Generación de Documentos con Contenido Variable en DPLfw*

Abel Gómez, Mª Carmen Penadés, and José H. Canós

ISSI – DSIC, Universitat Politècnica de València.
Cno. de Vera, s/n. 46022 Valencia. Spain.
{agomez,mpenades,jhcanos}@dsic.upv.es

Resumen Actualmente existen soluciones tecnológicas para la generación de documentos personalizados en cuanto a sus contenidos y apariencia. Sin embargo, todas ellas requieren de amplios conocimientos en lenguajes especializados (XML, XSLT o XPATH entre otros) y no contemplan tareas relacionadas específicamente con el dominio, como es la identificación de la variabilidad en el contenido de los documentos. En este trabajo presentamos DPLfw, un entorno de trabajo basado en modelos para la generación de documentos con contenido variable. DPLfw es una implementación de la propuesta de Líneas de Producto de Documentos, donde la variabilidad en el contenido se representa mediante características, y la generación de documentos se soporta sobre un proceso basado en Líneas de Productos. Este artículo describe la arquitectura de DPLfw, a la vez que muestra su uso en la generación de documentación de usuario.

Palabras Clave: Impresión de Datos Variables, Líneas de Productos de Documentos, Modelado de Características, Ingeniería Dirigida por Modelos, DITA

1. Introducción

Gestionar documentos con contenido variable es un aspecto fundamental en diversos ámbitos, tales como la enseñanza a distancia, el comercio electrónico o el gobierno electrónico. Los principales retos en la generación de manuales personalizados, contratos o documentos gubernamentales, por ejemplo, son la definición de variantes del documento y la reutilización de contenidos. En el campo de la Ingeniería de Documentos, la generación de documentos personalizados se conoce como Impresión de Datos Variables —Variable Data Printing (VDP)—. En las últimas décadas se han realizado diversas propuestas, desde las primeras al estilo de *Mail Merge* se ha llegado a propuestas con mayor grado de sofisticación, permitiendo la generación de documentos multimedia personalizados [13, 20]. Sin embargo, estas herramientas proporcionan una gestión de la variabilidad a bajo nivel donde el diseñador de documentos está obligado a tener un amplio

* Este artículo es una versión extendida de [7] preparada conforme a las recomendaciones de la llamada a contribuciones de JISBD 2012.

conocimiento de XML y tecnologías asociadas para poder definir el documento personalizado. Además, en numerosas ocasiones, la reutilización es oportunista, en vez de potenciar una reutilización sistemática. En este contexto el objetivo es proporcionar una alternativa para la generación automática de documentos, a la vez que oculte la complejidad intrínseca a una alta variabilidad.

Una solución para ocultar la complejidad consiste en el desarrollo de herramientas para usuarios finales que acerquen la definición de la variabilidad al dominio del problema: los usuarios seleccionan qué contenidos de un documento son fijos y cuáles pueden variar independientemente del formato final del mismo. Además, la funcionalidad de dichas herramientas debe soportarse por una metodología que guíe el proceso de generación de un documento. Igualmente, y con el fin de incrementar la eficiencia, es deseable que se dé un soporte integrado para la reutilización de componentes de documento de forma sistemática.

Este artículo describe DPLFW, un *framework* para la generación de documentos con contenido variable que cumple con los requisitos anteriores. Este framework, presentado inicialmente en [7], está basado en las Líneas de Producto de Documentos —Document Product Lines (DPL) [17]— que proporcionan un marco de trabajo alternativo a la aproximación tradicional seguida en VDP. DPL permite la creación de documentos con contenido variable a usuarios no expertos y asegura la reutilización de contenidos a nivel del dominio, siguiendo los principios de la Ingeniería de Líneas de Productos Software —Software Product Line Engineering (SPLE) [4]—. La clave para el éxito de un proceso de DPL reside, por una parte, en la definición de un modelo de variabilidad (llamado modelo de características) que describa cómo pueden variar los documentos; y por otra parte, en la existencia de una colección organizada de componentes de documento (*core assets* según la terminología de SPLE). Los componentes de documento son piezas de contenido que pueden combinarse para producir el documento final de acuerdo con el modelo de características. DPLFW implementa el proceso DPL siguiendo los principios de la Ingeniería Dirigida por Modelos —Model Driven Engineering (MDE) [12]—, ilustrándose su funcionalidad en el caso de estudio presentado.

El resto del artículo se estructura como sigue. La sección 2 introduce DPL, mostrando cómo se han adaptado las técnicas de SPLE a la generación de documentos. La sección 3 describe el *framework* DPLFW. La sección 4 ilustra la generación de documentación de usuario. La sección 5 resume diversos trabajos relacionados. Finalmente se presentan las conclusiones y trabajos futuros.

