OPENJADE DEVELOPMENT
Implementación y formateo del modelo de páginas

Cristian Antolin Tornador

OPENJADE DEVELOPMENT Implementación y formateo del modelo de páginas
por Cristian Antolin Tornador

Publicado 15/06/2007

Memoria del proyecto final de carrera correspondiente a los estudios de Ingeniería Superior en Informática presentado por Cristian Tornador Antolin y dirigido por Joan Borrell Viader como tutor del proyecto y por Javier Farreres como jefe del proyecto y profesor externo.

El bajo firmante, Joan Borrell Viader, profesor del Departamento de Informática de la Universidad Autonoma de Bellaterra

CERTIFICA:

Que la presente memoria ha sido realizada por Cristian Tornador Antolin bajo su dirección como tutor del proyecto

Firmado por Joan Borrell Viader.

Dedicatoria

Importante: Se lo dedico a mi familia que siempre me ha apoyado, y a Javier Farreres por la paciencia que ha demostrado tener todos estos años.

Tabla de contenidos

Lista de figuras

Capítulo 1. INTRODUCCION

A la hora de hacer una buena documentación o un buen artículo se plantean normalmente diversas cuestiones, cómo escribirlo, que estructura seguir y qué estilo se debe dar para que se ajuste a ese tema en concreto.

Finalmente, cuando se tiene buena parte de la documentación suelen salir diversas dudas referente a la elegancia del documento, si se está realizando de una forma apropiada al tema, o simplemente si se debería seguir con el estilo actual o cambiar a otro de mejor. Muchas veces se obta por no modificar nada debido al tiempo que supondría cambiar y retocar todo el documento.

Demasiadas preguntas, o demasiadas complicaciones, que al final lo que hacen es que se esté más pendiente de dar un correcto estilo al documento que no en introducir la información propia del documento. Pese a todo esto, hay que recordar que el estilo es muy importante, no es algo que se pueda dejar al margen. Además no es lo mismo hacer una documentación científica que una entrevista deportiva, y se ha de tener en cuenta para quien va dirigido. Normalmente una buena presentación tiene un peso muy importante a la hora de informar. Por lo tanto, la información del documento en si va ligado al estilo que se le quiere dar.

Este problema se agrava cuando se lleva escrita gran parte de la documentación y se observa que se debe de modificar el estilo. Normalmente esto supondría un coste bastante elevado.

A parte, cuando se quiere comenzar un nuevo artículo, quizás se quiera el mismo diseño y estructuración que alguno ya creado anteriormente, y en lugar de poder ahorrar tiempo, este acaba incrementándose, no consiguiendo en muchas ocasiones el objetivo planteado.

Con todo esto, hizo plantearse la manera de crear documentación. Llevo a crear hojas de estilo para la documentación, es decir, crear una serie de plantillas para dar estilo a diferentes tipos de documentos, para poder separar, el estilo del documento, del documento en si, haciendo al usuario que se centrara en lo que iba a escribir y como lo tenía que escribir.

De esta forma, al terminar la documentación, cada uno ya se encargaría de usar la plantilla de estilo adecuada, o reutilizar alguna ya anteriormente probada guardada con anterioridad.

Esta forma de hacer la documentación se le llama documentación estructurada y es algo diferente a la que se utiliza normalmente por la mayoría de usuarios. Esta forma de crear documentación requiere una reflexión previa, y no es sencilla de construir, ya que antes de ponerse a escribir el documento se tiene que saber muy bien como va a estar estructurado y cuales son sus partes. Si no se esta acostumbrado seguramente el proceso de creación ser verá incrementando en un principio, pero a la larga la documentación queda mucho mejor formada, y uno se puede concentrar mucho más en lo que se quiere

escribir. Además, permite reutilización del diseño y si se sabe utilizar adecuadamente en documentaciones largas, se rebaja mucho el tiempo de creación, o modificación del artículo o documento.

Cualquier webmaster con experiencia, seguramente poseé su propio repositorio de hojas de estilos y sus propias plantillas. De esta forma se ahorra mucho tiempo y se asegura que su diseño tiene cierta consonancia durante todo el proyecto. En ningún caso se pondría a crear páginas web sin tener claro como se va a estructurar y que diseño se quiere dar.

Si por lo contrario se va a crear una de sola, y se tiene mucha prisa, y da igual como quede la documentación, quizá este proyecto le resulte de poco interés, ya que esta herramienta no va dirigida a este tipo de usuarios ni a este tipo de escritos, ya que seguramente perdería mucho mas tiempo que de la forma natural de escritura.

Esta idea fue madurandose y finalmente en 1960 se creo el estándar SGML, en donde se definía además el cómo se tenía que estructurar la documentación y cúal era la forma correcta en cuanto a la creación de documentación se refería.

De allí finalmente, se definieron unas reglas para la documentación de diferentes tipos de documentos. Una vez definido las reglas para documentar, habría que fijar las reglas para dar estilo a este tipo de documentos, así pues más tarde salió en 1995 el estándar DSSSL.

Se tenía por un lado la forma de crear documentos (.sgml) y el como crear las hojas de estilo para estos documentos (.dsl), pero no se tenía ninguna aplicación para procesar estos documentos y darles estilo mediante estas hojas de estilo.

Así pues James Clark en 1996 creó la herramienta Jade, pero al ver que los pasos a seguir por el estándar en la creación de esta herramienta era muy compleja y el árbol de objetos era demasiado complicado, lo dejó a medias y se puso con XML, creando un lenguaje más simple y sencillo que aportara de cierta manera la esencia fijada en un principio.

De esta manera, la herramienta Jade quedó a medias y bastante limitada en varios aspectos, uno de estos aspectos es la creación de modelos de páginas.

En la actualidad, la herramienta que nos dejo James Clark, unicamente se podía escribir textos simples a partir de documentos SGML. Pero no se podían crear diferentes áreas, dentro de una página, y dentro de estas áreas crear más áreas, etc. Sólo, se podía escribir de arriba abajo en un área principal, que era el área total de la página en si.

En este proyecto lo que se intenta es conseguir solucionar esta deficiencia, para que los textos escritos en el estándar tengan más juego y dinamismo, y así abrir al estándar algo el mercado para que pueda convivir con xml.

Este lenguaje de marcas fue bastante bien acogido, pero está algo diversificado, poco centralizado y recogido, existiendo más parches que reglas para conseguir hacer cosas que en un principio no se podían hacer debido a su simplicidad.

Al dotar, a la herramienta, modelos de paginas al cuerpo de la documentación se espera una buena acogida en los diversos foros que existen y sea una herramienta más utilizada.

OpenJade, poseé diferentes maneras de formatear y crear el backend final. En uno de los backends tiene una carencia. Otro problema importante.

Si se quiere crear un documento pdf, dvi o ps a partir de una hoja de estilo y un documento SGML, primero se tiene que genera el formateo en latex, y a continuación mediante otros comandos se generan estos backends. Pero este backend tex no está escrito estrictamente en Latex/TeX, sino llamadas a macros definidas en la herramienta Jadetex. Esto supone que la creación de este tipo de documentos no es automática, y se necesiten dos procesos para generar el documento final. La idea es poder integrar todo en el mismo backend, de manera que el resultado sea definitivo y único.

En los comandos actuales que presentan todas las distribuidoras de Linux, ya se genera de forma automática la secuencia de comandos necesarios para crear los backends resultantes en pdf o ps. Pero lo que en verdad hacen, es hacer la llamada a jade openjade para crear primero el backend .tex y a continuación llamar al otro comando jadetex para crear el backend resultante tex definitivo, pdf o ps.

El problema de este fichero inicial de macros latex es que es ilegible y por lo tanto no modificable, o muy difícil de modificar. La prueba está en que si se desea crear el fichero de salida .tex y se intenta generar un backend final mediante el comando latex, este no consigue generar nada porque no entiende estas macros. La única manera es generarlo con este comando "adosado" a openjade, llamado Jadetex.

Por lo tanto, también es deseable añadir esta funcionalidad a la herramienta, ya que en un principio el proyecto se encara al análisis, creación y formateo en latex de los modelos de página.

Así pues, se marcaron para este proyecto los siguientes objetivos:

- Análisis de herramientas
  - Estudio de la herramienta JadeTex
  - Estudio de la herramienta OpenJade
  - Estudio de su conjunto

- Formación en Latex
- Incorporacion de JadeTex en OpenJade.
- Estudio de la posibilidad de la incorporación de las validaciones en la herramienta.
- Desarrollo de la incorporacion de los modelos de paginas en la herramienta y su formateo en latex.

El resto de memoria se estructura en los siguientes cápitulos:

- Capitulo 2.CONCEPTOS PREVIOS Y ANTECEDENTES. En este cápitulo se dan a conocer los conceptos más importantes necesarios para seguir el proyecto. Se aclaran posibles cuestiones que se podrían dar con la introducción de la memoria. Así pues un capítulo imprescindible para coger una base del problema que se plantea.
- Capitulo 3.PRESENTACIÓN DEL PROBLEMA. Aquí se presenta el motor de estilo de openjade, las clases necesarias para hacer el modelo de pagina y su posterior formateo.
- Capitulo 4. IMPLEMENTACIÓN Y FORMATEO. Se explica de forma breve los pasos a seguir en el caso de implementar otras funcionalidades similiares basandose en el modelo que se cita, y también se hace una referencia en donde y como se inserta el formateo en .tex y la extracción de la herramienta jadetex a openjade.
- Capitulo 5. RESULTADOS, JUEGO DE PRUEBAS. Aquí se pone de manifiesto con ejemplos si el modelo de páginas creado con anterioridad y su implementación en el backend .tex fue correcto o hay errores. Para ello se muestran imagenes de diferentes posibilidades y ejemplos que se pueden dar y como queda formateado en un documento pdf o ps.
- Capitulo 6. CONCLUSIÓN. En este capítulo final, se indica que objetivos han sido asolidos y cuales no, los problemas que se han presentado y futuras implementaciones y resultados que se pueden conseguir a partir de este proyecto.

Capítulo 2. CONCEPTOS PREVIOS Y ANTECEDENTES

Aquí se explicará los conceptos más importantes que hay que tener en cuenta para poder seguir el desarrollo del proyecto y poder tener una base referente al tema propuesto. También se explicarán los antecedentes de la herramienta para contextualizar el problema y una explicación de como se debe utilizar la herramienta OpenJade.

2.1. Que es OpenJade

Openjade(OJ) es una implementación del leguaje de estilo DSSSL, el cual, es una evolución de la antigua herramienta de estilo jade(JADE), desarrollada por James Clark en 1996. Esta fue la primera herramienta que implementaba parte de ISO DSSSL standard[DSSSL]. OpenJade es una de las pocas implementaciones del estandar y es utilizada en el proyecto Docbook(DBK). En definitiva, OpenJade es un procesador de DSSSL, en desarrollo, el cual es capaz de dar estilo a documentos SGML.

2.2. OpenSource

Cuando una persona escribe un programa, un script o sencillamente unas líneas de código se convierten en su autor. Los autores tienen derechos sobre lo que han creado y las leyes protegen sus obras de manera automática limitando el uso que los demás pueden hacer de sus obras, sin embargo un autor puede desear que su software sea "libre".

Muchas veces se escucha la expresión "código abierto" o "software libre", pues bien la expresión inglesa para el software libre es "free software" que en ningún momento significa que el software sea gratuito sino libre. Libre implica libertad de uso, de modificación y manipulación... como se hace evidente el límite de la libertad es algo difícil de establecer siguiendo el concepto que implica que algo sea libre.

Alguien podría querer vender una obra que fue realizada por otra persona diciendo incluso que es suyo. Por ello el software libre se rige por diferentes "licencias" que se adaptan a las distintas necesidades que puedan surgir. (Si desea encontrar una definición más extensa y precisa de lo que significa la palabra libertad en el marco que nos ocupa, una importante referencia es la que se encuentra disponible aquí: http://www.gnu.org/philosophy/free-sw.es.html)

Pero qué es una licencia?, se puede decir que una licencia es la forma que un autor tiene de expresar como quiere que se utilice su obra por los demás, siendo una definición concreta que se le aplica a dicho software si es declarado como software libre.

Un autor puede acogerse por una de las licencias existentes o bien crear la suya propia, aunque escoger esta última opción puede resultar algo complejo y posiblemente no sea necesario pues existen diversos tipos de licencias que cubren la mayoría de las necesidades existentes. Tal y como se acaba de decir se dispone de varias licencias muy bien definidas, algunas de las cuales se explican a continuación, de forma breve.

LICENCIA GLP (General Public License)

La GLP es la licencia que más se utiliza para el open source (código abierto). La General Public License exige que todo el código fuente está disponible, se pueden vender copias del software y se puede modificar creando los "proyectos derivados" que bajo ningún concepto podrán distribuirse sino bajo la misma licencia GLP. GLP nunca podrá incluirse en ningún software propietario o comercial.

LICENCIA LGPL (Lesser General Public License)

Es muy similar a la anterior, de hecho se puede decir que la única diferencia es que no obliga a los proyectos derivados a ser distribuidos bajo la misma licencia, así una persona podría distribuir, por ejemplo, sus librerías en software comercial y/o cerrado sin ningún tipo de problema.

LICENCIA AL ESTILO BSD

Permite el uso de la fuente y su redistribución, con o sin modificaciones siempre que se mantenga su copyright, la licencia de distribución quedando prohibido que el nombre del autor se utilice para promover la venta del producto salvo que se consiga un permiso por escrito.

Los binarios y las fuentes debe contener la licencia original y obliga a que toda publicidad sobre el software y/o derivados haga mención a los autores que en ella deben figurar.

Aquí se puede ver un extracto de la licencia BSD:

www.debian.org/misc/bsd.license

Copyright (c) 2000

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

this software is provided by the author ~as is~ and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. in no event shall the author be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory ofliability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

LICENCIA ARTÍSTICA

Permite hacer copias del software original sin modificaciones y distribuirlas sin restricciones incluyendo siempre la licencia en los duplicados. Las copias del software que se rige mediante una licencia artística siguen siendo originales siempre que las modificaciones sean para parches para fijar algunos agujeros y similares.

Las modificaciones públicas que se hagan sobre el código (productos derivados) deben ser disponibles de manera gratuita o bien debe ser únicamente para uso interno empresarial y/o personal o se haga algún acuerdo con su autor. Se permite la distribución de software compilado original o parcheado siempre que incluya documentación, pudiendo prescindir en este caso de incluir las fuentes del software.

Podemos obtener un extracto de licencia artística de la siguiente URL:

http://www.perl.com/language/misc/artistic.html

OTRAS LICENCIAS

Existen otras licencias de interés como la FDL (free Distribution License), cuyo preámbulo se expone a continuación a modo de explicación:

[...] 1. PREÁMBULO El propósito de esta licencia es hacer que un manual, libro de texto, u otro documento escrito sea libre en el sentido de libertad: para asegurar a todo el mundo la libertad efectiva de copiarlo y distribuirlo, con o sin modificaciones, bien de manera comercial o no comercial.

En segundo término, esta licencia preserva para el autor o para quien publica una manera de obtener reconocimiento por su trabajo, al tiempo que no es considerado responsable de las modificaciones realizadas por terceros.

Esta licencia es una especie de "copyleft" que significa que los trabajos derivados del documento deben a su vez ser libres en el mismo sentido.

Esta licencia complementa la Licencia Pública General GPL.[...]

Para encontrar su descripción completa en español y detalles acerca de la FDL, sería interesante visitar el siguiente link:

http://www.geocities.com/larteaga/gnu/gfdl.html

Se puede encontrar más información acerca de las licencias y algunos términos comunes no mencionados aquí como "shareware","propietario" o "semi-libre" en la siguiente

URL:

http://www.gnu.org/philosophy/categories.es.html

Como es de imaginar las posibilidades son infinitas y desde luego no es el objetivo de este trabajo hacer una exhaustiva definición del software libre ni sobre las licencias existentes en la actualidad, aunque se han comentado las más importantes y utilizadas por el momento.

2.3. Doxygen

En primer lugar es obligado explicar qué es Doxygen, ya que a parte de la implementación de las diferentes clases, en el proyecto también se va documentando en el código fuente mediante Doxygen. Se trata de un sistema de documentación válido para varios lenguajes tales como C++, Java, C e IDL incluso algunas extensiones de PHP y C# su autor es ~Dimitri van Heesch~.

Doxygen puede generar tanto documentación en HTML como en PDF, dicha documentación se extrae directamente de los archivos fuente resultando siempre una consistente ayuda. Por supuesto doxygen puede ser utilizado para escribir cualquier otra documentación normal.

Doxygen es una potente herramienta open source que cualquiera puede descargarse de la siguiente URL, donde además se pueden encontrar extensos manuales en inglés, FAQ~s, y todo tipo de referencias sobre la herramienta y su autor.

http://www.doxygen.org/download.html

Doxygen está disponible tanto para plataformas Linux como Windows, así pues se dispone de varias opciones tras la descarga de Doxygen.

Ahora vendría bien centrarse en la instalación mediante las fuentes bajo Linux, por ser el caso que nos ocupa, el proceso es realmente sencillo. Requisitos previos para una plena funcionalidad de Doxygen:

- Las herramientas GNU Flex y Bison.
- Make y Perl http://www.perl.com
- Un intérprete de Ghostscript http://www.ghostscript.com
- Qt v2.0 o superior http://www.trolltech.com/products/qt.html
- Una distribución de LaTex, sea teTEx 1.0 http://www.tug.org
- Graphic Visualization toolkit v1.5 o superior lo puedes obtener del siguiente enlace http://www.research.att.com/sw/tools/graphviz

Cada proyecto debería tener su propio fichero de configuración, para simplificar esto Doxygen puede crear un fichero de configuración de tipo plantilla (template) mediante la siguiente orden:

doxygen -g < fichero_configuración>

Se Puede indicar con un signo negativo '-' que Doxygen lea el archivo de configuración de STDIN. Pero hay que insistir en que no a todo el mundo le apetece editar a mano este fichero (parecido a un makefile) por es recomendable bajarse*'Doxygenwizard'* del sitio www.doxygen.org

El fichero de configuración se basa en una serie de ~tags~ que ayudan a preparar Doxygen para el trabajo que se quiere documentar de forma totalmente personalizada. Hay una larga lista de ~tags~ que se puede utilizar pero la mayoría se dejaran probablemente con su valor por defecto. Para este caso el fichero de configuración se encuentra en la raíz del proyecto y el resultado se guardará en la subcarpeta 'dox'.

2.4. Apostando por Documentación Estructurada

La documentación estructurada es aquella que da una mayor importancia a la estructura del documento que no a la forma de presentación. Esto supone una serie de ventajas:

- El autor se puede centrar en la labor de escritura y creación del documento olvidándose de cosas banales e irrelevantes.
- El documento es independiente a la plataforma, facilitando almacenamiento, consulta, manejo y proceso de creación
- Facilidad para su edición y manipulación para cualquier editor
- Mejora de una mejor calidad final del documento con ayuda de herramientas apropiadas

2.5. SGML

2.5.1. Que es SGML

SGML (Standard Generalised Markup Language) internacional estándar: ISO 8879/1986; es el lenguaje estándar que indica la forma en que se debería escribir o estructurar los documentos o lenguajes basándose en la filosofía de los lenguajes de las marcas:

...no limita los documentos a una única aplicación, a un estilo de formateado o a un sistema de procesamiento. Se basa en dos postulados:

- *El etiquetado debería describir la estructura del documento y otros atributos más que especificar el procesamiento que se va a llevar a cabo en dicho documento ya que el etiquetado descriptivo necesita efectuarse tan sólo una vez, siendo esta suficiente para todos los procesamientos futuros.*
- *El etiquetado debería ser riguroso de manera que las técnicas disponibles para el procesamiento de objetos rigurosamente definidos, como por ejemplo los programas y bases de datos, puedan utilizarse también para el procesamiento de documentos.*

Al ser un lenguaje para estructurar documentos, implementa las ventajas propias de la documentación estructurada (independencia de la representación y de la plataforma y abstracción pura de la información

Este lenguaje fue inventado por Charles Goldfard, el cual fue madurado desde 1960 y fue creado por el programa CALS del DoD en 1986

Hay muchos lenguajes "markup" que usan SGML. Un ejemplo de esto sería HTML, cuando lo utilizas no es que está escribiendo SGML sin que esté utilizando partes de SGML, Se podría decir que HTML es un dialecto de SGML También está: LinuxDoc y DocBook.

Existen editores para que la edición de textos, en sgml, sea más "fácil". Como pueden ser Emacs (mediante el modo PSGML que muestras las markup disponibles en cada momento, hace autocomplementacion...), Judit (editor hecho en java que puede mostrar el árbol de estructura del documento) y Vim (se puede configurar para no escribir las markup. Estos se basan en el proyecto DocBook

2.5.2. Composición de un documento SGML

Juego de caracteres: El juego de caracteres puede declararse explícitamente o, en su omisión, usar el juego predeterminado por el sistema.

Encabezado: Es donde se describe que tipo de documento SGML, puede hacer referencia a un archivo físico o a un identificador que después se resuelve en un catálogo.

Declaración de documento: La DTD (declaración de tipo de documento) especifica la sintaxis y la jerarquía y relación entre las marcas en las diferentes formas en las que puede construirse un tipo o familia concreto de documentos.

Elementos: Las secciones que componen al documento desde el punto de vista estructural. La jerarquía y relaciones de los elementos están definidas en la DTD. Familiarmente también se les llama etiquetas o marcas (book, title...).

Atributos: Los parámetro de cada elemento. Pueden ser opcionales (y tener o no valores predeterminados) u obligatorios.

Entidades: Caracteres que no pueden ser representados textualmente ya que tienen un significado directo en sgml, como por ejemplo " < " se tendrá que representar mediante el símbolo & y la palabra lt. Son de uso múltiple y flexible:

- Representación de signos no recogidos por el juego de caracteres
- Abreviaturas o macros en el código fuente
- Referencia a ficheros externos (como los "#includes" del lenguaje C)
- Variables, cuyo valor se especifica en el momento del procesamiento.

> **Nota:** Las entidades deben estar declaradas en el documento o en la DTD.

Contenido: El texto introducido entre marcas

Instancia: El documento SGML compuesto usando una DTD, sus elementos y los atributos de estos, las entidades internas/externas y el contenido entre marcas.

Comentarios: No son procesados, son aclaraciones sobre el documento.

2.5.3. XML vs SGML

XML es una simplificación de SGML, con una orientación a la creación de páginas Web, por lo tanto no incluye características de SGML. Es decir, XML está escrito en SGML y funcionalmente es equivalente a SGML. Además aunque XML es más sencillo no es tan potente:

- No permite la minimización de elementos
- No permite restricciones en la anidación de los elementos
- No distingue entre las mayúsculas y minúsculas en el nombre de los elementos
- Uso obligatorio de comillas en los atributos

XML posee la hoja de estilo XSLT, una versión simplificada de DSSSL que se genera a partir de la sintaxis XML. Esta herramienta está completamente funcional por lo que hace que XML está ampliamente extendida internacionalmente.

