, descendientes, etc.

En XMLEye se usa como parte de XSLT en las transformaciones y, por separado, en las bsquedas.

YAXML

YAXML (YAML XML binding) [?] es una correspondencia XML → YAML en borrador desde 2006. Se halla descrita a nivel informal de manera textual, y con mayor rigor a travs de una transformacin automtica implementada mediante una hoja de estilos XSLT 1.0.

1. Introduccin

`YAXML::Reverse` es un mdulo Perl que implementa la correspondencia inversa y al mismo tiempo sirve para depurar y mejorar YAXML, aprovechando el ciclo `YAML → XML → YAML` resultante.

2. Desarrollo del calendario

No se ha seguido una estricta planificacin durante la realizacin de este proyecto, dado que se ha desarrollado la mayor parte de l de forma simultnea a los estudios del ltimo curso del Segundo Ciclo en Ingeniera Informtica, durante mi participacin en el II Concurso Universitario de Software Libre [?].

Los contenidos de la forja [?] han sido revisados en paralelo con el proyecto desde la elaboracin de los paquetes Debian de cada producto.

2.1. XMLEye

2.1.1. Primera iteracin: rediseo

El diseo del visor original se hallaba fundamentado en la premisa de que slo era necesario visualizar un nico documento a la vez. Al cambiar esta premisa, se necesit revisar el diseo completo de la aplicacin. Se cambi de un diseo basado en fachadas a un diseo basado en el patrón arquitectónico Presentación-Abstracción-Control.

Las pruebas de unidad existentes fueron una gran ayuda a la hora de mantener la funcionalidad durante el cambio.

2.1.2. Segunda iteración: interfaz TDI

Con el nuevo diseo, se pudo integrar una interfaz multidocumento TDI o basada en pestaas, con controles para cerrar pestaas individuales y cerrar todas las pestaas. Se implement la capacidad de mantener el estado y opciones de visualización de cada documento de forma independiente.

2.1.3. Tercera iteracin: hipervnculos entre documentos

Se extendi la sintaxis existente para enlaces entre nodos de un documento para poder enlazar con nodos de otros documentos, aprovechando el soporte de demostraciones que usen libros por el lado de `ACL2::Procesador`. Al pulsar en uno de esos enlaces, se abrira o seleccionara la pestaa del documento en cuestin y se pasara al nodo especificado.

2.1.4. Cuarta iteracin: descriptores de formatos de documentos

Se retir la lgica restante especfica a ACL2 de XMLEye: en particular, se retiraron los dilogos explcitos de importacin. Se reemplaz por un sistema de descriptores de formatos de documentos que generalizaban el enfoque anterior a cualquier formato para el que hiciera falta un conversor y/o un editor, y que se pudiera identificar por su extensin. Las rdenes especificadas son personalizables por cada usuario del sistema a partir de unos valores globales por defecto.

2.1.5. Quinta iteracin: estabilizacin

Debido al importante nmero de cambios realizados en el diseo de la capa de aplicacin, se consider oportuno dedicar un tiempo prudencial a la depuracin en detalle y simplificacin del diseo en ciertos puntos. Un punto en el que se hizo mucho hincapi fue en el manejo de la concurrencia en XMLEye, que produca ocasionalmente problemas difciles de reproducir.

2.1.6. Sexta iteracin: paquete Debian

En esta iteracin se cre un paquete Debian para XMLEye. Se hicieron diversos ajustes en XMLEye debidos a la necesidad de acomodarlo a las prticas recomendadas por la poltica de Debian, y se aprovech para verificar el correcto funcionamiento de XMLEye con OpenJDK, la versin de cdigo abierto del JDK (Java Development Kit). Ahora XMLEye puede funcionar al 100 % slo con software libre. Tambin se aadi la posibilidad de abrir documentos con XMLEye desde la lnea de rdenes.

2.1.7. Sptima iteracin: manuales y traduccin al ingls

Se escribieron sendos manuales de usuario y desarrollador en DocBook, convirtiendose a los formatos HTML de una o varias pginas y PDF (Portable Document Format). La versin PDF fue producida a travs de unas hojas XSLT especializadas que producen

cdigo \LaTeX , puesto que todas las hojas equivalentes existentes daban resultados insatisfactorios.

Adems, se tradujeron al ingls todas las cadenas de la interfaz de XMLEye y los nombres de las clases de su cdigo. La arquitectura ya se hallaba internacionalizada, por lo que no hubo que hacer ms cambios aparte de indicar el idioma ingls como lenguaje por defecto.

2.1.8. Octava iteracin: creacin de un wiki

Se instal, configur y elabor una primera versin de los contenidos de un wiki [?]. Detalla tanto las motivaciones detrs de XMLEye, como su arquitectura general, los mtodos de instalacin y qu tener en cuenta al aadir nuevos formatos de entrada y hojas de estilo a XMLEye.

2.2. ACL2::Procesador

2.2.1. Primera iteracin: rediseo

Se revis la organizacin del post-procesador para que siguiera las mejores prcticas reconocidas del CPAN (Comprehensive Perl Archive Network). Entre otras cosas, se aadi documentacin al propio cdigo en formato POD (Plain Old Documentation), se convirtieron las pruebas de regresin existentes a guiones Perl, y se aadieron ms pruebas para asegurar la cobertura de la documentacin.

Una importante ventaja de reescribir las pruebas de regresin usando mejores herramientas es la capacidad de calcular las diferencias entre la salida esperada y la obtenida de forma ms inteligente, pudiendo ignorar de forma selectiva diferencias que no se consideren importantes a nivel semntico.

2.2.2. Segunda iteracin: eventos para libros

Se aadi soporte para algunos eventos relacionados con el manejo de libros: IN-PACKAGE, DEFPKG, INCLUDE-BOOK y CERTIFY-BOOK.

2.2.3. Tercera iteracin: grafos de dependencias

Se separaron de forma explcita los pasos de anlisis del cdigo fuente de las demostraciones, anlisis de la salida resultante y produccin de la salida. Esto posibilit la generacin previa al anlisis de la salida de una demostracin de su grafo de dependencias. Este grafo accltico dirigido lista todos los eventos obtenidos de cada libro en uso, y detalla la ruta relativa a la definicin del libro.

Empleando estos grafos, se pudieron refinar las hojas de estilos para que crearan los enlaces a las definiciones de aquellos elementos que se originaran en otros libros.

2.2.4. Cuarta iteracin: integracin con ACL2

Aprovechando la informacin de los grafos de dependencias, se aadi la lgica necesaria a `ACL2::Procesador` para invocar l mismo a ACL2 sobre los ficheros implicados en una demostracin, haciendo un recorrido en post-orden del grafo. Los resultados del anlisis son almacenados en ficheros, y slo son recalculados cuando es necesario.

2.2.5. Quinta iteracin: ejecutable monoltico y paquete Debian

Se integr la posibilidad de crear ejecutables monolticos con todas las bibliotecas y mdulos necesarios y el propio intrprete de Perl mediante el mdulo PAR, y se elabor un paquete Debian, empaquetando adems todas sus dependencias, y revisando el cdigo de `ACL2::Procesador` para que cumpliera la poltica de Debian.

2.2.6. Sexta iteracin: traduccin al ingls

Se tradujeron todos los mensajes de error y los POD ms importantes a ingls. La traduccin del cdigo se deja para posteriores versiones.

2.2.7. Sptima iteracin: actualizacin para ACL2 v3.3

Se revis `ACL2::Procesador` para tener en cuenta los ligeros cambios producidos en la salida de ACL2 en su versin 3.3.

2.3. YAXML::Reverse

2.3.1. Primera iteracin: Procesamiento de YAML 1.0

Se realiz una primera versin consistente en un solo guin Perl que cargaba el fichero YAML 1.0 fuente y hacia la correspondencia inversa a YAXML. Se implement la conversin del ejemplo de la factura de [?], y de los 5 ejemplos de [?].

2.3.2. Segunda iteracin: reorganizacin y traduccin

Pronto se vio que el problema era ms complejo de lo esperado, y se reestructur el guin en forma de un mdulo Perl, igual que se haba hecho con `ACL2::Procesador`. El cdigo y toda la documentacin se tradujo directamente a ingls.

2.3.3. Tercera iteracin: ejecutable monoltico y paquete Debian

De forma similar a `ACL2::Procesador`, se cre un paquete Debian siguiendo las directrices de la poltica de Debian, y se posibilit la generacin de ejecutables monolticos que incluyeran todos los mdulos, bibliotecas y el intrprete Perl necesario.

2.3.4. Cuarta iteracin: YAML 1.1/JSON y otras mejoras

La versin hasta el momento utilizaba `YAML::Syck`, que funcionaba correctamente para ficheros YAML 1.0. Sin embargo, cuando hubo que utilizar ficheros JSON, no funcionaba debidamente: aparentemente, JSON es subconjunto de YAML 1.1, no de la versin 1.0.

Se cambi a `YAML::XS`, y se corrigieron otras cuestiones relativas al procesamiento de ficheros de entrada codificados en UTF-8.

2.4. Diagrama Gantt

Se ha elaborado un diagrama Gantt (figura 2.1 en la pgina 39) para poder visualizar con mayor facilidad la distribucin de las iteraciones.

Se pueden ver las fechas exactas de inicio y fin de cada producto y cada iteracin de cada producto y sus duraciones D_R y D_I en das reales y das ideales, respectivamente, en el cuadro 2.1 de la pgina 40. Tomado de [?], un da ideal equivale a la cantidad

2. Desarrollo del calendario

de trabajo que puede realizarse en un día por una persona, sin distracción alguna y a máximo rendimiento.

Se ha empleado la herramienta *Gantt Project* para dibujar el diagrama y GNOME Planner para calcular las fechas con mejor precisión. Ambos son código abierto: la primera está hecha en Java y disponible en <http://ganttproject.sourceforge.net>, y la segunda se halla escrita en C++ con la biblioteca GTK+ (GIMP Toolkit) y se puede encontrar bajo la dirección <http://live.gnome.org/Planner>.

2.5. Porcentajes de esfuerzo

Los porcentajes aproximados de esfuerzo se detallan en los cuadros 2.2 al 2.4 (páginas 40 a 41).

La mayoría de los requisitos eran conocidos de antemano desde el final del Proyecto Fin de Carrera sobre el que se basa este, por lo que hay muy poca énfasis en la fase de análisis. El importante rediseño de XMLEye ha motivado que los porcentajes de esfuerzo dedicados a la prueba de XMLEye y a su propia construcción sean prácticamente equivalentes.

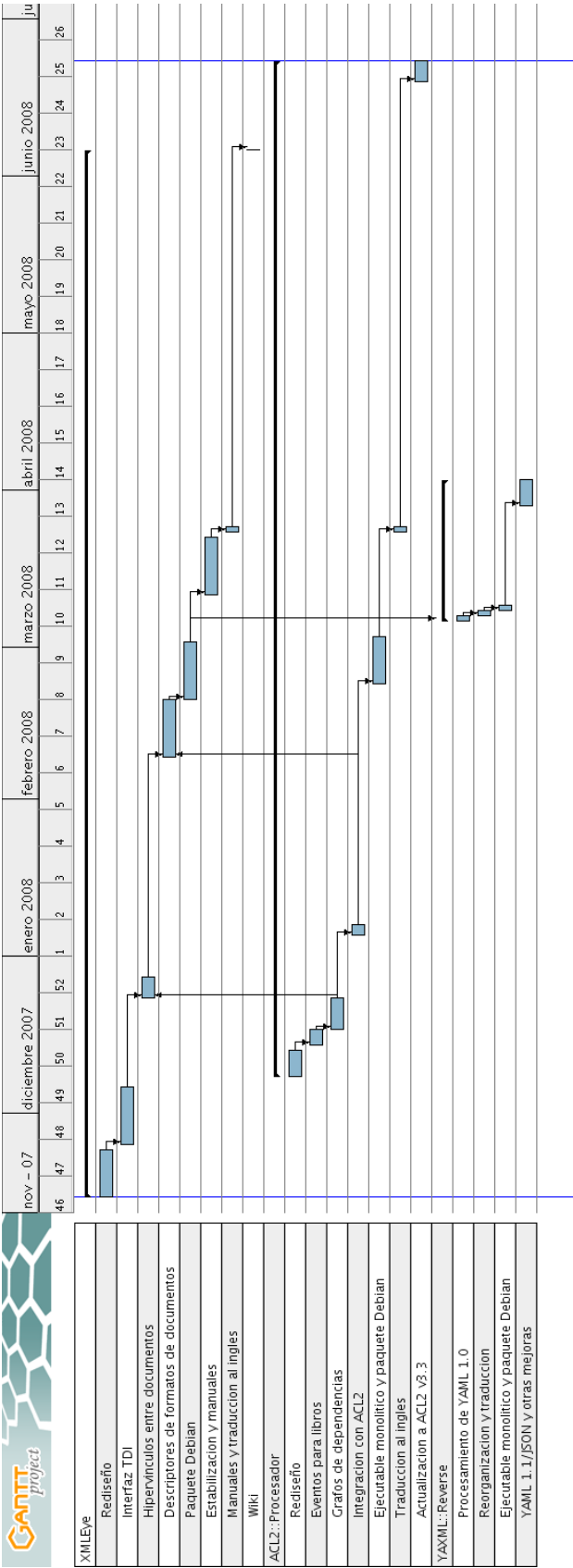


Figura 2.1.: Diagrama Gantt de iteraciones

2. Desarrollo del calendario

Producto	Inicio	Fin	D_I	D_R
<i>XMLEye</i>	15/11/2007	02/06/2008	34	53
Rediseo	15/11/2007	24/11/2007	6	8
Interfaz TDI	25/11/2007	06/12/2007	4	8
Hipervnculos entre documentos	27/12/2007	31/12/2007	2	5
Descriptores de formatos de documentos	07/02/2008	18/02/2008	6	10
Paquete Debian	21/02/2008	03/03/2008	5	10
Estabilizacin y manuales	09/03/2008	20/03/2008	8	8
Manuales y traduccin al ingls	21/03/2008	22/03/2008	1	2
Wiki	01/06/2008	02/06/2008	2	2
<i>ACL2::Procesador</i>	08/12/2007	19/06/2008	16	27,5
Rediseo	08/12/2007	13/12/2007	2	4
Eventos para libros	14/12/2007	17/12/2007	2	3,25
Grafos de dependencias	17/12/2007	23/12/2007	3	4,25
Integracin con ACL2	04/01/2008	06/01/2008	1	2
Ejecutable monoltico y paquete Debian	21/02/2008	01/03/2008	4	8
Traduccin al ingls	21/03/2008	22/03/2008	1	2
Actualizacin a ACL2 v3.3	14/06/2008	19/06/2008	3	4
<i>YAXML::Reverse</i>	06/03/2008	02/06/2008	5	6
Procesamiento de YAML 1.0	06/03/2008	06/03/2008	1	1
Reorganizacin y traduccin	07/03/2008	07/03/2008	1	1
Ejecutable monoltico y paquete Debian	08/03/2008	08/03/2008	1	1
YAML 1.1/JSON y otras mejoras	30/03/2008	04/04/2008	2	3

Cuadro 2.1.: Cuadro de fechas y duraciones de las iteraciones

Fase del proceso	Das ideales	Porcentaje
Anlisis	1,5	4,44 %
Diseo	6,25	18,38 %
Construccin	13,25	38,97 %
Pruebas	13	38,24 %
<i>Total</i>	34	100 %

Cuadro 2.2.: Porcentajes de esfuerzo: XMLEye

2.5. Porcentajes de esfuerzo

Fase del proceso	Das ideales	Porcentaje
Anlisis	0,5	3,125 %
Diseo	3	18,75 %
Construccin	7	43,75 %
Pruebas	5,5	34,375 %
<i>Total</i>	16	100 %

Cuadro 2.3.: Porcentajes de esfuerzo: ACL2::Procesador

Fase del proceso	Das ideales	Porcentaje
Anlisis	0,25	5 %
Diseo	0,5	10 %
Construccin	2,75	55 %
Pruebas	1,5	30 %
<i>Total</i>	5	100 %

Cuadro 2.4.: Porcentajes de esfuerzo: YAXML::Reverse

3. Descripci3n general del proyecto

3.1. Perspectiva del producto

3.1.1. Entorno de los productos

El proyecto est formado por XMLEye, ACL2::Procesador y YAXML::Reverse. Los dos primeros se basan sobre las versiones finales del visor y el post-procesador presentados en el Proyecto Fin de Carrera “Post-procesador y Visor de Demostraciones del Sistema ACL2”, y el tercero se halla escrito desde cero. Los tres productos tienen entidad propia, pudiendo usarse integrados a travs de una sencilla interfaz o por separado.

XMLEye, YAXML::Reverse y ACL2::Procesador se hallan en continuo desarrollo, y se seguirn mejorando ms all de este Proyecto. En el caso de YAXML::Reverse no se aadir nueva funcionalidad, sino que se seguir depurando su correcci3n ante ejemplos ms complejos y nuevas versiones de la especificaci3n de YAML.

ACL2::Procesador slo trata un subconjunto, aunque ampliado respecto a su versi3n original, de las demostraciones del sistema ACL2.

3.1.2. Interfaces software y hardware

La interfaz entre ACL2::Procesador y cualquier sistema que haga uso de su salida (por ejemplo XMLEye) est bien definida a travs de su POD, que indica la forma de invocaci3n correcta, y de su DTD (Document Type Definition), que permite conocer el formato de la salida sin tener que examinar el programa.

No existe una DTD explcita para YAXML::Reverse, puesto que el formato de la entrada resultante depende en gran medida del documento proporcionado: no olvidemos que YAML, el formato convertido, es un metalenguaje, al igual que XML, el formato destino.

Se ha de procurar tambi3n reducir el acoplamiento de los conversores y XMLEye con el sistema operativo y hardware al mnimo, para garantizar transportabilidad a lo largo de distintas arquitecturas y sistemas operativos.

3. Descripción general del proyecto

3.1.3. Interfaz de usuario

En cuanto a la interfaz de usuario, se ha procurado hacerla lo más sencilla y flexible posible, con una simple vista que permita relacionar la estructura en forma de árbol de la demostración con información acerca del elemento actualmente seleccionado.

La integración con el resto del entorno del usuario y la adaptabilidad a sus necesidades mediante la configuración de su comportamiento permitirán al usuario sacarle el mayor provecho posible a esta aplicación.

Todas las funciones deben estar disponibles por teclado y ratón, por razones de accesibilidad y usabilidad.

3.2. Funciones

3.2.1. XMLEye

- Navegación jerárquica simultánea por múltiples documentos a distintos niveles de detalle, con vínculos entre sus elementos y con elementos de otros documentos.
- Búsqueda a lo largo del documento, con diversas opciones de filtrado.
- Integración con nuevos formatos especificados por el usuario, sin necesidad de modificar el código. Cada formato puede especificar el editor y (opcionalmente) el conversor a XML con el que deberá integrarse XMLEye.

Los formatos son localizables a distintos idiomas, y personalizables por cada usuario del sistema. Se deben de poder restaurar los ajustes originales del formato en cualquier momento.

- Cambio en tiempo de ejecución del comportamiento de visualización del documento y sus nodos, que debe ser extensible por el usuario sin necesidad de añadir código específico a XMLEye.
- Monitorización en segundo plano del documento abierto, volviéndolo a abrir cuando se produzcan cambios en él. Esto posibilita su edición y visualización en paralelo.

3.2.2. ACL2::Procesador

- Conversión automática de un subconjunto de inters didáctico de la salida de ACL2 a XML, incluyendo demostraciones divididas entre múltiples ficheros.
- Generación de grafos de dependencias de una demostración con todos los libros usados, a cualquier número de niveles de profundidad.

- Invocación automática de ACL2 que tenga en cuenta las dependencias entre los distintos ficheros implicados y repita las demostraciones sólo ante cambios en las fuentes originales.
- Facilidad de mantenimiento y de actualización ante cambios en la salida entre versiones distintas de ACL2.
- Diversas hojas de estilo que produzcan información agregada a múltiples niveles en formato de hipertexto, manteniendo las dependencias directas e inversas entre todos los elementos, incluyendo a aquellos que pertenezcan a otros documentos.

3.2.3. YAXML::Reverse

- Conversión automática sin pérdidas ni cambios en la información de YAML a XML.
- Transformación de anclas y alias YAML a enlaces basados en identificadores únicos y atributos específicos.
- Hojas de estilo para XMLEye que tengan en cuenta los aspectos específicos de la salida producida por la conversión anterior.

3.3. Características del usuario

Podremos establecer varios tipos de uso para este proyecto:

XMLEye integrado con YAXML::Reverse Esta combinación permitirá a XMLEye abrir cualquier documento escrito en un lenguaje basado en el metalenguaje YAML. Esto incluye marcadores de Firefox 3, volcados de bases de datos del entorno de desarrollo web Django, o descriptores de módulos Perl, por ejemplo.

Tiene por lo tanto un amplio abanico de posibilidades dirigido a un gran número de usuarios, que se dividirán entre los *usuarios finales* que emplearán las hojas ya existentes, y que no requerirán más conocimientos que los necesarios para instalar la aplicación y YAXML::Reverse, y los *usuarios avanzados*, que podrán personalizar sus visualizaciones a su antojo.

Esta funcionalidad cobrará mayor utilidad cuando se le añada a XMLEye la capacidad de publicar la visualización en efecto como una página WWW. Se podrá entonces mostrar en una web de forma muy sencilla nuestros marcadores de Firefox, por ejemplo, y con más información que la exportación por defecto a HTML provee.

3. Descripción general del proyecto

XMLEye integrado con ACL2 : : Procesador El usuario slo emplea el post-procesador mediante su integracin con el visor para navegar por demostraciones de ACL2. Dentro de este modelo de uso, distinguimos dos tipos de usuarios:

Estudiante Un estudiante de ciclo superior estudiando mtodos formales de desarrollo de software usara esta herramienta para ayudarle a visualizar el proceso de demostracin de ACL2 sobre demostraciones sencillas.

Existen otras herramientas para ACL2 realizadas por motivos pedaggicos para este tipo de usuarios, pero se centran en aspectos distintos.

Un ejemplo es DrACuLa, basado en el entorno de Scheme (un dialecto de Lisp) llamado DrScheme. Entre algunos de los lenguajes disponibles, se puede hallar un dialecto simplificado de ACL2 extendido con unas librerías de manipulacin de grficos llamadas “teachpacks” para agilizar la enseanza, ilustrando el hecho de que los mtodos formales se pueden aplicar a proyectos software normales, y no slo a aquellos con un alto contenido terico.

Investigador Un investigador especializado en ACL2 preferira utilizar una herramienta que estructurara de alguna forma la demostracin en texto plano obtenida.

Podra adems aprovechar la informacin adicional como las dependencias inversas y otros enlaces y estadísticas para realizar su trabajo de forma ms eficiente.

Cuando se aadiera la posibilidad de publicar una demostracin como una pgina WWW, podra aadir a su pgina personal enlaces a algunas de sus demostraciones.

Ambos tipos de usuario tienen un alto nivel de experiencia en el uso de ordenadores, por lo cual se espera que no tengan problemas a la hora de configurar su entorno, si bien no se asumen conocimientos especficos de las herramientas usadas.

Notemos que, en la actualidad, ACL2 : : Procesador slo trata un subconjunto limitado de la entrada, y as hemos de limitarnos al uso puramente didctico, es decir, al usuario estudiante.

Sin embargo, la intencin de este proyecto es extender su soporte hasta el punto que sea til tambin para un investigador. El tratamiento de demostraciones divididas a lo largo de varios ficheros implementado en este Proyecto es un primer paso en esa direccin.

Slo XMLEye XMLEye no contiene lgica especfica para ningn formato en su cdigo Java para la visualizacin. Toda se halla en las hojas XSLT y los descriptores de formatos de documentos.

Aadiendo nuevas hojas, el usuario podra realizar sus propias visualizaciones de cualquier documento XML, pudiendo establecer enlaces entre los nodos o cambiar el icono o etiqueta mostrado en el rbol del documento, entre otras cosas.

Si el formato a tratar ya se basa en XML, elaborar un descriptor de formato de documento es completamente opcional: slo hara falta si quisiramos utilizar una extensin o un editor distintos a los usados para XML en general.

Slo ACL2 : :Procesador El usuario no utiliza XMLEye, y se limita a procesar de alguna forma la salida de ACL2 : :Procesador. Este tipo de usuario sera, realmente, un desarrollador que tendra cierta experiencia con ACL2 y deseara darle otra aplicacin a la salida. Los autores de Mizar tenan algo similar en mente cuando cambiaron el formato de su salida a XML [?].

Un ejemplo podra ser usar una hoja XSLT para generar una versin HTML de la demostracin, que se pudiera publicar en un servidor WWW, por ejemplo.

Tambin se podra generar un documento XSL-FO (eXtensible Stylesheet Language Formatting Objects) y, usando un procesador XSL-FO, obtener una salida en formato PDF o PostScript, entre otros.

Slo YAXML : :Reverse Aunque estn empezando a surgir validadores para YAML, como Kwalify (disponible en <http://www.kuwata-lab.com/kwalify/>), y en principio se podra transformar un documento YAML a partir de un algoritmo *ad hoc* sobre los datos deserializados, hay muchas menos herramientas que para XML.

Combinando YAXML : :Reverse y la versin refinada de YAXML que incluye, un desarrollador podra convertir el documento YAML a XML, aplicarle la herramienta XML en cuestin, y luego devolver el resultado XML a YAML.

La batera de pruebas de YAXML : :Reverse se ocupar de asegurar que en todo el ciclo YAML → XML → YAML no se produzcan prdidas ni cambios indeseados en la informacin del documento original.

3.4. Restricciones generales

3.4.1. Control de versiones

Al seguir un proceso incremental de desarrollo de software, se necesit un sistema de control de versiones de todas las fuentes del proyecto.

Estos sistemas permiten almacenar todas las versiones de un rbol de ficheros, pudiendo as manipular todas las revisiones de cualquier fichero en cualquier momento.

3. Descripción general del proyecto

Además de servir como una medida de seguridad contra la pérdida de información accidental, agilizan los cambios drásticos, ya que no hay que establecer medidas especiales por si fallaran: se pueden revertir los cambios hechos en cualquier momento.

En particular, `Subversion` es un sistema que trata de resolver las insuficiencias del conocido CVS (Concurrent Versions System), pudiendo mantener revisiones de directorios completos, establecer propiedades especiales sobre los elementos del repositorio y enviar nuevas revisiones de forma atómica, entre otras cosas.

Dispone de excelente documentación, con un libro [?] disponible bajo la licencia Creative Commons.

3.4.2. Lenguajes de programación

Con vistas a la transportabilidad, se eligieron dos lenguajes de programación con implementaciones interpretadas para el proyecto: Perl y Java.

Ambos lenguajes reducen significativamente el potencial de error gracias a su uso de recolectores de basura: Perl utiliza un recolector basado en recuento de referencias y la JVM (Java Virtual Machine) de Sun usa un algoritmo de barrido y marcado más avanzado.

Perl (Practical Extraction and Report Language) Este lenguaje es un candidato ideal para la labor de `ACL2::Procesador`, teniendo la implementación con mayor funcionalidad de expresiones regulares integrada dentro del propio lenguaje. En el caso de `YAXML::Reverse`, cualquier otro lenguaje con un sistema de tipos dinámico (como Ruby o Python) habría servido, pero por familiaridad con el entorno y por simplificar la instalación de ambos convertidores, se optó por el mismo lenguaje.

El rendimiento de Perl para el procesamiento automático avanzado de textos es alto, y se trata de una herramienta muy estable: la primera revisión de la versión de uso generalizado actual, la 5.8, se remonta al 2002. La versión estable más reciente actualmente, la 5.10, fue publicada en diciembre de 2007.

Es interpretado en el sentido que el código fuente de un guión Perl es compilado a instrucciones a ser interpretadas por una máquina virtual antes de cada ejecución.

Otra ventaja es la disponibilidad en forma de módulos de todo tipo de extensiones a las capacidades básicas de Perl en el CPAN.

Java Otro requisito era poder crear una interfaz gráfica de cierta complejidad con la mayor transportabilidad posible.

A pesar de la existencia de otros marcos de desarrollo de GUI basados en lenguajes con implementaciones interpretadas como PerlGTK, PyGTK o wxPython, se

deseaba adem's tener una plataforma integrada con soporte para XSLT sin necesidad de dependencias externas, y la capacidad de mantener un aspecto atractivo y uniforme entre plataformas.

Adem's, el API usado (Swing) es mucho m's maduro y estable que el de cualquiera de las tres alternativas anteriores. Otra opcin habra sido usar el SWT (Standard Widget Toolkit), alternativa de IBM que utiliza componentes nativos, pero su soporte de sistemas operativos distintos a Windows es inferior al de Swing.

El problema de la velocidad no es tan grave como se podra esperar, pues desde la versin 1.3 del JRE (Java Runtime Environment) la JVM o mquina virtual de Java incluye un compilador en tiempo de ejecucin o JIT (Just In Time) llamado HotSpot, que convierte a cdigo nativo y optimiza agresivamente las partes m's usadas del programa de forma dinmica.

3.4.3. Sistemas operativos y hardware

Aunque tanto Perl como Java estn disponibles para una gran cantidad de plataformas, para simplificar, este proyecto ha sido desarrollado y probado nicamente en Windows XP (con el Service Pack 2) y en GNU/Linux, ejecutando la versin 8.04 (Hardy Heron Long Time Support) de la distribucin Ubuntu, basada en Debian, y la versin 10.3 de la distribucin openSUSE.

XMLEye puede funcionar sobre cualquier entorno Java que implemente la J2SE (Java 2 Standard Edition) 5.0 como mnimo. Esto incluye evidentemente a los JRE 5.0 y 6.0 de Sun y recientemente a las versiones 6.0 y 7.0 de OpenJDK [?]. OpenJDK es una iniciativa liderada por Sun que, en combinacin con los esfuerzos del proyecto IcedTea [?], ha conseguido una versin prcticamente 100 % funcional y basada en software libre de las J2SE 6.0 y la futura 7.0.

Como mnimo, existen ediciones de la J2SE 5.0 o superiores para Solaris, MacOS X 10.4, Windows 32-bits y 64-bits y Linux 32-bits y 64-bits. El soporte de 64-bits depende de la CPU usada: no se pueden usar procesadores Intel Itanium, pero s todos los dem's chips de 64 bits de AMD (Advanced Micro Devices) e Intel.

Por el lado de Perl, la transportabilidad es mucho mejor:

- Apple Mac OS Classic 8.1 en adelante.
- Windows 95/98/ME/NT/2000/XP.
- Cualquier sistema basado en UNIX (los *BSD, Linux, Solaris y Mac OS X, entre otros).
- Otros sistemas menos comunes, como: TiVo, HP-UX, Novell Netware, NonStop, Plan 9, VMS, etc.

3. Descripción general del proyecto

3.4.4. Bibliotecas y módulos usados

Perl

Se ha usado la versión 5.8.8 para el desarrollo de este proyecto. La primera revisión de la versión 5.8 de Perl, la 5.8.0, introdujo diversas mejoras, pero sólo nos afecta realmente la introducción en la distribución estándar del módulo `Locale::Maketext`, usado para localización de los mensajes de error.

Es posible por lo tanto que los convertidores funcionen instalando manualmente dicho módulo y demás dependencias con las versiones 5.6.x, que implementan la imprescindible sintaxis orientada a objetos. No ha habido tiempo para hacer pruebas al respecto, sin embargo.

Ha sido necesario el uso de varios módulos: algunos vienen incluidos en la distribución estándar de Perl, y otros se han de instalar por separado. Estos últimos se instalan automáticamente durante la compilación de los convertidores siempre que las bibliotecas requeridas estén presentes en el sistema. Se tratan de:

File::ShareDir Permite calcular las rutas a ficheros instalados a través del mecanismo de directorios automáticos de Perl en localizaciones accesibles por todos los usuarios del sistema y acceder a ellos. De la instalación se ocupa `Module::Install`.

File::Spec Realiza transformaciones sobre rutas a ficheros de manera transportable, como convertir rutas relativas a absolutas, por ejemplo.

Getopt::Long Mediante este módulo, se pueden recibir opciones de la línea de órdenes en formato POSIX con las extensiones GNU, como la posibilidad de introducir argumentos entre las opciones. Se pueden también usar opciones en formato abreviado y reunir las, para usuarios expertos.

Module::Install Permite escribir fácilmente instaladores de módulos Perl que sigan las mejores prácticas del CPAN, e instalen de forma automática todas las dependencias, de manera recursiva. Los instaladores resultantes son compatibles con cualquier herramienta que funcione con instaladores generados mediante el módulo estándar *de facto* `ExtUtils::MakeMaker` o con `Module::Build`.

Incorpora funcionalidades para verificar dependencias de ejecución y compilación, crear distribuciones del código, lanzar las pruebas de unidad, e instalar ejecutables y ficheros para su posterior uso con `File::ShareDir`, entre otras.

PAR Permite crear ficheros `.par` de módulos y/o guiones Perl que funcionan de forma muy similar a los `.jar` en Java: son archivos comprimidos que renen todo el código necesario, y pueden ejecutarse de forma muy sencilla utilizando la herramienta de línea de órdenes `parl`.

PAR::Packer Este mdulo permite crear ejecutables monolticos especficos a la arquitectura y sistema operativo actual de cualquier guin Perl. Incorpora una herramienta de lnea de rdenes llamada `pp` que rene en el mismo ejecutable los mdulos, bibliotecas y dems ficheros usados, e incluso el propio intrprete de Perl.

A diferencia de `PAR`, los ficheros generados son completamente autosuficientes, y no requieren por lo tanto de ningn proceso de instalacin previo.

Pod::Usage Permite documentar fcilmente el uso correcto del programa, pudiendo usar la seccin POD incrustada en el guin principal para generar ayuda en lnea en formato `man` o texto puro.

Test::More Marco de pruebas de unidad basado completamente en Perl que reemplaza y extiende al mdulo `Test::Simple`.

Test::Pod::Coverage Mdulo ocupado de implementar pruebas de unidad sobre la cobertura de la documentacin incluida en el propio cdigo Perl en formato POD. Se asegura de que todo mtodo pblico definido por todos los mdulos tenga documentacin asociada.

XML::LibXML Es uno de los mdulos que puede emplear `XML::Validate` para implementar su funcionalidad. Provee de una interfaz Perl a la biblioteca C XML del proyecto GNOME, `libxml2` (vase <http://xmlsoft.org/>), que evidentemente ha de estar ya instalada.

XML::SemanticDiff Permite comprobar las diferencias entre dos documentos XML, notificndolas a travs de una serie de manejadores de eventos. Estos manejadores de eventos pueden aadir lgica especfica con informacin acerca de qu diferencias resultan de inters y qu otras diferencias no son tan importantes, y tambin pueden etiquetar dichos sucesos con informacin adicional.

XML::Validate Da la posibilidad de validar documentos XML fcilmente usando diversas bibliotecas externas. Esto permite usar ficheros DTD, por ejemplo. Actualmente slo se usa en las pruebas de unidad, pero en un futuro se podra emplear para la validacin de todos los resultados producidos por `ACL2::Procesador`.

XML::Writer Para salida con verificacin bsica de XML, usando un interfaz sencillo.

Java

Por otro lado, la amplitud de la J2SE 5.0 ha permitido implementar XMLEye limitando las dependencias externas a un *Look & Feel* libre distribuido bajo licencia BSD llamado JGoodies Looks, disponible en <https://looks.dev.java.net/>.

La implementacin de JAXP (Java API for XML Parsing) 1.3 de J2SE 5.0 usa dos bibliotecas de Apache:

3. Descripción general del proyecto

XSLTC 2.6.0 Implementa los estándares del W3C XSLT 1.0 y XPath 1.0. Basado en Apache Xalan, con una serie de parches de Sun.

Esta nueva versión reemplaza al antiguo motor XSLT Xalan interpretado. En este nuevo motor, toda hoja XSLT es compilada de antemano a instrucciones de la JVM, generando un *translet*, cuya ejecución es notablemente más rápida.

La limitación de XSLTC consiste en que no permite elementos de extensión, parte del estándar XSLT 1.0. Tampoco puede usar fuentes JDBC (Java DataBase Connectivity) en la transformación, ni programas incrustados escritos en otros lenguajes (Java, por ejemplo).

Xerces 2.6.2 Proporciona analizadores W3C DOM nivel 3 y SAX 2.0.2, junto con implementaciones de otras partes del JAXP 1.3. De forma análoga a XSLTC, se halla basado en la versión oficial de Apache Xerces, junto con una serie de parches seleccionados.

Esta biblioteca reemplaza a Crimson, otro analizador XML de Apache, incluido en J2SE 4.0 como parte del JAXP 1.1. Este analizador estaba centrado más en la simplicidad y bajo consumo de memoria, y se halla listado actualmente como “proyecto en hibernación” en <http://xml.apache.org>, la web de proyectos XML de Apache.

Para desarrollar XMLEye, se han usado las siguientes herramientas:

Ant 1.7.0 Sistema de gestión de tareas de compilación, empaquetado, generación de documentación y lanzamiento de pruebas de unidad, entre otras. Es similar al conocido GNU Make, siendo la principal diferencia su soporte multiplataforma, gracias a estar implementado en Java.

Eclipse 3.3.2 IDE (Integrated Development Environment) desarrollado en sus inicios por IBM que es, junto con NetBeans, uno de los más usados hoy en día. Aunque es particularmente popular en la comunidad Java, existen versiones para otros lenguajes, como C o C++. Dispone de herramientas para hacer rápidamente refactorizaciones en el código, facilitando enormemente cualquier cambio al diseño de una aplicación.

JUnit 3.8.2 Popular marco de desarrollo de pruebas de unidad, inspirado en el marco SUnit para Smalltalk, que ha originado otros marcos parecidos, como CppUnit para C++. Se ha usado la versión 3.8 y no las versiones 4.x dado que estas últimas no están soportadas por Ant, tras cambios en su API (Application Programming Interface).

3.5. Requisitos para futuras versiones

3.5.1. XMLEye

- Filtrado de las hojas de visualizacin y preprocesado segn el tipo de documento actualmente abierto.
- Bsqueda mediante consultas XPath arbitrarias para usuarios avanzados. Marcado de nodos coincidentes en una bsqueda en la vista de rbol.
- Creacin de nuevos tipos de documento en la propia interfaz.
- Crear nuevas hojas para otros formatos basados en XML, como las trazas de ejecucin del motor XSLT Saxon, los ficheros de proyecto de GNOME Planner, o las demostraciones de los sistemas Mizar [?] o Prover9 (<http://www.cs.unm.edu/~mccune/mace4/>).
- Cambio de la biblioteca Xalan a Saxon, pasando de XSLT 1.0 a XSLT 2.0, que incorpora muchas nuevas funcionalidades.
- Mejora del motor XHTML/CSS (Cascading Style Sheet(s)). Se evaluarn los motores del proyecto Lobo (ver <http://lobobrowser.org/>), que aade soporte para JavaFX, y del proyecto Flying Saucer (<https://xhtmlrenderer.dev.java.net>), aparentemente ms maduro y fcil de integrar. JavaFX podra ser un nuevo formato de visualizacin que permitira crear interfaces Swing completas.
- Integracin de lenguajes de *scripting* en el interior de las hojas XSLT, para poder implementar funcionalidad adicional de manera ms sencilla y sin modificar XML-Eye.
- Publicacin de documentos con visualizaciones XHTML del rbol y los nodos seleccionados, posiblemente integrando capacidades de bsqueda. Idealmente, publicar el documento debera producir una carga mnima en la mquina que lo sirviera.
- Creacin de un ejecutable para Windows mediante Launch4J (<http://launch4j.sourceforge.net/>).
- Creacin de una versin lanzable desde el navegador mediante Java WebStart.
- Envio de los paquetes Debian al repositorio oficial.
- Traduccin completa del cdigo al ingls.

3. Descripción general del proyecto

3.5.2. ACL2::Procesador

- Extracción de información de la salida de mayor variedad de eventos, como `defaxiom` o `verify-guards`, y de `rdenes` que se expandan a eventos, como `make-event` o cualquier uso de una macro Lisp.
- Mayor cantidad de información generada automáticamente, como por ejemplo sumarios globales.
- En paralelo con el requisito anterior, aumento de la variedad y potencia de las hojas de visualización y preprocesado.
- Integración con la ayuda HTML de ACL2.
- Envío de los paquetes Debian al repositorio oficial.
- Envío de la distribución al repositorio CPAN.
- Traducción completa del código al inglés.

3.5.3. YAXML::Reverse

- Implementación de hojas de visualización y preprocesado para diversos lenguajes basados en YAML.
- Depuración del procesamiento de anclas, alias y etiquetas.
- Soporte de YAML 1.2 cuando quede finalizado (actualmente se halla en estado de borrador).
- Envío del paquete Debian al repositorio oficial.
- Envío del módulo al repositorio CPAN.

4. Desarrollo del proyecto

Procederemos a describir el análisis, diseño, implementación y realización de pruebas del proyecto como un todo, considerando la separación de `YAXML::Reverse` y `ACL2::Procesador` respecto de `XMLEye` como un detalle de diseño, motivado por el deseo de aprovechar los puntos fuertes de cada lenguaje y hacer tanto al visor como a los conversores reutilizables.

En particular, las secciones de diseño y análisis describirán la última iteración de cada producto, al contener estas la arquitectura y diseño definitivos, que también han ido cambiando y han ido siendo refinados a lo largo de las iteraciones.

4.1. Proceso

Se ha seguido la metodología XP (eXtreme Programming) en el desarrollo de este proyecto. Fue creada por Kent Beck, Ward Cunningham y Ron Jeffries durante el desarrollo del C3 (Chrysler Comprehensive Compensation System), un sistema unificado de gestión de minas.

La parte “Extreme” de XP consiste en llevar prácticas conocidas y recomendadas de la ingeniería de software a su máxima expresión. Por ejemplo, si las pruebas son buenas, entonces todo el código deberá tener pruebas, y además estas deberán estar automatizadas y ejecutarse continuamente.

4.1.1. Origen

Su popularización se dio a través de la participación conjunta de estos tres autores, con intervenciones del resto de la comunidad, en la web [?], fundada en 1995.

En 1999 su popularidad se vio aumentada en gran medida tras la publicación de la primera edición de “Extreme Programming Explained”.

Para este proyecto, se siguió la segunda edición [?], publicada en 2004, que en respuesta a las críticas recibidas introdujo importantes cambios y reorganizaciones, como la sustitución de las 12 prácticas por un núcleo de prácticas obligatorias apoyado por una serie de prácticas opcionales.

4. Desarrollo del proyecto

Ya se mencion anteriormente en §2.5 otra obra relacionada con XP ([?]).

4.1.2. Características

XP es una metodología gil de desarrollo de software, aplicable por lo general a proyectos de pequeño y medio tamaño, en un equipo de hasta 12 personas.

En los mtodos giles, se descarta todo documento o procedimiento innecesario para el proyecto, permitiendo al equipo avanzar rpidamente concentrndose en lo importante y sin perder tiempo en formalismos.

En XP, los detalles se comunican en conversaciones directas entre las partes implicadas (incluyendo al cliente), aclarando confusiones y dudas en el momento. As, XP se dirige ms a las personas que al proceso, y ms a la obtencin de software til que al seguimiento de un proceso rgido y burocrtico.

En cada iteracin, se entrega una versin funcional del software que el cliente puede probar para ampliar la informacin disponible en la siguiente iteracin.

Una idea que se suele tener respecto a XP (por aquello de que es “Extreme”) es que no se realiza documentacin en absoluto, sea lo que sea. El requisito de documentacin externa al proyecto es muy comn, por ejemplo, y debe ser atendido. Ron Jeffries trata de corregir ste y otros malentendidos en [?].

A diferencia de otras metodologas, donde el cliente es una entidad externa al proyecto con el que se slo se comunica el analista, en XP es una parte integral del equipo. Se ocupa de formular los requisitos a nivel conceptual, darles su prioridad, y resolver dudas que pueda tener el resto del equipo.

XP nos recuerda as que el software existe para dar un valor aadido al cliente, y no por s mismo. As, las decisiones acerca de la funcionalidad deseada son del cliente, y las decisiones tcnicas y el calendario lo establecen los desarrolladores.

Valores

Aqu se describen algunos de los valores que todo proyecto basado en XP debe tener en mente:

Simplicidad La solucin ms sencilla suele ser la mejor. Esto no quiere decir que se use la primera solucin que se nos ocurra. La solucin escogida debe:

- Hacer todo lo esperado.
- No contener ninguna duplicacin.
- Pasar todas las pruebas.

- Tener el mnimo nmero de elementos.

Lo que este valor intenta evitar es el diseo excesivo al tratar de resolver problemas an no planteados. Dicho trabajo podra ser intil o incluso contraproducente si el cliente cambiara de opinin acerca del resto de los requisitos an no implementados. No se trata de anticipar el cambio, sino de adaptarse a l.

Comunicacin Consiste en la comunicacin transparente y sin obstculos entre todas las partes del equipo, incluido el cliente. La falta de comunicacin induce a problemas o malentendidos que podran haberse resuelto fcilmente de lo contrario.

Aprendizaje Ms que intentar capturar toda la informacin de una vez, un proyecto basado en XP debe de concentrarse en el da a da, aprendiendo paso a paso qu es lo que se espera exactamente del programa, dado que ni siquiera el cliente lo sabe con seguridad.

Dicha informacin puede venir de muchas fuentes: del cliente, de la experiencia propia, de las pruebas automatizadas, etc. Otras veces, cuando no se sabe cmo atacar un problema, se puede aprender realizando experimentos y evaluando varias alternativas, para as poder elegir la mejor y evitar discusiones improductivas.

Principios

Para seguir esos valores, se proponen una serie de principios, que se concretan a travs de una serie de prcticas recomendadas. Dichos principios incluyen por ejemplo:

Flujo Se deben de realizar constantemente todas las actividades del desarrollo de software: anlisis, diseo, implementacin, integracin y pruebas.

Esto evita la acumulacin de riesgos a la hora de integrar y validar, realizando entregas frecuentes con pequeos riesgos e incrementos de funcionalidad, ms que una nica entrega con alto riesgo y toda la funcionalidad.

Margen En todo plan, se debe de tener margen para en el caso de no tener suficiente tiempo poder dejar algunas historias para posteriores versiones.

ste ha sido el caso de sobre todo ACL2 : :Procesador, que slo procesa un subconjunto limitado de la salida de ACL2, pero es completamente funcional dentro de dicho subconjunto, y mantiene un diseo que facilitar futuras ampliaciones.

Calidad La calidad no es algo opcional, y disminuir el nivel aceptado de calidad no garantiza una reduccin del tiempo de desarrollo. De hecho, suele aumentarlo.

El control del tiempo de un proyecto no debe ser as basado en su calidad, sino sobre todo en su alcance. Es mejor hacer una sola cosa bien que varias mal.

4. Desarrollo del proyecto

Centrado en las personas Son las personas las que realizan el software, no la metodología de desarrollo.

Una metodología de desarrollo de software ha de tener en cuenta las necesidades y limitaciones de las personas. En una labor fundamentalmente creativa como es el desarrollo de software, se necesita que el equipo trabaje a su máximo rendimiento de forma sostenible.

Prácticas

Las prácticas de XP se dividen en dos partes: un núcleo de reglas que todo proyecto XP debe cumplir, y otras prácticas recomendadas pero no obligatorias, que suelen tener como requisito el cumplimiento de todas las reglas principales.

En el caso de este Proyecto, hay tareas que sencillamente no se han podido seguir, como “Programación en Parejas”, o “Trabajar Juntos” (reunir al equipo de desarrollo en un lugar), por razones obvias.

Algunas de las prácticas seguidas son:

Historias Las historias de usuario son algo completamente distinto de los casos de uso. Describen una funcionalidad o propiedad deseada del sistema, en palabras del usuario y no del desarrollador. Mediante ellos se pueden expresar todo tipo de requisitos, no sólo los funcionales, y en ellos se basan las pruebas de aceptación.

Por otro lado, un caso de uso representa de forma abstracta la interacción entre el sistema y los elementos externos, incluyendo el actor principal y los secundarios.

XP no prohíbe el empleo de los casos de uso. De hecho, se pueden usar en combinación, sirviendo el caso de uso como una generalización de una o varias historias de usuario particulares.

Así, puede verse que una historia de usuario es algo mucho más concreto, generalmente pequeño y fácilmente estimable, mientras que un caso de uso se podrá ver como una clase de interacciones concretas que el sistema ha de posibilitar. Ambos describen el *qué*, pero lo hacen bajo distintos enfoques.

El equipo de desarrollo puede realizar estimaciones del esfuerzo que requiera una historia y presentárselas al cliente, dándole más información para decidir qué historias quiere implementadas en cada iteración. Puede que se decida posponer una historia, o dividirla en varias.

En mi caso, los “clientes” han sido mis directores de proyecto, especialistas en ACL2, y por lo tanto usuarios potenciales del proyecto.

Integracin Continua La prueba e integracin de los cambios se realiza de forma continua y a pequenos pasos. Esto se sigue del hecho de que corregir los fallos introducidos en el software es tanto ms complicado cuanto ms amplios sean los cambios.

Ciclo Semanal La planificacin se ha realizado en ciclos cortos, de duracin aproximada de una semana, y se ha mantenido una comunicacin semana a semana con los “clientes”, en este caso los directores del proyecto.

Compilacin en 10 Minutos Aunque para un proyecto de este tamao no sea una tarea difcil de realizar, se puede considerar un requisito para “Integracin Continua”.

La idea consiste en mantener siempre el tiempo de compilacin y ejecucin de todas las pruebas automticas pertinentes por debajo de 10 minutos, para poder realizar integracin continua del cdigo de todo el equipo y evitar tanto problemas de integracin de ltima hora como esperas innecesarias.

Programacin Basada en Pruebas Los componentes ms importantes de XMLEye han sido desarrollados de esta forma usando JUnit, en cuyo desarrollo particip el propio Kent Beck. Existen otras herramientas para realizar pruebas de unidad: la obra [?] describe las ms conocidas en Java.

En este mtodo de programacin, se escribe primero una prueba que demuestra que la funcionalidad deseada est implementada. Una vez est escrita la prueba, se comprueba que sta falle, tras aadir el cdigo estrictamente necesario para que todo compile.

Es entonces cuando se aade el cdigo necesario a la implementacin para hacer que se superen dicha prueba y las anteriores.

Adems de garantizar mayor correccin, el uso de este estilo obliga al desarrollador a estructurar su cdigo para que sea fcil de probar, hacindolo modular y cohesivo. Refactorizaciones posteriores en el diseo pueden garantizar inmediatamente la misma funcionalidad que la versin anterior.

En el caso de `ACL2::Procesador`, se ha seguido un proceso similar:

1. Se decide el enunciado Lisp a tratar junto con su salida de ACL2.
2. Se define a grandes rasgos el resultado XML deseado a partir de la salida de ACL2.
3. Se lanza el post-procesador sobre el enunciado y la salida.
 - 3a. Se obtiene el resultado deseado: se continua con el proceso.
 - 3b. No se obtiene el resultado deseado: se refina el post-procesador y se vuelve al paso 3.
4. Se marca el resultado XML como vlido.

4. Desarrollo del proyecto

5. Se lanzan las pruebas de regresin, comparando automticamente cada resultado XML obtenido por la versin actual del post-procesador con las ltimas versiones validadas:
 - 5a. No hay diferencias: se escoge otro enunciado y su salida y se repite el proceso.
 - 5b. Hay diferencias: si constituyen regresiones, se corrigen y se vuelve al paso 5. En caso de falso positivo, se refina la comparacin lo estrictamente necesario para evitarlo si se puede, o de lo contrario se marca la salida XML como vlida y se reemplaza a la anterior.

El proceso no difiere mucho de la programacin basada en pruebas usual: se definen los resultados deseados y entonces se aade la funcionalidad necesaria. La diferencia radica en que la prueba es de todo el post-procesador, y no de una clase o mtodo, como las usuales pruebas de unidad.

El proceso completo se halla dirigido a travs de los mecanismos usuales de pruebas de unidad empleados para cualquier mdulo del CPAN, como `Test::More` o `Test::Simple`.

`YAXML::Reverse` hace algo muy parecido a `ACL2::Procesador`: toda mejora se realiza tras aadir un nuevo documento YAML 1.1 o JSON al conjunto de casos de prueba. Cada caso de prueba es ejecutado de la siguiente forma:

1. El documento origen se convierte a XML mediante `YAXML::Reverse`.
2. El documento XML resultante se convierte de nuevo a YAML mediante una versin mejorada de la hoja XSLT de `YAXML`.
3. Se deserializa el documento original y el documento final y se comprueban que las dos estructuras de datos resultantes son iguales elemento a elemento.

Diseo incremental Las metodologas tradicionales, siguiendo el modelo de ciclo de vida de cascada, intentaban realizar el anlisis y diseo de antemano, siguiendo el modelo de la ingeniera tradicional.

Se realizaba la analoga entre la construccin de una casa y la de un proyecto software: una vez estaban implantados los cimientos, cambiar la estructura del edificio era varios rdenes de magnitud ms caro.

XP afirma lo contrario: el diseo del programa no es algo que se haga una sola vez y quede fijo por el resto de la vida del programa. Se debe refinar continuamente en funcin de las necesidades del momento.

Para evitar el aumento del coste de las modificaciones, se dispone de otras prcticas adems de sta, como el uso de pruebas automticas, la comparticin de cdigo, la programacin en parejas y la comunicacin fluida con el resto del equipo.

As evitamos tanto el diseo excesivo, que supone una prdida de tiempo, como la falta de diseo (conocida como el anti-patrn *Bola de Barro* [?]), que dificulta la introduccin de cambios.

4.2. Herramientas de modelado usadas

4.2.1. BOUML

Genera cdigo Java/C++/Python/Perl a partir de diagramas UML (Unified Modelling Language), realiza ingeniera inversa de Java/C++, y permite dibujar diversos diagramas UML 2.0, como los diagramas de clases, despliegue, componentes o casos de uso, entre otros.

Est disponible para Windows, GNU/Linux y Mac OS X en la web <http://bouml.free.fr>. Se trata de un programa escrito en C++ usando la versin 3.3.8 de la biblioteca Qt de la compaa Trolltech, conocida por ser la biblioteca sobre la cual se halla implementado el gestor de escritorio KDE.

4.2.2. UMLet

Esta otra herramienta, a diferencia de BOUML, no trata de ser un CASE, sino una simple herramienta para dibujar diagramas.

Est diseada para ser flexible y muy fcil de aprender y usar. Dibujar un diagrama consiste en utilizar componentes de una serie de paletas personalizables, pudiendo modificar propiedades editando un pequeno bloque de texto asociado a cada elemento.

Se ha usado para cubrir algunas carencias de BOUML, como por ejemplo en el diagrama de arquitectura del sistema.

El programa se halla disponible como una aplicacin Java independiente o como un plug-in para Eclipse en <http://www.umlet.com>.

4.3. Requisitos

4.3.1. Interfaces externas

En este apartado se describen los requisitos que deben cumplir las interfaces con el hardware, el software y el usuario.

4. Desarrollo del proyecto

- La visualización permitir a un usuario acceder directamente a nodos relacionados del árbol del documento.
- La visualización debe de usar texto con formato, para proporcionar mayor información visual que el texto puro.
- Las operaciones de larga duración no deben de afectar al tiempo de respuesta de la interfaz gráfica.
- El estado de la interfaz mostrado al usuario debe de ser siempre coherente con el estado interno de la aplicación.
- El sistema operativo ha de posibilitar el lanzamiento y monitorización de otros procesos en segundo plano.
- XMLEye ha de ser capaz de adaptarse a los distintos esquemas de nombrado de ficheros disponibles en cada sistema operativo.
- El formato de salida de `ACL2::Procesador` debe estar bien definido.

Se adjunta un prototipo de la ventana principal en la figura 4.1 de la página 63. Puede verse cómo, en esta nueva versión de XMLEye, se integra el soporte para visualización simultánea de varios documentos en pestañas.

También hay prototipos para los diálogos:

- Búsqueda: figura 4.2, página 63.
- Gestión de descriptores de formatos de documentos: figura 4.3, página 64.
- Información del producto: figura 4.4, página 64.

4.3.2. Funcionales

- Generar ficheros XML que engloben la información de una fuente Lisp, su demostración por ACL2 asociada y referencias a todas las otras fuentes Lisp (libros de ACL2) de que dependa.
- Permitir la visualización de documentos pertenecientes a lenguajes basados en YAML 1.1, y la posterior creación de hojas específicas a stos.
- Navegar por múltiples documentos a la vez, abstrayendo al usuario si ste lo desea de los detalles de la estructura de sus versiones XML.
- Mantener un historial de documentos recientes, fácilmente accesible por el usuario.
- Permitir editar los documentos visualizados con el editor favorito del usuario para el formato correspondiente, si estn presentes en el sistema. Puede que en ciertos casos slo dispongamos del XML final.