2. Líneas de Producto de Documentos

DPL proporciona una guía metodológica para modelar los contenidos comunes y variables en familias de documentos como un conjunto de características. A partir de una selección de características para un documento concreto, se genera un editor personalizado (re)utilizando componentes, y siguiendo una aproximación de líneas de producto. Este editor es un artefacto que permite guiar el flujo de edición actuando de forma similar a un asistente. Su ejecución produce instancias

específicas de un documento con contenido variable que serán posteriormente visualizadas con una determinada presentación para producir la versión final del mismo. A continuación resumimos la metodología DPL.

2.1. Modelado de Características en Documentos

Un aspecto fundamental en DPL es la identificación de la variabilidad en documentos desde una perspectiva del dominio. Para soportar esta tarea, DPL adapta los modelos de características clásicos (notación FODA [10]) al dominio de la generación de documentos, teniendo en cuenta la visión general existente en la comunidad de Ingeniería de Documentos. De acuerdo con ella, un documento en DPL se define como la unión de contenido y presentación. Tomando por ejemplo los documentos para el pago de impuestos, encontramos que algunos contenidos son obligatorios, mientras que otros sólo son aplicables a casos específicos, pudiendo omitirse (variabilidad en el contenido). Por otra parte, los contenidos pueden mostrarse de diversas formas: por ejemplo, algunos pueden presentarse impresos, mientras que otros pueden presentarse mediante formularios electrónicos. Esta dimensión tecnológica se maneja en DPL de forma explícita.

Por lo tanto, en DPL se pueden tratar dos fuentes de variabilidad: contenido y tecnología, lo cual motiva distinguir dos tipos de características: aquellas relacionadas con el contenido del documento (*ContentDocumentFeature* o CDF); y aquellas relacionadas con la tecnología usada para representar dichos contenidos (*TechnologyDocumentFeature* o TDF). Una CDF puede asociarse a una o más TDFs. Además, se pueden definir relaciones cruzadas —como *requiere*, *excluye*, y combinaciones lógicas de ellas ($\wedge$ y $\vee$)— para expresar restricciones, como es habitual en el ámbito del modelado de características.

2.2. Proceso de Generación de un Documento

Un proceso DPL, al igual que en SPLE y según se describe detalladamente en [7], incluye dos subprocesos incrementales e iterativos. El primero, llamado *Ingeniería del Dominio* se compone de cuatro tareas. En la primera tarea, *Analizar la Familia de Documentos*, un ingeniero del dominio especifica la familia de documentos en términos de características de contenido y tecnología, produciendo un *modelo de características*. En la segunda tarea, *Diseñar la Familia de Documentos*, se define una arquitectura de referencia del documento, identificando los componentes de documento (relacionados con el contenido) y los componentes software (relacionados con la tecnología que soportan el contenido) que son necesarios según el modelo de características definido previamente.

En la tarea *Desarrollar Core-Assets* se identifican, se buscan y/o se desarrollan (en caso de que no existan) los componentes (de documento o software) necesarios para generar la línea de documentos, denominados genéricamente *core-assets*. Todos ellos se almacenan en un repositorio, y para su búsqueda y organización se hace uso de los metadatos asociados a cada uno de ellos. Finalmente, en la tarea *Generar Línea de Documentos* se diseña y obtiene un *plan de producción*, esto es, un proceso que especifica cómo se integran los diferentes

componentes de acuerdo a las relaciones definidas entre las características de la familia de documentos.

El segundo subproceso, *Ingeniería de la Aplicación*, da soporte al proceso específico de generación de documentos de contenido variable. En la tarea *Caracterizar Documento*, el ingeniero de documentos (encargado de generar un documento concreto) selecciona los puntos de variabilidad deseados, esto es, qué características opcionales y/o alternativas se incluirán en el documento. A continuación, en la tarea *Recuperar Core-Assets* se obtienen los *core-assets* según la selección realizada; y en la tarea *Generar Flujo de Creación del Documento*, los diferentes *core-assets* son ensamblados para construir un *Editor de Documento Personalizado*. Se incluyen tanto los de tecnología (componentes software capaces de mostrar un contenido) como los de contenido (contenido del documento que es conocido y muestra el editor *a priori*). Finalmente, en la tarea *Ejecutar Flujo de Creación del Documento*, se usa el editor para completar el contenido final (si es necesario), generándose el documento final.

3. El *Framework* DPLfw

DPLfw proporciona la base metodológica y tecnológica para crear documentos de contenido variable según DPL. DPLfw se ha desarrollado siguiendo los principios de la Ingeniería Dirigida por Modelos, que elevan el nivel de abstracción y reducen el esfuerzo necesario en el desarrollo mediante técnicas programación generativa. Además, DPLfw se ha diseñado para ser extensible y configurable.

