Product Line-based customization of e-Government documents

Mª Carmen Penadés[1], Pau Martí[1], José H. Canós[1], Abel Gómez[2]
[1] ISSI - DSIC, Universitat Politècnica de Valencia
46022-Valencia, Spain
{mpenades, pmarti, jhcanos}@dsic.upv.es
[2] AtlanMod, Mines-Nantes – Inria - Lina
44307 Nantes, France
abel.gomez-llana@inria.fr

Abstract. Content personalization has been one of the major trends in recent Document Engineering Research. The "one document for n users" paradigm is being replaced by the "one user, one document" model, where the content to be delivered to a particular user is generated by some means. This is a very promising approach for e-Government, where personalized government services, including document generation, are more and more required by users. In this paper, we introduce a method to the generation of personalized documents called Document Product Lines (DPL). DPL allows generating content in domains with high variability and with high levels of reuse. We describe the basic principles underlying DPL and show its application to the e-Government field using the personalized tax statement as case study.

Keywords: Document generation, personalized e-Government services, Software Product Lines, Document Product Line.

1 Introduction and motivation

Within the high diversity of activities placed under the e-Government umbrella, document generation and delivery are key activities required by most processes. Although content management issues – that is, document classification, organization, storage and retrieval – are well solved thanks to the advances on Digital Libraries research of last decades, current content generation solutions are far from effective. Many tools are in the marketplace supporting document-related processes within e-Government environments [1, 2]. Most of them provide tools to generate documents in different ways (from scratch, from templates, via copy/paste, drag&drop…), or workflow-like utilities to automate document circulation and publishing processes. Others provide document signing utilities, and content aggregators. However, there are many issues that still need to be addressed to improve the efficiency and quality of public e-services.

In this paper, we focus on personalization aspects of document generation, with the belief that adapting public documentation to the actual needs of citizens is the best way to keep them involved in the community's governance. Customization is especially relevant in multi-national agencies such as the UN or the EU, where a given document (e.g. an EU-wide law or a UNESCO white paper) must be produced

in a high variety of languages, but those parts of a document which are not language-sensitive can be reused in all the instances. Similarly, some regulations may apply to only a subset of the member countries, making no sense to be included otherwise. This is a variation of the "one user, one document" principle as the way to maximize one reader's satisfaction by providing him/her with just the information he/she needs. We face the problem of e-government document customization as one challenge that remains unsolved in a satisfactory way. Current solutions to personalization are hard-coded in Web-based applications, or require high knowledge of low level languages such as XML. We intend to raise the abstraction level, providing tools and methods close to the application domain, hiding the internals to the final users.

Our solution borrows principles and techniques of Product Line Engineering [3] and is based on the definition of families of documents rather than one single document. A family groups a number of documents that share content in some sections while differ in others. Creating a particular document means selecting the appropriate content for that document and having tools for the automatic generation of the document from a set of content components stored in a repository. Our method, called *Document Product Lines (DPL)* [4], is supported by a tool that allows the definition of the family, the management of content components, and the generation of customized documents.

We will use an example of what we call "*sparse documents"* by analogy with Matrix Calculus (http://en.wikipedia.org/wiki/Sparse_matrix). A sparse matrix is such that most of its cells are set to zero. A *sparse document* is one whose content has been designed to a broad audience and, as a consequence, many parts of it are of no interest to a specific user. Examples of *sparse documents* are laws in several languages (one of whose readers is only interested in his/her own language), large emergency plans (with documentation relevant to different rescue teams), or tax statement forms that include sections for different types of economic activities. To illustrate the potential of DPL in the field of e-Government, we have chosen the tax statement completion as our case study. We show how the definition of a family of tax forms avoids users to find irrelevant sections in their statements, simplifying this way their completion since only those sections that are relevant for a particular taxpayer are included.

The paper is structured as follows. Section 2 outlines the DPL approach to support the generation of customized documents. Section 3 describes the tax statement as case study and how DPL is used to generate the personal income tax. Section 4 summarizes related works. Finally, Section 5 presents the conclusions.

2 An overview of DPL

DPL [4] applies product line engineering principles to the generation of documents in domains with high content variability and reuse. Central to DPL is the notion of family of documents. By family we mean a set of documents that share some common, mandatory parts while differ in other, optional parts. Every member of the family is built by assembling a set of content components.