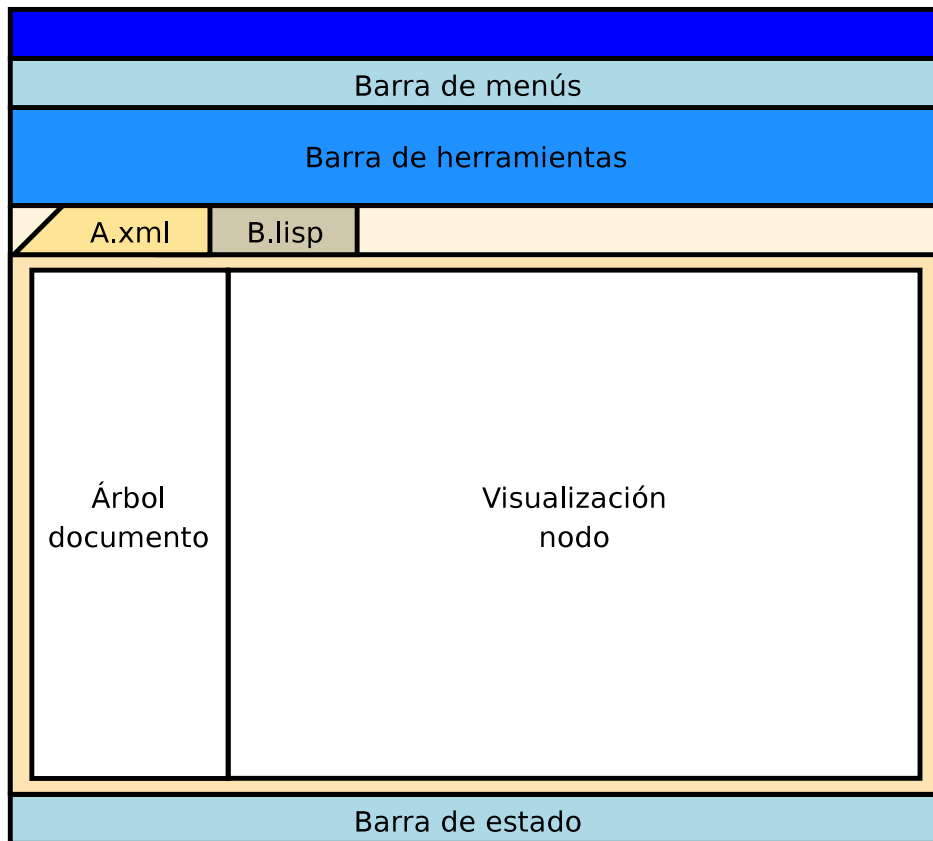


Figura 4.1.: Prototipo de la ventana principal

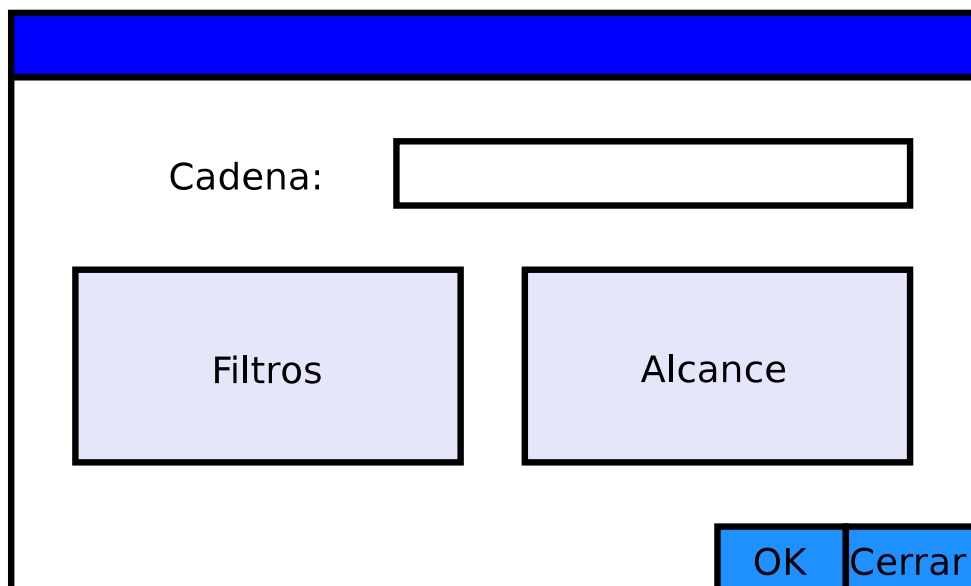
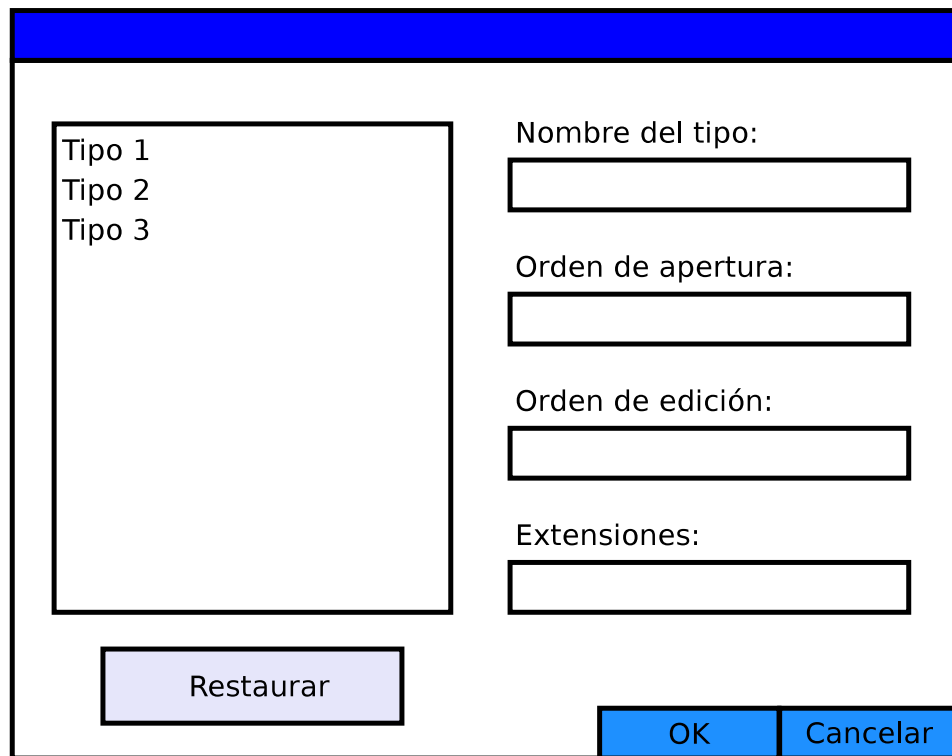


Figura 4.2.: Prototipo del dilogo de bsqueda



Prototipo de un diálogo de gestión de descriptores de formatos de documentos. El diálogo tiene un encabezado azul. A la izquierda, un panel con el título 'Tipo' contiene una lista con los elementos 'Tipo 1', 'Tipo 2' y 'Tipo 3'. A la derecha, hay cuatro campos de entrada etiquetados como 'Nombre del tipo:', 'Orden de apertura:', 'Orden de edición:' y 'Extensiones:'. En la parte inferior, hay tres botones: 'Restaurar' (de color gris), 'OK' (de color azul) y 'Cancelar' (de color azul).

Figura 4.3.: Prototipo del diálogo de gestión de descriptores de formatos de documentos



Prototipo de un diálogo de información del programa. El diálogo tiene un encabezado azul. En la parte superior, hay dos pestañas: 'Info' (activada, de color amarillo) y 'Licencia' (de color gris). El área principal del diálogo es un recuadro con un fondo azul claro y el texto 'Contenido HTML' en el centro. En la parte inferior derecha, hay un botón 'Cerrar' de color azul.

Figura 4.4.: Prototipo del diálogo de información del programa

- Regenerar cualquier documento abierto si se producen cambios en l en algñ momento, pudiendo mantener el editor y el visor abiertos a la vez.
- Relacionar los elementos del documento entre s, y con elementos específicos de otros documentos.
- Generar automáticamente informacin adicional que pueda ser de inters para el usuario.
- Realizar bsquedas por el documento, sin prdidas de informacin respecto del texto original, usando diversos filtros y pudiendo limitar su alcance.
- Repetir la ltima bsqueda realizada.
- Personalizar las rdenes usadas para lanzar el editor o el conversor de cualquiera de los formatos aceptados, y volver a los ajustes por defecto en cualquier momento.
- Cambiar el tipo y/o formato de la informacin mostrada durante la ejecucin del programa.

4.3.3. Atributos del sistema software

En este proyecto, la mantenibilidad es fundamental para que posteriores ampliaciones y correcciones se hagan de forma rpida y sencilla, dada la complejidad de la herramienta cuyo uso se pretende facilitar.

Tambin sera deseable la separacin entre la lgica de visualizacin y el resto del sistema por la posibilidad de que los usuarios (vase §3.3) extiendan sus capacidades de visualizacin sin tener que modificar el cdigo fuente del proyecto, slo modificando o aadiendo ficheros de datos. Separar la lgica de conversin de XMLEye de cuestiones específicas a `ACL2::Procesador` posibilitar su uso para formatos distintos a demostraciones de ACL2, y que tampoco tengan por qu ser basados en XML.

La transportabilidad, aunque no imprescindible, se corresponde con el deseo de permitir un libre uso de este proyecto como una herramienta ms para los usuarios de ACL2 que limite en lo mnimo posible su eleccin de sistema operativo.

Con vistas a ampliar la base de usuarios potencial de XMLEye y sus extensiones, cobrar importancia no solo la facilidad de uso, sino tambin la facilidad de instalacin, controlando el nmero de dependencias externas y, cuando esto no sea posible, reduciendo su impacto en la complejidad de la instalacin.

4.4. Anlisis del sistema

4.4.1. Historias de usuario

Como ya se coment en §4.1.2 (pgina 58), los proyectos basados en XP se estructuran en torno a las historias de los usuarios, que describen un comportamiento o propiedad deseada del sistema en sus propias palabras.

Mientras se redactan, el equipo de desarrolladores, tras realizar un anlisis superficial del trabajo implicado con ayuda del cliente, asigna a cada historia un tiempo estimado de finalizacin en das ideales y un riesgo. El cliente puede, en funcin de esos factores y de sus intereses, marcar la prioridad de una historia.

Una historia de usuario puede ser de muy alto nivel o de muy bajo nivel, pudiendo unirse varias historias de bajo nivel en una sola, o dividirse una de alto nivel en varias ms sencillas.

Pasaremos a listar las historias de usuario que se han ido acumulando a lo largo de la comunicacin con los clientes durante el desarrollo del PFC, es decir, mis directores de proyecto, usuarios de ACL2 y por lo tanto partes interesadas.

En los casos de `ACL2::Procesador` y `YAXML::Reverse`, hay varias historias pospuestas para futuras iteraciones, sin que ello impida entregar una versin funcional (recordemos el principio “Margen” de XP).

Dividiremos la mayora de las historias en tareas, dando estimaciones iniciales del tiempo necesario para cada una. Otras sern lo bastante concretas como para no necesitar ninguna subdivisin.

En esta lista de historias de usuario no se incluyen aquellas que se completaron durante el Proyecto Fin de Carrera “Post-procesador y Visor de Demostraciones del Sistema ACL2”.

XMLEye

1. “Aadir soporte para visualizar varios ficheros a la vez y enlazarlos entre s.”

Esta historia de usuario es muy corta para todo el trabajo que implica:

- a) Concentrar el estado de visualizacin del documento actual en un modelo de presentacin, para as poder cambiar entre los estados de cada documento: 3 das ideales. Completada.
- b) Integrar de nuevo y depurar toda la funcionalidad de la aplicacin con este nuevo diseo: 3 das ideales. Completada.

- c) Aadir una interfaz TDI a XMLEye, y asegurar su integracin con el *Look and Feel* instalado: 4 das ideales. Completada.
 - d) Aadir la posibilidad de establecer hipervnculos entre varios documentos: 2 das ideales. Completada.
2. “El visor no debera saber absolutamente nada de ACL2. Nada en su interfaz grfica ni en su implementacin nos debe recordar que podemos trabajar con ACL2. Pero s podra saber algo de un procesador externo que tradujera ficheros de un formato, del que el visor tampoco sabe nada, a XML. Tambin podra conocer la existencia de hojas de estilo XSLT y de herramientas externas, como un editor, como de hecho ocurre ahora.”
- a) Mover toda la lgica propia de ACL2 a `ACL2::Procesador`, y retirarla de XMLEye: 0,5 da ideal para retirarla. Vase la historia 5 de `ACL2::Procesador` para la estimacin de su recreacin. Completada.
 - b) Definir la estructura y contenidos de un descriptor de formato de documento: 0,5 da ideal. Completada.
 - c) Implementar y depurar la estructura en 2 niveles del repositorio de descriptores, empleando ciertas variables del entorno si se hallan definidas: 2 das ideales. Completada.
 - d) Internacionalizar las cadenas contenidas en los descriptores: 0,5 da ideal. Completada.
 - e) Integrar la informacin de los descriptores con los controles de apertura y edicin de documentos: 2 das ideales. Completada.
 - f) Crear un dilogo de gestin de los descriptores instalados: 1 da ideal. Completada.
3. “Vigilar directamente el fichero fuente en caso de cambios, en vez de solamente cuando se cierra el editor. As, si uno abre el editor, hace un par de cambios y guarda (sin cerrar el editor), tambin se actualizar. En caso de que se desactive temporalmente la reimportacin automtica, tambin debera funcionar correctamente si en ese intervalo se hicieron cambios.”

Esta historia implica:

- Resolver unas condiciones de carrera conocidas que produzcan fallos intermitentes en la reimportacin automtica. Tiempo estimado: 1 da ideal. Completada.

4. Desarrollo del proyecto

- Refinar el diseo del momento para evitar que al reabrir el documento, apareciera como una nueva pestaa. Tiempo estimado: 1 da ideal. Completada.
- Implementar la deteccin y notificacin con un hilo pausable de baja prioridad en segundo plano, activado de forma intermitente. Tiempo estimado: 1 da ideal. Completada.

ACL2::Procesador

1. “Normalizar la estructura de `ACL2::Procesador` para que siga la de un mdulo del CPAN y se porte mejor a la hora de en el futuro hacer un paquete Debian.”

Tiempo estimado: 2 das ideales. Completada.

2. “Llevar la cuenta de en qu paquete (espacio de nombres) estamos trabajando. Puede saberse por el indicador (prompt).”

Se requerira aadir el filtrado de la orden `in-package` y del indicador de ACL2, modificando en consecuencia la salida XML.

Tiempo estimado: 1 da ideal. Completada.

3. “Tratar el ejemplo de las Torres de Hanoi de [?], dividiendolo en tres partes: `books/hanoi.lisp`, libro que define el algoritmo para resolver el problema y todo lo necesario excepto el propio teorema del nmero de intercambios; `books/hanoi.acl2`, que se encarga de definir el paquete y certificar el libro; y `hanoi-use.lisp`, que incluye el libro y contiene el teorema con el nmero de intercambios.”

Una vez la capacidad de visualizar y enlazar entre s varios documentos est aadida en XMLEye (vase su historia 1), habra que:

- Crear un mdulo de deteccin de dependencias que utilice un simple anlisis superficial de sus rdenes, de forma recursiva, y genere un ndice de cada libro con los identificadores de los eventos que define y los libros de que depende a su vez. Tiempo estimado: 3 das ideales. Completada.
 - Implementar el procesamiento de la salida de los eventos `include-book`, `certify-book`, `defpkg` e `in-package`. Tiempo estimado: 2 das ideales. Completada.
4. “Mejorar los informes de errores de las pruebas de regresin, evitando que fallen con diferencias insignificantes.”

Requiere:

- Volver a implementar las pruebas de regresión, esta vez calculando la diferencia entre los documentos no a partir de las líneas de texto que lo componen, sino usando el árbol DOM resultado de analizar los dos documentos XML. Tiempo estimado: 2 días ideales. Completado.
 - Implementar el código necesario para ignorar aquellas diferencias que no sean de interés. Tiempo estimado: 1 día ideal. Completado.
5. “Integrar `ACL2::Procesador` con `ACL2`, haciendo que automáticamente genere las salidas y las site en ficheros `.out` y `.xml` en el mismo directorio, realizando una gestión de dependencias al estilo GNU Make entre ellos y la fuente Lisp. Así evitaremos importar varias veces el mismo fichero, y se podrá simplificar el visor.”

Una vez esté terminada la tarea 3, habrá que modificar el código que recibe el texto de la demostración de `ACL2` y lanza el proceso de análisis para que invoque a `ACL2` en caso de que el fichero `.xml` no exista o se halle desactualizado respecto a la fuente Lisp.

Tiempo estimado: 1 día ideal. Completada.

6. “Manejar los eventos definidos por el usuario con `make-event` y las expansiones de macros creadas con `defmacro`.”

No está del todo clara la forma de resolver esta historia de usuario. Un análisis superficial sugiere esta división en tareas:

- Instrumentar el código Lisp con llamadas previas a `:transl` sobre la orden desconocida, sin que se produzcan modificaciones en el espaciado de cualquiera del resto de las líneas del documento. Tiempo estimado: 2 días ideales. Pospuesta.
 - Usar la orden resultante de `:transl` como el enunciado de la orden para la siguiente salida. Tiempo estimado: 2 días ideales, por posibles cambios importantes a realizar en el diseño. Pospuesta.
 - Registrar de alguna forma la expansión producida por `make-event`, tomando como base el fichero fuente `basic.lisp` del libro `make-event` incluido en la distribución de `ACL2`. Tiempo estimado: 3 días ideales. Pospuesta.
7. “Tratar el caso en que un `include-book` aparezca dentro de un `local` (y entonces lo que se incluye es local al libro, no se ve fuera). Y esto puede aparecer dentro de un `encapsulate`”.

Hay que refinar la sintaxis de los grafos de dependencias para incluir menciones al ámbito de validez de cada entrada: puede que dentro de un `encapsulate` se haga referencia a un evento `P::F`, y en el nivel global u otro `local` o `encapsulate`, se dependa de otro evento distinto con el mismo nombre. Limitando el alcance de las definiciones se podrá tratar este problema.

4. Desarrollo del proyecto

Tiempo estimado: 3 das ideales. Pospuesta.

8. “Tratar las *forcing rounds*, demostraciones realizadas tras la demostracin principal sobre cualquier hiptesis cuya aplicacin se hubiera forzado durante la demostracin anterior (la demostracin inicial, u otra *forcing round*).”

Hay que refinar la divisin jerrquica de las metas y procesar la salida especfica a las *forcing rounds* estableciendo los enlaces correspondientes.

Tiempo estimado: 3 das ideales. Pospuesta.

9. “Tratar todas las opciones que pueden acompaar a un `defthm` y a un `defun`.”

En el caso de `defthm`, se han de tratar `:rule-classes`, `:instructions`, `:hints`, `:otf-flg` y `:doc`.

Para `defun`, se han de tratar las declaraciones y la cadena de documentacin (del estilo del argumento de `:doc`).

Esta historia se habra de dividir entonces en varias tareas independientes:

- a) Soporte para cadenas de documentacin, `:rule-classes` y `:otf-flg`: 2 das ideales. Completada parcialmente.
 - b) Soporte de `:instructions`: 2 das ideales. Pospuesta.
 - c) Soporte de `:hints`: 2 das ideales. Pospuesta.
 - d) Soporte de declaraciones (incluye `:hints` e `:instructions`): 2 das ideales. Pospuesta.
10. “Marcar las salidas de los nuevos apuntes de Moore [?], *Recursion and Induction*.”

El trabajo necesario para implementar esta historia de usuario es extensivo, requiriendo implementar todas las historias anteriormente propuestas, y aadir la capacidad a `ACL2::Procesador` de tratar macros Lisp.

Tiempo estimado: 6 das ideales. Pospuesta.

YAXML::Reverse

1. “Abrir documentos YAML en XMLEye sin que suponga una prdida o alteracin de la informacin disponible.”
 - Seleccionar un mdulo Perl que permita cargar documentos YAML. Idealmente, debera ser Perl puro, pero la velocidad de carga y la fiabilidad son ms importantes. Tiempo estimado: 0,5 da ideal. Completada.

- Definir un primer prototipo del mdulo que verifique la viabilidad del enfoque usado con algunos ejemplos sencillos: 0,5 da ideal. Completada.
- Convertir el prototipo en un mdulo completo al estilo del CPAN, con pruebas de unidad basadas en Perl que verifiquen la conservacin de la informacin: 1 da ideal. Completada.

2. “Procesar el ejemplo de la factura de la web [?] de YAXML.”

Hay que establecer un mecanismo para detectar la existencia de anclas y alias en la estructura de datos generada en memoria, evitando tener que volver a analizar el cdigo fuente YAML. Podran buscarse referencias duplicadas, por ejemplo. Tiempo estimado: 1 da ideal. Completada.

3. “Preparar hojas de estilos para los marcadores de Firefox y volcados del entorno de desarrollo web Django.”

- Comprobar que la biblioteca usada analiza correctamente documentos del subconjunto JSON de YAML 1.1 que emplean los volcados de Django y los marcadores de Firefox, y en caso contrario reemplazar la existente. Tiempo estimado: 1 da ideal. Completada.
- Comprobar que se procesan debidamente documentos codificados en UTF-8 con caracteres fuera del conjunto ASCII de 7 bits. Tiempo estimado: 1 da ideal. Completada.
- Implementar hojas de estilos especializadas para los marcadores de Firefox y volcados de Django. Tiempo estimado: 2 das ideales. Pospuesta.

4.4.2. Casos de uso

Estos casos de uso nos permiten analizar de manera general y abstracta qu es lo que se requiere en conjunto de las historias de usuario de XMLEye de este Proyecto y la versin del Proyecto Fin de Carrera sobre el que se basa. El diagrama que ilustra las relaciones entre los distintos casos de uso que cumplen las metas del usuario se puede hallar en la figura 4.5.

El lmite del sistema establecido engloba nicamente a XMLEye. Los conversores externos, entre los cuales se hallan `ACL2 : :Procesador` y XMLEye, participan nicamente como un agente externo. El mtodo de comunicacin con XMLEye se aclarar ms tarde, en la fase de diseo. A menos que se indique lo contrario, los casos de uso aqu mostrados son en general de nivel de meta de usuario.

4. Desarrollo del proyecto

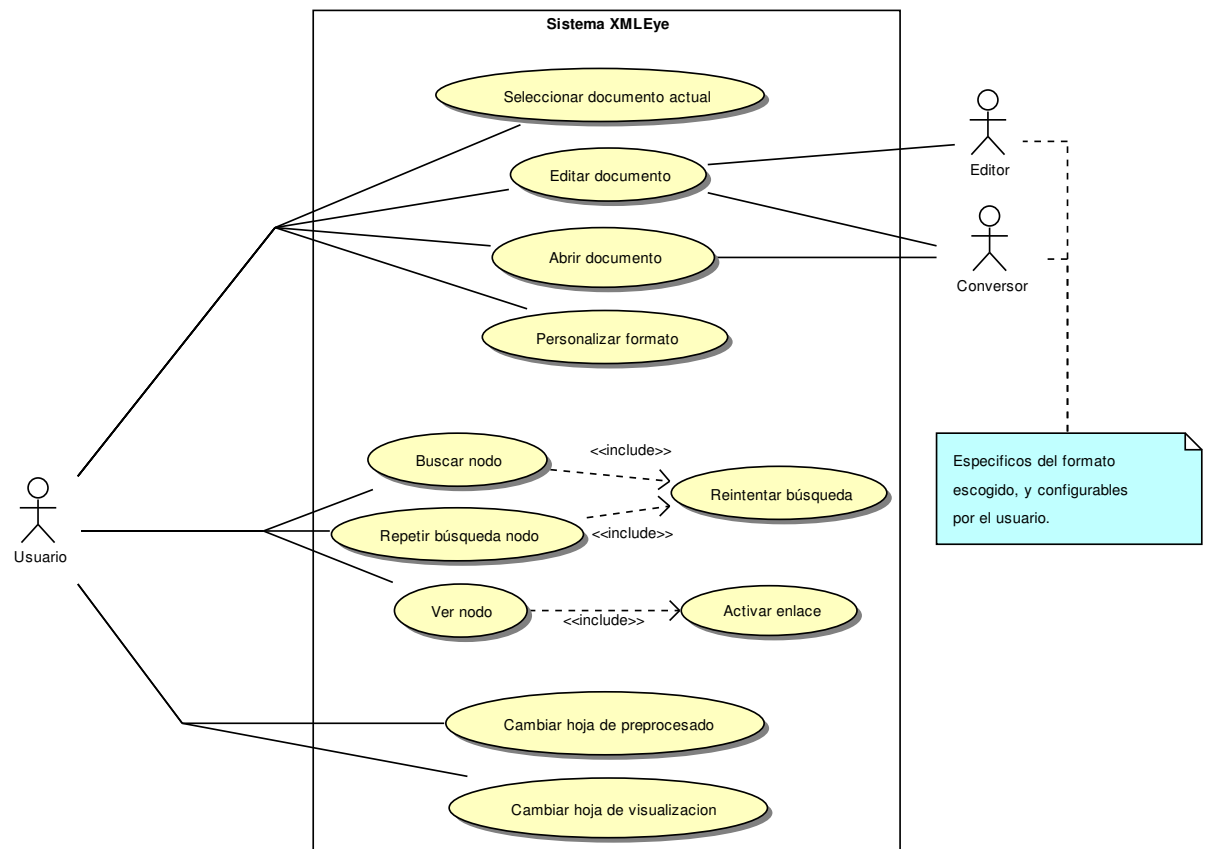


Figura 4.5.: Diagrama de casos de uso

Seleccionar documento actual

Actor principal Usuario: desea seleccionar el documento sobre el que actuar en el resto de casos de prueba de entre los documentos abiertos.

Precondiciones Existe al menos un documento abierto.

Postcondiciones Se muestra el documento seleccionado.

Escenario principal

1. El usuario indica su intencin de cambiar de documento.
2. El sistema proporciona una lista de los documentos disponibles.
3. El usuario selecciona el documento de la lista.
4. El sistema registra la seleccin, guarda el estado actual de la visualizacin actual y restaura el estado de la visualizacin del documento seleccionado, mostrndolo tal y como estaba antes de haber seleccionado otro documento.

Variaciones

- 3a. El documento es el mismo que el actual:
 1. El sistema cancela el cambio de uso.

Editar documento

Actor principal Usuario: desea editar en paralelo a la visualizacin el enunciado fuente del documento actual de forma rpida y sencilla usando su editor preferido.

Actores secundarios

Editor Editor preferido del usuario para el formato del documento actual.

Conversor Sistema externo ocupado de convertir el documento a XML, si es necesario.

Precondiciones Existe un documento abierto.

Postcondiciones Se ha abierto el documento con el editor especificado en el descriptor de formato correspondiente.

Escenario principal

1. El usuario indica su intencin de editar el enunciado del documento abierto.
2. El sistema lanza el editor, y continua con su funcionamiento normal, esperando en segundo plano a que se realicen cambios.
3. El usuario guarda algunos cambios en el documento.

4. Desarrollo del proyecto

4. El sistema detecta los cambios: se regenera la visualizacin completa del documento.
5. El sistema pasa de nuevo a la espera de ms cambios en segundo plano.

Variaciones

- 2a. La invocacin del editor ha fallado:
 1. El sistema indica el error y cancela el caso de uso.
- 2b. Se ha cerrado el documento:
 1. El sistema deja de monitorizar el documento.

Abrir documento

Actor principal Usuario: desea examinar la versin ms reciente del documento en general, para despus centrarse en los nodos de su inters.

Actores secundarios Conversor: sistema externo que se ocupa de convertir el documento proporcionado a XML.

Precondiciones Ninguna.

Postcondiciones Se carga la versin ms reciente del documento elegido por el usuario.

Escenario principal

1. El usuario indica su intencin de abrir un documento ya existente.
2. El sistema muestra los documentos disponibles.
3. El usuario selecciona un documento.
4. El sistema comprueba que el documento existe.
5. El sistema le asocia el primer formato con una extensin coincidente, y aplica su conversor, si tiene uno asociado, al documento. .
6. El sistema muestra el documento elegido, tras aadirlo a la lista de documentos abiertos.

Variaciones

- 3a. El usuario cancela la seleccin de documento:
 1. El sistema cancela el caso de uso.
- 4a. El documento no existe:
 1. El sistema muestra el error y cancela el caso de uso.

5a. No existe un formato con una extensin coincidente:

1. El sistema utiliza el formato por defecto, XML, que no tiene un conversor asociado.

Personalizar formato

Actor principal Usuario: desea cambiar las opciones de alguno de los formatos instalados.

Precondiciones Existe al menos un formato instalado en el sistema.

Postcondiciones Se cambian los ajustes locales del usuario para el formato indicado, sin cambiar los ajustes globales.

Escenario principal

1. El usuario indica su intencin de cambiar las opciones de alguno de los formatos.
2. El sistema muestra la lista de los formatos disponibles.
3. El usuario selecciona un formato de la lista.
4. El sistema muestra los campos modificables del formato:
 - La traduccin del nombre del formato que se ajuste mejor a la localizacin activa por defecto en el entorno del usuario.
 - La orden usada para invocar al editor preferido del usuario para dicho formato.
 - La orden usada para invocar al conversor a XML preferido del usuario para dicho formato.
 - La lista de extensiones, separadas por comas, con las que se asociar a este formato.
5. El usuario modifica el valor de los campos.
6. El sistema solicita confirmacin al usuario.
7. El usuario proporciona la confirmacin.
8. El sistema registra los cambios a nivel del usuario actual, sin modificar los ajustes globales, y los hace efectivos a partir del mismo momento.

Variaciones

2-7a. El usuario cancela la personalizacin:

1. El sistema cancela el caso de uso, sin que se hagan efectivos los campos.

4. Desarrollo del proyecto

- 7a. Los campos de nombre del formato, orden de edicin o extensiones se hallan vacos o slo contienen espacios en blanco, alguna de las extensiones pasadas contiene espacios o est vaca, o la orden de importacin contiene solamente uno o ms espacios en blanco:

1. El sistema informa de la situacin y solicita corregir dicha situacin.

Buscar nodo

Actor principal Usuario: desea localizar rpidamente los nodos que cumplan una serie de condiciones.

Precondiciones Hay un documento abierto.

Postcondiciones Se muestra el resultado de la bsqueda.

Escenario principal

1. El usuario indica su intencin de iniciar una nueva bsqueda.
2. El sistema solicita la clave de bsqueda.
3. El usuario introduce la clave de bsqueda.
4. El sistema comprueba que la clave de bsqueda es vlida.
5. El sistema presenta al usuario las opciones de filtrado.
6. El usuario elige las opciones de filtrado.
7. El sistema realiza la bsqueda a partir del nodo actual y muestra el primer resultado.

Variaciones

1-6a. El usuario cancela la bsqueda:

1. El sistema cancela el caso de uso.

4a. La clave no es vlida:

1. El sistema indica el error y pide una nueva clave.

7a. No hay resultados:

1. *include(Reintentar bsqueda)*

Repetir bsqueda

Actor principal Usuario: desea encontrar otro nodo que cumpla las mismas condiciones que el anterior, sin tener que volver a especificar las opciones de filtrado.

Precondiciones Hay un documento abierto.

Postcondiciones Se muestra el siguiente nodo que cumple las condiciones de la ltima bsqueda.

Escenario principal

1. El usuario desea repetir la bsqueda anterior:
2. El sistema comprueba que existe una bsqueda anterior.
3. El sistema realiza de nuevo la bsqueda a partir del nodo actual y muestra el primer resultado.

Variaciones

- 2a. No existen bsquedas anteriores:
 1. El sistema indica el error y cancela el caso de uso.
- 3a. No hay resultados:
 1. *include(Reintentar bsqueda)*

Reintentar bsqueda

Actor principal Usuario: desea visualizar un nodo que cumpla las condiciones, aunque tenga que relajar el alcance de la bsqueda.

Nivel de caso de uso Subfuncin.

Precondiciones Hay un documento abierto.

Postcondiciones Se muestra el primer nodo que cumple las nuevas condiciones relajadas.

Escenario principal

1. El sistema indica la situacin y ofrece repetir la bsqueda con parmetros ms generales.
2. El usuario acepta.
3. El sistema relanza la bsqueda y muestra el primer resultado.

Variaciones

4. Desarrollo del proyecto

2a. El usuario rechaza la oferta:

1. El sistema cancela el caso de uso.

3a. No hay resultados:

1. El sistema indica la situación y cancela el caso de uso.

Ver nodo

Actor principal Usuario: desea visualizar la información acerca de un nodo del documento.

Precondiciones Hay un documento abierto.

Postcondiciones Se muestra la información del nodo seleccionado.

Escenario principal

1. El usuario indica el nodo del documento que desea ver.
2. El sistema comprueba que existe.
3. El sistema muestra la información del nodo al usuario, dejando que interactúe con él.

Variaciones

2a. El nodo no existe:

1. El sistema muestra una visualización vacía.

3a. El usuario activa uno de los hipervínculos disponibles:

1. *include(Activar enlace)*

Activar enlace

Actor principal Usuario: desea visualizar el recurso al que señala el enlace activado.

Nivel de caso de uso Subfunción.

Precondiciones Hay un documento abierto, y se está visualizando un nodo.

Postcondiciones Se muestra el destino del enlace pulsado, si tiene uno definido.

Escenario principal

1. El usuario activa el enlace en cuestión de la visualización del nodo actual.

2. Si el enlace seala a otro nodo, se ejecuta el caso de uso “Ver nodo” sobre l, y se termina el caso de uso.
3. Si el enlace seala a una URL (Uniform Resource Locator), se muestra el documento en uno de los navegadores Web disponibles, y se termina el caso de uso.
4. Si el enlace seala a un ancla de la visualizacin HTML del nodo actual, se desplaza la posicin actual a la del ancla, y se termina el caso de uso.
5. Si el enlace seala a un nodo de otro documento, se visualiza dicho documento mediante “Visualizar documento”, posteriormente se ejecuta “Ver nodo” sobre el nodo especificado, y se termina el caso de uso.
6. El sistema cancela el caso de uso.

Cambiar hoja de preprocesado

Actor principal Usuario: desea cambiar la forma en que se muestra la estructura del documento.

Precondiciones Existe al menos un documento abierto y seleccionado.

Postcondiciones Se le muestra al usuario la nueva estructura del documento, procesada desde el texto XML original a travs de la hoja seleccionada. Se usar esta hoja para todos los ficheros que se abran a continuacin.

Escenario principal

1. El usuario indica su intencin de cambiar la hoja de preprocesado en uso.
2. El sistema muestra una lista de las hojas de preprocesado disponibles.
3. El usuario selecciona una hoja de la lista.
4. El sistema recupera el texto XML original del documento y lo procesa mediante la nueva hoja seleccionada.
5. El sistema muestra la nueva estructura del documento, y vaca la visualizacin del nodo actual, al no haber ninguno seleccionado.
6. El sistema registra la hoja seleccionada como hoja de preprocesado por defecto para todos los documentos abiertos en adelante.

Variaciones

1-3a. El usuario cancela la seleccin:

1. El sistema cancela la ejecucin del caso de uso.

4. Desarrollo del proyecto

Cambiar hoja de visualizacin

Actor principal Usuario: desea cambiar la forma en que se muestra la informacin de los nodos de un documento.

Precondiciones Existe al menos un documento abierto y seleccionado.

Postcondiciones Se visualiza el nodo actual y todos los posteriormente seleccionados en el documento actual y en todos los documentos abiertos posteriormente bajo la nueva hoja.

Escenario principal

1. El usuario indica su intencin de cambiar la hoja de visualizacin para el documento actual.
2. El sistema muestra la lista de las hojas de visualizacin disponibles.
3. El usuario selecciona una de las hojas de la lista.
4. El sistema actualiza la visualizacin del nodo actualmente seleccionado, si lo hay.
5. El sistema registra la hoja seleccionada como hoja de visualizacin por defecto para los documentos abiertos de ahora en adelante.

Variaciones

1-3a. El usuario cancela la seleccin:

1. El sistema cancela la ejecucin del caso de uso.

4.4.3. Modelo conceptual de datos del dominio de ACL2

Notacin

Se ha usado UML 2.0, con las siguientes modificaciones:

- Por claridad, algunas clases se hallan duplicadas en los diagramas. Para evitar confusiones, las duplicaciones ocultan los atributos y se hallan en color azul.
- Igualmente, se han omitido las multiplicidades unitarias en las asociaciones y composiciones.

Descripcin general

Para `ACL2::Procesador`, los conceptos a tratar son entes abstractos relacionados con el proceso de demostracin de ACL2.

As, partimos de un *Enunciado*, coleccin de *Ordenes* de ACL2 que produce una *Demostracin*. Esta *Demostracin* asocia a cada *Orden* un *Resultado* con la salida obtenida de ACL2. Esta contendr diversos tipos de informacin segn la *Orden* usada.

Puede que nuestro *Enunciado* sea la raz de un proyecto de ACL2, y se halle definido en base a una serie de definiciones ya demostradas en *Libros* existentes. Es importante ver que un *Libro* puede requerir de una serie de definiciones previas a su vez: segn la terminologa de ACL2, sera su *mundo inicial*. Todo *Evento* pertenece a un *Paquete*: si no se indica su nombre previamente con una orden `in-package`, ser “ACL2”.

Adicionalmente, pueden aparecer avisos, errores y observaciones de ACL2 al inicio del resultado de la ejecucin de una *Orden* dada.

A su vez, cada *Resultado*, en caso de necesitar demostrar alguna S-expresin, usar una *Meta* principal, que se podr dividir recursivamente en submetas. Cada *Meta* usa un determinado *Proceso* de demostracin.

Se ha dividido el diagrama de clases conceptuales en tres:

1. Visin general del dominio del problema: figura 4.6 de la pgina 82.
2. Tipos de Orden: figura 4.7 de la pgina 83.

Es interesante ver cmo algunas rdenes utilizan a otras. Por ejemplo, como paso de verificacin tras la certificacin de un libro con `certify-book`, se intentan cargar sus contenidos en el mundo de ACL2, justo como si se hubiera ejecutado una orden `include-book`.

3. Tipos de Proceso: figura 4.8 de la pgina 84.

Restricciones textuales

1. *Clave externa*: “nombre” de *Documento*.
2. *Clave externa*: “nombre” de *Paquete*.
3. *Clave externa*: “nombre” de cualquier descendiente de *Evento*, si lo tiene.
4. *Clave externa*: “clavedoc” de cualquier descendiente de *Evento*, si la tiene y est especificada.
5. Todo *Resultado* pertenece o a otro *Resultado* o a una *Demostracin*.
6. Todo *Meta* pertenece o a otra *Meta* o a un *Resultado*.

4. Desarrollo del proyecto

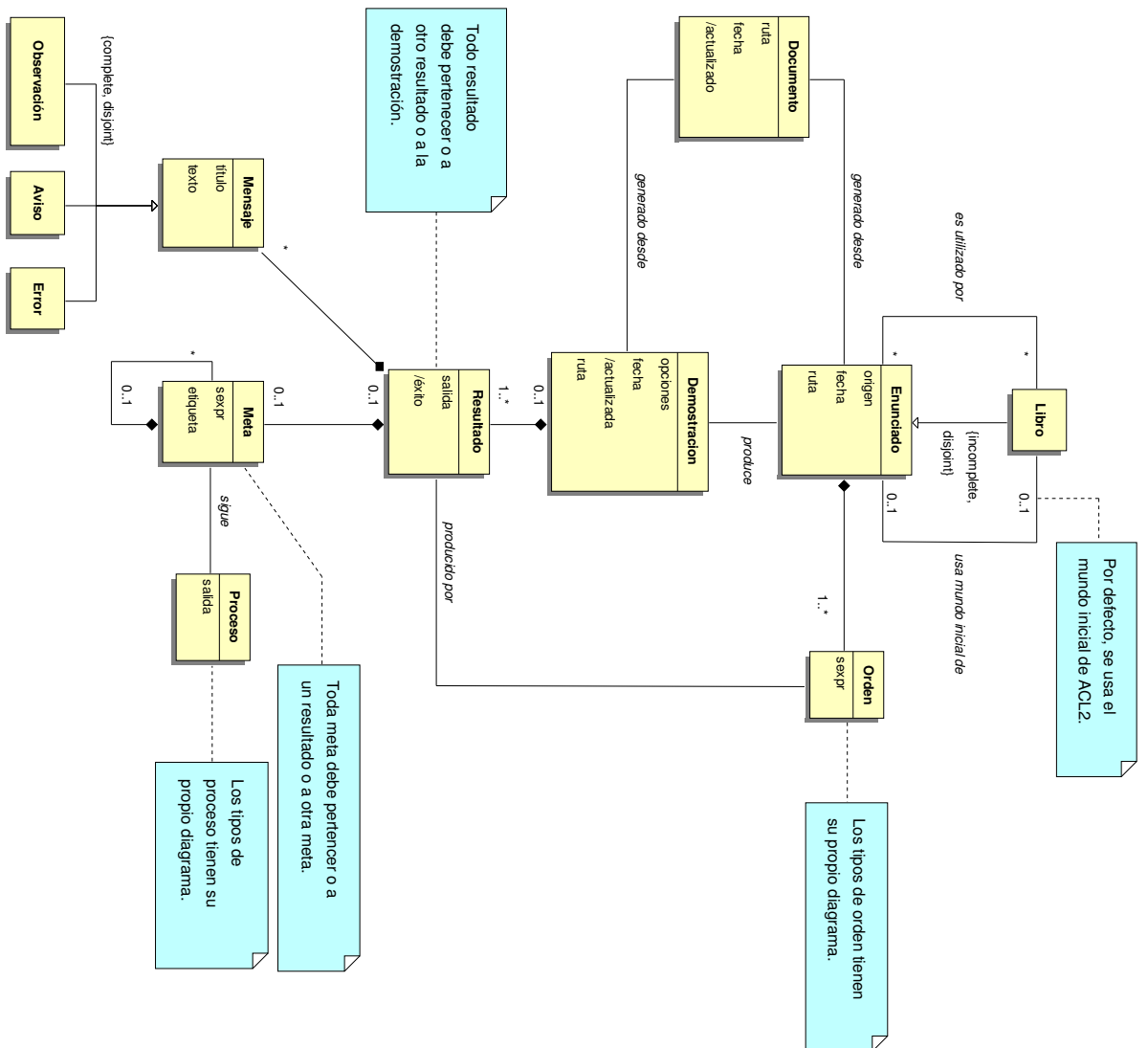


Figura 4.6.: Diagrama de clases conceptuales general de ACL2

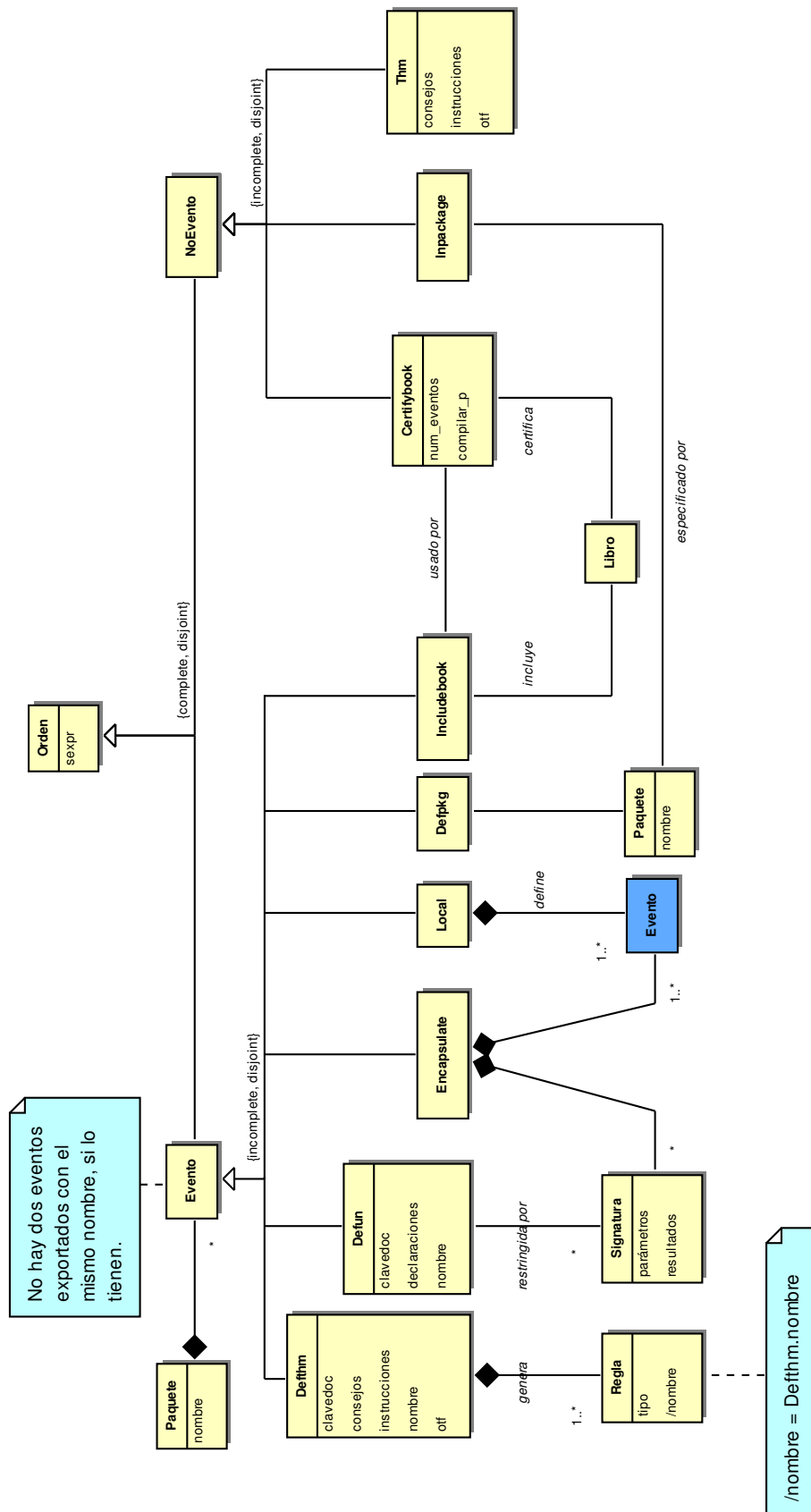


Figura 4.7.: Diagrama de clases conceptuales de rdenes de ACL2

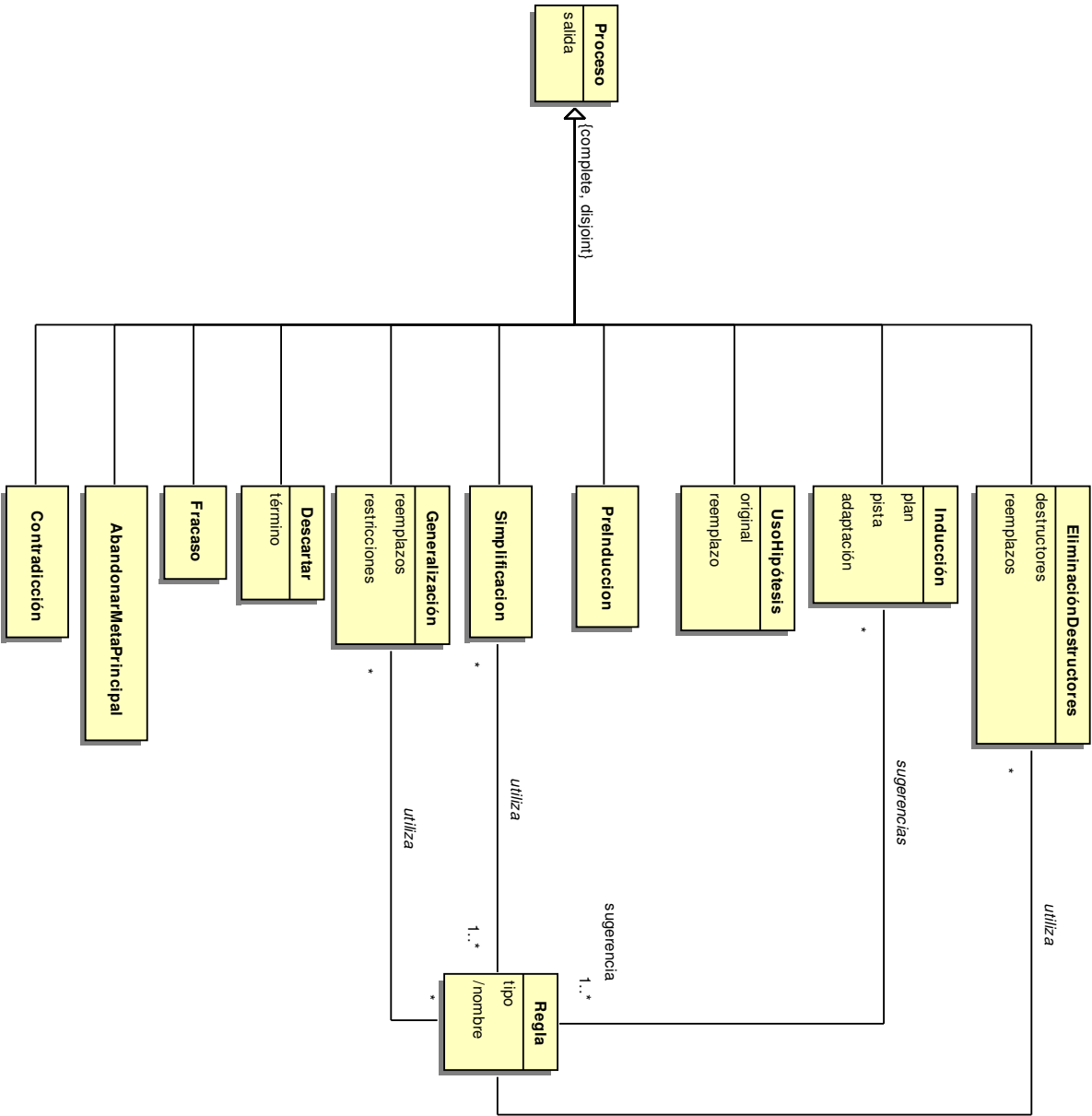


Figura 4.8.: Diagrama de clases conceptuales de procesos de ACL2

7. Toda *Meta* tiene una etiqueta nica dentro de una *Demostracin*.
8. *Resultado.xito* $\in \{\text{verdadero, falso}\}$.
9. *Documento.actualizado* $\in \{\text{verdadero, falso}\}$.
10. *Demostracin.actualizada* $\in \{\text{verdadero, falso}\}$.

Derivaciones

1. *Demostracin.actualizada* = “verdadero” si *Demostracin.fecha* es una fecha y hora posterior a *Enunciado.fecha*. “falso” de lo contrario.
2. *Documento.actualizado* = “verdadero” si *Demostracin.actualizada* y *Documento.fecha* es una fecha y hora posterior a *Demostracin.fecha*. “falso” de lo contrario.
3. *Resultado.xito* = no contiene *Meta* con el *Proceso Fracaso* y sus *Resultados* de inferior nivel (si los hay) tienen xito.
4. *Regla.nombre* = *Defthm.nombre*.

4.4.4. Modelo conceptual de datos del dominio de YAML

Notacin

Se seguir la misma notacin que en §4.4.3 (pgina 80).

Secuencia de procesado

Toda implementacin de YAML establece una correspondencia entre las estructuras de datos nativas y los flujos de caracteres Unicode a travs de una transformacin compuesta por tres pasos en cada direccin, tal y como puede verse en la figura 4.9 de la pgina 87. Los detalles de cada paso son invisibles a los dems. Segn en qu direccin nos desplazemos, hablaremos de:

Volcado Tomaremos una estructura de datos nativa del lenguaje de programacin y la presentaremos en forma de una sucesin de bytes dentro de un flujo. En la mayora de las implementaciones, esta operacin recibe el nombre de *Dump*. Seguiremos estos pasos:

1. *Representacin* de la estructura de datos nativa en forma de un grafo dirigido con raz, siguiendo el modelo de informacin de representacin de YAML que describiremos posteriormente.

4. Desarrollo del proyecto

2. *Serializacin* del grafo a un rbol de serializacin. Habr que imponer un orden sobre los nodos del grafo, y evitar los problemas asociados con representar nodos con mltiples arcos entrantes y caminos con ciclos en un rbol. Parte de estos detalles tambin aparecen en nuestro modelo conceptual de datos.
3. *Presentacin* en forma de una sucesin de bytes de la informacin del rbol, recorriendolo para lanzar y despus capturar una serie de eventos de serializacin. Se emplearn diversos estilos y notaciones para generar resultados fcilmente legibles por humanos.

Carga Este es el proceso inverso al volcado, y se suele encontrar bajo el nombre de `Load`. Cada paso va descartando los detalles especficos al paso correspondiente del volcado. Los pasos a seguir son:

1. *Anlisis* de los bytes del flujo, lanzando y capturando los eventos de deserializacin correspondientes para producir el rbol de serializacin que refleje la informacin original.
2. *Composicin* del grafo de nodos a partir del rbol, restableciendo los enlaces originales entre los nodos.
3. *Construccin* de las estructuras de datos nativas del lenguaje a partir del grafo de nodos.

Modelos de informacin

Normalmente, las aplicaciones slo necesitan las estructuras de datos finales, que se corresponden en muchos lenguajes (Perl, por ejemplo, o Java, en menor grado) de manera bastante fiel a los grafos del modelo de informacin de representacin. `YAXML: -Reverse`, sin embargo, tiene unos requisitos algo especiales: ha de asegurar una correspondencia entre los grafos de YAML y los rboles de XML, por lo que ha de operar a un nivel menor de abstraccin, utilizando el modelo de informacin de serializacin descrito en [?]. Una adaptacin a UML de dicho modelo se halla en la figura 4.10 de la pgina 88.

Explicaremos primero los elementos del modelo de informacin de representacin de YAML, y luego describiremos las modificaciones que introduce el modelo de informacin de serializacin.

Modelo de informacin de representacin Un documento YAML se halla representado por el *Nodo* raz del grafo. Existen varias categoras (*kind* en el original) de *Nodos*: *Escales* que contienen una cadena opaca de caracteres Unicode, *Secuencias* ordenadas de nodos, y vectores asociativos (objetos de la clase *VectorAsociativo*) que contienen un conjunto no ordenado de pares clave-valor (*ParClaveValor*). Es interesante ver que

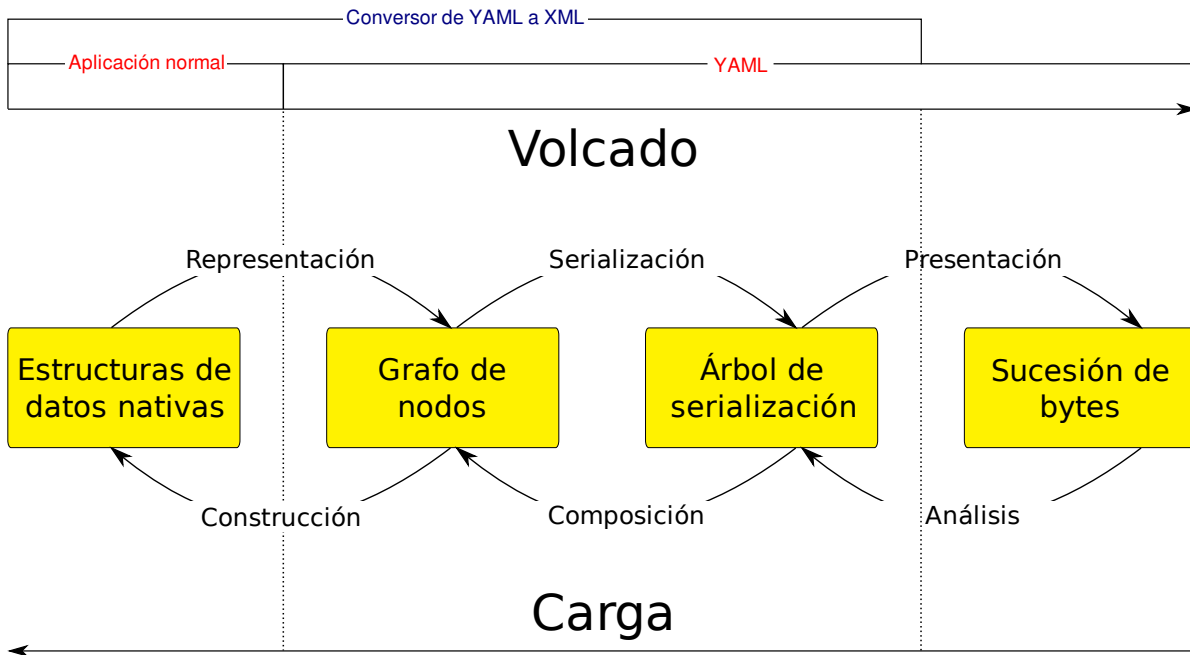


Figura 4.9.: Secuencia de procesamiento de YAML

tanto la clave como el valor pueden ser nodos arbitrariamente complejos, pudiendo referirse a cualquier otro nodo del grafo.

Todo nodo dispone de una etiqueta (*tag* en el original) que identifica el tipo abstracto de su contenido. Las etiquetas sirven también en el caso de los escalares para unificar cadenas con el mismo significado a una serie de formas canónicas. Por ejemplo, las reglas de la etiqueta `tag:yaml.org,2002:int` utilizan base 10 y un signo negativo opcional en sus formas canónicas: un valor hexadecimal como `0xA` será convertido por cualquier analizador al valor decimal 10.

Modelo de información de serialización Modifica el modelo de representación en dos aspectos:

1. Aade un orden a los pares clave-valor de los vectores asociativos. Este orden es un detalle de serialización, y no tiene impacto alguno en los grafos y estructuras de datos nativos generados.
2. Permite asociar a cualquier nodo un identificador o *ancla*, y usar nodos *alias* sealando a esa ancla en posteriores referencias a ese nodo. Esto permite crear presentaciones compactas de grafos complejos, que podrán incluso contener ciclos en su interior.

Un ancla no tiene por qué ser única: el nodo al que hace referencia un *alias* no es más que el último que definió esa ancla en el orden establecido.

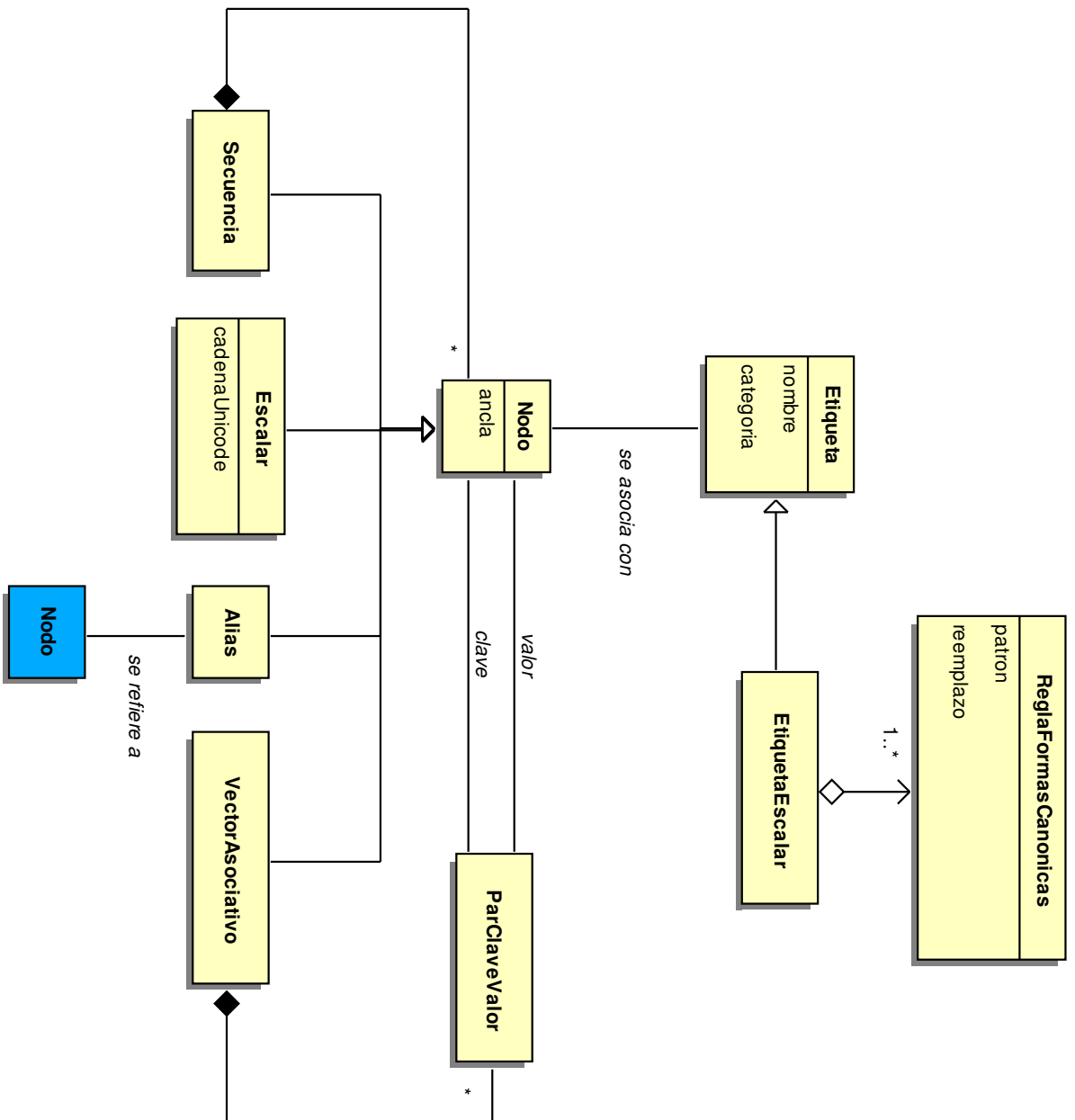


Figura 4.10.: Diagrama de clases conceptuales para YAML.