2.6. DSSSL

DSSSL (Document Style Semantics Specificacion Language) estándar internacional ISO/IEC 10179:1995, es una hoja de estilo (o documento para dar estilo a otros documentos), es decir, define cual va a ser la presentación que va a tener el fichero de salida. A parte del procesamiento para la representación de documentos, permite transformar una DTD a otra y hace de base para la gestión de los documentos.

Por todo ello supuso un gran avance en la manipulación de la documentación. Proporciona:

- Entorno uniforme de manejo y modificación de documentos
- Entorno de creación de filtros mediante hojas de estilo
- Entorno de programación uniforme, basado en Scheme "parecido al LISP" (lenguaje imperativo maduro, robusto, sencillo y perfectamente especificado -R4RS-)

Mediante el DSSSL (.dsl) se puede dar estilo a los documentos SGML mediante las DTDs correspondientes. La hoja de estilo equivalente a DSSSL para XML es, como ya dijimos, el XSLT (file.xsl).

2.7. Historia

James Clark, nacido el 23 de Febrero de 1964 en Londres, comenzó en 1991 a trabajar en el area SGML/XML

- Primero desarrollo un analizador escrito en C para SGML, SGMLS. ftp://ftp.jclark.com/pub/sgmls/
- A partir de este desarrollo SP, un analizador de SGML escrito en C++, basandose en este se desarrollo que ahora es el modulo OpenSP del proyecto OpenJade
- En 1996 creó el estandar DSSSL ISO/IEC 10179:1996, (Documment Style Semantics and Specification), esto le llevó desde 1991
- Finalmente, se crea tal como se conoce Jade, la version 1.2.1 en octubre de 1998, la cual está escrita en C++

Jade era una herramienta que tenía muchas limitaciones, pero James la dejó de lado para centrarse en el desarrollo de XML (sobre 1998).

Más tarde la organización SourceForge continuó su desarrrollo sacando la nueva versión Open 1.2.2 de OpenJade en junio de 1999 y en octubre de ese año se crea la versión estable 1.3; en esta versión se agrega muchas mejoras para aumentar el alcance de la puesta en práctica de DSSSL.

El módulo OpenSP estable 1.4 se crea en 2000 y posteriormente en el 2001-02 la versión 1.5.

A partir de aquí hubo un estancamiento debido a la falta de desarrolladores, hasta principios del 2002, que se empezaron a desarrollar diferentes versiones del 1.3. En Enero de 2002 se crea una nueva versión 1.3.1 de OJ, (tarball donde está el modulo jade y sp juntos), este lanzamiento fue sobre todo un lanzamiento del mantenimiento; a ayuda para las plataformas que existen actualmente(compiladores y sistemas operativos) y mejoras para el backend TeX . Pocos meses más tarde se se crea la 1.4-devel pero se desestima debido a la cantidad de problemas tanto de compilación como de ejecución, pasando a la versión actual 1.3.2, el cual te permite compilarlo con OpenSP 1.5 y el cual el OpenSP se distribuye en un paquete a parte. También este, mejora aspectos sobre el backend MIF que se omitieron en el 1.3.1

Para más información sobre las nuevas diferencias entre las diferentes versiones: http://belnet.dl.sourceforge.net/sourceforge/openjade/releasenotes.html o descargarse el fichero realesenote de la version 1.3.2 en www.sourceforge.net/projects/openjade

> **Nota:** En esta versión no estan implementadas ni las constraints ni el desarrollo de este proyecto, por lo que probablemente saldrá una nueva versión a posteriori.

2.8. Construcción y Utilización de OpenJade

Existe distribución tanto para entorno Windows como para Unix/Linux, estas se pueden encontrar en http://www.sourceforge.net/projects/openjade, ya que es un proyecto abierto de la organización sourceforge.net.

Se puede compilar las fuentes tanto Linux/Unix como en Windows/Windows NT. En Windows es soportado por Visual C++ y para Linux es necesario tener unos requisitos mínimos, aunque cualquier distribución de linux actual debería tener los packages necesarios para su construcción.

> **Nota:** Se podría compilar bajo el emulador Cygwin32 en windows una distribución linux, aunque "se desconozce" las limitaciones de este

http://members.tripod.com/~mhoenicka/openjadesp.html

2.9. Requerimientos necesarios para el desarrollo bajo plataforma Unix/Linux

Para poder compilar y construir OpenJade/SP bajo Linux se necesita tener instalados los siguientes paquetes como mínimo: gcc, cpp, libc, libstdc, sgml, perl, autotools, automake, m4, autoconf, gettex (como guía). Normalmente si se tiene el compilador de gcc, junto con los paquetes de sgml que vienen en toda distribución de Linux, ya se debería poder construir sin problemas la herramienta a partir de los fuentes, como con cualquier otra aplicación.

> **Nota:** Generalmente, si elige la opción de desarrollador en linux, ya vendrán los paquetes necesarios para poder desarrollar openjade o como mínimo usar jade o openjade.

2.10. Compilando en linux

Para poder compilar OpenJade sin problemas, primero se debería tener las librerías de SP. Si se escoge compilar la ultima versión de openjade es aconsejable compilar antes el proyecto OpenSP. La compilación bajo linux es muy simple ya que la versión Openjade/SP se compila mediante el autoconf/automake

Si se tiene el modulo SP y JADE por separado los pasos son los siguientes:

- Para SP: en la raíz del módulo: sh ./configure; make; make install
- De igual forma para OpenJade

Si se tiene las dos módulos incluidos en uno; como la versión 1.3.1; la forma de compilarla será la misma: en la raíz del proyecto openjade: sh ./configure; make; make install. El configure lo que hará es, mediante el autoconf, analizar su sistema para saber que librerías y que compiladores tiene instalados. Después mediante el "make" llamará al Makefiles el cual recursivamente compilará los fuentes creando las librerías .lo .o y .so. Finalmente el "make install" situará estas librerías en su sistema para que puedan ser llamadas. Generalmente en el subdirectorio /usr/local

2.11. Uso de OpenJade

La Forma de utilización para crear, a partir de un documento sgml (.sgml) y la hoja de estilo (.dsl), el backend indicado:

Usage: openjade -t [fot/mif/rtf/sgml/tex/html] -d file.dsl file.sgml

Nota: Esta linea de comandos solo fue probada bajo entorno Linux

Descripción de la linea de comando:

- openjade : comando
- -t : indica el tipo de backend de salida. Que pueden ser:
  - fot: genera en un fichero la representación del arbol XML (arbol de objetos de flujo)
  - mif: (Adobe interchange Format) documentos FrameMaker
  - rtf: fichero que puede ser abierto mediante word o abiword(linux)
  - sgml/xml: un simple transformador, transforma de sgml a sgml/xml. Se utiliza para unir los objetos que no contempla el estándar DSSSL generando documento SGML o XML
  - tex: genera un fichero TeX. Después con JadeTeX o PdfJadeTex se podrá formatear a PDF , PS , DIV
  - html: genera código en html, para los exploradores de internet
- -d: indica que es una hoja de estilo
- fichero.sgml: documentación estructurada en formato SGML

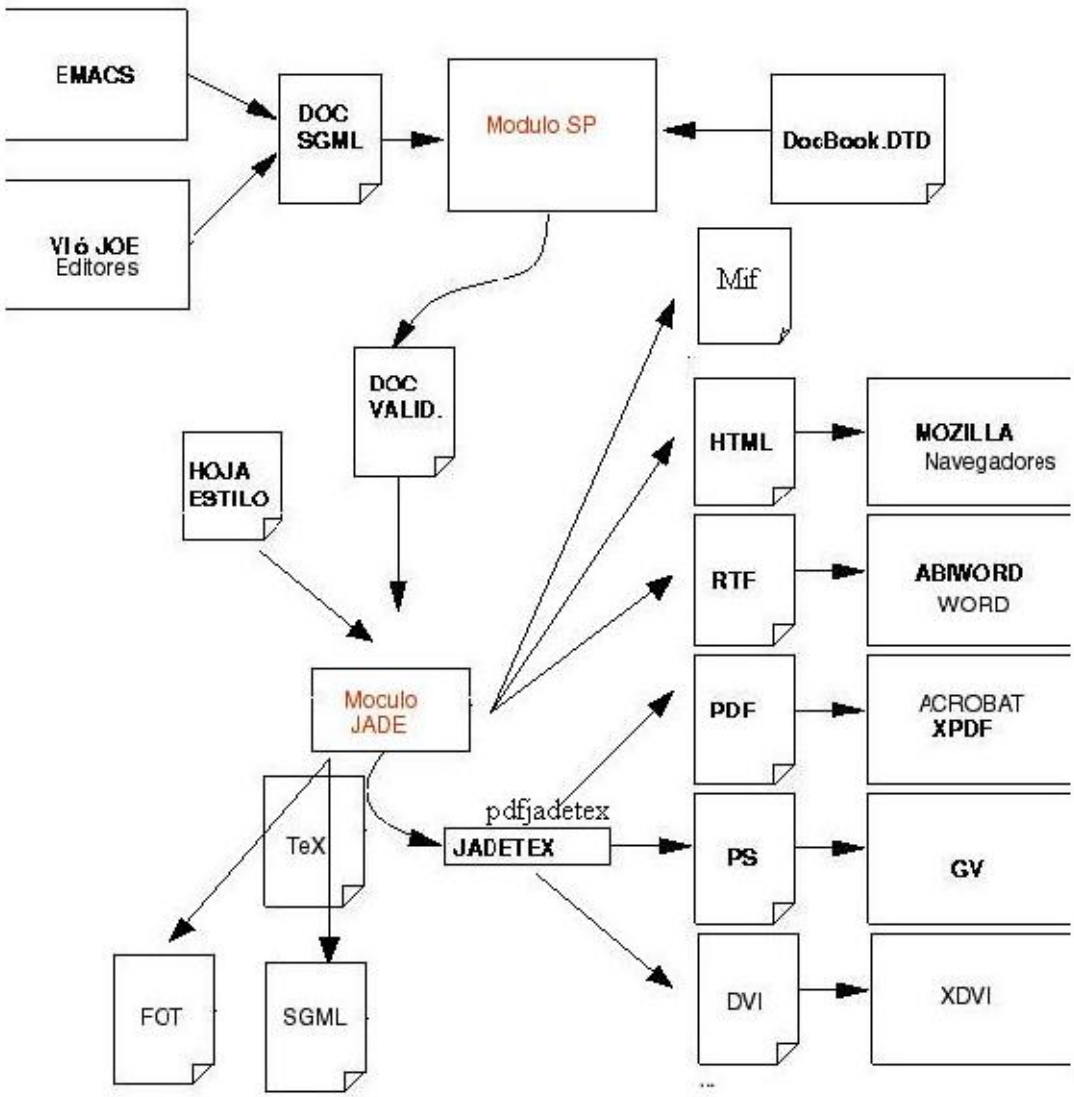
Otras opciones para la linea de comando:

- -G: entra en modo debuguer
- -out file_name: indica el nombre de fichero de salida, si no lo especificas te mantedrá el nombre del fichero de entrada (.sgml)
- -v name_variable: es equivalente a hacer (define name_variable #t)

2.12. Proceso de creación de estilo mediante OpenJade

En este esquema se observa la fabricación de diferentes tipos de documentos aplicando el estilo al documento sgml. Este esquema esta basado en entorno Linux pero la aplicación para Windows es muy similar)

Figura 2-1. Esquema de Openjade - creación de backends



Primero se crea un documento en SGML mediante un editor para ahorrar trabajo y "dolor de cabeza", como puede ser Emacs con el entorno psgml instalado; de esta forma se puede elegir, de forma sencilla, que elementos se pueden insertar en el punto donde nos encontramos del documento (.sgml). A estos elementos se les conoce bajo el nombre de

"markup" tags o marquillas, que son como unas marcas que indican que ese texto forma parte de un ámbito, y que a su vez, ese ámbito puede formar o no parte de otro. A esto se le llama documentación técnica o estructurada. Una vez creado el documento de texto, se compilará mediante el proyecto OpenJade para darle estilo. Para esto se necesitará otro documento (.dsl), un documento de estilo, que es un documento independiente al de sgml. Únicamente es un documento escrito en DSSSL el cual indica que estilo se le quiere dar al documento sgml (si será todas las páginas iguales, diferentes, con parrafos, con índices, los títulos de tal tamaño ...). Por lo tanto, se podrán generar miles de documentos con estilos diferentes y en diferentes formatos a partir de un mismo documento SGML.

Una vez que se ha construido un documento .sgml y otro .dsl se procesará mediante el comando openjade, como ya se dijo, para generar el backend final. El módulo SP, mediante los DTDs que tenga instalados para procesar ese tipo de documento .sgml, se encargará de generar y validar el documento .sgml. Una vez validado, OpenJade mediante el motor de estilo y el documento .dsl, generará el backend solicitado

2.13. Resumen

En este capítulo, se ha conseguido adquirir conceptos básicos necesarios para situarse y seguir el problema. También se ha presentado el contexto de la herramienta, para que sirve, que salida se obtiene y como se puede utilizar. También se da una explicación simplificada de los estandares que procesa.

Capítulo 3. PRESENTACIÓN DEL PROBLEMA

En este capítulo, primeramente, se contextualizará el problema, es decir, la situación en la que se encontraba antes de realizar el proyecto. A continuación, se hará una descripción de la herramienta openjade describiendo las diferentes partes de la cual esta formada, su organización y estructura interna. Y Finalmente, se explicará en que consiste el proyecto realizado, los puntos y sus características más importantes.

En primer lugar aclarar que la construcción de las clases necesarias para la implementación del proyecto es la consecuencia del seguimiento del estandar en donde se especifica el proceso y que características y clases deben existir.

En los estandares se especifica claramente, los pasos obligatorias a seguir en la implementación de la herramienta en todas sus partes, exceptuando en la parte final, el formateo o construcción del backend. Así pues, a la hora de construir las clases de la cual se formarían las bases del proyecto, no sólo se tubo en cuenta el cómo construirlas, sinó también el dónnde situarlas. Se intento insertarlas en el mejor sitio posible para ajustar la implementación lo más posible al estandar.

La implementación del formateo, en cambio, fue algo más flexible debido a que en el estándar sólo se comenta el hecho de que hay que conseguir un backend para su realización pero no se comenta cómo se debería llevar a cabo, ni el tipo de backends que se deberían construir.

3.1. Los comienzos

Al principio, cuando se comenzó a desarrollar el proyecto, hubo una dificultad añadida a la hora de documentarse y ubicarse; ya que no existía suficiente documentación.

No existía ningún libro que hiciera referencia a OpenJade (OJ) o Jade o DSSSL, únicamente unas pocas notas en inglés en Internet y por supuesto nada en español (de DSSSL mucho más que de OpenJade).

A parte, el código de OpenJade no está demasiado documentado, por no decir, que apenas existen explicaciones en los métodos de las clases o de las clases en si.

Además, aunque está desarrollado bajo el compilador GNU gcc, se utiliza librerías propias del lenguaje OpenJade, y prácticamente no se usa directamente ninguna de las de C++; con lo que primero se tubo que dominar los tipos y funciones básicas de este lenguaje.

A parte, a modo de introducción, para hacerse una idea, comentar que se sobrecargan los métodos de memoria dinámica, y hay una limitación a la hora de crear objetos en

memoria, dificultando aun más la creación de estos no pudiendo utilizar y crear objetos de cualquier forma.

Esto es debido a que se utilizar un módulo, el cual tiene una construcción de lista doblemente enlazada, que se encarga de ir recogiendo los objetos existentes en memoria e ir gestionandolos. Se puede llegar a decir que existe un garbage collector implementado, que simularía el comportamiento de lenguajes más avanzados en el tiempo, como por ejemplo Java.

Por todo esto, y más, era totalmente inviable intentar comprender todos las funciones y tipos básicos y menos básicos del propio lenguaje, debido a la gran cantidad de líneas existentes de código. Para hacerse una idea, sólo el motor de estilo está formado por unos 40 ficheros, y esto supone un total aproximado de 40 mil líneas de código.

Con lo cual, después de tener unos conceptos básicos del lenguaje de estilo junto con sgml (véase la introducción) y tener claros los objetivos a desarrollar en el proyecto abierto y el esquema interno (de forma teórica) de OpenJade; se toma la decisión de encarar el código en diferentes etapas.

Primero, se debía desarrollar y saber como estaba organizado de forma teórica el proyecto OpenJade. Se intentó localizar, de la forma más concreta posible, los ficheros, los cuales, podrían tener una relación directa con el proyecto. Simplificando al máximo los ficheros a mirar.

Con ayuda de la documentación que dejó James Clark y la que Sourceforge.net facilitó en su momento, se pudo ubicar bastante bien el problema.

Una vez situado el problema, al intentar analizar el código, se observó que existe una gran cantidad de clases y herencias, y una gran cantidad de asociaciones del tipo friend.

Decir, que no existía separación de clases por ficheros, sinó que la separación se realiza de forma funcional, como si se tratara de librerías y de una forma muy compacta. Esto supone una gran cantidad de clases diferentes en un mismo fichero no respetando una buena orientación a objetos.

Esto supuso, crear diagramas UML, para poder comprender el código y su estructura de una forma mucho más clara y no perder mucho tiempo recordando que atributos o características posee una determinada clase y que tipo de relaciones existen entre ellas.

De esta forma se puedo ver que papel juega cada una de ellas en el la totalidad del proyecto, formando una mejor idea de como solucionar el principal problema de todos, donde colocar la clase central y principal de nuestro proyecto.

Aprovechando la situación, ahora valdría la pena explicar con algo más de detenimiento la estructura y organización tanto de los ficheros como la de las clases principales.

El siguiente apartado, es importante si se desea saber donde se debería buscar a la hora de añadir nuevas funcionalidades. James clark, juntó ficheros y directorios por funcionalidades. Esto fue de gran ayuda para comenzar la implementación del proyecto y situar el contexto.

3.2. Esquema Interno de OpenJade

En este apartado y en el siguiente, se hará un resumen de las partes más importantes de cómo y de qué esta formado la herramienta Openjade. Así pues, a continuación se explica el esquema interno de esta herramienta.

El constructor de árbol (GroveManager) crea en primera instancia, el árbol Grove a partir de analizar (Parser) el documento escrito en SGML.

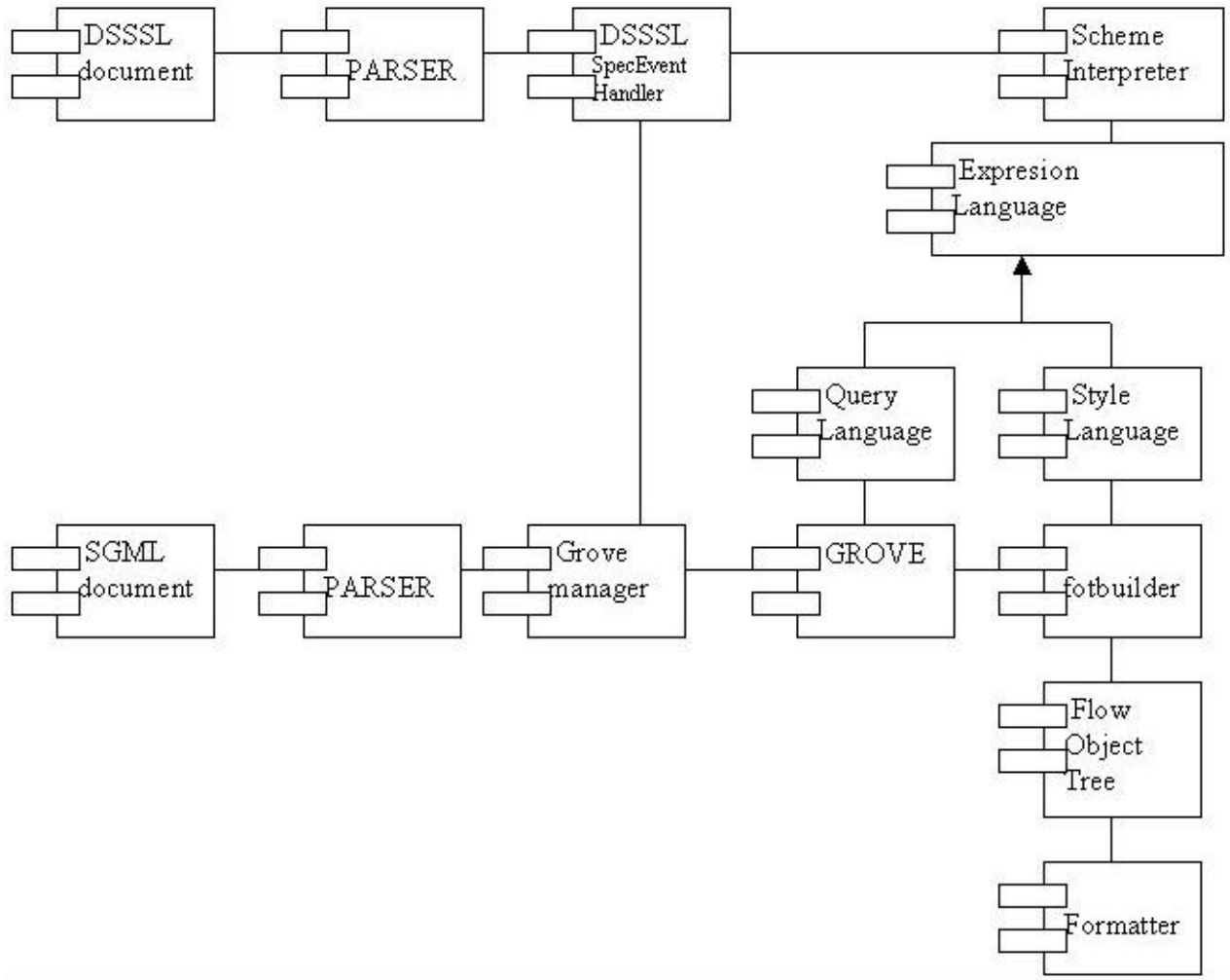
El analizador y validador de Scheme Parser nsgml, es el encargado de parsear los documentos SGML y enviar la información al Grove. Este proyecto se gestiona independientemente actualmente, a Openjade. El proyecto en si se llama SP, y viene junto con las librerías que utilizará tanto OpenJade para su desarrollo como el motor de estilo. OpenJade utiliza este validador en su implementación.

También se validará por otra parte, los paquetes de hojas de estilo DSSSL. Del empaquetado se encargará SpecEventHandler. Grove Manager puede utilizar llamar al SpecEventHandler para acabar de construir el arbol Grove.

Openjade tiene un intérprete de Scheme, el cual irá interpretando los paquetes de estilo y los irá procesando mediante los objetos del lenguaje de expresión (ELObj). ELObj es la clase padre abstracta que engloba a todas las clases que forman el lenguaje.

A parte, el estandar, da la posibilidad de que se puedan hacer consultas o query sobre los objetos Grove construidos, mediante el lenguaje de expresión, pero de esta parte hay muy poco construido.

Figura 3-1. Modulos en el proceso de construcción



Estas reglas de expresión, se subdividen principalmente en dos grandes grupos: las de consulta (Query Language) y las que forma las reglas de construcción del lenguaje (Style Language).

Este proyecto, se centra, en las reglas de construcción del lenguaje, dejando de lado las de consulta.