DPL is structured as a two-stage process. First, in the *Domain Engineering* stage, a domain expert defines the characteristics of a family of documents in terms of features. There are two types of features: content features, which represent parts of

content that will eventually be components of some document of the family, and technology features, that will allow to specify the different ways a given content feature can be rendered. For instance, a content feature like "street map" could be rendered as a high resolution image, or as an URL linking to some mapping service.

After the family feature model has been defined, every content feature must be linked to a content component. Such pieces are called *InfoElements*, and their granularity varies from very simple items (words or phrases) to large documents. As a rule, an *InfoElements* represents a self-contained reusable piece of content and, as such, can be used to build different documents in the same or even in different document families. DPL assumes the existence of a repository where document components are stored and organized for reuse. The repository is explored to find existing *InfoElements*. If no *InfoElements* in the repository can satisfy a requirement specified in the feature model, a new one should either be developed or retrieved from other repositories. New *InfoElements* development requires the availability of a library of applications to create and/or modify different types of them. Finally, the document product line is generated, that is, a process that specifies how the *InfoElements* are integrated according to the different relationships defined between content features, and between content and technology features.

The second stage, *Application Engineering,* exploits the document product line defined and supports the generation of customized documents. First, the user selects the variability points that will be included in the particular document (an instance of the family) to create a configuration. At that point, an automatic process generates the final document by assembling the different *InfoElements*. The output of the product line is user-defined; we can obtain documents in different formats (PDF, HTML…).

DPLFW [5] is an implementation of DPL developed following Model Driven Engineering principles. The main elements are shown in Fig. 1. The Feature Editor is used to characterize the variability of the domain as a document feature model. The Repository contains the *InfoElements* that will be reused in the generation of the

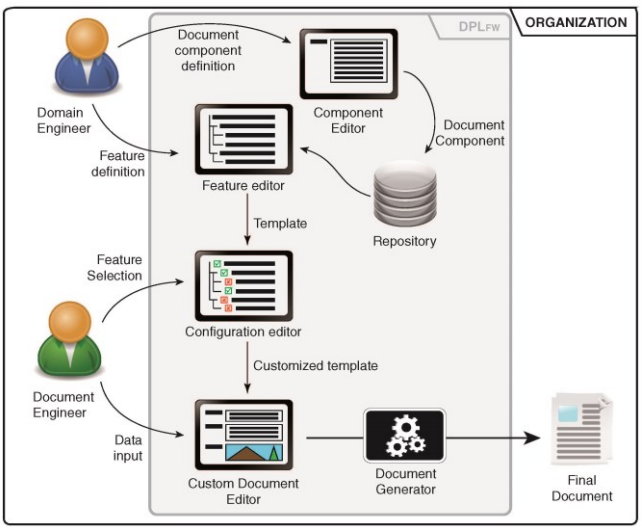


Fig. 1. An overview of the DPLfw

document. The Component Editor is used to create new *InfoElements* and add them to the Repository. All of the above elements support the Domain Engineering stage of DPL. The remaining elements are related to the Application Engineering subprocess. The Configuration Editor supports the selection of variability points. A given configuration is used to generate the Custom Document Editor. We talk about editor since, in some cases; an *InfoElements* may not be complete since some user-provided data is required. For instance, as we will see in our example, a tax statement is a set of forms that a final user has to fill in a given order. Then, the document generated will act as the editor of the true final version of the statement, which will be produced by the user following the logic of the forms generated with DPL. Finally, the Document Generator integrates these components to obtain a fully instantiated document generated in a specific format.

2.1 Variability supported in DPL

DPL supports two types of variability. On one hand, content variability is supported by the configuration editor, which allows selecting optional parts that are added to the mandatory part of the family to generate a new member. On the other hand, variable data support refers to the ability to use partially instantiated document components that contain placeholders that are replaced with values at the mail-merge style when the document is generated or, even, when a user completes the document.