Restricciones textuales

- *Clave externa*: “nombre” de Etiqueta.

4.5. Diseo del sistema

4.5.1. Arquitectura del sistema

Para poder decidir la arquitectura del sistema, primero habra que definir cules son las condiciones que afectarn a su estructura.

Dichas condiciones se basan tanto en la funcionalidad esperada del sistema, como en los requisitos no funcionales antes mencionados y otros factores de calidad del software, como la flexibilidad, mantenibilidad, reusabilidad o fiabilidad, entre otros.

La mantenibilidad requiere una arquitectura que evite la propagacin de cambios realizados sobre una parte del sistema, definiendo interfaces estables entre estas partes. Dicha arquitectura debe ser adems lo ms sencilla posible, para facilitar cambios en su estructura y elementos.

La divisin en partes con lmites e interfaces bien definidas mejora tambin la reusabilidad y facilita las pruebas, evitando acoplamientos innecesariamente complejos que dificulten el desarrollo de una aplicacin fiable y con la funcionalidad deseada.

Separacin de responsabilidades: divisin en capas

En el caso de este proyecto, se deseaba separar XMLEye de los detalles de los conversores externos, como `ACL2::Procesador` y `YAXML::Reverse`, y viceversa. En un futuro, se implementarn nuevas interfaces de usuario para XMLEye, por lo que tambin habr que separar la lgica de aplicacin y de presentacin de XMLEye.

Bajo este contexto, se puede ver que la aplicacin del patrón arquitectónico Capas (descrito con mayor detalle en [?]) es natural. Se ha dividido la aplicacin en cuatro capas:

Presentacin Se ocupa de la interacción persona-máquina en general y en particular de la visualización de la información y la navegación a través de esta. Define las transformaciones a realizar sobre las representaciones creadas en la capa de filtrado.

4. Desarrollo del proyecto

Aplicacin Responde a las peticiones de la capa superior, ejecutndolas de forma independiente a la interfaz escogida. Implementa la infraestructura necesaria para la ejecucin de las transformaciones de la capa de usuario, posibilitando su seleccin en tiempo de ejecucin, enlazado de nodos y localizacin de cadenas. Incorpora la lgica necesaria para la integracin con los conversores de la capa de filtrado.

Filtrado A diferencia de la usual capa de dominio en un modelo de tres capas, no manipula instancias de los objetos del dominio, sino que los transforma a una representacin manejable por las capas superiores, de ah su nombre.

Se halla constituida por una serie de conversores independientes entre s y de XMLEye, como `ACL2::Procesador` o `YAXML::Reverse`. XMLEye slo ha de conocer la orden a ejecutar, que ha de seguir unas pautas muy sencillas:

- El cdigo de retorno de la orden debe indicar el xito o fracaso en la conversin.
- El resultado XML ha de volcarse por la salida estndar, y el estado de la conversin por la salida estndar de errores.

Servicios tcnicos En esta capa hemos situado todas las bibliotecas, mdulos externos y dems que proporcionan parte de la funcionalidad comn del sistema, y son internos a ste.

Se ha detallado la visin en capas (modeladas por paquetes UML) a lo largo del eje vertical y en particiones a lo largo del eje horizontal en la figura 4.11 de la pgina 91.

Ms tarde examinaremos cada una de estas capas por separado.

Transformaciones configurables: tuberías y filtros

Descripcin Otro patrón arquitectónico usado (con ciertas licencias) en el diseño de este proyecto fue Tuberías y Filtros. Dicho patrón es familiar a todo usuario de un sistema UNIX, de donde se inspiró precisamente.

Consiste en el encadenamiento de una serie de Filtros, partiendo de una Fuente de informacin, y llegando hasta un Sumidero, destino de la informacin transformada. A cada paso, la informacin llega a un Filtro a través de una Tubería, saliendo por otra Tubería hacia el siguiente Filtro o el Sumidero final.

El patrón original (ver [?]) tiene las ventajas de ser fácilmente paralelizable en filtros incrementales que no necesiten leer toda la entrada antes de poder operar, y permitir una gran flexibilidad cambiando el filtro exacto usado a cada paso, dado que están diseñados para ser intercambiables entre sí, con ciertas limitaciones.

Sus desventajas consisten en la dificultad de mantener un estado compartido y de realizar un control detallado de errores, al ser usualmente tratada toda la transformación como una unidad.

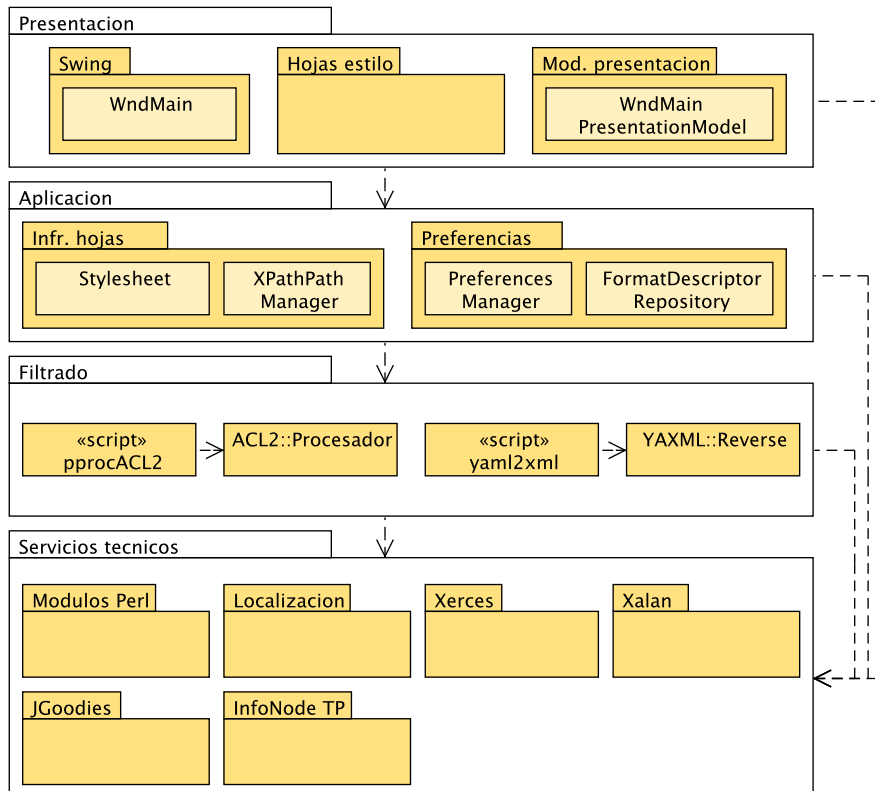


Figura 4.11.: Diagrama arquitectónico del sistema

4. Desarrollo del proyecto

Uso El uso de este patrón a lo largo de este proyecto se fundamenta en la necesidad de permitir la extensibilidad de la visualización por parte del usuario sin necesidad de una recompilación, usando un enfoque puramente dirigido por datos.

Así, en el caso de una demostración de ACL2, si consideramos al sistema externo ACL2 como la fuente de la información y a la visualización por parte de la biblioteca Swing como el sumidero, tendremos cuatro filtros intermedios:

1. `ACL2::Procesador`, con el enunciado Lisp para ACL2 como entrada, genera una representación en XML del contenido de la demostración realizada. El formato de salida se conoce a través de una DTD, incrustada en el propio documento de salida.
2. El segundo filtro, definido en la capa de presentación, realiza una transformación previa, potencialmente costosa, que afecta a todo el documento, y que sólo se realiza una vez.

Aquí el usuario puede definir el aspecto que tendrá el árbol del documento, ocultando nodos o cambiando la etiqueta o icono a mostrar. La descripción de la hoja define las entradas esperadas y el formato de la salida.

3. El tercer filtro, también en la capa de presentación, transforma un nodo a su visualización XHTML utilizando una hoja XSLT para el formato de la salida. La transformación se repite cada vez que el usuario cambia de nodo.

En este caso, se sabe que la salida es XHTML, y que sigue por lo tanto una DTD conocida. También podemos deducir a partir del nombre de la hoja que entrada espera y qué salida produce.

Así, sabemos que la hoja `ppACL2` de visualización tomará la salida de la hoja de preprocesado `ppACL2` o una derivada suya y mostrará la información relevante a demostraciones de ACL2 contenida en dicho documento.

Una hoja de visualización es diseñada suponiendo que la salida del preprocesado sigue el formato de salida de ciertas hojas de usuario de preprocesado. En el caso contrario, la transformación no fallará, pero no se obtendrán los resultados esperados.

4. El cuarto filtro, controlado por la capa de aplicación, marca las coincidencias de la clave de la búsqueda en curso en el texto, si se da el caso. También se aplica una hoja de estilos CSS, que no modifica la estructura pero sí el aspecto del resultado final.

Como vemos, los dos filtros intermedios se ocupan de la mayor parte de la visualización, y según los requisitos funcionales, estos deberán ser modificables por el usuario sin necesidad de recompilación.

Implementacin Se eligi XSLT para implementar estos dos filtros por la flexibilidad que ofrece al estar basado en ficheros de texto fcilmente intercambiables y por su capacidad de describir declarativamente las transformaciones sobre la fuente XML a otros formatos.

Se opt por una visualizacin basada en XHTML para evitar el uso de numerosos componentes discretos Swing, complicando el diseo de la interfaz y las extensiones por parte del usuario. El uso de hipertexto permite, adems de ofrecer mayor informacin visual al usuario, establecer enlaces entre los distintos elementos de la demostracin.

Existen varias diferencias con el patrón original. Los filtros aquí usados no son incrementales (el uso de XML no lo permite), perdiéndose la capacidad de paralelización.

Por otro lado, la pérdida de capacidad de detección detallada de errores no aparece en nuestro caso, al basarse las tuberías en lógica de la capa de aplicación y no en otros mecanismos de menor nivel de abstracción, como memoria compartida.

No ha sido necesario ningún estado compartido, por lo que esta otra desventaja tampoco ha resultado ser ninguna limitación.

En cuanto a rendimiento, hay que considerar el uso de *translets* por el procesador XSLT Xalan XSLTC. Estos son representaciones compiladas a bytecode Java de hojas XSLT cuya ejecución es notablemente más rápida que la ejecución interpretada clásica, a costa de cierta funcionalidad innecesaria para este proyecto.

La disposición e interconexión de los componentes que intervienen en la transformación se ha recogido en el diagrama UML de componentes de la figura 4.12 de la página 94.

La capa de aplicación no aparece en él, pero se halla implícita, como capa de control. Tampoco han de confundirse las hojas XSLT usadas, que pertenecen a la capa de presentación, con su lógica de control, parte de la capa de aplicación.

Integración de las capas de aplicación y presentación: patrones MVP

En el visor del Proyecto que precede a este, se había establecido una comunicación entre las capas de aplicación y de presentación mediante Fachadas. Estas Fachadas constituyen el único punto de acceso a la funcionalidad de cada capa, y evitaban que la capa superior tuviera que conocer los detalles de la capa inferior.

Aunque para sistemas con interfaces más sencillos se podría haber continuado su uso, en el caso de XMLEye se vio cómo las Fachadas iban tomando demasiadas responsabilidades al mismo tiempo. También se dio el caso de que la capa inferior forzaba a la capa superior a una serie de restricciones, siendo la más grave la de tratar un único documento.

4. Desarrollo del proyecto

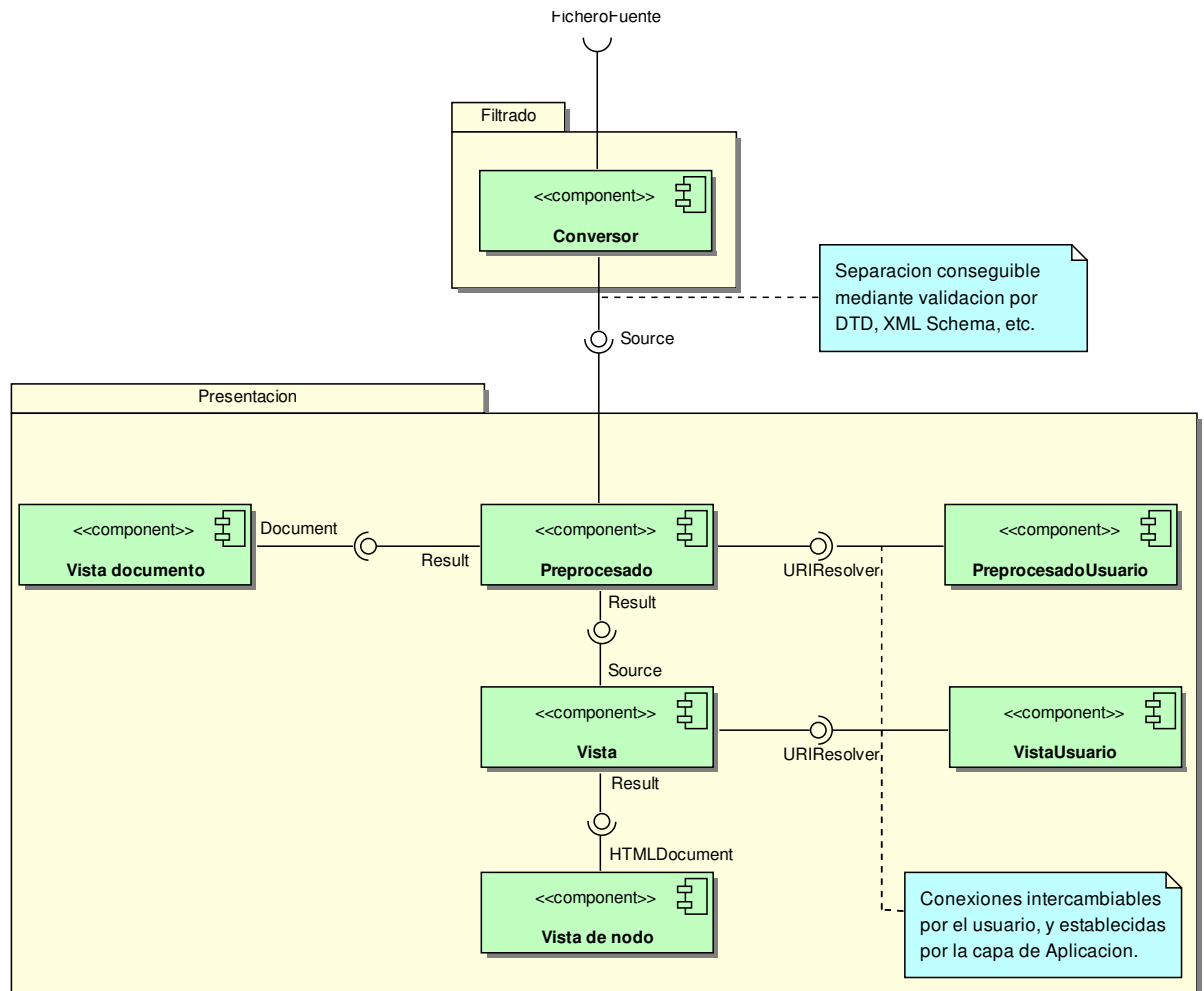


Figura 4.12.: Diagrama de componentes para visualización

Otro problema que aquejaba al antiguo diseo es la prcticamente nula capacidad para poder realizar pruebas de unidad de forma cmoda en la aplicacin. Existen herramientas que permiten automatizar pulsaciones y entrada de teclado, pero las pruebas de este tipo son extremadamente frgiles: cualquier cambio en la disposicin de los controles de la interfaz las invalidara.

Para resolver ambos problemas, se estudiaron las diversas alternativas ofrecidas en [?], y en particular las derivaciones del patrón MVP (Model View Presenter). Este patrón es una derivación del MVC (Model View Controller) [?] original en el que en vez de tener un tro modelo-vista-controlador para cada control de la interfaz (como un campo de texto, por ejemplo), tenemos un tro para el formulario completo. Ahora el modelo del formulario es el conjunto de clases del dominio usadas, la vista es la estructura formada por todos los controles grficos del formulario, y el presentador es el componente al que se notifican todas las acciones de inters realizadas por el usuario.

El presentador es el ocupado de modificar el modelo, y posteriormente la vista se actualizar con los cambios hechos por el presentador al modelo. Hay diversos mecanismos para garantizar esta sincronización, constituyendo variantes distintas del patrón MVP bsico:

Controlador Supervisor Para los casos simples, la sincronización modelo-vista se establece a partir del patrón Observador o algún otro enfoque declarativo. En casos ms complejos, el presentador interviene directamente sobre los controles de la interfaz.

Vista Pasiva La vista en este caso carece de cualquier lgica de sincronización, y es el presentador el que rellena todos los campos. Una posibilidad para evitar acoplar demasiado el presentador a los controles usados es definir una interfaz en el formulario para su relleno: de esta forma, durante las pruebas de unidad podemos sustituir la vista por un Doble para Pruebas [?] y depurar prcticamente toda la funcionalidad.

Modelo de Presentación El presentador contiene el estado abstracto que debera presentar el formulario. Puede que sea el modelo de presentación el que manipule la vista, o que sea la vista la que se actualice a partir del modelo de presentación. En el primer caso, ambos elementos estn ms acoplados pero podemos probar la sincronización. En el segundo caso, la sincronización no se puede depurar tan fcilmente, pero el modelo de presentación es completamente independiente de la interfaz usada.

De entre estos enfoques, se escogi el Modelo de Presentación, ya que en un futuro se desean crear nuevas interfaces para XMLEye: el presentador no debera de saber nada acerca de la vista. Por la misma razón, es la vista la que se actualiza al estado encapsulado por el modelo de presentación.

4. Desarrollo del proyecto

El uso de un modelo de presentacin tambien facilita la implementacin de una interfaz multidocumento: cada documento puede imponer su propio estado de presentacin sin afectar a los dems.

En cuanto a la divisin en modelo-vista-presentador, tendremos a los modelos y otros servicios de alto nivel en la capa de aplicacin, y a las vistas y presentadores en la capa de presentacin.

4.5.2. Capa de filtrado: ACL2::Procesador

Volviendo al diagrama de capas de la figura 4.11 de la pgina 91, vemos que este conversor ha de:

- Convertir la salida de ACL2 a XML sin prdidas.
- Definir claramente el formato de dicha salida.

Por la necesidad de manipular el texto de la forma ms sencilla y potente posible, se escogi usar un guin escrito en Perl, un lenguaje diseado a este efecto.

A raz de la complejidad inherente en el manejo de la salida en lenguaje humano de un programa como ACL2, se ha seguido el paradigma orientado a objetos a la hora de desarrollar `ACL2::Procesador`.

Procesado general de proyectos de ACL2

El diseo bsico se fundamenta en el modelo conceptual de datos, con ciertas modificaciones. El guin lanzado por la capa de aplicacin emplea la funcin `convertir_a_xml` del mdulo `Procesador`, pasndole la ruta al *Enunciado* raz del proyecto completo, y vuelca sus resultados por la salida estndar.

Este mtodo es realmente un envoltorio que crea un *Proyecto* con raz en el *Enunciado* indicado, actualiza por completo de forma recursiva todas las salidas de ACL2 y sus versiones en XML y devuelve la versin XML de la salida de la raz del proyecto.

Las *Dependencias* del *Proyecto* son obtenidas a partir de un anlisis esttico del cdigo Lisp, empezando por el *Enunciado* Lisp raz y continuando de forma recursiva. Esto genera un rbol anidado de *Proyectos* que dependen unos de otros. Decimos rbol y no grafo pues un subproyecto puede aparecer varias veces en el mismo rbol. Esto supone un coste mayor en espacio, pero no en tiempo: si tuviera que ser actualizado, slo sera procesado una vez. Se estn considerando representaciones ms eficientes en espacio para versiones posteriores.

El proceso de actualizacin consiste en recorrer el rbol e ir actualizando de forma recursiva:

1. Se solicita la actualizacin de las *Dependencias* del proyecto actual, que forman *Proyectos* a su vez.
2. Si algn subproyecto fue actualizado o el fichero temporal con la salida de ACL2 no existe o tiene una fecha de modificacin ms antigua que la del *Enunciado*, se invocar a ACL2 para crear o actualizar dicho fichero temporal.
3. Si el fichero temporal con la salida fue actualizado o el fichero temporal con el documento XML resultante no existe o tiene una fecha de modificacin ms antigua que la de la salida, se actualizar empleando una instancia de la clase *Demostracin*, que encapsula el proceso de anlisis de una demostracin de ACL2.

El diagrama de secuencia correspondiente a esta lgica se halla en el cuadro 4.14 de la pgina 99. El diagrama con las clases participantes, integradas con las ocupadas de procesar las *Demostraciones* en s, forma el cuadro 4.13 de la pgina 98.

Generacin de modelos en memoria

Un concepto importante para comprender el diseo de `ACL2::Procesador` en esta versin es la necesidad cada vez mayor de atrasar el momento de la generacin de la salida XML. Originalmente, ambos estaban intercalados. Sin embargo, en muchos casos el orden en que se procesa la informacin no coincide con el orden en que se debe producir la salida, y esto supona un impedimento para el procesamiento de demostraciones cada vez ms complejas.

A lo largo de las iteraciones, se ha hecho una divisin mayor y mayor, hasta finalmente separar el procesado y la salida a nivel de una *Demostracin* completa, en vez de *Resultados* individuales. Esto permite dar nuevos usos al anlisis ya implementado. Puede verse esta separacin en el diagrama de secuencia de la figura 4.15 de la pgina 100: la inicializacin se hace slo durante la ejecucin del constructor `new`, y la salida se produce exclusivamente dentro del mtodo `escribir_xml`.

Por ejemplo, se puede analizar una *Demostracin* sin contar con su salida, realizando un anlisis esttico sobre el cdigo Lisp: toda *Orden* dispone de mtodos separados para recuperar informacin de la S-expresin y de la salida de ACL2, y el ltimo slo es usado cuando se proporciona dicha salida en el constructor. Este anlisis esttico proporciona informacin de todas las dependencias de un proyecto.

Relacin con capa de servicios tcnicos

Viendo el diagrama general de clases de diseo para la capa de filtrado, se puede observar que la mayora de las clases tienen dependencias con `XML::Writer` y `Localizacin`.

Esto es lo usual con los elementos de la capa de servicios tcnicos: `XML::Writer` es un mdulo usado para la salida XML, que incorpora ciertas comprobaciones automticas

4. Desarrollo del proyecto

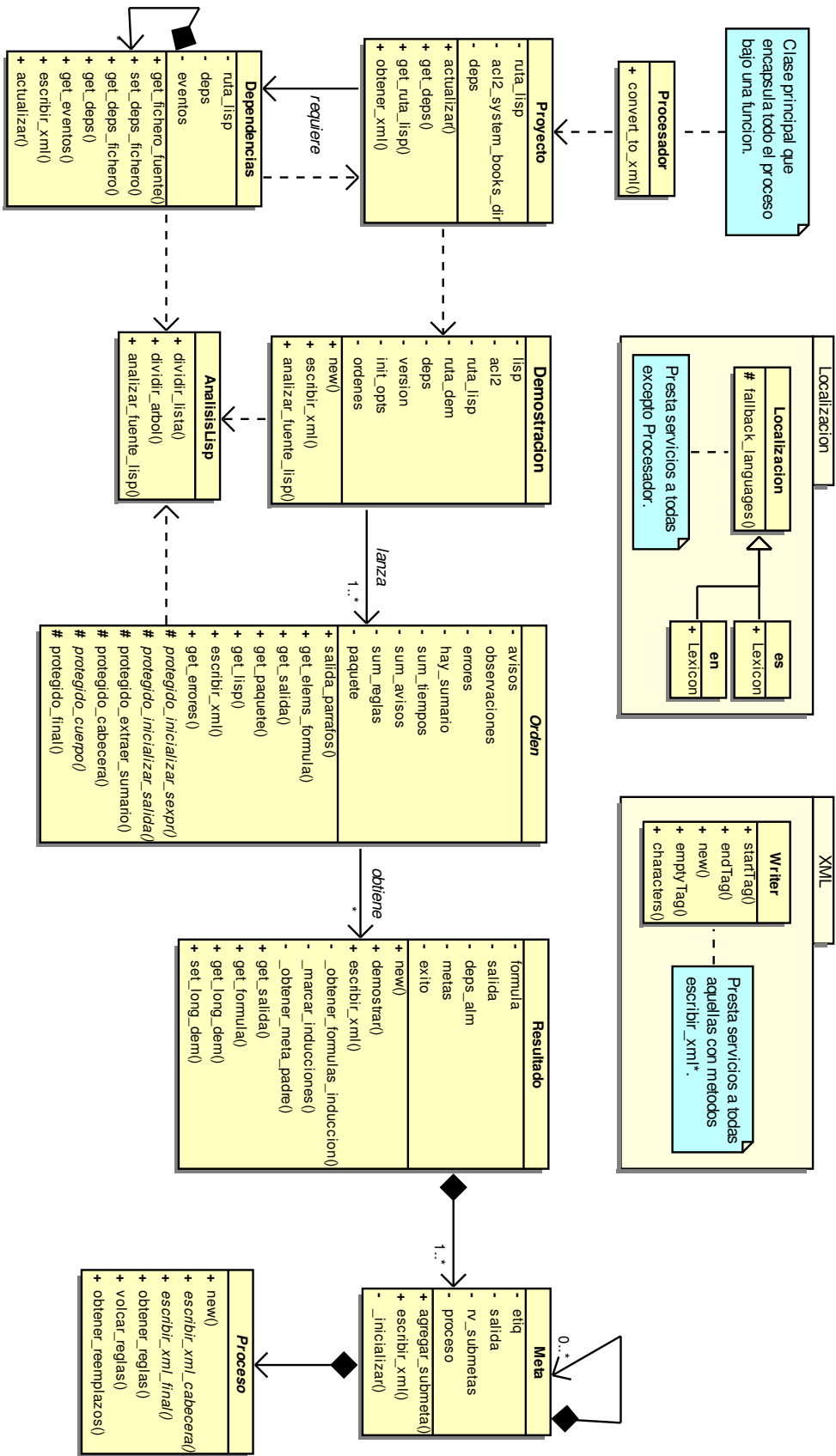


Figura 4.13.: Diagrama de clases de ACL2::Procesador (general)

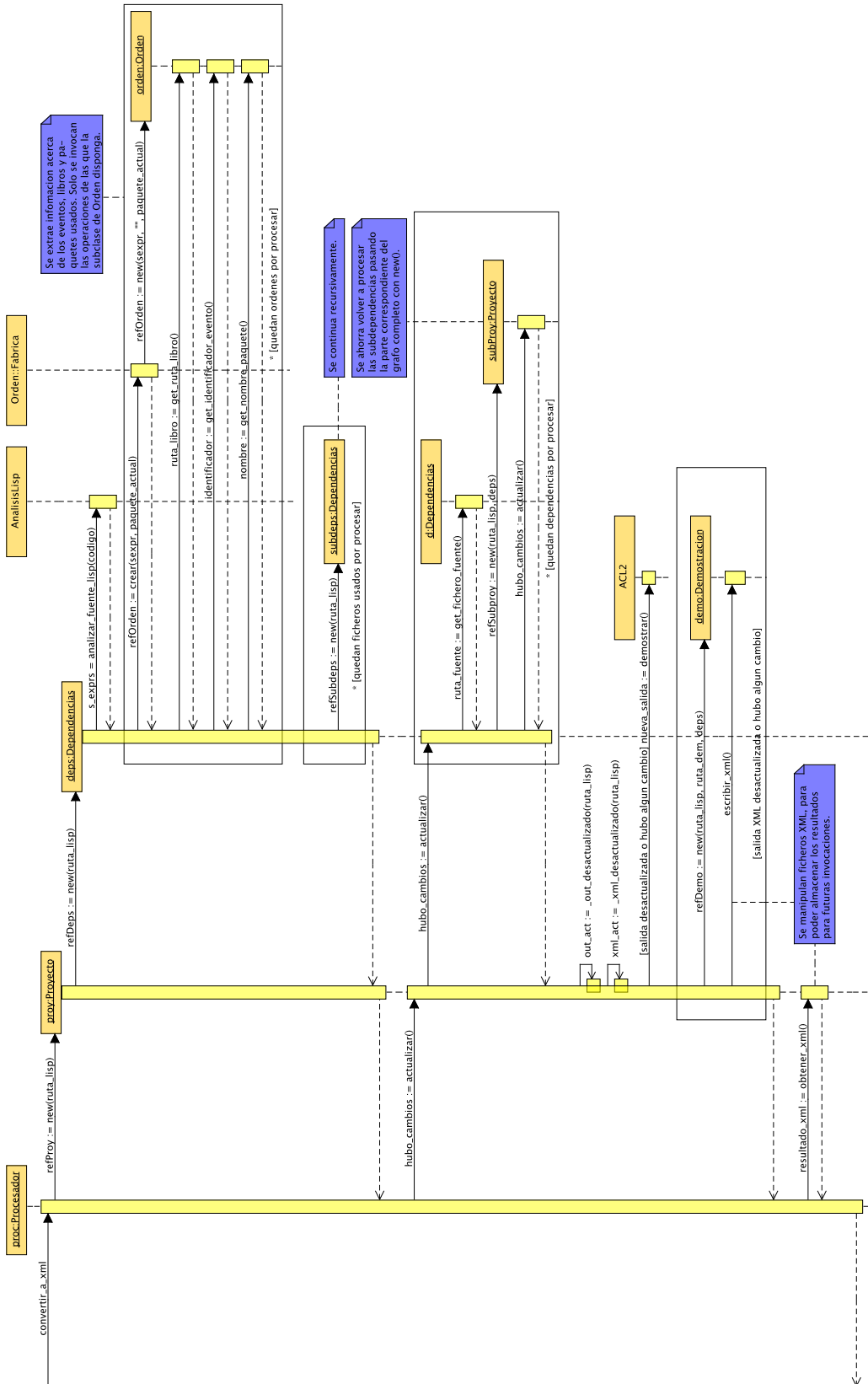


Figura 4.14.: Diagrama de secuencia general de filtrado

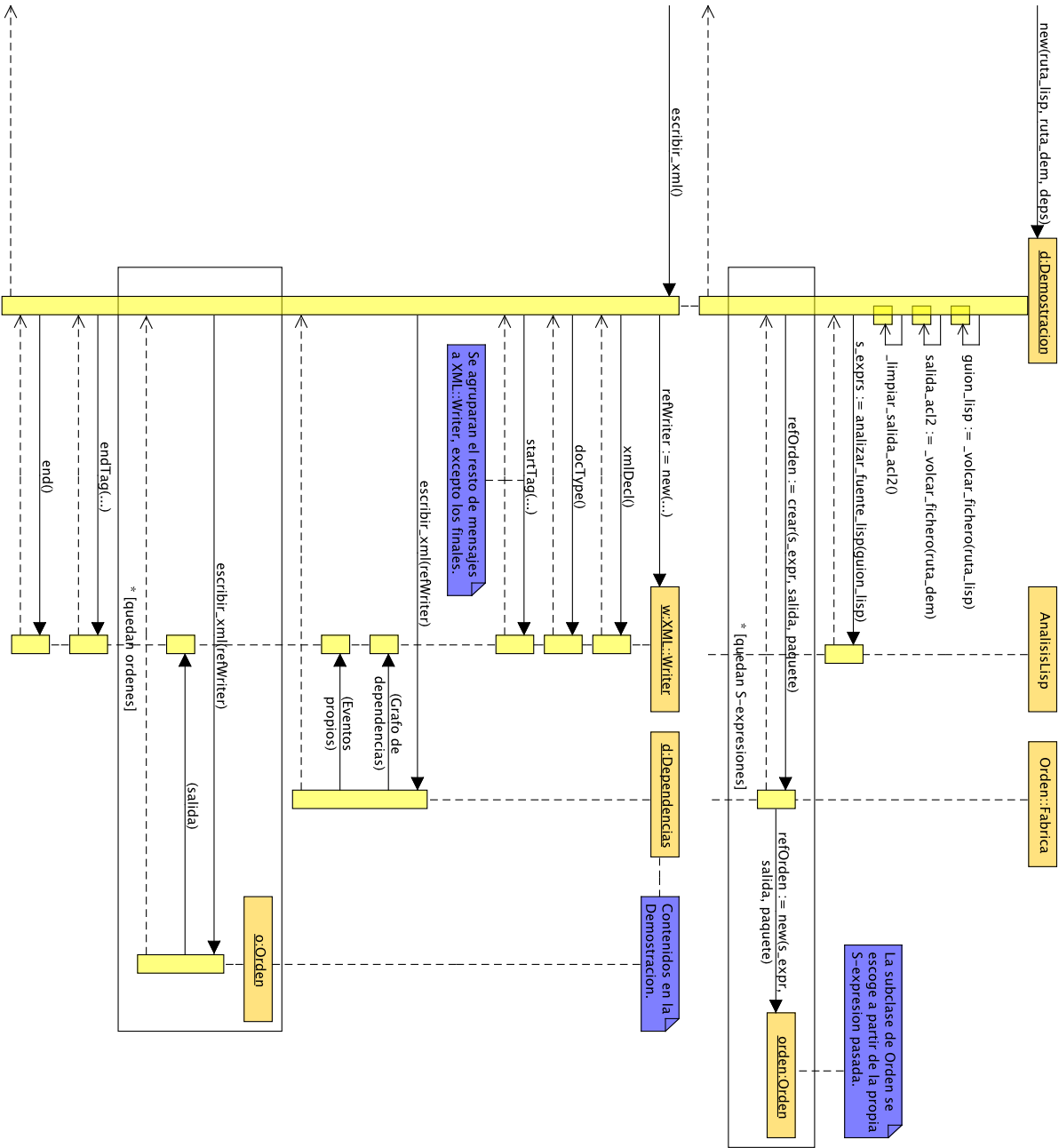


Figura 4.15.: Diagrama de secuencia de filtrado de una *Demostracin*

para garantizar la ejecucin de generacin de documentos bien formados, y *Localizacin* es el mdulo usado para localizacin de mensajes de error o avisos.

No hay que confundir dichos servicios tcnicos con clases de utilidad como *AnlisisLisp*, que asla la lgica necesaria para convertir listas y rboles Lisp a estructuras de datos equivalentes de Perl.

Reflexin y tipos basados en comportamiento: simplificacin del anlisis esttico

Al implementar el anlisis esttico necesario para conseguir el grafo de dependencias de un proyecto, surge la necesidad de conocer cul sera el identificador final y paquete de cada evento, y de las rutas de todos los libros. Evidentemente, hay que distinguir entre los *Eventos* con nombre, los *Eventos* sin nombre y los *NoEventos*, que no producen cambios en el mundo lgico de ACL2.

Sin embargo, se desea conservar toda la lgica propia a cada orden en su propia subclase de *Orden*. La forma ms lgica sera aadir niveles a la estructura de herencia, distinguiendo as entre los *Eventos* con nombre, los *Eventos* sin nombre y los *NoEventos*, pero esto slo resolvera parte del problema: seguiramos teniendo que preguntar si la *Orden* usada pertenece a la lista de aquellas que afectan al paquete actual, o de la lista de *Ordenes* ocupadas del manejo de libros. Tendramos entonces que utilizar herencia mltiple. Como vemos, preguntar por la subclase exacta de la *Orden* segn el criterio que necesitemos en un momento determinado aade una complejidad al cdigo y crea un acoplamiento mucho mayor de lo esperado.

La alternativa se halla en el concepto conocido como “duck typing” [?], muy popular en lenguajes con sistemas de tipos dinmicos como Python, y defendido (aunque no bajo ese mismo nombre) por autores como Bruce Eckel [?]. La definicin del glosario de Python [?] es la siguiente:

Pythonic programming style that determines an object's type by inspection of its method or attribute signature rather than by explicit relationship to some type object (f it looks like a duck and quacks like a duck, it must be a duck.") By emphasizing interfaces rather than specific types, well-designed code improves its flexibility by allowing polymorphic substitution. Duck-typing avoids tests using `type()` or `isinstance()`. Instead, it typically employs `hasattr()` tests or EAFP programming.

Es decir, no distinguimos a los tipos por “quines” son, con `instanceof` en Java o `typeid` en C++, por ejemplo, y luego hacemos *upcasting* a una clase o interfaz comn. En su lugar, los distinguimos por cmo se comportan: si `anda()` como un *Pato* y `hace_cuac()` como un *Pato*, entonces es un *Pato* para nuestro cdigo.

4. Desarrollo del proyecto

Se podra ver el “duck typing” como una versin de la programacin genrica de C++ en que la comprobacin de tipos se hace en tiempo de ejecucin y slo se requieren aquellos mtodos que son realmente usados, en vez de todos las que aparecen en el cdigo.

Existen dos estilos para su implementacin, como sugiere la definicin anterior. El segundo estilo hace referencia al acrónimo EAFP (Easier to Ask for Forgiveness than Permission): asumimos que los mtodos necesarios se implementan y capturamos los errores si nos equivocamos. Dado que en el anlisis esttico se mezclan las *Ordenes* que cumplen o no cada uno de los criterios libremente, este mtodo no nos sirve: el manejo de excepciones no es un mecanismo de control de flujo, al fin y al cabo.

El primer mtodo s es til, pero notemos que hace uso de algo que no todo lenguaje tiene: la capacidad de consultar y operar sobre la propia estructura del programa, en tiempo de ejecucin. Dicho proceso es conocido como *reflexin*, y se halla integrado dentro de muchos lenguajes, como Java, C#, Perl o Python.

As, el anlisis esttico asume que si una clase implementa el mtodo `get_nombre_paquete` cambia el paquete actual, si implementa `get_identificador_evento` representa a un *Evento* con un nombre referenciable posteriormente, y si implementa el mtodo `get_ruta_libro`, que de alguna forma ha incluido los contenidos del libro bajo la ruta especificada bajo el mundo lgico de ACL2.

Patrón Método Fábrica: creacin de rdenes y Procesos

Se us el patrón *Método Fábrica* de [?] para aislar al resto del post-procesador de la lgica necesaria para identificar qu tipo de *Orden* o *Proceso* hay que crear exactamente.

Podemos ver en el diagrama de clases de diseo general de la figura 4.13 en la pgina 98 dos clases abstractas, *Orden* y *Proceso*. Las instancias de sus subclases se crean mediante los mtodos `crear` de `ACL2::Orden::Fabrica` y `ACL2::Proceso::Fabrica`.

En ambos casos, son los valores de los argumentos proporcionados los que determinan qu subclase concreta de la clase abstracta base se va a crear. En el caso de *Orden*, se enva la S-expresin, el paquete Lisp actual y la salida. En el caso de *Proceso*, se enva la salida.

`ACL2::Proceso::Fabrica` requiere algo ms de participacin por parte de las subclases de *Proceso*. Todas deben registrar una subrutina `annima` que determine si la salida enviada por *Meta* se corresponde con ella, y que devuelva los parmetros requeridos al constructor en dicho caso. Esto podra considerarse como un uso del patrón *Orden*, descrito en ms detalle en la capa de presentacin.

Dicha fábrica queda as como un punto central fcilmente accesible por *Meta* de creacin de *Procesos*, y la lgica especfica a cada proceso se halla concentrada y separada del resto en un nico fichero.

Patrón Método Plantilla

Este patrón, también proveniente de [?], se usa en la jerarquía de *Orden* para reunir todo el comportamiento común en el filtrado de las órdenes.

Toda orden en ACL2 produce una serie de mensajes de error, aviso u observación. También se añade usualmente (pero no siempre) un resumen, cuyo formato es fijo en toda orden. Se da además que la lógica de manejo de errores de ACL2 es siempre la misma.

Toda esta lógica común es reunida en un método de una clase base, que emplea una serie de “operaciones de enganche” protegidas, que permiten especializar mediante herencia partes de ese método con la lógica específica para cada orden, facilitando añadir nuevas órdenes o modificar las existentes. El nombre de “método plantilla” viene de esa capacidad de rellenar los “huecos” de su lógica.

Orden dispone de dos métodos plantilla:

1. `inicializar` dispone de los enganches:

- `protegido_inicializar_sexpr`: operación abstracta.
- `protegido_inicializar_salida`: operación abstracta.
- `protegido_extraer_sumario`: tiene una implementación por defecto.

2. `escribir_xml` se puede especializar a partir de:

- `protegido_cabecera`: tiene una implementación por defecto.
- `protegido_cuerpo`: operación abstracta.
- `protegido_final`: tiene una implementación por defecto.

El uso del prefijo “`protegido_`” en los nombres se trata de un esquema de nombrado, siguiendo la práctica común en muchos módulos del CPAN, y no implica la existencia de un mecanismo de control de acceso real, por sencillez.

Jerarquía de órdenes

El diagrama de clases de diseño UML puede verse en la figura 4.16 de la página 104.

Puede verse como cada clase redefine los métodos de enganche necesarios, añadiendo algunas operaciones privadas para su propia funcionalidad.

4. Desarrollo del proyecto

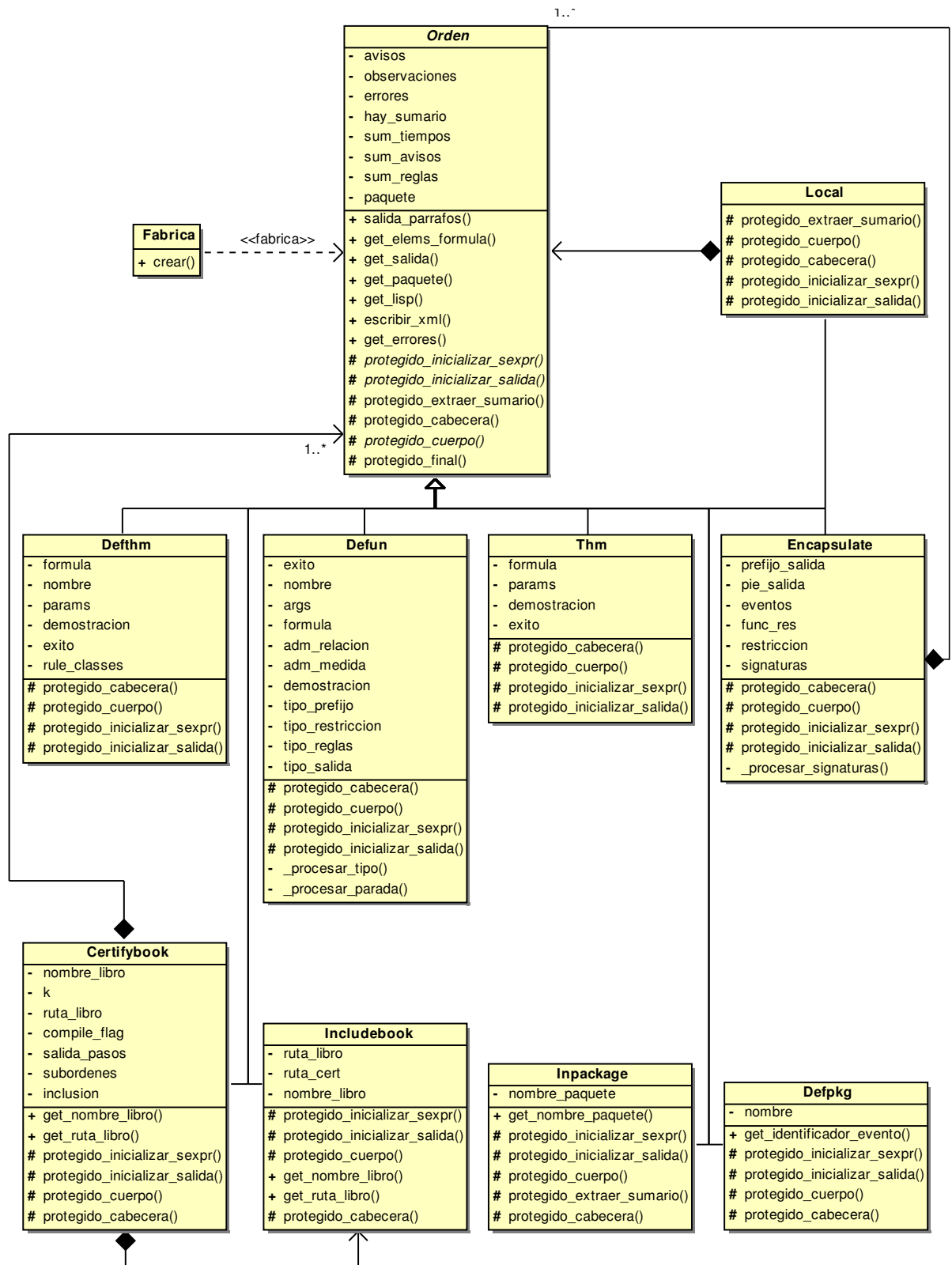


Figura 4.16.: Diagrama de clases de rdenes de ACL2 :: Procesador

Jerarquía de Procesos

El diagrama de esta jerarquía se halla en la figura 4.17 de la página 106.

Los métodos privados estáticos `_identificar` contienen el código que se registra en la *Fabrica*, utilizando una subrutina `annima` (parecida a una clausura lambda). Para ocultar los detalles, *Fabrica* emula en su método `registrar` control de acceso por paquete.

Un ejemplo: definición de teorema

Para explicar en más detalle qué es lo que se requiere para el filtrado de una orden de ACL2, usaremos como ejemplo el procesamiento de un supuesto evento de definición de un teorema, `defthm`.

Tras crearse la *Orden* a través de la *Fabrica Abstracta Orden::Fabrica*, se genera la versión XML de la salida de ACL2 en dos pasos. Cada paso se corresponde con uno de los métodos plantilla antes mencionados.

En primer lugar, se captura toda la información disponible en la salida de ACL2 (si se proporciona) y la S-expresión original mediante `inicializar`:

1. `_init_desde_salida` definido por *Orden* retira toda la información de la salida no específica a la *Orden Defthm*, simplificando así su inicialización.

Ello incluye retirar los mensajes de ACL2, los mensajes del colector de basura del intérprete Lisp y el sumario de la ejecución de la orden.

2. `protegido_inicializar_sexpr`, propio de *Defthm*, obtiene información específica a *Defthm* a partir de su S-expresión:

- Fórmula a demostrar.
- Nombre del teorema.
- Clases de regla a generar, con el valor por defecto “:REWRITE”, indicando que se usará una regla de reescritura de miembros.
- Activación o desactivación de la bandera OTF (“on through the fog”), obligando a ACL2 a que, si con sólo simplificar no basta, no intente inducir sobre la fórmula original, sino sobre las múltiples fórmulas resultantes de las simplificaciones.
- Captura de otros argumentos de interés.

4. Desarrollo del proyecto

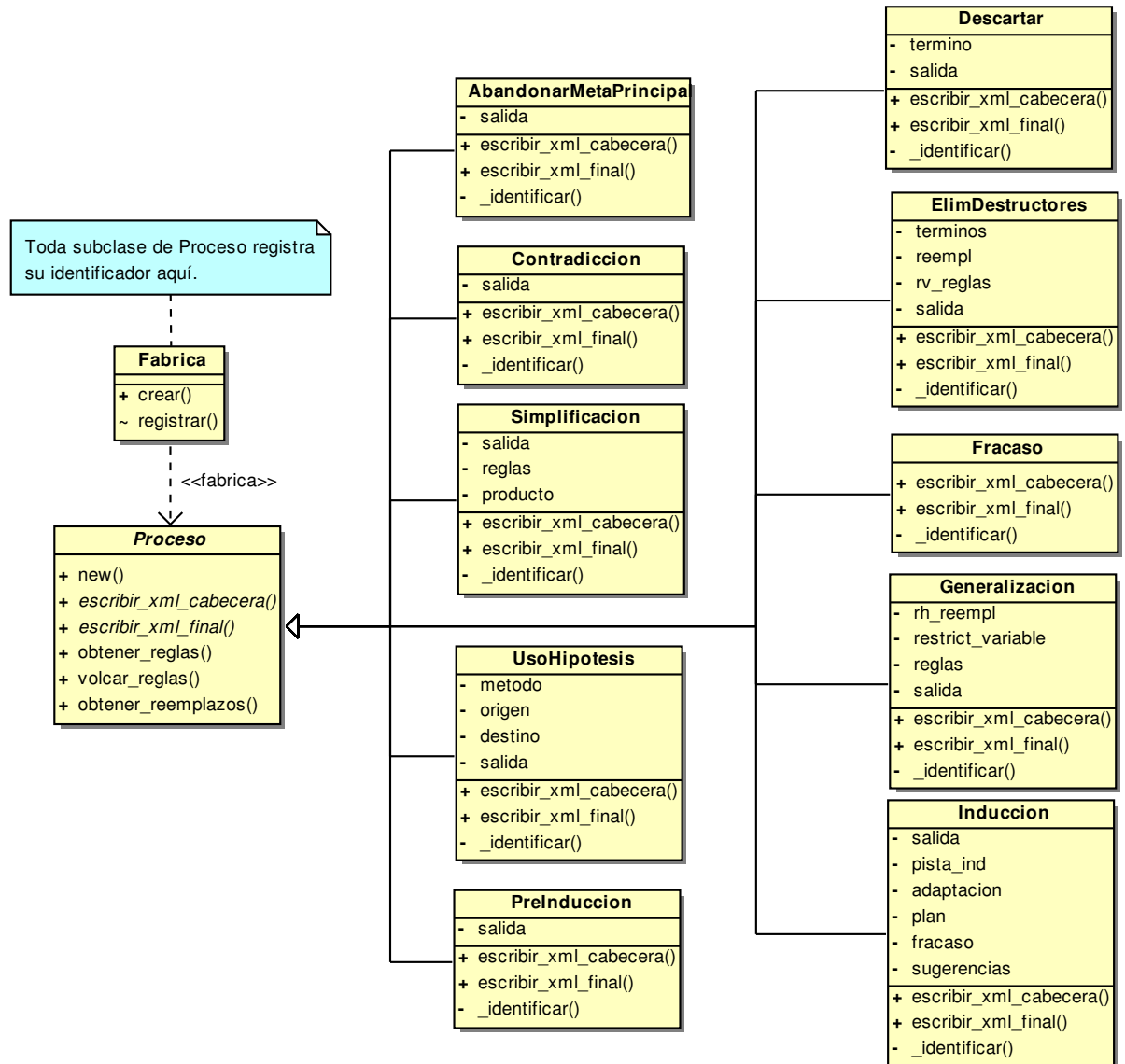


Figura 4.17.: Diagrama de clases de Procesos de ACL2 :: Procesador

3. `protegido_inicializar_salida`, definido por *Defthm*, se ocupa de recabar toda la informacin de inters del texto de la salida de ACL2. Nos interesan las dependencias de almacenamiento de las reglas generadas, y los pasos de la demostracin realizada.

En primer lugar, recuperaremos la informacin mediante el mtodo plantilla `inicializar`:

- a) Obtener cada una de las metas e inicializarlas. Tras dividir la salida completa en la salida de cada meta, se emplear el *Mtodo Fbrica* `crear` de *Proceso::Fbrica*, que invoca a las subrutinas annimas registradas para reconocer e instanciar el *Proceso* exacto seguido.

Cada meta introducida se guardar en un vector asociativo para el siguiente paso, utilizando su etiqueta, como “Subgoal *1”.

- b) Organizar jerrquicamente las metas.

Esto se puede hacer fcilmente a partir de las etiquetas: una meta con la etiqueta “Subgoal *1.1” tiene como padre la meta con la etiqueta “Subgoal *1”, por ejemplo.

Gracias al vector asociativo anterior, slo hay que calcular la etiqueta del padre, recuperarlo y aadirle la submeta.

A continuacin, invocaremos al otro mtodo plantilla, `escribir_xml`, para obtener el documento XML que lista toda la informacin antes recogida:

1. `protegido_cabecera`: escritura de la cabecera de la salida XML correspondiente al *defthm*. Se abre un nuevo elemento *defthm* mediante `startTag` de *XML::Writer*.
2. `_escribir_mensajes`: este mtodo es parte de la funcionalidad implementada por *Orden*, y vuelca los mensajes de ACL2: avisos, observaciones y errores.
3. `protegido_cuerpo`: escritura del cuerpo de la salida XML del *defthm*. Se vuelca la informacin conseguida antes en `protegido_inicializar`, incluyendo la demostracin, delegando en el mtodo `escribir_xml` de *Resultado*.
4. `_escribir_sumario`: tambin parte de la funcionalidad predefinida por *Orden*, vuelca el sumario de la ejecucin del evento, incluyendo tiempos, reglas usadas y avisos dados.
5. `protegido_final`: en este caso, *defthm* no redefine la funcionalidad predefinida por *Orden*, cerrndose el elemento inicial escrito en `protegido_cabecera` con un elemento del mismo nombre que la orden.

4. Desarrollo del proyecto

Se ha descrito también de manera gráfica este proceso utilizando diagramas de secuencia UML 1.4. Se necesita dividir el diagrama en dos: el primero, dedicado al análisis, es la figura 4.18 de la página 109, y el segundo, dedicado a la salida, es la figura 4.19 de la página 110.

Por razones de espacio, algunas de las líneas de vida no se extienden hasta el final de la hoja, sin que esto implique su destrucción. El nivel de detalle escogido no describe las operaciones internas de *Orden* o *Defthm* que no requieran interacción por parte de otras clases, o los mensajes de poco interés enviados a *XML::Writer*.

Los vectores asociativos de Perl han sido modelados como objetos de la clase *Hash*, por uniformidad.

4.5.3. Capa de filtrado: YAXML::Reverse

Descripción general

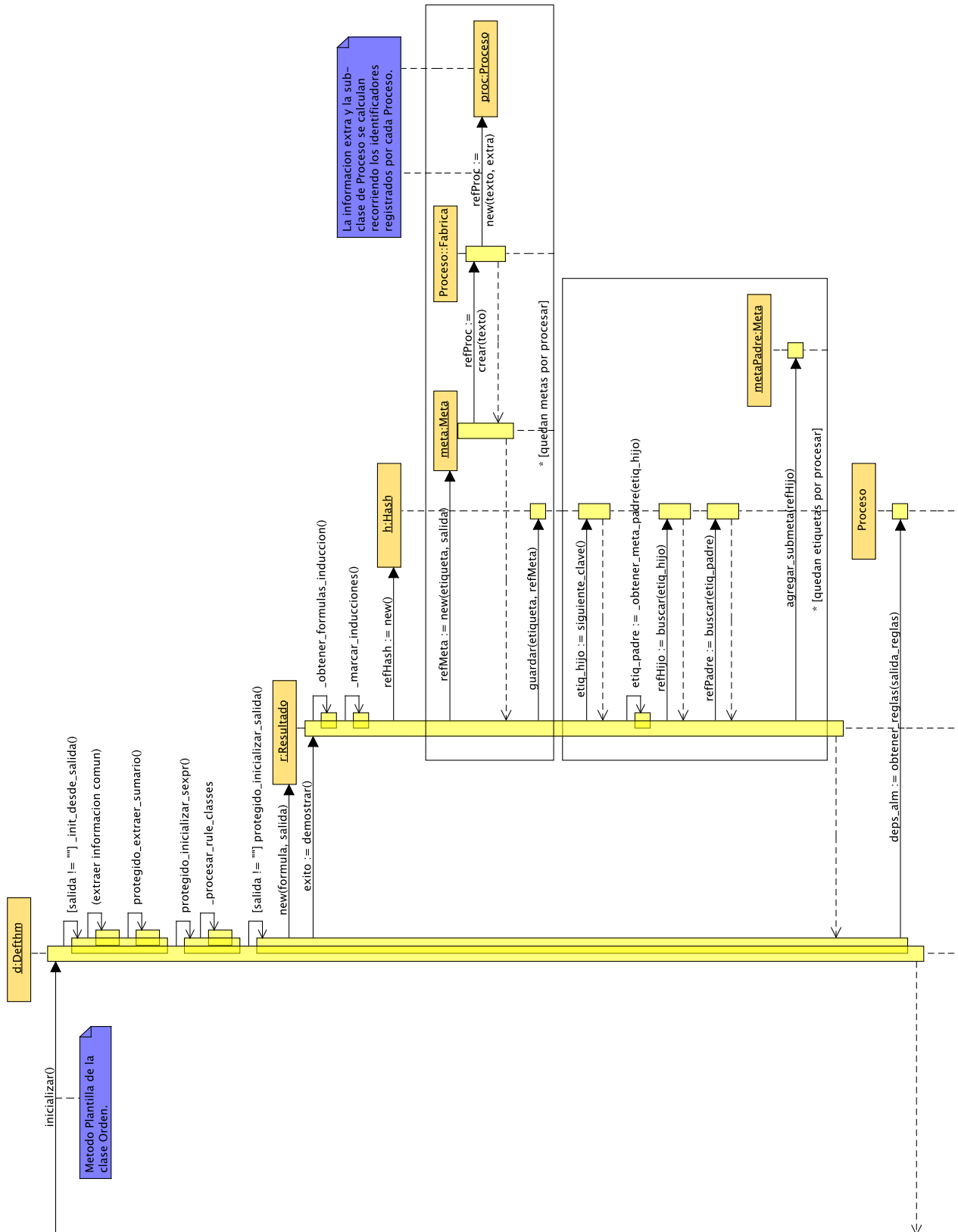
Este conversor implementa la correspondencia XML \rightarrow YAML conocida como YAXML [?] en sentido inverso, con dos objetivos en general:

- En primer lugar, para permitir a XMLEye abrir cualquier documento basado en el metalenguaje YAML, o su subconjunto (desde YAML 1.1) JSON. Ello permite tratar una familia completa nueva de formatos.
- En segundo lugar, para todos aquellos que necesiten aplicar alguna herramienta a sus documentos que no está disponible para YAML, como transformaciones definidas de forma funcional mediante XSLT.