A continuación, el constructor de árbol de flujo FOTBuilder se encargará de la construcción del árbol de objetos de flujo (FOT), construirá el árbol FOT, mediante: estas reglas del lenguaje sumado al árbol de base Grove, ya construido, y ayudado de las consultas que le puedan proporcionar el lenguaje, sobre este Grove.

> **Nota:** La versión actual de OpenJade tiene un problema de eficiencia debido seguramente a esta falta de querys en el lenguaje de expresión.

Tanto el Query Language como Style Language forman el motor de estilo y son subclases de Expression Language. Una vez construido el árbol de objetos se generarán los respectivos árboles para los diferentes backends. Después mediante herramientas externas se pasará a su formateo.

El FOT o el arbol de objetos de flujo, es un árbol de donde se cuelgan los diferentes objetos de flujo (FlowObject) de una forma determinada, tal como se especifica en el estandar. Esta determinación vendrá dada por las reglas de construcción

> **Nota:** Este proyecto se centrará en la construcción del lenguaje, añadiendo nuevas funcionalidades al motor de estilo de la herramienta, modificación del arbol de objetos de flujo y su posterior foramteo en el backend tex (Formatter).
>
> Para poder insertar el objeto básico de nuestro proyecto, será necesario que la clase maestra sea una clase del lenguaje de expersión nueva y diferente al resto, debido a sus características tal como se explicará posteriormente.
>
> Es decir, será una subclase de la clase abstracta ELOBJ.

3.3. Estructura y Organización de OpenJade

Este apartado es fundamental para saber exactamente están ubicados los ficheros a partir de la función de las clases que la componen, y por lo tanto, puede ayudar a saber donde se ha de tocar a la hora de añadir cualquier funcionalidad nueva.

El proyecto está dividido en dos grandes módulos independientes, pero se complementan en el proceso de creación: JADE y SP. El modulo SP, a grandes rasgos, se encarga de proporcionar a JADE un entorno de librerías para su desarrollo.

También se encarga de analizar y validar el o los documentos SGML con ayuda de los DTD mediante nsgml, dando como resultado un documento SGML validado.

El modulo JADE se encarga, mediante el motor de estilo, de dar estilo a los documentos SGML validados, obteniendo; con la ayuda de las hojas de estilo DSL; diferentes tipos de backends.

Estos backends se utilizarán para producir, mediante otras herramientas externas dándoles formato, los diferentes tipos de ficheros donde se podrán ver el resultado final de la aplicación del estilo.

> **Nota:** El estándar no contempla el formateo externo una vez generados los backends.

Si se mira bajo el punto de vista de capas, Openjade esta dividido en las diferentes capas:

- La capa más baja es una biblioteca de fines generales de la clase, que es independiente de SGML/XML
- La siguiente capa es, básicamente, un interfaz para dar servicio a la capa de análisis de SGML. En esta capa esta:
  - El envío de mensajes API (MessageReporter, Message)
  - Catalogo API (EntityCatalog)
  - El juego de caracteres API (CharsetInfo)
  - EntityManager

- La capa siguiente es la base del análisis de documentos SGML, y esta depende de las dos anteriores. Las clases publicas son: SgmlParser y Event. Las plantillas para esto están en parser_inst.m4.
- La cuarta capa esta el motor que genera la estructura, esta depende de la anterior. La clase pública es ArcEngine y la plantilla para la generación de las instancias están en arc_inst.m4
- La siguiente capa es la puesta en práctica de EntityManager, pero no depende ni del analizador de SGML ni de ArcEngine, esto por ejemplo se determina cuál es la sintaxis de un identificador del sistema. La clase pública es ExtendEntityManager y la plantilla para las instancias de esto están en xentmgr_inst.m4
- En esta capa se encuentra un interfaz genérico para los árboles (groves). Esta no depende de ninguna capa anterior. Este interfaz se puede encontrar en grove/Node.h
- La siguiente capa es la puesta en practica de la anterior utilizando el modulo SP. Esta no depende del interfaz EntityManager. Se encuentra en el directorio spgrove y la clase principal pública es GroveBuilder
- En la penúltima capa se encuentra el motor de estilo de OJ, es la puesta en practica del lenguaje de estilo DSSSL. Es la parte de la construcción, no del formateo. Y esto esta en el directorio style. Lo podemos dividir en dos partes:
  - El empaquetado de las hojas de estilo DSSSL como un documento SGML usando el ArcEngine. Esta no depende de la implementación de EntityManager ni del interfaz grove. Las principales clases son DssslSpecEventHandler y StyleEngine
  - El procesamiento de los elementos de DSSSL. Este depende solo de la EntityManager y del interfaz de grove. El principal interfaz es FOTBuilder el cual hace de interfaz entre el proceso de construcción del árbol de flujo y el formateo.

- En la capa más externa estaría las diferentes implementaciones del interfaz FOTBuilder (los backends).

Nota: Como ya se dijo, este proyecto se centra básicamente en las dos ultimas capas.

También existe una organización por paquetes o directorios:

Los paquetes necesarios durante la construcción están ubicados en los siguientes subdirectorios de Openjade:

- all : proceso de la estructura Visual C++
- config: proceso de la estructura autoconf
- jadelist: crea una distribución.

Los siguientes directorios contienen los ficheros de documentación:

- doc: documentación del SP
- jadedoc: documentación de JADE
- develdoc: encontraras información sobre el desarrollo del openjade, lo nuevo y lo que falta por hacer (en concreto en el fichero missing)

Los siguientes directorios contienen ficheros de texto SGML/DSSSL:

- dsssl: DTDs para las hojas de estilo y ejemplos, aquí es el directorio por defecto donde se podrán probar los ficheros .sgml y .dsl mediante el comando openjade
- pubtext: diferentes DTDs y catálogos.
- unicode: declaración SGML y catalogo para trabajar con Unicode

Los siguientes directorios contienen los fuentes de diferentes librerías:

- include: cabecera para hacer referencia a el modulo SP o cabeceras de la librería SP , según sea la distribución, si es tarball o modular
- lib: solo existe en la distribución creada en forma de tarball por ejemplo la 1.3.1, aquí estará toda la librería SP
- style: es donde se encuentra el motor de estilo DSSSL
- grove: un interfaz abstracto para los árboles
- spgrove: una implementación del interfaz grove
- groveoa: una versión OLE de constructor de grove, solo para Windows
- generic: un simple API para aplicaciones derivadas del SP

Los siguientes directorios contienen aplicaciones:

- nsgmls: un analizador de SGML
- sgmlnorm: un normalizador de SGML
- spam: un sencillo editor de markup
- spent: un interfaz para SP entity manager
- sx: un simple conversor de SGML a XML
- jade: es donde estará la aplicación openjade después de ser compilada, y los diferentes backends que puede utilizar

Nota: Nos centraremos en los directorios style & jade

3.4. Situación actual, exposición del problema

En la actualidad, la herramienta Openjade, únicamente es capaz de procesar documentos ~simples~, es decir, que las instrucciones para procesar un documento SGML, junto con su documento de estilo DSSSL, no presentan gran complejidad.

Se podría decir que sólo es capaz de procesar un documento, el cual únicamente esté compuesto por una cabecera, un cuerpo y un pie de página, o simplemente conste de una secuencia de objetos en una misma región.

Sin embargo dentro de este cuerpo o región, la herramienta es capaz de albergar todo tipo de características ya sean heredables, comunes, o características especificas, o contengan objetos de todo tipo.

Esto ha de ser siempre sobre la misma región en la cual se prefijan los límites al principio del documento. Sobre estos objetos se basan los estándares donde vienen recogidos todas las formas posibles de objetos y características

La intención de este proyecto, era conseguir que se creara un modelo de página junto su formateo posterior para una secuencia de páginas, y este modelo de página no debía ser simple, sino complejo.

Es decir, conseguir crear un backend donde en la misma página o en una secuencia de páginas se pudiera contener diversas regiones, y estas se pudieran incluir todo tipo de objetos con sus características correspondientes.

Sería divagar mucho si nos centráramos a explicar todos los objetos y características que poseen cada objeto para conseguir el procesamiento de este tipo de documentos, cosa que no es el problema que se nos plantea. Si que sería conveniente, en cambio, explicar muy por encima conceptos básicos o la sintaxis utilizada en este entorno, ya que en las secciones o los capítulos siguientes se harán referencia a estos conceptos. De esta forma, se consigue que se siga mejor el hilo del desarrollo del proyecto.

Hasta el momento han salido conceptos como objetos o objetos de flujo, árbol de objetos de flujo o FOTBuilder, características heredables o no heredables, comunes, etc.

Son conceptos que están directamente ligados con el proyecto y no se puede continuar explicando otras cosas sin detenerse un momento a definirlos, aunque sea de forma resumida. Sin embargo, estos conceptos están más ampliamente explicados en ~**The DSSSL Book**~ por *Javier Farreres.*

La herramienta openjade, esta compuesta por objetos, entre ellos están los objetos de flujo o Flow Objects. Estos objetos son los encargados de modelar el proceso de creación del backend en uno de los procesos de transformación, llamado el proceso de estilo. Estos están directamente involucrados.

Cuando en el documento SGML se crea una marca en la ejecución lo que se estará creando es un objeto o una serie de objetos de flujo con una serie de características. Estas propiedades o características que poseeran los objetos de flujo cuando se construyan se definen y son construidas a partir de las definiciones que se encuentren en la hoja de estilo DSSSL del documento.

Cada objeto de flujo está definido en el estándar y cada uno puede que tenga una serie de características específicas o no heredables, o/y de otras que pueden ser heredables.

También hay una serie de características que son comunes para cualquier objeto de flujo, aunque después las específicas puedan predominar sobre las comunes o las heredables.

Todas estas características y objetos de flujo están recogidos y bien estructurados en ~**The DSSSL Book**~ por *Javier Farreres*. Libro aconsejable si se quiere complementar la información que aquí se da, ya que esta documentación va dirigida al problema que se plantea y carence de explicación en algunos aspectos.

Las características heredables o también llamadas IC (inherited caracteristics) en la implementación de los métodos, son todas aquellas características que se heredarán de los objetos de flujo padres. Dicho de otra forma, son los valores de cierto tipo de atributos de los objetos que engloban a otros. Estos valores serán los valores por defectos de los objetos de flujo que estén por encima de ellos, es decir, si en la definición de un artículo se le indica que se usará un determinado tipo de letra junto con su tamaño, cualquier objeto de flujo ya sean párrafos que se van creando, texto dentro de las tablas, etc., por defecto tendrá ese tipo de valor, a no ser que se le indique específicamente lo contrario.

Lo mismo pasaría si se redefinen estos valores a una tabla, los objetos que contienen esta tabla deberían coger los valores por defecto de este.

Las características no heredables o NIC (Non Inherited Caracteristics), son todas las características propias que no se heredan de ningún otro objeto de flujo. Aunque en la documentación se comentará por encima, ya que no es el problema que se plantea en el marco de este proyecto, decir que en el constructor de árbol de flujo o FOTBuilder existe una estructura NIC creada para recoger este tipo de valores de las características no heredables.

Llegados a este punto, se debería explicar que es FOTBuilder o Árbol de Objetos de flujo (Flow Object Tree), aunque ya se haya hecho referencia en este mismo capítulo con anterioridad.

Como se puede observar en el esquema interno de la herramienta Openjade, el FOTBuilder pertenece al motor de estilo (directorio style, véase en la sección: estructura y organización). De hecho, es la pieza central del motor de estilo de Openjade. Es el encargado de construir el FOT o el árbol de objetos de flujo (su código se puede encontrar en style/FOTBuilder.cxx).

En el árbol de objetos de flujo, se van colocando por orden de aparición los objetos de flujo, colocándolos debajo del objeto padre que les toca. Para esto existe un concepto que se repite bastante llamado puerto, que sirve para indicar si se puede colocar o no un determinado objeto en este objeto de flujo.

Hay que decir, que en cualquier tipo de documento ya sea artículo o libro, se parte de un puerto raíz, en el cual, en principio, sólo se debería poder situar los objetos ~*sequence*~ o ~*simple-page-sequence*~, es decir, el objeto secuencia por defecto o el actual objeto de secuencia de páginas simples.

> **Nota:** En este momento, la herramienta Openjade no restringe el uso de puertos, es decir, que objetos de flujo puede estar dentro de cual, así pues se podría insertar dentro de un objeto sequence un simple-page-sequence, o una secuencia de paginas dentro de una página simple etc.

De FOTBuilder se heredarán los diferentes árboles que corresponden con cada uno de los backends que se pueden generar, que junto con el proceso de formatter o formateo se creará el documento final.

Como ya se menciona en la esta sección, estos backends se guardan en el directorio jade en los cuales hay un constructor de árbol para cada backend y estos a su vez corresponden a un fichero .cxx diferente. Una vez, dicho esto, a partir de ahora, la explicación se centrará en la clase ~page-sequence~ junto con su modelo de página definido en el estándar, y su formateo final.

Para realizar el problema planteado, simple-page-sequence será la clase que se utilizará como referencia, ya que aunque tiene diferencias obvias con page-sequence, ambas tienen algo en común: que son la base para poder introducir nuevos objetos de flujo en estas clases, es decir que ambas deben partir del nodo raíz.

Pero más que comentar como estarán implementadas y esto haría que se alargara excesivamente el problema planteado, el page-sequence se enfocará partiendo de dos premisas, la primera para que se vea la diferencia de las limitaciones de un modelo de página simple ~simple-page-sequence~ respecto al nuevo modelo, el cual se intenta implementar. Este es un modelo de página complejo ~page-sequence + page model~.

Por si no hubiera quedado quizá del todo claro, al objeto de flujo ~page-sequence~ a diferencia de ~simple-page-sequence~, se le puede definir un modelo de página complejo llamado ~page-model~. Este modelo de página se definirá en el documento DSSSL, mediante *define-page-model.*

A continuación se muestra como se define un modelo de página en la hoja de estilo para SGML, y que parámetros debe o puede contener:

Figura 3-2. Definición del modelo de página

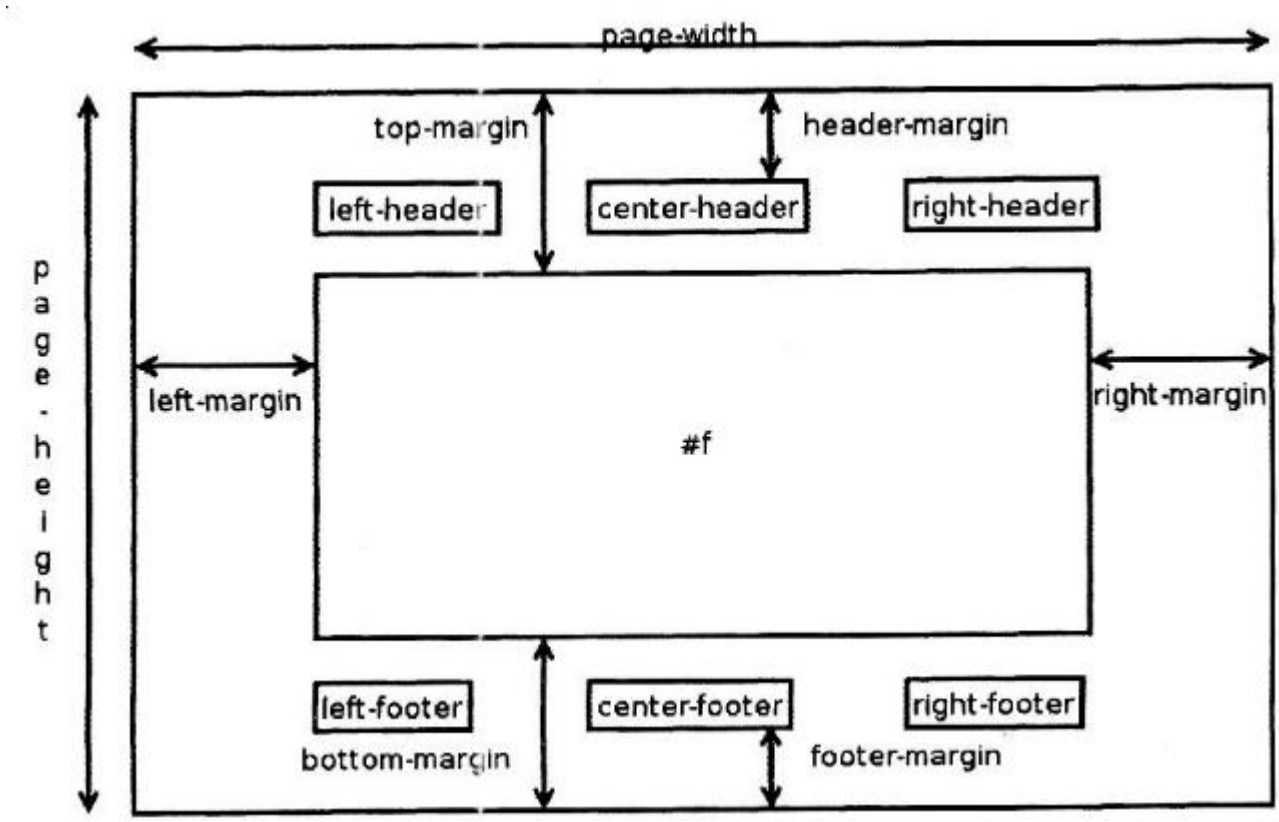
```
(define-page-model page-model-name
  (width length) ;required
  (height length) ;required
  (filling-direction ;optional
    'left-to-right | 'right-to-left | 'top-to-bottom)
  (decorate (decoration-area ...)) ;optional
  (region ;at least one region required
    (x-origin length) ;required
    (y-origin length) ;required
    (width length) ;required
    (height length) ;required
    (header ;optional
      (height length) ;optional
      (width length) ;optional
      (filling-direction ...);optional
      (contents-alignment ;optional
        'start | 'end | 'center | 'justify)
      (generate unlabeled-sosofo)) ;required
      ;the unlabeled-sosofo is a generated-object, see Section 7.3.3.4
    (footer ;optional
      (height length) ;optional
      (width length) ;optional
      (filling-direction ...);optional
      (contents-alignment ...);optional
      (generate unlabeled-sosofo)) ;required
    (flow port ...) ;optional
    (filling-direction ...);optional
```

```
;if page-region has no filling-direction, it inherits the
;filling-direction of its containing page-model. It is an
;error if neither the region nor its containing page-model
;defines the filling direction.
(decorate (decoration-area ...)))) ;zero or more
```

Además en el objeto de flujo de secuencia de páginas simple, solamente se puede definir una única vez, al principio la definición de cabecera, la región del cuerpo del texto y la región de pie de página.

Para ilustrar esto, a continuación se muestra que es una secuencia de páginas simples:

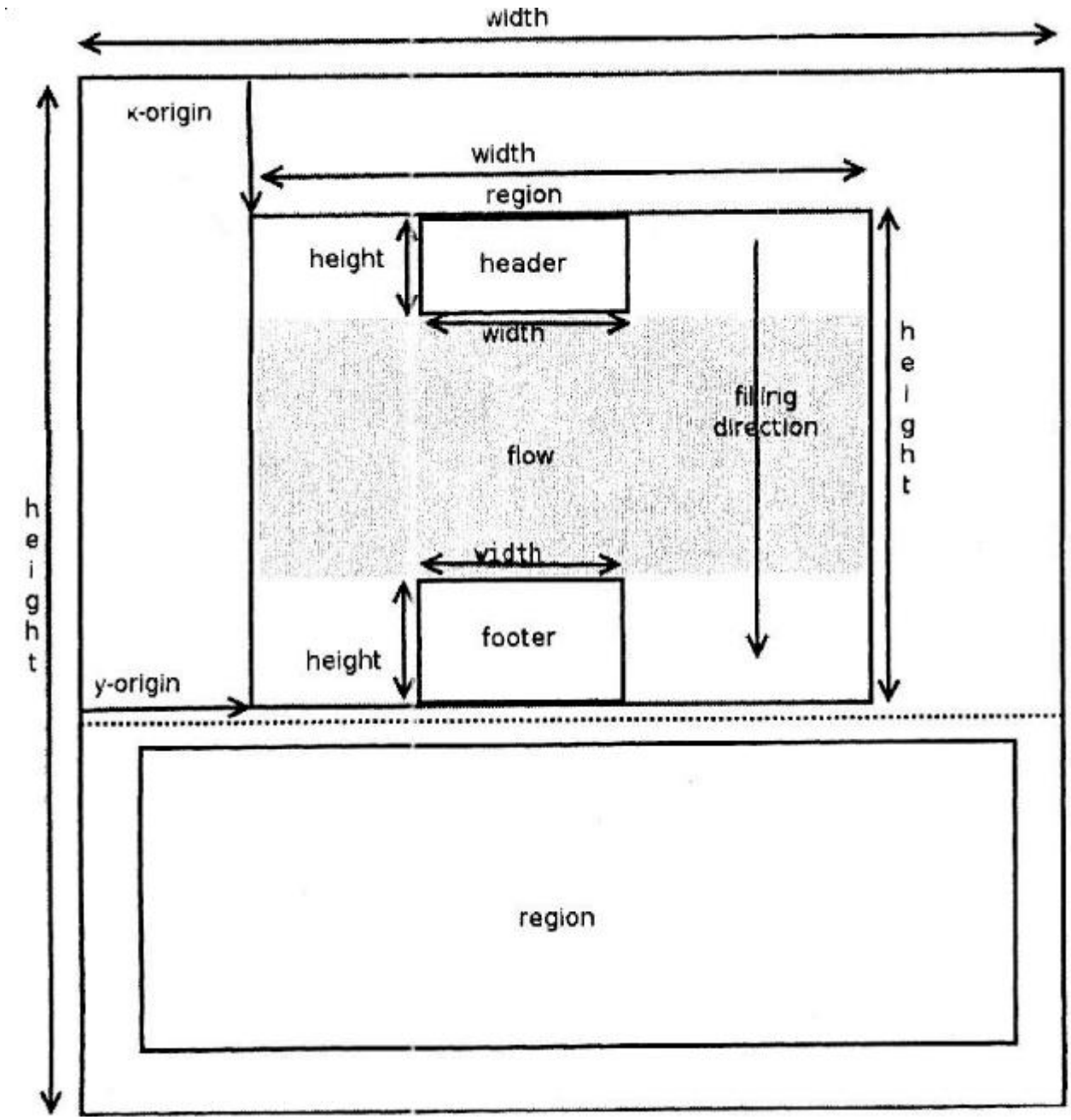
Figura 3-3. Simple Page Sequence



Como se puede observar, sólo existe una única región base para insertar los objetos de flujo (#f), la cual vendrá definida por las diferentes partes que constituyen una página: márgenes, cabecera y pie de página.

