DPLfw: a Framework for the Product-Line-Based Generation of Variable Content Documents

Abel Gómez[1], Pau Martí[2], M. Carmen Penadés[2], and José H. Canós[2]

[1] AtlanMod team (Inria, Mines Nantes, LINA)
4 rue Alfred Kastler. 44307 Nantes, France
abel.gomez-llana@inria.fr
[2] ISSI – DSIC, Universitat Politècnica de València.
Cno. de Vera, s/n. 46022 Valencia. Spain.
{pmarti,mpenades,jhcanos}@dsic.upv.es

Abstract. *Document Product Lines* (DPL) is a document engineering methodology that applies product-line engineering principles to the generation of documents in high variability contexts and with high reuse of components. Instead of standalone documents, DPL promotes the definition of families of documents where the members share some common content while differ in other parts. The key for the definition is the availability of a collection of content assets which can be parameterized and instantiated at document generation time.
In this demonstration, we show the features of the DPL framework (DPLfw), the tool that supports DPL. DPLfw implements the domain engineering and application engineering stages of typical product line engineering approaches, supports different asset repositories, and generates customized documents in different output formats. We use the case study of the generation of customized emergency plans in a University campus [http://youtu.be/ueKGfmfkyI0].

1 Motivation

The concept of document has changed in last decades from the classical printed artifact to a purposeful and self-contained collection of information in a technology-neutral way [1]. In more and more domains, documents are the central pieces of the business processes; moreover, in most cases the generation of customized documents has become the final milestone. Examples are customized emergency plans for organizations, customized learning objects based on students' profiles, or customized book catalogs made according to customers' preferences.

Customization can be made in three dimensions, namely content, structure and presentation. Specifically, content customization has been tackled from two different perspectives. On the one hand, proposals on Variable Data Printing (VDP) use variables within documents that are used as placeholders for content that are replaced with values to generate customized documents. These approaches are mostly XML-based [5, 8, 10]: document components are defined

in XML, and the customized document is generated using XSLT, XPath, and other related technologies. In these proposals, the variability model is implicit (that is, it is embedded in XML, XPath and XSLT expressions), forcing document engineers to have a high knowledge about the XML world. On the other hand, more recent proposals such as [4, 9] use a product line approach to model the variability explicitly. These proposals use feature models to identify the variability points from a domain-oriented perspective, hiding the XML complexity.

The *Document Product Lines* approach (DPL) [2, 7] is an example of the latter. DPL was created with a twofold goal: first, to make variable content documents creation affordable to non-expert users by including a domain engineering process previous to document generation; and second, to enforce content reuse at domain level following principles of Software Product Line Engineering (SPLE). We implemented a tool, DPLfw [3], to provide the methodological and technological background to creating variable content documents by the DPL approach. DPLfw implements a true product line engineering process where the content variability is represented using document feature models and different variants of the document may be generated by defining different document configurations. DPLfw was developed following the MDE and Model Driven Architecture (MDA) paradigms, which allowed us to take advantage of code generation techniques for the implementation of the tool prototype.

2 The *Document Product Lines* methodology

DPL is a method for the generation of variable content documents. As in SPLE, the DPL process is the concatenation of two main subprocesses, namely Domain Engineering and Application Engineering. Fig. 1 shows the most relevant tasks and artifacts in each subprocess using the BPMN notation.

The goal of the *Domain Engineering* is to define a family of documents and related artifacts. A family is a set of documents that share some mandatory content while differ in other optional content. To enforce reuse, the content of a