Las principales tecnologías que dan soporte a la herramienta son dos: Equinox [14] y *Eclipse Modeling Framework*(EMF) [21]. Equinox es la implementación del proyecto Eclipse para estándar OSGi [16], un modelo dinámico de componentes así como una plataforma de servicios para construir aplicaciones Java modulares y extensibles. Por su parte, EMF es un marco de trabajo para construir aplicaciones basada en técnicas de MDE. DPLfw supone un paso más allá en la implementación de DPL con respecto a trabajos previos [17, 18], donde en lugar de usar una herramienta específica, se usaban plataformas de SPLE genéricas. Además, la herramienta actual proporciona mecanismos de verificación (gracias a el uso de FaMa Tool Suite [8]) y validación avanzados, mejorando la propuesta inicial de DPLfw realizada en [7].

La figura 1 muestra los principales elementos de DPLfw. El *Editor de Características* se usa para definir la variabilidad del dominio en un modelo de características de documento. Éste se comunica con el módulo de verificación de FaMa que ayuda al usuario en la definición del modelo. El *Editor de Componentes* permite crear nuevos *core-assets* que pueden almacenarse en el *Repositorio*. Los elementos citados son los que dan soporte a la fase de *Ingeniería del Dominio*, mientras que el resto dan soporte a la fase de *Ingeniería de la Aplicación*. En concreto, el *Editor de Configuraciones* permite la selección de los diferentes puntos de variabilidad, obteniendo una *configuración*. Gracias el módulo de validación se garantiza que la configuración definida por el usuario es válida y no viola ninguna de las restricciones definidas en el modelo de características. Dicha

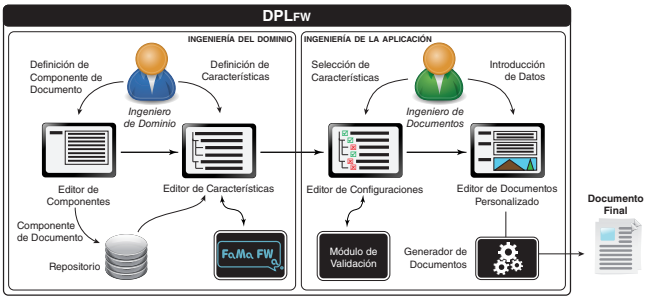


Figura 1: Vista general de DPLFW

configuración se utiliza para generar un *Editor de Documentos Personalizado*, que permite completar cualquier dato variable que pueda quedar por rellenar. Finalmente, el *Generador de Documentos* construye un documento final en un formato concreto (impreso, hipermedia, etc.).

3.1. El Repositorio

El proceso de automatización en la generación de documentos depende en gran medida en la disponibilidad de diferentes componentes que puedan ser reutilizados sistemáticamente. En DPLFW esta disponibilidad está garantizada por el *Repositorio*, un recurso que permite gestionar (crear, actualizar y borrar) y recuperar los distintos componentes (por ejemplo mediante búsqueda de palabras clave), siguiendo una arquitectura cliente/servidor. Se ha desarrollado siguiendo la filosofía de MDE y empleando la tecnología de EMF, por lo que éste es accedido vía *Connected Data Objects*(CDO) [6]. CDO proporciona un acceso de alto nivel (basado en modelos) para las aplicaciones, ofreciendo mecanismos de autenticación, acceso concurrente, transaccionalidad, así como de recuperación y almacenamiento de componentes independientemente del sistema de base de datos usado. A continuación, y por motivos de claridad y espacio centraremos el discurso en los componentes de contenido.

Los servicios proporcionados por el *Repositorio* dependen en gran medida de la estructura de los componentes de documento. En DPL, estos componentes de documento se denominan *InfoElements* [17]. Un *InfoElement* se compone de dos bloques principalmente: datos y metadatos. El primero es una codificación del contenido real del componente de documento (por ejemplo, texto, imagen, o cualquier otro objeto multimedia). El segundo permite proporcionar información adicional que permite describir y gestionar el bloque de contenido. El sistema de metadatos elegido en DPL está basado en el estándar *Dublin Core*. Ejemplos de metadatos son *Título*, *Descripción*, *Idioma*, *Creador*, *Materia*, *Tipo*, etc.

Dada esta descripción, hemos modelado el *Repositorio* en DPLFW según muestra la figura 2. Un *Repositorio* es accesible mediante una localización representada por un identificador único (URI), y contiene un conjunto de nodos (*resourceNodes*) que permiten organizarlo de forma jerárquica. Existen dos tipos