En lugar de esto, en este proyecto se intenta crear y formatear el modelo de páginas complejas para el objeto de flujo secuencia de páginas. Consiste en crear diferentes regiones en la misma página. Para ilustrar esto se muestra la siguiente figura:

Figura 3-4. Presentación de secuencia de página



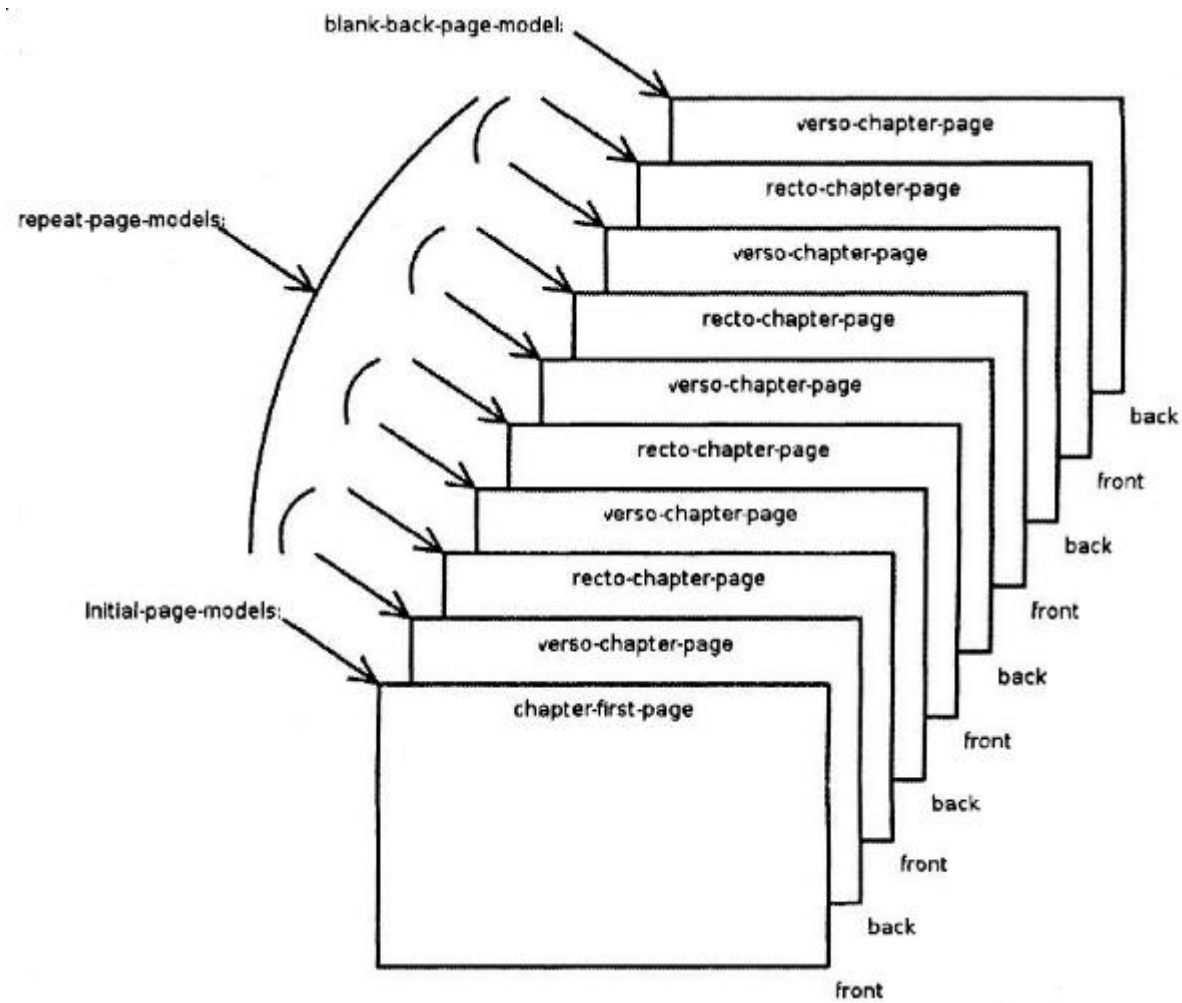
Se puede observar que cada región posee las mismas propiedades que una pagina simple la cual se muestra en la anterior figura: cabecera, región para insertar objetos de flujo y pie de página. Pero a parte se puede insertar más regiones en esa misma página. Las coordenadas donde se insertan las regiones, vendrán definidas por el usuario mediante la representación de puntos en los ejes (x, y) respecto a la esquina izquierda superior de la hoja. El usuario también indicara que tamaño deberá tener la region a partir del punto descrito anteriormente.

Nota: En principio, en el estándar no se indica en ningún momento que no se puedan solapar las regiones, aunque no se encuentra ninguna imagen que así lo indique, con lo que queda a manos del usuario.

Además, pese a que en este proyecto nos centraremos más en el modelo de pagina y su formateo, va bien comentar que otra de las diferencias respecto al modelo simple de secuencias de página, es que los modelos de página complejos pueden ir variando en un libro o artículo. Por ejemplo las partes de un libro varían de modelo; de aquí el objeto de flujo ~page-sequence~ y la definición de modelos de página.

En un libro, se ha de poder tener un modelo de página para las páginas de inicio, que pueden ser diferentes al resto de páginas del libro. De igual forma pasa en la conclusión del libro o en las ultimas páginas de este, este modelo se debería de poder diferenciar del resto y no tendría porque ser el mismo que los anteriores modelos de página.

Figura 3-5. Diferentes modelos que se pueden dar en una secuencia de páginas



Además, en las páginas del contenido del libro también se debería de poder insertar cambios de estilo por ejemplo, las páginas pares e impares deberían de ser ligeramente diferentes, formando espejo, para facilitar la composición y la creación de estas. En la ilustración anterior se muestra un modelaje para la creación de un libro mediante el objeto de flujo de secuencia de páginas y modelos de página compleja.

Para conseguir esto, en el estándar se definen estas características en el objeto de flujo ~page-sequence~ que en la implementación estarán recogidas en la estructura NIC del FOTBuilder:

Figura 3-6. Tabla de características de la secuencia de página

Characteristic	Inh	Values	Default	Description
initial-page-models	yes	'(page-model ...)	empty list	Specifies the page-models to be used for the initial pages. There should be one page model for each initial page.
repeat-page-models	yes	'(page-model ...)	empty list	Specifies the page-models to be used for the pages after the initial pages. These page models form a repeating sequence. For example, if the list contained two page models, these models would be used for alternating pages.
force-last-page	yes	#f, front, back	#f	Forces a type for the last page of the sequence. If needed, a blank page will be generated using `blank-back-page-model:`.
force-first-page	yes	#f, front, back	#f	Forces a type for the first page of the sequence. If needed, a blank page will be generated using `blank-front-page-model`.
first-page-type	yes	#f, front, back, parent	parent	Specifies the type of the first page. It doesn't cause the generation of blank pages; it only informs that this page will be of the specified type when printed or bound. It is used to determine which pairs of pages are spreads.
blank-back-page-model	yes	page-model	#f	Specifies the page model to be used for a blank back page. This is only used if the final page is a back page that was required by a `force-last-page:` or `force-first-page:` value.
blank-front-page-model	yes	page-model	#f	Specifies the page model to be used for a blank front page. This is only used if the final page is a front page that was required by a `force-last-page:` or `force-first-page:` value.
justify-spread?	yes	boolean	#f	Specifies whether the lower part of each page in a spread is justified. Potential spreads are those pages that, once bound, lie side by side upon opening the book, forming a space of a double page.

Characteristic	Inh	Values	Default	Description
page-category	yes	object for which the *equal?* procedure is defined	Undefined.	Specifies the category of the pages of this flow object. It is used for page numbering purposes with *category-page-number* procedure.
binding-edge	yes	left, right, top, bottom	left	Specifies the side where the binding will be. It affects whether a page side is considered in the inside or outside.

Nota: Todas estas imágenes son sacadas de: ~The DSSSL Book~ por Javier Farreres~

Así pues, cuando se defina las características en la hoja de estilo para un objeto de flujo de secuencia de páginas, servirán para indicar los diferentes modelos de página que en este libro o artículo se van a poder dar.

De esta forma, remodelando en la hoja de estilo los diferentes modelos de página, ya sea en un documento tipo libro o artículo, se podrían formar diferentes composiciones en su conjunto o diferentes composiciones en las diferentes partes de este documento.

Por último, decir, que el proceso de estilo para el formateo se hará sobre el backend Latex/Tex. En la actualidad la salida de en latex no es más que llamadas a macros de Latex/TeX. Esto seguramente fue debido a que posiblemente los que intervinieron en el proceso de creación de la herramienta conocían más Latex/Tex que el cómo estaba construido la herramienta en si.

Así que cuando se le indica a la herramienta Openjade que te construya el backend tex te genera un fichero el cual no se puede compilar directamente con latex, ya que contiene macros que están definidos en parte en una nueva herramienta que se hizo para esto, llamada Jadetex.

Por lo tanto, una de las intenciónes de este proyecto, a parte de conseguir un formateo del modelo de pagina, es conseguir crear un nuevo backend que sea completamente independiente de la herramienta Jadetex; es decir, extraer todas las definiciones de esta herramienta externa e incorporarlas de manera más eficiente en el proyecto Openjade. *Véase el anterior capítulo sección: Proceso de creación de estilo.*

Nota: Estas imagenes fueron extraidas tal cual, del libro *The DSSSL Book An XML/SGML Programming Language* de Javier Farreres.

3.5. Resumen

En este capítulo, se ha descrito y planteado el problema que se debe resolver de forma que en los capítulos posteriores se tenga un mejor enfoque y se pueda seguir mejor los conceptos más importantes sobre la herramienta Openjade. Se ha explicado el porqué es necesaria la creación del nuevo objeto de flujo ~page-sequence~ junto la definición del modelo de pagina, y que se espera conseguir. En el siguiente capítulo se despejará el cómo y en dónde.

Capítulo 4. IMPLEMENTACIÓN

Hacer un diagrama de clases es fundamental, para poder encarar el proyecto.

Una vez que se tenía claro el problema, la idea del esquema interno de la herramienta, y que ficheros podrían estar directamente involucrados, se pasó a hacer un estudio de clases.

Este estudio era importante debido a que se quería insertar nuevas funcionalidades en el proyecto, y entre ellas una de muy importante, el modelo de página.

A partir de este modelo, giraría buena parte del proyecto así que era importante seguir el estandar y no equivocarse en su inserción e implementación.

La existencia de un gran número de casos de multiherencia y clases amigas, debido seguramente a un mal diseño para evitar repetición de clases o estructuras locales, dificulto el diseño y su comprensión a la hora de crear el diagrama de clases. Así como la puesta en marcha de la implementación del proyecto.

Se optó por crear un diseño de clases principal, y otro que se fue separando en grupos por fucionalidades. En muchas ocasiones esta separación iba guiada por la información que daba el nombre de la propia clase o estructura, o en el contexto donde estaba situada.

En este proyecto se explicará únicamente las clases más importantes del proyecto, las que tienen una relación directa con el problema planteado, debido a que existen unas 250 clases diferentes con sus relaciones y métodos.

> **Nota:** En la etapa del estudio de la herencia de las clases se pudo observar un diseño algo enrevesado no propio de un buen diseño orientado a objetos, cosa que se ha intentado corregir y se ha evitado al menos en el desarrollo de este proyecto. Se intentó extraer las clases en ficheros separados pero intentando mantener el diseño ya creado, es decir, si se veía que al extraer la clase la cual se disponía a implementar, suponía modificar las clases ya existentes o que de esta manera en vez de claridad lo único que se conseguía es empeorar el diseño se desestimaba la opción, de extraer la clase, y se dejaba esta dentro del mismo fichero junto con las demás.

En una primera aproximación se optó por hacer un diseño de clases de lo que nos interesaba, del motor de estilo, ya que era aquí donde se tenía que insertar el modelo de página.

El modelo de página, pertenece al motor de estilo ya que es la clase principal para crear el árbol de flujo necesario cuando se quiere crear un estilo de página con diferentes regiones en una misma pagina.

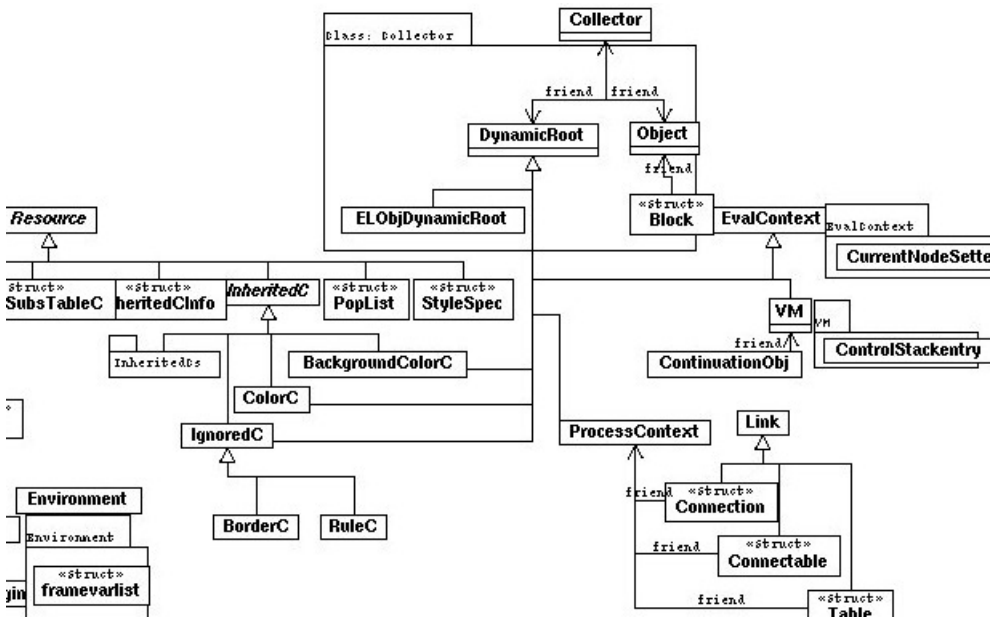
A partir de aquí se crean los diferentes nodos del árbol con los diferentes objetos de flujos.

Aunque antes de presentar, el objeto PageModel (modelo de pagina), es conveniente presentar primero otras clases importantes que han sido decisivas en el desarrollo del proyecto. Ya sea porque estan directamente ligadas con los objetos del proyecto o se han tenido que conocer para la implementación

4.1. Clase Collector

La primera clase a presentar es la clases Collector, aunque no aparece directamente esta clase se encarga de gestionar la memoria dinámica de todos los objetos de la aplicación y sobrecarga los operadores new a la hora de creación de nuevos objetos de forma dinámica. Así pues, partiendo de que el modelo de página es un objeto del lenguaje de expresión, la gestion de memoria dinámica lo realizará el collector de objetos.

Figura 4-1. Collector



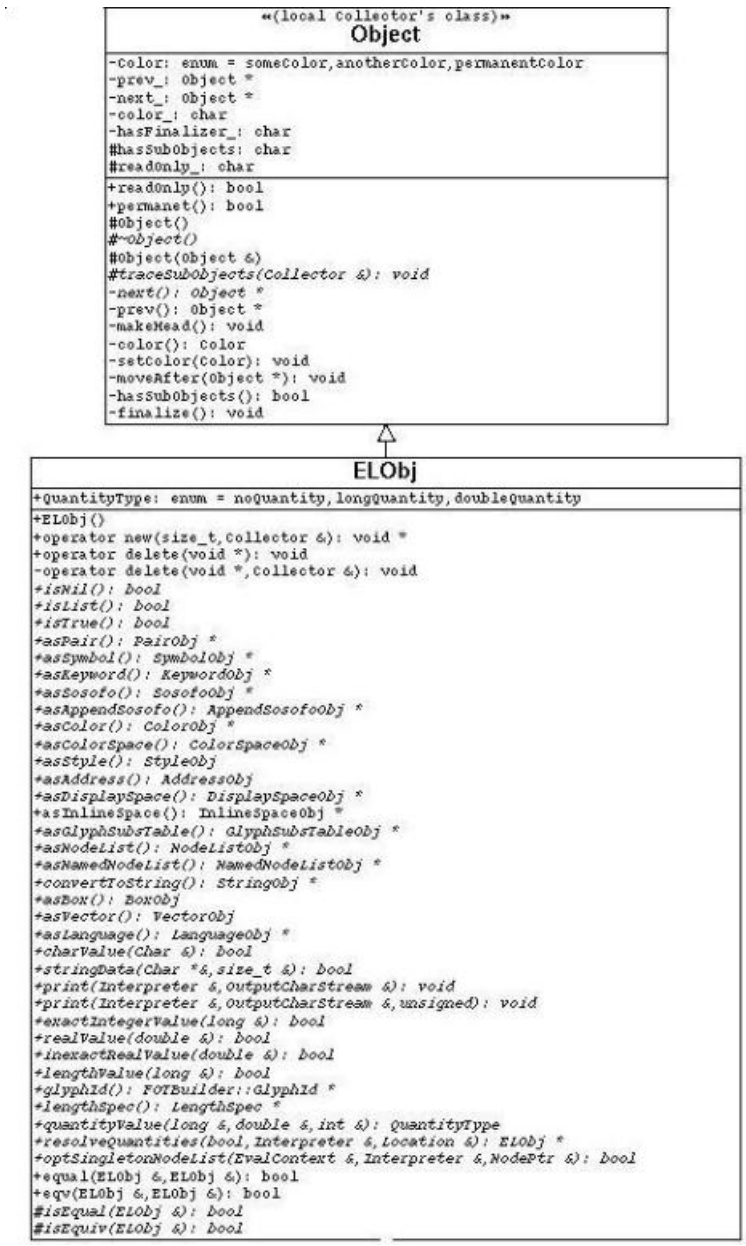
Esta clase se encarga de ir recogiendo los nodos de los objetos tipo Object que se van creando. La organización de la memoria dinámica en este proyecto consiste en una lista doblemente enlazada. La clase Collector irá gestionando la memoria dinámica que se dispone. Existe una limitación de memoria por nodo, si se sobrepasa la ejecución del programa dará una excepción por falta de memoria.

4.2. Clase Elobj

Una vez presentado el Collector la segunda clase en la importancia del desarrollo del proyecto sería Elobj

A diferencia de la clase Collector que es clase amiga de Object, para poder manejar métodos de esta, Elobj es subclase de Object por lo que hereda las características de esta misma. Digamos que si la clase Object es la clase abstracta que representa cualquier objeto de la aplicación, Elobj es la clase abstracta de la cual descenderán todos los objetos de expresión, las que forman parte del motor de estilo.

Figura 4-2. Elobj



Esta clase, fue junto con la que se presentará a continuación la primera la cual se modificó. Una vez que se tenía claro que representación y que significaba esta clase abstracta, el resto fue mucho más sencillo.

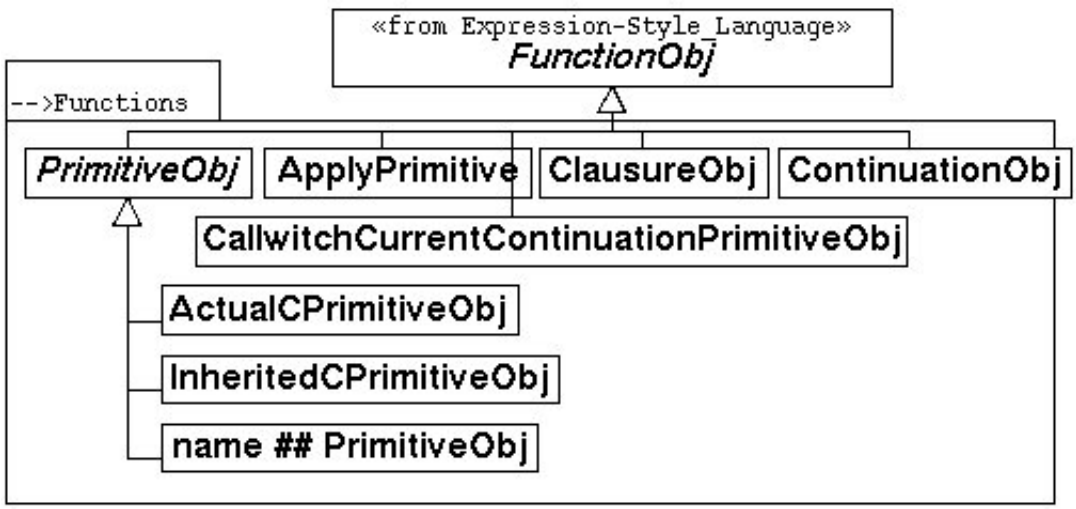
En esta clase se implementaron dos métodos virtuales, que obligan a las clases que heredan de ella a ser implementadas. Una sirve para devolver un objeto de tipo modelo de pagina y la otra es de tipo consulta, a partir de un una estructura del arbol de flujo, te indica si se trata de la estructura que guarda las propiedades de un PageModelObj o no.

4.3. Clase PrimitiveObj

A continuación, se añadio la definición de estas instrucciones, como *PRIMITIVE*. De esta forma, si en un futuro se implementa la parte query, o de consulta sobre el árbol de flujo, poder preguntar si un determinado objeto es un modelo de página o no, o si la estructura con la que se está trabajando es una estructura donde se guardaran las propiedades de modelo de página o no.

Para acabar de crear esto, se necesito insertar estos métodos en la clase FuntionObj. A continuación se muestra el diagrama de clases de lo anteriormente explicado.

Figura 4-3. PrimitiveObj



4.4. Clase PageSequenceFLowObj

A continuación, se explicara la clase PageSequenceFlowObj. Esta clase esta directamente relacionada con el modelo de objetos. De hecho en un proyecto anterior se tubo que construir el objeto que representa la secuencia de paginas. Este objeto en si tiene sus propias características heredables y no heredables (NIC - non inherited characteristics), y estas se construyeron cuando se creo este objeto.

Pero las características referentes al modelo de pagina se ignoraron, formando parte de este proyecto. De hecho es la unica clase que puede contener objetos del tipo modelos de pagina, ya que no tiene sentido en otro contesxto.

En my en resumen, PageSequenceFlowObj, es una secuencia de paginas, parecido a lo que sería SimplePageSequenceFlowObj o SequenceFlowObj en cuanto a la forma de como empezar a construir el arbol de flujo (FOT - Flow Object Tree). Debido a que solo existen estas tres formas en el estandar para comenzar a crear el arbol, Es decir, quizá lo que se quiere crear es una secuencia de objetos de flujo, unicamente (SequenceFlowObj), una secuencia de paginas simples para los objetos de flujo (SimplePageSequenceFlowObj), o tal vez una secuencia de paginas complejas (PageSequenceFlowObj).

Es decir, no se puede construir el árbol donde estaran todos los objetos de estilo, sino se parte de alguna de estas tres clases, igual que una de estas no puede estar dentro de alguna de las otras clases padres, ya que no tendría sentido. Es decir, es absurdo pensar que dentro de una secuencia de hojas simples puede englobar una secuencia de paginas complejas.

Como ya dijimos, una secuencia de paginas simples, son aquella secuencia de paginas en donde el area donde se pueden contener objetos de flujo es la pagina entera, exceptuando margenes, cabecera y pie de pagina, en cambio una secuencia de paginas compuestas, son aquella secuencia de páginas que contiene diferentes regiones dentro de la misma página.

De aqui el hecho que se necesiten asignar dentro de una secuencia de paginas un modelo de paginas, o una lista de modelos de paginas (PageModelObj). Y por este motivo, se dice que un modelo de paginas no tiene sentido sino esta dentro de una clase PageSequenceFlowObj.