El proceso es relativamente sencillo:

1. Se analiza el documento origen YAML, deserializándose a una estructura de datos Perl. Este primer paso se consigue mediante el módulo `Perl::YAML::XS`, un *binding* que permite aprovechar la funcionalidad de la biblioteca `libyaml` (disponible en <http://pyyaml.org/wiki/LibYAML>), y analizar documentos YAML 1.1 y JSON. Existen una serie de restricciones, que veremos posteriormente en el apartado 4.5.3 de la página 113.
2. Se representa dicha estructura como un árbol XML, con una versión mejorada del formato sugerido por YAXML para tratar ciertos casos especiales.

Sin embargo, hay una serie de detalles de interés en la inversión de YAXML que consideramos merece la pena mencionar. También se han encontrado algunas limitaciones debidas en primer lugar al módulo `YAML::XS`, y en última instancia al propio lenguaje Perl, pero se estima que tendrán realmente poco impacto en su uso.

Figura 4.18.: Diagrama de secuencia de filtrado de `d.f.thm` (anlisis)

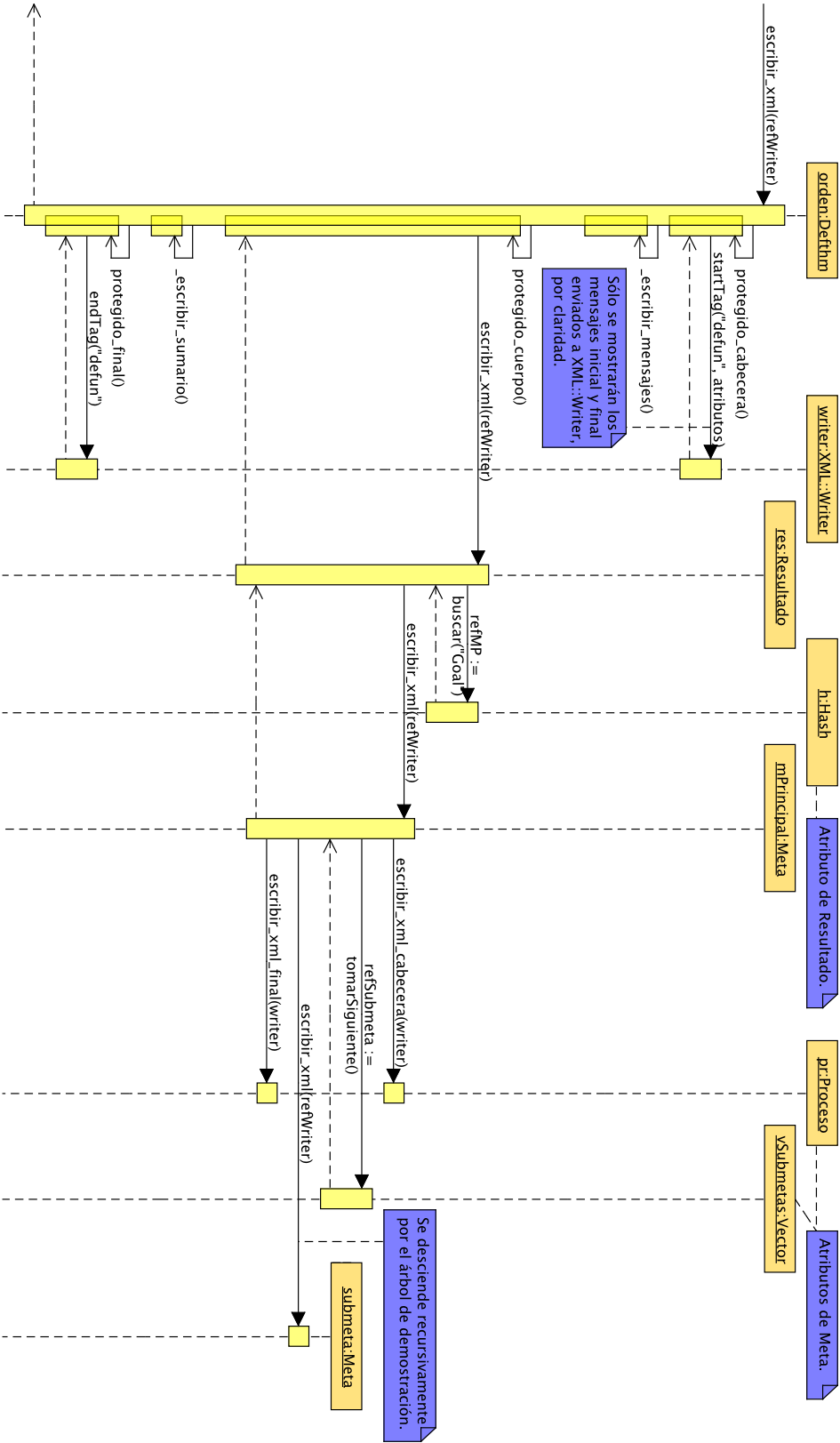


Figura 4.19.: Diagrama de secuencia de filtrado de defun (generacin de salida)

Regeneracin de anclas y alias en el fichero XML

Hay una diferencia muy importante en los modelos de informacin comnmente usados en YAML y XML: el modelo de informacin de presentacin de YAML, tal y como se coment en §4.4.4 (pgina 86), utiliza grafos dirigidos con raz, y el modelo DOM comnmente usado en XML emplea rboles. Esto implica dos problemas, fundamentalmente:

- Un primer problema, de menor importancia, es el hecho de que si un nodo del grafo tiene varios arcos entrantes, tendr que ser repetido varias veces en el rbol final. Este problema se agrava cuando el nodo a repetir no es un nodo hoja, sino un nodo raz de un subgrafo: tendremos que repetir el subgrafo completo. De todas formas, este problema es ms una cuestin de eficiencia, y en ciertos casos se podra incluso aceptar el coste adicional.
- El problema mayor es que en los grafos de YAML, se admiten ciclos: no existe forma de representar directamente estas estructuras cclicas en un rbol XML.

La forma en que YAXML resuelve estos problemas es utilizando no el modelo de informacin de representacin, sino el modelo de informacin de serializacin, que s est basado en rboles. En dicho modelo, se definen los conceptos de *ancla* y de *alias*, pudiendo as establecer enlaces entre los nodos del rbol, y resolver los dos problemas anteriores. En YAXML, se representan como atributos de un elemento sin contenido (como `<a/>`). Pueden compararse los resultados para un caso sencillo en la figura 4.20 de la pgina 112.

Existe un problema, sin embargo: utilizando `YAML: :XS`, lo que obtenemos es el grafo final. Para reconstruir el rbol con anclas y alias a partir del grafo, hemos de distinguir qu elementos del grafo constituyen anclas y qu elementos constituyen alias a esas anclas. Podemos aprovechar el hecho de que la estructura creada por `YAML: :XS` se basa en referencias a direcciones de memoria con vectores asociativos, listas y objetos: si en dos o ms puntos del rbol aparecen referencias a la misma direccin de memoria, sabremos que la primera aparicin ser el ancla, y todas las dems sern sus alias.

El mayor problema es que no sabremos si un nodo constituye un ancla hasta que sea usado como tal por primera vez en cualquier otro punto del documento. Por ello, tendremos entonces que dividir la generacin de la salida en dos pasos:

1. En una primera pasada, se recopila en un vector asociativo qu referencias constituyen anclas, junto con su posicin en orden de uso, a partir de la cual se generar el identificador final.
2. En la segunda pasada, se genera el cdigo XML, cuidando de generar los atributos `anchor` y `alias` en los nodos apropiados.

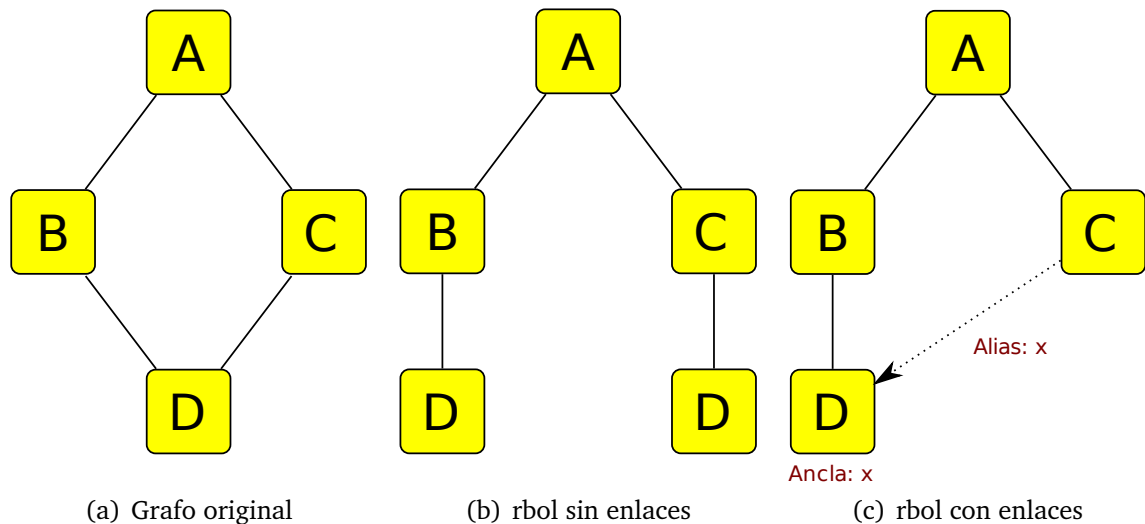


Figura 4.20.: Representacin de grafos mediante rboles XML

Pruebas de unidad automticas basadas en equivalencia

Un problema fundamental en la realizacin de pruebas de software es la elaboracin de los *orculos* [?] necesarios para comprobar si el comportamiento del software es el esperado. Existe una gran variedad de tipos, segn la naturaleza del software bajo prueba.

En lo que respecta a `YAXML::Reverse`, necesitamos un orculo que nos diga si un documento YAML ha sido convertido a XML sin prdidas ni cambios en su informacin a nivel semntico. Lo ideal es que sea lo ms automtico posible: no es factible programar un caso de prueba para cada entrada. No podemos hacer comparaciones directas del texto, pues un documento YAML puede ser escrito en muchos estilos distintos, y los documentos XML tambin disponen de ciertos mecanismos para el manejo del espacio, por ejemplo.

Una posibilidad es generar un analizador de los documentos XML obtenidos, que construyera representaciones del estilo de las de `YAML::XS` y las comparara con los grafos originales. Hay una forma de conseguir algo similar con un esfuerzo mucho menor: podemos aprovechar el hecho de que combinando `YAXML` y `YAXML::Reverse` cerramos el ciclo en torno a YAML y XML: podemos as hacer una transformacin `YAML → XML → YAML`, y comparar los grafos resultantes de procesar los documentos YAML original y final. Si son equivalentes, dado que `YAXML` opera nicamente sobre el documento XML, sabremos que ste contiene toda la informacin del YAML original sin cambios. Se puede ver un esquema del proceso completo en la figura 4.21 de la pgina 113.

El principal inconveniente de este enfoque es que `YAXML` tambin introduce sus propios fallos, y stos son vistos como fallos de `YAXML::Reverse` por el conjunto de pruebas.

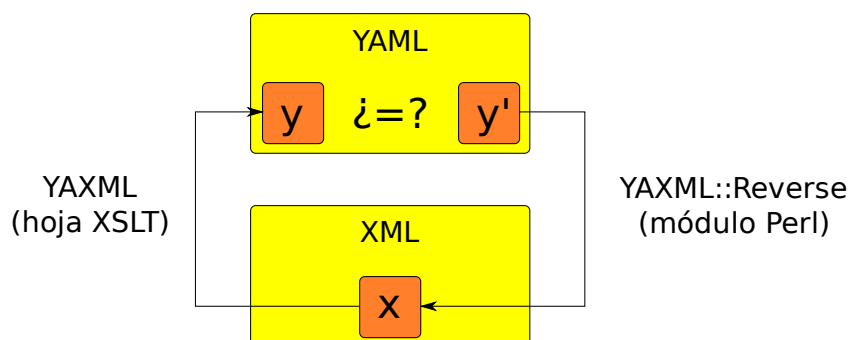


Figura 4.21.: Orculo de pruebas para YAXML::Reverse

Esto no tiene por qu ser malo: de hecho, puede verse como una forma de depurar los dos programas al mismo tiempo, usando `YAXML::Reverse` como parte del orculo de YAXML y viceversa.

Tiene la ventaja de ser completamente automtico: aadir un nuevo caso de prueba es simplemente aadir un fichero `.yaml` bajo el directorio `t/testInputs`.

Limitaciones de Perl frente a YAML

YAML se caracteriza por ser una especificacin muy flexible, pudiendo combinar los tipos bsicos de nodos y etiquetas de muchas formas. Este fragmento de la especificacin [?], que describe a los nodos de vector asociativo, es particularmente interesante:

The content of a mapping node is an unordered set of key: value node pairs, with the restriction that each of the keys is unique. YAML places no further restrictions on the nodes. In particular, keys may be arbitrary nodes, the same node may be used as the value of several key: value pairs, and a mapping could even contain itself as a key or a value (directly or indirectly).

Como vemos, podemos tener claves no escalares e incluso referencias cclicas de un mapa como clave de uno de sus elementos. El problema de los ciclos ya ha sido superado gracias a las anclas y alias, pero usar claves no escalares es ms difcil de resolver: los vectores asociativos de Perl, usados como estructura de datos nativa por `YAML::XS`, no admiten claves no escalares.

Existen mdulos como `Tie::RefHash` que permiten usar clases que implementan vectores asociativos con claves escalares y no escalares como si fueran vectores asociativos normales, pero no los reemplaza directamente, sino que modifica la forma en que Perl evala el cdigo que usa el identificador de la variable “sobrecargada”. Es una tcnica

4. Desarrollo del proyecto

de bajo nivel: no podemos pasar ese vector asociativo y usarlo en una subrutina como de costumbre, por ejemplo. Adems, y lo que es ms grave, no nos sirve una vez `YAML::XS` ha terminado su procesado: tendra que usarse antes de asignarle cualquier elemento.

Si usamos los diccionarios de Python, por ejemplo, uno podra imaginarse emular claves basadas en nodos secuencia mediante tuplas, y claves basadas en nodos de vector asociativo usando tuplas de tuplas. Python no permite usar listas ni diccionarios como claves. Sin embargo, al igual que en el caso de Perl, todo dependera del *binding* que estuviéramos usando en ese momento.

De todas formas, examinando los lenguajes basados en YAML disponibles actualmente, no parece ser una restriccin importante: an no he podido encontrar un solo formato basado en YAML que tenga claves no escalares. JSON sencillamente no tiene este problema.

Por ello, por lo pronto esta restriccin se considera de baja prioridad: solventarla requerira la reescritura completa de `YAXML::Reverse` bajo otro entorno (seguramente una aplicacin escrita en C, C++ o Python), y no supondra realmente una ventaja para su uso general.

4.5.4. Capa de aplicacin

La capa de aplicacin se ocupa de implementar las clases del modelo de la arquitectura MVP, y de proveer servicios de alto nivel independientes de la interfaz grfica. Esto permite tener separados todos los aspectos de la interfaz grfica respecto de aquellos relacionados con la lgica bsica de XMLEye.

Veremos a continuacin cules son esas clases del modelo y servicios ofrecidos, y qu diseo siguen.

Modelos de documentos

El concepto fundamental en XMLEye es el de un *Documento* XML abierto por el usuario y preprocesado. Esta clase se ocupa de encapsular toda esa informacin y gestionar los cambios en la hoja de preprocesado utilizada y la realizacin de bsquedas.

Ntese que no se ocupa de la hoja de visualizacin: como detalle de presentacin de que se trata, este aspecto se gestiona en su modelo de presentacin, que veremos en §4.5.5 (pgina 124).

Este modelo de documento integra los predicados de bsquedas mediante palabra completa e ignorando minsculas definidos mediante extensiones XPath, dejando a las capas superiores nicamente la responsabilidad de definir las *PeticionBusqueda* necesarias.

El conjunto de todas las clases empleadas, tanto del J2SE como de XMLEye, se halla en el cuadro 4.22 (pgina 116).

Hojas de estilos

Las hojas de estilos participan tanto en la capa de presentacin como en la capa de aplicacin. En la capa de aplicacin se halla toda la infraestructura necesaria para su ejecucin, atendiendo a su estructuracin en repositorios y soporte para localizacin, en combinacin con el uso de herencia entre distintas hojas. Las propias hojas son parte de la capa de presentacin.

Distinguiremos entre:

Hoja de usuario Conjunto cohesivo de una o ms hojas XSLT individuales seleccionable por el usuario que conforma un modo de preprocesar el documento o visualizar un nodo.

Se distinguen dos tipos:

Preprocesado Contienen transformaciones que afectan al documento XML completo tras su apertura. Se usan para modificar el rbol del documento o para realizar operaciones costosas como el establecimiento de enlaces.

Visualizacin Contienen transformaciones a ejecutar sobre el nodo DOM seleccionado por el usuario en cada momento.

Toda hoja de usuario contiene una o varias hojas XSLT especiales cuyo nombre de fichero sigue el patrón `(nombre hoja)[idioma[_pas]].xsl`, donde `[]` indica una parte opcional y `()` una parte obligatoria. Llamamos a dichas hojas sus *puntos de acceso*. Se intentar antes acceder a los puntos de acceso ms especficos respecto del país e idioma de la localización actual.

Hoja XSLT Fichero individual que contiene un documento XML definido sobre el vocabulario W3C XSLT.

Las hojas de usuario han de organizarse cumpliendo una serie de requisitos:

- Han de poder ser fcilmente localizables segn el país y el idioma de la localización establecida por el usuario por defecto en su sistema.
- Han de ser fciles de instalar, sin requerir configuracin alguna.
- Han de ser fciles de elaborar, pudiendo basarnos en hojas de usuario ya existentes.

4. Desarrollo del proyecto

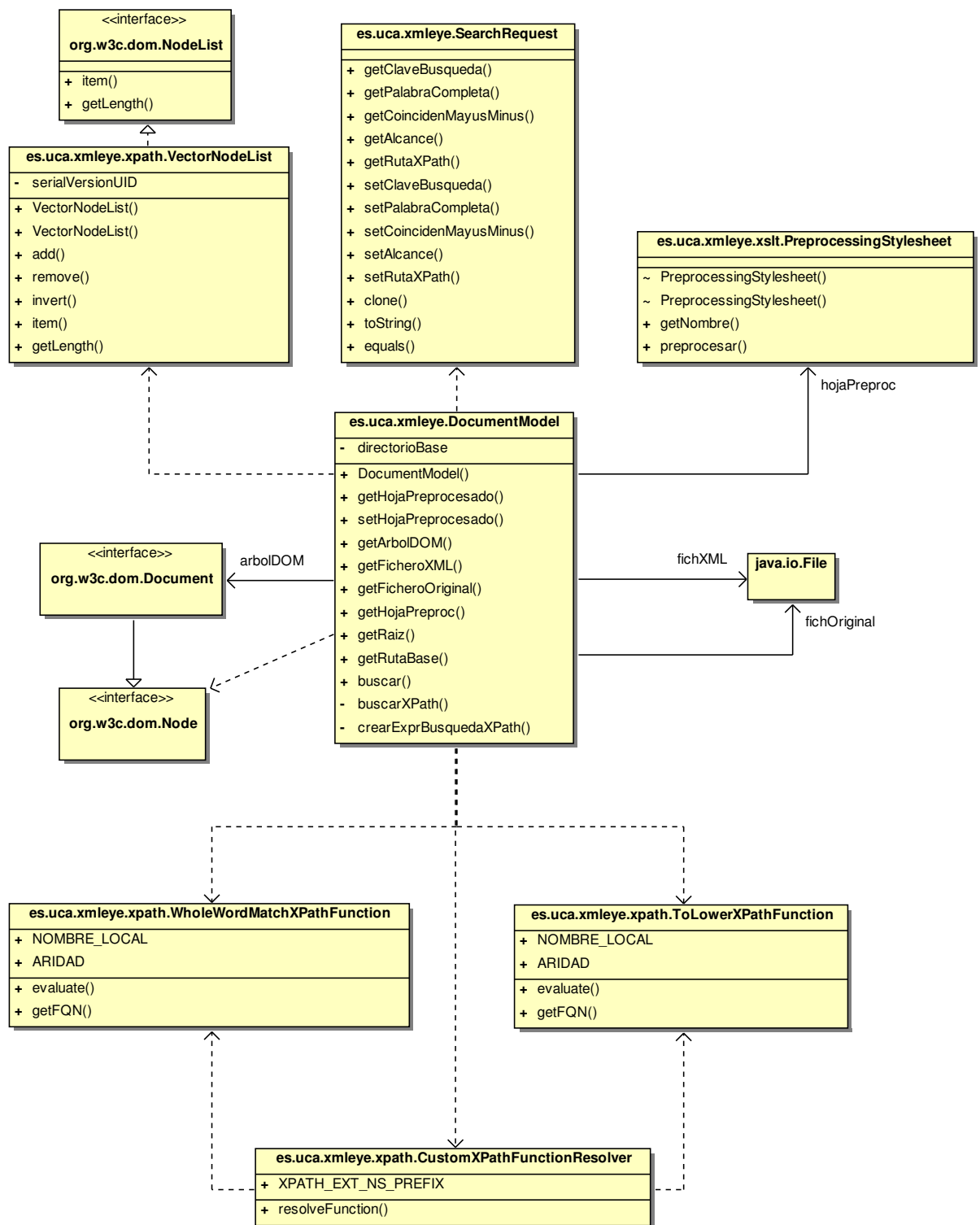


Figura 4.22.: Diagrama de clases de diseo para modelado de documentos

Los detalles de implementacin de cada tipo de hoja de usuario y aquellos comunes a toda hoja de usuario han sido encapsulados en clases, permitiendo a la capa de presentacin realizar transformaciones sin conocer los detalles de la biblioteca XSLT usada.

En cuanto a los requisitos sobre la organizacin de las hojas de usuario, se ha mantenido la separacin de responsabilidades moviendo estos requisitos a una clase que acta de *repositorio*, que se ocupa de localizar las hojas de usuario y configurarlas para conseguir al mismo tiempo la capacidad de reutilizar cdigo y localizar las hojas. La carga del repositorio se realiza al iniciar XMLEye.

La organizacin general de las clases utilizadas se halla en la figura 4.23 de la pgina 119.

Organizacin del repositorio El esquema de almacenamiento del repositorio no es configurable, siendo as posiblemente menos flexible, pero muy sencillo de utilizar. La organizacin de las hojas de usuario se explica mejor con un ejemplo. Una posible distribucin de hojas XSLT de una hipottica hoja de usuario de visualizacin llamada *perso* bajo el directorio base con la ruta relativa *xslt* sera:

- `xslt/view/perso/perso.xsl`: punto de acceso inicial para la localizacin por omisin.
- `xslt/view/perso/perso_en.xsl`: punto de acceso inicial para la localizacin inglesa, sin especificar el pas.
- `xslt/view/perso/perso_en_GB.xsl`: punto de acceso inicial para la localizacin inglesa en el Reino Unido.
- `xslt/view/perso/logica.xsl`: hoja complementaria que permite que los tres idiomas compartan la misma lgica.

Si fuera una hoja de preprocesamiento, se usara *preproc* en vez de *view*.

Extensibilidad Las hojas de usuario son importadas por la hoja principal no modificable de su tipo, en la raz del repositorio, que define la mnima funcionalidad comn: `view.xsl` para las hojas de visualizacin y `preproc.xsl` para las de preprocesado.

Adems, una hoja de usuario puede importar a otras y as especializarlas. Es el caso de *ppACL2*, que importa a *xml*.

Para abstraer a las hojas de la lgica de internacionalizacin, una hoja de usuario que desee importar a otra puede usar URI (Uniform Resource Identifier) especiales como `view_ppACL2`.

Mediante esta URI, se accede a la hoja de usuario de visualizacin *ppACL2* a travs del punto de acceso que mejor se ajuste a la localizacin actual del usuario.

4. Desarrollo del proyecto

Internacionalizacin El ejemplo de `perso` anterior ilustra el mecanismo de localizacin establecido por la lgica de la capa de aplicacin. Dicho mecanismo imita al usado por los *ResourceBundles* de la biblioteca estndar de Java.

Si la localizacin actual por defecto es “es_ES” (espaol, Espaa) y se busca la hoja de visualizacin `ppACL2`, *ControlHojasXSL* intentar localizar un punto de acceso en el siguiente orden:

1. `xslt/view/ppACL2/ppACL2_es_ES.xsl`
2. `xslt/view/ppACL2/ppACL2_es.xsl`
3. `xslt/view/ppACL2/ppACL2.xsl`

Descriptores de formatos

Otro servicio tambin a prestar por la capa de aplicacin es el de la gestin de los descriptores de formatos de documento. Al igual que en el caso de las hojas, se necesitaba abstraer a las capas superiores de los detalles de un descriptor individual y de la forma en que se hallan organizados.

Volvemos a tener otro repositorio, esta vez de descriptores, en el que se siguen una serie de convenciones para evitar la necesidad de realizar cualquier tipo de configuracin. A diferencia del repositorio de hojas, no tratamos con un nico rbol de directorios: se pueden encadenar varios, y as poder tener una serie de ajustes a nivel global para todos los usuarios, y otros locales para el usuario actual. Para restaurar los ajustes locales a los globales, slo hemos de borrar el descriptor a nivel local y actualizarlos.

Este repositorio abstrae tambin la lgica que hace corresponder un descriptor a un fichero determinado, y se ocupa de validar contra un XML Schema todos los descriptores. El repositorio es un *AbstractListModel* de Swing, por lo que puede integrarse fcilmente como modelo de un formulario e incluso registrar *Observadores* sobre l si se estima necesario, pero no es obligatorio.

Por otro lado, las clases para los descriptores individuales encapsulan el formato de estos ficheros y proporcionan toda la validacin necesaria de su contenido, lanzando excepciones cuando se intentan establecer valores no vlidos en algn atributo.

Las clases implicadas se hallan en el cuadro 4.24 de la pgina 120.

Notificacin de cambios sobre ficheros

Un requisito muy interesante era el de poder vigilar los cambios realizados sobre los documentos durante su visualizacin, pudiendo mantener el editor y XMLEye abiertos en paralelo. Para ello se ha implementado una clase que ofrece una interfaz basada en

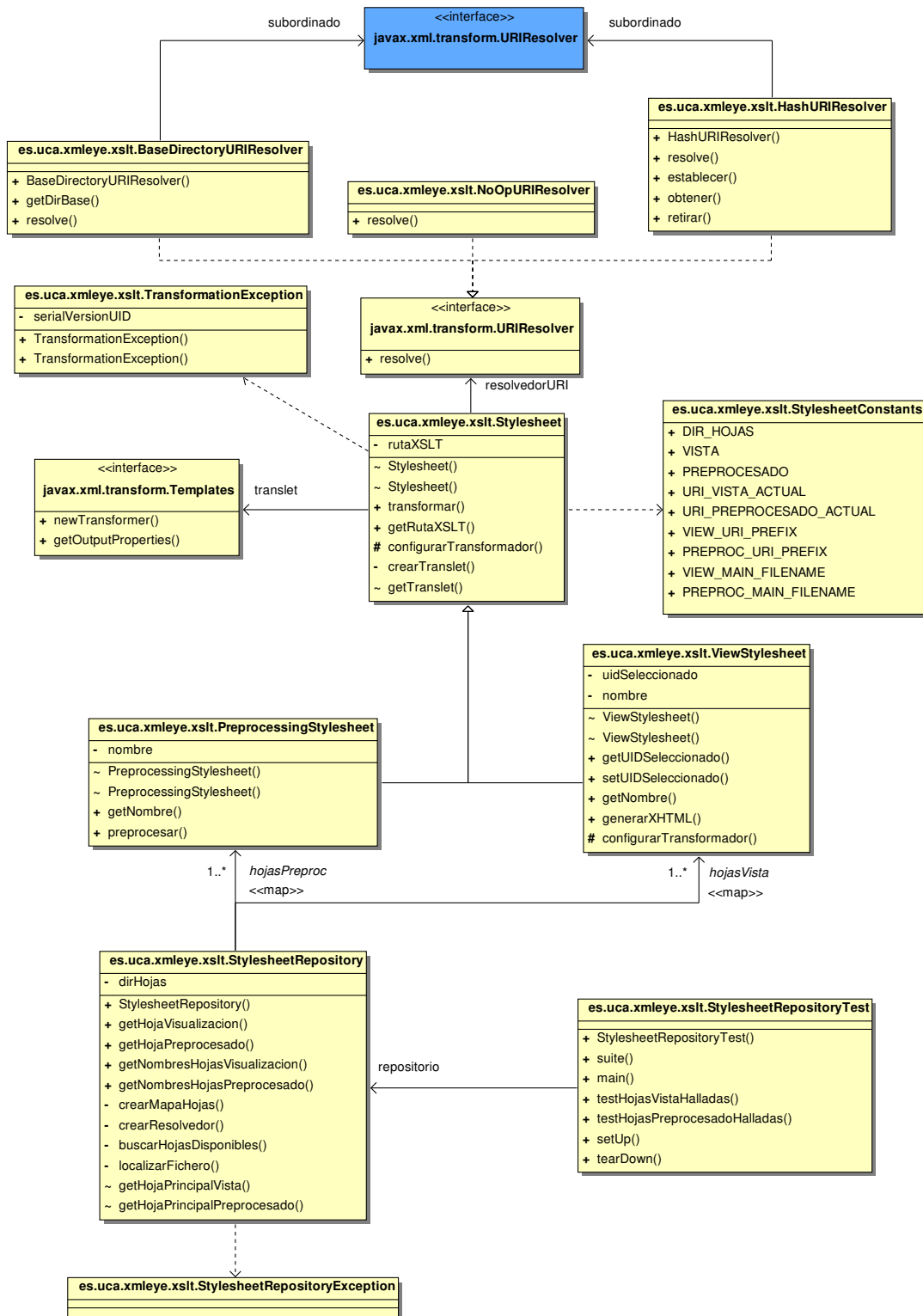


Figura 4.23.: Diagrama de clases de diseño para las hojas de usuario

4. Desarrollo del proyecto

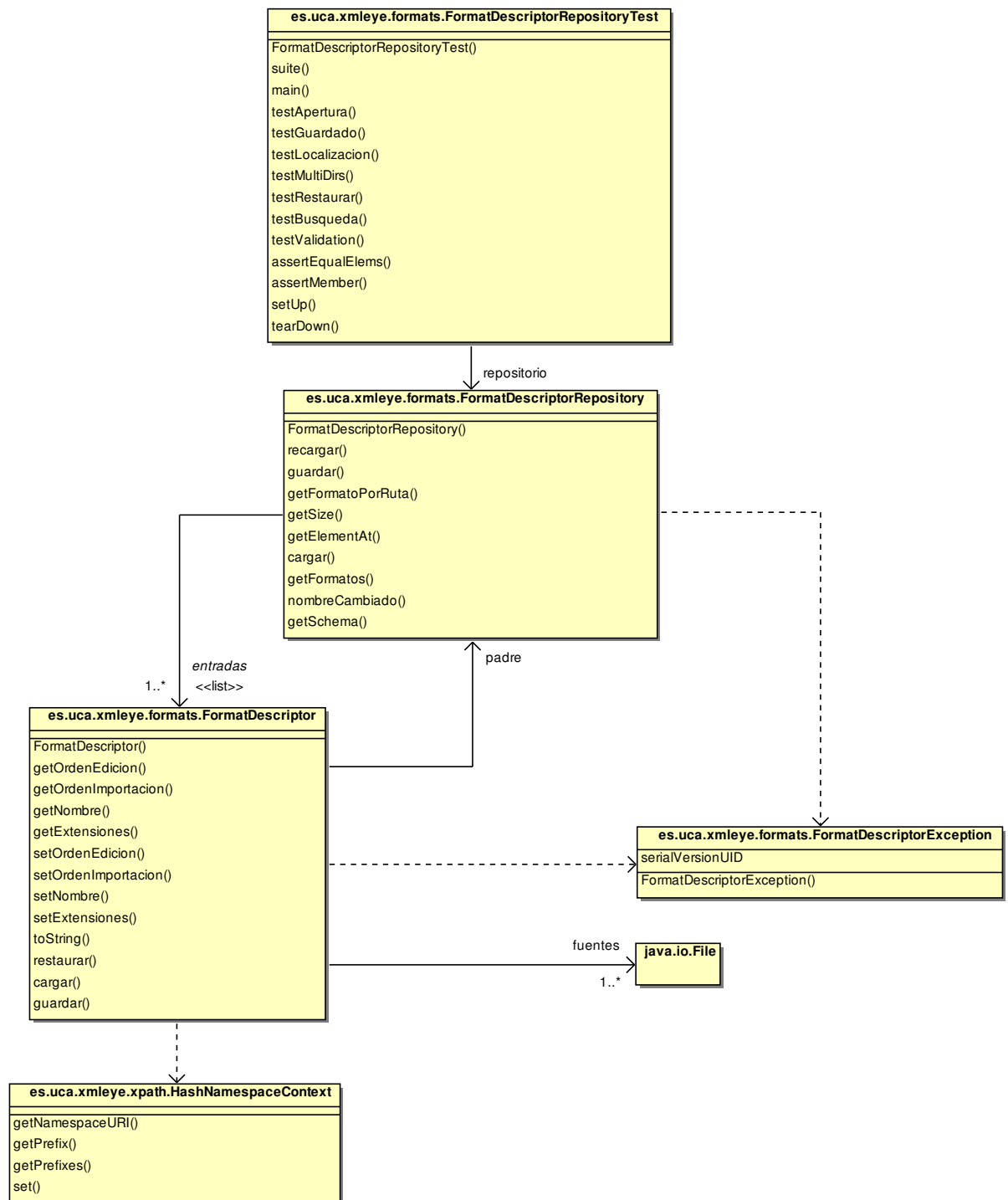


Figura 4.24.: Diagrama de diseo de clases de descriptores de formatos

el patrón Observador, que se basa en la fecha de modificación para notificar de cambios en los contenidos del fichero.

Para ello, se usa un hilo en segundo plano de baja prioridad, que se despierta cada cierto tiempo, realiza la comprobación y vuelve a dormir. Esto evita la pérdida de rendimiento que supondría un bucle de espera activa. Además, las actualizaciones pueden pausarse y reactivarse a voluntad.

Persistencia de preferencias

En este caso es fundamental tener un punto de acceso central a todas las preferencias de los usuarios. Por estas razones, en la versión original del visor se utilizó el patrón *Singleton*, asegurando la existencia de una única instancia de una clase que implementaría una interfaz. Se utilizó una versión mejorada de la implementación usual tomada de [?], simplificando sobre la sugerida en [?].

Sin embargo, durante el desarrollo de esta nueva versión, se vio que el uso del *Singleton* dificultaría la posterior realización de pruebas con sustitutos, e introduciría una dependencia innecesaria.

Para retirarla, se aplicó el patrón *Inyección de Dependencias*: ahora no son las clases las que solicitan el gestor de preferencias, sino las que lo reciben a través de su constructor. De esta forma, no se necesita el punto único de acceso como antes, siendo muy fácilmente sustituible durante pruebas, por ejemplo, y manteniendo la independencia respecto a la implementación a través del uso de una interfaz.

El diagrama de las clases involucradas en la gestión de preferencias se halla en el cuadro 4.25 de la página 122. Podemos ver que el *Singleton* ha desaparecido, al ser ahora completamente innecesario.

Se dispone de un valor booleano almacenado directamente en dependencias, de tal forma que una clase que requiera un atributo con persistencia entre las preferencias sólo tendrá que instanciar apropiadamente esta clase y luego utilizar `get()` y `set()` sin más complicaciones.

En el diagrama de secuencia 4.26 de la página 123 puede verse cómo se gestionan las preferencias a lo largo del programa:

- Al iniciarse XMLEye, se crea una instancia de la clase principal que representa a toda la aplicación. Esta clase es la encargada de ensamblar a todas las demás clases de mayor nivel, inyectando las dependencias consideradas oportunas y evitando que dependan demasiado entre sí.
- La clase principal ensambladora crea una instancia de la implementación de gestión de preferencias a usar, y solicita que se carguen las preferencias.

4. Desarrollo del proyecto

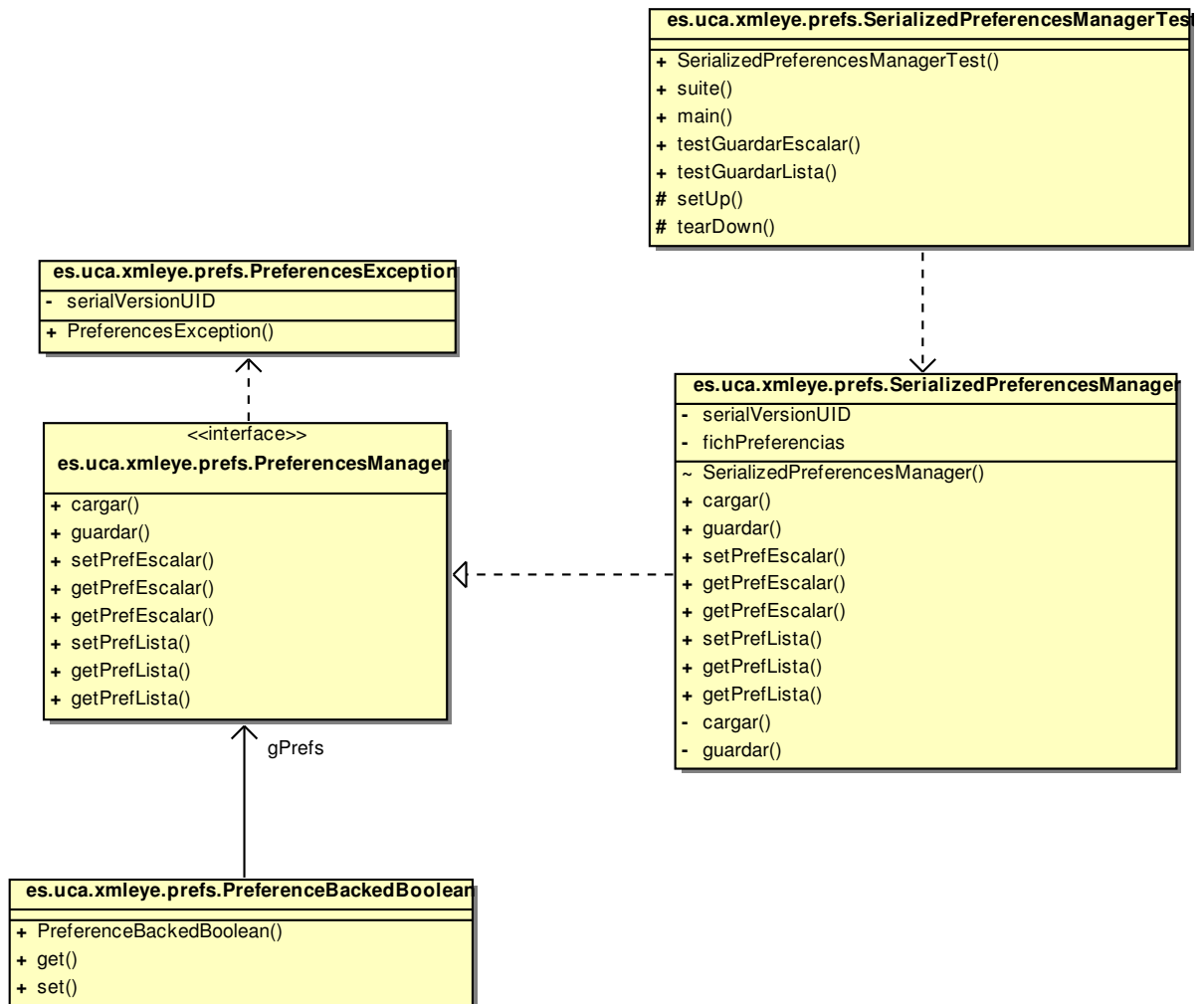


Figura 4.25.: Diagrama de clases de diseo de gestin de preferencias

- La clase principal ensambla el modelo de presentacin de la ventana principal con el gestor de preferencias (visto a travs de su interfaz) y otros parmetros. Tambin ensambla y lanza la ventana principal.
- Al cerrar la aplicacin, la misma clase principal solicita el guardado de las preferencias antes de terminar la ejecucin del programa.

4.5.5. Capa de presentacin

La capa de presentacin se encarga de la comunicacin persona-mquina, realizando peticiones sobre la capa de aplicacin y mostrando los resultados.

Durante el diseo de la interfaz, se usaron las obras [?, ?] como gua, como en el caso del diseo del dilogo “Buscar”, donde el reparto de los componentes sigue sus recomenda-

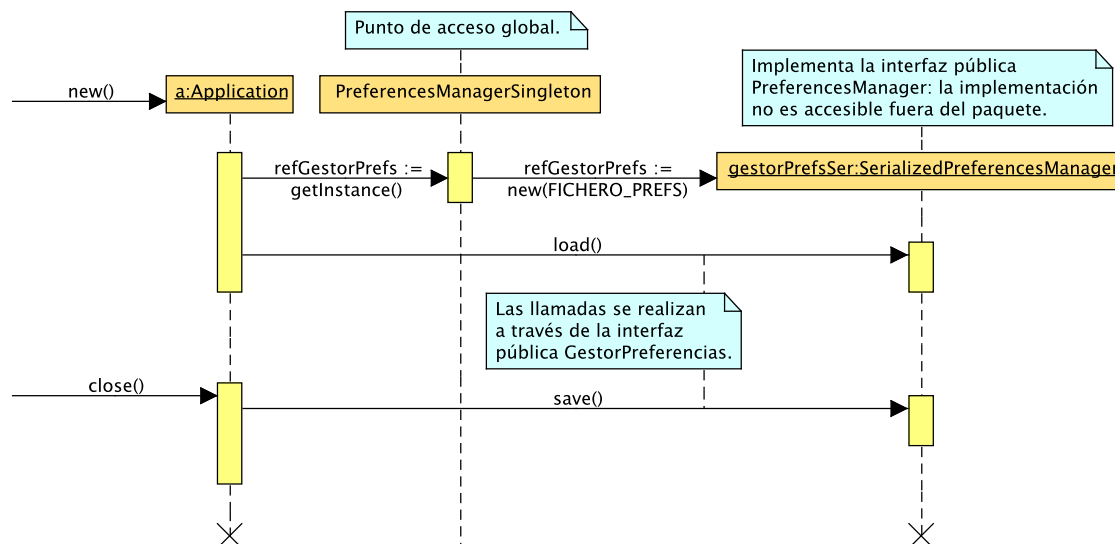


Figura 4.26.: Diagrama de secuencia de gestión de preferencias

ciones. Otras recomendaciones seguidas son la ejecución de las tareas largas en segundo plano, los elementos de los que deben constar los menús, o las combinaciones de teclas recomendadas para cada opción.

Estructura general

Como ya se comentó en § 4.5.1 (página 93), la capa de presentación se ocupará de los elementos Vista y Presentación del trozo Modelo-Vista-Presentación. En particular, se utilizará la variante *Modelo de Presentación*, que nos permite definir la lógica de una forma completamente separada de la interfaz, y así poder crear nuevas interfaces de manera mucho más libre.

XMLEye dispone básicamente de 4 formularios, tal y como se estableció durante el análisis (apartado 4.3.1 en la página 61): el formulario principal, el formulario de gestión de descriptores de formatos, el formulario de realización de búsquedas, y por último el formulario con información acerca de XMLEye.

Descartando el último de este apartado por su sencillez, podemos ver que el formulario principal y el de gestión de formatos son los más complejos. Por ello, modelaremos su estado con *Modelos de Presentación*, permitiendo una mejor comprensión y depuración de su código. El formulario de búsqueda es más sencillo, y por lo tanto no requiere dicho enfoque, como veremos posteriormente.

Además de los modelos de presentación, hablaremos en el resto de esta sección de otros aspectos interesantes del diseño de la capa de presentación.

4. Desarrollo del proyecto

Modelos de presentacin y su ensamblaje

En los formularios ms complejos, se pudo ver rpidamente que el uso del patrón *Observador* para aspectos como el control de hojas usadas y dems hacia el código cada vez ms difícil de estructurar y depurar. Además, este código impeda poder aislar de manera efectiva el estado de múltiples documentos.

Por ello, se cambi a un enfoque donde unas clases encapsulan el estado de la interfaz, de forma independiente a los controles Swing. Estas clases ofrecen una serie de métodos `get` públicos para acceder al estado de la interfaz, y disponen también de un método por cada gesto de interés que pueda provenir del usuario.

Las vistas se reducen, por lo tanto, a informar de los gestos del usuario al modelo de presentación y posteriormente sincronizarse con los modelos. Al quedarse sin la mayor parte de su lógica, el hecho de no poder realizar pruebas sobre los propios controles Swing no es tan importante: podemos hacer pruebas sobre el modelo de presentación, enviando los gestos y estudiando su estado posterior.

Los modelos de presentación pueden anidarse: así, el modelo de presentación del formulario principal contiene a los modelos de presentación de cada documento, que contienen a su vez a los modelos (ya no de presentación) de los documentos correspondiente, junto con otros elementos como una referencia al repositorio de hojas o al repositorio de descriptores de formato. Entre los campos de todo modelo de presentación se encuentra el nodo seleccionado actualmente, la hoja de visualización que está siendo usada o la última búsqueda realizada, por ejemplo.

Los modelos de presentación dependen de otros objetos, como ya hemos visto, pero no deberán quedarse acoplados a una implementación o instancia particular de ellos. En su lugar, estas dependencias se proporcionan a través del constructor, constituyendo una *Inyección de Dependencias*. Los modelos de presentación y sus dependencias y la vista del formulario principal son creados y configurados por un ensamblador central: la clase principal de la aplicación. Esto asegura que toda la lógica relevante se halle en un mismo sitio, y el resto de las clases se mantengan lo más flexibles posible.

El diagrama de clases que describe las relaciones entre las distintas vistas principales (formularios, listas de pestañas y pestañas) y los modelos de presentación se halla en el cuadro 4.27 de la página 125.

Diseño e integración del formulario de búsqueda

El formulario de búsqueda es una excepción al uso del patrón MVP. Este diálogo carece prácticamente de lógica propia, siendo únicamente una interfaz a partir de la cual el usuario puede construir la petición de búsqueda y enviarla a sus suscriptores, entre los cuales se halla el modelo de presentación del formulario principal.

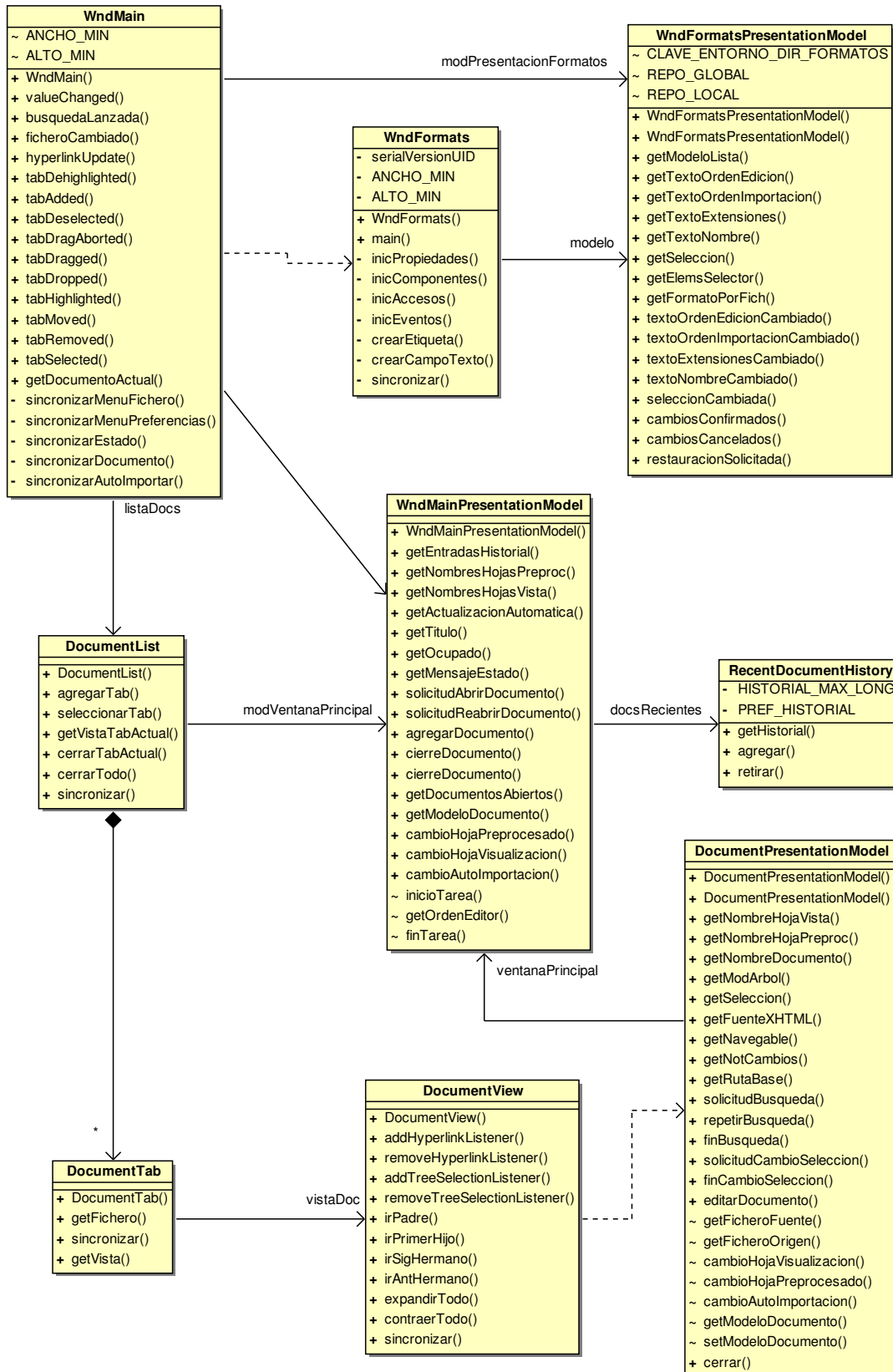


Figura 4.27.: Diagrama de clases de diseo de los modelos de presentacin

4. Desarrollo del proyecto

Aplicando de esta forma el patrón *Observador*, se evita el acoplamiento del formulario de búsqueda al formulario principal, y se puede guardar el evento para posteriormente repetir la búsqueda sin necesidad de volver a mostrarlo, por ejemplo.

El formulario principal, como es común, se limita a notificar del gesto al modelo de presentación, que utiliza la funcionalidad del modelo del documento para implementar la búsqueda, añadiendo algo más de inteligencia en lo que respecta a búsquedas repetidas.

Puede verse el diagrama de clases de diseño correspondiente en el cuadro 4.28 de la página 127.

Gestión de diálogos de mensajes

A diferencia del caso de §4.5.4, se ha mantenido el uso del patrón *Singleton* para la gestión de diálogos de mensaje: el punto de acceso global en este caso es realmente necesario, y la lógica que provee es muy concreta.

En particular, la clase a la que da acceso el *Singleton* permite mostrar esos diálogos de forma independiente al hilo en que nos hallemos: normalmente, para poder lanzar un diálogo debemos hacerlo en el hilo de Swing, o se crean condiciones de carrera indeseadas. Esta clase hace uso de las *SwingUtilities* para enviar las tareas apropiadas al hilo de Swing desde cualquier otro de los hilos, como cuando estamos convirtiendo un documento de entrada, por ejemplo.

Sin embargo, en este caso, no se ha dividido en una interfaz y una implementación, ya que a diferencia de la gestión de preferencias, no se están considerando implementaciones alternativas.

Presentación de roles XML dirigidos por datos

Los componentes usados para visualizar roles DOM XML emplean, como todo componente Swing, una modificación [?] del patrón *Modelo-Vista-Controlador* original de [?].

En esta versión modificada, llamada “arquitectura de modelo separable” por Sun y similar al patrón *Documento-Vista* mencionado en [?], se dividen los componentes en tres partes:

- El componente Swing que implementa los aspectos de vista y controlador.
- El objeto del Look & Feel instalado en el componente Swing que define el aspecto exacto del componente.
- El modelo del componente, es decir, la información que contiene, y que modifica su visualización o comportamiento. En Swing, el modelo puede contener también detalles de la interfaz gráfica: botón activado o desactivado, por ejemplo.

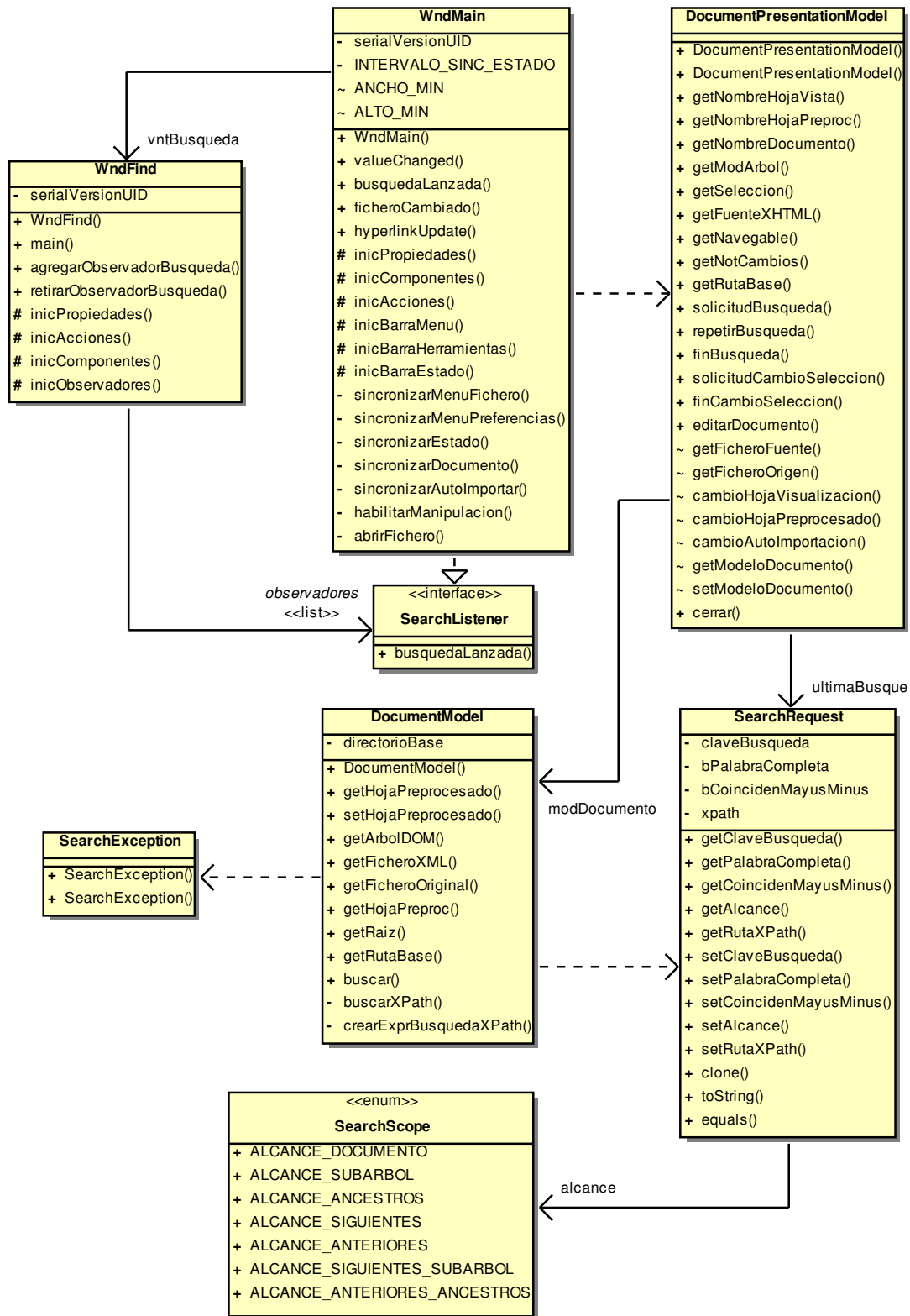


Figura 4.28.: Diagrama de clases de diseo de bsquedas

4. Desarrollo del proyecto

As, se han definido dos *TreeModel*:

DOMTreeModel Modelo sobre un rbol DOM XML estndar, que hace accesibles los nodos de tipo elemento.

DataDrivenTreeModel Especializacin del anterior, este modelo incorpora la capacidad de realizar falsas podas, ocultando nodos de la visualizacin sin eliminarlos del rbol DOM.

La decisin de ocultar un nodo o sus hijos se halla en el propio documento XML. De esta forma, el usuario puede controlar este aspecto a travs de las hojas XSLT, dirigiendolo todo por datos.

En los campos `ATTRIBUTE_LEAF` y `ATTRIBUTE_HIDDEN` de *DataDrivenTreeModel* se hallan los atributos que deben tomar el valor del campo `ENABLED_VALUE` para activar la ocultacin de los hijos o del propio nodo, respectivamente.

Por las mismas razones que *DataDrivenTreeModel*, se implement lgica de dibujado dirigida por datos de los nodos del *JTree* mediante la clase *DataDrivenTreeCellRenderer*. Este dibujador puede modificar el icono y la etiqueta de un nodo a travs de sus propios atributos `NODELABEL_ATTRIBUTE` y `NODEICON_ATTRIBUTE`.

Se extendieron las capacidades de *JTree* en *DOMTree*, implementando operaciones para encapsular la navegacin por el rbol y abstraer al resto de la interfaz de los detalles, como `selectParent`.

En la figura 4.29 (pgina 129) se ilustran en un diagrama UML las relaciones entre las clases relacionadas con la visualizacin de rboles XML, incluyendo sus casos de prueba.

Mltiples mtodos de acceso a una accin por el usuario

Otro patrn de diseo empleado comnmente en las interfaces grficas es el patrn Orden, y esta interfaz no es una excepcin.