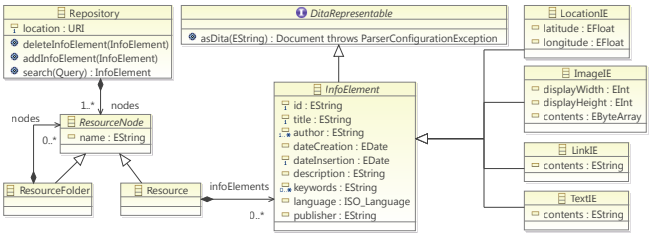


Figura 2: Modelo del *Repositorio*

de nodos: recursos (*Resources*) y carpetas(*ResourceFolders*). Los recursos pueden contener *InfoElements*, mientras que las carpetas sólo pueden contener otras carpetas o recursos. Los atributos de los *InfoElements* permiten representar los metadados. Por simplicidad, el metadato *Tipo*, que determina la codificación del contenido, se ha implementado mediante herencia. La figura muestra distintos tipos de *InfoElements*: texto (*textIE*), imagen (*ImageIE*), enlaces (*LinkIE*) y coordenadas geográficas (*LocationIE*). Finalmente, los *InfoElements* implementan la interfaz *DitaRepresentable*, lo que permite mantener compatibilidad hacia atrás con trabajos previos [17, 18] a la vez que permiten el uso de herramientas automatizadas para la generación de documentos (por ejemplo, *Dita Open Toolkit* [1]), mediante una representación XML según el estándar DITA [15].

3.2. El Editor de Características

El *Editor de Características* ha sido generado de forma automática empleando las herramientas de EMF. La figura 3 muestra una versión simplificada del metamodelo soportado por DPLfw. En DPLfw, un modelo de características de documento (*DocumentFeatureModel*) se compone de un conjunto de características de documento (*DocumentFeature*), las cuales, según la sección 2 pueden referirse al contenido (*ContentDocumentFeature*) o a la tecnología (*TecnologyDocumentFeature*). Los tipos de características, así como las restricciones aplicables, son las comunes en el área de SPLE tal y como se indica en la sección 2.1.

El contenido real de un documento se asocia a las características de contenido mediante instancias de la clase *InfoElement*. Dichas características de contenido están asociadas a una o más características de tecnología que definen cómo se representarán. El artefacto encargado de representar el contenido, según la terminología de [9], es un diseminador (*Disseminator*). Ejemplos de diseminadores son: editores de texto, visores de imágenes, reproductores de vídeo o cualquier otra aplicación (por ejemplo, un servicio web).

Una familia de documentos es, por tanto, un modelo conforme al metamodelo descrito, y que se define mediante el editor de características. Los elementos de dicho modelo serán los que guíen la definición de la arquitectura de referencia del editor personalizado que se generará.

Cabe destacar que el *Editor de Características* proporciona capacidades avanzadas de validación y verificación, como se demuestra en la sección 4.1. Para ello

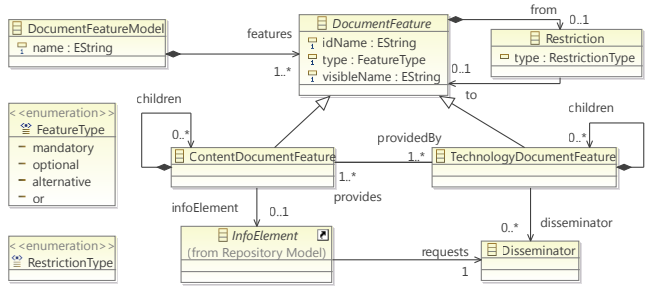


Figura 3: Metamodelo de Características de Documentos

se ha integrado de forma transparente para el usuario con FaMa Tool Suite [8], un entorno de trabajo para el análisis de modelos de características que se apoya en las representaciones lógicas y solucionadores más comunes en la literatura sobre modelado de características [2].

3.3. El Editor de Configuraciones

El *Editor de Configuraciones* permite guiar el proceso de caracterización (o selección de características) de un documento específico. Cada vez que el usuario realice una selección, se comprobará que todas las características obligatorias decidibles se seleccionen de forma automática; mientras que las características opcionales y alternativas deberán seleccionarse de forma explícita. Igualmente, cuando se deseleccione una característica, también se deseleccionarán todas sus características hijas. Por otra parte, todas aquellas características que no puedan seleccionarse serán marcadas como no seleccionables en el editor. Finalmente, en aquellos casos en los que se incumplan restricciones del modelo y no sea posible determinar de forma automática una solución (por ejemplo, en relaciones de dependencia o exclusión complejas), el editor mostrará en la vista de problemas aquellas relaciones conflictivas para que el usuario decida la acción correctiva, como se demuestra en la sección 4.2.