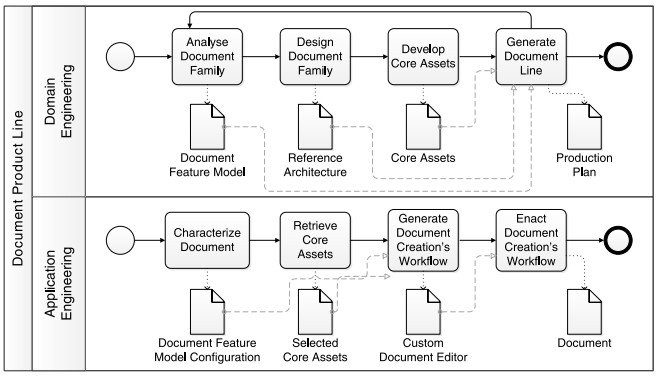

Fig. 1. DPL-based Document Generation Process

family of documents is structured via a document feature model. In the *Analyse Document Family* task, a domain engineer specifies the documents in terms of features and feature attributes. The result is a document feature model including mandatory, optional, alternative features and their corresponding attributes (if applicable). In the *Design Document Family* task, the generic document architecture is defined by identifying the document components (called *core assets* in the SPLE terminology and *InfoElements* in DPL) required according to the feature model built in the previous stage. These *InfoElements* may define a set of variables corresponding to a set of placeholders, i.e., parts of their content that may be instantiated at later stages of the document generation process. Specific instances of the architecture are created later in the process, after the variability points and document variable have been fixed for a specific document. DPL assumes the existence of a *Repository* where *InfoElements* are stored and organized for reuse. Metadata are attached to each *InfoElement* in order to support retrieval processes in the *Develop Core Asset* task to find existing components. Finally, in the *Generate Document Line* task, a production plan is obtained; it is a process that specifies how the components are integrated according to the different relationships defined between the document features.

In the *Application Engineering* subprocess, a member of the document family is generated. The process starts with the *Characterize Document* task by selecting in the configuration the specific document features and variable values to be included in the final document. Next, the core-assets associated to the selected document features are retrieved and put together to *Generate a Document Creation's Workflow* model that it is used to generate a set of *Custom Document Editors*. The editors provide guidance for *Enacting the Document's Creation Workflow* by giving both a task-oriented and user-centered view of the document based on editing task and permissions producing the final document.

3 The DPLfw Framework

Figure 2 describes how a (simplified) DPL process is carried out in DPLfw. The *Domain Engineering* stage is an iterative process. For the sake of simplicity no specific order is enforced to execute its tasks as far as there is a fully populated

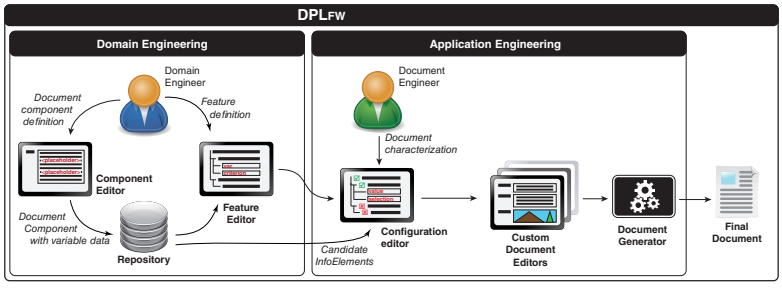

Fig. 2. DPLfw overview

document feature model describing the domain at the end of the stage. The *Feature Editor* is used by the *Domain Engineer* to characterize the variability of the domain as a document feature model. The *Feature Editor* interfaces with the *Repository*, which contains the *InfoElements* that will later be reused. All these elements support the *Analyse Document Family* task (cf. Figure 1).

It is noteworthy that, for the sake of simplicity, the *Reference Architecture* matches the structure of the feature model, and thus, the *Design Document Family* task does not require user-interaction. The *Component Editor* supports the *Develop Core Assets* task and is used to create new *InfoElements* and add them to the *Repository*. The *Generate Document Line*, which will describe how to retrieve and integrate the different components to obtain the final product, is also hidden from the user-point of view.

Regarding the *Application Engineering* subprocess, the *Configuration Editor* supports the *Characterize Document* task through the selection of variability points. Once a *document feature model configuration* is defined, the *Enact Document Creation's Workflow* task starts and the *Custom Document Editors* are generated by composing the *InfoElements*. These editors are used to fill in any remaining variable data. Finally, the *Document Generator* (a DITA-based [6] document generator engine) integrates the different components to obtain a fully instantiated document generated in a specific format (printed, hypermedia, etc.).