> **Nota:** En la implementación de la versión actual ni en esta, se tiene en cuenta restricciones de este tipo en la construcción del arbol de flujo. Es una tarea que ha quedado pendiente hace mucho tiempo por hacer. De hecho ahora mismo si se construye un PageSequenceFlowObj dentro de un SimplePageSequenceFlowObj el PARSER no se queja, permitiendolo.

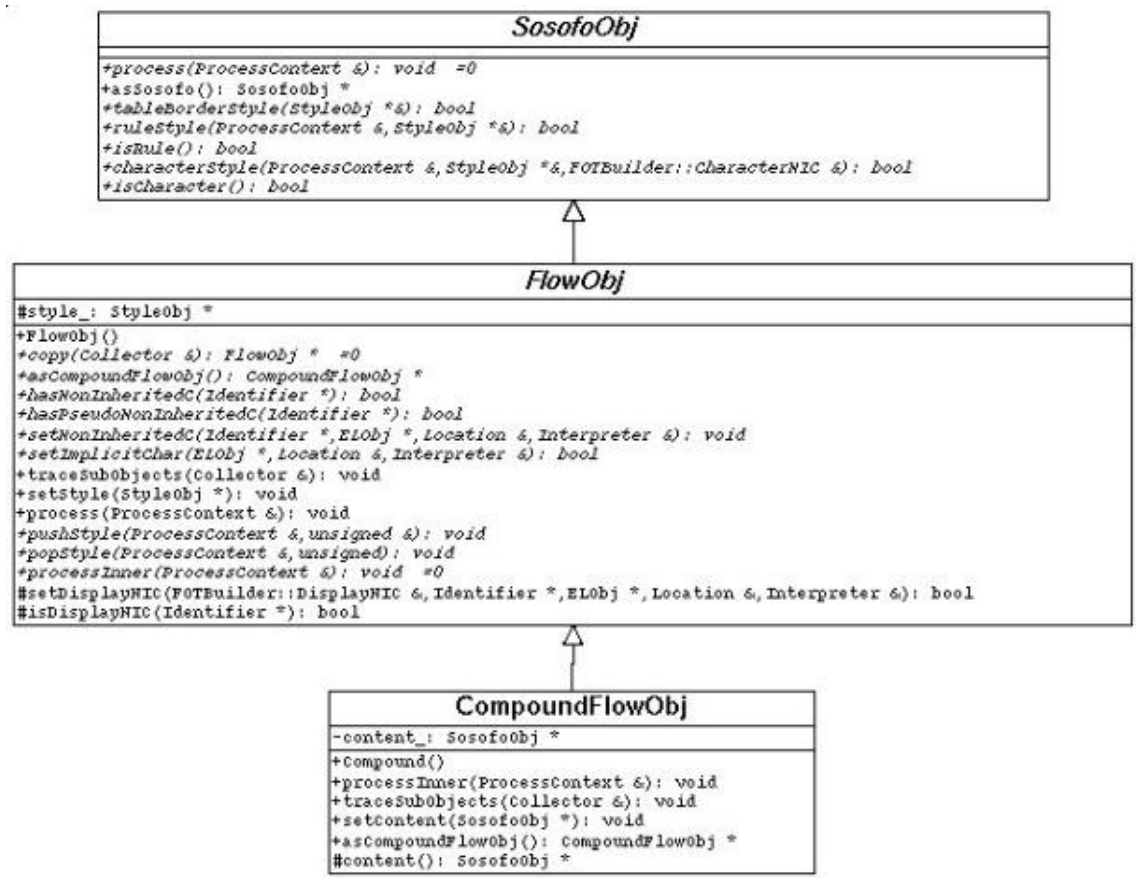
Como ya se ha visto la relación que existe entre el modelo de páginas y la secuencia complejas de páginas es muy estrecha, y esta relación viene ligada a partir de las carácteristicas que posea la secuencia de paginas en el momento de crear el objeto. Es decir,

si la secuencia de páginas no posee en la definición ninguna característica que contenga un elemento PageModelObj o una lista de elementos de este tipo, no se generarán

4.4.1. Clases FlowObj, SosofoFlowObj y CompoundFlowObj

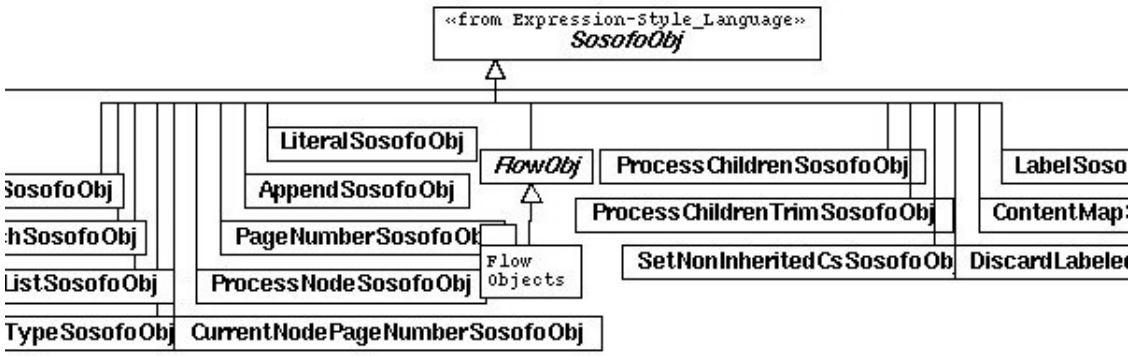
A continuación ubicar mejor la clase PageSequenceFlowObj es conveniente explicar aunque sea por encima las clases SosofoObj, FlowObj, y particularmente Compound-FlowObj, ya que ciertas propiedades que se utilizan en la implementación provienen de estas clases

Figura 4-4. Flow Object



Aunque no aparece en este diagrama, la clase SosofoObj desciende directamente de la clase Elobj, por lo que se puede ya hacer una idea de que contexto nos estamos moviendo. Objetos que son del lenguaje de expresión

Figura 4-5. SosofoFlowObj

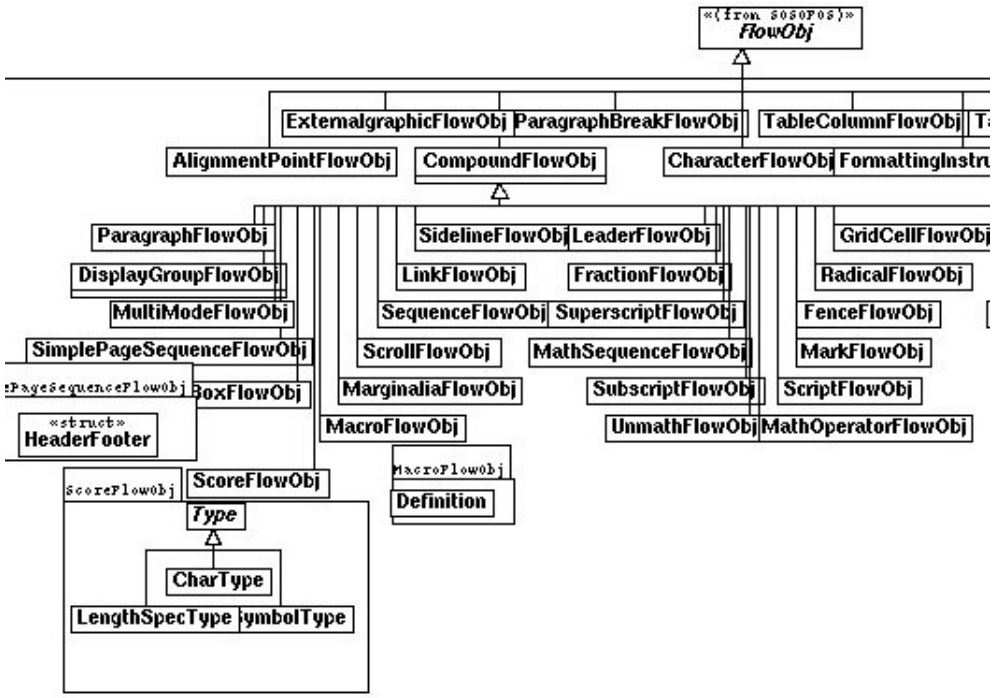


Aunque estas tres clases no se modificaron en el proyecto, vale la pena comentarlas para una mejor contextualización y ubicación del problema, ya que ciertas características son heredadas por estas clases debido a que PageSequenceFlowObj hereda directamente de la clase CompoundFlowObj, que esta a su vez hereda de la FlowObj y esta misma a su vez de la primera, SosofoFlowObj.

La idea del porque PageSequenceFlowObj cuelga de la clase CompoundFlowObj es fácil de entender. Se entiende como FlowObj o Objeto de flujo cualquier objeto que forma parte del flujo de datos del documento en si. Este tipo de objetos tendrá unas propiedades y reglas para expresar este tipo de comportamiento. Además la clase CompoundFlowObj son todos los objetos del lenguaje de expresión que estan formados por diferentes objetos de flujo. Sin extendernos demasiado, estos objetos de flujo compuestos, pueden estar relacionados en el arbol de objetos a partir de lo llamado como puertos, otros objetos compuestos y otros simples o atómcos.

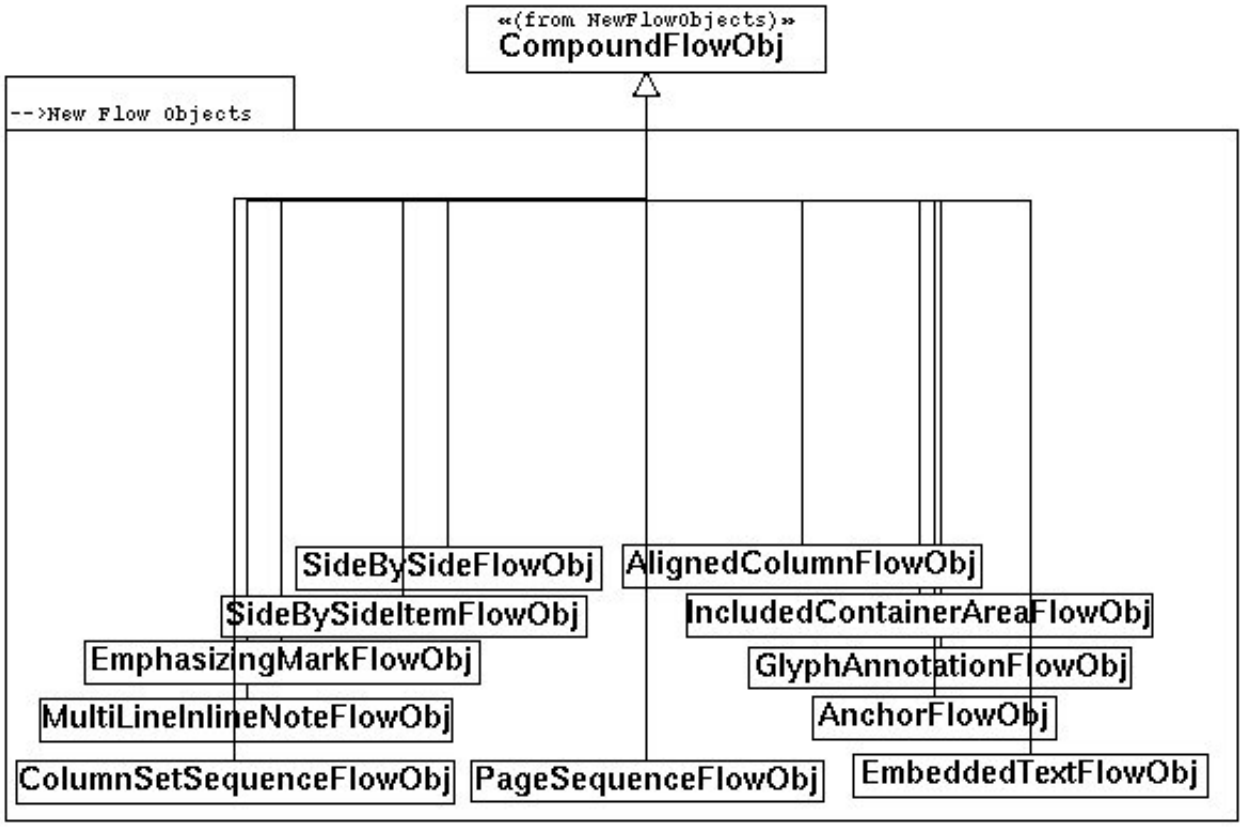
En este contexto se puede entender que casi todos los objetos que se manejan en el arbol de flujo que no sean atomicos o simples heredaran de esta ultima clase. Así pues tanto SequenceFlowObj, PageSequenceFlowObj y SimplePageSequenceFlowObj heredan de la clase CompoundFlowObj

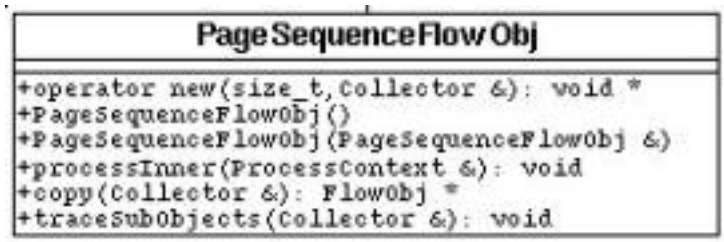
Figura 4-6. Objetos que heredan del FlowObj



En el siguiente diagrama se muestra la situación del objeto secuencia de paginas en el contexto actual y en la siguiente se pueden ver los métodos implementados que utiliza esta clase en concreto:

Figura 4-7. PageSequenceFlowObj





Una de las cosas que ya se ha hablado, y que poseen todos los objetos de flujo, son las características heredables y la propiedad de tener características no heredables. Si son heredables se van propagando hacia las clases hijas, siempre y cuando se puedan aplicar y tengan sentido y no se diga lo contrario. De aquí que las clases formen un arbol de objetos de flujo.

> **Nota:** Esto se hablará de forma más clara cuando se explique una de las clases mas importantes en el desarrollo del proyecto: el arbol de objetos de flujo; junto con modelo de pagina y la secuencia de paginas complejas.

Referente al PageSequenceObj no hubo modificaciones en la clase en si, ya que estaba preparada para recoger las características que contenian modelos de paginas. Todas estas características al ser especificas son de tipo no heredables. Así pues este tratamiento, es decir, asociar los modelos de paginas que se definan por el usuario en la hoja de estilo con las características que posea esa secuencia de paginas definido en el documento sgml, se realizarán en las posteriores clases que se explicarán a continuación

4.5. Clase SchemeParser y Interpreter

Scheme Parser y Interpreter, son las clases que estan directamente involucradas en el proceso que hemos descritor anteriormente, asociar las caracteristicas definidas en la hoja de estilo al tag o clase definida en el documento sgml. Interpreter y Scheme Parse estan estrechamente relacionadas. Scheme Parse se encarga de parsear la hoja de estilo definida por el usuario, mientras que Interpreter interpreta esos Strings convertiendolos a las clases correspondientes.

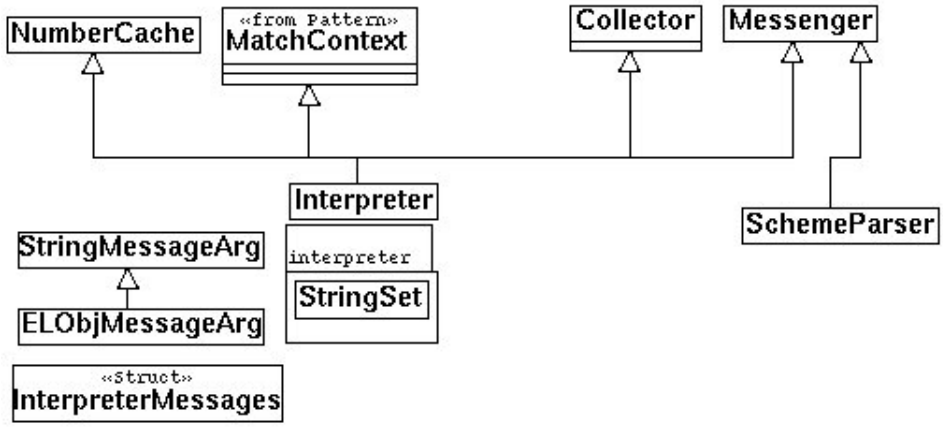
Es decir, Scheme Parse mira que este bien definido, y pasa el token al interpreter que se encargará de crear la estructura para recoger el valor y asociarlo. Luego una vez acabado Scheme Parser recogera esos objetos y los hara permanentes en memoria, para que el constructor de arbol de flujo FOTBuilder pueda tratarlos.

Interpreter es el encargado por lo tanto, de crear un objeto PageModelObj o una lista de objetos PageModelObj, segun el token que se reciba de la clase Scheme Parser. Esta información se pasa mediante mensajes. Interpreter tambien se encarga de asignar por lo tanto las caracteristicas no heredables a al tag de PageSequence.

Las características heredables se tratan de diferente forma y es la clase InheritedC la encargada de esta implementación, pero al no se explique en este proyecto.

Así pues, se tubieron que implementar los métodos en SchemeParser que chequeaban las reglas para parsear el documento de estilo. También se tubieron que añadir tanto los metodos para la interacción entre las dos clases y las instrucciones necesarias para reservar la memoria dinámica necesaria para contener tanto un solo objeto PageModelObj o una lista de modelos de página en el caso de que fuera necesario en la clase Interpreter.

Figura 4-8. Scheme Parser y Interpreter



4.6. Clases FOTBuilder, Save/SerialFOTbuilder

Una vez explicado, como se gestiona la memoria, y que clases estan implicadas en el proceso de recoger y asignar los valores de la definicion de la hoja de estilo y del documento sgml, para el modelo de pagina; el siguiente paso sería como se encapsula esta información para su manipulación.

Como se ha explicado anteriormente, en la clase Interpreter es donde se crean los PageModelObj o la lista de PageModelObj para contener esa información y para que el constructor del arbol de objetos pueda trabajar con estos.

A continuación se explica la implementación del modelo de página. El PageModelObj no es un objeto de flujo en si, sino es un objeto donde se guardará la información referente a las propiedades que poseerá el objeto de flujo Page Sequence. Por lo tanto, es independiente a los objetos de flujo vistos anteriormente y no extenderá de la clase CompoundFlowObj como el SimplePageFlowObj o PageSequenceFlowObj.

De hecho no tienen nada que ver estos dos conceptos. La única relación es que la secuencia de paginas utiliza el modelo de paginas en alguna de sus propiedades. Es más como ya se comentó anteriormente se puede construir una secuencia de páginas sin crear modelo de paginas. En este caso se construiria como un model de paginas simple.

La clase PageModelObj se extiende de ELobj ya que es un objeto como otro más del lenguaje de expresión.

El PageModelObj contiene todos los atributos necesarios para crear una region dentro de una pagina, como se especifica en el estandar explicado en el capitulo anterior de esta documentación.

Se crea un fichero para esta implementación a parte: pageModelObj.cxx/h. Contiene métodos getters y setters necesarios para obtener los valores de sus atributos, y para que el constructor FOTBuilder pueda obtener estructuras propias suyas a partir de un objeto de este tipo.

Para facilitar el proceso de extracción de información, y como FOTBuilder maneja estructuras propias que se crearon a continuación, se creó como atributo una estructura de este tipo del FOTBuilder (StModel).

También se tubo que incluir otro atributo que contenia el identificador del modelo de pagina, por si hubiera mas de un modelo de pagina dentro de un objeto de flujo de secuencia de paginas para poderlos identificar. Este identificador se entenderá el porque cuando se llegue al proceso de formateo en la parte final.

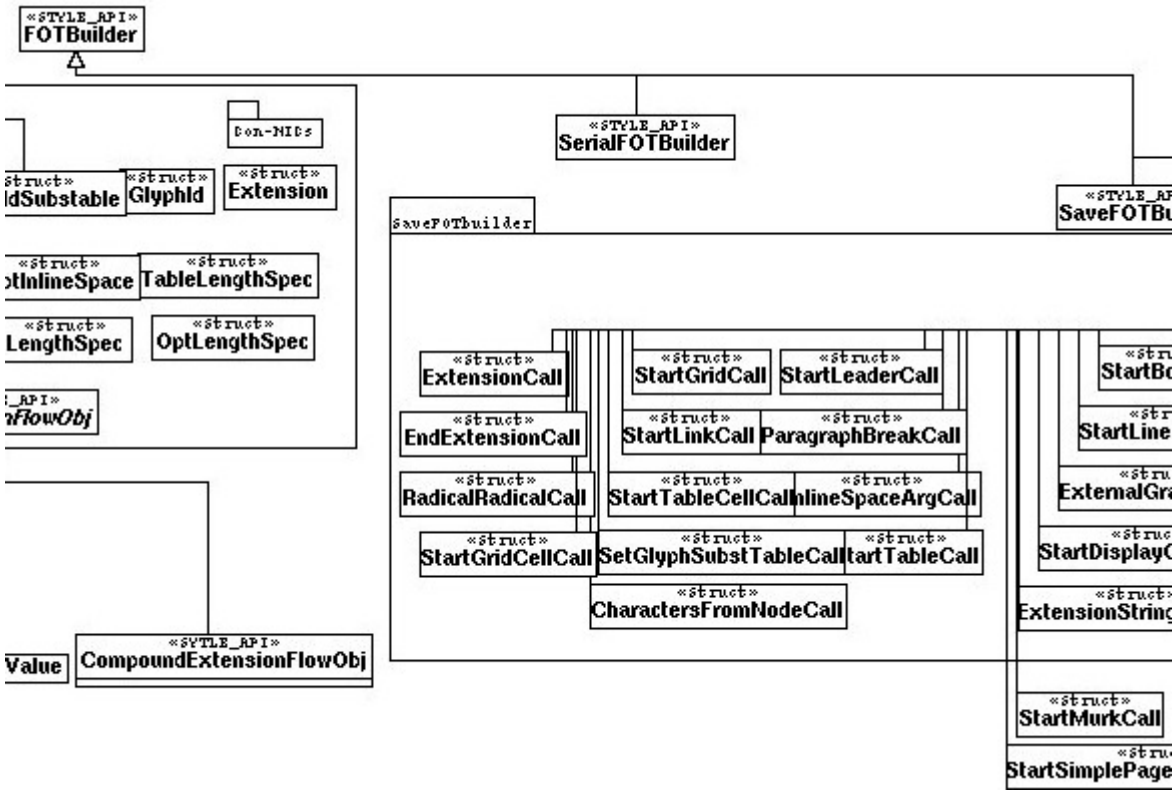
La siguiente clase que se implementó en el proyecto, fue el constructor de arbol de objetos de flujo o FOTBuilder. Esta clase se encarga de la parte final de la construcción del motor de estilo, y a partir de este se construiran los diferentes backends de Openjade.

El primer objetivo que se marco fue, crear el backend fot. Este backend consistia en mostrar en fichero unicamente la composición del arbol de objetos de flujo. Luego se supuso que a partir de esta construcción en los posteriores fortamteos el procedimiento seria similar.