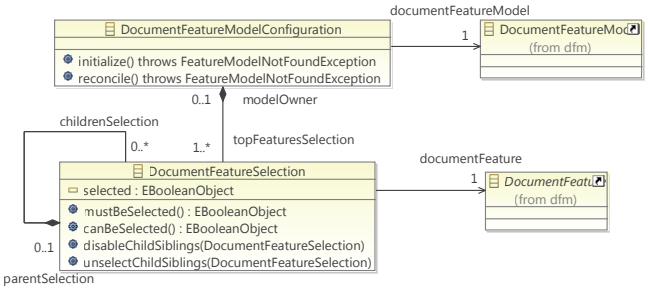


Figura 4: Metamodelo de Configuraciones de Documentos

Cada configuración de un modelo de características se almacena como un artefacto separado, cuyo contenido está enlazado al modelo de características asociado, manteniendo una estructura simple. Según la figura 4, una configuración de un modelo de características (*DocumentFeatureModelConfiguration*) está enlazado a un modelo de características (*DocumentFeatureModel*, véase la figura 3), y contiene una jerarquía de selecciones (*DocumentFeatureSelections*) cuya estructura se asemeja a la del modelo de características asociado. Una selección puede tener tres posibles valores: `true`, si la característica asociada está seleccionada; `false`, si la característica está deseleccionada de forma explícita; o `null` si el estado de la característica no se ha decidido todavía. En caso de que un modelo de características cambie, su modelo de configuración cambia automáticamente mediante un mecanismo de reconciliación. En caso de que, por ejemplo, se hayan añadido nuevas características que invaliden la configuración previa, los mecanismos de notificación de errores permiten guiar al usuario en las tareas de reconciliación que deben realizarse necesariamente de forma manual por no disponer de información suficiente.

3.4. Editor de Documentos Personalizado y Generación del Documento Final

El *Editor de Documentos Personalizado* implementa el *flujo de creación del documento*, permitiendo especificar por completo el documento final antes de ser generado. Siguiendo el ejemplo previo sobre documentos para el pago de impuestos (sección 2.1), el flujo de creación del mismo especifica qué contenidos deben rellenarse y en qué orden. Este editor personalizado se genera a partir de la configuración definida. Finalmente, y una vez que los contenidos han sido rellenado por completo, el *Generador de Documentos* construye el documento final con su contenido y visualización definitivos.

DPLfw, en su estado actual de desarrollo, no soporta por completo la generación de editores personalizados. Sin embargo, para cubrir esta fase de DPL, se han implementado diversos mecanismos que proporcionan un comportamiento por defecto. Esto permite llevar a cabo las tareas de *Generación del Flujo de Creación del Documento* y *Ejecución del Flujo de Creación del Documento* empleando herramientas alternativas. En concreto, puesto que los *InfoElements* pueden representarse como especificaciones DITA, una configuración de un documento se transforma de forma automática a un *mapa DITA*, que representa el flujo de edición del documento. Este *mapa DITA* puede editarse empleando el editor integrado en DPLfw (que juega el papel del *Editor de Documentos Personalizado*) y se usa posteriormente para generar el documento final empleando el motor *DITA Open Toolkit* [1]. De esta manera, DPLfw cubre un proceso DPL por completo, permitiendo la edición y generación de cualquier documento con contenido variable, pudiendo ser su formato final un archivo PDF, RTF, un conjunto de documentos HTML enlazados, etc.

4. Caso de estudio: Generación de un Manual de Usuario

La generación de documentos con contenido variable pretende cambiar la filosofía de «1 documento para N personas» a «1 documento para 1 persona». En el caso de documentación de aplicaciones software, puede ser relevante disponer de distintos manuales que documenten distintos aspectos de una herramienta, por ejemplo: requisitos e instrucciones de instalación para los administradores de sistemas; guía de primeros pasos y manual de uso para usuarios finales; combinaciones de ambos para usuarios avanzados; etc. A lo largo de esta sección mostraremos cómo DPLfw ha soportado la generación de documentación para la propia herramienta.

4.1. Ingeniería del dominio

La figura 5 muestra el modelo de características que describe la variabilidad de la familia de manuales de la herramienta DPLfw. En el modelo encontramos cuatro características de contenido (CDF) de primer nivel: (1) *Introducción*, que realiza una introducción a la herramienta dependiendo de dos tipos de usuarios distintos (administradores de sistemas o usuarios finales); (2) *Instrucciones de Instalación*, que describe los requisitos de la herramienta y los pasos para instalar tanto el servidor (el repositorio) como el cliente de DPLfw; (3) la guía de *Primeros Pasos*, que enseña a los usuarios cómo empezar a trabajar rápidamente con la herramienta; y (4) el *Información de Versiones*, que describe la evolución de la herramienta a lo largo de sus distintas versiones. Por simplicidad el modelo sólo muestra una única característica de tecnología (TDF), el *Tipo de Manual*, que puede ser como documento impreso o como documento multimedia. Igualmente, se han definido diferentes restricciones de dependencia y exclusión. Por ejemplo, si se selecciona una introducción para *Administradores de Sistemas* se requiere que posteriormente se incluyan las instrucciones de instalación del servidor, así como la información de versiones. Por su parte si se realiza una introducción para usuarios finales, se requiere incluir la guía de primeros pasos.