The DPL document feature metamodel describes a document family as a set of document features and these are distinguished as content features (*ContentDocumentFeature or CDF*) and technological features (*TechnologyDocumentFeature or TDF*). A *CDF* can be associated to one or more *TDF*. As in a classical feature model, the features can be mandatory, optional or alternative, and may be related to other features by cross-like relationships such as "requires" or "excludes". For instance, "*f1* requires *f2*" means that every document containing the feature *f1* must also include the feature *f2*. Relationships are important to define document variability because all of these constraints bring the possibility to define that features appear in the whole or only in a group of documents of a document family. Each CDF need an *InfoElement* to provide their content which is visualized using a Disseminator [6].

Variable data in *InfoElement* is represented with the *VariableAttribute* class. The scope of a variable may be local to an *InfoElement*, or global to a feature model (that is, to the entire document). Finally, the *CriterionAttribute* is used to search and retrieve *InfoElements* when a custom document is being instantiated.

3 A case study: Customizing Tax Statement Documents

To illustrate how DPL works to generate customized documents, we use a practical case study of the e-Gov domain, namely the Personal Income Tax Statement in Spain. According to the Spanish regulations, all individuals who obtained incomes over a given threshold in Spain are obliged to file the Personal Income Tax yearly. The Spanish Tax Agency has published every year a set of tax document templates that taxpayers had to fill either by hand or using a Web interface. For the average taxpayer, the template was a large, sparse document that included pages for a variety

of cases, most of which were relevant to a subset of the taxpayers (e.g. those owning stock options), being blank in the remaining cases. In the most usual case, more than a 40% of the tax statement was not filled. In the paper version, this meant an unnecessary waste of paper; in the Web version, a number of "Skip" buttons had to be pressed; in both cases, the statement became much more complicated for taxpayers, who had sometimes to interpret the non-trivial meaning of some sections, resulting in errors in the statement that, sometimes, resulted in fines from the Tax Agency. From 2014 the Tax Statement will only be filled electronically. Also, some customization has been included in the Web application that allows hiding unnecessary content to the user; however, the customization is part of the application logic. Using the DPL approach, things work differently. Although, as in the case of the Web application, the taxpayer selects those sections which are relevant to his/her status, such a selection is not part of the tax processing application, which makes changes much easier since the application would not need to be changed.

3.1 Specifying the Personal Income Tax document family

The first step in DPL is the definition of the family of tax statements by specifying the commonalities and variation points as a document feature model (recall Fig. 1). Using the feature model editor, a domain expert can build a document feature model like the one shown in Fig. 2. There, personal data (*Taxpayer* CDF), marital status (*Marital Status* CDF), income to declare (*Income* CDF), deductions to apply (*Deductions* CDF) and returns obtained according to law (*Tax Returns* CDF) are defined as mandatory content features (labelled with an exclamation point). On the other hand, the number of children (*Children* CDF) and the residence of the taxpayer (*Resident*

