Proceedings of the Federated Conference on Computer Science and Information Systems pp. 1325–1334
DOI: 10.15439/2017F106
ISSN 2300-5963 ACSIS, Vol. 11

Documentation Management Environment for Software Product Lines

Stan Jarzabek
Faculty of Computer Science
Bialystok University of Technology, Poland
s.jarzabek@pb.edu.pl

Daniel Dan
Info-Software Systems ST Electronics Pte. Ltd.,
Singapore
ddan8807@gmail.com

Abstract--**Similar documents arise in software and business domains. Examples are user guides for different versions of a software product, contracts between vendors and clients, or legal documents. The usual practice is to capture common document formats and contents in templates that must be manually customized to a new context – often a slow, tedious, and error-prone process. We propose a method based on a proven approach developed for software reuse that simplifies and automates routine tasks involved in creating and updating families of similar documents. Our Document Management Environment (DME) provides functions to create templates capable of higher levels of document contents reuse than templates supported by word processors such as MS Word. DME allows users to designate any arbitrary document part as a template's variation point that can be customized to produce a specific document. DME automates document production by syncing inter-dependent customizations occurring at different variation points. The paper describes two "proof of concept" implementations of DME as Word add-in: The first one uses Content Control mechanism and is specific to MS Word. The second one is based on ART (Adaptive Reuse Technique), a general text manipulation method and tool, and can be used to manage similar documents in any editor that provides an access to the internal representation of documents.**

Keywords: Documentation, Reuse, Productivity, Document Generation, Templates

I. Introduction

Document Management Environment (DME) facilitates and automates reuse of documents written in WORD. DME is useful in Software Product Line (SPL) engineering [1], where we manage a family of similar software products from a common set of reusable SPL core assets such as SPL architecture shared by products, source code components, documentation, test cases, etc. SPL core assets help developers build a custom product. They play the role of templates that are reused after suitable adaptations to derive custom products. All the SPL members are similar, but each one also differs from others in client-specific features. The impact of features shows as many changes that must be applied throughout the product code and documentation.

Creation and evolution of documentation for SPL members involves much repetitive work. Developers can benefit from reuse of software documentation just as much as they benefit from reuse of other SPL assets. For example, User Guides for different SPL members are similar, but also different. The differences in User Guide versions reflect product-specific variant features implemented into some custom products, but absent from others. With understanding of commonalties and differences among User Guide versions, we can design documentation templates from which to derive custom User Guides for specific products.

Document versions typically share common structure with possible variations such as optional sections. Various document fragments may recur in variant forms in many places, within and across documents. The usual practice is to capture similarities in templates that must be copied and manually customized to create new document versions. Templates of word processors such as MS Word support reuse of text "as is". However, in reality, templates must be extensively adapted to form a new document version by changing, adding or deleting text fragments. Such adaptation is weakly supported by templates of word processors known to the authors, which often hinders documentation management, making it a slow, tedious, and error-prone process.

Our proposed approach to managing families of similar documents overcomes this limitation, providing means for flexible and semi-automated adaptation of document templates. DME automates routine tasks involved in creating and updating similar documents. The goal is to boost document management productivity.

The challenge of managing a document family is to understand what's common and what's different among document versions. The differences between any two documents (irrespective of the degree of their similarity) can be trivially expressed as a sequence of text addition/deletion operations that applied to one document produce the other one. However, such a simple-minded perspective on document differences poorly addresses human-cognitive aspects of document management. In the course of empirical studies, we identified seven basic document variation types that collectively provided a basis for building powerful document templates that are easy to grasp and could be adapted in flexible ways to form document versions (Section IV). DME provides seven text manipulation operations that handle these basic document variation types such as parameter instantiation throughout the documents, selecting text from a list of options, inserting or deleting text at designated points in documents, or repeatedly generating custom text according to a specified template. DME automatically propagates custom changes across templates in

the process of creating new document versions or in updating existing ones. Our intention was to minimize the need for manual customizations of templates, hoping to reduce effort involved in managing documents.

Given popularity of MS Word, we decided that our "proof of concept" implementation of DME would help users manage families of MS Word documents. DME extends the concept of MS Word templates and styles, to provide better controls over reuse of document structure, contents and formatting. We implemented DME user interface as an MS Word add-in.

We considered two different strategies for manipulating the internal representation of MS Word documents. The first strategy used the Content Control API. This solution was straightforward, however it was specific to Microsoft technology. In the second solution, we demonstrated how the seven basic operations could be implemented in any editor providing access to its internal textual representation of documents. For that we applied a general-purpose mechanism of ART (Adaptive Reuse Technique, http://art-processor.org/) that we developed and used as a variability management technique for software reuse. We demonstrated how the seven document variation types could be expressed in ART and illustrated document management with an example.

We believe our proposed solution to document management will be particularly useful in any software or business domain that involves large volumes of related documents, with many repetition patterns, and detailed variations propagating across documents in complex ways. DME complements capabilities of commercially available document generation systems.