> **Nota:** En el estandar se incluye en la parte final la parte de construir el arbol de objetos de flujo (FOTBuilder), pero no incluye los diferentes backends, ya que esta parte se indica como formatter, pero no se especifica. Crear el fichero .fot o .tex incluidos en este proyecto ya formaría parte del proceso de formateo y por lo tanto fuera del estandar.

El proceso de construir el arbol de objetos de flujo y crear un backend parte de la clase base FOTBuilder, pero intervienen dos clases mas. SaveFOTBuilder y SerialFOTBuilder. Estas dos ultimas son extensiones, heredan de la primera la FOTBuilder.

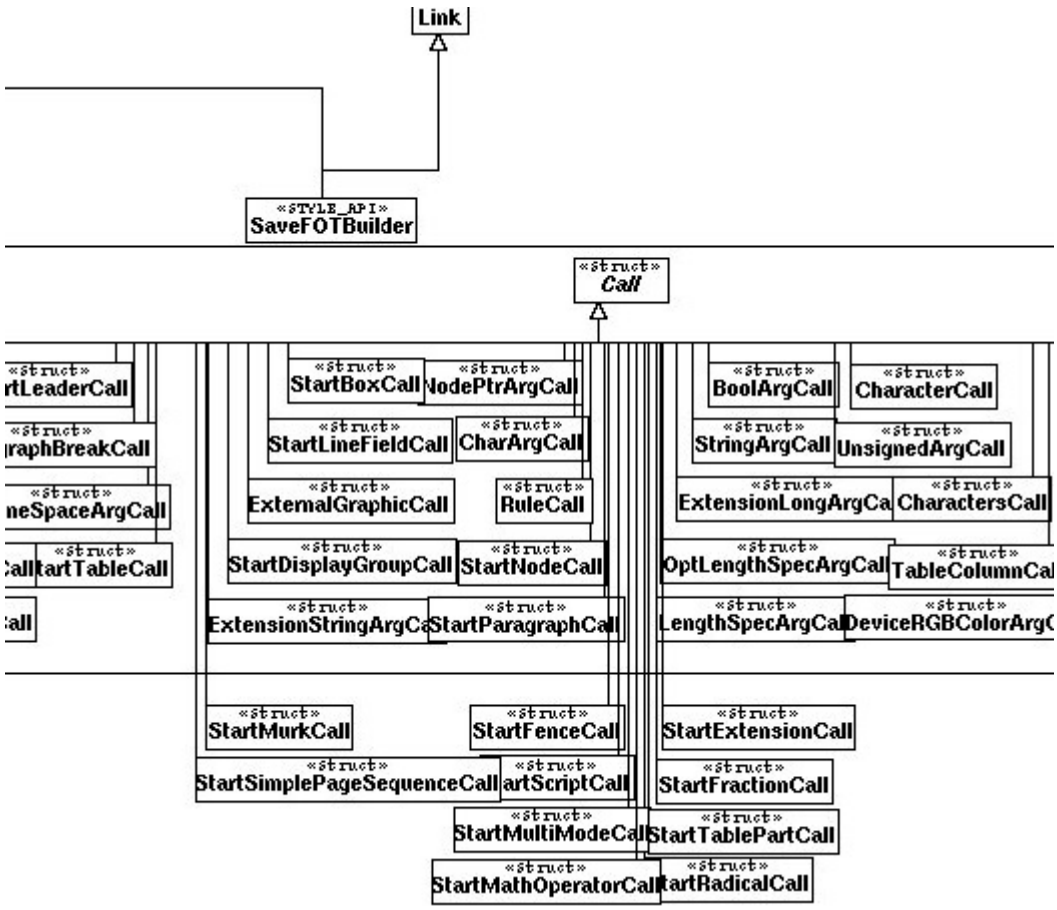
Figura 4-9. FOTBuilder



El proceso es bastante complicado, y complejo en este punto, intervienen muchas clases pero basicamente de lo que se encarga la clase padre es recoger los objetos guardados que han sido interpretados y guardados anteriormente por Interpreter y SchemeParser, utilizando estructuras internas de la propia clase de la forma correspondiente. Asi pues, a continuación la clase SaveFotbuilder, puede guardarlas mediante estructuras que llamadas Call, que estas a su vez seran llamadas desde la otra clase SerialFotbuilder en el proceso de Formateo.

A continuación se puede observar las estructuras internas de la clase SaveFotBuilder antes de implementar la estructura call de StModel.

Figura 4-10. Save/SerialFOTBuilder



La estructura StModel, junta otra llamada StID son estructuras internas publicas de la clase FOTBuilder para guardar información referente al model o de pagina y para nada mas. De hecho se podrían haber creado como objetos pero todo el proceso estaba creado así y se decidio no cambiarlo por no liar aun más el procedimiento y seguir un poco la filosofía de la implementación en todo el proceso de creación del arbol de objetos de flujo.

En StModel es una estructura donde se guardará las características que debe de poseer todo modelo de página, de hecho el pageModel utiliza internamente un atributo de este tipo para guardar la información y StID es una estructura que se usa para identificar información referente a que modelo de pagina con que secuencia de pagina esta asociada.

> **Nota:** En el primer diagrama mostrado del FOTBuilder, se muestra bajo el nombre con-NIC. NIC sale bastantes veces en el código en esta clase, significa características no heredables. Así pues todas las estructuras internas las cuales posean como sufijo NIC significaran que son estructuras donde se guardan las características no heredables de un objeto de flujo.

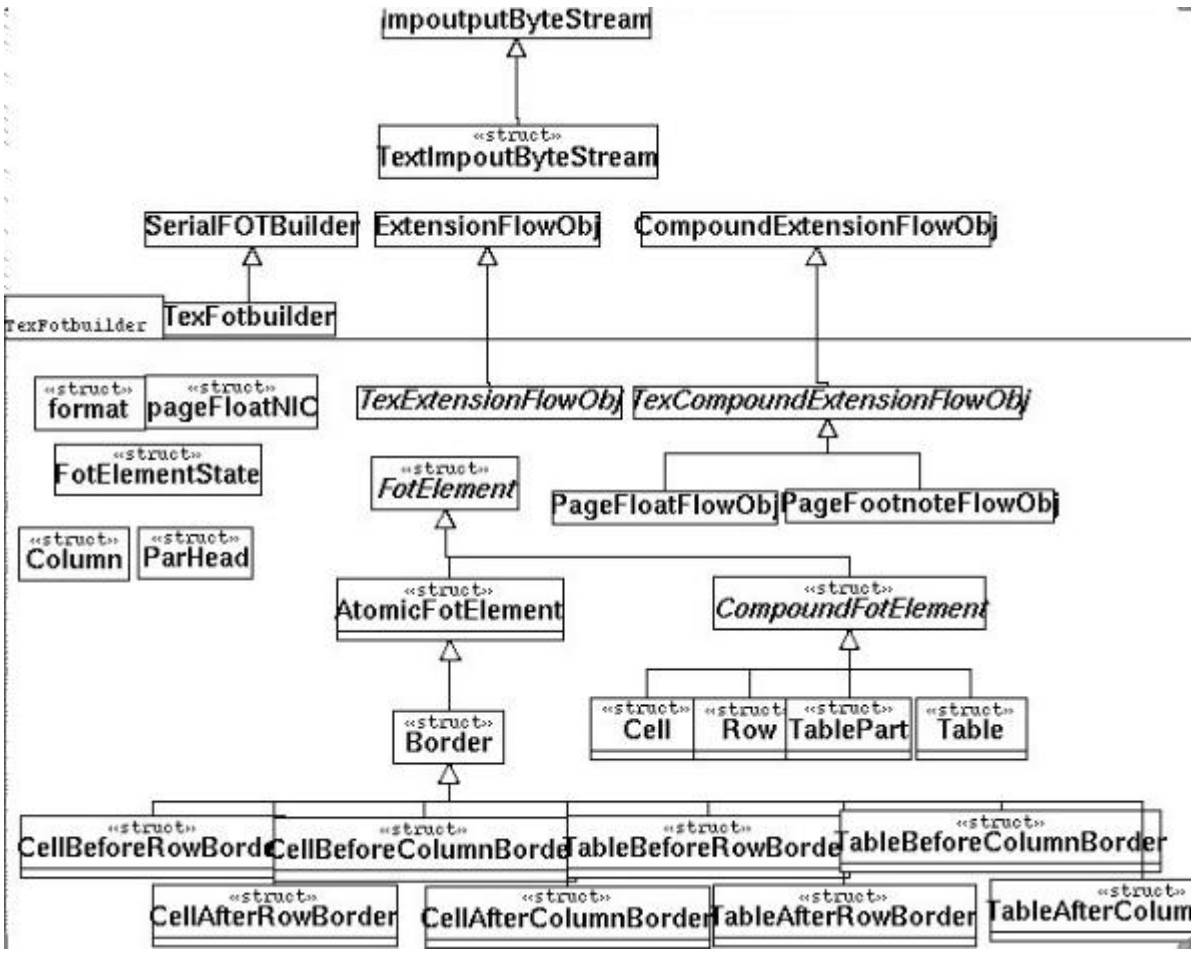
A parte de la implementación de las estrucuturas ya comentadas referentes al modelo de pagina, se tubo que implementar los métodos necesarios para guardar y recuperar la información en estas por cada uno de las características que se podían aplicar a una secuencia de páginas. Estos métodos utilizaban estructuras Call del SaveFOTBuilder, asi pues existe una estructura por cada una de las características.

A continuación se tubieron que propagar los métodos de recogida de información de las estructuras internas en las clases hijas. Esto es porque, en la parte de formateo se utilizarán para recoger la información necesaria para crear el backend correspondiente. Todos los backends se Extienden de la Clase SerialFOTbuilder, ya que esta hace un preproceso de empilar, para dar la vuelta a los objetos de flujo para que a la hora de recogida de los objetos queden en el orden correspondiente.

4.7. Proceso de Formateo

En el proceso de formateo se hcieron varias cosas, primero se construyo el backend .fot. En donde se muestra basicamente en un backend el arbol de objetos de flujo. Y a continuación se pasó a crear el backend .tex, en donde la salida está en formato de documento latex/tex. Se modificaron los ficheros correspondientes que se encuentran en el directorio jade, ya que entramos en la parte de formateo. Todos los constructores para cada tipo de backend se encuentra en este directorio. Aquí como ya se comento en anteriores capitulos se creará el ejecutable Openjade.

Figura 4-11. Backend Tex



La construcción del backend fot fue bastante sencillo, ya que se tenian implementados ya todos los métodos para recoger la información y unicamente se tubo que sacar por la salida, ya preparada, los datos de forma tal como venían. Aquí eso si, se tubo que idear una buena manera, para crear la salida, ya que en ningun sitio y tampoco en el estandar al ser la parte de formateo, se indicaba cual seria la mejor forma para mostrar el modelo de página.

Primero en el fichero .fot se inserta toda la información del modelo de pagina dentro del objeto de flujo pageSequence, pero finalmente se obta, por que todos los modelos de página se insertaran al comienzo del fichero indexados por un identificador para que luego en el tag pageSequence se haga referencia mediante el ID a o a los modelos de página que utilizará cada característica. Por esto la creación de la estructura StID y del atributo ID en el modelo de pagina.

El proyecto finalizó en una primera aproximación de implementación en el backend Latex/Tex. En principio, como se explica en los primeros capítulos de este documento, para crear una salida en pdf o ps, se generaba una salida en Latex/TeX. Pero esta salida en

supuestamente en latex, no era una salida limpia, ya que se construia sobre macros de jadetex, una herramienta externa.

Por esto antes de implementar los métodos para crear la salida en Latex/TeX se tubo que incorporar a la herramienta Openjade, todo lo de JadeTex para que no se tubiera que preprocesar con una herramienta externa. Se incorporaron todas las definiciones que llamaban las macros en el constructor del backend TexFOTBuilder. Y a continuación se iban limpiando a medida que se iban creando. Debido al trabajo que suponía reescribir todas las macros para cada tag, se delimitó unicamente a ir limpiando los tags que fueran siendo necesarios.

A continuación se paso a hacer los metodos para la salida en Tex, a partir de las definiciones que existian y reescribiendolas de forma mas inteligible tanto como fuera posible. Se creo para eso una definicion para secuencia de pagina compleja, similar al de la pagina simple. Y se crearon definiciones para el modelo de pagina pero no se encontraro ninguna que se ajustara a ninguna instrucción existente en Latex

La implementación se hizo mediante strings de latex en los ficheros .cxx. De esta forma la salida era código en Latex/Tex. El proyecto finalizo en este punto debido a que hubo problemas en el formateo, y se perdio mucho tiempo en diferentes busquedas de instrucciones en Latex que fueran compatibles. Estos aspectos se explicarán en el siguiente capítulo, donde se mostrarán resultados y hasta que punto se consiguió finalizar la implementación del backend Latex/TeX.

4.8. Resumen

En este capítulo, se ha descrito de forma muy resumida los aspectos más importantes de la implementación de este proyecto. También se han mostrado las clases más importantes que han intervenido en la realización de este proyecto junto con sus correspondientes diagramas de clases. Se ha explicado de forma más o menos cronológica los pasos que se han seguido y los que se deberían de seguir para añadir nuevas funcionalidades a esta herramienta. De esta forma se podría llegar a construir objetos del mismo tipo que antes no existían ejemplos al respecto, como podría ser, objetos que aun no se han implementando con similares características y definidos en el estandar.

Sólo sirve de guía para futuros desarrollos, aunque se precisa bastante en los temas más puntuales y difíciles que se han ido encontrando. Aclarando muchos conceptos y formas de implementación de esta herramienta. Hay que decir, que el proceso de implementación no ha sido sencillo debido principalmente a la falta de documentación y se ha podido realizar gracias a un profundo estudio de las clases, mediantes diagramas de clases. Aquí sólo se muestran las más importantes para poder contextualizar el problema y ver por una parte la conexión que existe entre las diferentes clases y poder seguir de mejor forma el proceso de implementación.

Además con este capítulo se adquieren conocimientos generales de como se gestiona

la memoria, en que consiste el modelo de pagina, que relación tiene con el objeto de flujo de secuencias de pagina, cómo se implementa, cómo se extrae la información y se parsea las hojas de estilo y cómo se asocia a los tags de un documento sgml. También se entiende de forma general como se construye el árbol de objetos de flujo, de que estructuras está compuesta y para que sirve. Y de aquí el cómo se crean los diferentes backends para las diferentes tipo de salidas que posee la herramienta Openjade.

Capítulo 5. RESULTADOS, JUEGO DE PRUEBAS

En el capítulo anterior se explicó como implementar un modelo de página, como guardar su información y como asociar este modelo de página definido por el usuario en el fichero .dsl con una secuencia de página o tag pageSequence, que se crea en el documento sgml, mediante sus características. También se comento como recoger esta información, guardarla en forma de árbol y a partir de aquí construir los diferentes backends, .fot y .tex.

En este capítulo se mostraran los resultados de estos dos backends, el fichero fot muestra que los objetos de flujo junto con el modelo de página son correctos, en su construcción, y el fichero tex muestra una salida utilizable en Latex/TeX de la implementación. En este fichero tendría que salir un fichero latex correctamente formateado a partir de procesar la hoja de estilo .dsl junto con su correspondiente documento .sgml

Primero de todo se mostrarán y se explicarán los resultados del backend fot

5.1. Backend Fot

Como el propio nombre indica este backend muestra el árbol de objetos de flujo en un backend o fichero .fot. Este fue el primer objetivo que se marcó, debido a que en un principio si se conseguía construir el árbol de objetos de flujo y crear su proceso de formateo, el crear los diferentes backends sería un proceso relativamente sencillo.

En la siguiente imagen se muestra una posible hoja de estilo (fichero .dsl) para un documento .sgml:

Figura 5-1. Fichero DSSSL

```
<!doctype style-sheet PUBLIC "-//James Clark//DTD DSSSL Style Sheet//EN">

<style-sheet>
<style-specification>
<style-specification-body>

(define-page-model *cap_one*
 (width 21)
 (height 54)
 (filling-direction 1))

(define-page-model *cap_two*
 (width 12)
 (height 42)
 (filling-direction 2))

(define-page-model *cap_end*
 (width 12)
 (height 57)
 (filling-direction 3))

(element prova
 (make page-sequence
   force-last-page: 'front
   force-first-page: 'back
   justify-spread?: #t
   binding-edge: 'right
   blank-back-page-model: *cap_end*
   repeat-page-models: '(*cap_end* *cap_one* *cap_two*)
   initial-page-model: '(*cap_end* *cap_one*)
   blank-front-page-model: *cap_end*
   (make included-container-area
    span: 2)))

</style-specification-body>
</style-specification>
</style-sheet>
```

En esta hoja de estilo se puede observar diferentes partes. En esta imagen se representa una hoja de estilo con tres modelos de páginas distintas relativamente sencillas. La primera parte es la especificación y se le indica que tipo de DTD se va a utilizar, que tipo de hoja de estilo. En la segunda parte se definen los tres modelos de página con tamaño altura y dirección de texto. Esto fue una prueba inicial que se hizo, bastante básica al principio de la implementación en el modelo de página.

En la tercera parte, se ve como se asocia el tag del documento sgml con una secuencia de página. En esta se ve los diferentes modelos de páginas que puede tener. Tiene alguna característica donde unicamente se construirá con un modelo de página, y de otras que se construira a partir de una lista de modelos de página.

A continuación se procesa un documento sgml cualquiera que contenga el tag page-

sequence y se procesa junto con este fichero de estilo, con la opcion -t fot.Dando como resultado el siguient fichero .fot:

Figura 5-2. Salida Fot -1

```
<?xml version="1.0"?>
<fot>
<define-page-model id=cap_one width="21" height="54" filling-direction="false">
<region x-origin="0" y-origin="0" width="0" height="0" filling-direction="false">
<header height="0" width="0" filling-direction="false" contents-alignment="false"/>
<footer height="0" width="0" filling-direction="false" contents-alignment="false"/>
</region>
</define-page-model>
<define-page-model id=cap_end width="12" height="57" filling-direction="false">
<region x-origin="0" y-origin="0" width="0" height="0" filling-direction="false">
<header height="0" width="0" filling-direction="false" contents-alignment="false"/>
<footer height="0" width="0" filling-direction="false" contents-alignment="false"/>
</region>
</define-page-model>
<page-sequence force-last-page="front" force-first-page="back" jystify-spread="true"
blank-back-page-model="cap_one" initial-page-model="cap_one cap_end cap_end" repeat-
<included-container-area scale="max-uniform" span="2">
<a name="0"/>
<a name="1"/>
<text>Hola aixo es una prova.</text>
</included-container-area>
</page-sequence>
</fot>
```

En la siguiente figura, se muestra otra salida de un fichero .fot pero algo más compleja, es sacada de un documento .sgml bastante extenso:

Figura 5-3. Salida Fot -2

```
<?xml version="1.0"?>
<fot>
<sequence font-size="12pt" font-weight="medium" font-posture="upright" font-family-name="Ti
writing-mode="left-to-right" left-margin="72pt" right-margin="72pt" top-margin="72pt" botto
footer-margin="48pt" page-width="612pt" page-height="792pt" quadding="start">
<page-sequence start-indent="0pt" input-whitespace-treatment="collapse" left-margin="56.692
top-margin="56.692pt">
<sequence>
<paragraph space-before="54pt" space-after="12pt">
<sequence>
<a name="0"/>
<a name="1"/>
<a name="2"/>
<line-field field-width="54pt" font-family-name="Arial">
<text>Date:</text>
</line-field>
<text>January 26, 1998</text>
</sequence>
</paragraph>
<paragraph>
<sequence>
<a name="3"/>
<line-field field-width="54pt" font-family-name="Arial">
<text>To:</text>
</line-field>
<text>The Honorable William J. Clinton President of the United States Washington, DC</text>
</sequence>
</paragraph>
<paragraph space-after="24pt">
<sequence>
<a name="4"/>
<line-field field-width="54pt" font-family-name="Arial">
<text>Subject:</text>
</line-field>
<text>Project for the new American Century</text>
</sequence>
</paragraph>
<paragraph space-after="12pt">
<text>Dear </text>
<text>The Honorable William J. Clinton President of the United States Washington, DC</text>
<text>:</text>
</paragraph>
</sequence>
<paragraph space-after="6pt">
<a name="5"/>
<a name="6"/>
<text>Dear Mr. President:</text>
</paragraph>
```

Si este mismo fichero se intenta procesar con un openjade de distribución se observará que da problemas en la creación del fichero y no se construira, o no se construirá correctamente. En la segunda figura, se observa el contenido final del fichero con los objetos de flujo y sus características correspondientes. Es una buena de debugar si el proceso

ha estado correcto o no, ya que se muestra tal cual con sus características. Por eso se hizo primero este backend para luego poder pasar al siguiente.

En este fichero se observan claramente dos partes. En la primera parte se construye el modelo de página con sus correspondientes valores indexados por un identificador, y en la segunda parte aparecen los objetos de flujo o tags del documento sgml con sus características y textos correspondiente. En el caso de las características de la secuencia de página, page-sequence, se ve como hacen referencia a los modelos de página definidos en la primera parte.

5.2. Backend Latex/TeX

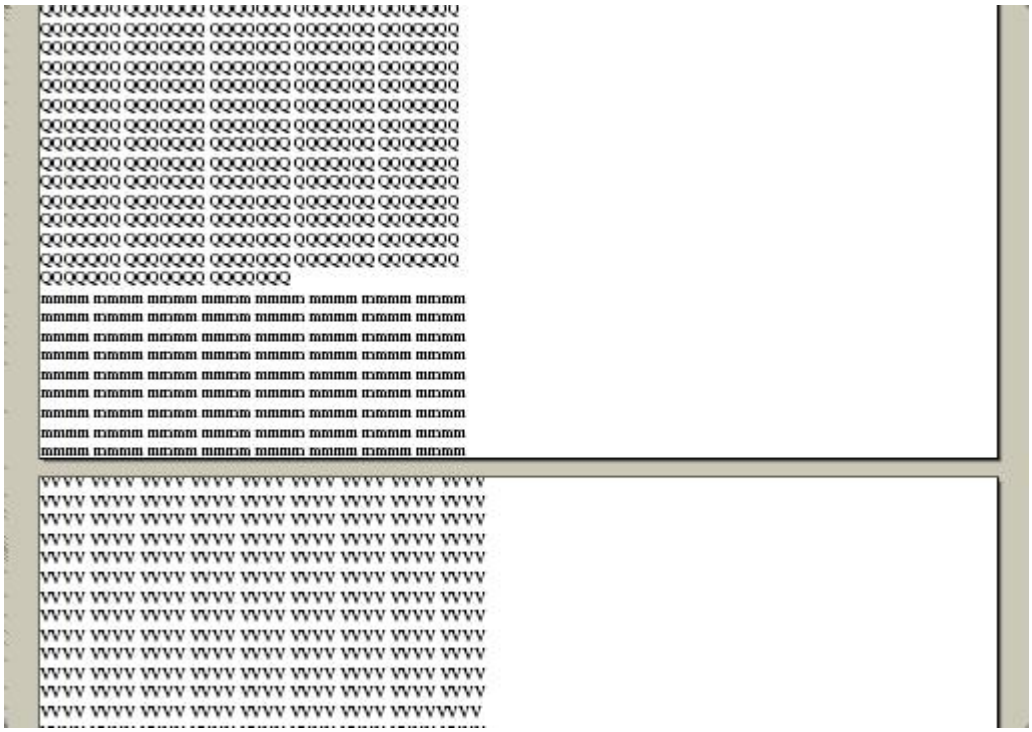
Como el propio nombre indica este es el backend para construir un fichero latex/tex a partir de un documento .sgml y una hoja de estilo .dsl. Primero de todo,como ya se comentó en el anterior capítulo, se pasó a incorporar el programa jadetex dentro de Openjade para no tener que utilizar una herramienta externa para poder traducir las macros que se creaban al generar un backend del tipo tex. Luego pasarlas a tex y de alli otro backend.