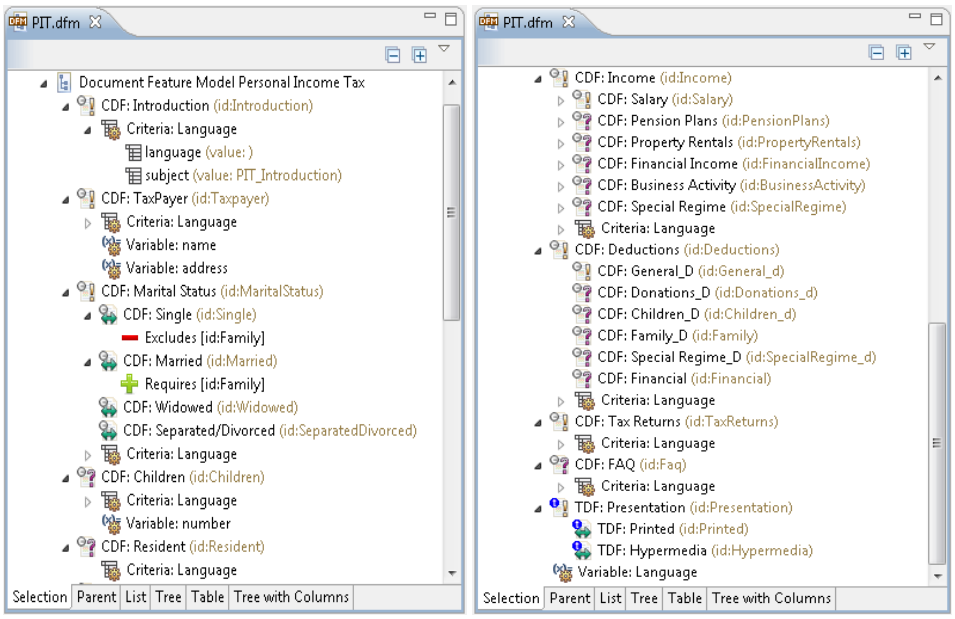


Fig. 2. Document Feature Model of the Personal Income Tax Statement family

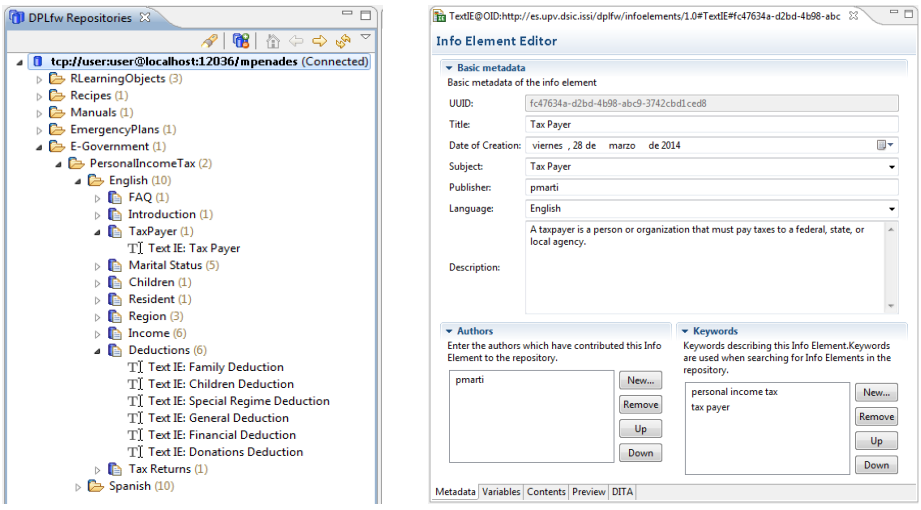
CDF) are modeled as optional CDF (labeled with either question mark or a double-head arrow in case of optional or alternative content features, respectively).

A taxpayer must declare some of the following incomes: salary derived from labour relationships (*Salary* CDF), pension plans (*Pension Plans* CDF), rentals from estate property (*Property Rentals* CDF), financial interest or dividends (*Financial Income* CDF), professional fees derived from independent activities (*Business Activity* CDF), and the special tax regime, if any (*Special Regime* CDF). Only the salary is a mandatory CDF, being the remaining ones optional. Similarly, the deductions to apply to the tax amount are represented as children of the *Deductions* CDF. The model is completed with "requires" or "excludes" relationships between CDFs that are used to comply with the rules established by law. For instance, in Fig. 3, the content feature *Single* is exclusive with regard to the *Family* feature.

There is a global variable in the model of Fig. 2, namely *Language*. It is being used to select the final language of the statement. This illustrates another application of DPL to support variable content in multilingual e-government applications. The variable declaration is complemented with its inclusion as search criteria in the model. Notice that some of the CDFs have a child named *Criteria: Language.* This means, on one hand that the actual content for the CDF will be selected at document generation time according to the value of the *Language* variable (see section 3.3).

3.2 Managing the Personal Income Tax content assets

After creating the document feature model, CDFs must be linked to *InfoElements* in the repository. The repository provides facilities to manage *InfoElements* (i.e. creating, deleting and updating) and retrieving them using metadata-based searches. Furthermore, *InfoElements* can be organized hierarchically using folders and resources (a special container which only may contain *InfoElements*). Fig. 3.a shows a view of the DPLFW's repository manager. Notice that there are several folders