In Section II, we set our assumptions regarding the document management process and explain the role of DME in that process. We introduce a working example in Section III. We discuss basic document variation types in Section IV. In Section V, we present users' perspective of DME implemented as a Word Add-in, and in Section VI we comment on implementation of DME. Section VII illustrates salient features of a general-purpose text manipulation method and tool ART. Discussion of future work, related work and conclusions end the paper.

II. Approach at a Glance

The lifecycle of DME-supported documentation processing fits into the usual SPL lifecycle, with two major phases, namely *Domain Engineering* and *Product Development* (Figure 1). *Document Architect* (or Senior Clerk) analyzes similarities and differences among subject documents (e.g., User Guides for some products), and uses DME to create a template based on text that recurs in documents in variant forms. Templates are richly parameterized, to let our tool manage document variability and reuse at coarse- and fine-granularity levels.

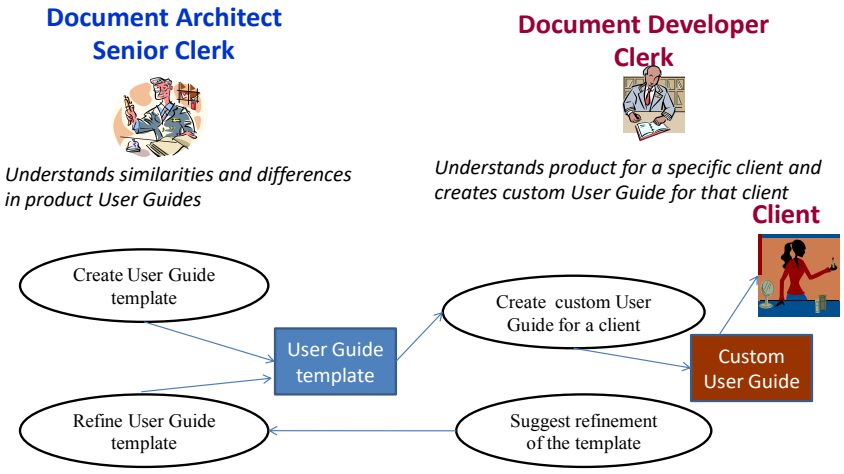

Figure 1. Managing User Guides with DME

To create a template for User Guides, we start with a User Guide for a typical product. We designate various document fragments – words, sentences, paragraphs, sections – as *variation points* that can differ from the sample document. We can also control each variation point's characteristics, such as its format and repeatability. DME converts this annotated document into a User Guide template that we then use to generate User Guides for other – similar but also different – document variants. All variation points in a DME template are formally inter-linked which allows DME to propagate customizations within and across documents.

Document Developer uses DME to customize templates and generate custom documents from them. DME propagates custom changes across documents, streamlining and automating customizations (adaptive reuse) of document variant parts. Customizations defined for one User Guide can be easily reused in creating other User Guides.

III. A Working Example

Project Collaboration Environment (PCE) is a web portal supporting software development teams in project planning and

execution. PCE facilitates sharing of project information within and across teams. In particular, PCE allows users to create and maintain records of domain entities such as projects, staff, tasks and relationships among them (e.g., tasks assigned to staff).

Suppose we developed a PCE for the mass market. There will be many PCE versions in use. All such PCEs will be similar but will also differ one form another depending on the team size, development style, and other project- and team-specific details.

A User Guide for PCE describes how to Create, Edit, Delete Staff, Project, or Task records. User Guide contains Staff-Section, Project-Section, Task-Section and in each section descriptions of relevant operations (Figure 2). The actual lists of domain entities (Staff, Project, or Task), their respective operations and the details of operation description may differ across PCEs, and those differences must be reflected in User Guides.

In the same way as PCE portals form a family of similar but also different portals (that might be supported in reuse-based way using a Software Product Line approach), PCE User Guides form a family of similar, but also different documents that could benefit much from reuse.

PCE for Agile Development: User Guide

Project Collaboration Environment is an integrated environment that supports project teams in software development. PCE stores staff, project data, facilitates project progress monitoring, communication in the team, etc.
The following sections provide detail description of operations for domain entities supported by PCE.

Staff Section

This section describes operations to manage Staff information in a Project Collaboration Environment. A Staff profile contains th following information:

Name: Full name of Staff

…

Create a new Staff

Create operation allows users to add new staff data to PCE. Once added, this new information can be manipulated by using Edit, Delete or Display operations.

Edit Staff information

Edit operation allows users to edit staff data. Once edited, this new information can be manipulated again by using Edit, Delete o Display operations.

Delete Staff record

..

Display Staff information

..

Sort Staff

..

Print an individual Staff

..

Project Section

…

Create a new Project

…

Edit Project information

…

Link a Project with another Project

…

Delete a Project link

…

Delete Project record

…

Display Project information

..

Sort Projects

.

Task Section

Figure 2. User Guide for PCE

IV. Document Variation Types

We analyzed families of similar documents such as User Guides to understand how we could capture their commonalties and differences in an intuitive way, leading to templates that would be both powerful in terms of reuse and easy to grasp for users. Differences among documents look ad hoc at first, but after analysis and conceptualization we decided that the following seven basic document Variation Types would help us achieve the goal:

Comment: Below, a 'fragment' means any arbitrarily selected segment of contiguous text in a document such as word, sentence, paragraph, section, or any part of them.