En este patrn, se define una interfaz para una accin genrica. El lanzador de las acciones recibe objetos que cumplen tal interfaz. Al generarse el evento de inters, reacciona ejecutando cada una de las acciones a travs de dicha interfaz.

Este patrn permite aislar al elemento que lanza la accin del que realiza la accin. As, podemos asociar cualquier accin a un control Swing, sin que deba saber exactamente qu hace esa accin. Slo conoce la interfaz que debe cumplir la accin, limitndose a invocar al mtodo correspondiente.

Adems de reducir el acoplamiento, tambin mejora la estructura del programa evitando la duplicacin de cdigo al codificar la misma lgica en varios componentes.

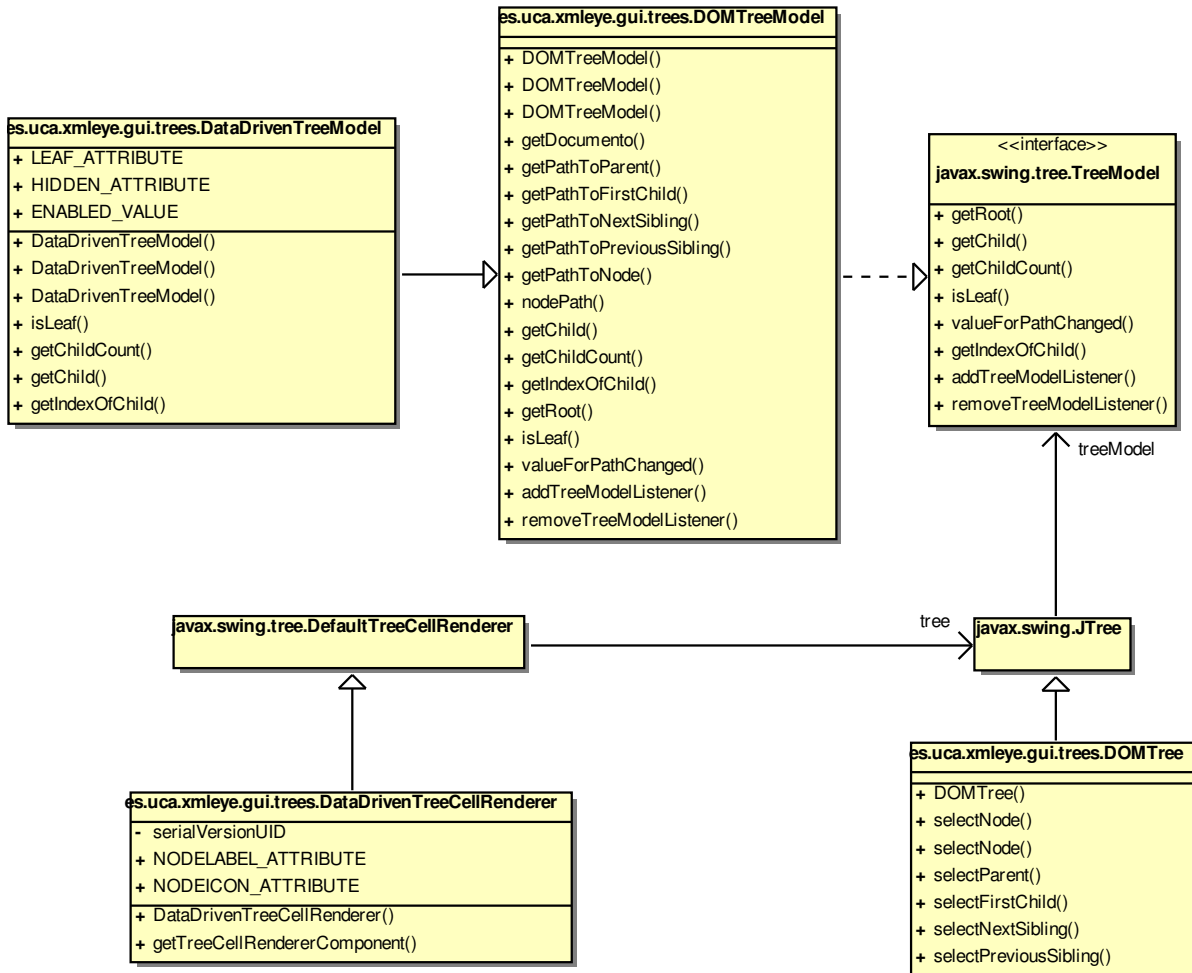


Figura 4.29.: Diagrama de clases de manejo de rboles

4. Desarrollo del proyecto

Es el caso usual de las acciones disponibles mediante la barra de herramientas, un acceso de teclado y un elemento del men, por ejemplo: no importa exactamente quin lanza el evento, slo qu accin se lanza.

En Swing, la interfaz a cumplir es *Action*, que permite no slo unificar la orden a ejecutar, sino tambin parte de su visualizacin. Podemos asignar un acelerador de teclado, una descripcin corta y un mnemnico a una accin, y al crear un botn o elemento de men a partir de ella, stos se inicializarn con los valores correctos.

4.6. Implementacin

4.6.1. Perl

Se usaron la obra [?] y la web [?] como referencias para los aspectos de divisin en mdulos y la implementacin en Perl del paradigma orientado a objetos.

Varios detalles de implementacin de ciertos patrones y otras prcticas recomendadas para Perl se obtuvieron de [?].

Perl y el paradigma OO

No hay que confundir el hecho de emplear el paradigma orientado a objetos con el uso de un lenguaje orientado a objetos.

Perl proporciona mecanismos para implementar la mayora de la funcionalidad necesaria para este paradigma, pero no proporciona un soporte directo a travs del lenguaje de muchas construcciones, como una sintaxis especfica para declaracin de clases con palabras reservadas para control de acceso, por ejemplo.

Otras veces ha de usarse una sintaxis algo ms compleja de lo usual, como en el caso de las variables de instancia.

Para implementar los mtodos, por ejemplo, Perl permite “bendecir” una referencia cualquiera con el nombre de un mdulo. A partir de ese momento, se busca la subrutina `metodo` dentro del mdulo con el que se bendijo `$variable` al usar esta sintaxis:

```
$variable->metodo(@args);
```

Un mdulo puede declarar a otros como base, que sern consultados mediante un recorrido primero en profundidad sobre el rbol de generalizaciones durante la bsqueda de un mtodo. Este mecanismo es el que implementa la herencia y los mtodos virtuales.

Internacionalizacin

La internacionalizacin de los mensajes se consigue en `ACL2::Procesador` a travs del mdulo `Locale::Maketext`, que permite usar una jerarquía de clases donde la clase base define la lgica común y los idiomas por omisión y sus hijas contienen un diccionario que realiza la localización para cada idioma (posiblemente especificando el país).

El uso es muy sencillo:

Listado 4.1: Ejemplo de uso de `Locale::Maketext`

```
# Obtenemos el manejador a la localizacin que ms se ajuste,
# subclase de ACL2::Localizacion, que es a su vez subclase
# de Locale::Maketext.
my $lh = ACL2::Localizacion->get_handle();

# Buscamos en el diccionario y sustituimos
# la cadena original: buscando 'Hola [_1]' se hallara
# 'Hello [_1]' y luego se sustituirá, dando 'Hello Pablo'.
print $lh->maketext("Hola [_1]", 'Pablo');
```

Dicho mdulo es técnicamente superior al conocido `Gettext`, por la flexibilidad que su enfoque orientado a objetos le aporta: se pueden definir funciones para manejar las formas plurales según el lenguaje, por ejemplo, simplemente redefiniendo un método en la clase correspondiente.

También numera los parámetros de las cadenas, permitiendo mayor flexibilidad que la usual cadena con parámetros para `printf`, que da resultados pobres en casos en los que haya que cambiar el orden de las palabras.

Distribucin mediante PAR

Distribuir `ACL2::Procesador` era bastante complejo e incómodo en su anterior versión. Se deseaba facilitar este aspecto en la mayor medida de lo posible, permitiendo instalar de forma automática las dependencias o empaquetarlas en un formato cómodo de utilizar.

Este problema no es nuevo a `ACL2::Procesador`: empezando por `YAXML::Reverse`, prácticamente todo mdulo sufre de lo. Existen muchas alternativas, como `perlcc` (retirado a partir de Perl 5.10), `Perl2Exe` o `Perl2App`, pero entre todas ellas destacan `PAR` y `PAR::Packer`.

De forma muy similar a los ficheros `.jar` de Java, `PAR::Packer` y su herramienta `pp` nos permiten crear ficheros `.par` con todo el código fuente de los conversores y sus dependencias que no forman parte de los módulos estándar de Perl.

4. Desarrollo del proyecto

Instalar estos conversores y sus hojas se convierte en algo tan sencillo como descargar el `.par` correspondiente a nuestra plataforma y descomprimirlo sobre el directorio raíz de XMLEye. Además, si nuestra aplicación es Perl puro, el `.par` generado puede ser usado en cualquier arquitectura y sistema operativo.

Si no queremos obligar a nuestros usuarios a tener que instalar un entorno Perl y los módulos `PAR` y `PAR::Packer` (cosa particularmente molesta en Windows), también podemos crear ejecutables monolíticos, específicos del sistema operativo y arquitectura bajo el que se creen, que incluyen hasta el propio intérprete de Perl con sus módulos estándar.

Estas imágenes no son tan grandes como se esperara: los `.par` para `YAXML::Reverse` y `ACL2::Procesador` no llegan a los 100KiB, y los ejecutables, una vez comprimidos, no alcanzan los 2MiB. Además, estas distribuciones comprimidas incluyen todos los descriptors de formato y hojas de usuario necesarias.

4.6.2. Java

Un texto til ha sido la referencia del lenguaje [?], escrita por los creadores de Java. La web [?] detalla además algunas prácticas recomendadas, funcionalidades no documentadas y soluciones temporales a fallos conocidos de Swing.

Un ejemplo es la disponibilidad no documentada de fuentes con anti-aliasing bajo la J2SE 5.0, o la necesidad de forzar manualmente un tamaño mínimo sobre los hijos de *JFrame*, un fallo documentado en la propia base de datos de Sun y corregido en la próxima J2SE 6.0 “Mustang”.

El uso del IDE Eclipse 3.3.2 ha sido fundamental gracias a su soporte para la realización sencilla de refactorizaciones a gran escala, pudiendo renombrar métodos y clases completas sin introducir errores en el código, entre otras cosas.

Controles HTML

Los controles para navegación HTML incluidos en Swing dejan bastante que desear a la hora de usar el teclado para navegar o seleccionar enlaces.

Para cubrir estas carencias, se especializa la clase *JEditorPane* en *Browser*.

Otros componentes tiles incluyen un observador de enlaces capaz de lanzar uno de los navegadores web del usuario (el navegador por defecto en Windows y Mac OS X) al pulsar enlaces con URL válidas.

Internacionalizacin

En el lado de Java, la internacionalizacin se obtiene a partir de los *ResourceBundles*, de los que hay dos tipos:

- *PropertyResourceBundles*, que acceden a ficheros `.properties` de fcil mantenimiento, al slo contener cadenas de texto.
- Especializaciones de *ListResourceBundle*, que pueden contener cualquier tipo de objeto.

La capa de aplicacin slo maneja mensajes de texto y usa el primer tipo, y la capa de presentacin usa el segundo tipo, al necesitar adems iconos y atajos de teclado.

La clase *ResourceBundle* de la biblioteca estndar de Java es la ocupada de hallar e instanciar el *PropertyResourceBundle* o *ListResourceBundle* que se ajuste mejor a la localizacin del usuario.

Componentes especializados Para facilitar la internacionalizacin, se han definido una serie de clases genricas que se inician a partir de un *ResourceBundle*.

En particular, se ha definido una especializacin de *AbstractAction* llamada *ResourceBackedAction*, que extrae sus valores de un *ResourceBundle* que sigue un esquema especfico de nombrado de claves.

La clase *AbstractAction* es la clase base de todas nuestras *Ordenes* (ver §4.5.5), que proporciona implementaciones por defecto para la mayora de los mtodos de la interfaz *Action*, a seguir por toda *Orden* de Swing.

Todas las rdenes usadas son clases internas de *WndMain*: las ms complejas tienen entidad propia, y las dems son animas. Puede verse un diagrama UML de las primeras en la figura 4.30 de la pgina 134.

Tambin hay mens (*ResourceBackedMenu*) y botones (*ResourceBackedButton*) que extraen sus valores de los *ResourceBundle*.

4.6.3. Hojas XSLT

Las propias hojas XSLT deben seguir una serie de convenciones en su cdigo y forma de operar. Estas convenciones se hallan documentadas en el apartado del manual de usuario dedicado a explicar cmo crear nuevas hojas XSLT.

4. Desarrollo del proyecto

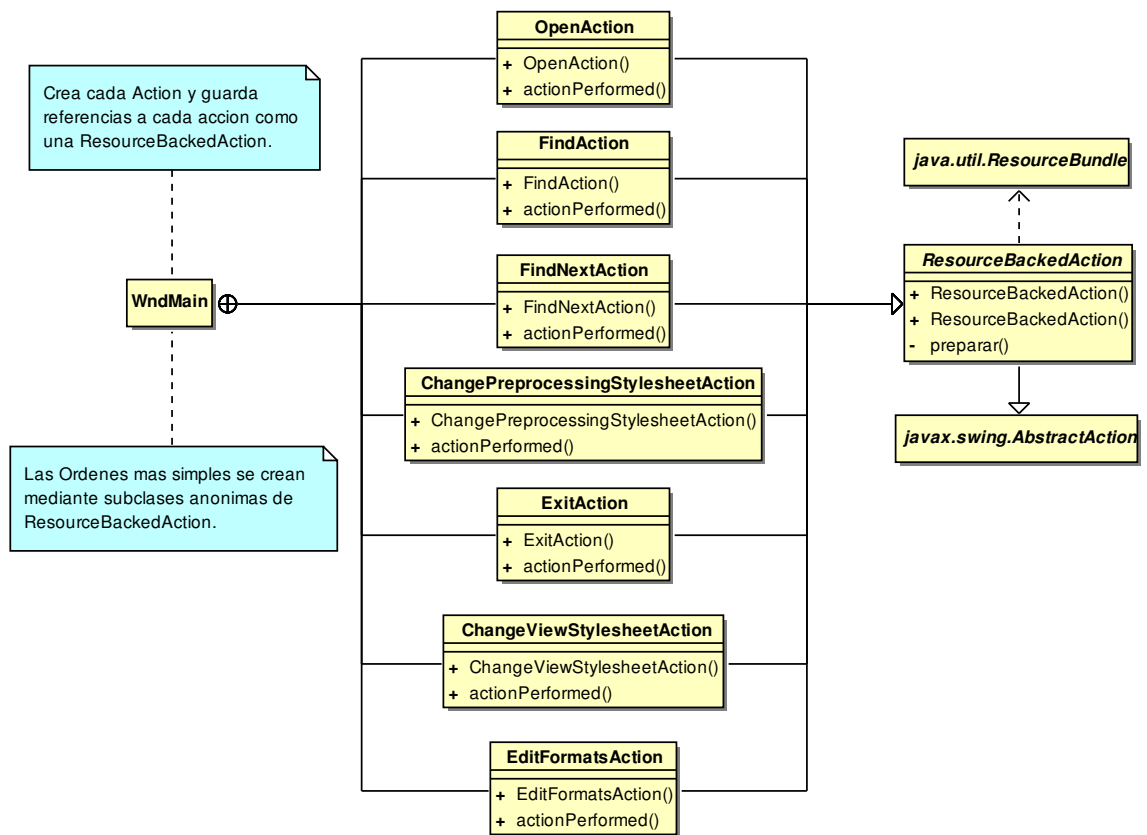


Figura 4.30.: Diagrama de clases de rdenes

4.7. Pruebas y validación

Se dar una breve introducción a los conceptos relacionados con las pruebas de software en XP y a continuación se describir el plan de pruebas, junto con la especificación de los casos de prueba y su procedimiento de ejecución.

4.7.1. Pruebas en XP

La metodología XP realiza cuatro tipos de pruebas:

Pruebas de unidad Las pruebas de unidad comprueban de forma automática la funcionalidad de un conjunto reducido y cohesivo de clases. Son escritas por los propios programadores, siguiendo la práctica de XP de la programación basada en pruebas.

Pruebas de aceptación Las pruebas de aceptación son escritas por el propio cliente con asistencia de los miembros especializados en pruebas del equipo de desarrollo.

Más que probar el funcionamiento de un par de clases o de un método determinado, lo que se intenta probar es la implementación satisfactoria de una historia de usuario.

Así, las pruebas de aceptación suelen describirse mediante una serie de pasos en los cuales se utiliza el sistema completo para llevar a cabo una tarea. Cada paso tiene un resultado esperado asociado.

Pruebas de integración En XP, las pruebas de integración se realizan de forma constante. Aunque en mi caso no son importantes al ser un proyecto individual, XP requiere que todo cambio hecho en una sesión de programación (en parejas, por supuesto) sea integrado inmediatamente en el repositorio central.

Por supuesto, esto implica que dichas modificaciones deben de haber pasado antes por todas las pruebas con éxito, asegurándonos de que no se han introducido nuevos fallos en otras partes del programa inadvertidamente.

Pruebas de implantación Una de las prácticas recomendadas (que no obligatorias) de XP es la implantación incremental. Desde el inicio de un proyecto, el sistema debe de implantarse en un entorno similar al real, posiblemente a menor escala, y comprobarse su correcto funcionamiento.

Estas pruebas, sin embargo, no son especialmente importantes para una aplicación de escritorio como la de este proyecto.

4. Desarrollo del proyecto

4.7.2. Plan de pruebas

Alcance

Se determinó que, con las limitaciones de tiempo establecidas, no se podían realizar pruebas de todas y cada una de las partes de XMLEye, YAXML::Reverse y ACL2::Procesador.

Se decidió implementar pruebas automáticas sobre los elementos fundamentales y utilizar pruebas de aceptación manuales sobre lo demás. En particular, se descartaron aquellas clases cuya funcionalidad dependa en su mayoría de bibliotecas externas, como la interfaz gráfica o el controlador de transformaciones XSLT.

En un futuro, se espera reemplazar la mayor parte de las pruebas sobre la interfaz por pruebas automáticas, utilizando los nuevos modelos de presentación. Actualmente ya se ha comenzado esta labor sobre el diálogo de gestión de formatos.

Tiempo y lugar

Además del propio uso frecuente de los conversores y XMLEye, mis directores de proyecto contribuyeron a la ejecución de pruebas de aceptación de manera informal a lo largo de la ejecución del PFC.

Las pruebas se han realizado en todo momento durante el desarrollo del PFC (especialmente las automatizadas), siguiendo la práctica “Flujo” de XP. Al final de cada iteración, se dedicó un tiempo adicional para la realización de pruebas de aceptación manuales.

Naturaleza de las pruebas

La totalidad de las pruebas usadas son pruebas de caja negra, por ser fáciles de mantener a pesar de cambios en la implementación.

4.7.3. Diseño de pruebas

A primera vista, se puede determinar que existen cuatro elementos fundamentales a probar:

- ACL2::Procesador: ¿procesa correctamente la salida de ACL2? ¿Se obtiene un documento que corresponda sin pérdidas al original y que siga el formato de la DTD? ¿Se trata de un módulo Perl correctamente estructurado y documentado?

- **YAXML::Reverse**: ¿analiza sin errores los ficheros de entrada? ¿Realiza una conversin sin prdidas ni modificaciones de la informacin del fichero fuente YAML? Al igual que en el caso anterior, ¿tiene un nivel mnimo de calidad como mdulo Perl?
- El visor **XMLEye**: ¿se validan las entradas? ¿Se reacciona bien ante acciones no vlidas del usuario? ¿Obtiene el usuario los resultados esperados?
- La integracin entre los conversores y **XMLEye**: ¿se realiza con xito? ¿Se capturan y muestran correctamente los fallos?

En base a esto, cada parte requiere una combinacin determinada de varios tipos de pruebas:

ACL2::Procesador El desarrollo de este conversor se halla guiado por los enunciados Lisp y las salidas de ACL2 que ha de procesar. Por lo pronto, resulta natural estructurar las pruebas simplemente alrededor de dichos casos.

Se puede implementar un mecanismo de pruebas de regresin usando estos enunciados: una vez se haya validado una salida determinada como correcta, se puede mantener dicha salida y comparar las salidas posteriores con ella, notificando cualquier cambio de inters de forma automtica.

En un futuro, cuando el alcance del procesador se ample a demostraciones de mayor envergadura, se aadirn pruebas de unidad sobre las clases de mayor importancia.

YAXML::Reverse De forma similar al caso anterior, el desarrollo y revisin de este conversor se apoya sobre una serie de entradas conocidas: cuando alguna entrada falla, se aade como caso de prueba y se revisa **YAXML::Reverse** hasta que vuelven a superarse todas las pruebas.

A diferencia de **ACL2::Procesador**, no necesitamos registrar las salidas anteriores “buenas” y compararlas con las actuales, sino que podemos usar el ciclo **YAML** → **XML** → **YAML** implementado y establecer directamente la equivalencia semntica de los dos ficheros **YAML** original y final. Este proceso se halla descrito a mayor nivel de detalle en §4.5.3 (pgina 112).

XMLEye Se podra dividir **XMLEye** en tres partes:

1. La interfaz de usuario: Las limitaciones de tiempo, combinadas con la dificultad y la escasez en herramientas automatizadas de prueba de las interfaces grficas obliga a usar pruebas de aceptacin manuales por lo pronto. En un futuro cercano, el nuevo diseo de la capa de presentacin basado en modelos de presentacin permitir automatizar estas pruebas, limitndonos a enviar notificaciones de gestos a los modelos de presentacin.

Sin embargo, los componentes relacionados con los rboles **XML** son cruciales para la aplicacin, y deben de incluir sin falta pruebas automatizadas

4. Desarrollo del proyecto

sobre las propiedades que deben cumplir y el comportamiento que deben seguir.

2. Las hojas de usuario: aunque su realizacin sera tcnicamente factible, las pruebas de unidad de estas hojas no podran mantenerse estables, puesto que el formato de salida no lo es: al ser XHTML, est sujeto a muchos cambios y retoques a lo largo del tiempo. Se puede decir lo mismo acerca del paso de preprocesado.
3. La capa de aplicacin: esta capa provee los servicios bsicos sobre los que se apoya la capa de presentacin. No probaremos aquellas partes que nicamente dependan de la capa de servicios tcnicos, como el funcionamiento de las transformaciones XSLT, por ejemplo, pero s los servicios de nivel superior.

Por ejemplo, habr que depurar los repositorios de hojas de estilo y de descriptores de formatos, la validacin de los campos de los descriptores de formato, la correspondencia entre rutas XPath y nodos de un rbol DOM XML, o las extensiones XPath, entre otras cosas.

Integracin La integracin es otro aspecto difcil de probar automticamente, dado que depende en gran medida del entorno empleado por el usuario y de la configuracin que haya establecido.

Sin embargo, se presta bien a pruebas de aceptacin manuales sobre una configuracin realizada de antemano. Se habr de probar que la integracin reacciona bien a errores de los programas y de su invocacin, fallando cuando debe y permitiendo una recuperacin rpida.

4.7.4. Especificacin de los casos de prueba

ACL2::Procesador

Cada una de las fuentes Lisp y su salida correspondiente a filtrar por `ACL2::Procesador` segn las historias de usuario planteadas es utilizada como una prueba de regresin semi-automtica.

Describiremos brevemente el contenido de las fuentes Lisp usadas como casos de prueba. Por claridad, usaremos notacin infija. Todos estos enunciados se encuentran bajo el subdirectorio `t/testInputs` de `ACL2::Procesador`.

Durante las pruebas se desactiva todo almacenamiento de resultados intermedios y se obliga a actualizar el grafo completo de dependencias, para evitar que en ejecuciones distintas de las pruebas se consigan resultados distintos.

■ triple-rev

`triple-rev` fue el objetivo de la primera iteracin. En esta demostracin se comprueba para una funcin de inversin de listas Lisp `rev` que

$$\text{rev}(\text{rev}(\text{rev}(x))) = \text{rev}(x)$$

para toda lista que le pasemos, sea cual sea su estructura.

■ equal-app

En esta demostracin se comprueban ciertas propiedades acerca de las funciones `app` para concatenacin de listas, `dup` para duplicacin de cada elemento y `properp` para comprobacin de listas propias, terminadas mediante `nil`, como `(a b)` o `(a . (b))` pero no `(a . b)`.

- Asociatividad: $\text{app}(\text{app}(a, b), c) = \text{app}(a, \text{app}(b, c))$
- Distributividad de la duplicacin de cada elemento respecto de la concatenacin: $\text{app}(\text{dup}(a), \text{dup}(b)) = \text{dup}(\text{app}(a, b))$
- Toda lista a la que se aade `nil` es propia: $\text{properp}(\text{app}(a, \text{nil}))$.

■ fallo-defun

Realmente, este guin no trata de probar nada: est diseado para ver si `ACL2 : -` Procesador es capaz de tratar los fallos en las demostraciones de los `defun`.

En primer lugar, declaramos una funcin de clculo de factorial correcta, que siempre da un resultado correcto y siempre para.

La siguiente versin errneamente usa `=` en vez de `zp`, y por lo tanto no para nunca para otra cosa que no sea un natural. `zp` es verdadero para toda cosa que no sea un nmero mayor que cero: listas, tomos, enteros no positivos (incluyendo el cero, claro), etc. Sin embargo, `=` slo es verdadero si lo que se pasa es cero, por lo que ante un nmero negativo, por ejemplo, nunca llegara al caso base y se quedara siempre en el caso recursivo.

En la tercera versin, no hay caso base: se ha reemplazado por una llamada recursiva a s mismo. Y en la cuarta y ltima, no se reduce el tamao de la entrada, imposibilitando tambin la parada.

■ hanoi

No es uno, sino realmente cuatro casos de prueba, al tratarse de un proyecto de `ACL2` con tres ficheros Lisp. Tenemos un caso de prueba por fichero y otro caso de prueba ms para la deteccin del grafo que forman y su correcto uso:

- `hanoi-use.lisp` es el fichero raz del proyecto. Utiliza la orden `include-book` para acceder a la definicin de la funcin que resuelve el problema de las torres de Hanoi, `hanoi : : hanoi`, que se halla en el libro `BOOKS/HANOI`.

4. Desarrollo del proyecto

- `books/hanoi.acl2` es el fichero Lisp que sirve para certificar los contenidos del libro HANOI dentro de su mundo lgico inicial. Se podra haber llamado `cert.acl2` si se hubiera deseado emplear para cualquier libro dentro de `books`, siguiendo las convenciones usuales de manejo de libros de ACL2 [?], pero se ha preferido esta forma especifica, para cuando se aadan ms libros posteriormente.
- `books/hanoi.lisp` es el propio libro con las definiciones necesarias, que asume que se halla dentro del mundo lgico inicial definido por el fichero de certificacin anterior. Define las funciones `move`, que genera una instruccin del tipo “mover un disco de la pila A a la pila B” y `hanoi`, que resuelve el problema de las Torres de Hanoi para n discos. Tambin incluye un teorema, `len-append`, que establece que la longitud de la concatenacin de dos listas es la suma de sus longitudes.

■ `mapnil`

En este caso, se demuestra la conmutatividad entre la duplicacin de los elementos mediante `dup` y la sustitucin de todo elemento de una lista Lisp por `nil` mediante `mapnil`.

■ `memp`

Dada una funcin `memp` que comprueba la pertenencia de un elemento a una lista y otra funcin `app` que concatena dos listas, se demuestra que

$$memp(e, app(a, b)) \equiv mem(e, a) \vee mem(e, b),$$

es decir, que todo elemento de la concatenacin de las dos listas pertenecer a una u a otra.

■ `miscDefs`

Comprobamos que `ACL2::Procesador` es capaz de filtrar una diversidad de funciones, como bsquedas en diccionarios, implementacin de aritmtica mediante el uso de la longitud de las listas, recoleccin de elementos nicos, potenciacin, etc.

Algunos ejemplos:

- `mapnil(x)` convierte todos los elementos a `nil`.
- `add(x, y)` realiza la suma mediante el nmero de elementos de ambas listas. Es decir, implementamos la suma creando una lista de 5 elementos cuando nos dan una de 2 y otra de 3.
- `app(a, b)` Concatenamos dos listas.
- `mem(e, x)` Devuelve el primer par de la lista tal que el primer elemento sea `e`, o `nil`.

- *lonesomep*(*e*, *lst*) Comprobamos que *e* es nico en la lista.
- *collect* – *lonesomep*(*a*, *b*) Recoge los elementos de la lista *a* que son nicos en la lista *b*.
- *mult*(*x*, *y*) Multiplica a travs de longitudes de listas. Si le damos una lista con 2 elementos y otra con 3, nos crea una lista con 6.
- *fact*(*n*, *a*) Calcula el productorio en aritmtica sobre longitud de listas de las sublistas de la lista que le pasemos.
- *fact1*(*n*, *a*) Versin recursiva final de *fact*.
- *foundp*(*x*, *a*) Busca una clave dentro de un diccionario Lisp. Si el diccionario es la lista Lisp de la forma ((*a* 2) (*b* 3)), sus claves son *a* y *b*.

■ suma

Este caso comprueba que se tratan de forma correcta los *defthm* que no requieren induccin alguna. Aqu, ACL2 slo tiene que aplicar sus reglas predefinidas sobre la aritmtica decimal para ver que $2 + 2 = 4$ o que $a + b + c = b + c + a$.

Tambin se comprueba que se manejan bien los elementos de listas Lisp con la sintaxis `|texto|`.

■ swaptree

Se define una funcin *swaptree* que invierte dos rboles Lisp, y se demuestra que es idempotente.

■ tail-rev

Se define una implementacin trivial de la inversin de una lista *rev* sabiendo que es correcta. Hecho eso, definimos una implementacin recursiva final obtenida por la transformacin mediante desplegado-plegado *rev1*. Ahora comprobamos que est bien hecha, es decir, que da los mismos resultados que la versin original.

■ treecopy

Es parecida a *swaptree*, pero en este caso lo que se hace es copiar un rbol Lisp, y comprobar que dicha funcin repite siempre la entrada que se le suministra.

■ triple-rev-misspell

Este caso de prueba es una modificacin de *triple-rev* con un fallo en el nombre de un parmetro formal introducido intencionadamente, que origina una demostracin fallida de la longitud suficiente como para ser de inters. Adems, permite comprobar que se tratan bien los errores de *defthm*.

4. Desarrollo del proyecto

- `tutorialWebACL2`

Este otro caso de prueba es el proyecto de un solo fichero ms completo de todos. En este caso, se desea probar que un algoritmo para la inversin de una lista Lisp que slo hace uso de las funciones predefinidas *endp* (predicado para detectar el final de una lista), *cons* (creacin de un par Lisp), *car* (extraccin de la cabecera) y *cdr* (extraccin de la cola) es equivalente al algoritmo trivial.

El proceso es algo ms complicado que los ejemplos anteriores, y puede verse en [?], donde se usa “El Mtodo”, acumulndose teoremas a demostrar en forma de una pila, donde un teorema puede requerir para su demostracin verificar otros teoremas o lemas.

Se hacen uso de algunos elementos ms avanzados, como `thm`, que lanza una demostracin pero no guarda sus resultados en el mundo lgico de ACL2, `local` que evita que las reglas generadas por un evento sean accesibles fuera de un `encapsulate` o libro, y `encapsulate`, que permite realizar demostraciones controlando qu reglas se producen y cul es la signatura de las funciones definidas.

Las pruebas de regresin comparan los rboles XML de los resultados esperados con los obtenidos, empleando el mdulo `XML::SemanticDiff`. Las diferencias detectadas son filtradas por un predicado que determinan si constituyen una regresin o no. Actualmente, algunas de las diferencias ignoradas son:

- Cambios en los tiempos de ejecucin.
- Cambios en la versin de ACL2.
- Cambios de mayculas y minculas y barras en nombres de fichero en Windows.
- Mensajes de compilacin durante la certificacin de un libro (dependen del compilador y sistema operativo usado).

Las pruebas de regresin no son las nicas realizadas. Integradas dentro del marco de pruebas creado por el mdulo `ExtUtils::MakeMaker` se hallan tambin algunas pruebas de carcter ms general:

- El mdulo principal debe de poder cargarse con xito. Esto nos asegura de que no nos hayamos olvidado de instalar o referenciar a algn mdulo importante en nuestro cdigo.
- Se debe de haber retirado el texto intil de las plantillas inicialmente generadas por `h2xs`, la herramienta ocupada de crear el esqueleto bsico de todo mdulo Perl.
- Todos los ficheros fuente del mdulo Perl han de incluir el aviso de la licencia GPL.

YAXML::Reverse

Como otro mdulo Perl ms, `YAXML::Reverse` emplea el mismo marco de pruebas y las pruebas de carcter general de `ACL2::Procesador`. Este mdulo usa tambin pruebas de regresin, pero son completamente automticas en este caso, como ya se explic anteriormente. Ciertas pruebas de regresin son ms bien para la versin refinada de YAXML implementada para cerrar el ciclo `YAML → XML → YAML` que para `YAXML::Reverse` en s.

Algunos casos de prueba se corresponden con documentos reales usados por herramientas conocidas, y algunos slo depuran ciertos aspectos esenciales. Existe cierto solapamiento entre ambos tipos. Los ficheros `.yaml` utilizados como casos de prueba son:

- `djangoJSON`

Este es un ejemplo de un volcado de una base de datos realizado por el entorno de desarrollo de aplicaciones Web Django (<http://www.djangoproject.com/>). Tiene la particularidad de usar JSON, un subconjunto de YAML 1.1, motivando el cambio del mdulo `YAML::Syck` a `YAML::XS`.

- `djangoJSON_blockIndicatorTest`

Otro volcado de Django en el cual se hace patente la necesidad en YAXML de tener cuidado con el espaciado en blanco y no cambiarlo inadvertidamente. En particular, hay que tener cuidado con el ltimo salto de lnea final, evitando introducirlo cuando originalmente no estaba, utilizando el indicador “|-” en el estilo de bloque para escalares.

- `firefoxBookmarks`

Una copia de seguridad de los marcadores de Firefox 3 realizada tambin en JSON, que en este caso tena la complicacin de emplear caracteres de UTF-8, en particular ideogramas japoneses. Se identificaron y resolvieron diversas cuestiones relacionadas gracias a este ejemplo.

- `fiveMinutes1`

Es uno de los casos ms bsicos: un flujo de 4 documentos que emplean secuencias de YAML. Es compatible con YAML 1.0 en adelante, al igual que todos los casos de prueba que vienen a continuacin.

- `fiveMinutes2`

Este ejemplo muestra ejemplos de vectores asociativos de YAML 1.0, y fue el primer caso de prueba que mostraba la necesidad de rodear las limitaciones de XML respecto a los identificadores vlidos para una etiqueta utilizando los elementos `_key` y `_value`.

4. Desarrollo del proyecto

- `fiveMinutes3`

Desarrollando sobre el ejemplo anterior, este caso de prueba también usa claves de vectores asociativos no directamente representables en XML, pero además una de ellas requiere un especial cuidado a devolverla a YAML, o producirá un error de sintaxis, al contener la secuencia “: ”.

- `fiveMinutes4`

Este caso de prueba utiliza los estilos literal y de líneas reunidas para los escalares. En el estilo literal el espaciado no es modificado excepto por el indentado de cada línea. En el estilo de líneas reunidas, todas las palabras separadas por únicamente un carácter de espaciado (como un espacio o un salto de línea) son reunidas en una sola línea.

- `fiveMinutes5`

El último ejemplo de la serie [?] ilustra cómo pueden escribirse también las colecciones con distintos estilos, ya sea de bloque, con un elemento por línea, o de flujo, que permite definir más de un elemento en cada línea.

- `invoice`

Este caso de prueba es uno de los ejemplos incluidos en la especificación YAML [?], y precisamente el que ilustra el uso de anclas y alias para evitar tener que repetir el contenido común a dos nodos.

- `mappingAnchoredSequence`

Aquí probamos a transformar un documento en que el documento anclado no es un mapa, sino una secuencia.

- `podExample`

Este ejemplo tan sencillo es el usado en la documentación POD de `YAML::Reverse`, para poder garantizar que funcione.

- `yaxmlReverseMETA`

Se trata del fichero `META.yaml` de `YAML::Reverse`. Estos ficheros son utilizados por diversas herramientas automáticas, como las que utiliza el CPAN, para obtener información de autoría y dependencias.

Visor e integración

Pruebas de unidad Las pruebas de unidad definidas para cada clase comprueban una serie de propiedades acerca de su comportamiento. En lugar de dar una especificación exacta de cada prueba (que solo será repetir el código), listaremos las propiedades que verifican.

Estas pruebas, basadas en el marco JUnit, se ejecutan de forma automática y son un objetivo más dentro del fichero de tareas Ant.

Toda clase de pruebas tiene el nombre formado por el nombre de la clase cuyo funcionamiento comprueba y a continuación el sufijo “Test”.

CommandBuilderTest Se comprueba que se realiza la sustitución de las claves y la división en argumentos correctamente, independientemente de que los argumentos contengan espacios, mientras tengan comillas alrededor de la orden original.

DataDrivenTreeModelTest Todo nodo del documento XML oculto mediante el atributo correspondiente debe ser inaccesible a través de un recorrido exhaustivo.

Además, el modelo ha de informar de que todo nodo con el atributo requerido para la falsa poda activado no tiene hijos, y estos deben ser completamente inaccesibles desde dicho modelo.

DOMTreeModelTest El modelo ofrecido por dicha clase debe de ser el mismo que ofrece el árbol DOM original del documento, limitándose a los nodos de tipo elemento.

DOMTreeTest Debe de poderse realizar una navegación completa a través del árbol, ignorando toda petición no válida, como intentar ir al padre de la raíz.

ExtensionFileFilterTest Se prueba que se añade automáticamente la extensión al nombre de fichero si no está presente, que se conserva si ya está, y que se conserva el nombre de fichero si no se usa un filtro *FiltroExtensión*.

FormatDescriptorRepositoryTest Utilizando un repositorio a dos niveles con el descriptor de formato `xml.format` incluido con XMLEye, se comprueba que se inicializa correctamente. También se prueba a cambiar un descriptor, guardarlo y volver a cargar, para ver si los cambios se han realizado.

A continuación se hacen pruebas sobre la capacidad de localizar las cadenas de un descriptor de formato, de que los cambios realizados a un descriptor vayan al repositorio de mayor prioridad, y que se puedan restaurar las opciones originales de un formato.

Finalmente, se comprueba que los mutadores de todo descriptor de formato lancen ciertas excepciones al pasarles algún valor no válido.

4. Desarrollo del proyecto

SerializedPreferencesManagerTest Se comprueba que se puede crear un fichero de preferencias nuevo, y que inicialmente est vaco. Debe adem poder guardar de forma persistente escalares y listas.

StylesheetRepositoryTest Se verifica que las hojas incluidas en XMLEye pueden localizarse sin problemas dentro del repositorio, que se hallan debidamente configuradas y que su cdigo XSLT compila sin problemas.

VectorNodeListTest Se prueba que el mtodo `invertir` realmente cumple su tarea, y que la inversin es idempotente.

WndFormatsPresentationModelTest Se comprueba que cuando el usuario seleccione un elemento, cambie el nombre y confirme los cambios estos se realicen de forma correcta y persistente.

XHTMLSearchKeyHighlighterTest Al destacar una clave dentro de un documento XHTML los elementos XHTML deben de conservarse intactos. No debe modificarse otra cosa que el contenido del documento en s, es decir, el fragmento situado dentro del elemento `body`.

Deben adem de manejarse bien los casos en que la clave a marcar est justo antes o despues de un elemento XHTML. Otros casos importantes a tratar incluyen el manejo correcto de marcado de slo palabras completas o sin tener en cuenta mayculas y minculas.

XPathExtensionsTest Se comprueba que se obtienen coincidencias de la forma esperada para varias combinaciones posibles de tipos de clave y cadena de bsqueda.

As, la clave de bsqueda podra componerse slo de caracteres alfanumricos. Tambin podra contener espacios o caracteres no alfanumricos. Se podra dar el caso de que el usuario introdujera algn carcter que debiera escaparse para no ser interpretado como un metacarcter en una expresin regular.

El texto en el que buscar podra contener espacios iniciales o finales, o saltos de lnea.

Tambin podra darse el caso en que la clave y el texto fueran iguales, o que alguno de los dos estuviera vaco.

XPathPathManagerTest Se comprueba que la ruta XPath generada de cualquier nodo del rbol DOM de un determinado documento XML es correcta, y que la obtencin de un nodo a partir de su ruta XPath se realiza tambin con xito.

Pruebas de aceptacin Pasaremos a establecer una prueba de aceptacin para cada una de las historias de usuario acumuladas desde las primeras versiones de XMLEye hasta ahora. Dichas pruebas se ejecutan de forma manual.

- “Visualizar un documento XML arbitrario en forma de rbol, permitiendo expandir y contraer los nodos y buscar informacin.”

Cuadro 4.1, pgina 148.

- “El visor no debera saber absolutamente nada de ACL2. Nada en su interfaz grfica ni en su implementacin nos debe recordar que podemos trabajar con ACL2.”

Cuadro 4.2, pgina 149. Se asume en esta historia que XMLEye ha sido instalado desde cero, sin ningn otro aadido.

- “Aadir soporte para visualizar varios ficheros a la vez y enlazarlos entre s.”

Cuadro 4.3, pgina 150. Se asume que se ha instalado ACL2, `ACL2::Procesador`, sus hojas de usuario y su descriptor de formato de la manera descrita en el manual de usuario.

- “Proporcionar una lista de documentos recientes.”

Cuadro 4.4, pgina 151.

- “Integrar el visor y el editor preferido del usuario.”

Cuadro 4.5, pgina 152.

- “Vigilar directamente el fichero fuente en caso de cambios, en vez de solamente cuando se cierra el editor. As, si uno abre el editor, hace un par de cambios y guarda (sin cerrar el editor), tambin se actualizar. En caso de que se desactive temporalmente la reimportacin automtica, tambin debera funcionar correctamente si en ese intervalo se hicieron cambios.”

Cuadro 4.6, pgina 153.

- “Guardar las preferencias automticamente al cerrar el programa.”

Cuadro 4.7, pgina 153.

- “Visualizar un fichero Lisp integrando XMLEye con `ACL2::Procesador`.”

Cuadro 4.8, pgina 154. Se asume que se ha instalado ACL2, `ACL2::Procesador`, sus hojas de usuario y su descriptor de formato de la manera descrita en el manual de usuario.

4. Desarrollo del proyecto

Paso seguido	Resultado esperado
1. El usuario solicita la apertura de un fichero.	Se muestra el dialogo de eleccin de ficheros.
2. El usuario elige un fichero XML.	Tras la compilacin de las hojas y el pre-procesado, se muestra el primer nodo del documento.
3. El usuario cambia la hoja de preprocesado a <code>xml</code> .	El rbol del documento muestra el rbol XML original.
4. El usuario cambia la hoja de visualizacin a <code>xml</code> .	La visualizacin muestra todos los atributos, ancestros e hijos.
5. El usuario cambia la hoja de visualizacin a <code>xmlSource</code> .	La visualizacin muestra el cdigo XML completo de cada nodo seleccionado tras su preprocesado.
6. El usuario navega por el rbol, usando cada una de las opciones disponibles de navegacin [...]	La visualizacin se actualiza en cada cambio de nodo, mostrando la informacin correcta.
7. El usuario solicita expandir el rbol.	El rbol del documento se expande en su totalidad.
8. El usuario solicita contraer el rbol.	El rbol del documento se contrae, y se muestra solamente el primer nodo del documento.
9. El usuario realiza una bsqueda de una clave existente en el documento.	Se muestra el primer resultado, con el marcado de la clave de bsqueda en la visualizacin.
10. El usuario pulsa de nuevo en el botn de Buscar del dialogo o utiliza el elemento del men Navegar "Buscar Siguiente".	Se muestran el resto de los resultados.
11. El usuario lanza de nuevo la bsqueda al llegar al ltimo resultado.	Se muestra de nuevo el primer resultado.
12. El usuario prueba con el resto de las opciones de filtrado en secuencia y compara con los resultados anteriores [...]	Se comprueba el filtrado del conjunto de resultados obtenido anteriormente.

Cuadro 4.1.: Prueba de aceptacin para "Visualizar XML genrico"

Paso seguido	Resultado esperado
1. El usuario abre XMLEye.	Se muestra la ventana principal.
2. El usuario abre un documento XML cualquiera.	Se muestra su primer nodo.
3. El usuario comprueba la lista de hojas de preprocesado disponibles.	Slo se dispone de la hoja <code>xml</code> .
4. El usuario comprueba la lista de hojas de visualización disponibles.	Slo se dispone de las hojas <code>xml</code> y <code>xmlSource</code> .
5. El usuario intenta abrir otro documento, y prueba a desplegar el filtro de tipos de documento.	Slo se dispone de las opciones “Todos los ficheros” o “Fichero XML normal”.
6. El usuario cancela su acción y abre el diálogo de gestión de formatos.	Se muestra el diálogo con un nuevo formato: “Fichero XML normal”.

Cuadro 4.2.: Prueba de aceptación para “Hacer a XMLEye independiente de ACL2”

- “Implementar la hoja `summaries`, que rodee el árbol y deje solamente los resúmenes, y la hoja `reverse`, que invierta el orden de los eventos, para mostrar primero las conclusiones y luego sus antecedentes.”

Cuadro 4.9, página 154. Se asume que se ha instalado `ACL2::Procesador`, su descriptor de formato y sus hojas de usuario tal y como se describe en el manual de usuario.

- “Añadir información acerca del uso de una meta, como sus dependencias inversas.”

Cuadro 4.10, página 155. Se asume que `ACL2::Procesador` se halla debidamente instalado.

- “Abrir documentos YAML en XMLEye sin que suponga una pérdida o alteración de la información disponible.”

Cuadro 4.11, página 155. Se asume que se ha instalado `YAXML::Reverse` con su descriptor de formato y hojas de usuario debidamente.

4. Desarrollo del proyecto

Paso seguido	Resultado esperado
1. El usuario abre un documento XML cualquiera.	Se muestra su primer nodo.
2. El usuario cambia las hojas de visualizacin y preprocesado a las hojas bsicas xml.	Se muestra el documento XML tal y como est, listando en cada nodo sus atributos y dando las cabeceras y pies apropiados.
3. El usuario abre el caso de prueba <code>hanoi-use.lisp</code> de <code>ACL2::Procesador</code> .	Se muestra su primer nodo en una pestaa aparte, con las mismas hojas que en la primera pestaa.
4. El usuario cambia las hojas de visualizacin y preprocesado a las hojas <code>ppACL2</code> .	Se muestran las rdenes <code>include-book</code> y <code>defthm</code> que conforman al fichero.
5. El usuario selecciona el sumario del <code>defthm</code> .	Aparecen enlaces a <code>HANOI::HANOI</code> entre las reglas usadas.
6. El usuario activa el enlace a <code>HANOI::HANOI</code> .	Se abre <code>hanoi.acl2</code> en una pestaa, con el nodo en cuestin seleccionado.
7. El usuario vuelve a <code>hanoi-use.lisp</code> , y esta vez activa <code>HANOI::MOVE</code> .	Se cambia de nuevo a la pestaa de <code>hanoi.acl2</code> , movindose al nodo seleccionado.
8. El usuario cambia a la pestaa del documento XML.	El documento mantiene la seleccin anterior y utiliza las hojas de preprocesado y visualizacin xml.

Cuadro 4.3.: Prueba de aceptacin para “Visualizar y enlazar varios documentos entre s.”

Paso seguido	Resultado esperado
1. El usuario abre un documento XML no presente en el historial.	Se muestra dicho documento y aparece su entrada en el historial.
2. El usuario cierra XMLEye.	Se guarda el historial de documentos recientes.
3. El usuario vuelve a ejecutar el programa y examina el historial.	Se sigue mostrando entre sus entradas el anterior fichero abierto.
4. El usuario selecciona dicha entrada.	Se abre de nuevo el documento.
5. El usuario cierra XMLEye, cambia al documento de localización y vuelve a ejecutar XMLEye.	Se sigue listando al fichero en su historial.
6. El usuario selecciona la entrada anterior.	Se informa que el fichero no existe y elimina la entrada del historial.

Cuadro 4.4.: Prueba de aceptación para “Historial de documentos recientes”

4. Desarrollo del proyecto

Paso seguido	Resultado esperado
1. El usuario abre un documento XML genrico.	El sistema muestra dicho documento segn las hojas actuales.
2. El usuario solicita abrir el dilogo de gestin de formatos admitidos.	Se abre el dilogo de gestin de formatos admitidos, incluyendo al menos al elemento “Fichero XML normal”. No se puede acceder al formulario principal.
3. El usuario comprueba el editor establecido y cierra el dilogo.	Se cierra el dilogo, volviendo a poder acceder al formulario principal.
4. El usuario solicita la edicin del documento.	Se lanza el editor que antes estaba especificado sobre el fichero actual. La interfaz grfica puede seguir usndose en paralelo sin problemas.
5. El usuario cierra el editor, utiliza el dilogo de formatos admitidos para cambiar la orden a otro editor existente en el sistema y confirma sus cambios.	Los cambios son guardados.
6. El usuario solicita de nuevo la edicin del documento.	Se lanza el editor especificado anteriormente, y la interfaz grfica se puede seguir usando sin problemas.

Cuadro 4.5.: Prueba de aceptacin para “Integrar editor”

Paso seguido	Resultado esperado
1. El usuario abre un documento XML cualquiera.	El sistema muestra dicho documento según las hojas actuales.
2. El usuario solicita la edición del documento.	Se abre el editor registrado por el usuario para el formato empleado.
3. El usuario activa la actualización automática en las preferencias.	La entrada del menú pasa a estar marcada como activa.
4. El usuario cambia el contenido del documento, manteniendo su corrección sintáctica, y guarda los cambios.	El documento se vuelve a abrir, y los cambios se reflejan en él.
5. El usuario desactiva la actualización automática en las preferencias.	La entrada del menú pasa a estar marcada como inactiva.
6. El usuario deshace el cambio anterior, y guarda los cambios.	El documento no se reabre.
7. El usuario activa de nuevo la actualización automática en las preferencias.	La entrada del menú vuelve a aparecer como activa y el documento se reabre, reflejando los cambios realizados.
8. El usuario hace cambios que violan la corrección sintáctica de la entrada.	El proceso de análisis sintáctico falla, notificándose al usuario del hecho, y por lo demás operando de forma normal. La copia antigua del documento sigue abierta y funciona sin problemas.

Cuadro 4.6.: Prueba de aceptación para “Actualizar automáticamente”

Paso seguido	Resultado esperado
1. El usuario, tras abrir un documento XML, cambia la hoja de preprocesado a un valor distinto.	Se actualiza el documento según la nueva hoja.
2. El usuario cierra XMLEye.	Se guarda las preferencias.
3. El usuario ejecuta de nuevo XMLEye, y abre un documento XML.	Se conserva la selección de hoja de preprocesado de la ejecución anterior.

Cuadro 4.7.: Prueba de aceptación para “Guardar preferencias”

4. Desarrollo del proyecto

Paso seguido	Resultado esperado
1. El usuario abre el caso de prueba <code>triple-rev-misspell.lisp</code> de <code>ACL2::Procesador</code> .	El sistema muestra el primer nodo del documento.
2. El usuario selecciona la hoja de preprocesado <code>ppACL2</code> .	Se actualiza el documento y muestra el rbol debidamente decorado con los nombres de los eventos y los iconos de xito y fracaso, seleccionando y mostrando el primer nodo de la visualizacin. En particular, debera de haber un nico elemento marcado como fracaso: <code>REV-REV</code> , hijo directo del nodo <code>raz</code> .
3. El usuario selecciona la hoja de visualizacin <code>ppACL2</code> .	Se actualiza la visualizacin del nodo actual, pasando a mostrar solamente la informacin relacionada con <code>ACL2</code> : los nombres de cada evento y el cdigo Lisp asociado.
4. El usuario navega por el rbol del documento [...]	Se visualiza la informacin correspondiente al nodo de la forma esperada. En particular, debera de informar del punto exacto del fracaso de <code>REV-REV</code> a travs de una cadena de iconos de fracaso.

Cuadro 4.8.: Prueba de aceptacin para “Integrar XMLEye y `ACL2::Procesador`”

Paso seguido	Resultado esperado
1. El usuario abre un documento Lisp.	El sistema muestra el primer nodo del documento.
2. El usuario selecciona la hoja de visualizacin <code>ppACL2</code> .	El sistema actualiza el nodo seleccionado.
3. El usuario selecciona la hoja de preprocesado <code>summaries</code> .	El sistema actualiza el documento, que ahora slo muestra las rdenes y los sumarios.
4. El usuario selecciona la hoja de preprocesado <code>reverse</code> .	El sistema actualiza el documento, en el que los eventos se hallan en orden inverso al que siguen en la fuente Lisp.

Cuadro 4.9.: Prueba de aceptacin para “Hojas `summary` y `reverse`”

Paso seguido	Resultado esperado
1. El usuario abre un documento Lisp.	El sistema muestra el primer nodo de dicho documento según las hojas actuales.
2. El usuario selecciona la hoja de preprocesado <code>ppACL2</code> y la hoja de visualización <code>ppACL2</code> .	Se actualiza el documento y la visualización, ahora estructurados ambos en ordenes y demostraciones de ACL2.
3. El usuario selecciona el primer evento usado por otros eventos.	Se muestra dicho nodo, con una lista de enlaces a los eventos que lo mencionan.

Cuadro 4.10.: Prueba de aceptación para “Ver usos de una meta”

Paso seguido	Resultado esperado
1. El usuario abre el caso de prueba <code>invoice.yaml</code> de <code>YAXML::Reverse</code> .	Se muestra su primer nodo.
2. El usuario cambia a las hojas de preprocesado y visualización <code>yaxml</code> .	Se muestra un árbol con un nuevo documento anónimo, que contiene los mismos elementos que el documento YAML original.
3. El usuario selecciona el elemento <code>bill-to</code> del documento.	Se muestran los atributos del elemento, apareciendo un enlace en el atributo <code>yaml:alias</code> .
4. El usuario activa el enlace.	Se selecciona el elemento <code>ship-to</code> del documento al que hace referencia el ancla.

Cuadro 4.11.: Prueba de aceptación para “Integrar XMLEye y `YAXML::Reverse`”

5. Resumen

Puede verse como, en este trabajo, ocupado más de la conversión de una serie de formatos que de la usual manipulación de objetos del dominio presente en las aplicaciones de gestión, y basado en otro Proyecto anterior, el análisis ha tomado una porción relativamente pequeña de la duración del proyecto.

A esto se le añade que los dos “clientes” eran expertos del dominio de ACL2, y que se tenía una comunicación constante con ellos, como sugiere la metodología XP. Para YAML no hubo clientes, pero esta carencia ha sido cubierta a través de su especificación formal y el amplio número de herramientas que implementan y utilizan este metalenguaje.

En cuanto al diseño, este se ha modelado de manera incremental, intentando evitar el diseño excesivo fuera de las necesidades estrictas del proyecto. Todos los componentes han sufrido importantes cambios en su diseño durante este proyecto: `ACL2::Procesador` y `YAXML::Reverse` han pasado a ser módulos Perl con la estructura estándar recomendada por el CPAN, y `XMLEye` se ha hecho completamente genérico y multidocumento.

5.1. Pruebas continuas

Un factor fundamental para el éxito de este proyecto fue la realización y planificación continua de pruebas durante toda su duración.

Así, se realizaban pruebas de aceptación constantemente, junto con las pruebas automáticas de regresión y de unidad. La principal ventaja de automatizar las pruebas es que aseguran una cierta funcionalidad continuamente sin afectar a la velocidad del desarrollo del proyecto.

Durante el desarrollo del proyecto, se presentaron algunos casos en los que las propias pruebas de unidad no habían capturado algunos fallos. Tras el refinamiento de las pruebas para detectar el fallo y la corrección de este, no solamente se consiguió arreglar el fallo, sino también garantizar que no volviera a aparecer en el proyecto.

5.2. Diseo iterativo y dirigido por pruebas

A menudo, intentar establecer un diseo sin comprender o conocer totalmente los requisitos resulta en software que es innecesariamente ms complejo de lo que podra ser. Mayor complejidad implica mayor propensin a fallos y mayor dificultad en su mantenimiento y posterior expansin. Los argumentos usados para defender este diseo temprano es que el coste de realizar cambios en el diseo crece rpidamente a medida que se avanza en el desarrollo.

Esta es otra de las razones por las que existe tanto inters en las pruebas continuas automatizadas en el desarrollo de software. estas pruebas ayudan a reducir una de las fuentes principales del coste de estos cambios en el diseo: la introduccin de nuevos defectos. Cada vez que se realiza un cambio, se ejecutan de nuevo todas las pruebas, y el cambio slo se enva al repositorio central si son superadas. De esta forma, podemos asegurar que toda la funcionalidad garantizada en la anterior versin sigue funcionando actualmente.

No es slo cuestin de evitar defectos y mantener al diseo flexible: se ha comprobado que los diseos resultantes de escribir primero las pruebas y luego implementar la funcionalidad son de mucha mayor calidad. En muchos casos, el cdigo es difcil de probar no tanto por la complejidad de la lgica que entraa, sino porque no se pens en las pruebas desde un comienzo, y por lo tanto no es lo bastante flexible y cohesivo.

Otra importante fuente de costes a la hora de revisar un diseo es la realizacin de los cambios necesarios sobre el propio cdigo, sobre todo cuando el elemento a cambiar es referenciado por muchos otros. Estos costes se pueden reducir en gran medida si disponemos de editores con funcionalidades de refactorizacin y deteccin de referencias entre elementos del programa: as ha sido el caso con Java, para el que se utiliza el IDE Eclipse como editor. A diferencia de otros IDE, Eclipse no nos ata a l: el fichero que dirige toda la compilacin es un fichero Ant normal y corriente, que puede ser usado fuera de Eclipse sin problemas. Se echa en falta una herramienta parecida para mdulos Perl, que sin duda se halla motivada por la mayor dificultad que su desarrollo entraa.

5.3. Transformaciones declarativas dirigidas por datos

Uno de los objetivos en el desarrollo de XMLEye para este Proyecto fue su completa separacin de ACL2. El hecho de definir las transformaciones no a travs de cdigo Java, sino como ficheros externos fcilmente instalables en XMLEye fue un punto clave: extraer la lgica de ACL2 de XMLEye y moverla a `ACL2::Procesador` fue slo cuestin de retirar el antiguo dilogo de importacin y reemplazarlo por el nuevo sistema de descriptores de formatos.