(a) **(b)**

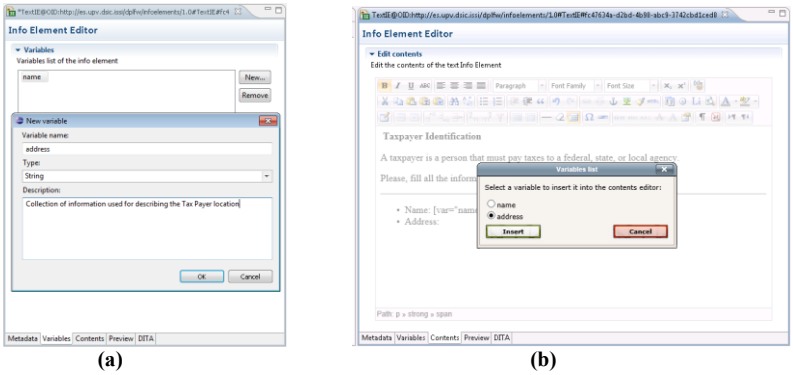
Fig. 3. a. DPL Repository Explorer **b.** Describing the *Tax Payer InfoElement* with metadata

available, one per each domain we are working on, although a DPLFW user can opt for alternative organizations (e.g. storing *InfoElements* for every domain in a single folder). There is a folder named *E-Government* whose content is partially shown in the figure. The leaves in the tree correspond to *InfoElements*, which contain the actual content that will be linked to the CDFs in the model.

When the content assets are not stored in the repository, we need to create a new *InfoElement* using the DPLFW's *InfoElement* Editor. There are several aspects of an *InfoElement* that must be defined, which can be selected by clicking on the tabs situated in the bottom left area of the editor window. Fig. 4.b shows the descriptive metadata capture window when editing the *Tax Payer InfoElement* inspired in the Dublin Core Metadata Set. To edit the content on an *InfoElement*, the contents tab in the editor provides a rich text editor (see Fig. 4.b). This editor allows defining variable data which will be instantiated in the configuration of a member of the document family. In this instance, the *Tax Payer InfoElement* contents two previously defined variables, namely name and address (see Fig. 4.a).