Para ilustrar las capacidades de análisis de DPLfw se han introducido además dos relaciones de exclusión mutua (que invalidan el modelo) entre la información de las versiones *v0.2.1* y *v0.3*. De esta manera, se puede observar el resultado del análisis en la vista de problemas (figura 5, derecha). Específicamente, esta vista indica que existen *características muertas* (nunca pueden seleccionarse sin violar las restricciones del modelo). Esto se debe a que las características *v0.2.1* y *v0.3* son obligatorias en caso de que *Información de versiones* esté seleccionada, sin embargo al ser estas dos últimas mutuamente excluyentes, no pueden seleccionarse ninguna de las tres. A su vez, la característica opcional *Administración de sistemas* tampoco puede ser seleccionada nunca puesto que requiere *Información de versiones*, que como se ha visto, es una característica muerta. Por último, el editor también indica que *Primeros pasos* es una característica *obligatoria falsa*, es decir, a pesar de estar declarada como opcional, todas las configuraciones posibles la contienen, comportándose como si fuera obligato-

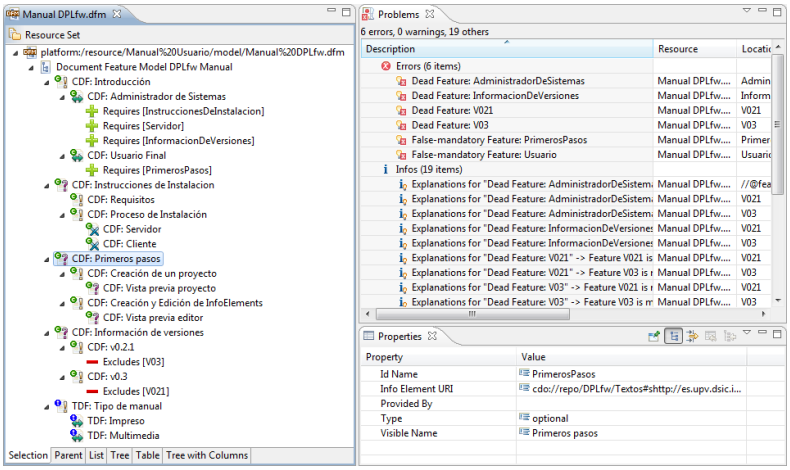


Figura 5: Modelo de características de ejemplo (con errores)

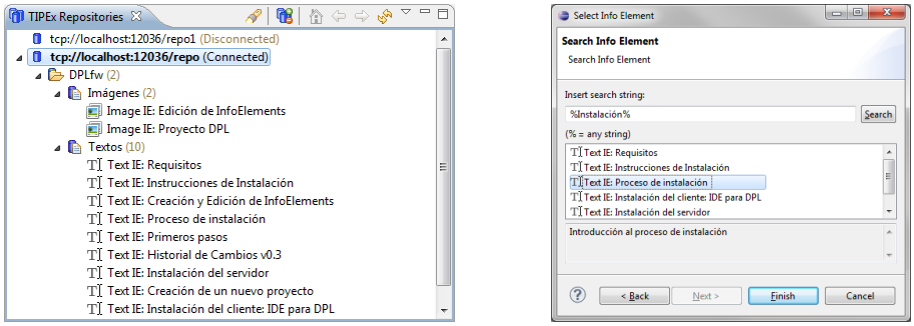
ria. Para obtener un modelo de características válido basta con eliminar las dos relaciones de exclusión.

A la vez que se define la familia de documentos se deben desarrollar/recuperar los *core-assets* que se van a asociar a cada característica mediante el editor mostrado en la figura 5. Para añadir, eliminar y gestionar los diferentes componentes de documento se dispone del *Explorador de repositorios* mostrado en la figura 6a. Como se observa, en un repositorio los componentes se organizan de forma jerárquica. Sin embargo, para la búsqueda y recuperación de componentes se dispone de una interfaz de búsqueda que facilita aún más la tarea, tal y como muestra la figura 6b. Para la creación y edición de componentes de documento DPLfw proporciona distintos editores, que, empleando una metáfora de formulario, permiten rellenar tanto los metadatos como el contenido de éstos.

El último paso de la *Ingeniería del Dominio* es la generación de la línea de productos de documento, obteniendo el plan de producción. DPLfw implementa un plan de producción por defecto donde: la estructura de la familia de documentos está determinada por el modelo de características; cada componente de contenido (*InfoElement*) tiene un diseminador por defecto; y los mecanismos para recuperar los diferentes componentes del repositorio están predefinidos.

4.2. Ingeniería de la aplicación

En la fase de *Ingeniería de la Aplicación*, el ingeniero de documentos emplea los modelos de variabilidad definidos previamente para generar el documento final. La figura 7a muestra el *Editor de Configuraciones* mientras se está definiendo un documento dirigido a un administrador de sistemas, que será el encargado de implantar DPLfw en una organización. Se puede observar cómo la configuración está parcialmente definida, siendo ésta incorrecta. La vista de problemas informa



(a) (b)