Al mismo tiempo, estos ficheros externos utilizan un lenguaje especialmente indicado para realizar transformaciones sobre los documentos XML, XSLT. Esto hace que sean mucho ms fciles de definir que si, por ejemplo, usramos el API DOM.

5.4. Mejora como producto software

Un aspecto que se identific al final del anterior Proyecto sobre el que se basa el actual fue la necesidad de facilitar otros aspectos no directamente relacionados con su funcionalidad: la instalacin era completamente manual, y la documentacin se hallaba en un nico formato.

Durante el desarrollo de este Proyecto, se han mejorado notablemente los aspectos de instalacin, con distribuciones para los mdulos Perl que slo requieren ser descomprimidas para los usuarios e instalacin y comprobacin automtica de dependencias para los desarrolladores. Adems, se han elaborado las primeras versiones de los paquetes Debian de los tres componentes, y se han recogido las experiencias obtenidas en una gua de elaboracin de paquetes Debian pblicamente disponible a travs de Internet, e incluida en la presente memoria.

En adelante, toda la documentacin se har en el wiki pblico (<http://wiki.shoyusauce.org/>), que adems dispone de la posibilidad de mantener mltiples traducciones de cada artculo. Actualmente toda la documentacin del wiki se halla disponible en ingls.

6. Conclusiones

6.1. Valoracin

El trabajo realizado durante este PFC se puede considerar un ejemplo de cmo Extreme Programming permite, incluso con limitaciones de tiempo, entregar un producto funcional y de calidad, limitando en la medida necesaria el alcance.

El subconjunto de la salida de ACL2 sigue estando limitado a casos de inters sobre todo didctico, pero se han superado las primeras barreras frente a proyectos de inters cientfico, con el soporte para proyectos con mltiples ficheros y la separacin completa del anlisis de los documentos Lisp, las salidas de ACL2, y la produccin del cdigo XML.

Al mismo tiempo, se ha podido comprobar la generalidad obtenida en el visor, ahora con identidad propia bajo el nombre de XMLEye, aadindole soporte no para un lenguaje concreto, sino para la familia completa de lenguajes del metalenguaje YAML 1.1 y su subconjunto JSON.

Todas las propiedades deseables conseguidas en el Proyecto sobre el que se basa el presente se han mantenido e incluso reforzado:

- La extensibilidad no pasa solamente ya por nuevas formas de ver la salida de ACL2, sino que se pueden aadir nuevos formatos, integrndose con los editores y conversores que necesitemos. Todo sin tener que modificar una sola lnea de XMLEye. Esto supone tambin una ampliacin de la capacidad de reutilizacin de la lgica de XMLEye.
- La transportabilidad entre mltiples sistemas operativos y microarquitecturas se mantiene, y ahora es mucho ms fcil instalar XMLEye y sus conversores, especialmente en Windows.
- Se ha traducido la interfaz completa al ingls, y el cdigo tambin est en proceso de traduccin. El nuevo wiki de XMLEye [?] permitir localizar tambin fcilmente toda la documentacin, y la forja [?] en RedIris recopilar los informes de errores, peticiones de nueva funcionalidad y los ficheros relacionados.

6.2. Mejoras y ampliaciones

6.2.1. Funcionalidad

Se extender el soporte de la salida de ACL2, empezando por tutoriales de mayor alcance como [?], cuya salida es 4 veces ms larga que la mayor tratada por este programa, y hace uso de varias caractersticas an no filtradas, como macros Lisp.

En cuanto a YAML, se aadirn hojas de usuario para formatos derivados de inters, como los marcadores de Firefox 3, por ejemplo. Habr que ver si las limitaciones impuestas por el lenguaje Perl sobre `YAXML::Reverse` realmente suponen una prdida de funcionalidad importante o no.

En futuras versiones, XMLEye incluir la capacidad de publicar las demostraciones en formato de pgina Web. Queda por determinar si se har a travs de ficheros estticos, o si se incrustar un servidor Web sencillo, como Jetty (<http://www.mortbay.org/jetty-6/>).

Habiendo traducido la interfaz y la documentacin al ingls, queda por completar la traduccin del cdigo en s. Esto ser ms complicado en `ACL2::Procesador`, ya que no hay herramientas de refactorizacin para Perl. El cdigo de `YAXML::Reverse` ya se encuentra completamente en ingls.

6.2.2. Diseo

El diseo de XMLEye ya no cambiar en gran medida, pero continuar siendo refinado y pulido, al mismo tiempo que se implementan ms pruebas de unidad sobre los propios modelos de presentacin de los documentos y los formularios principal y de formatos admitidos.

La creacin de nuevas interfaces, como la interfaz Web, ayudarn a localizar nuevas vas de mejora de las pruebas y del diseo actual. Se definirn nuevos tipos de visualizaciones, basadas en tecnologas como SVG (Structured Vector Graphics) o JavaFX, y se integrarn motores XHTML ms avanzados, como el de los proyectos Lobo (<http://www.lobobrowser.org/>) o Flying Saucer (<https://xhtmlrenderer.dev.java.net>).

Se investigarn formas de definir algntipo de pruebas de unidad sobre las hojas de usuario XSLT: una posibilidad es utilizar asertos XPath, por ejemplo, que son lo bastante flexibles como para asegurar comprobaciones potentes, y al mismo tiempo mucho ms robustos que comparar directamente el cdigo fuente con los resultados esperados.

El diseo general de los mdulos ya sigue las mejores prcticas del CPAN, pero es posible que `ACL2::Procesador` siga cambiando a una escala considerable: el manejo de

macros seguramente requerir una revisin importante del diseo actual, con la posibilidad de instrumentar el cdigo Lisp original para obtener ms informacin.

6.3. Otros aspectos de inters

El empaquetado ha sido notablemente mejorado, pero an hay cosas por hacer: mediante Launch4J se podran crear ejecutables para XMLEye en Windows que detectaran la carencia de un JRE y dirigieran al usuario a la pgina de descarga de Sun. De todas formas, existen paquetes para las versiones ms recientes de la distribucin GNU/Linux Ubuntu y archivos comprimidos listos para usar, adems de las usuales distribuciones de cdigo fuente.

Adems, ahora el Proyecto completo se puede ejecutar al 100% sobre software libre, gracias a los esfuerzos de los proyectos OpenJDK [?] e IcedTea [?] para crear una versin completamente libre de J2SE 5.0.

Se desean enviar los paquetes Debian de XMLEye y sus conversores al repositorio de paquetes Debian, y YAXML::Reverse al CPAN, para facilitar su adopcin por usuarios potenciales.

7. Manual del usuario de XMLEye y conversores asociados

7.1. Instalacin de XMLEye

En este apartado cubrir la instalacin de XMLEye en sus diferentes formas, tanto a nivel de usuario local como a nivel de sistema. Para instalar los conversores asociados y sus hojas de usuario y descriptores de formato, referirse al apartado 7.2 (pgina 171).

7.1.1. Requisitos previos

Se necesita tener instalado un entorno Java compatible con J2SE 5.0 o superior. Su instalacin se realiza automticamente si utilizamos los paquetes Debian.

Windows

Tendremos que ir a <http://java.sun.com/javase/downloads/index.jsp> y descargar una edicin reciente del JRE de acuerdo a nuestro sistema operativo. Tanto si hemos obtenido la versin que no requiere conexin como la que descarga slo lo necesario a travs de Internet, todo lo que tendremos que hacer es ejecutar el instalador y seguir las instrucciones.

GNU/Linux

Si utilizamos una distribucin basada en Debian, podemos probar a instalar los paquetes `icedtea-7-jre` o `openjdk-6-jre`. OpenJDK es la iniciativa de Sun, que a fecha de hoy (03/07/2008) es prcticamente 100 % libre salvo por algunas pequeas partes que el proyecto IcedTea ha reemplazado utilizando cdigo del proyecto GNU Classpath. Podemos instalar una versin de OpenJDK 6.0 con los reemplazos de IcedTea en Ubuntu 8.04 "Hardy Heron" usarla como entorno Java por defecto con estas rdenes:

```
sudo aptitude install openjdk-6-jre
sudo update-alternatives --config java
```

7. Manual del usuario de XMLEye y conversores asociados

Escogeremos la entrada de `openjdk-6-jre` y pulsaremos Intro, terminando con este paso. Si estamos utilizando openSUSE 10.3, podemos usar sin problemas el entorno J2SE 5.0 de Sun que incluye de fbrica. En caso de que no estuviera instalado por alguna razn, tendremos que instalar los paquetes `java-1_5_0-sun*` a travs del gestor de paquetes de YaST.

NOTA

Actualmente, OpenJDK sigue teniendo pequenos defectos en la forma en que sita los componentes Swing. Es posible que algunos dilogos tengan un aspecto distinto al normal, con botones demasiado grandes, por ejemplo. El JRE original de Sun no tiene estos problemas, pero no es 100 % libre. De todas formas, los efectos de este problema son puramente estticos.

7.1.2. Instalacin desde distribuciones precompiladas

NOTA

Si se usa una distribucin basada en Debian, se debera considerar el uso de los paquetes Debian, que son mucho ms cmodos de usar.

Un nico usuario, GNU/Linux

El proceso es muy sencillo: slo hay que descargar el `-dist-tar.gz` ms reciente de XMLEye de https://forja.rediris.es/frs/?group_id=233 y descomprimirlo bajo nuestro directorio personal. Para ejecutar XMLEye, basta con ejecutar el guin `/home/usuario/xmleye/xmleye` tras pulsar la combinacin ALT + F2. Una opcin ms cmoda para los habituales de la lnea de rdenes es aadir la siguiente lnea a `{}/~/.bashrc`:

```
export PATH=$PATH:~/xmleye
```

Haciendo esto, se puede abrir cualquier documento compatible desde la lnea de rdenes mediante:

```
xmleye (ruta absoluta o relativa)
```

Segn este procedimiento, las hojas de usuario se instalarn en `/home/usuario/~xmleye/xslt`, y los descriptores de formatos en `/home/usuario/xmleye/formats`. Las opciones se guardarn en `/home/usuario/xmleye`.

Mltiples usuarios, GNU/Linux

Un caso ms complejo es cuando queremos instalarlo para varios usuarios, pero queremos tener ajustes distintos para cada usuario (las hojas son comunes a todos). Al igual que en el caso anterior, deberemos tener instalado un JRE compatible con J2SE 5.0 o superior antes que nada, pero despus usaremos una distribucin diferente de los ficheros.

La distribucin *deal* sera la del paquete Debian, pero como es un poco ms compleja de lo que necesitamos, nos limitaremos a un trmino medio. Primero descomprimiremos la ltima distribucin de XMLEye (el enlace de descarga est en la seccin anterior) a `/opt`, que crearemos si no existe ya:

```
sudo mkdir /opt
cd /opt
sudo tar xjf (ruta a distribucin)
```

Tendremos que retocar el guin de lanzamiento un poco. XMLEye cuenta con dos variables de entorno, `XMLEYE_PREF_DIR` y `XMLEYE_FORMATS_DIR`, que indican la ruta en la que se guardarn las preferencias y los formatos personalizados por el usuario. Adems, XMLEye supone que las hojas de usuario se hallan bajo el subdirectorio `xs1t` de la ruta sobre la cual es lanzado.

Teniendo todo esto en cuenta, habr que sustituir las lneas que afectan a `PROGRAM_DIR` y `PROGRAM_JAR` por:

```
PROGRAM_DIR=/opt/xml-eye
PROGRAM_JAR=xml-eye.jar
export XMLEYE_PREF_DIR=$HOME/.xml-eye
export XMLEYE_FORMATS_DIR=$HOME/.xml-eye
mkdir -p $XMLEYE_PREF_DIR
```

De forma similar a la seccin anterior, los usuarios podrn directamente ejecutar XMLEye a travs de la ruta completa. Para que todos los usuarios puedan ejecutar XMLEye por nombre, se puede aadir esta lnea a `/etc/environment`:

```
export PATH=$PATH:/opt/xml-eye
```

Si queremos que aparezca una entrada de men y que se asocien los ficheros XML con XMLEye, entonces tendremos que, adems de seguir el paso anterior, crear el fichero `/usr/share/applications/xml-eye.desktop` con este contenido:

7. Manual del usuario de XMLEye y conversores asociados

```
[Desktop Entry]
Encoding=UTF-8
Version=1.0
Type=Application
Terminal=false
Exec=xmleye %U
Comment[es]=Visor genrico de XML dirigido por datos
Name=XMLEye
Comment=Generic data-driven XML Viewer
GenericName=Generic XML viewer
GenericName[es]=Visor genrico de XML
Icon=accessories-text-editor
Categories=Utility;TextEditor;
MimeType=application/xml;
```

Actualizaremos las bases de datos y reiniciaremos los mens de GNOME con:

```
sudo update-desktop-database
sudo update-menus
sudo killall gnome-panel nautilus
```

Ya debera de aparecer la entrada de XMLEye en el men principal de GNOME.

Las hojas de usuario estarn bajo `/opt/xmleye/xslt`, y los descriptores de formato en `/opt/xmleye/formats`.

Windows

Descargamos y descomprimos en alguna carpeta el `-dist-tar.gz` ms reciente de XMLEye de https://forja.rediris.es/frs/?group_id=233.

Para ejecutar XMLEye, haremos doble clic en el fichero `xmleye.jar` de la carpeta en que hemos descomprimido la distribucin.

7.1.3. Instalacin desde paquetes Debian

Esta es la opcin a seguir siempre que sea posible, ya que adems de ser ms sencilla, permitir recibir actualizaciones de forma automtica.

Los paquetes han sido desarrollados para Ubuntu Gutsy, pero deberan funcionar en cualquier distribucin basada en Debian reciente.

Los pasos a seguir son:

1. Descargaremos la firma digital de los paquetes disponible bajo <http://www.shoyusauce.org/packages/claveDebian.asc>.
2. Aadiremos la firma al anillo de confianza de Apt. En primer lugar lanzaremos la opcin *Sistema* → *Administracin* → *Orgenes de software* bajo el men principal de GNOME.

Hecho esto, seleccionaremos la pestaa *Autenticacin* y pulsaremos en el botn *Importar clave...*, tras lo cual seleccionaremos la firma que antes descargamos.

An no cerraremos la ventana: nos queda una cosa por hacer.

3. Ahora aadiremos los repositorios de paquetes binarios y paquetes de fuentes de *XMLEye* y sus hojas de estilos. Esta vez iremos a la pestaa *Software de terceros*.

Pulsaremos en *Aadir* e introduciremos esta lnea tal y como est:

```
deb http://www.shoyusauce.org/packages/ubuntu/ gutsy main
```

Volvemos a pulsar en *Aadir*, pero esta vez introducimos esta lnea:

```
deb-src http://www.shoyusauce.org/packages/ubuntu/ gutsy main
```

Ya podemos pulsar en *Cerrar* para cerrar este dilogo, y solicitar la actualizacin de nuestras listas de paquetes en el dilogo subsecuente pulsando en *Recargar*. Una vez haya terminado, estaremos listos para instalar *XMLEye* y otros paquetes de apoyo, como *pprocACL2*.

4. Lanzaremos *Sistema* → *Administracin* → *Gestor de paquetes Synaptic* y pulsaremos en el botn *Buscar* de la barra de herramientas.

Introduciendo "xmleye.^{en}" el campo de bsqueda obtendremos un nico resultado en el que podremos hacer doble clic para marcar para su instalacin. Tambin podremos seleccionar otros paquetes con los conversores, descriptores y hojas de estilos especficas de otros formatos, como "libacl2-procesador-perl.^o" "libxml-reverse-perl", de la misma forma.

Una vez todos los paquetes que deseamos instalar se hallen marcados, pulsaremos en *Aplicar* de la barra de herramientas para confirmar los cambios.

5. ¡Listo! Ya podemos lanzar *XMLEye* a travs de *Aplicaciones* → *Accesorios* → *XMLEye* del men principal de GNOME.

NOTA

Slo un detalle: todas las opciones que establezcamos irn a parar al subdirectorio `.xmleye` bajo nuestro directorio personal, es decir, `/home/nombredeusuario`.

La ruta bajo la cual tendremos que instalar las hojas de usuario ser `/usr/share/xml-eye/xslt`, y en `/usr/share/xml-eye/formats` se hallarn los descriptores de formato.

7.1.4. Compilacin del cdigo fuente

NOTA

Aunque hay instantneas disponibles del cdigo fuente, stas son ms para los usuarios que los desarrolladores. En caso de querer participar como desarrollador, recomiendo encarecidamente usar una copia de trabajo del repositorio Subversion. Basta con instalar el paquete `subversion` y seguir algunas instrucciones sencillas. Para ms informacin, vase el excelente libro [?].

Tras obtener un JDK compatible con J2SE 5.0 o superior, como el de OpenJDK 6.0 o 7.0, IcedTea 6.0 o 7.0, o los originales de Sun, tendremos que instalar adems la herramienta Apache Ant y el entorno de pruebas de unidad JUnit, en una de sus versiones 3.X. En Ubuntu 8.04 "Hardy Heron", esto se puede hacer mediante:

```
sudo aptitude install openjdk-6-jdk ant junit
```

Ahora tendremos que crear una copia de trabajo local de la ltima revisin de la rama principal de desarrollo del repositorio de RedIris:

```
svn checkout \  
  https://forja.rediris.es/svn/csl2-xml-eye/XMLEye/trunk \  
  xml-eye
```

Ya podemos introducirnos en `xml-eye` y aprovechar los objetivos ya definidos en el fichero `build.xml` de Ant. Si se desea instalar `xml-eye` a partir de fuentes, se recomienda usar el objetivo `dist` e instalar la distribucin segn alguno de los mtodos antes expuestos.

clean Limpia el rbol de directorios existente.

compile Compila todo el cdigo fuente.

dist Compila las fuentes ,ejecuta las pruebas de unidad y genera una distribucin autocontenida en el subdirectorio `dist`.

dist-jar Tras compilar y ejecutar las pruebas de unidad, genera un fichero `.jar` bajo `dist`, pero no llega a empaquetarlo con todo lo dems.

docs Genera la documentacin del API en formato HTML en el subdirectorio `docs` a travs de Javadoc.

run Se trata del objetivo por defecto (ejecutado a travs de **ant**). Compila el cdigo y ejecuta la versin as compilada de XMLEye.

run-about Compila el cdigo y ejecuta nicamente la ventana de "Acerca de". til a la hora de diseaar la interfaz.

run-find Como la anterior, pero para el dilogo de bsqueda.

run-types Ms de lo mismo, pero para el dilogo de edicin de tipos.

test Ejecuta las pruebas de unidad. La salida de cada conjunto de pruebas se halla bajo el fichero `TEST-*` correspondiente.

Ms cosas a tener en cuenta: esta aplicacin depende de InfoNode Tabbed Panel 1.5.0 (licenciado bajo la GPL para uso no comercial) y del look and feel JGoodies Looks 2.1.4 (licenciado bajo BSD), disponibles en <http://www.infonode.net/index.html?itp> y <https://looks.dev.java.net/> respectivamente. De todas formas, los ficheros `.jar` necesarios se hallan en el propio repositorio, por lo que no hay que hacer nada al respecto.

Adems, el desarrollo puede hacerse mucho ms cmodamente si se emplea el plugin Subclipse para Eclipse, disponible en <http://subclipse.tigris.org/>, y se importa a travs de l el proyecto Eclipse desde el repositorio. As podemos contar con sus funcionalidades de refactorizacin y notificacin de errores y avisos de compilacin en directo.

7.2. Instalacin de conversores asociados

7.2.1. ACL2::Procesador: convertidor de demostraciones de ACL2

Como muestra de las capacidades de XMLEye, desarroll en paralelo un conjunto de hojas de usuario de preprocesado y visualizacin, junto con un tipo de documento para ACL2. Esto nos permite demostraciones para dicho sistema en formato `.lisp` siguiendo una estructura arbrea donde cada nodo se muestra como un hipertexto con enlaces a otras partes de la demostracin.

La subcarpeta `t/testInputs` del fichero de fuentes con nombre de la forma `ACL2-Procesador-*.tar.gz` ms reciente contiene algunos ejemplos de inters, extrados de tutoriales reales de aprendizaje del uso de ACL2.

Primero nos ocuparemos de las dependencias, y luego veremos tres formas de instalar el convertidor, el descriptor de formato y las hojas de usuario.

Dependencias

Las dependencias a instalar variarn segn el mtodo de instalacin.

ACL2 2.9 o superior Se ha depurado ACL2::Procesador tambin con ACL2 3.1 y 3.3.

GNU/Linux En una distribucin basada en Debian, instalar ACL2 es tan sencillo como ejecutar la siguiente orden (necesitamos gcc para poder certificar libros):

```
sudo aptitude install acl2 gcc
```

Otras distribuciones requerirn del proceso manual de instalacin, documentado bajo <http://www.cs.utexas.edu/users/moore/acl2/v3-3/installation/installation.html>. Es sencillo, pero la compilacin de los libros puede llevar mucho tiempo (alrededor de 8-10 horas).

Windows Descargaremos e instalaremos la distribucin completa de ACL2 preparada por Jared Davis, disponible en <http://www.cs.utexas.edu/users/moore/acl2/v3-3/distrib/windows/>. Incluye todo lo que podremos necesitar para trabajar con ACL2:

- ACL2 3.3
- GCL 2.6.7, el entorno Lisp que necesitamos, junto con el compilador GCC necesario para certificar libros.
- El editor GNU Emacs en su versin 21.3.
- Documentacin de ACL2.
- Una coleccin de libros previamente certificados.

Perl 5.8.6 o superior

GNU/Linux La gran mayora de las distribuciones lo incluyen de fbrica, pero en el caso en que no lo tuvimos, podremos instalarlo en una distribucin basada en Debian con:

```
sudo aptitude install perl
```

Otra opcin sera descargar el cdigo fuente de <http://www.perl.com/download.csp> y compilarlo, pero no se cubrir dicha alternativa aqu.

Windows Utilizaremos la edicin 5.10 ms reciente de Strawberry Perl, disponible bajo <http://strawberryperl.com/>. Slo hemos de seguir los pasos del instalador.

Mdulos Perl: *PAR* Una vez hayamos instalado Perl (vase la seccin), slo necesitamos ejecutar una orden ms. No es necesario seguir la configuracin manual del acceso a CPAN en ninguno de los dos casos.

GNU/Linux

```
sudo cpan PAR
```

Windows

```
cpan PAR
```

Mdulos Perl: *PAR::Packer*

GNU/Linux Si estamos usando una distribucin basada en Debian reciente, podemos ejecutar:

```
sudo aptitude install libpar-packer-perl
```

De lo contrario, ejecutaremos:

```
sudo cpan PAR::Packer
```

Windows Tendremos que obtener la ltima copia del cdigo fuente de *PAR::Packer* del repositorio SVN, ya que la versin ms reciente en el CPAN a da de hoy (03/07/2008), la 0.980, no incluye el arreglo de un defecto que imposibilitaba su instalacin en Strawberry Perl. Para ello, instalaremos el cliente Subversion TortoiseSVN, disponible bajo <http://tortoisesvn.tigris.org/>, y haremos un *checkout* de la direccin <http://svn.openfoundry.org/par/PAR-Packer/trunk/>.

Una vez hayamos descargado el cdigo, abriremos una ventana del intrprete de rdenes, y dentro del directorio con el cdigo fuente, ejecutaremos:

7. Manual del usuario de XMLEye y conversores asociados

```
perl Makefile.PL
dmake
dmake test
dmake install
```

Si se nos pide instalar alguna dependencia en alguno de los pasos, aceptaremos.

Biblioteca *libxml2*

GNU/Linux Si estamos usando una distribucin basada en Debian, ejecutaremos esta orden:

```
sudo aptitude install libxml2-dev
```

Windows No hay que hacer nada: viene ya incluida con Strawberry Perl.

Instalacin mediante paquetes Debian

Si instalamos XMLEye mediante el paquete Debian, slo tendremos que instalar el paquete `libacl2-procesador-perl`. La prxima vez que iniciemos XMLEye tendremos todo lo necesario a nuestra disposicin, incluido ACL2.

Instalacin desde distribucin monoltica

Necesitaremos previamente haber instalado ACL2, y haber instalado XMLEye descomprimiendo la distribucin de la forja. Descargaremos del rea de ficheros (http://forja.rediris.es/frs/?group_id=233) de la forja de XMLEye el fichero `ACL2-Procesador*-standalone*.tar.gz` ms reciente que se corresponda con la microarquitectura de nuestra CPU y nuestro sistema operativo, y lo descomprimiremos sobre el directorio en el que instalamos XMLEye.

Instalacin desde distribucin basada en PAR

Necesitaremos ACL2, Perl y el mdulo PAR. El proceso es idntico al de 7.2.1 (pgina 174), pero en este caso el fichero a descargar y descomprimir es `ACL2-Procesador*-par-noarch.tar.gz`, que es independiente de la CPU y el sistema operativo escogido.

Instalacin manual a partir de fuentes

Los pasos, tras instalar todas las dependencias, son los siguientes:

1. Descomprimos el fichero de fuentes bajo `/tmp`, y nos introducimos en sus contenidos:

```
cd /tmp
tar xzf (ruta a las fuentes)
cd ACL2-Procesador-*
```

2. Instalamos el preprocesador tal y como instalaramos cualquier paquete Perl. Este proceso resultar muy familiar a cualquier usuario de las *autotools*:

```
perl Makefile.PL
sudo make
make test
sudo make install
```

Es muy probable que Perl nos solicite en la primera orden instalar algunas dependencias. Siempre le diremos que s. La ltima orden se podra sustituir por **sudo checkinstall** si deseamos poder desinstalarlo fcilmente, instalando `ACL2 : -Procesador` como un paquete Debian.

3. Lo siguiente es instalar las hojas de usuario, que es tan fcil (suponiendo que la ruta de hojas de usuario es `/home/tunombredeuaurio/xmleye/xslt` como:

```
cp -r xslt/* $HOME/xmleye/xslt
```

4. Terminaremos instalando el descriptor de tipo a su sitio:

```
cp types/* $HOME/xmleye/types
```

Ejecucin independiente de XMLEye

Podemos comproabr que todo va bien ejecutando el guin principal. La forma de hacerlo variar segn el mtodo de instalacin usado:

- Desde el paquete Debian, o desde fuentes: **pprocACL2** desde cualquier ruta.
- Desde la distribucin monoltica: **(ruta)/pprocACL2**, utilizando la ruta bajo la cual se halla instalado XMLEye. Podramos aadir esta ruta al PATH si quisiramos.
- Desde la distribucin basada en PAR: **perl -MPAR (ruta)/pprocACL2.par**, utilizando la ruta bajo la cual se halla instalado XMLEye.

7. Manual del usuario de XMLEye y conversores asociados

Deberamos obtener una salida como la que viene a continuacin. Por razones tcnicas, es posible que en ciertos entornos la distribucin basada en PAR no produzca ninguna salida, pero esto no es un problema: seguir funcionando de forma normal una vez le pasemos un fichero de entrada, imprimindose el XML resultante por salida estndar, y los mensajes de estado por la salida de errores.

Usage:

```
pprocACL2 [opciones] [entrada Lisp ACL2]
```

Options:

```
--help, -h, -?
```

Se limita a mostrar un mensaje corto de ayuda.

```
--man, -m
```

Muestra un mensaje detallado de ayuda en formato man.

7.2.2. YAXML::Reverse: conversor de documentos YAML a XML

Este otro conversor se desarroll para abrir con XMLEye la familia completa de documentos con lenguajes basados en YAML 1.1 o JSON. Al igual que `ACL2::Procesador`, incorpora las hojas de usuario y descriptores de formatos necesarios para su funcionamiento. Incluye soporte para anclias y alias de YAML.

Tambin en este caso se pueden ver algunos ejemplos de entradas aceptadas en `testInputs` de las fuentes, disponibles como ficheros con nombres del estilo de `YAXML-Reverse-*.tar.gz` en el rea de ficheros de la forja (http://forja.rediris.es/frs/?group_id=233).

Para evitar repeticiones innecesarias, aprovecharemos las instrucciones de instalacin de algunas de las dependencias y algunos de los pasos de los mtodos de instalacin de `ACL2::Procesador`. Comentaremos nicamente los aspectos que cambian.

Dependencias

Hay algunas dependencias de `YAXML::Reverse` cuyos mtodos de instalacin ya se han mencionado en la seccin §7.2.1 anloga de `ACL2::Procesador`:

Perl Vase la pgina 172.

PAR Vase la pgina 173.

PAR::Packer Vase la pgina 173.

Bibliotecas XML Vase la pgina 174.

Bibliotecas XSLT Tambin puede que nos hagan falta las bibliotecas para XSLT. Dependiendo del sistema operativo sobre el que trabajemos, los mtodos de instalacin cambiarn.

GNU/Linux Necesitaremos instalar los paquetes de desarrollo para las bibliotecas *exslt*, *gdbm* y *gcrypt*. En una distribucin basada en Debian, usaremos:

```
sudo aptitude install libexslt-dev \
                        libgdbm-dev \
                        libgcrypt-dev
```

Windows Como instalar manualmente las bibliotecas es demasiado complejo, usaremos una distribucin precompilada del mdulo *XML::LibXSLT*. Para ello, ejecutaremos esta orden bajo la interfaz de lnea de rdenes:

```
ppm install XML::LibXSLT
```

Instalacin mediante paquetes Debian

Una vez instalemos XMLEye mediante el paquete Debian, slo habr que instalar el paquete *libyaxml-reverse-perl*, cerrar y volver a abrir XMLEye y todo quedar listo.

Instalacin desde distribucin monoltica

En este caso utilizaremos el fichero ms reciente que concuerde con nuestro entorno que siga el patrn *YAXML-Reverse-*-standalone-*.tar.gz*.

Instalacin desde distribucin basada en PAR

Esta vez las distribuciones (que siguen el patrn *YAXML-Reverse-*-par-*.tar.gz*) sern especficas del entorno, pero por lo dems es todo igual.

Instalacin manual a partir de fuentes

El patrn del nombre de fichero cambia a *YAXML-Reverse-*.tar.gz* ms reciente, y no se necesita ACL2, sino las bibliotecas de XSLT. El resto de las dependencias siguen siendo necesarias.

Ejecucin independiente de XMLEye

El proceso es anlogo al de la seccin 7.2.1 (pgina 175), pero esta vez el guin principal es `yaml2xml` (y el PAR es `yaml2xml.par`), y la salida que deberamos obtener tendra que ser similar a sta:

```
$ yaml2xml
```

```
yaml2xml, using YAXML::Reverse 0.3.4  
Usage: yaml2xml [path to .yaml]
```

7.2.3. Instrucciones genricas

Estas instrucciones son las ms detalladas, y sirven para cualquier hoja de usuario disponible actualmente y en el futuro, siempre que no vare el diseo de XMLEye.

Instalacin de hojas de usuario

Una hoja de usuario nos permite mejorar XMLEye en una de las siguientes formas:

- Ver la estructura de un documento XML de otra forma. Un ejemplo sera ver slo un resumen del documento, cambiar el orden para su preprocesamiento, o decorar los nodos del rbol con nuevas etiquetas e iconos.
- Ver el contenido de un nodo del documento de otra forma. Podemos hacer cualquier cosa que se nos ocurra con XHTML 1.1 y CSS (no hay soporte para Javascript), y establecer vnculos entre nodos del rbol y nodos de otros documentos.

Realmente, no son ms que conjuntos de hojas XSLT que siguen una determinada estructura para permitir su localizacin y su integracin con otras hojas, ya que una hoja de usuario puede ser una especializacin de otra hoja.

La ruta donde se guardan las hojas de usuario variar dependiendo del mtodo de instalacin que hayamos seguido. Para ms detalles, referirse al apartado 7.1 (pgina 165). Nos encontraremos bajo dicha ruta una serie de ficheros (slo de inters para desarrolladores) y dos subdirectorios, `preproc` y `view`, en cuyo interior habr un subdirectorio por hoja de usuario de preprocesado o visualizacin, respectivamente.

Instalar una hoja de usuario no es ms entonces que asegurarnos que su subdirectorio se halle en el lugar correcto, y reiniciar XMLEye. Encontraremos entradas con los mismo nombres que los subdirectorios correspondientes en las entradas *Preferencias* → *Preprocesamiento* y *Preferencias* → *Visualizacin* de XMLEye. Por defecto tendremos

instalado como mnimo las hojas de usuario de preprocesado y visualizacin xml, y la hoja de visualizacin xmlSource a partir de la versin 1.22.

Instalacin de nuevos formatos de documentos

A pesar de su nombre, XMLEye puede visualizar documentos de cualquier formato, usando un conversor si no se trata de un documento XML originalmente, e integrarse con su editor (monitorizando el fichero en busca de cambios) siempre que se le d la informacin correspondiente. Esa informacin viene integrada en un *descriptor de formato de documento*.

Dichos descriptores se nombran usando la extensin `.format` y siguen un formato XML sencillo, en cuyos detalles no entraremos aqu (vase el Manual del Desarrollador). Lo nico que nos importa es su instalacin, que es tan sencilla como copiar el fichero `.format` correspondiente a la ruta de tipos que le corresponde segn el tipo de instalacin que hayamos realizado.

Una vez instalado y reiniciado XMLEye, podremos abrir cualquier documento que tenga las extensiones especificadas en el tipo de documento instalado sin ms problemas, siempre que su editor y preprocesador (si tiene alguno) se hallen debidamente instalados y tengan rdenes que puedan ejecutarse desde el directorio principal de XMLEye (posiblemente usando ejecutables bajo los directorios dentro de la variable de entorno PATH).

7.3. Uso y configuracin

Este apartado de la gua trata acerca del empleo de XMLEye y de su configuracin. Para cuestiones relacionadas con la instalacin o con detalles de implementacin, referirse a los apartados anteriores o al Manual del Desarrollador, respectivamente.

A lo largo de esta gua nos referiremos exclusivamente a las opciones de los mens, pero prcticamente toda accin tiene una combinacin equivalente en el teclado, indicada en la parte derecha del elemento del men correspondiente (y si no, es un fallo del que se agradecera una notificacin ;-): informe de l en https://forja.rediris.es/tracker/?group_id=233). Adems, las opciones ms comunes poseen un botn en la barra de herramientas, cuyo icono es una versin ampliada del icono de la entrada de men correspondiente.

Podemos cambiar entre los botones de la barra de herramientas, el rbol y el navegador mediante CTRL + TAB en de izquierda a derecha y de arriba abajo, o mediante SHIFT + CTRL + TAB en direccin inversa. TAB y SHIFT + TAB siguen funcionando tambin de la forma usual, pero se hallan redefinidos en el navegador para navegar por los hipervnculos.

La forma de lanzar el programa se detalla en la guía de instalación, y variará según el método seguido.

7.3.1. Apertura y edición de documentos

Bajo el menú *Fichero*, disponemos de *Abrir...*, en donde podremos seleccionar un fichero de cualquier tipo, o uno de los formatos que tengamos instalados.

Una vez hayamos seleccionado el fichero, XMLEye detectará a partir de la extensión de qué formato se trata y aplicará el convertidor indicado, si lo hay. Una vez el fichero se halla disponible en formato XML, pasará por la hoja de preprocesado y se mostrará su árbol en pantalla junto con la visualización de su nodo raíz.

Para editar el fichero fuente con el editor definido en el descriptor de formato, usaremos *Editar fuente*. Si *Preferencias* → *Actualización automática* se halla activo, XMLEye monitorizará el fichero en segundo plano y actualizará el árbol XML cada vez que se produzcan cambios en él, independientemente de si estuviera o no el editor abierto.

Por último, podemos acceder a los 5 últimos documentos abiertos con éxito, o *Salir* del programa.

Para cerrar un documento, actualmente no hay ninguna entrada en el menú *Fichero*, pero puede hacerse pulsando la cruz de su pestaña correspondiente. Para cerrar todos los documentos, podemos pulsar en el botón de cierre general en el extremo derecho del componente de pestañas.

7.3.2. Navegación por los documentos

El documento XML tras ser preprocesado sigue una estructura arbórea normal y corriente, por lo que podemos realizar las acciones usuales mediante las entradas del menú *Navegar*. En particular:

Buscar... Abre un diálogo de búsqueda en el que podemos buscar siguiendo diversos parámetros. Además de las opciones usuales de búsqueda por palabras completas o por concordancia de mayúsculas, se puede marcar el ámbito de la búsqueda.

Buscar siguiente Repite la última búsqueda realizada, mostrando un error en caso de que no se halla realizado una anteriormente. Si hemos llegado al último resultado de la última búsqueda, volveremos al primero después.

La repetición de una búsqueda es muy rápida, ya que XMLEye se ocupa de almacenar los resultados tras la primera petición y va dando directamente los resultados que le siguen después.

Padre Sube un nivel en la jerarquía del árbol, o no hace nada si estamos en la raíz.

Combinacin de teclas	Efecto
ARRIBA	Selecciona el nodo anterior en la vista actual del rbol.
ABAJO	Selecciona el nodo siguiente en la vista actual del rbol.
IZQUIERDA	Sube al nodo padre si el nodo est cerrado, y cierra el nodo actual de lo contrario.
DERECHA	Abre el nodo actual si est cerrado, y baja al primer hijo de lo contrario.
INICIO	Selecciona el primer nodo en la vista actual del rbol.
FIN	Selecciona el ltimo nodo en la vista actual del rbol.
ALT + ARRIBA	Selecciona al nodo padre del actual.
ALT + ABAJO	Selecciona al primer hijo del nodo actual.
ALT + IZQUIERDA	Selecciona al hermano anterior del nodo actual.
ALT + DERECHA	Selecciona al hermano siguiente del nodo actual.

Cuadro 7.1.: Accesos de teclado para navegacin por el rbol

Hijo Baja un nivel en la jerarquía del rbol, o no hace nada si estamos en un nodo hoja.

Hermano anterior Se desplaza al anterior nodo hijo del padre del nodo actual, o no hace nada si estamos en su primer hijo.

Hermano siguiente Se desplaza al siguiente nodo hijo del padre del nodo actual, o no hace nada si estamos en su ltimo hijo.

Expandir todo Expande el contenido de todos los nodos del rbol, de tal forma que todo nodo quede directamente visible, a menos que haya sido ocultado explícitamente por la hoja de visualización.

Contraer todo Contrae el contenido de todos los nodos del rbol, de tal forma que únicamente el nodo raíz quede visible.

Aunque casi toda acción es accesible mediante ratón, se recomienda aprender los accesos directos mediante teclado, ya que agilizan enormemente los desplazamientos. En la tabla 7.1 (página 181) se listan todos los accesos directos mediante teclado para navegar por el rbol del documento, tras hacer clic en l.

Podemos cambiar entre los documentos haciendo clic en sus pestañas correspondientes, o pulsando en la flecha hacia abajo justo a la izquierda de la cruz en el extremo derecho de la barra de pestañas: se desplegar una lista con los nombres de los documentos abiertos, pudiendo saltar directamente al documento que deseemos.

Combinacin de teclas	Efecto
Flechas	Desplazamiento unitario en la direccin indicada.
Inicio	Desplazamiento al inicio de la vista.
Fin	Desplazamiento al final de la vista.
AvPg	Desplazamiento de un bloque hacia delante.
RePg	Desplazamiento de un bloque hacia atrs.
Tab	Seleccin cclica del siguiente enlace.
SHIFT + TAB	Seleccin cclica del anterior enlace.
RePg	Activacin del enlace seleccionado.
CTRL + TAB	Cambio al siguiente componente.
CTRL + SHIFT + TAB	Cambio al anterior componente.

Cuadro 7.2.: Accesos de teclado para navegacin por la vista del nodo actual

7.3.3. Navegacin por la vista del nodo actual

Este rea muestra la informacin generada por la hoja de usuario de visualizacin a partir del nodo actual en formato XHTML, pudiendo activar los enlaces a travs del ratn o pulsando INTRO tras seleccionarlo a travs de un recorrido con TAB y SHIFT + TAB.

Para viajar a los componentes Swing anterior o siguiente, se usarn las combinaciones CTRL + TAB o CTRL + SHIFT + TAB respectivamente, siguiendo las recomendaciones de Sun.

Estas combinaciones y otras se recogen en la tabla 7.2 (pgina 182).

7.3.4. Personalizacin de la presentacin

Podemos cambiar en cualquier momento la hoja de usuario empleada para el pre-procesado del documento ocupada de generar la estructura arbrea que vemos en el panel izquierdo. Lo haremos seleccionando una opcin distinta del submen *Preferencias* → *Preprocesado*. Se regenerar el rbol XML del documento y se nos mostrar el nodo raz.

Podemos hacer lo mismo con la visualizacin de cada nodo bajo el submen *Preferencias* → *Preprocesado*. Se actualizar el nodo actual de forma automtica.

NOTA

Los dos tipos de hojas de usuario son independientes, pero a veces conseguir muchos mejores resultados si usa una hoja de visualizacin que est diseada de forma acorde a la hoja de preprocesado utilizada. Tal es el caso de la hoja de visualizacin ppACL2, por ejemplo.

7.3.5. Personalizacin de los formatos aceptados

Cualquier usuario puede personalizar los ajustes de los descriptores de formatos instalados, cambiando su nombre mostrado en el dilogo de seleccin, la orden de conversin, la orden de edicin y las extensiones correspondientes a su libre albedro. Esto se hace en el dilogo disponible bajo *Preferencias* → *Formatos admitidos...*

La clave %s ser sustituida por la ruta al documento en las rdenes de conversin y edicin. La orden de conversin puede dejarse vaca para formatos que ya se hallen en XML. Ninguna extensin puede contener espacios.

La copia global de todo descriptor siempre se mantiene intacta, por lo que podemos volver a ella cuando queramos pulsando el botn *Restaurar*. Podemos confirmar los cambios realizados con el botn *Aceptar*, o cancelarlos mediante *Cancelar*.

8. Manual del desarrollador

8.1. Cmo abrir nuevos formatos con XMLEye

8.1.1. Introduccin

XMLEye, como el nombre indica, se trata de un visor genrico de documentos XML. Pero ello no lo limita a nicamente ficheros XML: si se le proporciona la informacin necesaria acerca de cmo distinguir un cierto formato, y cmo convertirlo a XML, podr abrirlo de forma transparente tal y como abre un fichero XML.

La informacin referente a cmo identificar y, opcionalmente, qu hacer para convertir un formato determinado se halla en un *descriptor de formato*. Este descriptor puede incluir una referencia a cualquier ejecutable que haga las veces de convertidor a un documento XML.

En este captulo veremos qu condiciones debe cumplir un convertidor, y cmo habra que escribir posteriormente el descriptor para integrarlo con XMLEye. En cuanto a su instalacin, vase el Manual de Usuario: realmente es tan sencillo como asegurar que el convertidor est bajo la ruta correcta y copiar el descriptor de formatos a su sitio.

8.1.2. Creacin de un convertidor

Un convertidor puede ser cualquier cosa que podamos ejecutar: desde un programa en cdigo mquina hasta guiones Perl o Python. Yo en particular prefiero usar Perl, ya que se halla mejor ajustado a la tarea de procesar texto, pero cualquier cosa sirve.

Salvado el tema del lenguaje, las restricciones que ha de cumplir nuestro convertidor son:

1. Debe de poder aceptar a travs de la lnea de rdenes la ruta absoluta del fichero a convertir.
2. Debe de ofrecer el resultado a travs de la salida estndar, e imprimir errores por la salida estndar de errores.
3. Ha de comportarse como cualquier ejecutable tipo UNIX, devolviendo el cdigo de estado 0 en caso de xito y distinto de cero en caso de error.

8. Manual del desarrollador

4. Ha de poder ejecutarse desde el directorio de instalacin de XMLEye. Es decir, o bien pertenece a un directorio en nuestro PATH, o se halla instalado bajo el directorio de XMLEye, o se lanza a travs de otro programa, que s cumple la condicin anterior (por ejemplo, un intrprete de Python o Perl).

Sera recomendable que adems contara con su propia ayuda, en caso de aceptar ms opciones, y su pgina *man*, pero no es imprescindible.

Un ejemplo muy simple de qu podra hacerse puede extraerse del guin **yaml2xml** del mdulo YAXML::Reverse:

Listado 8.1: Guin principal de YAXML::Reverse

```
#!/usr/bin/perl

use strict ;
use warnings;
use YAXML::Reverse qw(ConvertFile);

# Argument processing
if (scalar @ARGV != 1) {
    print "Usage: $0 [path to .yaml]\n";
    exit 1;
}
my ($path) = @ARGV;

# Convert the YAML file
print ConvertFile($path);
exit 0;
```

Como puede verse, no es ms que un guin Perl normal y corriente: la primera lnea indica con qu intrprete debera ejecutarse al intentar ejecutarse. Es decir, si intentamos hacer esto:

```
./yaml2xml
```

, realmente haremos esto:

```
/usr/bin/perl ./yaml2xml
```

A continuacin activamos las opciones habituales de comprobacin de errores de Perl *strict* y *warnings*, e importamos la funcin `ConvertFile` del mdulo YAXML::Reverse, que implementa realmente toda la funcionalidad.

Ofreciendo esta funcionalidad separada en un mdulo aparte nos aseguramos de que pueda ser reutilizada en un futuro a travs de otros medios.

A continuacin procesamos los argumentos, recibiendo la ruta absoluta al fichero `.yaml` que antes mencionamos, y mostrando un mensaje de uso (junto con el cdigo de estado correspondiente) si no se ha proporcionado.

Por ltimo imprimimos el resultado de la conversin por la salida estndar e indicamos mediante el cdigo de estado 0 que todo ha ido bien.

8.1.3. Creacin de un descriptor de formato

Para decirle a XMLEye que acepte un nuevo formato, y que se integre con un determinado editor y convertidor, todo lo que hemos de hacer es escribir un corto fichero XML como el siguiente:

Listado 8.2: Descriptor de formato para YAXML::Reverse

```
<?xml version="1.0" encoding="UTF-8"?>
<format xmlns="http://xmleye.uca.es/xmleye/accepted-doc">
  <name>Fichero YAML/JSON</name>
  <name language="en">YAML/JSON file</name>
  <edit_cmd>emacs %s</edit_cmd>
  <import_cmd>yaml2xml %s</import_cmd>
  <extensions>
    <extension>yaml</extension>
    <extension>yml</extension>
    <extension>json</extension>
  </extensions>
</format>
```

Yendo lnea a lnea por el fichero anterior, nos vamos encontrando con lo siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Se trata de la declaracin que todo fichero XML debiera (aunque no tiene por qu) incluir. Indica que estamos empleando la versin 1.0 del estndar: aunque tambin existe la versin 1.1, las diferencias no nos importan en este caso.

```
<format xmlns="http://xmleye.uca.es/xmleye/accepted-doc">
...
</format>
```

El descriptor de documento viene representado globalmente por un elemento de nombre `format` y del espacio de nombres de descriptores de formatos de XMLEye. Es

importante usar el espacio de nombres correcto, ya que XMLEye validar cada descriptor durante su carga mediante un XML Schema, que tiene en cuenta dicho detalle.

```
<name>Fichero YAML/JSON</name>
<name language="en">YAML/JSON file</name>
```

Estas dos líneas indican el nombre del formato de documento que se le mostrará al usuario. Un detalle importante es el atributo *language*: esto nos permite localizar la misma descripción a distintos idiomas e incluso dialectos, si así lo deseamos. El algoritmo de búsqueda es bastante sencillo:

1. Sea *P* el código del país (según el estándar ISO-3166) de la localización del usuario, detectada a partir de los ajustes de su sistema operativo, y *L* el código de idioma, según el estándar ISO 639.
2. Se busca una entrada cuyo atributo *language* tenga la forma '*L_P*', coincidiendo tanto el país como el lenguaje. Esto nos permite, por ejemplo, usar distintas entradas para inglés americano e inglés británico.
3. A continuación se busca solamente por el idioma ('*L*').
4. Si aún así no hemos conseguido nada, tomaremos el nombre por defecto (aquel sin atributo *language*).

```
<edit_cmd>emacs %s</edit_cmd>
<import_cmd>yaml2xml %s</import_cmd>
```

Estas son las órdenes que XMLEye debe de usar para editar el fichero original (y no el resultado de la conversión, si la hubo), y para convertir el fichero a XML. De hecho, XMLEye invocará al convertidor cada vez que considere que ha habido cambios en el fichero original, si se ha activado la opción *Actualización automática*.

```
<extensions>
  <extension>yaml</extension>
  <extension>yml</extension>
  <extension>json</extension>
</extensions>
```

Por último, el elemento *extensions* incluye una serie de subelementos *extension* que relacionan ciertas extensiones de fichero con el formato de documento en cuestión. En otros estándares parecidos como la especificación de *shared-mime-info* de freedesktop.org ([?]) implementan también un sistema basado en el contenido del fichero, pero por simplicidad no lo he considerado.

8.2. Cmo aadir nuevas visualizaciones de documentos y elementos

8.2.1. Introduccin

A travs de los descriptores de formatos de documentos, XMLEye puede efectivamente abrir cualquier formato para el que podamos imaginarnos una conversin a XML. Con ello ya disponemos no de un visor genrico de XML, sino de un visor genrico *en* XML.

Sin embargo, no sera tan interesante si slo pudiramos ver el resultado de la conversin (o el documento XML original) tal y como queda. Utilizaremos las tecnologas XPath [?] y XSLT [?] para definir nuestras *hojas de usuario*: conjuntos cohesivos de hojas de estilos XSLT dedicadas a realizar una determinada transformacin. XMLEye implementa algunas extensiones sobre dichas tecnologas para permitir su localizacin y especializacin.

Por ejemplo, es posible que queramos poner icono o una etiqueta amigable a algunos nodos, ocultar otros, aadir nuevos nodos, o simplemente reorganizarlos. stas son las tareas de cualquier *hoja de usuario de preprocesado*.

Incluso as, seguimos teniendo un rbol XML normal y corriente. ¿Y si quisiramos ver en detalle toda la informacin concerniente a un nodo determinado, con hipervnculos a todo aquello que nos pueda ser de inters? Ah es donde entran en juego las *hojas de usuario de visualizacin*, que crean esos resmenes en formato XHTML para cada nodo del rbol seleccionado por el usuario, una vez preprocesado.

En cuanto a posibles problemas de rendimiento: las hojas son compiladas a bytecode Java en su primera ejecucin, siendo mucho ms rpidas en las posteriores transformaciones. Dicha representacin se conoce en Xalan con el nombre de *translets*. De hecho, si quisiramos, podramos generar versiones empaquetadas de esas hojas y usarlas dentro de la lnea de rdenes. Adems, si algn paso de la transformacin de visualizacin es particularmente costoso, podemos usar la hoja de preprocesado para aadir nodos ocultos y almacenar los resultados temporales que necesitemos.

En el resto de esta seccin veremos la estructura general de los repositorios que agrupan a las hojas de usuario, mencionaremos cmo localizar las hojas de usuario y heredar de una hoja de usuario base, y los aspectos particulares de las hojas de preprocesado y de visualizacin. Terminaremos ilustrando estos conceptos mediante un paseo por las partes ms importantes del cdigo de una hoja de preprocesado y de una hoja de visualizacin.

8.2.2. Estructura de los repositorios de hojas de usuario

Todas las hojas de usuario empleables por XMLEye se hallan instaladas en lo que llamaremos un repositorio de hojas de usuario. Normalmente este repositorio se halla bajo el subdirectorio `xslt` de donde est `xmleye.jar`, o en su defecto en el directorio desde el cual se lance XMLEye: en el paquete Debian se trata de `/usr/share/xmleye`, por ejemplo.

Mirando en su interior, veremos tres ficheros XSLT:

preproc.xsl ste es el punto de entrada a partir del cual siempre se comienza toda transformacin de preprocesado. Adems de importar la hoja `util.xsl` y definir variables con rutas a iconos predefinidos, se importa una hoja con la URI `current_preproc`.

Tal hoja no existe en ninguna parte: de hecho, se trata de una de las extensiones propias de XMLEye. Es una URI especial que se resuelve, a grandes rasgos, a la ruta de la hoja de preprocesado seleccionada automticamente por el usuario. Veremos los detalles despues.

view.xsl De forma anloga al caso anterior, ste es el punto de entrada para toda visualizacin en formato XHTML. Adems de importar la hoja de utilidades y la hoja de visualizacin elegida por el usuario, activa el modo de salida en XHTML y, lo que es ms importante, inicia la transformacin por el nodo que haya seleccionado el usuario (proporcionado mediante el parmetro `selectedUID` de esta hoja XSLT) importando a la URI especial `current_view`.

¿Y de dnde viene este UID? Pues del preprocesado, precisamente. Es una de las pocas cosas que absolutamente *toda* hoja de preprocesado debe hacer. De todas formas, si hacemos que nuestra hoja de preprocesado herede de la hoja base `xml`, como veremos posteriormente, no tendremos que preocuparnos por esto.

NOTA

Si lo tenemos que implementar de todas formas, recomiendo utilizar la funcin XPath **`generate-id`** definida en el estndar XSLT.

Es obvio que buscar por todo el documento el nodo que tenga un determinado identificador tiene que ser bastante costoso, y de hecho, lo es. Por ello, `view.xsl` crea un ndice a nivel del documento completo de todos los identificadores y sus nodos correspondientes, con lo que la bsqueda posterior a la primera ser mucho ms rpida.

util.xsl Esta hoja ya no cumple un papel tan importante: simplemente define algunas funciones de utilidad para la manipulacin de cadenas, como `util:to-lower` (cambio a minsculas), `util:to-upper` (cambio a maysculas) o `util:substring-after-last` (que toma la subcadena posterior a la ltima aparicin de la clave, y de lo contrario la cadena completa).

En un futuro puede que estas funciones se hagan redundantes en el cambio a XSLT 2.0, pero las dejar ah de todas formas, ya que tampoco tienen ningn impacto negativo.

Las hojas de usuario son subdirectorios de `view` (para las hojas de visualizacin) y `preproc` (para las de preprocesado), y aparecen bajo XMLEye con el mismo nombre que tenga el subdirectorio.

Toda hoja de usuario tendr como mnimo un fichero XSLT, pero puede tener varios. De hecho, en el siguiente apartado veremos por qu nos interesara hacerlo as.

8.2.3. Localizacin de una hoja de usuario

Una requisito fundamental del diseo de XMLEye ha sido siempre su internacionalizacin, es decir, la implantacin de la infraestructura necesaria para poder localizar todo el contenido de la interfaz a distintos lenguajes, atendiendo incluso a la presencia de dialectos distintos entre pases.

Las hojas de usuario generan contenidos que el usuario podr ver, as que no son ninguna excepcin. El proceso de localizacin es realmente muy simple, y se basa en una seleccin inteligente del punto de entrada a travs del cual se carga el resto de los contenidos de una hoja de usuario. Sea `L` el cdigo de idioma de la localizacin actual segn ISO 639 y `P` el cdigo de pas segn ISO 3166. Si el usuario ha seleccionado actualmente la hoja `H` de usuario de preprocesado, se intentar resolver la URI de su punto de entrada, `current_preproc`, a uno de estos ficheros XSLT en el mismo orden:

1. `xslt/preproc/H/H_L_P.xsl`
2. `xslt/preproc/H/H_L.xsl`
3. `xslt/preproc/H/H.xsl`

El proceso para las hojas de visualizacin es anlogo a ste, sustituyendo `preproc` por `view` en las rutas. Una consecuencia de este esquema es que si vamos a presentar varias traducciones del texto generado, no deberamos repetir la funcionalidad de la hoja varias veces, sino dividirla en dos partes:

1. Cada punto de entrada concretar la misma serie de variables y/o plantillas con nombre con contenidos específicos de la combinación idioma-dialecto escogida, e incluir con `xsl:include` a las hojas que proporcionen la lógica requerida. Al usar una inclusión y no una importación (con `xsl:import`, que después usaremos en otro contexto), nos aseguraremos de que se produzcan errores en caso de colisión de identificadores, y evitaremos sorpresas.
2. La hoja principal con toda la funcionalidad no contendrá cadenas de texto, sino que quedará completamente dependiente de las variables y plantillas con nombre de la hoja que actúe de punto de entrada.

Esto le resultará muy familiar a cualquiera que haya trabajado con *gettext*, por ejemplo: por un lado tenemos el diccionario de cadenas, y por otra el código propiamente dicho.

Este es el enfoque que se ha seguido en la hoja de visualización por defecto, `xml`, por ejemplo: se tiene el fichero `xml.xsl`, que actúa como punto de entrada por defecto e incluye a `principal.xsl`, que realmente implementa la funcionalidad necesaria.

NOTA

A la hora de importar ficheros XSLT concretos, hay que emplear la ruta relativa completa a partir del directorio `xslt:` así, cuando `xml.xsl` incluye a `principal.xsl`, ha de emplear la ruta relativa completa, aunque se halle en el mismo directorio: `view/xml/principal.xsl`, por ejemplo.

8.2.4. Herencia a partir de una hoja base

El propio estándar XSLT ya define mecanismos para la herencia: a través del elemento `xsl:import` podemos importar todas las reglas de otro fichero XSLT, con menos precedencia que las actuales, de tal forma que podamos efectivamente especializar dicha hoja, redefiniendo y ampliando ciertos aspectos de forma parecida a la herencia del mundo orientado a objetos.

Sin embargo, no se debe usar directamente de esa forma: el estándar sólo hace referencia a URL estáticas, con lo que perderíamos la capacidad de emplear la infraestructura de internacionalización que antes mencionamos.

Para poder seguir usando el esquema de resolución dinámica de puntos de entrada para la hoja que vayamos a especializar, tendremos que seguir usando URI especiales. En particular, URI de la forma `view_X` se resuelven al punto de entrada correcto de la hoja X de usuario de visualización. Las URI de la forma `preproc_X` son sus análogas para las hojas de preprocesado.

La hoja de preprocesado para ACL2, ppACL2, especializa la hoja de preprocesado `xml` as:

```
<xsl:import href="preproc_xml"/>
```

Como de costumbre, podemos especializar a varios niveles. As, `summaries` y `reverse` son a su vez especializaciones de ppACL2.

8.2.5. Ejemplo de hoja de preprocesado: `xml`

Para tener una mejor idea de cmo elaborar una nueva hoja de preprocesado, daremos un paseo por el cdigo de la hoja ms sencilla de preprocesado, `xml`, que normalmente especializaremos para cubrir nuestras necesidades concretas. Iremos mencionando al mismo tiempo algunos conceptos clave de XSLT, pero se recomienda de todas formas completar la informacin aqu presente mediante la especificacin oficial [?]. En la prxima seccin examinaremos una hoja de visualizacin.