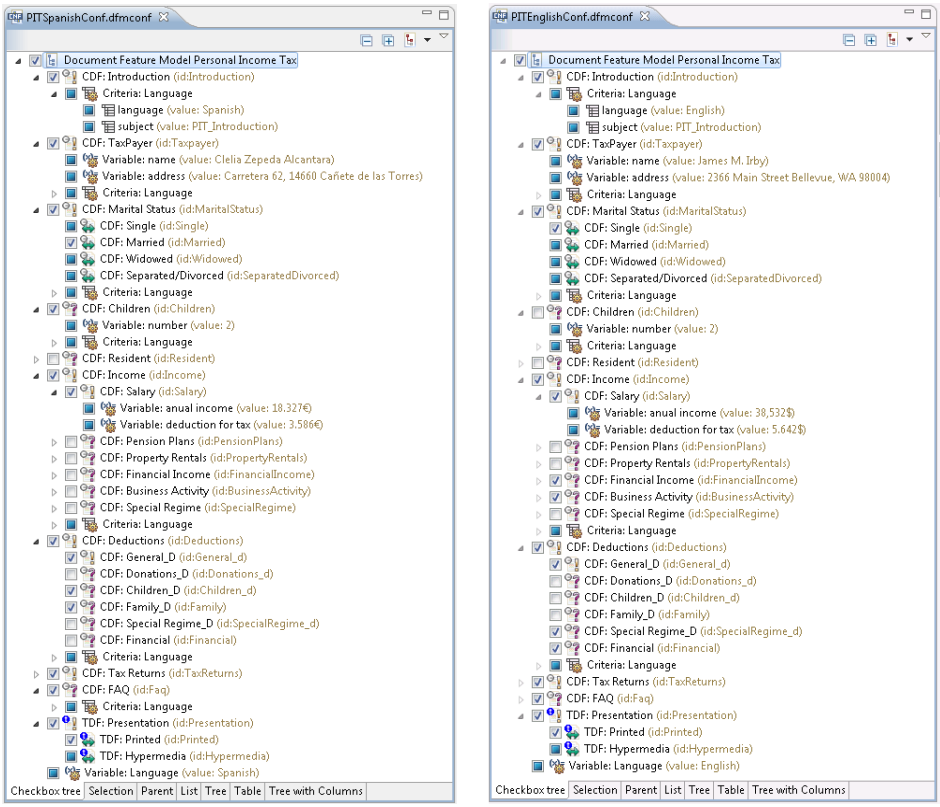
3.2 Configuration of the tax statement

Once the document family has been defined as a document feature model and a Personal Income Tax repository is available, the DPL process is ready to generate customized tax statements. This is performed in two steps. The first one is known as document configuration. A configuration consists of set features that have been selected according to the variability constraints defined by the document feature model; mandatory features are always selected. Figure 5 shows two different configurations of the tax statement. The first one, shown in Fig 5a, corresponds to the case of a woman, married and with two children whose only income is her salary. The language selection of the statement is Spanish. When the married and children features are selected, the deduction features related to family and children are automatically selected according to the document feature model. If the woman has no other deductions, such as donations, the selection of features has finished. At this point, if a given feature (or the full document) has variable attributes, a value must be introduced for each one. For instance, the values of name and address for *TaxPayer* feature have to be introduced, as well as the salary's. Additionally, if a feature has a criterion attribute, the *InfoElement* will be searched and retrieved according to its



(a) **(b)**

Fig. 4. a. Defining variable data on the *Tax Payer InfoElement* **b.** Editing their content



(a) **(b)**

Fig. 5. a. Spanish Personal Tax configuration **b.** English Personal Tax configuration

value. Next, the customized PIT document can be generated as a printed document. Another tax statement configuration is shown in Fig. 5b. In this case, the language selected is English, and the configuration corresponds to a man who has a business activity as manager and financial income, so that the deductions marked are special regime and financial.

3.4 Customized tax statement generation

When a configuration is finished, an automatic process generates a map describing the structure of the Personal Income Tax Statement configured. The map, along with the *InfoElements* retrieved, is used to generate the customized tax statement. Fig. 6 shows a screenshot of the final Personal Income Taxes obtained.

4 Related work

Some proposals have been presented to deal with the variability in the document generation in the last years. For example, proposals on Variable Data Printing (VDP) use XML and their associated technologies to implicitly represent the variability in

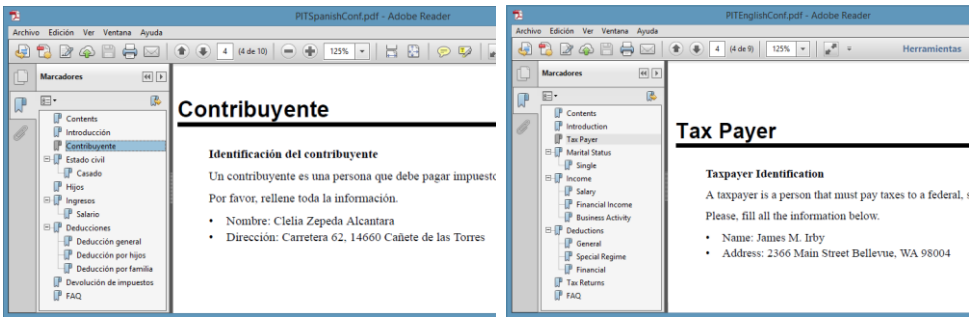


Fig. 6. The Personal Income Taxes generated

documents [7, 8]. In these cases, the variability model is usually represented by the transformations rules and the links defined among a set of XML-based document fragments. The customized document is generated using XSLT or XPath, and a high knowledge about XML is required. The VDP approaches are authoring/editing tools to generate the final XML document and not provide methodological guidance to the document engineer.

More recent proposals such as [9, 10] have explored a product line approach to model the variability explicitly. A feature model identifies the variability points from a domain-oriented perspective and is the basis to support the generation of the customized document. These approaches emphasize the definition of a document generation process based in SPL principles. However, DPLFW supports variable data and variable content in the document generation process, whereas other proposals do not support the variable data ([9]) or they do not generate the final document ([10]).

Some commercial solutions, such as HotDocs (www.hotdocs.com) or Exari (www.exari.com), provide a workflow-based suite tool for document automation. The former has a document generator server which takes a web-based *interview* and generates a document from it. The latter focuses on the domain of contract automation, and it defines a contract lifecycle to improve and automate the document generation. These solutions have similarities to DPLFW since all of them generate a final customized document, but their starting points are different – i.e. DPLFW uses feature models to identify variability points in a document family, and enforces content reuse at domain level following a product line approach.