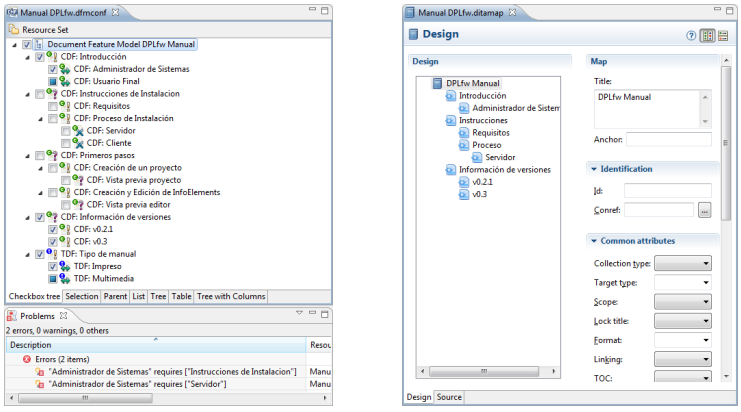
Figura 6: Explorador de repositorios (6a) y Buscador de InfoElements (6b)

de que al seleccionar una introducción para administradores de sistemas, también es necesario incluir las instrucciones de instalación del servidor de DPLFW según se definía en el modelo mostrado en la figura 5. En la figura también se observa que se ha seleccionado la característica de tecnología *tipo de manual impreso*, que nos permitirá generar un documento PDF.

Tras seleccionar las características *Instrucciones de Instalación*, y *Servidor* (*Requisitos* y *Proceso de Instalación* se seleccionan de forma automática al ser obligatorias), ya se dispone de una selección válida y el proceso continúa. Partiendo de esta caracterización se puede generar un documento acorde a ella. Para ello, en primer lugar, se recuperan los componentes de documento de los respectivos repositorios y se almacenan localmente como especificaciones DITA; y en segundo lugar, se genera automáticamente un mapa DITA que describe la estructura del documento. Todos estos elementos pueden editarse con los correspondientes editores que DPLFW integra. Por ejemplo, la figura 7b muestra el editor de mapas DITA correspondiente al caso de estudio. Por último, el generador construye el documento final. La figura 7c muestra el documento PDF que se genera para la selección de características descrita anteriormente.

5. Trabajos relacionados

El interés en los documentos de contenido variable ha crecido de forma significativa en los últimos años, especialmente en la comunidad de Ingeniería de Documentos. DARWIN VDP [5] y PageFlex [3] son ejemplos de soluciones tecnológicas que proporcionan mecanismos para la personalización de la presentación así como instrucciones avanzadas para el procesado de datos. Otros autores han abordado el problema mediante la edición de documentos XML, por ejemplo [13, 20]. Las herramientas están orientadas a la presentación, centrándose en el documento final y cómo las partes variables del documento se instancian. Sin embargo, no proporcionan ninguna guía metodológica para identificar y gestionar de forma temprana dichas fuentes de variabilidad, aspecto que es esencial



(a) (b)

(c)

Figura 7: Configuración (con errores) (7a), mapa DITA generado tras la corrección (7b y manual de usuario final en PDF (7c)

en dominios complejos. Además, su orientación puramente tecnológica y de bajo nivel les impide gestionar de forma genérica la variabilidad tecnológica.

Un caso interesante de gestión temprana de la variabilidad en documento se describe en [19]. En este trabajo se aborda la problemática de alinear el contenido de los documentos con las variantes de una línea de productos empleando DOPLER, un entorno para el desarrollo de SPLs orientadas por decisiones. En este trabajo se presenta, además, una metodología para llevar a cabo el proceso de generación de documentos. Sin embargo a diferencia de DPLfw, el principal objetivo de esta propuesta no es la generación de documentos, entendiendo estos como los artefactos resultantes de una línea de productos. Así, la propuesta más próxima a DPL la encontramos en [11], que utiliza FODA para especificar la variabilidad en documentos. Así, las características son asociadas a documentos específicos, y mediante un mecanismo de asignación se construye una determinada variante a partir de una selección de características. A diferencia de DPL, no existe ninguna distinción entre características de contenido y tecnología. Además, esta propuesta no sigue una aproximación de SPLE pura, sino que se utiliza una

aproximación transformacional. Finalmente, su uso está dirigido a aplicaciones de oficina, mientras que DPL es una herramienta de propósito general.

6. Conclusiones y trabajo futuro

La generación de documentos personalizados con reutilización de contenido es una cuestión fundamental en numerosos ámbitos. Para abordar este problema hemos presentado DPLfw, un *framework* y una herramienta que dan soporte a la metodología DPL para la generación de documentos basada en reutilización. Un proceso DPL empieza con la definición de un modelo que define las características de una familia de documentos, y a partir del cual se deriva un arquitectura de documento genérica. Así, un documento concreto (un miembro de la familia) se crea siguiendo un proceso donde se obtienen diferentes componentes de un repositorio y se ensamblan posteriormente, maximizando la reutilización. DPL está basada en los principios de SPLE, capturando la variabilidad en etapas tempranas, a diferencia de otras aproximaciones que están orientadas a la presentación y la tecnología de generación del documento final.