Se consiguió también hacer independiente a openjade de jadetex, a la hora de crear ficheros tipo .tex, pero no se consiguió limpiar todas las macros de los tags para que el código fuera mas legible y de esta forma poder editarlo con Latex/TeX. Se consiguió sin embargo, limpiar algunas definiciones tanto en cuanto se iban utilizando los tags. Cada tag corresponde a una macros la cual definida en jadetex.

Después de las pruebas de creación del backend fot y de la implementación de los métodos en el constructor del backend TeX, y de la incorporación de la herramienta de jadetex en el constructor de la clase, para cargar las macros, se pasó a crear la definición para la secuencia de página, siguiendo la base de la secuencia de paginas simples, ya implementada en este backend.

Una vez construido la secuencia de paginas, y validado de que funcionaba se pasó a incorporar un modelo de página. Para esto se tubo que cambiar la manera de cargar la secuencia de página, cosas no necesarias que estaban copiadas de las páginas simples, como marges y cabeceras, ya que estas definiciones estarían dentro del modelo de página en el caso de que tubieran. Por esto el area en donde se podían insertar los modelos de página o construir las diferentes regiones, eran todo el tamaño de la página sin necesidad de margenes. En la siguiente figura se muestra un ejemplo de secuencia de paginas, tocando el margen derecho y elimando tanto las cabeceras como pie de página:

Figura 5-4. Backend Tex prueba1

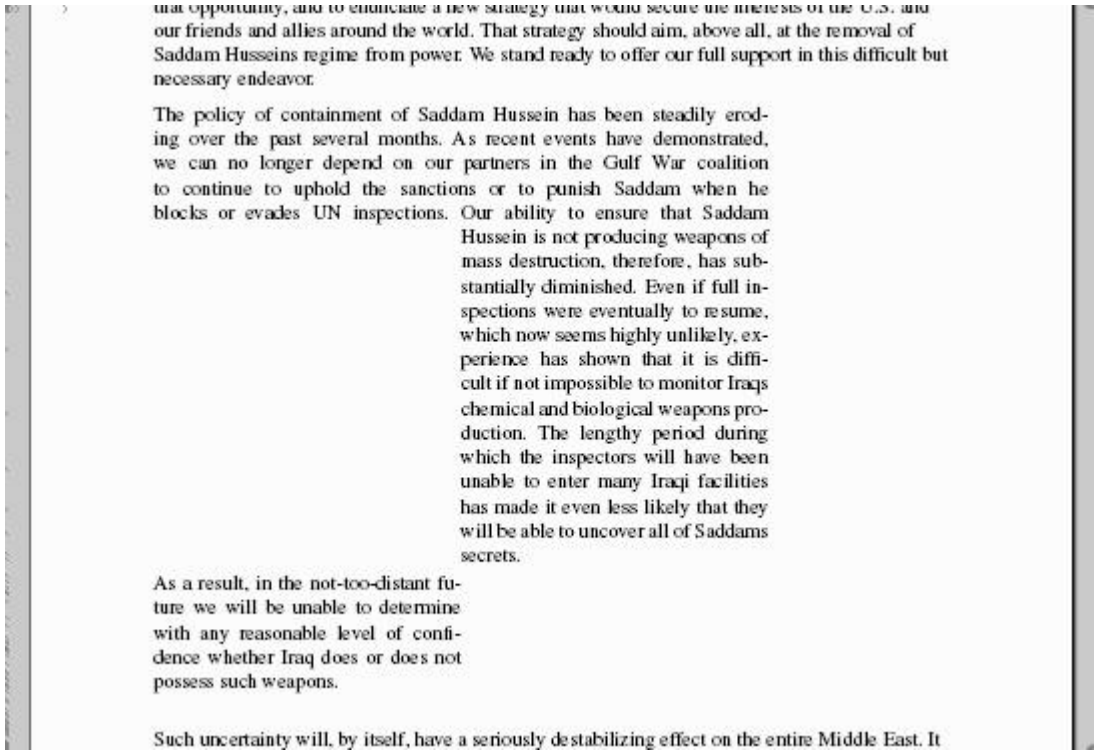


Hasta este punto, el proceso de formateo fue bien y bastante rápido, se tenía un backend (fot) construido, una definición en latex para la secuencia de páginas y los métodos en el constructor del backend TeX implementados para obtener la información tanto de los modelos de páginas como de las características que poseia los objetos de flujo,y como los objetos en sí que participarían en el proceso.

El problema surgió a la hora de construir un modelo de página y ya por no decir, constuir dos modelos de páginas no tanto dentro de una página sino que estos modelos de páginas estubieran en tantas páginas como en las características así se indicara. Como ya se dijo, la idea del modelo de página es que además de poder crear regiones dentro de una página, estas se puedan repetir en todas o algunas páginas del documento. Basicamente este fue el gran problema que se encontró en la creación del backend Latex/TeX

En un principio se intentó crear diferentes regiones dentro de una misma página con la instrucción minipage ya que se podían hacer pequeñas regiones e incluso dentro de esta, poner cabecera y pie de pagina. Por lo que se acercaba bastante a lo que se estaba buscando en cuanto a la idea. Se hicieron diferentes pruebas cambiando los parámetros pero el resultado no fue nada satisfactorio:

Figura 5-5. Backend TeX prueba 2



that opportunity, and to enunciate a new strategy that would secure the interests of the U.S. and our friends and allies around the world. That strategy should aim, above all, at the removal of Saddam Husseins regime from power. We stand ready to offer our full support in this difficult but necessary endeavor.

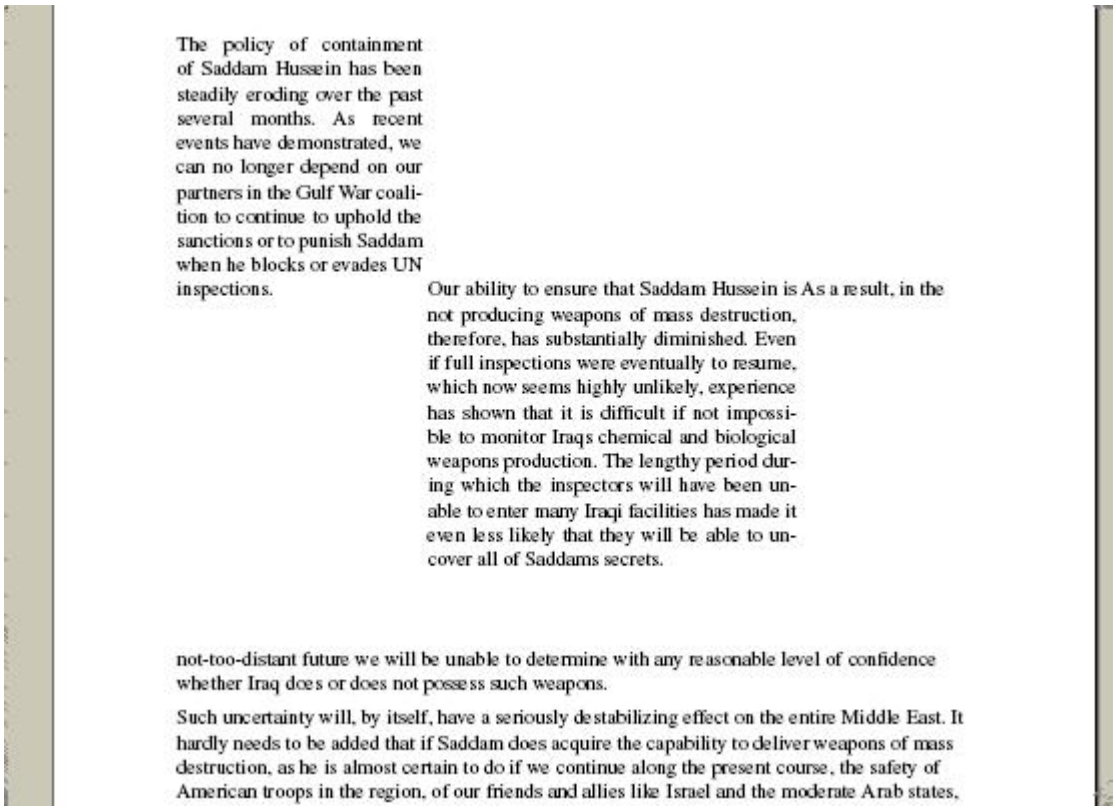
The policy of containment of Saddam Hussein has been steadily eroding over the past several months. As recent events have demonstrated, we can no longer depend on our partners in the Gulf War coalition to continue to uphold the sanctions or to punish Saddam when he blocks or evades UN inspections. Our ability to ensure that Saddam Hussein is not producing weapons of mass destruction, therefore, has substantially diminished. Even if full inspections were eventually to resume, which now seems highly unlikely, experience has shown that it is difficult if not impossible to monitor Iraqs chemical and biological weapons production. The lengthy period during which the inspectors will have been unable to enter many Iraqi facilities has made it even less likely that they will be able to uncover all of Saddams secrets.

As a result, in the not-too-distant future we will be unable to determine with any reasonable level of confidence whether Iraq does or does not possess such weapons.

Such uncertainty will, by itself, have a seriously destabilizing effect on the entire Middle East. It

Se puede observar, que se intentan crear tres regiones diferentes en el mismo documento pero que, el minipage, continua a continuación de la última región. Esto supone que no se pueda crear una independencia entre las regiones. Lo mismo pasa si se cambian los parámetros del minipage. Cualquier combinación con los parametros sucede algo similar, no se puede crear una independencia, a parte, crearía limitaciones, debido a que las regiones van una a continuación de la otra, no pudiendose solapar y mostrando una falta de flexibilidad con muchas regiones en el mismo nivel. En el estandar en ningún sitio se dice que no se pueda solapar, y se han de fijar mediante coordenadas fijadas x,y.

Figura 5-6. Backend TeX prueba 3


The policy of containment of Saddam Hussein has been steadily eroding over the past several months. As recent events have demonstrated, we can no longer depend on our partners in the Gulf War coalition to continue to uphold the sanctions or to punish Saddam when he blocks or evades UN inspections.

Our ability to ensure that Saddam Hussein is not producing weapons of mass destruction, therefore, has substantially diminished. Even if full inspections were eventually to resume, which now seems highly unlikely, experience has shown that it is difficult if not impossible to monitor Iraqs chemical and biological weapons production. The lengthy period during which the inspectors will have been unable to enter many Iraqi facilities has made it even less likely that they will be able to uncover all of Saddams secrets.

As a result, in the not-too-distant future we will be unable to determine with any reasonable level of confidence whether Iraq does or does not possess such weapons.

Such uncertainty will, by itself, have a seriously destabilizing effect on the entire Middle East. It hardly needs to be added that if Saddam does acquire the capability to deliver weapons of mass destruction, as he is almost certain to do if we continue along the present course, the safety of American troops in the region, of our friends and allies like Israel and the moderate Arab states,

El comando minipage a parte de su falta de flexibilidad para crear regiones y las limitaciones que pueda tener a la hora de fijar coordenadas, tiene un problema de base, que no salta de página o entre regiones. No hay forma de que una minipage sea lo suficientemente inteligente para que calcule el tamaño de las regiones y si sobrepasa la región de final de página, salte a la siguiente página. Cualquier cosa que se construya en un minipage debe estar dentro de esa pagina ya que si no se produce un desbordamiento de página. En estos casos se observa como las letras salen de la página.

Podría surgir la idea de hacer las regiones con tablas, se descartó esa idea desde un principio, debido a que estamos hablando un flujo de objetos continuado, y las regiones de una página deben estar comunicadas con las regiones de la otra página, etc. Es un flujo continuo de texto, en donde a priori no se sabe cuanto va a ocupar. No se sabe porque se procesa automáticamente con la herramienta Openjade a partir de un documento escrito en .sgml independiente. Además dentro de una región puede existir cabecera , objetos, tablas, pie de página, etc. En las tablas, estos problemas no se puede resolver.

Así que había un problema básico, poder saltar de página automáticamente cuando se acabara la página, o la ultima región de esa página, para continuar en la siguiente región de la siguiente página teniendo en cuenta que el proceso debía ser automatico, independiente, transparente y continuo en todas las páginas que tubieran ese modelo de

página definido.

Finalmente, se optó por calcular mediante variables en Latex/Tex el tamaño de texto, que se escribía, ya que se conocía el tamaño de las regiones a priori. De esta forma él, automaticamente, mediante instrucciónes de Latex podría cambiar a la siguiente región de la siguiente página. Como idea mejoraba las expectativas de la limitación del minipage en este aspecto.

> **Nota:** No es posible calcular el volumen de texto o la altura aun se conozca anchura y altura, debido a que el texto es dinámico y segun el estilo con el cual se formatee, y tipo de lestras etc, puede variar considerablemente, por lo tanto la única forma que se conoce para calcular el volumen, es una vez ya formateado. Para esto, se le tiene que dejar la responsabilidad a Latex/Tex

Así pues se paso a implementar por salida de gcc los strings de latex para construir esta idea. Se hizo mediante cajas. De hecho, el funcionamiento de la gran parte de las macros en latex funcionan creando cajas o box. En la siguiente figura se muestra una caja con bordes, para que se viera bien la idea del funcionamiento.

Figura 5-7. Backend TeX - prueba 4

Esta caja, representa una región que se podría fijar mediante coordenadas. El número que muestra en la parte superior de la caja es la altura, como se puede observa a priori, antes de construir la caja. Esto es posible ya que latex primero calcula cómo quedaría el texto en latex respecto a la página y a continuación pasa a realizar el formateo de esta.

De hecho, se consiguió ir saltando de páginas con el método de cálculo de cajas. Y de esta forma se consiguió salvar el primer obstáculo en la construcción automática de texto en las areas. Cada vez que en el comando de latex sobrepasaba el area indicada que venía del constructor de backend Tex, latex lo sabría y pasaría a construir una región en la siguiente página.

El método del cálculo de cajas, finalmente, quedó aparcado. Se observó que en sitios donde había tags sencillos era razonable construir el método del calculo de cajas. Pero en cuanto se complicaba mucho, y esto sucede en la mayoría de casos, no era posible.

Esto es debido a un problema que no se había tenido en cuenta. Los tags. En Openjade, como ya se indicó, cada tag representa un nuevo objeto de flujo, y este tendrá ciertas propiedades. Y como está en forma de árbol, puede tener uno o más hijos y así N veces,

incluso ocupar todo el documento. Supongamos pues, que en la caja que se tiene que calcular la altura aparece un tag donde no sólo hay flujo de texto sinó que también aparecen sub-tags que estos a su vez tienen más tags que finalmente incluyen texto. Podría perfectamente ocupar varias páginas. Y no se puede a calcular únicamente el texto final, ya que hay herencia de características y el estilo puede variar bastante.

Si varía bastante el cálculo ya es erroneo y puede haber overflow y las letras sobresalir de la página en cuestión. Además si se inserta la caja entre medias, se rompe el estilo.

Después de mucho buscar, se llegó a la conclusión que con Latex no se podía crear de forma automática un modelo de páginas para diferentes páginas. Existían diferentes problemas no flaqueables mediante las macros de Latex. Latex no estaba preparado para este tipo de utilización automática, no era un lenguaje diseñado para enmaquetado de páginas o al menos en el mismo flujo de texto.

Se conoce que existen diferentes macros de Latex que hacen cambiar de página, como puede ser cuando el texto no cabe debido a un pie de página o en el comando longtable, si la tabla pasa de página salta y continua en la siguiente de forma automática.

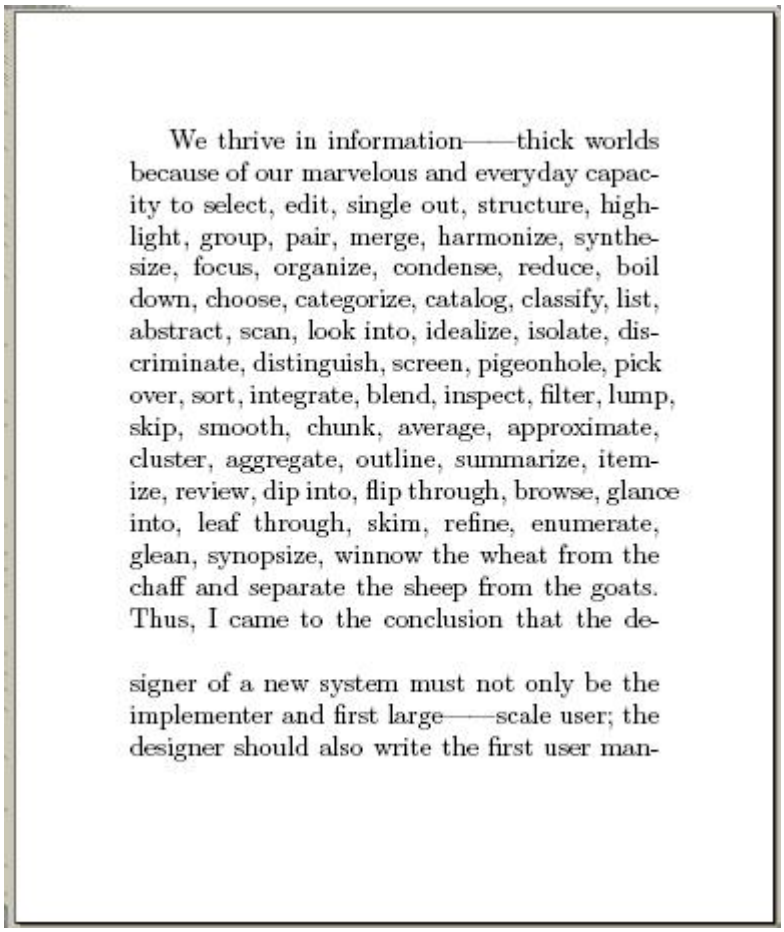
Pero esto se realiza a nivel de TeX. Por lo que se llegó a la conclusión que sí era posible formatear las regiones en TeX para simular este comportamiento, pero no con Latex. Las instrucciones en Latex no eran suficientemente robustas para hacer este tipo de procesamiento.

Así pues, la idea sería programar a bajo nivel, en TeX y crear una macro para el modelo de páginas. Pero esto suponía aprender TeX y buscar como se podría hacer mediante un estudio de las librerías creadas, etc.

> **Nota:** Buscando, se descubrió también, un lenguaje de programación creado sobre TeX, en donde se da mucha más flexibilidad y seguramente se podría crear facilmente algo parecido a lo que se estaba buscando. Fue una opción también aparcada en un principio ya que no sería una salida límpia en Latex/Tex, sino que ya se estaría creando una salida dependiente de otra herramienta externa, lo mismo que sucedía con Jadetex. Aunque podría ser otra posible alternativa. La lógica, de todas formas, nos indicaba que lo mejor sería crear un nuevo comando en Latex, creado desde abajo, en Tex.

Por falta de tiempo se obtó por finalizar el proyecto en este punto, pero en un último acercamiento, se pensó en separar el flujo de la salida de Openjade. Se pensó de crear las páginas del tamaño de una caja, de esta forma no habría problemas para crear todo el flujo de texto a lo largo de todas las cajas. En la siguiente figura se muestra esta idea:

Figura 5-8. Backend TeX prueba 5



We thrive in information——thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats. Thus, I came to the conclusion that the designer of a new system must not only be the implementer and first large——scale user; the designer should also write the first user man-

El tamaño de la página no se puede variar a lo largo de todas las páginas escritas, por lo que estas representarían la misma región por página. Por cada nuevo modelo de página se tendría que crear una nueva salida. Cada página, pues, representa una nueva región.

Luego estas cajas ya procesadas anteriormente mediante ficheros que podrían ser temporales se insertarían en una de base. En este caso si que es posible, porque LateX ya sabe de antemano las dimensiones de las minipáginas a la hora de insertarlas. La idea es que TeX las procesa como imagenes.

Esto también tiene limitaciones. El principal problema es que no se puede hacer texto continuo entre regiones de la misma página, sino que el texto es continuo en las regiones del mismo tipo de las diferentes páginas. Unicamente sería continuo si el modelo de página fuera identico para todas las regiones de la página, como se muestra en la siguiente figura:

Figura 5-9. Backend Tex prueba 6

5.3. Resumen

En este capítulo, en una primera sección, se ha descrito el éxito de la construcción del backend fot, con ejemplos que lo corraboran, que se traduciría en el éxito en todo lo referente a la construcción de una nueva funcionalidad del modelo de página y su construcción final en un backend correctamente formateado.

En la segunda parte de este capítulo, muestra los problemas que fueron apareciendo en un intento por construir un backend en Latex/TeX, sus diferentes salidas en el formateo y las posibles alternativas que se fueron generando de forma cronológica. Basicamente, el

problema de base fue que se pensó desde el principio que con comandos o macros de Latex sería suficiente para solucionar cualquier dificultad en la construcción del backend. Luego se vió que se tendría que haber planteado desde un principio como una nueva macros reescrita en TeX, ya que no existe ninguna lo suficientemente eficaz para un procesamiento de texto de flujo continuo de forma automatica.

Latex no esta preparado para este tipo de contextos, y de ahí el problema. En toda la segunda sección de este capítulo se muestra con ejemplos el porqué de esta situación. La idea es que en un futuro se pueda construir una macros que se ajuste a la salida de Openjade para el modelo de página. Finalmente se construyó un prototipo, tampoco del todo eficaz; ya que o las regiones eran del mismo tamaño, cosa que no tiene mucha utilidad para la comunidad, o el flujo de texto saltaría entre las regiones del mismo tipo pero de diferentes páginas. Esto hecho tendría mucho más utilidad en el formateo de algún tipo de documentación

Se barajaron pues, diferentes alternativas algunas apartadas y algunas en las que se podrían estudiar las ideas, aunque una vez visto el problema en su totalidad, la idea más consistente sería hacer una nueva macros de Latex, construida desde la base, desde TeX.

Capítulo 6. CONCLUSIÓN

En este capítulo se explican las diferentes tareas que se podría dar para mejorar la herramienta Openjade así como las conclusiones y reflexiones más importantes a raíz de la realización del proyecto.