Finally, there are some proposals that are tailored to customization of e-government documents. In [11], an adaptive hypermedia application reasons about concepts and conditions to produce web materials. The input is a generic document published in the Web by the Public Administration, and using semantic web technologies, a customization of this document is produced (containing the relevant information for a specific user only). In this scenario, the DPL approach would behave as follows: The Public Administration produces a set of content document (the *InfoElements*) and DPLFW generates the customized documents on demand. We also find some domain-oriented proposals within the e-government field: software solutions in the tax statement domain – such as TurboTax, UDoTaxes, StudioTax, QuickTax Online, etc. – create a full tax form for each user. DPLFW also generates a full tax assessment according to the configuration phase and the variable data introduced by the user.

5. Conclusions and Futher Work

The European eGovernment Action Plan 2011-2015 is among a series of initiatives that appeared all around the world to enforce the growth of e-Government. The ultimate goals are, among others, to increase the citizenship awareness by easing the access to the information, and making citizens closer to the governance by means of better public participation mechanisms. Quoting: *"Increasing effective eGovernment means that services are designed around users' needs and provide flexible and personalised ways of interacting and performing transactions with public Administrations"*. Similar initiatives have been launched in other countries such as the USA (http://www.usa. gov/) or India (http://www.indiaegov.org/), to name a few. All the above goals have document personalization at the core of any solution.

In this paper, we have shown how the DPL approach may provide the flexibility that current tools lack in terms of customization and reuse. DPLFW allows the generation of customized documents without writing a single line of code. We switch from programming to feature modelling in a domain-oriented language, hiding the internals of the document generation processes. We have used DPL in domains other than e-Government, such as emergency management, e-learning and cooking recipes generation. All of them share the same personalization and reuse requirements, and show that a general solution is more flexible than domain-oriented tools under some circumstances. We are working on the improvement of the DPLFW to support new facilities like external the document configuration process in a web application to improve the user interface.

ACKNOWLEDGMENTS. The work of M.C. Penadés and J.H. Canós is partially funded under grant TIPEx (TIN2010-19859-C03-03).

References

1. e-Gov Office. http://www.newgensoft.com/products/egovoffice
2. eGov-Suite. FabaSoft. http://www.egov-suite.com/en/dms/create.html
3. Pohl, K., B Böckle, G. and van der Linden, F., Software Product Line Engineering – Foundations, Principles, and Techniques. Springer, Berlin, Heidelberg, New York. (2005)
4. Penadés, M., Canós, J.H., Borges, M.R., Llavador, M. Document product lines: variability-driven document generation. In Proc. 10th ACM DocEng '10, NY, USA, 2010.
5. Gómez, A., Penadés, M., Canós, J.H., Borges, M. and Llavador, M. A framework for variable content document generation with multiple actors. Information and Software Technology. http://dx.doi.org/10.1016/j.infsof2013.12.006. In Press (2014)
6. Kahn, R.,Wilensky, R. A Framework for Distributed Digital Object Services. Handle: cnri.dlib/tn95 - 01. http://www.cnri.reston.va.us /k - v.html. (1995)
7. Lumley, J., Gimson R., Rees, O., A Framework for Structure, Layout & Function in Documents. In Proc. 5th ACM Symposium on DocEng (2005)
8. Quint, V., Vatton, I., Editing with Style. Proc 7th ACM Symposium on DocEng (2007)
9. Rabiser, R., Heider, W., Elsner, C., Lehofer, M., Grünbacher, P. and Schwanninger, C. A flexible approach for generating product-specific documents in product lines. In Proc. SPLC'10 , LNCS vol 6287, Springer. (2010)
10. Pichler, C. Huemer, C. Feature modeling for business document models. Proc. ACM SPLC'11., LNCS, Vo. 2, New York, NY, USA. (2011)
11. Colineau, N, Paris, C, Linder, K.V. An evaluation of tailored Web materials for Public Administration. In Proc. ACM HT'12. New York, NY, USA. (2012)