Iremos alternando cdigo y explicaciones. Comencemos:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xmleye="http://www.uca.es/xmleye">

<!-- ... a continuacin ... -->

</xsl:stylesheet>
```

Aqu tenemos primero a la declaracin XML, indicando que se trata de un documento XML 1.0 codificado en UTF-8. Despus tenemos al elemento `raz` de toda hoja de estilos XSLT, `stylesheet`, bajo el espacio de nombres de XSLT. Estamos usando XSLT 1.0: existe XSLT 2.0 actualmente, pero an no est disponible en XMLEye. Vayamos avanzando por el interior del interior del elemento `stylesheet`:

```
<xsl:template match="/">
  <xsl:apply-templates select="." mode="process-node"/>
</xsl:template>
```

Este es el primer patrón de nuestra hoja de usuario. En XSLT, vamos básicamente recorriendo el árbol XML original a partir de la `raz`, `/`, un nodo que representa al documento completo, y no a un elemento en particular. A cada paso aplicamos la plantilla (`xsl:template`) ms específica cuya condición (vase el atributo `match`) se cumpla.

Utilizando el elemento `xsl:apply-templates`, pedimos que se siga recursivamente procesando el nodo actual, pero cambiando del modo por defecto por el que empezamos la transformacin a `process-node`. Cambiando entre modos podemos cambiar entre distintos conjuntos de plantillas fcilmente, y procesar varias veces el mismo nodo. Es interesante ver que si en una especializacin de esta hoja redefinimos esta regla y cambiamos la ruta del `xsl:apply-templates`, podemos podar todo excepto una determinada parte del documento. La siguiente plantilla no produce ninguna salida:

```
<xsl:template match="*" />
```

Esta plantilla vaca coincide con cualquier elemento (que no nodo: excluye a los atributos y nodos de texto, por ejemplo) en el modo por defecto. Al ser tan general, cualquier otra plantilla que definamos tendr prioridad sobre ella. La utilidad de esta regla es desactivar las reglas por defecto de XSLT, a saber:

- Los atributos no son recorridos.
- Se desciende recursivamente por los elementos.
- Se imprimen los nodos de texto como estn.

Mejor dicho: si transformamos un documento XML con una hoja XSLT vaca sin opciones ni plantillas, lo que haramos sera quitarle todo el marcado XML y dejarlo en texto puro. Pasamos a la siguiente plantilla:

```
<xsl:template match="*" mode="process-node">
  <xsl:copy>
    <xsl:copy-of select="@*" />
    <xsl:attribute name="UID">
      <xsl:value-of select="generate-id()" />
    </xsl:attribute>
    <xsl:apply-templates select="." />
    <xsl:copy-of select="text()" />
    <xsl:apply-templates select="." mode="process-children" />
  </xsl:copy>
</xsl:template>
```

Otra plantilla para cualquier elemento, pero para el modo `process-node` que vimos antes. Esta plantilla es el corazn de la hoja de preprocesado `xml`, y hace un par de cosas por nosotros:

1. Crea una copia del elemento actual.
2. Copia los atributos del elemento actual.
3. Aade el atributo `UID` con el identificador nico que necesitamos para ser capaces de seleccionar un nodo.

4. Procesa recursivamente el nodo actual en el modo por defecto. En esta hoja no hace nada de por s, pero al especializarla podremos aadir plantillas en el modo por defecto que sean aplicadas para aadir los atributos o elementos que consideremos necesario. Podramos por ejemplo usar algunos de estos atributos especiales:

hidden Si este atributo se pone a "1", el elemento no ser visible por el usuario.

leaf Si este atributo se pone a "1", el elemento ser considerado como hoja, y sus hijos no sern visibles al usuario.

He aqu un ejemplo, extrado de la hoja de visualizacin `yaxml`, que oculta todos los nodos con nombre `_key`:

```
<!-- Hide all yaml: _key elements -->
<xsl:template match="*[local-name()='_key']">
  <xsl:attribute name="hidden">1</xsl:attribute>
</xsl:template>
```

nodeicon Contiene la ruta relativa a partir del mismo directorio donde se halla el repositorio de hojas de usuario (es decir, el directorio `xslt`) al icono de 16x16 pxeles que se desee mostrar para dicho nodo.

La hoja de visualizacin de demostraciones de ACL2 usa esto para mostrar de forma sencilla qu elementos han tenido xito en su demostracin y en cules no.

nodelabel Contiene la etiqueta a usar para el nodo en cuestin. Esto permite levantar as algunas restricciones tpicas sobre los rboles XML normales, en los cuales los nombres de elementos no pueden tener espacios o ciertos caracteres.

La hoja de visualizacin de ficheros YAML/JSON usa este atributo para poner etiquetas en los pares clave/valor de los mapas: YAML es ms permisivo en las claves de los mapas que XML, permitiendo espacios, por ejemplo.

```
<!-- Label map values using their keys -->
<xsl:template match="*[local-name()='_value']">
  <xsl:attribute name="nodelabel">
    <xsl:value-of
      select="preceding-sibling::*[local-name()='_key'][1]"/>
  </xsl:attribute>
</xsl:template>
```

5. Copia los nodos de texto.

6. Procesa recursivamente el nodo actual en el modo `process-children`. As podemos determinar qu hijos del nodo actual sern procesados, y podremos retirar los que no nos interesen.

Como vemos, esta ltima plantilla sigue el patrón de diseo Mtodo Plantilla (valga la redundancia), en una versin del principio de Hollywood: "No nos llames; nosotros te llamaremos". En las especializaciones de esta hoja, aadiremos plantillas que redefiniran partes de su comportamiento y esta plantilla sera la que las aplicara. Hecha esta puntualizacin, seguimos:

```
<xsl:template match="*" mode="process-children">
  <xsl:apply-templates select="*" mode="process-node"/>
</xsl:template>
```

Esta plantilla se aplica a todos los nodos en el modo `process-children`: por defecto se procesan todos los hijos del nodo actual. Toda especializacin de `xml` puede aadir plantillas ms especficas que slo procesen algunos de los hijos de ciertos elementos, retirando los dems.

En resumen, si queremos crear nuestra hoja de preprocesado a partir de la hoja `xml`, haremos lo siguiente:

- Aadiremos este elemento como primer hijo de `xsl:stylesheet`:

```
<xsl:import href="preproc_xml"/>
```

Al importar y no incluir la hoja, nos aseguramos que todas las reglas y variables de la hoja base tengan menos prioridad, y as podremos redefinirlas como queramos sin que se produzcan colisiones.

- Para aadir nueva informacin a un elemento, definiremos una plantilla en el modo por defecto que cree los atributos y elementos deseados con `xsl:attribute` o `xsl:element`, por ejemplo.
- Para podar elementos del rbol, definiremos plantillas bajo el modo `process-children` que procesarn slo algunos de los hijos, efectivamente retirando al resto.
- Para podar todo excepto una parte del rbol, redefiniremos la plantilla del nodo raz en el modo por defecto para que la ruta del `xsl:apply-templates` seale a esa parte.
- Para cambiar la forma de procesamiento de algunos nodos por completo, podemos definir nuevas plantillas en el modo `process-node` para ellos. Tendremos que tener cuidado de no olvidar aadir el atributo `UID` con el identificador nico.

8.2.6. Ejemplo de hoja de visualizacin: xml

En este caso no llegaremos a mostrar otra vez todo el cdigo, ya que muchas partes se repiten. En su lugar, nos limitaremos a las partes ms interesantes. Primero nos centraremos en su punto de entrada por defecto, `xml.xml`, que proporciona cadenas en ingls, y luego pasaremos a describir una plantilla interesante en `main.xml`.

Punto de acceso por defecto: `ppACL2.xml`

```
<xsl:stylesheet version="1.0"
  xmlns:xm1s="http://www.uca.es/xm1eye/xml-stylesheet"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <!-- ... omitido ... -->

</xsl:stylesheet>
```

Hemos definido un nombre de espacios para esta hoja de usuario. Lo necesitaremos para nuestras variables con las cadenas de texto localizadas: as evitaremos que otras hojas que especialicen a la nuestra tengan problemas de colisiones de identificadores. Pero primero tenemos que hacer algo ms:

```
<xsl:include href="view/xml/main.xml"/>
```

Este `xsl:include` incluye a la hoja XSLT principal que implementa toda la funcionalidad, y que emplea las cadenas de texto que define este punto de acceso. Como ya se dijo antes, aqu s conviene usar inclusiones y no importaciones para que, si existen colisiones de identificadores, se produzca un mensaje de error y se eviten sorpresas. En el caso de la herencia, el razonamiento es el contrario: si estuviramos definiendo una especializacin de `xml`, utilizaramos un `xsl:import` a la URI `view_xml`, para que nuestras definiciones tomaran prioridad sobre las de la hoja base. Ahora ya podemos listar las variables e ir dndoles las cadenas localizadas de texto:

```
<xsl:variable name="xm1s:name">Name</xsl:variable>
<xsl:variable name="xm1s:value">Value</xsl:variable>
<xsl:variable name="xm1s:attributes">Attributes</xsl:variable>
```

En casos ms complejos necesitaremos plantillas con nombre en vez de variables, pero aqu no son necesarias.

Resumiendo:

- Utilizaremos variables y plantillas con nombre en los puntos de acceso para almacenar las cadenas o rutinas de generacin de salida localizadas al idioma y dialecto en cuestin. Sus identificadores se hallarn en un espacio de nombres propio de la hoja de usuario, para evitar colisiones.
- Incluiremos con `xsl:include`, en vez de importar, las hojas XSLT con la funcionalidad de la hoja de usuario desde el punto de acceso. Ello nos asegura de que si existe alguna colisin de identificadores, se produzca un error, y no nos llevemos despus sorpresas.

Hoja XSLT principal: `main.xsl`

Esta es la hoja que se ocupa de generar el cdigo XHTML a mostrar al usuario. Se compone de dos plantillas:

```
<xsl:template match="*">
  <xsl:call-template name="skeleton">
    <xsl:with-param name="rtf">
      <xsl:if test="@*">
        <h2><xsl:copy-of select="$xsl:attributes"/></h2>
        <table border="1" padding="1"
          width="60%" align="center">

          <!-- ... omitido ... -->

        </table>
      </xsl:if>
    </xsl:with-param>
  </xsl:call-template>
</xsl:template>
```

Esta es la plantilla a travs de la cual se genera la visualizacin de cualquier nodo seleccionado. Resulta interesante ver cmo llama (usando `xsl:call-template`) a una plantilla con nombre, pasndole mediante `xsl:with-param` el fragmento de XHTML generado internamente como el argumento `rtf`. Usar esta plantilla con nombre nos permite asegurar que tendremos un esqueleto bsico para todo nodo, en el que podremos rellenar los distintos espacios con el cdigo que necesitemos. Adems, las hojas derivadas podrn aprovechar esta plantilla con nombre sin problemas, o incluso la redefinir.

Esta plantilla slo se ejecuta una vez por seleccin: el punto de entrada global a toda transformacin de visualizacin, `view.xsl`, dirige directamente la transformacin al nodo seleccionado. Es lo contrario a lo que se haga en el preprocesado, en el que se iba recorriendo recursivamente el rbol. La ventaja de cambiar el nodo actual al elemento

seleccionado en vez de usar slo el subrbol con l como raz como entrada a la transformacin es que, si lo necesitamos, podemos usar toda la informacin del rbol para generar la visualizacin. Ahora veremos la plantilla `skeleton` a la que se le proporciona el fragmento de XHTML:

```
<xsl:template name="skeleton">
  <xsl:param name="rtf"/>
  <html>
    <body>

      <!-- Lista de ancestros -->
      <xsl:if test="ancestor::*">
        <h2><xsl:copy-of select="$xmls:ancestors"/></h2>
        <ol>
          <xsl:for-each select="ancestor::*">
            <li>
              <a href="#xpointer({ext:getPath(.,/)})">
                <xsl:choose>
                  <xsl:when test="@nodelabel">
                    <xsl:value-of select="@nodelabel"/>
                  </xsl:when>
                  <xsl:otherwise>
                    <xsl:value-of select="name(.)"/>
                  </xsl:otherwise>
                </xsl:choose>
              </a>
            </li>
          </xsl:for-each>
        </ol>
        <hr/>
      </xsl:if>

      <xsl:copy-of select="$rtf"/>

      <!-- ... lista de hijos ... -->
    </body>
  </html>
</xsl:template>
```

Esta plantilla nunca ser utilizada automticamente, pues carece de un atributo `match`, sino que la invocaremos por su nombre, definido en el atributo `name`. Se declara el argumento antes utilizado mediante `param`, y el resto consiste en crear el esqueleto de la visualizacin XHTML con la lista de los ancestros y los hijos, dentro del cual insertaremos el fragmento XHTML proporcionado.

Antes de insertar ese fragmento XHTML, se aade una lista de enlaces a los nodos ancestros del actual, si los hay (`xsl:if` implementa dicho comportamiento condicional). Podemos ver cmo se utiliza la variable `xm1s:ancestors` con una de las cadenas localizadas, y cmo podemos implementar estructuras iterativas y selectivas mltiples utilizando los elementos `xsl:for-each` y `xsl:choose`, respectivamente.

En este fragmento podemos ver adems cmo crear enlaces entre los nodos del documento. Empleamos un enlace normal y corriente de XHTML, pero con un ligero matiz: la sintaxis de los enlaces utiliza una variante del esquema `xpointer()` de XPointer [?] sin sus extensiones XPath. A nivel general, se permiten URL del formato `(ruta al fichero) #xpointer(patrn XPath)`, donde debe de estar al menos el `patrn`. Si no est la parte del fichero, se asume que la ruta enlaza a un nodo del mismo documento, y si est, entonces enlaza a un nodo determinado de otro documento.

Como crear rutas tiene cierta complejidad, se ha definido en el espacio de nombres `xalan://es.uca.xml1e.xpath.XPathPathManager` la funcin de extensin XPath `getPath`, que calcula la ruta entre un nodo y un ancestro suyo (el nodo actual y la raz del documento en este caso). Evidentemente, no podemos usar esta funcin cuando enlazamos a nodos de otros ficheros. En ese caso, deberemos de buscar nuestra propia forma de identificar de forma unvoca al nodo destino con una ruta XPath.

NOTA

Tambin podemos usar enlaces a direcciones Web (del estilo de `http://...`, o anclas HTML (`#ancla`)).

En resumen:

- La transformacin de visualizacin no recorre recursivamente el rbol, sino que pasa directamente al nodo actualmente seleccionado.
- Resulta til utilizar plantillas con nombre para refactorizar nuestro cdigo XSLT y evitar repetirnos. Un uso concreto en la generacin de XHTML es el de implementar un esqueleto mnimo de la visualizacin.
- Podemos crear enlaces entre nodos del mismo documento y de documentos distintos. En caso de que se traten de nodos del mismo documento, se dispone de la funcin de extensin `getPath` para ayudar a generar la ruta XPath a incluir en el enlace.

A. Gua de desarrollo de paquetes Debian

A.1. Introduccin

A.1.1. ¿Qu es un paquete Debian?

Un paquete Debian, en principio, no es ms que un fichero comprimido que contiene los ficheros de la aplicacin, cierta informacin adicional y un par de guiones especficos de apoyo. A travs de un paquete Debian, podemos instalar de forma muy sencilla cualquier software que necesitemos, sin tener que entrar en detalles de cmo est hecha o cmo se configura inicialmente la aplicacin.

Adems, la informacin adicional contenida en dicho paquete permite al sistema que se ocupa de gestionarlos (*dpkg*) mantener un control de las dependencias. As, si instalamos una determinada aplicacin, todas las bibliotecas requeridas por sta sern automticamente instaladas. El robusto control de dependencias del sistema de empaquetado de Debian es precisamente uno de sus puntos ms fuertes frente a otros sistemas de empaquetado, como *RPM* (Red Hat Package Manager).

A.1.2. ¿Por qu se desarrollan paquetes?

Uno podra preguntarse las razones detrs del desarrollo de un paquete: ¿no podramos simplemente compilar nosotros mismos el cdigo? Efectivamente, podramos, pero existen una serie de factores que hacen poco factible dicho enfoque:

- *Compilar lleva tiempo, y esfuerzo*: adems del considerable tiempo de CPU requerido para compilar cualquier aplicacin con un nivel mnimo de complejidad, hay que pensar en sus dependencias. stas son transitivas, con lo que tendremos que compilar una biblioteca que sea usada por otra biblioteca que s use directamente el programa que queremos.

Un ejemplo extremo puede ser el conocido reproductor multimedia *mplayer*, cuyas dependencias se extienden de forma recursiva a lo largo de decenas de bibliotecas, en una versin con toda la funcionalidad activada.

A. Gua de desarrollo de paquetes Debian

Tambin hay que pensar en los sistemas «exóticos» de compilacin que algunos sistemas usan: no siempre tenemos la suerte de contar con sistemas estndar como el de las autotools, donde sabemos de antemano que basta con hacer esto en el 80 % de los casos:

```
./configure
make
make install
```

- *Falta de uniformidad*: Supongamos que ya hemos compilado todo. ¿Cmo instalamos el programa? No existe un enfoque uniforme a lo largo del gran nmero y variedad de aplicaciones existentes: cada proyecto es un mundo. Tampoco sabemos en principio cmo configurarlo.

Adems, es muy probable que el programa en su estado actual no est pensado para ser instalado de dicha forma: puede que haya rutas escritas directamente en el cdigo, o que se hagan ciertas suposiciones poco estndares.

Ms an: ¿y las asociaciones de fichero? ¿Y la entrada del men? ¿Estarn bien hechas, y funcionarán bajo todos los entornos de escritorio? ¿Se podrá desinstalar el programa?

En resumen, podrá decirse que la utilidad de un paquete Debian se halla en garantizar una experiencia de instalacin cmoda y uniforme a lo largo de todo el software disponible, independientemente de cmo est hecho. Así, se pone algo de orden al problema que se presenta ante la gran configurabilidad y heterogeneidad que presentan los sistemas GNU/Linux.

De todas formas, para no perder la libertad que tenemos al descargar el cdigo fuente, podemos instalar no un paquete con los ejecutables y bibliotecas compilados, sino con su cdigo fuente adaptado para nuestra distribucin. Son los paquetes de cdigo fuente (*source packages*): normalmente se usan cuando queremos compilar partiendo de la misma base que el paquete binario de nuestra distribucin.

Por ejemplo: el paquete `kernel-source-2.4.27` contiene el cdigo fuente de la versión del kernel de Linux que emplea Debian. Este cdigo trae de por sí una serie de modificaciones que lo diferencian del kernel que podremos bajarnos de la página oficial, y se halla ya configurado con las mismas opciones que nuestro propio kernel.

A.1.3. ¿Quin desarrolla paquetes?

As, vemos que hacer un paquete no es algo trivial, pudiendo requerir todo tipo de cambios a un programa. Sin embargo, su conveniencia les hace imprescindibles: no podemos esperar obtener un nmero aceptable de usuarios a menos que nuestro software sea fácil de instalar, configurar y usar.

Por ello, tendremos que distinguir entre dos roles, que podrn ser cubiertos por las mismas o distintas personas:

- Equipo de desarrollo del programa en s (equipo original de desarrollo o *upstream*): son los que realmente se ocupan de aadir funcionalidad y corregir errores. El vocablo ingls, cuyo significado es «ro arriba», hace referencia al hecho de que las nuevas funciones «fluyen» de este equipo hacia nuevas revisiones del paquete.
- Desarrolladores del paquete: son los que deben adaptar el programa al sistema de empaquetado, controlar las dependencias, y en resumen, hacer que el programa se integre bien con el resto del sistema. Sobre todo en las primeras versiones, es posible que tenga que hacer cambios o mejoras al cdigo de la aplicacin, que tendr que discutir con el equipo original de desarrollo.

Es frecuente que un paquete vaya pasando por distintos responsables, y que haya todo un equipo detrs y no slo un responsable (como en el caso del Ubuntu MOTU Team¹, y se halla formado sobre todo por usuarios de dicha distribucin).

A.1.4. Notas acerca de esta gua

Escrib esta gua para recoger cierta informacin que anda dispersa por la Red, y parte de mi (poca) experiencia como la combinacin de un repositorio *reprepro* con sistemas de versionado y dems.

Existen otras guas, fantsticas sin duda, como [?]. No intento reemplazarlas, sino complementarlas con ms informacin de otros tipos que podra ser til, como, por ejemplo, como conseguir tener fcilmente nuestro propio repositorio.

A.2. Preparativos

Antes de empezar a crear nuestro primer paquete, vamos a preparar el entorno con todas las herramientas que necesitaremos. Por supuesto, si no deseamos distribuir nuestro paquete a travs de Internet, o gestionar varios paquetes dependientes entre s, no nos ser realmente necesario crear un repositorio local y publicarlo bajo un servidor Web.

¹Es el responsable de mantener los paquetes de los repositorios *universe* y *multiverse* de Ubuntu.

NOTA

A lo largo de esta gua, supongo que el usuario emplear Ubuntu Linux «Gutsy Gibbon» 7.10. Las rdenes exactas podran variar ligeramente, pero por lo general todo debera funcionar en cualquier sistema basado en Debian.

A.2.1. Cmo instalar las herramientas bsicas

En primer lugar, requeriremos ciertas herramientas para empezar a desarrollar paquetes Debian. Para ello, ejecutaremos (como haremos a lo largo de esta gua) bajo una terminal de texto, como la disponible desde el men principal en *Aplicaciones* → *Accesorios* → *Terminal*, la siguiente orden:

```
sudo aptitude install cdbtools debhelper autotools-dev fakeroot \
    desktop-file-utils linda lintian devscripts dh-make \
    debian-policy automake-1.9 autoconf
```

Posteriormente a lo largo de este manual iremos describiendo para qu sirven algunos de estos paquetes. Adelantaremos algunos usos tiles de *dpkg*, que despus nos sern de mucha ayuda:

dpkg -l *patr*n Permite buscar paquetes que cumplan un cierto patr. As, **dpkg -l '*sdl*'** nos buscar todos los paquetes relacionados con la biblioteca SDL. Es importante usar comillas simples, para evitar que una posible sustitucin automtica del shell interfiera con la orden.

dpkg -L *paquete* Lista todos los ficheros de un paquete determinado.

dpkg -S *fichero* Esta orden es muy til: lista el paquete al cual pertenece un determinado fichero del sistema.

dpkg-deb -c *fichero* Lista los contenidos de un paquete Debian. til para comprobar que todo est en su lugar.

dpkg-deb -x *fichero* *ruta* Extrae los contenidos del paquete Debian a la ruta que le indiquemos.

dpkg-deb -e *fichero* Extrae el directorio de control DEBIAN de un paquete. As podemos ver los guiones de postinstalacin y postdesinstalacin, por ejemplo, y otra informacin de inters.

A.2.2. Cmo conseguir paquetes limpios

Normalmente, desarrollamos el paquete Debian dentro de nuestro sistema habitual. Esto quiere decir que en nuestro sistema ya habr un gran nmero de paquetes instalados por nosotros mismos. ¿Cmo podemos asegurarnos de que no hemos olvidado alguna dependencia? Peor an: ¿se est compilando el programa con las bibliotecas que debiera, o est usando alguna versin que nosotros olvidamos haber instalado hace ya tiempo?

En el sistema de empaquetado Debian, estos problemas se resuelven creando e instalando el paquete no dentro del propio sistema, sino de una imagen de un sistema «limpio», sin otros paquetes que los esenciales. Dicha imagen se conoce como una *jaula chroot*, a partir de la aplicacin del mismo nombre, que nos «encierra» dentro de un rbol de directorios, hacindonos pensar de forma temporal que se trata del sistema de ficheros completo.

Adicionalmente, el uso de esta jaula nos permite probar el proceso de instalacin completo una y otra vez sin peligro de daar a nuestro sistema, y evitar posibles conflictos que puedan surgir.

Aunque nosotros mismos podramos (con el suficiente tiempo y esfuerzo) hacernos una jaula y ejecutar las rdenes de construccin en ella manualmente, la mejor opcin es hacer uso de alguna serie de guiones escritos especficamente para la construccin de paquetes Debian en stas. Bsicamente, lo que hacen es aadir el proceso de entrada y salida de la jaula sobre lo que ya hacen otros guiones como *dpkg-buildpackage* y *debuild*.

En principio, con *pbuilder* nos bastara. Sin embargo, tiene un problema: la imagen que crea es simplemente un gran fichero *tar.gz*, que descomprime cada vez que construimos el paquete. Esto hace que sea mucho ms lento que una construccin habitual. Lo combinaremos con *cowdancer*, que usa un mecanismo de *copy-on-write*² para evitarlo, reduciendo en gran medida dicha sobrecarga.

As, los pasos a seguir son:

1. Instalamos los dos paquetes que necesitamos, junto con sus dependencias:

```
sudo aptitude install pbuilder cowdancer
```

2. Generamos de forma automatizada la imagen del sistema Ubuntu con la siguiente orden. Este proceso puede llevar bastante tiempo: tiene que descargar un Ubuntu mnimo a travs de la red.

```
sudo cowbuilder --create \
  --othermirror \
  "deb http://archive.ubuntu.com/ubuntu gutsy universe multiverse"
```

²Este mecanismo usa el rbol de directorios original directamente en un principio, y realiza todas las posteriores escrituras sobre copias de los originales, para mantener el rbol de directorios original limpio, y no tener que realizar una lenta copia completa por otro lado.

A. Gua de desarrollo de paquetes Debian

3. Tras un buen rato, nuestra imagen estar lista. Aadimos las siguientes lneas al fichero `~{}/.pbuilderrc`:

```
# Seleccionamos los componentes de Ubuntu
# (son distintos a los de Debian, que son
#  main, contrib y non-free)
COMPONENTS="main universe multiverse"

# Uso de cowbuilder en vez de pbuilder normal (ms rpido)
export PDEBUILD_PBUILDER=cowbuilder

# Aqui pondremos nuestro correo como desarrollador de
# paquetes
export DEBEMAIL="tudireccion@decorreo"
```

A.2.3. Cmo simplificar la distribucin

Supongamos que alguien quiere instalar nuestro paquete. ¿Cmo lo consigue? ¿Se lo descarga de alguna web, teniendo que buscar una y otra vez cada vez que necesite algo? ¿Tendr el cliente que comprobar una y otra vez dicha pgina web, o tendr el desarrollador de la aplicacin que implementar algn mecanismo de actualizacin automtica, aunque slo se trate de un guin Perl? No olvidemos que, adems, en el caso de una distribucin GNU/Linux, podemos tener miles de paquetes instalados (1646 en mi caso, retirando los paquetes esenciales). No parece factible simplemente descargarlo de alguna pgina web.

En realidad lo que se hace es agrupar los paquetes en repositorios. Se conoce como *repositorio Debian* a un rbol de directorios con una cierta estructura, desde el que el sistema de empaquetado de software de Debian, *dpkg*, pueda instalar software. Dicho repositorio puede estar disponible de forma local en nuestro disco duro, como el que haremos aqu, o en otra mquina que aloje un servidor HTTP (web) o FTP.

Este paso slo es estrictamente necesario cuando tengamos que construir nosotros mismos varios paquetes que dependan entre s. De todas formas, recomiendo seguirlo, ya que nos permite depurar ciertas cosas como que el paquete se halle correctamente firmado y nuestra clave pblica bien distribuida. Adems, podemos usarlo para actualizar a partir de l nuestra versin instalada del mismo paquete, y as reproducir exactamente lo que un usuario normal vera.

Podramos gestionar manualmente el contenido del repositorio a travs de las herramientas del paquete *apt-utils*, pero por simplicidad, usaremos el sistema *reprepro* (anteriormente *mirrorter*) para gestionar todo el proceso.

Seguiremos estos pasos:

1. Instalamos el paquete correspondiente con:

```
sudo aptitude install reprepro
```

2. Creamos el repositorio dentro de `/var/packages/ubuntu/`:

```
sudo mkdir -p /var/packages/ubuntu/conf
```

3. Situamos en el fichero `/var/packages/ubuntu/conf/distributions` las siguientes líneas:

```
Origin: Mi Nombre  
Label: Mi Nombre  
Codename: gutsy  
Architectures: i386 source  
Components: main  
Description: Mi propio repositorio
```

Como vemos, tenemos una distribución, *gutsy*, que contiene paquetes binarios para la arquitectura IA-32 (Intel 80386 o superior) y paquetes de código fuente repartidos en un único componente *main*.

4. Ahora ya podemos manipular el repositorio de distintas formas. Posteriormente veremos cómo obtener los ficheros `.deb` y `.dsc`. Por ejemplo:

- Añadimos paquetes binarios con:

```
sudo reprepro -b /var/packages/ubuntu \  
includedeb gutsy (fichero .deb)
```

- Añadimos paquetes de fuentes con:

```
sudo reprepro -b /var/packages/ubuntu \  
includedsc gutsy (fichero .dsc)
```

- Retiramos un paquete mediante:

```
sudo reprepro -b /var/packages/ubuntu \  
remove gutsy (nombre)
```

A.2.4. Control de versiones

Todo desarrollo de software se puede beneficiar del uso de un sistema de control de versiones. A travs de ste, podremos siempre acceder a todas las versiones realizadas en el pasado de cualquier fichero, y mantener mltiples copias de trabajo siempre sincronizadas. Son especialmente importantes en proyectos de software libre, donde puede haber potencialmente muchos participantes.

Adems, el uso explcito de un sistema de control de versiones elimina una molestia a la hora de desarrollar paquetes Debian: la avalancha de ficheros que tendramos que gestionar de otra forma, con ficheros `tar.gz` (ms conocidos como *tarballs*), parches y dems que habr que mantener versin a versin.

Como siempre, existe una gran variedad, pero en este caso eleg Subversion, descendiente a nivel conceptual del conocido CVS, por su excelente documentacin, estabilidad y simplicidad. Adicionalmente, podremos hacer uso de un conjunto de guiones de ayuda desarrollados por la comunidad Debian, que nos ayudan a estandarizar un poco la estructura del repositorio.

NOTA

Para los preocupados por su espacio en disco: el enfoque seguido por Subversion, al igual que en la mayora de los sistemas de control de versiones, es mucho ms eficiente que lo que nosotros podramos hacer simplemente guardando todas las versiones. Slo guarda, comprimidos, los cambios de una versin a otra.

El proceso de preparacin de nuestro propio repositorio de Subversion y una copia de trabajo local es sencillo:

1. Instalamos Subversion y el paquete de desarrollo de paquetes (valga la redundancia) bajo repositorios Subversion:

```
sudo aptitude install subversion svn-buildpackage
```

2. Creamos el repositorio central en `~/svnDebian`:

```
svnadmin create .svnDebian
```

3. Ahora creamos una copia de trabajo, sobre la que crearemos y modificaremos ficheros, que luego enviaremos al repositorio central para que otras personas puedan propagar los cambios a sus propias copias de trabajo:

```
svn checkout \  
file:///home/minombredeusuario/.svnDebian packages
```


Por lo pronto, no haremos nada ms: la estructura de nuestro repositorio ser generada de forma automtica, y diferir ligeramente de la estndar que el libro de Subversion describe.

A ttulo de referencia para la fase de creacin, he aqu algunas de las rdenes que podemos usar dentro de una copia de trabajo. Para ms informacin, referirse al libro electrnico [?].

svn add *fichero* Marca un fichero o directorio para aadir al repositorio.

svn rm *fichero* Marca un fichero o directorio para eliminar del repositorio.

svn mv *fichero ruta* Mueve un fichero o directorio a otra ruta. El fichero o directorio no debe haber sufrido modificaciones desde su ltimo envo.

svn cp *fichero ruta* Copia un fichero o directorio a otra ruta. El fichero o directorio no debe haber sufrido modificaciones desde su ltimo envo.

svn mkdir *directorio* Crea un nuevo directorio y lo aade al repositorio.

svn commit -m *mensaje* Confirma todos los cambios hechos hasta el momento, y los enva al repositorio, registrando el envo con el mensaje que hayamos proporcionado. Tiene semntica transaccional: el envo se produce por completo, o no se produce en absoluto.

svn status Muestra el estado de todos los ficheros bajo el rbol actual, con una letra antes de su ruta. Algunas de las que pueden aparecer son:

? No pertenece al repositorio (quizs habra que aadirlo con **svn add**).

A El fichero va a ser aadido en el prximo envo.

D El fichero va a ser borrado en el prximo envo.

M El fichero ha sido modificado desde el ltimo envo.

svn export *rutaDirOrigen rutaDestino* Exporta el directorio origen, que forma parte de una copia de trabajo, a la ruta destino, retirando todos los ficheros usados por Subversion. Muy til cuando queremos redistribuir cdigo, por ejemplo, y no la copia de trabajo entera.

A.2.5. Autenticacin

Slo nos queda una cosa ms para tener el entorno completamente listo antes de desarrollar nuestro paquete Debian: preparar nuestra firma digital personal. Adicionalmente, la integraremos con nuestro repositorio local, y usaremos un agente de claves para evitar escribir una y otra vez su contrasea a la hora de construir el paquete.

¿Por qu querramos usar una? No olvidemos que estamos haciendo un paquete que efectivamente va a ser instalado en el sistema final bajo la propiedad del superusuario. Una persona malintencionada podra manipular los contenidos del paquete y comprometer a los sistemas que lo instalen, aadiendo *rootkits* o puertas traseras entre nuestros ficheros.

Para un usuario muy centrado en seguridad, el mayor nivel de seguridad obtenible con un nivel razonable de esfuerzo, sin llegar a tener que analizar en profundidad el cdigo, sera directamente descargar el cdigo de alguna fuente fiable, y comprobar su integridad mediante algn algoritmo de *hashing*. Lo ms comn es encontrar *checksums* MD5, que aunque no criptogrficamente seguros, son rpidos de calcular, y cumplen el propsito original de evitar descargas corruptas.

No llegaremos a esos extremos. En lugar de eso, aprovecharemos una de las aplicaciones de los algoritmos de cifrado asimtrico. En ellos, todo usuario tiene dos claves: la pblica, que damos a todo el mundo, y la privada. Si ciframos algo con una clave, se puede descifrar con la otra. As no nos hace falta transmitir la clave a la otra parte, como en los algoritmos tradicionales simtricos. En particular, si ciframos algo con nuestra clave privada, y el usuario lo descifra con nuestra clave pblica, ste puede estar seguro que el paquete es autntico.

Dado que cifrar todo un paquete puede ser potencialmente muy costoso, lo que se suele hacer es cifrar el resultado de alguna funcin de hash aplicada sobre el fichero, como SHA-1 o alguna variante criptogrficamente ms fuerte. De hecho, en los repositorios Debian, se va un paso ms all: slo se firma el fichero central con el listado de los paquetes disponibles, sus tamaos y sus *checksums* MD5, para tener el menor impacto posible en rendimiento. En nuestra configuracin, se trata de `/var/packages/ubuntu/dists/gutsy/Release`, y su firma se halla en `/var/packages/ubuntu/dists/gutsy/Release.gpg`.

El proceso de instalacin es algo ms elaborado que en los anteriores casos, dado que se extiende a lo largo de unos cuantos ficheros de configuracin:

1. Primero instalamos los paquetes que necesitamos: `gnupg` implementa la funcionalidad de cifrado en s, `gnupg-agent` es el agente ocupado de almacenar temporalmente nuestra contrasea y `pinentry-gtk2` nos permite usar un dialogo grfico para introducir la contrasea, en vez de usar la terminal:

```
sudo aptitude install gnupg gnupg-agent pinentry-gtk2
```

2. Ahora generaremos nuestro par de claves, siguiendo las instrucciones que se nos irn indicando:

```
gpg --gen-key
```

El campo de correo electrnico es muy importante para el resto de pasos, y debe coincidir con el que usamos anteriormente en DEBEMAIL (ver seccin A.2.2 (pgina 205)).

3. Aadiremos los ajustes necesarios al agente de GnuPG, en el fichero `~/.gnupg/gpg-agent.conf`:

```
pinentry-program /usr/bin/pinentry-gtk-2
no-grab
default-cache-ttl 1800
```

As, slo tendremos que introducir nuestra contrasea cada 30 minutos como mucho, y emplearemos una interfaz grfica basada en GTK 2.0 (estilo GNOME).

4. Hemos de indicar a GnuPG que haga uso del agente para pedir la contrasea de la clave privada. Hay que descomentar la siguiente lnea de `~/.gnupg/gpg.conf`:

```
use-agent
```

5. Al instalar el paquete `gnupg-agent` se nos aadi el script necesario para su arranque en `/etc/X11/Xsession.d/90gpg-agent`. Hemos de reiniciar el servidor X, saliendo de nuestra sesin y pulsando CTRL + ALT + Retroceso.
6. Debemos decirle a *reprepro* que use dicha clave para firmar los paquetes. Aadiremos la siguiente lnea con el email que usamos en la clave GPG a `/var/packages/ubuntu/conf/distributions`:

```
SignWith: <tudireccion@decorreo>
```

7. Por ltimo, hemos de exportar nuestra clave pblica a un fichero, para poder pasrsela a los dems, y aadirla a la lista de claves confiables de *apt*:

```
gpg -a --export tudireccion@decorreo > claveDebian.asc
sudo apt-key add claveDebian.asc
sudo cp claveDebian.asc /var/packages
```

A.2.6. Distribucin a travs de Internet

Instalacin del servidor

Podemos aprovechar nuestro repositorio local para distribuir nuestros paquetes a travs de la red, simplemente publicando el directorio `/var/packages` bajo un servidor HTTP o FTP. En particular, aqu veremos cmo hacerlo con el servidor Apache 2.

Evidentemente, hacerlo o no es completamente opcional. Los pasos a seguir son:

1. Si no tenemos una direccin IP esttica, podemos en su lugar registrarnos en sitios como no-ip (<http://www.no-ip.org>) o DynDNS (<http://dyndns.com>), donde conseguiremos un nombre de dominio que actualizaremos peridicamente con nuestra IP.
2. Ahora hemos de asegurarnos de que el puerto 80 por TCP se halle accesible, cambiando los ajustes del cortafuegos y del encaminador en caso de que usemos NAT (Network Address Translation). Con Ubuntu recin instalado, el cortafuegos no tendr ninguna regla definida, por lo que en principio no habra que hacer nada en cuanto a la configuracin del servidor.
3. Instalamos el servidor Apache mediante:

```
sudo aptitude install apache2
```

4. Crearemos el fichero `/etc/apache2/sites-available/repoDebian`, con el siguiente contenido, donde sustituiremos la direccin de nuestro servidor y el correo que empleamos en la firma GPG:

```
<VirtualHost *:80>
    ServerAdmin tudireccion@decorreo

    DocumentRoot /var/packages
    ServerName direccion.delservidor.org:80
    ErrorLog /var/log/apache2/error.log

    LogLevel warn

    CustomLog /var/log/apache2/access.log combined
    ServerSignature On

    # Permite que la gente navegue por el directorio
    <Directory "/var/packages">
        Options Indexes FollowSymLinks MultiViews
        DirectoryIndex index.html
        AllowOverride Options
```

```

        Order allow,deny
        allow from all
    </Directory>

    # Oculta el directorio conf
    <Directory "/var/packages/*/conf">
        Order allow,deny
        Deny from all
        Satisfy all
    </Directory>

    # Oculta el directorio db
    <Directory "/var/packages/*/db">
        Order allow,deny
        Deny from all
        Satisfy all
    </Directory>
</VirtualHost>

```

5. Activaremos dicha pgina web y desactivaremos la pgina por defecto:

```

cd /etc/apache2/sites-enabled
sudo a2dissite default
sudo a2ensite repoDebian

```

6. Reiniciamos el servidor mediante:

```

sudo /etc/init.d/apache2 restart

```

7. ¡Listo! Habra que probar a cargar en un navegador la direccin <http://127.0.0.1/> y ver que todo coincide.

Instrucciones para un usuario de nuestro paquete

Ahora nos pondremos por un momento en la piel de un usuario de nuestro paquete. ¿Cmo podra tener nuestro paquete siempre a la ltima, con la seguridad de que es la versin genuina?

En este caso, vamos a intentar hacerlo todo a travs de interfaces grficas, ms amigables para un usuario medio. Recomendando aadirnos a nosotros mismos como un cliente ms, para facilitar la depuracin. Slo tendremos que sustituir nuestro nombre de host real con 127.0.0.1 en el resto de instrucciones, o mejor an, aadir la siguiente lnea a `/etc/hosts`, reemplazando la que tuviera la misma direccin si hubiera alguna:

```

127.0.0.1 localhost direccion.delservidor.org

```

A. Gua de desarrollo de paquetes Debian

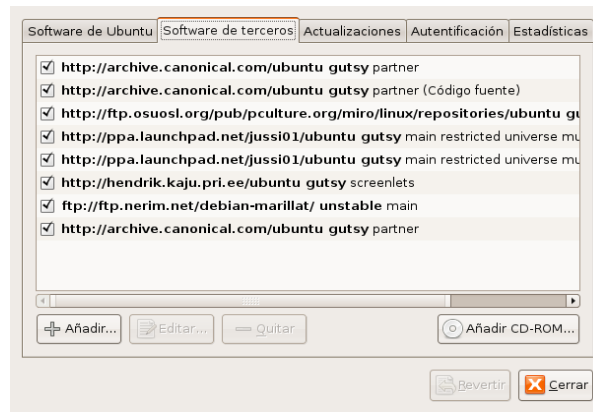


Figura A.1.: Pestaa de Software de Terceros

En primer lugar, el usuario debe de haber descargado el fichero <http://direccion.delservidor.org/claveDebian.asc> que exportamos antes, con nuestra clave pblica.

Accederemos, a partir de la barra de programas en la parte superior de la pantalla, al elemento *Sistema* → *Administracin* → *Orgenes de software*.

En el dilogo que aparece, pasamos a la pestaa *Software de terceros*, cuyo aspecto ser similar al de la figura .

Pulsaremos en el botn **Aadir** una primera vez, y pegaremos la siguiente lnea:

```
deb http://tunombre.dservidor.org/ubuntu gutsy main
```

Volvemos a pulsarlo, y ahora introducimos *sta*, correspondiente a un repositorio de paquetes de cdigo fuente (de ah el *deb-src*):

```
deb-src http://tunombre.dservidor.org/ubuntu gutsy main
```

Cambiamos a la pestaa *Autenticacin*, que ser parecida a la de la figura , y pulsamos **Importar clave**. Seleccionaremos el fichero *claveDebian.asc* antes descarga-do.

Guardamos los cambios realizados pulsando **Cerrar**, y finalmente pulsamos el botn **Recargar** de la barra de herramientas de la ventana principal. Cuando se cierre au-tomticamente el dilogo que aparece, ya podremos usar como siempre *Synaptic*, empleando el botn **Buscar** para hallar el paquete que deseamos, marcarlo con un doble clic e instalarlo mediante **Aplicar**. Adems, a partir de ahora, el sistema *apt* se ocupar de monitorizar dicho repositorio y mantener actualizado el sistema.

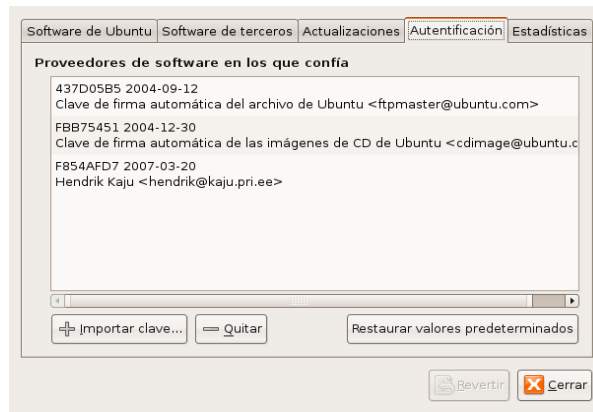


Figura A.2.: Pestaña de Autenticación

A.3. Creación y mantenimiento del paquete

En este capítulo, daré un ejemplo de un paquete razonablemente sencillo, pero completo: un emulador de SNES, conocido como ZSNES. Veremos todas las fases, desde que nos descargamos el código fuente hasta que tenemos el paquete instalado en nuestro sistema y funcionando. Existen muchas guías de creación de paquetes, pero en mi opinión la información se halla bastante fragmentada. De todas formas, en el wiki de Ubuntu hay una excelente introducción acerca del tema [?], y también hay una guía por parte de la comunidad Debian [?], aunque en mi opinión la de Ubuntu está más actualizada.

Para simplificar, describiré el proceso de forma secuencial. Sin embargo, lo normal es que sea iterativo, teniendo muchas revisiones intermedias del paquete hasta dejarlo listo para su distribución. Se ven aspectos más avanzados en el siguiente capítulo.

A.3.1. Adaptaciones previas al uso de Subversion

Creación de un esqueleto

Antes de poder introducir los archivos fuente de nuestro paquete en el repositorio Subversion que previamente preparamos, hemos de crear una primera versión de nuestro paquete.

Tras descargar el código fuente de la página oficial (<http://www.zsnes.com/index.php?page=files>) a `/tmp/packages/zsnes151src.tar.bz2`, crearemos el esqueleto básico del paquete mediante **dh_make**, una de las muchas herramientas del paquete *debhelper* de ayuda. Ejecutaremos las siguientes órdenes en una terminal dentro del directorio `/tmp/packages`:

A. Gua de desarrollo de paquetes Debian

```
tar -xjf zsnes151src.tar.bz2
mv zsnes_1_51 zsnes-1.510
cd zsnes-1.510
dh_make -e tudireccion@decorreo -c GPL -s --createorig
```

La carpeta que hemos creado (`zsnes-1.510`) obedece al convenio seguido por Debian `nombrepaquete-versionUpstream`, donde la versin del programa original se entiende como una serie de nmeros separados por puntos: as, `zsnes-1.510` es ms reciente que `zsnes-1.6`, y menos que `zsnes-1.600`.

Por otro lado, las opciones pasadas a **dh_make** son:

- e tudireccion@decorreo** Especifica nuestra direccin email como desarrollador del paquete. Se usa en el registro de cambios (de ahora en adelante el *Changelog*), y para realizar firmas digitales.
- c GPL** Indica que el cdigo original sigue la General Public License. Otras opciones incluyen la LGPL, BSD o la licencia artstica. En otro caso, se nos dejar un hueco (posteriormente veremos dnde) para que lo rellenemos con el texto de la licencia en cuestin.
- s** Existen varios tipos de paquete Debian: de un solo binario, de varios, bibliotecas, o paquetes que emplean CDBS (el resto usan nicamente *debhelper*). Aqu hemos decidido hacer un paquete de un solo binario, mediante *debhelper*. Despues veremos tambin cmo hacer un paquete con CDBS.
- createorig** Creamos en el directorio padre un fichero `zsnes_1.510.orig.tar.gz` con el cdigo fuente original, para poder comparar con la versin que usemos al construir el paquete y volcar las diferencias a un fichero `diff`.

Edicin

Ya tenemos el esqueleto del paquete. Todos los ficheros especficos de l se hallan bajo el directorio `/tmp/packages/zsnes-1.510/debian`. Si examinamos dicho directorio, veremos que hay un gran nmero de ficheros. No utilizaremos los ejemplos incluidos, indicados por la extensin `.ex`, as que los retiraremos, junto con el fichero `README.Debian`, dado que no hay nada especial acerca de nuestro paquete:

```
rm debian/*.ex,EX debian/README.Debian
```

Iremos rellinando cada fichero de control en `debian` con los datos necesarios. Iremos detallando su sintaxis y semntica a lo largo de esta seccin.

changelog este es el registro de cambios de nuestro paquete. Aqu iremos indicando los cambios realizados a lo largo de cada versin del paquete, no del software original. Este fichero es el que nuestros usuarios leern para ver qu hay de nuevo en cada versin del paquete.

Escribiremos nuestra primera entrada:

```
zsnes (1.510-0ubuntu1) gutsy; urgency=low

* Versin inicial del paquete

-- Antonio Garcia <nyoescape@gmail.com>  Fri, 19 Feb 2008 13:49:12 +0100
```

Vemos cmo la versin actual del paquete junto con su ltima fecha y autor del cambio se hallan codificados en el registro. Tambin se tiene en cuenta la distribucin (en nuestro caso *gutsy*, de Ubuntu), y la urgencia del cambio (por lo general baja, a menos que se trata de una vulnerabilidad de seguridad o algo del estilo).

Para una misma versin del software original, tendremos distintas versiones del paquete, separadas del nmero de versin original por un guin, como vemos aqu. En particular, los paquetes de Ubuntu [?] usan el esquema *-XubuntuY*, indicando que se trata de la Y-sima versin del paquete de Ubuntu originado de la X-sima versin del paquete Debian (0 si no proviene de un paquete Debian). Los nmeros de versin de paquete comienzan por 1.

NOTA

La razn de este esquema de versionado es para permitir una fcil integracin con los paquetes Debian. Normalmente, la poltica de Ubuntu es slo crear nuevos paquetes o versiones de stos si el paquete Debian est anticuado o tiene algn problema. As, si los de Debian sacan una nueva versin, como la *-3*, a partir de *-2ubuntu3*, se reflejar dicha informacin de forma correcta.

As, para la prxima versin del paquete, slo tendremos que aadir la entrada en cuestin al registro, y nuestros guiones de ayuda harn el resto del trabajo.

NOTA

Mucho cuidado con el formato del registro, es muy rgido. El espaciado debe ser exactamente el mismo que en el ejemplo, como los dos espacios entre la direccin de correo y la fecha, o el espacio inicial al inicio de la misma lnea.

A. Gua de desarrollo de paquetes Debian

compat Para este fichero no hay que hacer nada: slo contiene un nmero entero, indicando qu versin del paquete *debhelper* estamos usando.

control Este fichero es muy importante: describe todos los paquetes que estamos definiendo y enuncia sus dependencias. El formato es tambin bastante rgido, pero muy simple. Utiliza una serie de campos delimitados por ':' y saltos de lnea.

Por supuesto, no existe ninguna receta mgica que nos diga las dependencias de un programa cualquiera. Para ello, normalmente tendremos que examinar la documentacin del desarrollador original, y/o el guin de compilacin que utilice: como aqu usan las *autotools*, podramos consultar `src/configure.in`. Una buena referencia respecto a las *autotools* es el Autobook [?].

Por suerte, los desarrolladores de ZSNES han incluido dichas dependencias en `docs/-install.txt`, con lo que no tendremos que ir buscando en los guiones de compilacin.

El fichero que usaremos ser ste:

```
Source: zsnes
Section: games
Priority: optional
Maintainer: Antonio Garcia <nyoescape@gmail.com>
Build-Depends: cdb, debhelper (>= 4.1.0), autotools-dev, fakeroot,
    desktop-file-utils, g++ (>= 4), libsdl1.2-dev, nasm (>= 0.98),
    zlib1g-dev (>= 1.2.3), libpng12-dev (>= 1.2), libncurses5-dev,
    libgl1-mesa-dev
Standards-Version: 3.7.2

Package: zsnes
Architecture: i386
Depends: $shlibs:Depends
Description: Emulador de Super Nintendo
    Emulador de la consola Super Nintendo con ms funciones
    disponibles. Permite guardar y cargar estados, grabar
    demostraciones, y aplicar diversos filtros. Tiene una
    compatibilidad inmejorable.
```

NOTA

En ste y en cualquier otro fichero de control de Debian, no debemos olvidar poner un salto de lnea justo al final del fichero.

Examinando el fichero de campo a campo, tenemos:

Source: zsnes Indica que el paquete fuente del que derivan todos se llama "zsnes".

Section: games Por la poltica de Debian [?], todo paquete se halla en alguna seccin de las disponibles. As indicamos qu tipo de aplicacin es: un juego, un editor, etc.

Priority: optional Indica la importancia del paquete: desde imprescindibles (*required*), pasando por importantes (*important*), estndar (*standard*), opcionales (*optional*), y extra (tienen conflicto con alguno de ms prioridad).

Maintainer: Antonio Garcia <nyoescape@gmail.com> Nombre y direccin de contacto del desarrollador del paquete.

Build-Depends: ... Paquetes requeridos para poder compilar este paquete. Incluye las herramientas para paquetes Debian y las dependencias del propio programa.

Standards-Version: 3.7.2 Versin de la poltica de Debian que este documento sigue. Realmente se halla compuesta por varios documentos, todos situados bajo el directorio `/usr/share/doc/debian-policy`.

Package: zsnes Nombre de uno de los paquetes binarios generados a partir del fuente. Aqu slo hay uno y tiene el mismo nombre.

Architecture: i386 Arquitectura a la que va dirigida el paquete. Existe una gran variedad de valores, pero nos interesan sobre todo *i386* (la IA-32 habitual), *source* (cdigo fuente) y *all* (cdigo sin una arquitectura definida, como programas Java, o guiones de algn lenguaje interpretado como Perl o Python).

Depends Paquetes requeridos para que ste se instale y funcione correctamente. La variable `${shlib:Depends}` incluye las dependencias deducidas de forma automtica en cuanto a bibliotecas dinmicas se refiere.

Description Incluye una descripcin corta de una sola lnea y otra ms larga de varias lneas del contenido del paquete. Al igual que siempre, su formato es muy rgido: toda lnea de la descripcin larga comienza por un espacio, y lneas vacas nicamente aaden un punto ('.'). El campo termina tras la primera lnea sin dicho espacio inicial.

NOTA

Mucho cuidado con los acentos y dems en el nombre del desarrollador del paquete y otros campos: podran causar problemas en el interior de la jaula *chroot*, que slo tiene soporte para los caracteres ASCII de 7 bits.

A. Gua de desarrollo de paquetes Debian

copyright Contiene la informacin relativa a la licencia del paquete y del programa original, junto con datos acerca de los autores originales, su copyright y de dnde descargamos el cdigo fuente.

En este caso slo tenemos que rellenar sin ms los campos. No se fuerza ningn formato particular sobre el fichero. Es importante sustituir `pstream Author(s)` por `pstream Authorsz` fijarnos en la informacin en `docs/authors.txt` del cdigo fuente, o los verificadores de paquetes que veremos despus darn avisos al respecto.

dirs En este fichero listamos los directorios en que vamos a instalar algn fichero. Si no lo listamos aqu, dicho directorio no va a hallarse disponible durante la construccin del paquete, as que hay que tener cuidado. Las rutas deben de seguir el Filesystem Hierarchy Standard (FHS), disponible a travs de la orden **man hier** desde cualquier terminal.

Algunas rutas importantes y sus contenidos son:

/bin Ejecutables usados en modo monousuario. Normalmente realizan tareas de mantenimiento a bajo nivel, entre otras cosas. Instalados a travs de paquetes Debian.

/boot Configuracin de GRUB, ficheros de imagen de los *kernels* disponibles, etc.

/dev rbol de directorios donde cada dispositivo conectado al sistema es un fichero.

/etc Ficheros de configuracin global (para todos los usuarios).

/home Directorios de casa de cada usuario, con espacio para cada uno de ellos.

/mnt Dispositivos externos montados temporalmente: particiones de Windows, CD, DVD, pendrives, etc.

/proc rbol de directorios con informacin del *kernel* en cada fichero: procesos en ejecucin, dispositivos disponibles, etc.

/root Directorio de casa del superusuario.

/sbin Ejecutables para uso del superusuario.

/usr Datos, programas y bibliotecas compartidos por todos los usuarios.

/usr/bin Ejecutables para todos los usuarios, instalados a travs de paquetes Debian.

/usr/lib Bibliotecas para todos los usuarios, instalados a travs de paquetes Debian.

/usr/local Similar a `/usr`, pero para uso del administrador.

/usr/share Datos compartidos por todos los usuarios.

/usr/share/man Pginas de *man* disponibles. Toda pgina se halla dentro de una seccin. En particular, la de *zsnes* estara en la 1, tras ver las instrucciones disponibles a travs de la orden **man man**.

Dado que tenemos que instalar el ejecutable para todos los usuarios y una pgina *man*, nuestro fichero `dirs` contendr:

```
usr/bin
usr/share/man/man1
```

NOTA

En este fichero, las rutas *no* incluyen una barra inicial, como suelen hacer. Para ser ms exactos, son rutas a crear dentro del rea temporal de construccin del directorio `debian/zsnes`, donde colocaremos todos los ficheros tal y como se descomprimirn despus bajo el directorio `raz`, `/`.

rules Aqu est el fichero ms importante de todos. Es el que decide qu hay que hacer exactamente para compilar e instalar el paquete completo: documentacin, binarios, guiones y ficheros de datos.

De todas formas, en trminos generales, no es ms que un *makefile*, si bien uno que puede hacerse muy complejo: el objetivo *build* compila, *install* instala dentro del rea de construccin del paquete, y *clean* retira los ficheros generados durante la construccin. Esta ltima se halla bajo la ruta relativa `debian/zsnes` respecto del directorio principal del paquete.

Dado que escribir una y otra vez un *makefile* completo para muchas aplicaciones parecidas era una prdida de tiempo, se han desarrollado diversos paquetes que factorizan cierta funcionalidad comn, como instalar pginas *man*, tipos MIME, entradas de men, y cosas del estilo: son los guiones del paquete `debhelper`. Prcticamente nadie hoy en da desarrolla sus paquetes sin estos guiones.

Algunos desarrolladores han decidido ir un paso ms all, y factorizar reglas para perfiles completos de aplicaciones. As, si sabemos que se trata de una aplicacin desarrollada a travs de las autotools, slo tendremos que aplicar dicho perfil, aadiendo las opciones oportunas que pasar a `configure`, por ejemplo. Esto es el Common Debian Build System (CDBS) [?], que usaremos en esta gua. Existen perfiles para aplicaciones Python, Perl, GNOME, KDE, Java (basadas en Ant), o incluso para aquellas con un simple *makefile*.

Por supuesto, para paquetes complicados, este sistema se queda corto, pero son minora comparados con los dems. Adems, no es slo cuestin de simplicidad: factorizando la

A. Gua de desarrollo de paquetes Debian

mayor proporcin posible de reglas, blindaremos nuestro paquete ante cambios en la poltica de Debian en el futuro.

De todas formas, en general, la comunidad de desarrolladores se halla muy dividida entre usar o no CDBS: aunque factoriza mucha complejidad, resulta difcil de comprender y aprovechar en casos difciles, a menos que seamos capaces de leer complejos ficheros *makefile* por nosotros mismos, dado que no existe mucha documentacin detallada al respecto: para CDBS, el cdigo es la mejor documentacin.

Dado que resulta imposible entender bien CDBS si no se comprende antes el sistema tradicional, en esta seccin explicaremos las dos alternativas. Comenzaremos por el sistema "tradicional" on los guiones de debhelper, y luego veremos cmo CDBS factoriza la mayor parte de estas reglas.

Reglas con *debhelper* Partiendo del esqueleto que automticamente nos ha creado **dh_make**, lo retocamos para este paquete en particular. Vamos a ver qu tal ha quedado, y luego explicaremos qu partes exactamente hemos cambiado, y por qu:

Listado A.1: Reglas de un paquete Debian creado con debhelper

```
#!/usr/bin/make -f
# -*- makefile -*-

# This file was originally written by Joey Hess and Craig Small. As a
# special exception, when this file is copied by dh-make into a dh-make
# output file, you may use that output file without restriction. This
# special exception was added by Craig Small in version 0.37 of dh-make.