DPLfw es una herramienta completamente funcional que cubre por completo el ciclo de vida de un documento, desde la fase de análisis del dominio (donde se especifica la familia de documentos), hasta la obtención de un documento final. DPLfw mejora trabajos previos [7] y permite además modelar de forma sencilla modelos de características complejos: en primer lugar, permite garantizar que los modelos no contienen errores en la fase de modelado gracias a herramientas como FaMa; en segundo lugar, asiste al usuario en la fase de caracterización mediante selecciones automáticas y avisos; y en tercer lugar, permite la edición concurrente de modelos de características y configuraciones que se sincronizan mediante mecanismos automáticos de reconciliación.

Además del caso de estudio empleado para ilustrar la herramienta, DPLfw ha sido utilizado en casos de estudio reales, especialmente en la generación de los llamados *Planes de Emergencia* (documentos que sirven de guía de seguridad en el caso de situaciones de riesgo). Finalmente, cabe destacar que a pesar de que DPLfw es completamente funcional, aún resta por completar la construcción de editores para ejecutar el flujo de edición que permitirían explotar las capacidades de DPLfw al máximo. En este sentido, DPLfw ya soporta la implementación y reutilización de distintos tipos de diseminadores, restando únicamente la composición de éstos para la generación de editores personalizados.

Agradecimientos Este trabajo está parcialmente financiado por el Ministerio de Educación y Ciencia bajo el proyecto TIPEx (ref. TIN2010-19859-C03-03).

Referencias

[1] Anderson, Robert D and Shen, Jian Le and Elovirta, Jarno and Sirois, Eric and Weiser, Reuven: DITA Open Toolkit (2012), `http://dita-ot.sourceforge.net/`

[2] Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: A literature review. Information Systems 35(6), 615 – 636 (2010)
[3] Bitstream Inc.: Pageflex (2012), http://www.pageflex.com/
[4] Clements, P., Northrop, L.: Software Product Lines: practices and patterns, vol. 0201703327. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2001)
[5] CREO: DARWIN VDP (2012), http://www.creo.com/data/Products/Workflow%20Solutions/Darwin%20VI%20authoring%20tool/
[6] Eclipse Foundation: CDO (2012), http://www.eclipse.org/cdo/
[7] Gómez, A., Penadés, M.C., Canós, J.H., Borges, M.R., Llavador, M.: DPLFW: A framework for Variable Content Document Generation. In: 16th International Software Product Line Conference, SPLC 2012, Salvador, Brazil, September 2-7 (2012)
[8] ISA Research Group: FaMa FW (2012), http://www.isa.us.es/fama/
[9] Kahn, R., Wilensky, R.: A framework for distributed digital object services. Int. J. Digit. Libr. 6, 115–123 (April 2006)
[10] Kang, K., Cohen, S., Hess, J., Nowak, W., Peterson, S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study (1990)
[11] Karol, S., Heinzerling, M., Heidenreich, F., Assmann, U.: Using feature models for creating families of documents. In: DocEng '10. pp. 259–262. ACM, New York, NY, USA (2010)
[12] Kent, S.: Model driven engineering. In: Butler, M.J., Petre, L., Sere, K. (eds.) IFM 2002, Finland. LNCS, vol. 2335, pp. 286–298. Springer (2002)
[13] Lumley, J., Gimson, R., Rees, O.: A framework for structure, layout & function in documents. In: DocEng '05. pp. 32–41. ACM (2005)
[14] McAffer, J., VanderLei, P., Archer, S.: OSGi and Equinox: Creating Highly Modular Java Systems. Eclipse series, Addison-Wesley (2009)
[15] OASIS: Darwin Information Typing Architecture (DITA) v.1.2. Organization for the Advancement of Structured Information Standards (Dec 2010), http://docs.oasis-open.org/dita/v1.2/spec/DITA1.2-spec.html
[16] OSGi Alliance: OSGi Service Platform Core Specification. OSGi Alliance (2008), http://www.osgi.org/Specifications/HomePage/
[17] Penadés, M.C., Canós, J.H., Borges, M.R., Llavador, M.: Document product lines: variability-driven document generation. In: DocEng '10. pp. 203–206. ACM (2010)
[18] Penadés, M.C., Canós, J.H., Camarasa, S., Borges, M.R., Vivacqua, A.S.: Generación de documentos basada en líneas de producto. JISBD'11. Spain (2011)
[19] Rabiser, R., Heider, W., Elsner, C., Lehofer, M., Grünbacher, P., Schwanninger, C.: A flexible approach for generating product-specific documents in product lines. In: Bosch, J., Lee, J. (eds.) SPLC. LNCS, vol. 6287, pp. 47–61. Springer (2010)
[20] Sellman, R.: VDP templates with theme-driven layer variants. In: DocEng'07. pp. 53–55. ACM (2007)
[21] Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework. Addison-Wesley, Boston, MA, 2nd edition edn. (2009)