4 Demonstration: The *UPV Campus* Emergency Plans

Emergency plans development is the field we selected for our case study. An emergency plan is a document that contains all the knowledge required to respond to any incident in an organization. Emergency plans development is a domain that brings together a set of requirements that makes it an interesting real case study. In our demonstration we will show how a family of emergency plans is modeled, and how a customized emergency plan is generated. The *UPV Campus* is a real case study, and its document feature model represents the family of emergency plans of the university campus; where each building of the campus has its own emergency plan obtained as a configuration of the family.

Prior to the adoption of the DPL methodology in the UPV, new emergency plans were manually created using a text editor (MS Word). Applying DPL to the development of emergency plans requires that a set of document fragments (i.e. *InfoElements*) need to be created to be combined in the final document and a Document Feature Model describing the family of plans. However, sometimes an *InfoElement* needs some small parts to be changed from one emergency plan to another (e.g. the building name or the specific maps or warning systems, etc.). This scenario allows us to show how both approaches for document customization (variability-based customization and Variable Data Printing) have been combined in DPL and DPLFW.

5 Conclusions

DPLfw supports the generation of variable content documents in a product-line style. Such an approach has several advantages. On one hand, a domain-oriented variability specification helps to hide the complexity intrinsic to document description languages such as XML. On the other hand, the definition of generic content components increases the reusability of content significantly. DPLfw supports the generation of customized documents with high levels of reuse. Its foundation, DPL, aims at raising the level of abstraction in comparison with previous approaches, helping domain engineers and document engineers to develop families of documents without knowledge of the underlying document representation techniques such as XML, XPath, DITA or DocBook.

We have illustrated the use of DPLfw with an example taken from our collaboration with the emergency planning team at the *Universidad Politècnica de València*. The DPLfw documentation and prototype are publicly available for download in [3].

References

1. Glushko, R., McGrath, T.: Document Engineering: Analyzing and Designing Documents for Business Informatics & Web Services. MIT Press (2005)
2. Gómez, A., Penadés, M.C., Canós, J.H., Borges, M.R., Llavador, M.: A framework for variable content document generation with multiple actors. Information and Software Technology 56(9), 1101 – 1121 (2014), special Sections from "Asia-Pacific Software Engineering Conference (APSEC), 2012" and "Software Product Line conference (SPLC), 2012"
3. ISSI Research Group: DPLfw (2014), `http://dpl.dsic.upv.es/`, (spanish only)
4. Karol, S., Heinzerling, M., Heidenreich, F., Assmann, U.: Using feature models for creating families of documents. In: Proceedings of the 10th ACM symposium on Document engineering. pp. 259–262. ACM, New York, USA (2010)
5. Lumley, J., Gimson, R., Rees, O.: A framework for structure, layout & function in documents. In: Proceedings of the 2005 ACM symposium on Document engineering. pp. 32–41. ACM, New York, USA (2005)
6. OASIS: Darwin Information Typing Architecture (DITA) Version 1.2 (Dec 2010), `http://docs.oasis-open.org/dita/v1.2/spec/DITA1.2-spec.html`
7. Penadés, M.C., Canós, J.H., Borges, M.R., Llavador, M.: Document product lines: variability-driven document generation. In: Proceedings of the 10th ACM symposium on Document engineering. pp. 203–206. ACM, New York, USA (2010)
8. Piccoli, R.F.B., Chamun, R., Cogo, N.C., de Oliveira, J.a.B.S., Manssour, I.H.: A novel physics-based interaction model for free document layout. In: Proceedings of the 11th ACM symposium on Document engineering. pp. 153–162. ACM, New York, USA (2011)
9. 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. Lecture Notes in Computer Science, vol. 6287, pp. 47–61. Springer (2010)
10. Sellman, R.: VDP templates with theme-driven layer variants. In: Proceedings of the 2007 ACM symposium on Document engineering. pp. 53–55. ACM, New York, USA (2007)