# Uncomment this to turn on verbose mode.
#export DH_VERBOSE=1

# El cdigo se halla en un subdirectorio, no en la raz
SRCDIR = src
# Opciones a pasar a configure (--enable-release activa optimizaciones)
CONFIGURE_FLAGS = --disable-cpucheck --enable-release --with-x --with-opengl
# Opciones a usar en el compilador
CFLAGS = -Wall -g

ifneq (, $(findstring noopt, $(DEB_BUILD_OPTIONS)))
    CFLAGS += -O0
else
    CFLAGS += -O2
endif

configure: configure-stamp
configure-stamp:
    dh_testdir
```

```

touch configure-stamp

build: build-stamp

build-stamp: configure-stamp
    dh_testdir

    cd $(SRCDIR) && \
        force_arch=i586 ./configure $(CONFIGURE_FLAGS) --prefix=/usr
    $(MAKE) -C $(SRCDIR)

    touch $@

clean:
    dh_testdir
    dh_testroot
    rm -f build-stamp configure-stamp

    # Limpiamos tambien las herramientas internas usadas por ZSNES en su
    # compilacin
    -$(MAKE) -C $(SRCDIR) clean tclean
    # Borramos los ficheros temporales generados por la compilacin
    $(RM) $(SRCDIR)/tools/depbuild $(SRCDIR)/config.{log,status,h} $(SRCDIR)/Makefile

    dh_clean

install : build
    dh_testdir
    dh_testroot
    dh_clean -k
    dh_installdirs

    $(MAKE) -C $(SRCDIR) install DESTDIR=$(CURDIR)/debian/zsnes

# Build architecture –independent files here.
binary-indep: build install
# We have nothing to do by default.

# Build architecture –dependent files here.
binary-arch: build install
    dh_testdir
    dh_testroot
    dh_installchangelogs
    dh_installdocs
    dh_installexamples
#    dh_install

```

A. Gua de desarrollo de paquetes Debian

```
# dh_installmenu
# dh_installdebconf
# dh_installogrotate
# dh_installemacsen
# dh_installpam
# dh_installmime
# dh_python
# dh_installinit
# dh_instalrcron
# dh_instalinfo
dh_installman $(SRCDIR)/linux/zsnes.1
dh_link
dh_strip
dh_compress
dh_fixperms
# dh_perl
# dh_makeshlibs
dh_installdeb
dh_shlibdeps
dh_gencontrol
dh_md5sums
dh_builddeb
```

binary: binary–indep binary–arch

.PHONY: build clean binary–indep binary–arch binary install configure

Aunque es bastante largo, conceptualmente es sencillo, gracias al uso de *debhelper*. Un par de cosas a destacar:

1. Dado que nuestro cdigo fuente se halla bajo un subdirectorio y no en el directorio raz del paquete, aadimos una variable `SRCDIR`, cuyo valor tendremos en cuenta para realizar un cambio de directorio antes de cada orden de compilacin.
2. Por otro lado, `CURDIR` contiene la ruta del directorio raz actual desde el cual se est construyendo el paquete. La ruta `$(CURDIR)/debian/zsnes` contiene un rbol de directorios que sigue el FHS, y se corresponde con los ficheros contenidos en el paquete `zsnes`.
3. Bajo el objetivo de compilacin `build-stamp` aadimos las rdenes requeridas para compilar: invocamos al guin `configure` con las opciones necesarias e iniciamos la compilacin. La opcin `-C` pasada a **make** hace el cambio de directorio antes de comenzar, justo como `cd` hace para las `cd`s. Hay que hacerlo para cada orden y no al principio debido al hecho de que tras cada orden volvemos al directorio original, al restaurarse el estado anterior del shell.
4. Repetimos el cambio en la invocacin a **make** para los otros objetivos `clean` e `install`. Puede verse cmo se pasa la variable de entorno `DESTDIR` con la ruta

al rea de construccin para la instalacin: evidentemente, el *makefile* debe de estar hecho para tener esto en cuenta. Tenemos la suerte para este paquete de que ya sea as: de lo contrario, tendrmos que adaptar dicho fichero, iy posiblemente el resto del programa!

5. Por ltimo, en el objetivo `binary-arch` tenemos una serie de llamadas a distintos guiones de *debhelper*. Comentaremos y descomentaremos segn nos haga falta: as, por ejemplo, un programa escrito en Perl no necesita **`dh_link`** ni **`dh_strip`**, al no generar ejecutables.

Hemos aadido un argumento a **`dh_installman`** con la pgina *man* que queremos que se instale. Al igual que con todo lo dems, si no hubiera una, tendrmos que crearla nosotros. Lo ms usual en este caso es escribir un fichero SGML o XML DocBook y transformarlo a *nroff* (el formato de las pginas *man*) mediante *docbook-to-man* o una hoja de estilos XSLT, por ejemplo. Podramos partir del ejemplo creado antes por **`dh_make`**>, `zsnes.sgml.ex`.

Un detalle importante: la mayora de los guiones suponen que los ficheros bajo nuestro directorio `debian` siguen una serie de convenciones. Por ejemplo, **`dh_installdocs`** supone que existe algn fichero `debian/zsnes.docs` o `debian/docs` que liste la documentacin a instalar. En este caso, aprovechamos la documentacin que ya trae ZSNES, con lo que tendrmos esto en `debian/docs`:

```
docs/srcinfo.txt
docs/README.SVN
docs/opengl.txt
docs/stdards.txt
docs/authors.txt
docs/todo.txt
docs/install.txt
docs/thanks.txt
docs/support.txt
docs/README.LINUX
docs/readme.txt/about.txt
docs/readme.txt/faq.txt
docs/readme.txt/history.txt
docs/readme.txt/gui.txt
docs/readme.txt/advanced.txt
docs/readme.txt/index.txt
docs/readme.txt/games.txt
docs/readme.txt/netplay.txt
docs/readme.txt/readme.txt
docs/readme.txt/support.txt
docs/readme.htm/styles/release.css
docs/readme.htm/styles/print.css
```

A. Gua de desarrollo de paquetes Debian

```
docs/readme.htm/styles/jipcy.css
docs/readme.htm/styles/radio.css
docs/readme.htm/styles/corner.png
docs/readme.htm/styles/plaintxt.css
docs/readme.htm/styles/shared.css
docs/readme.htm/images/zsneslogo.png
docs/readme.htm/images/netplay.png
docs/readme.htm/images/quick.png
docs/readme.htm/images/saveslot.png
docs/readme.htm/images/cheat.png
docs/readme.htm/images/gui.png
docs/readme.htm/images/config.png
docs/readme.htm/images/game.png
docs/readme.htm/images/fl_menu.png
docs/readme.htm/images/misc.png
docs/readme.htm/netplay.htm
docs/readme.htm/about.htm
docs/readme.htm/games.htm
docs/readme.htm/advanced.htm
docs/readme.htm/gui.htm
docs/readme.htm/support.htm
docs/readme.htm/readme.htm
docs/readme.htm/history.htm
docs/readme.htm/license.htm
docs/readme.htm/faq.htm
docs/readme.htm/index.htm
docs/readme.1st
```

NOTA

Hemos usado ya otro fichero ms del mismo estilo: `dirs` es realmente el fichero que **dh_installdirs** utiliza, por ejemplo.

Reglas con CDBS Ahora que ya sabemos cul es la estructura real de un fichero de reglas, lo reescribiremos empleando CDBS:

Listado A.2: Reglas de un paquete Debain creadas utilizando CDBS

```
#!/usr/bin/make -f
# -*- makefile -*-
```

```
DEB_SRCDIR = src
```

```
include /usr/share/cdb/1/rules/debhelper.mk
include /usr/share/cdb/1/class/autotools.mk

DEB_CONFIGURE_SCRIPT_ENV += force_arch=i586
DEB_CONFIGURE_EXTRA_FLAGS = --disable-cpucheck --with-x \
    --enable-release --with-opengl
DEB_INSTALL_MANPAGES_zsnes += $(DEB_SRCDIR)/linux/zsnes.1
```

Sorprendentemente, esto es todo: ocho lneas. Los dos **include** se ocupan de importar los conjuntos de reglas de apoyo para el uso interno de *debhelper* e implementar el soporte para el perfil de las autotools, respectivamente.

A continuacin, pasamos las mismas opciones a `configure` que antes: pedimos que compile para Pentium o superior, que emplee aceleracin 3D y un interfaz grfico, optimice algo ms de lo normal (`--enable-release`) y no intente autodetectar nuestra CPU. Finalmente, indicamos que instale la pgina `src/linux/zsnes.1` que incluye ZSNES.

Usaremos esta versin de `debian/rules` para realizar el resto del documento, aprovechando algunas funcionalidades adicionales que aporta. Sin embargo, internamente, es exactamente lo mismo de antes.

Construccin preliminar

Con todo listo, ya podemos construir el paquete. Sitndonos en el directorio principal del paquete, `/tmp/packages/zsnes-1.510`, ejecutaremos:

```
dpkg-buildpackage -rfakeroot
```

Tras un cierto tiempo, nos preguntan la contrasea de nuestra clave privada para firmar el paquete de forma automtica. Poco despus, tendremos en `/tmp/packages` nuestra primera versin del paquete Debian, `zsnes_1.510-0ubuntu1_i386.deb`, junto con un fichero `.dsc` que describe el paquete fuente que tambin hemos construido, y un `diff.gz` con las diferencias respecto a las fuentes originales.

Inyeccin en el repositorio

Con nuestra primera versin del paquete lista, slo nos queda inyectar el paquete en el repositorio Subversion. Nos situaremos en `~/packages` y ejecutaremos:

```
svn-inject -c2 -o /tmp/packages/zsnes_1.510-0ubuntu1.dsc \
    file:///home/tunombredeusuario/.svnDebian
```

A. Gua de desarrollo de paquetes Debian

Tras un cierto tiempo, ya tendremos enviado al repositorio central nuestro paquete, y nuestra copia de trabajo habr sido creada, con lo que no necesitaremos ms `/tmp/packages`.

La opcin `-o` evita que se guarde el cdigo fuente en el repositorio, usando nicamente archivos `tar.gz` en el subdirectorio `tarballs` del directorio principal del repositorio, que no se hallar bajo control de versiones.

El repositorio creado tiene la siguiente estructura:

- `~/packages/tarballs` Contiene los ficheros `tar.gz` con las fuentes originales de nuestros paquetes.
- `~/packages/zsnes/build-area` Se trata del directorio destino en el que se depositarn todos los paquetes y dems ficheros que vayamos produciendo.
- `~/packages/zsnes/branches` Almacena las distintas ramas de desarrollo. Normalmente contendra las distintas versiones del cdigo original, pero al usar `tarballs`, es prcticamente intil en nuestro caso.
- `~/packages/zsnes/tags` Permite asociar nmeros de versin con determinadas revisiones del repositorio. Posteriormente veremos cmo se usa.
- `~/packages/zsnes/trunk` Aqu se almacena la versin actual del paquete. Slo almacenamos el directorio `debian`, para ahorrar la complejidad y tiempo de descarga necesario de otra forma.

A.3.2. Preparacin de una primera versin

Construccin definitiva

Es el momento de reconstruir el paquete, pero esta vez haciendo uso de la jaula *chroot*, para asegurar que efectivamente hemos listado todas las dependencias correctamente.

Dado que la orden es bastante larga, lo mejor es aadir un alias a dicha orden, o mejor an, declarar una funcin de *dash* (la versin reducida del shell *bash* de Ubuntu) que haga dicha tarea, en `~/ .bashrc`. Aadiremos estas lneas:

```
function svn-b ()
  buildarea="`pwd`/../build-area";
  sudo cowbuilder --update
  svn-buildpackage \
    --svn-builder="pdebuild --auto-debsign --buildresult $buildarea"
    --svn-postbuild="rm $buildarea/*_source.changes";
```

```
function svn-tag ()  
    svn-buildpackage --svn-only-tag
```

Nos situaremos en `~/packages/zsnes/trunk` y ejecutaremos:

```
svn-b
```

Tras un tiempo prudencial, e introducir nuestras contraseas de GPG y superusuario, tendremos una primera versin del paquete, comprobada dentro de una jaula *chroot*.

Verificacin

Sin embargo, no basta con que se compile e instale correctamente. Adems, el paquete debe de cumplir todas las polticas de Debian, como pertenecer a un determinado conjunto de secciones, tener todos sus ficheros de control bien escritos, respetar el estndar FHS que antes mencionamos, etc.

De hecho, si queremos que nos lleguen a aceptar cualquier paquete en un repositorio Debian oficial, como la seccin *universe* o *multiverse* de Ubuntu, o la seccin *unstable* o *testing* de Debian, lo mejor que podemos hacer para evitar hacer perder tiempo a nuestro supervisor o supervisores (que sern los que efectivamente suban al repositorio el paquete las primeras veces) es pasar antes nuestro paquete por los dos verificadores disponibles: Linda y Lintian, y retirar todos los avisos.

NOTA

Segn parece, Linda, escrito en Python, es ms reciente que Lintian, y tambin ms rpido. Por otro lado, Lintian est escrito en Perl. Aparte de eso, no hay muchas diferencias: simplemente ejecutaremos ambos por seguridad, ya que tampoco supone un esfuerzo adicional considerable.

Lanzaremos los verificadores sobre el `.deb`, que se hallar en `~/packages/zsnes/-build-area`, pidiendo explicaciones de los avisos y errores con `-i`:

```
linda -i ~/packages/zsnes/build-area/zsnes_1.510-0ubuntu1_i386.deb  
lintian -i ~/packages/zsnes/build-area/zsnes_1.510-0ubuntu1_i386.deb
```

Segn parece, no todo est bien en nuestro paquete:

A. Gua de desarrollo de paquetes Debian

```
W: zsnes: extra-license-file usr/share/doc/zsnes/license.htm
N:
N:   All license information should be collected in the debian/copyright
N:   file. This usually makes it unnecessary for the package to install
N:   this information in other places as well.
N:
N:   Refer to Policy Manual, section 12.5 for details.
N:
E: zsnes: FSSTND-dir-in-usr usr/man/
N:
N:   As of policy version 3.0.0.0, Debian no longer follows the FSSTND.
N:
N:   Instead, the Filesystem Hierarchy Standard (FHS), version 2.3, is
N:   used. You can find it in /usr/share/doc/debian-policy/fhs/ .
N:
E: zsnes; Manual page zsnes.1.gz installed into /usr/man.
  The manual page shown is installed into the legacy location of
  /usr/man, where they should be installed into /usr/share/man.
E: zsnes; FSSTND directory /usr/man in /usr found.
  As of policy version 3.0.0.0, Debian no longer follows the FSSTND.
  Instead, the Filesystem Hierarchy Standard (FHS), version 2.1, is
  used. You can find it in /usr/share/doc/debian-policy/fhs/ .
E: zsnes; FSSTND directory /usr/man/man1 in /usr found.
W: zsnes; File /usr/share/doc/zsnes/license.htm is considered to be an
  extra license file. The file shown above is considered to be another
  license file, where as the license for a package should be contained
  in the copyright file, which should be installed into
  /usr/share/doc/<pkg>.
```

Analizando esta salida, vemos que hay cuatro problemas con nuestro paquete segn Linda, y dos segn Lintian. El primer problema es fcil de tratar: hemos incluido un fichero de licencia con la documentacin (extra-license-file), que no debera estar, ya que el archivo `copyright` debera contener toda la informacin. Basta en este caso con retirar la lnea problemtica de `debian/docs`.

El segundo problema es ms complejo: el directorio `/usr/man` no es parte del estndar actual, FHS, sino de uno ms antiguo, el FSSTND (FileSystem Standard). En principio, nuestras reglas estn bien escritas. Veamos si efectivamente nuestro paquete pone la pgina `man` en su sitio:

```
~/packages/zsnes/trunk$ dpkg \
-c ../build-area/zsnes_1.510-0ubuntu1_i386.deb
```

S que lo hace: `/usr/share/man/man1/zsnes.1.gz` aparece como debera. Sin embargo, tambin aparece `/usr/man/man1/zsnes.1.gz`. As, hay algo fuera de nuestras reglas que est instalando en dicho sitio la pgina *man*.

Vamos a descomprimir en un directorio temporal el cdigo fuente original, y echar un vistazo con la inestimable ayuda de *grep*, empleando la opcin *-R* para hacer una bsqueda recursiva por el rbol de directorios y *-l* para solamente listar los ficheros con coincidencia y no el contenido que sigue el patrón:

```
cd /tmp
tar -xzf ~/packages/tarballs/zsnes_1.510.orig.tar.gz
cd zsnes-1.510.orig
grep -Rl zsnes.1 *
docs/srcinfo.txt
docs/readme.txt/readme.txt
docs/readme.htm/readme.htm
src/Makefile.in
```

Ya lo hemos encontrado: descartando los tres ficheros que forman parte de la documentacin, slo queda *src/Makefile.in*. Tiene que ser el culpable, ya que este fichero es usado por el guin *configure* para crear el *makefile* con el que hacer la compilacin e instalar el programa.

Veremos exactamente dnde est el problema, empleando la opcin *-C* para ver un contexto de un determinado número de líneas alrededor de las apariciones:

```
grep -C4 zsnes.1 src/Makefile.in
install:
    @INSTALL@ -d -m 0755 $(DESTDIR)/@prefix@/bin
    @INSTALL@ -m 0755 @ZSNESEXE@ $(DESTDIR)/@prefix@/bin
    @INSTALL@ -d -m 0755 $(DESTDIR)/@prefix@/man/man1
    @INSTALL@ -m 0644 linux/zsnes.1 $(DESTDIR)/@prefix@/man/man1
uninstall:
    rm -f @prefix@/bin/$(notdir @ZSNESEXE@) @prefix@/man/man1/zsnes.1

clean:
    rm -f $(Z_OBJS) $(PSR) $(PSR_H) @ZSNESEXE@
tclean:
```

Ya hemos encontrado las dos líneas problemáticas, al final del objetivo *install*: crean el directorio e instalan la página *man*. Slo hay que quitarlas. Sin embargo, hay un problema: no podemos tocar el código directamente, ya que no lo tenemos bajo control de versiones. Es más: no se considera buena práctica, ya que en futuras versiones los desarrolladores originales podran arreglar ese problema, y se hara difícil ver quin ha hecho qu tras hacer unos cuantos cambios más.

Por eso, se considera buena práctica emplear un sistema de parches para hacer este tipo de modificaciones. Algunas alternativas [?] incluyen el más antiguo, *dpatch*, el más

A. Gua de desarrollo de paquetes Debian

reciente (y aparentemente superior, segn los desarrolladores de SUSE) *quilt* y el que usaremos aqu, *simple-patchsys*, que como su nombre indica, est ms inclinado hacia ser sencillo que ser potente. Sin embargo, nos basta para casos sencillos como ste.

Para usarlo, aadimos la lnea que instala soporte para l en `debian/rules`, justo despus del primer `include`:

```
include /usr/share/cdb/1/rules/simple-patchsys.mk
```

Pasaremos a crear el parche en s. Para ello, tenemos que obtener el rbol de fuentes tal y como lo vera `debuild` antes de compilar, y no como lo tenemos ahora. Para ello, usaremos **svn-buildpackage**, no sin antes enviar nuestros otros cambios al repositorio (de otra forma, no nos dejar seguir):

```
~/packages/zsnes/trunk$ svn commit -m \
    "Integrado simple-patchsys del CDBS y corregido documentacin"
Enviando          trunk/debian/docs
Enviando          trunk/debian/rules
Transmitiendo contenido de archivos ..
Commit de la revisin 6.
~/packages/zsnes/trunk$ svn-buildpackage --svn-export
buildArea: /home/antonio/packages/zsnes/build-area
[...]
```

I: mergeWithUpstream property set, looking for upstream source tarball...

[...]

Exportacin completa.

```
rm -rf /home/antonio/packages/zsnes/build-area/tmp-0.00194501814349124
Build directory exported to /home/antonio/packages/zsnes/build-area/zsnes-1.
```

Iremos al directorio exportado de construccin, que contendr todos los ficheros necesarios, y generaremos el parche usando **cdbedit-patch**:

```
~/packages/zsnes/trunk$ cd ../build-area/zsnes-1.510/
~/packages/zsnes/build-area/zsnes-1.510$ cdbedit-patch 01-man-fhs.p
```

El programa nos indicar que nos hallamos ahora en un subshell, y que hagamos las modificaciones necesarias. Usando un editor cualquiera, retiraremos dichas lneas de `src/Makefile.in`. Hecho esto, pulsaremos CTRL + D o introduciremos la orden **exit** para salir del subshell y dejar que cree el parche `debian/patches/01-man-fhs.patch` con los cambios que hemos hecho.

Slo hemos de colocar ahora el parche en el sitio correcto, y guardar los cambios en el repositorio:

A.3. Creacin y mantenimiento del paquete

```
cd ~/packages/zsnes/trunk
cp -r ../build-area/zsnes-1.510/debian/patches debian
svn add debian/patches
svn commit -m "Arreglado problema con la pgina man en /usr/man"
```

Ahora reconstruiremos el paquete:

```
svn-b
```

Ya, por fin, Lintian y Linda no dan ningn aviso. Probaremos a instalar el paquete, y ver qu tal funciona:

```
sudo dpkg -i ../build-area/zsnes_1.510-0ubuntu1_i386.deb
```

Tras probar un poco el emulador con alguna ROM libremente disponible, como las de PDRoms (<http://www.pdroms.com>), decidimos que el paquete est en condiciones de ser usado. Usaremos la otra funcin que definimos antes para marcar esta versin del paquete, copiando los contenidos de `trunk` en un nuevo subdirectorio de `tags` cuyo nombre ser la versin actual del paquete, para despues confirmar los cambios que se habrn realizado automticamente en `debian/changelog`, en preparacin para la siguiente versin de nuestro paquete:

```
svn-tag
svn commit -m "Copiado a tags versin 1.510-0ubuntu1 del paquete"
```

Envio al repositorio Debian

Despus de haber verificado nuestro paquete y haber marcado la versin en el repositorio Subversion, es momento de enviarlo al repositorio Debian, para que nuestros usuarios puedan acceder fcilmente a l. Aadiremos tanto el paquete fuente como el paquete compilado:

```
sudo -E reprepro includedeb gutsy \
~/packages/zsnes/build-area/zsnes_1.510-0ubuntu1_i386.deb
sudo -E reprepro -S main -P low includedsc gutsy \
~/packages/zsnes/build-area/zsnes_1.510-0ubuntu1.dsc
```

Deberamos probar el paquete justo como un usuario normal lo hara. Pero, primero, tenemos que desinstalar la versin que instalamos antes a partir de un fichero:

```
sudo aptitude remove zsnes
sudo aptitude update
sudo aptitude install zsnes
```

A.3.3. Actualizacin del paquete a una nueva versin del programa

Cada cierto tiempo, deberemos de actualizar nuestro repositorio con los cambios realizados en el cdigo. Para ello, disponemos de la orden **svn-upgrade**, a la que le pasaremos un fichero `tar.gz` con el nuevo cdigo fuente del paquete.

Ahora probaremos a actualizar nuestro paquete con el cdigo correspondiente a la ltima versin en el repositorio Subversion del equipo de ZSNES. Primero, prepararemos el tarball con el cdigo original. Descargamos el cdigo fuente, y lo exportamos a otro directorio, para retirar los ficheros usados por *Subversion*:

```
svn checkout https://svn.bountysource.com/zsnes/trunk zsnes-svn
svn export zsnes-svn zsnes-1.510.SVN5215
```

He decidido usar el esquema `1.510.SVN5215` para indicar que se trata de la revisin 5215 del repositorio. De esta forma, si sale una nueva versin oficial, o si empleo una revisin ms reciente del repositorio Subversion, como `1.520` o `1.510.SVN5216`, stas sern consideradas ms recientes, y la actualizacin se podr hacer de forma correcta. Podemos comprobar que es efectivamente as mediante la siguiente orden:

```
dpkg --compare-versions "1.510-0ubuntu1" gt "1.510.SVN5215-0ubuntu1"
echo "1.510-0ubuntu1 mayor que 1.510.SVN5215-0ubuntu1" || \
echo "1.510-0ubuntu1 menor que 1.510.SVN5215-0ubuntu1"
```

Ahora que ya tenemos el cdigo, inspeccionaremos un momento para ver si todos los ficheros importantes se hallan en su sitio. Normalmente, no se suelen almacenar ficheros generados automticamente en el repositorio, as que pueden que falten algunas cosas importantes. Mirando en `src`, vemos que falta el guin `configure` que necesitamos para compilar. Lo generaremos ejecutando las siguientes rdenes bajo `src`, el mismo directorio donde se halla su fichero fuente, `configure.in`:

```
aclocal
autoconf
```

Tambin faltan los directorios `docs/readme.txt` y `docs/readme.htm`, que aadiremos en su sitio. Con esto tenemos los ficheros fuente listos. Vamos a crear el tarball que necesitamos:

```
/tmp$ tar -czf zsnes-1.510.SVN5215.tar.gz zsnes-1.510.SVN5215
```

A.3. Creacin y mantenimiento del paquete

El siguiente paso es aadir dicho cdigo a nuestra rea de trabajo, e indicar que vamos comenzar a empaquetar una nueva versin del programa. Volvemos al directorio con la versin actual de nuestro paquete y ejecutamos:

```
~/packages/zsnes/trunk$ svn-upgrade /tmp/zsnes-1.510.SVN5215.tar.gz
```

Se harn los cambios necesarios en `debian/changelog` y el resto del repositorio. Cambiaremos el nmero de versin del paquete a `0ubuntu1` y confirmamos dichos cambios antes de volver a reconstruir el paquete:

```
~/packages/zsnes/trunk$ svn commit -m \  
  "Actualizado con rev 5215 upstream"  
~/packages/zsnes/trunk$ svn-b
```

Sin embargo, la reconstruccin falla. Buscando entre los distintos mensajes, podemos ver una lnea del estilo:

```
Trying patch 01-man-fhs.patch at level 1 ... 0 ... 2 ... failed.
```

En resumen: el parche que antes hicimos para corregir el problema de `Makefile.in` no se ha podido aplicar. Tras inspeccionar sus contenidos, vemos que efectivamente `Makefile.in` ha cambiado demasiado:

```
install:  
@INSTALL@ -d -m 0755 $(DESTDIR)/@bindir@  
@INSTALL@ -m 0755 @ZSNESEXE@ $(DESTDIR)/@bindir@  
@INSTALL@ -d -m 0755 $(DESTDIR)/@mandir@/man1  
@INSTALL@ -m 0644 linux/zsnes.1 $(DESTDIR)/@mandir@/man1
```

Vaya, parece que ellos mismos han corregido ya el problema que tenamos antes. sta es precisamente la razn por la que usamos un parche, en vez de cambiar directamente el cdigo. Slo tenemos que eliminar el parche, confirmar los cambios y reconstruir:

```
~/packages/zsnes/trunk$ svn rm debian/patches  
D  debian/patches/01-man-fhs.patch  
~/packages/zsnes/trunk$ svn commit -m \  
  "01-man-fhs.patch: Retirado (corregido por upstream)"  
Deleting          zsnes/trunk/debian/patches/01-man-fhs.patch  
Transmitting file data .  
Committed revision 11.  
~/packages/zsnes/trunk$ svn-b
```

A. Guia de desarrollo de paquetes Debian

Esta vez ya no da el fallo del parche, pero indica que faltan las bibliotecas de desarrollo de Qt para la nueva interfaz. Aadiremos la dependencia en `libqt4-dev` al fichero `control` y volveremos a intentarlo tras confirmar otra vez nuestros cambios.

Esta vez falla diciendo que no puede construir el fichero `ui_zsnes.h`, que hace falta para compilar. Probaremos a compilar sobre el directorio `/tmp/zsnes-1.510.SVN5215/-src` tras instalar las dependencias en nuestro propio sistema:

```
./configure
make
```

Da el mismo problema, as que no es culpa de nuestro paquete. Vamos a mirar con *grep*, a ver qu puede ser:

```
grep -R ui_zsnes.h *
makefile.ms:$GUI_D/gui.cpp: $GUI_D/ui_zsnes.h
src/gui/gui.h:#include "ui_zsnes.h"
Makefile.in:GUI_QO=$(GUI_D)/moc_gui.cpp $(GUI_D)/ui_zsnes.h
Makefile:GUI_QO=$(GUI_D)/moc_gui.cpp $(GUI_D)/ui_zsnes.h
makefile.dep:gui/gui.o: gui/gui.cpp gui/gui.h ui_zsnes.h
```

Si nos fijamos, se puede ver que en `makefile.dep` la ruta no se corresponde con la de las anteriores entradas, por lo que seguramente ah estar el fallo. Esta vez buscamos por este fichero:

```
grep -R makefile.dep *
src/configure:touch -t 198001010000 makefile.dep
src/Makefile.in:main: makefile.dep $(Z_QOBS) $(Z_OBS)
src/Makefile.in:include makefile.dep
src/Makefile.in:makefile.dep: $(TOOL_D)/depbuild Makefile
src/Makefile.in: $(TOOL_D)/depbuild @CC@ "@CFLAGS@" @NASMPATH@
"@NFLAGS@" $(Z_OBS) > makefile.dep
src/Makefile.in: rm -f makefile.dep $(Z_OBS) $(Z_QOBS) $(PSR)
$(PSR_H) @ZSNESEXE@
src/Makefile:main: makefile.dep $(Z_QOBS) $(Z_OBS)
src/Makefile:include makefile.dep
src/Makefile:makefile.dep: $(TOOL_D)/depbuild Makefile
src/Makefile: $(TOOL_D)/depbuild gcc " -pipe -I. -I/usr/local/include
-I/usr/include -D__UNIXSDL__ -I/usr/include/SDL -D_GNU_SOURCE=1
-D_REENTRANT -D__OPENGL__ -DNO_DEBUGGER -DNDEBUG -march=athlon-xp -O2
-fomit-frame-pointer -s -DQT_SHARED -I/usr/include/qt4
-I/usr/include/qt4/QtCore -I/usr/include/qt4/QtGui " nasm "
-w-orphan-labels -D__UNIXSDL__ -f elf -DELFB -D__OPENGL__ -DNO_DEBUGGER
```

A.3. Creacin y mantenimiento del paquete

```
-O1" $(Z_OBJS) > makefile.dep
src/Makefile: rm -f makefile.dep $(Z_OBJS) $(Z_QOBS) $(PSR) $(PSR_H)
zsnes
src/autom4te.cache/output.0:touch -t 198001010000 makefile.dep
src/autom4te.cache/output.1:touch -t 198001010000 makefile.dep
src/configure.in:touch -t 198001010000 makefile.dep
```

Puede verse que en `src/Makefile` es donde se crea a travs de una herramienta propia de los desarrolladores de ZSNES, que obtiene automticamente las dependencias. Recordando la lista anterior de ficheros que mencionaban a `ui_zsnes.h`, se nos viene a la cabeza `gui/gui.h`. Vamos a probar a cambiar la ruta de inclusi3n a `gui/ui_zsnes.h`:

```
cd ~/packages/zsnes/trunk
svn-buildpackage --svn-export
cd ../build-area/zsnes-1.510.SVN5215
cdbs-edit-patch 02-fix-depbuild.patch
```

Hacemos el cambio antes mencionado con el editor *vim* y salimos del shell, tras lo cual reintentaremos la construcci3n del paquete:

```
vim src/gui/gui.h
exit
```

Ahora la reconstrucci3n ha tenido xito. Ya slo tendremos que seguir los mismos pasos anteriores de verificaci3n, validaci3n manual, marcado en el repositorio (tras retirar el aviso «NOT RELEASED YET» aadido automticamente a `debian/changelog`, claro), y envo. Cualquiera de nuestros usuarios ser notificado eventualmente de la actualizaci3n y podr instalar la versi3n ms reciente del paquete.

Una ltima nota: si se quiere, se puede integrar la verificaci3n y publicaci3n en la funci3n **svn-b** antes definida, cambiando las lneas correspondientes en `~/bashrc` por:

Listado A.3: Funciones Bash de ayuda para construcci3n de paquetes

```
function svn-b () {
    buildarea="`pwd`/../build-area";
    ruta paquete="${buildarea}/${package}_${debian_version}*.deb";
    rutadsc="${buildarea}/${package}_${debian_version}*.dsc";
    rutarepo="/var/packages/ubuntu";
    sudo cowbuilder --update
    svn-buildpackage \
        --svn-builder="pdebuild --auto-debsign --buildresult ${buildarea}" \
        --svn-postbuild="rm ${buildarea}/*_source.changes; \
```

A. Gua de desarrollo de paquetes Debian

```
lintian -i ${rutapaquete}; linda -i ${rutapaquete}; \  
cd ${rutarepo} && \  
sudo -E reprepro remove gutsy \${package} && \  
sudo -E reprepro includedeb gutsy ${rutapaquete} && \  
sudo -E reprepro -S main -P low includedsc gutsy ${rutadsc}";  
}  
function svn-tag () {  
    svn-buildpackage --svn-only-tag  
}
```

A.4. Otros aspectos de inters

A.4.1. Integracin de repositorio propio con la jaula *chroot*

En este tutorial no har falta, pero s es posible que nos haga falta para casos ms complejos. Cuando haya interdependencias entre varios paquetes que deseemos crear, tendremos que aadir nuestro repositorio (que estar alojado en su propio servidor web) a la lista de los repositorios de la jaula *chroot*.

Para ello, nos introduciremos en la jaula e importaremos la firma digital de nuestros paquetes:

```
sudo cowbuilder --login --save-after-login  
wget http://direccion.delservidor.org/claveDebian.asc  
apt-key add claveDebian.asc
```

Con esto, la jaula ya confiar en nuestros paquetes. Tenemos que aadir entonces la siguiente lnea a `/etc/apt/sources.list`, usando el editor *vim*:

```
deb http://direccion.delservidor.org/ubuntu gutsy main
```

Actualizamos la lista de paquetes:

```
aptitude update
```

Hemos terminado, as que slo queda salir del shell de la jaula:

```
exit
```

A.4.2. Sincronizacin de un repositorio en Internet con un repositorio local

En muchos casos no dispondremos de los medios necesarios como para dejar nuestro ordenador como servidor web al exterior para que sirva nuestros paquetes. En estos casos, podemos subir nuestros ficheros `.deb` a una forja o algo del estilo y dejar que los usuarios se los bajen.

Pero, ¿y si son muchos paquetes, o si queremos mantener actualizados a los usuarios? Lo mejor sera replicar completo el repositorio en un servidor conectado permanentemente a Internet. La mejor opcin es, por tanto, obtener espacio de alojamiento con acceso mediante SFTP o FTP.

Una vez lo hayamos conseguido, podramos subir el rbol completo de directorios de nuestra mquina al servidor remoto, pero hacer esto una y otra vez tardara demasiado. La mejor opcin es hacer un *mirror inverso* a travs de la herramienta *lftp*, que nos pedir la contrasea antes de conectarse:

```
lftp -d -e "mirror -venR ruta local \
                ruta remota \
                direccin del servidor"
```

Si queremos que no nos pregunte una y otra vez la contrasea, siempre podemos aadir lneas como stas a `~/.netrc`, cuidando de que slo sea legible por nosotros:

```
machine mi.servidor.com
login minombredeusuario
password micontrasea
```

A.4.3. Adaptacin de aplicaciones Java

La principal diferencia al crear un paquete Java es, sin duda, el hecho de que, a pesar de tener que compilar, el paquete en s no tiene ninguna arquitectura definida. Ello debera verse reflejado en `debian/control`.

Adems, debera tener en `Build-Depends` algn compilador de Java, y en `Depends` el paquete virtual `java2-runtime`, adems de una alternativa concreta, como la de Sun, `sun-java6-jre`, o preferiblemente la basada en OpenJDK, `icedtea-java7-jre`, usando el operador lgico OR (`|`). Los paquetes virtuales no aaden ninguna funcionalidad: se limitan a hacer cosas como ayudarnos a instalar varios paquetes de una sola vez (usndolos en su campo `Depends`), o permitindonos usar un paquete cualquiera que nos provea de una cierta funcionalidad.

A. Gua de desarrollo de paquetes Debian

As, si curioseamos en el paquete `sun-java6-jre`, veremos que tiene a dicho paquete virtual en el campo `Provides` («Proporciona»):

```
$ aptitude show sun-java6-jre
Paquete: sun-java6-jre
[...]
Proporciona: java-virtual-machine, java1-runtime, java2-runtime
[...]
```

Si usamos el compilador de Sun en `Build-Depends` (paquete `sun-java6-jdk`), habremos de cambiar nuestro `.pbuilder` para que use un interfaz distinto de entrada, que nos deje aceptar los trminos de la DLJ de Sun. Aadimos estas lneas:

```
# Para poder aceptar la licencia DLJ
export DEBIAN_FRONTEND="readline"
```

Adems de eso, tendremos como de costumbre que adaptar nuestro programa Java para que sea fcilmente compilable de forma automtica, y para que se integre bien dentro del sistema. Un problema es que no deberamos escribir rutas absolutas en el cdigo Java, o si no perderemos toda la transportabilidad que deseamos en un primer momento. La forma ms fcil de implementar lo que deseamos es simplemente utilizar variables de entorno o propiedades del sistema.

A veces no nos servir cualquier JRE de los disponibles para ejecutar nuestro programa, y tendremos que prescindir de usar `java2-runtime`. Es el caso usual de las aplicaciones basadas en Swing: el JRE por defecto en Gutsy, GCJ, slo implementa completamente SWT. Tendremos que hacer que el sistema instale y active otro JRE.

Para poder establecerlas de forma separada del programa Java antes de lanzarlo, lo que necesitamos es un guin de shell, que ser el que instalaremos en `/usr/bin`. El fichero `.jar`, siguiendo la poltica de Debian, debe ir en `/usr/share/java`.

El que sea fcilmente compilable de forma automtica depende de qu sistema hayamos estado usando. Si nos hemos basado en un IDE, seguramente aqu tengamos problemas. Lo que deberamos hacer es reescribir la fase de compilacin usando un guin para *Apache Ant*. Esto, de paso, nos permitir aprovechar el perfil de CDBS, simplificando bastante nuestra tarea. *Ant* es efectivamente un sustituto del *GNU Make* escrito en Java, para aplicaciones Java. Utiliza ficheros muy del estilo de los *makefile*, slo que escritos en XML.

Con todo esto, an hay un par de complicaciones adicionales. Normalmente, los ficheros fuente Java se hallan escritos en UTF-8. Sin embargo, la jaula *chroot* que antes hicimos en A.2.2 (pgina 205) no tiene en principio una localizacin compatible instalada. Vamos a tener que entrar en la jaula e instalar los paquetes necesarios, comprobando la situacin que aqu describimos con **locale** (nos mostrar que usamos la localizacin estndar «C»):


```
sudo cowbuilder --login --save-after-login  
aptitude install language-pack-es  
exit
```

Ahora, lo que hemos de hacer es asegurarnos que cuando llamemos a *Ant*, lo hagamos estableciendo una localizacin con UTF-8:

```
LANG=es_ES.UTF-8 ant compile
```

Deberamos adems invocar las pruebas de unidad sobre el cdigo, para asegurarnos de que el cdigo usado en el paquete no presente regresiones. Adems de aadir el paquete `junit` a `Build-Depends`, tenemos que tener en cuenta que si en alguna prueba de unidad usamos un componente `Swing`, aunque no se muestre nada en pantalla, requeriremos un servidor `X`, o de lo contrario la prueba fallar. Dado que ejecutar un servidor `X` completo en una jaula *chroot* es un gasto intil de recursos, vamos a usar un falso servidor `X`, que aadiremos tambin a `Build-Depends`: `xvfb`. Para tener disponible el servidor `X` durante una orden, nicamente ponemos **`xvfb-run`** antes de ella. Combinndolo con lo de las localizaciones, tendremos que ejecutar una orden como sta:

```
LANG=es_ES.UTF-8 xvfb-run ant (objetivo-compilacin)
```

A.4.4. Actualizacin del escritorio

Antes conseguimos crear un paquete con un binario, junto con su pgina *man*. Sin embargo, para que realmente consideremos al paquete completo, deberamos integrarlo con el gestor de escritorio del usuario. Veremos cmo aadir el acceso directo al men, y cmo asociar los ficheros ROM de Super Nintendo al emulador ZSNES.

Esta tarea se ha hecho en los ltimos aos mucho ms sencilla gracias a las especificaciones ofrecidas por el grupo `FreeDesktop.org` [?], que hace de centro de reunin para diversos proyectos relacionados con entornos grficos. As, `KDE` y `GNOME` emplean el mismo formato para describir las entradas de su men, por ejemplo.

Accesos directos e iconos

Normalmente, existen dos sitios del men de aplicaciones donde podemos instalar un acceso directo a nuestra aplicacin: el men `Debian`, y el propio men normal de aplicaciones. Por completitud, trataremos ambos, aunque hoy en da el men `Debian` no se usa demasiado. Adems, no aparece por omisin en un sistema `Ubuntu`: hay que instalar el paquete `menu` para poder usarlo.

A. Gua de desarrollo de paquetes Debian

Para que se aada una entrada al men Debian, hemos de asegurarnos de que la lnea con **dh_installmenu** se halle descomentada en `debian/rules`. Dado que nuestro paquete se llama `zsnes`, crearemos un fichero `debian/zsnes.menu` con este contenido:

```
?package(zsnes):needs="X11" section="Games/Arcade" \  
  title="ZSNES" command="/usr/bin/zsnes" \  
  icon="/usr/share/pixmaps/zsnes.xpm"
```

El significado de los campos es el siguiente:

needs="X11" Indica que el programa tiene una interfaz grfica, requiriendo un servidor X.

section="Games/Arcade" Especifica dnde debera situarse el acceso directo. Las secciones se hallan prefijadas por la poltica de empaquetado de Debian.

title="ZSNES" ste es el texto que se mostrar en la entrada del men.

command=/usr/bin/zsnes" sta es la orden exacta que ser ejecutada.

icon=/usr/share/pixmaps/zsnes.xpm" Tal y como indica el grupo FreeDesktop.org, hemos instalado el icono en formato XPM y en la ruta `/usr/share/pixmaps`. Para convertir dicha imagen, podemos usar simplemente *Gimp*, o el programa **convert** del paquete `imagemagick`, de esta forma:

```
convert zsnes.png zsnes.xpm
```

Ahora hemos de centrarnos en que aparezca en los mens de KDE y GNOME. Para que sea as, hemos de crear otro fichero ms, `debian/zsnes.desktop`, con el siguiente contenido:

```
[Desktop Entry]  
Name=ZSNES  
Type=Application  
Comment=SNES (Super Nintendo) emulator that uses x86 assembly  
Comment[es]=Emulador de SNES que emplea cdigo ensamblador de x86  
Exec=/usr/bin/zsnes %f  
Icon=zsnes.png  
MimeType=application/x-snes-rom;  
Categories=Game;Emulator;
```

Dicho fichero ha de hallarse codificado tal y como indica el campo `Encoding`: en UTF-8. Adems de indicar el nombre y el tipo de la aplicacin, podemos especificar una descripcin ms larga. Adelanto un detalle para realizar la asociacin de ficheros: el campo `Exec`, que contiene la orden a ejecutar, permite especificar sustituciones como `%f`, que ser sustituido por el primer fichero con el que se active el acceso directo. Si hay varios, se ejecutar el emulador tantas veces como ficheros haya, con un fichero distinto cada vez.

En `Categories` incluimos las categoras a las que pertenece esta aplicacin: son ms bien anotaciones semnticas que una descripcin jerrquica de dnde debera colocarse exactamente. Debe de haber al menos una de las categoras principales segn [?] (o si no no aparecer en el men), y pueden haber categoras adicionales para dar ms informacin.

Los campos con texto descriptivo como `Name` y `Comment` se pueden personalizar segn el idioma aadiendo al nombre de campo, entre corchetes, un identificador de idioma de ISO 639. He incluido un ejemplo para `Comment`. El campo `Icon` no requiere la ruta, ya que el directorio `/usr/share/pixmaps` es uno de los consultados de forma automtica.

Otro detalle tambin importante para crear la asociacin de fichero es especificar el tipo MIME de los ficheros que esta aplicacin abre. Dado que no existe un tipo estndar MIME para ROM de Super Nintendo, definimos uno nosotros, cuidando de poner el prefijo `x-` en la segunda mitad, indicando que no es estndar.

Validaremos el fichero mediante **`desktop-file-validate`**, sin ms problemas. En caso contrario habra que hacer las correcciones oportunas.

Ahora que tenemos los dos ficheros con la informacin necesaria, vamos a definir las acciones requeridas para instalarlos debidamente. En primer lugar, la llamada que necesitamos hacer a **`dh_installmenu`** ya la hace CDBS por nosotros, con lo que ya tendremos la parte de instalar la entrada en el men Debian. An tenemos que instalar el fichero `.desktop`, el icono, y aadir el icono al cach de GNOME, para mejorar la eficiencia. Aadimos estas lneas a `debian/rules`:

```
install/zsnes::
    dh_install -m 0644 $(CURDIR)/debian/zsnes.desktop \
                  /usr/share/applications
    dh_install -m 0644 $(CURDIR)/debian/zsnes.xpm \
                  /usr/share/pixmaps
    dh_desktop
    dh_iconcache
```

Cuando reconstruyamos el paquete, veremos que nos aparecern dos ficheros nuevos en el directorio `debian`: `zsnes.postrm.debhelper` y `zsnes.postinst.debhelper`.

A. Gua de desarrollo de paquetes Debian

Estos ficheros son guiones Bash generados automaticamente que se ejecutan despues de la desinstalacin e instalacin. Tambin hay guiones equivalentes para antes de la instalacin y la desinstalacin, llamados `preinst` y `prerm`, respectivamente. El sufijo `.debhelper` indica al mecanismo de construccin de paquetes que debe concatenar su contenido con el que pudieran tener los ficheros sin tal sufijo, como `zsnes.postrm`, que nosotros mismos habramos escrito manualmente.

En particular, **dh_desktop** ha aadido una llamada a `update-desktop-database`, y **dh_installmenu** otra llamada a `update-menu`. Con esto, nos aseguramos de que el men sea actualizado debidamente y que el entorno de escritorio sepa que nuestro programa est disponible para abrir cualquier ROM de Super Nintendo.

Posteriormente, tras instalar el paquete, dichos guiones son guardados en `/var/lib/dpkg/info`. Si alguna vez nos equivocamos al escribir alguno de ellos, es posible que no podamos ni terminar de desinstalar ni de instalar el paquete. Podemos forzar a que su ejecucin termine con xito aadiendo al inicio del guin en dicho directorio la lnea:

```
exit 0
```

As ya podremos retirar el paquete e instalar una versin con dicho fallo corregido.

Actualizacin de los tipos MIME

Ya nuestro sistema sabe que puede abrir ROM de Super Nintendo con ZSNES. Tenemos el acceso directo, y el icono. Slo falta decirle al sistema cmo identificar una ROM de Super Nintendo.

En general, hay dos formas de identificar el tipo de un fichero: a travs de una secuencia binaria especfica en la cabecera (como en el caso de las imgenes en formato BMP, que incluyen siempre los caracteres «BM» al inicio del fichero), conocida como *magic cookie*, o a travs de la extensin.

Aunque usar el contenido del fichero es mucho ms robusto, no conozco realmente si siguen algn formato especfico (probablemente no), as que nos limitaremos a la extensin, algo mucho ms sencillo. En particular, nos centraremos en las dos ms populares: `.sfc` y `.smc`.

Aadimos a nuestro paquete el fichero XML `debian/zsnes.sharedmimeinfo` siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<mime-info
  xmlns="http://www.freedesktop.org/standards/shared-mime-info">
```

```

<mime-type type="application/x-snes-rom">
  <comment>SNES ROM</comment>
  <comment xml:lang="es">ROM de SNES</comment>
  <glob pattern="*.sfc"/>
  <glob pattern="*.smc"/>
</mime-type>
</mime-info>

```

Tras la declaracin XML, incluimos el elemento `mime-info`, con la declaracin del espacio de nombres para documentos de tipos MIME de FreeDesktop.org. Ahora tendremos una serie de tipos MIME con `mime-type`, dando una lnea descriptiva en posiblemente varios idiomas (de nuevo usando cdigos de ISO 639), y posteriormente especificando patrones con los que identificar los ficheros que pertenecen a dicho tipo: `glob` usa un simple patrón de ficheros del shell (no se trata de una expresin regular, como podemos ver), y `magic` nos permitira identificar por contenido.

Ahora deberamos asegurarnos de que la lnea que llama a `dh_installmime` se halle descomentada, si usamos slo `debhelper`, pero con CDBS no hay que hacer nada ms. La llamada a `update-mime-database` ser aadida automticamente a los guiones de postinstalacin y postdesinstalacin por el guin anterior, y el fichero ser instalado en su lugar correcto, `/usr/share/mime/packages`. Podemos comprobar, tras instalar el paquete, que la asociacin se ha realizado bien inspeccionando `/usr/share/mime/globs`, que debera contener las lneas:

```

application/x-snes-rom:*.sfc
application/x-snes-rom:*.smc

```

A.4.5. Generacin automtica de paquetes

Mdulos Perl

El guin `dh-make-perl` nos permite crear rpidamente versiones preliminares de paquetes Debian para mdulos Perl del CPAN (Comprehensive Perl Archive Network) [?]. De esta forma, podemos integrar todos los mdulos necesarios para nuestra aplicacin Perl en paquetes, y que as el usuario pueda instalarla justo como cualquier otra aplicacin.

As, si queremos empaquetar el mdulo `XML::Writer` del CPAN, escribiremos:

```
dh-make-perl --cpan XML::Writer
```

A. Gua de desarrollo de paquetes Debian

Es posible que tengamos que hacer algunos retoques a `debian/control` o a `debian/rules` si el sistema de compilacin del paquete no es compatible con el habitual (Make-Maker), o si tiene dependencias que no puedan detectarse automticamente. Por lo general, no habr mucho que hacer: el CPAN impone ciertas cosas que hacen que la tarea sea bastante ms sencilla que con otros programas.

Una nota: el servicio CPAN tiene enlaces a otro servicio muy til que indica exactamente las dependencias de un paquete Perl cualquiera de forma recursiva, y nos indica cules mdulos se hallan preconfigurados (y no requieren de un paquete por lo tanto) y cules no.

Tambin es til instalar el paquete *apt-file*, que es capaz de encontrar el paquete que contiene un determinado fichero, de tal forma que pueda ser aadido automticamente a las dependencias del paquete generado. Es importante actualizar el cach cada cierto tiempo de esta forma:

```
sudo apt-file update
```

Adaptacin de guiones de instalacin

Un caso muy comn es cuando nosotros mismos tenemos que compilar alguna aplicacin, porque no dispongamos de un paquete apropiado. ¿Deberamos directamente instalarla, y perder as las ventajas de una desinstalacin limpia y segura, y la posibilidad de posteriormente actualizar a una versin posterior si finalmente aparece un paquete?

No hace realmente falta: *checkinstall* [?], cuyo paquete del mismo nombre tendremos que instalar, se ocupa de invocar **make install**, seguir todo el rastro de la instalacin, y crear un rudimentario paquete Debian (tambin hay soporte para paquetes Slackware y RPM) a partir de l. Realiza la instalacin automticamente, y nos deja un paquete en el mismo directorio, que podemos pasar a cualquiera para que tambin lo instale. Por supuesto, el paquete no podremos subirlo a ningn repositorio serio: no tiene ninguna informacin de dependencias, por ejemplo, ni se halla firmado.

Si por ejemplo estamos compilando un programa basado en las *autotools* (unin de *autoconf*, *automake* y a veces *libtool*), slo habra que cambiar el ltimo paso:

```
./configure --prefix=/usr
make
sudo checkinstall
```

A.4.6. Otros formatos de paquete

Comparativa con otros formatos

Los paquetes Debian son slo un tipo ms de paquete que podemos encontrar. Otras distribuciones han desarrollado sus propios sistemas de empaquetado, con mayor o menor funcionalidad, y con mayor o menor xito. En esta gua, mencionaremos los otros tres formatos ms populares: RPM, Portage y Slackware.

RPM El formato RPM (RedHat Package Manager) es el empleado actualmente en las distribuciones SUSE y Fedora, entre otras, y es originario de la ahora desaparecida RedHat Linux. Tiene funcionalidad en el mismo nivel de los paquetes Debian, con guiones de pre/postinstalacin y desinstalacin, y metadatos, como por ejemplo las dependencias. Sin embargo, esta informacin de dependencias es menos rica que la de los paquetes Debian: slo existe un tipo de dependencia, mientras que en los paquetes Debian tenemos recomendaciones (paquetes que casi siempre necesitaremos junto con el actual) y sugerencias (paquetes que mejoraran la funcionalidad del actual). Adems, en los paquetes Debian podemos especificar alternativas en las dependencias, y usar paquetes virtuales, que indiquen la disponibilidad de una cierta funcionalidad, como «navegador web» o «entorno de ejecucin Java», ms que un software determinado.

Por otro lado, a diferencia de los paquetes Debian, incorpora guiones de verificacin de la correcta instalacin de un paquete, y disparadores al cambiar el estado de algn otro paquete. Tambin incluye la capacidad de definir dependencias sobre ficheros, aunque a muchos no les parece realmente til.

El principal problema de este formato era la falta de resolucin automtica de dependencias en la herramienta de gestin de paquetes **rpm** estndar: sta simplemente informaba de las dependencias directas que no haban sido cumplidas, obligndonos a reintentar el proceso bajando un paquete tras otro hasta finalmente conseguir instalar el paquete deseado. Sin embargo, ya existen herramientas que aaden esta resolucin de forma transparente, como **yum** de Fedora, **YaST** de SUSE, o **urpmi** de Mandriva.

Gracias a esas herramientas, hoy en da el problema es distinto, y se basa ms en la forma y contenido de los paquetes en s: no existe una poltica estricta y estndar de empaquetado como la que tienen los paquetes Debian [?], ni herramientas de validacin. Por ello, un RPM no suele funcionar bien fuera de la versin especfica de la distribucin en que se desarroll. Adems, no hay tanto software empaquetado en formato RPM como lo hay en formato Debian, ni de forma tan accesible.

Portage El sistema Portage de Gentoo es una versin para Linux del sistema *ports* de FreeBSD: realmente, un paquete de Gentoo se halla formado por un fichero `ebuild`,

A. Gua de desarrollo de paquetes Debian

que contiene las instrucciones de cmo obtener, compilar, instalar y configurar el software.

Normalmente, tras compilar el paquete, obtenemos tambin un paquete parecido a los que estamos acostumbrados a ver. As, si tenemos varias mquinas con la misma versin de Gentoo, no tendremos que compilar en cada mquina. De la misma forma, existen tambin paquetes binarios precompilados si los necesitamos. De esa forma evitaremos tener que recompilar muchas aplicaciones grandes, como OpenOffice o KDE, por ejemplo.

Los ficheros `ebuild` incluyen informacin de dependencias, y la herramienta usada para la instalacin, **emerge**, las sigue de forma transitiva. Tambin disponemos de paquetes virtuales, con la misma semntica que en Debian. El nico problema se halla en las dependencias inversas: a la hora de retirar un paquete, Portage no comprueba si estamos rompiendo algn otro paquete. Existen herramientas como **revdep-rebuild** que hacen esta comprobacin por nosotros, pero no se hallan integradas con el proceso de desinstalacin.

Otros problemas incluyen el tener que realizar mantenimiento constante sobre los ficheros de configuracin de los paquetes que actualicemos, o el ciclo de desarrollo ms corto de los paquetes Gentoo frente a los de Debian. Aunque normalmente tendremos software mucho ms reciente, no estar tan probado como lo podra estar un paquete Debian, y podemos llevarnos ms de un disgusto.

Por lo dems, se trata de un sistema muy completo, con la capacidad de actualizar todo nuestro sistema comprobando nuestras dependencias de forma transitiva automticamente, entre otras cosas. Obtenemos las actualizaciones sincronizando de forma peridica nuestro rbol de ficheros `ebuild` con un directorio remoto mediante **rsync**.

Slackware El formato `tgz`, de Slackware: es nicamente un `tar.gz` que se descomprime directamente a `/`, para despus ejecutar un guin de postinstalacin. Se corresponde con la filosofa de dicha distribucin: el usuario es el que sabe qu hay que hacer.

No incorpora metadatos de ningn tipo, ni control de dependencias. Es posiblemente el formato ms sencillo de empaquetado que podremos imaginarnos.

Conversin desde paquetes Debian

Puede que queramos que usuarios de SUSE, Slackware o Fedora, por ejemplo, empleen nuestro programa. Quizs no tengamos tiempo como para mantener un paquete para todos y cada uno de los otros muchos formatos disponibles.

La herramienta *alien* (como siempre, antes deberemos instalar su paquete) nos deja convertir entre paquetes Debian, Slackware, Solaris, RPM, LSB y Stampede. Es bastante experimental, y los paquetes generados no tendrán la misma calidad que uno hecho a mano, pero puede ser til en muchos casos.

As, para convertir cualquier paquete a formato Debian, usaremos:

```
alien (ruta al paquete)
```

Por otro lado, si queremos crear un paquete RPM a partir de un paquete Debian, podremos usar, como en el caso de ZSNES:

```
alien --to-rpm \  
  \~{}/packages/build-area/zsnes\_1.510.SVN5113-0ubuntu1\_i386.deb
```

Slo hemos de tener cuidado de que la conversin se haya hecho lo bastante bien: por ejemplo, los guiones de preinstalacin, postinstalacin y dems no se convierten bien al formato RPM. Posiblemente tendremos que editar los ficheros generados para asegurarnos de que funcionen bien.

B. GNU Free Documentation License

Version 1.2, November 2002

Copyright ©2000,2001,2002 Free Software Foundation, Inc.

51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “**Document**”, below, refers

B. GNU Free Documentation License

to any such manual or work. Any member of the public is a licensee, and is addressed as “**you**”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “**Modified Version**” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “**Secondary Section**” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “**Invariant Sections**” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “**Cover Texts**” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “**Transparent**” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “**Opaque**”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools

are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The **“Title Page”** means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, **“Title Page”** means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section **“Entitled XYZ”** means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as **“Acknowledgements”**, **“Dedications”**, **“Endorsements”**, or **“History”**.) To **“Preserve the Title”** of such a section when you modify the Document means that it remains a section **“Entitled XYZ”** according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of

B. GNU Free Documentation License

the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.

- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

B. GNU Free Documentation License

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you

follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

B. GNU Free Documentation License

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.