A partir de lo visto en la creación del proyecto han ido surgiendo posibles tareas, trabajos o incluso nuevos proyectos para realizar en la herramienta Openjade. La primera al ver los resultados, de este proyecto, sería conseguir crear una macros implementada en TeX, para el modelo de página. Luego implantar esto en el código dejado del proyecto sería muy sencillo y fácil de aplicar. La idea de construir esta macros sería o bien centrarla específicamente al modelo de pagina de Openjade o quizás mejor, crear una macros en Latex, que cuando detectara que la caja finaliza te lo indicara y así poder saltar a la siguiente página o región.

Otra posible tarea sería acabar de limpiar todas las macros que se incorporarón de jadetex a Openjade, para que el código de Latex fuera más legible y por lo tanto más editable.

Luego existen otras tareas encontradas a raíz de realizar el proyecto, como crear el objeto para las columnas complejas, que sería tan fácil como seguir este documento y la implementación del modelo de página para su implemntación. Corregir posibles ineficiencias presentadas en objetos ya creados en versiones anterires, posiblemente debido a la mala configuración de las macros del proyecto JadeTex o de la implementación del código en gcc de Openjade.

Existe, un problema ya comentado sobre el parseo en la construcción de las hojas de estilo donde no existe un control de los puertos en los objetos. Es decir, hay objetos de flujo en donde no se controla que objetos de flujos pueden o no pueden insertarse bajo ellos. Por ejemplo, se puede crear una secuencia de páginas debajo de una secuencia.

Para la comunidad de código libre sería de mucha utilidad conseguir un backend en TeX utilizable, ya que, pese a sus obvias limitaciones, esta herramienta es bastante utilizada para documentación bien formada y viene en todas las distribuciones de linux.

El trabajo de busqueda, estudio y creación, ya se hizó y expertos en TeX hay muchos pero en OpenJade en cambio no hay. Por lo que se puede decir, que lo dificil ya está construido y faltaría dar el último empujón al asunto. Hay que decir que el proyecto estaba parado desde hacía mucho tiempo, pero aun así, la comunidad está pendiente de la nueva versión por lo que seguro que tendrá una buena acogida, y quien sabe, quizá será lo necesario para volver a impulsar el uso de sgml, es decir del estandar.

Apéndice A. CÓDIGO DE LAS PARTES MÁS IMPORTANTES Y FICHERO CONFIGURACIÓN DE DOXYGEN

Aquí se mostrará las partes más importantes en la implementación del código de las diferentes clases. A parte también se dará el fichero de configuración para generar la documentación escrita en doxygen para esta herramienta.

A continuación se muestra el fichero de configuración creado en OpenJade para generar la documentación en Doxygen:

```
# Doxyfile 1.3.8

#---------------------------------------------------------------------------
# Project related configuration options
#---------------------------------------------------------------------------
PROJECT_NAME           = "openjadedox"
PROJECT_NUMBER         = 0.2
OUTPUT_DIRECTORY       = dox/
CREATE_SUBDIRS         = NO
OUTPUT_LANGUAGE        = English
USE_WINDOWS_ENCODING   = NO
BRIEF_MEMBER_DESC      = YES
REPEAT_BRIEF           = YES
ABBREVIATE_BRIEF       = "The $name class" \
                         "The $name widget" \
                         "The $name file" \
                         is \
                         provides \
                         specifies \
                         contains \
                         represents \
                         a \
                         an \
                         the
ALWAYS_DETAILED_SEC    = NO
INLINE_INHERITED_MEMB  = NO
FULL_PATH_NAMES        = YES
STRIP_FROM_PATH        = ./
STRIP_FROM_INC_PATH    =
SHORT_NAMES            = NO
JAVADOC_AUTOBRIEF      = NO
MULTILINE_CPP_IS_BRIEF = NO
DETAILS_AT_TOP         = NO
INHERIT_DOCS           = YES
```

```
DISTRIBUTE_GROUP_DOC   = NO
TAB_SIZE               = 8
ALIASES                =
OPTIMIZE_OUTPUT_FOR_C  = NO
OPTIMIZE_OUTPUT_JAVA   = NO
SUBGROUPING            = YES
#---------------------------------------------------------------------------
# Build related configuration options
#---------------------------------------------------------------------------
EXTRACT_ALL            = YES
EXTRACT_PRIVATE        = YES
EXTRACT_STATIC         = YES
EXTRACT_LOCAL_CLASSES  = YES
EXTRACT_LOCAL_METHODS  = YES
HIDE_UNDOC_MEMBERS     = NO
HIDE_UNDOC_CLASSES     = NO
HIDE_FRIEND_COMPOUNDS  = NO
HIDE_IN_BODY_DOCS      = NO
INTERNAL_DOCS          = YES
CASE_SENSE_NAMES       = YES
HIDE_SCOPE_NAMES       = NO
SHOW_INCLUDE_FILES     = YES
INLINE_INFO            = YES
SORT_MEMBER_DOCS       = YES
SORT_BRIEF_DOCS        = NO
SORT_BY_SCOPE_NAME     = NO
GENERATE_TODOLIST      = YES
GENERATE_TESTLIST      = YES
GENERATE_BUGLIST       = YES
GENERATE_DEPRECATEDLIST= YES
ENABLED_SECTIONS       =
MAX_INITIALIZER_LINES  = 30
SHOW_USED_FILES        = YES
#---------------------------------------------------------------------------
# configuration options related to warning and progress messages
#---------------------------------------------------------------------------
QUIET                  = NO
WARNINGS               = NO
WARN_IF_UNDOCUMENTED   = YES
WARN_IF_DOC_ERROR      = YES
WARN_FORMAT            = "$file:$line: $text"
WARN_LOGFILE           =
#---------------------------------------------------------------------------
# configuration options related to the input files
#---------------------------------------------------------------------------
INPUT                  = ./
FILE_PATTERNS          = *.c \
                         *.cc \
                         *.cxx \
```

```
                         *.cpp \
                         *.c++ \
                         *.java \
                         *.ii \
                         *.ixx \
                         *.ipp \
                         *.i++ \
                         *.inl \
                         *.h \
                         *.hh \
                         *.hxx \
                         *.hpp \
                         *.h++ \
                         *.idl \
                         *.odl \
                         *.cs \
                         *.php \
                         *.php3 \
                         *.inc \
                         *.m \
                         *.mm \
                         *.C \
                         *.CC \
                         *.C++ \
                         *.II \
                         *.I++ \
                         *.H \
                         *.HH \
                         *.H++ \
                         *.CS \
                         *.PHP \
                         *.PHP3 \
                         *.M \
                         *.MM
RECURSIVE              = YES
EXCLUDE                =
EXCLUDE_SYMLINKS       = NO
EXCLUDE_PATTERNS       =
EXAMPLE_PATH           =
EXAMPLE_PATTERNS       = *
EXAMPLE_RECURSIVE      = NO
IMAGE_PATH             =
INPUT_FILTER           =
FILTER_PATTERNS        =
FILTER_SOURCE_FILES    = NO
#---------------------------------------------------------------------------
# configuration options related to source browsing
#---------------------------------------------------------------------------
SOURCE_BROWSER         = NO
```

```
INLINE_SOURCES         = NO
STRIP_CODE_COMMENTS    = YES
REFERENCED_BY_RELATION = NO
REFERENCES_RELATION    = NO
VERBATIM_HEADERS       = NO
#---------------------------------------------------------------------------
# configuration options related to the alphabetical class index
#---------------------------------------------------------------------------
ALPHABETICAL_INDEX     = NO
COLS_IN_ALPHA_INDEX    = 5
IGNORE_PREFIX          =
#---------------------------------------------------------------------------
# configuration options related to the HTML output
#---------------------------------------------------------------------------
GENERATE_HTML          = YES
HTML_OUTPUT            = html
HTML_FILE_EXTENSION    = .html
HTML_HEADER            =
HTML_FOOTER            =
HTML_STYLESHEET        =
HTML_ALIGN_MEMBERS     = YES
GENERATE_HTMLHELP      = NO
CHM_FILE               =
HHC_LOCATION           =
GENERATE_CHI           = NO
BINARY_TOC             = NO
TOC_EXPAND             = NO
DISABLE_INDEX          = NO
ENUM_VALUES_PER_LINE   = 4
GENERATE_TREEVIEW      = NO
TREEVIEW_WIDTH         = 250
#---------------------------------------------------------------------------
# configuration options related to the LaTeX output
#---------------------------------------------------------------------------
GENERATE_LATEX         = YES
LATEX_OUTPUT           = latex
LATEX_CMD_NAME         = latex
MAKEINDEX_CMD_NAME     = makeindex
COMPACT_LATEX          = NO
PAPER_TYPE             = a4wide
EXTRA_PACKAGES         =
LATEX_HEADER           =
PDF_HYPERLINKS         = NO
USE_PDFLATEX           = NO
LATEX_BATCHMODE        = NO
LATEX_HIDE_INDICES     = NO
#---------------------------------------------------------------------------
# configuration options related to the RTF output
#---------------------------------------------------------------------------
```

```
GENERATE_RTF           = NO
RTF_OUTPUT             = rtf
COMPACT_RTF            = NO
RTF_HYPERLINKS         = NO
RTF_STYLESHEET_FILE    =
RTF_EXTENSIONS_FILE    =
#---------------------------------------------------------------------------
# configuration options related to the man page output
#---------------------------------------------------------------------------
GENERATE_MAN           = NO
MAN_OUTPUT             = man
MAN_EXTENSION          = .3
MAN_LINKS              = NO
#---------------------------------------------------------------------------
# configuration options related to the XML output
#---------------------------------------------------------------------------
GENERATE_XML           = NO
XML_OUTPUT             = xml
XML_SCHEMA             =
XML_DTD                =
XML_PROGRAMLISTING     = YES
#---------------------------------------------------------------------------
# configuration options for the AutoGen Definitions output
#---------------------------------------------------------------------------
GENERATE_AUTOGEN_DEF   = NO
#---------------------------------------------------------------------------
# configuration options related to the Perl module output
#---------------------------------------------------------------------------
GENERATE_PERLMOD       = NO
PERLMOD_LATEX          = NO
PERLMOD_PRETTY         = YES
PERLMOD_MAKEVAR_PREFIX =
#---------------------------------------------------------------------------
# Configuration options related to the preprocessor
#---------------------------------------------------------------------------
ENABLE_PREPROCESSING   = YES
MACRO_EXPANSION        = NO
EXPAND_ONLY_PREDEF     = NO
SEARCH_INCLUDES        = YES
INCLUDE_PATH           =
INCLUDE_FILE_PATTERNS  =
PREDEFINED             =
EXPAND_AS_DEFINED      =
SKIP_FUNCTION_MACROS   = YES
#---------------------------------------------------------------------------
# Configuration::additions related to external references
#---------------------------------------------------------------------------
TAGFILES               =
GENERATE_TAGFILE       =
```

```
ALLEXTERNALS           = NO
EXTERNAL_GROUPS        = YES
PERL_PATH              = /usr/bin/perl
#---------------------------------------------------------------------------
# Configuration options related to the dot tool
#---------------------------------------------------------------------------
CLASS_DIAGRAMS         = YES
HIDE_UNDOC_RELATIONS   = YES
HAVE_DOT               = NO
CLASS_GRAPH            = YES
COLLABORATION_GRAPH    = YES
UML_LOOK               = NO
TEMPLATE_RELATIONS     = NO
INCLUDE_GRAPH          = YES
INCLUDED_BY_GRAPH      = YES
CALL_GRAPH             = NO
GRAPHICAL_HIERARCHY    = YES
DOT_IMAGE_FORMAT       = png
DOT_PATH               =
DOTFILE_DIRS           =
MAX_DOT_GRAPH_WIDTH    = 1024
MAX_DOT_GRAPH_HEIGHT   = 1024
MAX_DOT_GRAPH_DEPTH    = 1000
GENERATE_LEGEND        = YES
DOT_CLEANUP            = YES
#---------------------------------------------------------------------------
# Configuration::additions related to the search engine
#---------------------------------------------------------------------------
SEARCHENGINE           = NO
```

A continuación se muestra las partes de código más importantes en la implementación:

```
Clase Interpreter.h/cxx
-----------------------

#include "PageModelObj.h"
//para PageModelObj
PageModelObj *makePageModel(const FOTBuilder::StModel &);
PageModelObj *makePageModel(const FOTBuilder::StModel &, const StringC &);


//para PageModel
PageModelObj *makePageModelNone();
void sendPageModel(PageModelObj *);
PageModelObj* getSendPageModel(int);
//para PageModel -> Generic IC
bool convertPageModelC(ELObj *, const Identifier *, const Location &, FOT      Bui
//Para PageModel
PageModelObj *pageModelNoneObj_;
```

```
PairObj *pageModelsSends_;
PairObj *actualpair_;

//creando un PageModelObj
inline
PageModelObj *Interpreter::makePageModel(const FOTBuilder::StModel & stpm)
{
  return new (*this) PageModelObj(stpm);
}

inline
PageModelObj *Interpreter::makePageModel(const FOTBuilder::StModel & stpm, const S
{
  return new (*this) PageModelObj(stpm,str);
}

//para PageModel
inline
PageModelObj *Interpreter::makePageModelNone()
{
  return pageModelNoneObj_;
}

En SchemeParser.cxx
-------------------
//para PageModel
bool SchemeParser::doDefinePageModel()
{
  StringC *nomdefine = 0;
  //obtengo la localizacion de los datos de entrada actuales
  Location loc(in_->currentLocation());
  //enum token, todos los tokens posibles
  Token tok;
  //mira a ver si es un token valido con el enum
  if (!getToken(allowIdentifier, tok))
    return 0;
  //obtenemos el token a identificar
  Identifier *ident = lookup(currentToken_);
  nomdefine = & currentToken_;
  //miramos si es un syntacticKey
  Identifier::SyntacticKey key;
  if (ident->syntacticKey(key) && (key <= int(Identifier::lastSyntacticKey)))
    message(InterpreterMessages::syntacticKeywordAsVariable,
            StringMessageArg(currentToken_));

  Location defLoc;
  unsigned defPart;
  if (ident->defined(defPart, defLoc)
      && defPart <= interp_->currentPartIndex()) {
```

```
  if (defPart == interp_->currentPartIndex()) {
    message(InterpreterMessages::duplicateDefinition,
            StringMessageArg(ident->name()),
            defLoc);
    return 0;
  }
 }

//pagemodelObj_ = interp_->makePageModel(FOTBuilder::StModel(), * nomdefine);
pagemodelObj_ = interp_->makePageModel(FOTBuilder::StModel());
StringC tmpnom = *nomdefine;

 for (;;) {
  //si es token abrir parentesis, sale, otra vez?
  if (!getToken(allowOpenParen|allowCloseParen, tok))
    return 0;
  //token cierre de parentesis, fin!
  if (tok == tokenCloseParen){
     break;
   }

  //token identificador (width,height...)
  if (!getToken(allowIdentifier, tok))
    return 0;

  //obtiene identificador
  const Identifier *ident = lookup(currentToken_);
  Identifier::SyntacticKey key;
  if (!ident->syntacticKey(key))
    return 0;
  else {
    switch (key) {
    case Identifier::keyWidth:
      if (!doWidth())
 return 0;
      break;
    case Identifier::keyHeight:
      if (!doHeight())
        return 0;
      break;
    case Identifier::keyFillingDirection:
      if (!doFillingDirection())
        return 0;
      break;
    case Identifier::keyRegion:
   if (!doRegion())
        return 0;
      break;
    default:
```

```
        return 0;
      }
    }
  }
  //creamos el objeto donde guardara la info de la hoja de estilo
  if (nomdefine) {
            pagemodelObj_->setID(tmpnom);
            // debuger
     // cout << "........................................." << endl;
            // cout << "SchemeParser.cxx--Nombre de define-page-model: ";
            // pagemodelObj_->whatID();
            // cout << endl;
  }
   else pagemodelObj_ = interp_->makePageModel(FOTBuilder::StModel());

  //lo hace permanente en interpreter!!
  interp_->sendPageModel(pagemodelObj_);

  Owner< Expression> expr;
  expr = new ConstantExpression(pagemodelObj_, in_->currentLocation());
  pagemodelObj_ = 0;
  ident->setDefinition(expr, interp_->currentPartIndex(), loc);
  return 1;
}
```

Clase PageModelObj

```
/**
@file PageModelObj.h

Cabecera y definición para la clase PageModelObj
*/


/**
Clase PageModelObj

PageModelObj se encarga de guardar los datos de la hoja de estilo del define-page-

@author Cristian Tornador Antolin
*/

// Copyright (c) 1996 James Clark

#ifndef PageModelObj_INCLUDED
#define PageModelObj_INCLUDED 1

#include "types.h"
#include "Collector.h"
```

```
#include "OutputCharStream.h"
#include "Boolean.h"
#include "Location.h"
#include "Node.h"
#include "FOTBuilder.h"
#include < string.h>
#include "ELObj.h"

#ifdef DSSSL_NAMESPACE
namespace DSSSL_NAMESPACE {
#endif

class PageModelObj : public ELObj{
public:
  void *operator new(size_t, Collector & c) {
    return c.allocateObject(1);
  }
  void operator=(PageModelObj &);
//---------------------------------------CONSTRUCTORES
  PageModelObj();
  //construye el objeto a partir de una estructura FOTBuilder::StModel
  PageModelObj(const FOTBuilder::StModel &);
  PageModelObj(const FOTBuilder::StModel &, const StringC &);
  PageModelObj(PageModelObj& pm): mystmodel_(new FOTBuilder::StModel(pm.mystmodel

  //dev. la referencia de este objeto.
  PageModelObj *asPageModel();
  //void whatID(Interpreter &, _IO_ostream_withassign &);
  void whatID();
  bool isEqual(ELObj &);
  //muestra los valores de este objeto.
  void print(Interpreter &, OutputCharStream &);
  //void print(Interpreter &, _IO_ostream_withassign &);
  void print();
//---------------------------------------MODIF ATRIBUTOS
  void setWidth(long);
  void setHeight(long);
  void setFillingDirection(FOTBuilder::Symbol);
  void setRegion_X_Origin(long);
  void setRegion_Y_Origin(long);
  void setRegion_Width(long);
  void setRegion_Height(long);
  void setRegion_Header_Height(long);
  void setRegion_Header_Width(long);
  void setRegion_Header_FillingDirection(FOTBuilder::Symbol);
  void setRegion_Header_ContentsAlignment(FOTBuilder::Symbol);
  void setRegion_Footer_Height(long);
  void setRegion_Footer_Width(long);
  void setRegion_Footer_FillingDirection(FOTBuilder::Symbol);
```

```
  void setRegion_Footer_ContentsAlignment(FOTBuilder::Symbol);
  void setRegion_FillingDirection(FOTBuilder::Symbol);
  void setID(StringC &);
  //extrae los valores en forma de la estructura FOTBuilder::StModel, que es la so
  bool pageModelData(FOTBuilder::StModel &);
//---------------------------------------CONSULTORES ATRIBUTOS
  long getWidth();
  long getHeight();
  FOTBuilder::Symbol getFillingDirection();
  long getRegion_X_Origin();
  long getRegion_Y_Origin();
  long getRegion_Width();
  long getRegion_Height();
  long getRegion_Header_Height();
  long getRegion_Header_Width();
  FOTBuilder::Symbol getRegion_Header_FillingDirection();
  FOTBuilder::Symbol getRegion_Header_ContentsAlignment();
  long getRegion_Footer_Height();
  long getRegion_Footer_Width();
  FOTBuilder::Symbol getRegion_Footer_FillingDirection();
  FOTBuilder::Symbol getRegion_Footer_ContentsAlignment();
  FOTBuilder::Symbol getRegion_FillingDirection();
  StringC getID();
  FOTBuilder::StModel mystmodel();
//---------------------------------------ATRIBUTOS
private:
 FOTBuilder::StModel *mystmodel_;
 StringC idstr;
};

#ifdef DSSSL_NAMESPACE
}
#endif

#endif /* not PageModelObj_INCLUDED */


Parte mas importante de PageModelObj.cxx
----------------------------------------

bool PageModelObj::pageModelData(FOTBuilder::StModel& stpm)
{
 StringObj res(idstr);
 const Char *s = res.data();
 char ptrid[30];
 size_t i;
 for (i = 0; i < res.size(); i++)
   switch (s[i]) {
       case '\\':
```

```
        case '"':
          // fall through
  break;
        default:
          ptrid[i]=s[i];
          break;
    }
  ptrid[i]='\0';
  stpm = FOTBuilder::StModel(*mystmodel_,ptrid);
  return true;
}

void PageModelObj::operator=(PageModelObj& pm)
{
idstr = pm.getID();
mystmodel_ = new FOTBuilder::StModel(pm.mystmodel());
}

FOTBuilder::StModel PageModelObj::mystmodel()
{
 return *mystmodel_;
}

//FALTA .....
bool PageModelObj::isEqual(ELObj & obj)
{
  FOTBuilder::StModel stpm;
  //mira que se trate de PageModelObj y dev. los valores de los atributos anchura
  return (obj.pageModelData(stpm)
           && (stpm.stwidth_ == getWidth())
    && (stpm.stfillingdirection_ == getFillingDirection())
           && (stpm.stheight_ == getHeight()));
}

//void PageModelObj::whatID(Interpreter & interp, _IO_ostream_withassign & out)
//Para DEBUG
void PageModelObj::whatID()
{
/*
 StringObj res(idstr);
 cout << "\"";
 const Char *s = res.data();
 for (size_t i = 0; i < res.size(); i++)
   switch (s[i]) {
        case '\\':
        case '"':
          cout << "\\";
          // fall through
        default:
```

```
          cout.put(s[i]);
           break;
    }
 cout << "\"";
*/
}


#ifdef DSSSL_NAMESPACE
}
#endif
```

Bibliografía

Javier Farreres, *The DSSSL Book: An XML/SGML Programming Language*, Kluwer Academic Publishers.

Foro sobre Opejade, OpenSP y sus fuentes.

http://sf.net/projects/openjade/

http://openjade.sourceforge.net/

James Clark, JC, *James Clark's Home Page*.

http://www.jclark.com/

Página especializada en SGML/XML.

http://www.mulberrytech.com/

Documentación y tutoriales sobre DSSSL.

http://dsssl.netfolder.com/

Colofón

Firmado: Cristian Tornador Antolin

Bellaterra, 15 de Junio del 2007

Resumen. Este proyecto trata de añadir nuevas funcionalidades a la herramienta Openjade, la cual es el único procesador de hojas de estilo DSSSL para documentos SGML, en la comunidad de Open Source. En concreto se intenta insertar los modelos de páginas para una secuencia de páginas complejas, y a partir de aquí crear un backend en Latex/TeX. Esto supondría un gran salto en la creación de documentos SGML ya que permitiría crear diferentes regiones en una misma página. Se explica todo el procedimiento de implementación conceptos básicos a tener en cuenta y se muestra como se consigue formar el árbol de objetos para poder crear el backend TeX. Finalmente, aunque se consigue construir el backend, donde se muestra que la construcción de la nueva funcionalidad es correcta, no es posible formar une backend en Latex sin limitaciones importantes, debido a que Latex no está preparado para este tipo de procesamiento automatico. Sin embargo se crean diferentes salidas que se fueron formando hasta llegar a la conclusión que se debía implementar una nueva macros en TeX para Latex que soportara el modelo de página.