VT 1. *Parametric variations*: A parameter has the same values within a given document, but may have different values across document versions. Examples: date, section name, syntactic variations: for example spelling (English or US), whether or not we put "," before "and", etc. Parameters become placeholders in document templates.

VT 2. *Selection variation*: This kind of a difference among documents happens when at a specific point each document version should include one or more pre-defined fragments (options). Such variation point is represented by a selection construct in a template that allows the required options to be selectively included into document versions.

VT 3. *Extra fragment*: It is a fragment that appears in only small number of documents. An extra fragment may recur in many places in each of such documents. Such fragments must be also parameterized, as each of its occurrences may differ from other occurrences (in the same or in different document versions). Extra fragments do not become an integral part of templates, instead, they are included into documents when they are needed.

VT 4. *"Almost common" fragment*: It is a fragment that is a part of most (but not all) of the document versions. An "almost common" fragment may recur in many places in each of such documents. "Almost common" fragments must be parameterized. Unlike "extra" fragments that need be included on demand in small number of documents, "almost common" fragments become template defaults, simplifying template customizations. As extra and "almost common" fragments require different treatment during document management, they are distinguished as separate Variation Types.

VT 5. *Repeated section*: A section that recurs a number of times in a given document. Such section may recur different number of times in different documents. Repeated sections must be also parameterized.

VT 6. *Formatting variations*: A fragment that can be formatted using different font type or color in different documents.

VT 7. Linked documents: Large documents can be decomposed to parts that are stored in separate files. A link can be placed in a template to show how documents should be composed together. Document composition rules may be different for each document being generated from a template.

Each variation point in a DME template (i.e., a point at which a template can be customized) corresponds to one of the above document Variation Types.

V. How DME Works

DME provides seven text manipulation operations corresponding to seven basic Variation Types. In DME interface implemented as an MS Word add-in, these seven operations are accessed via menu buttons shown under "Document Management Environment" toolbar (from "Parameter" to "Link" in Figure 3).

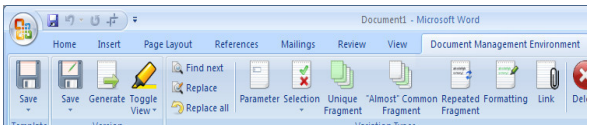

Figure 3. DME menu extending MS Word toolbar

During template creation, these buttons allow a Document Architect to create templates. The same buttons are used by Document Developer to create custom documents from templates.

A. Creating a User Guide template

As a Document Architect (Figure 1), we must first comprehend variability in a document family such as User Guides, i.e., identify common and variant document parts. Common parts become "frozen" in a template, while variant parts become *variation points* at which template can be customized.

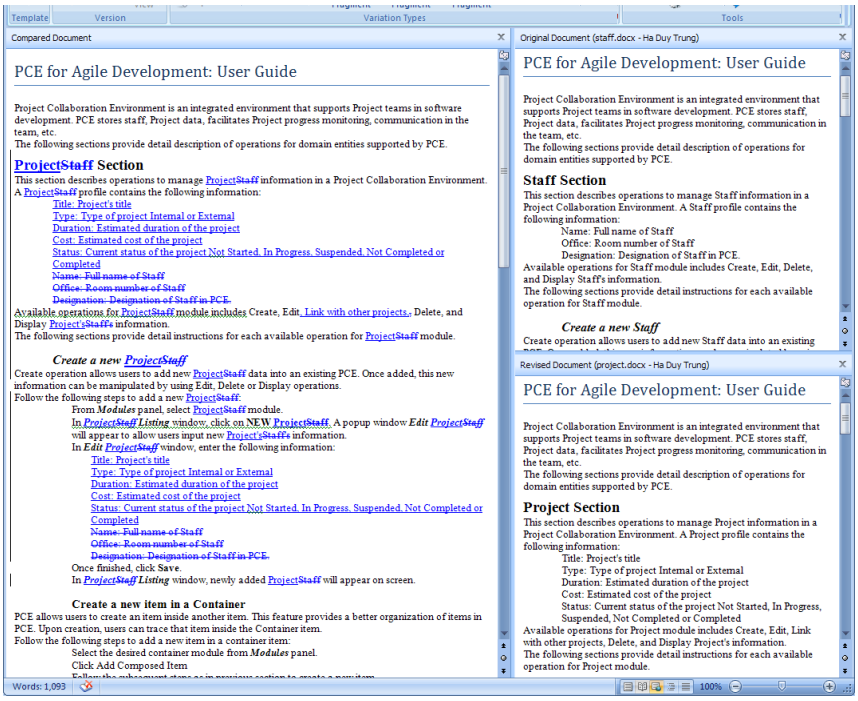

Figure 4. Staff and Project sections compared

The best is to start building a template from a "typical" User Guide, i.e., the one that is "most similar" to other User Guides. Such a User Guide already contains much of the common text that can be reused among User Guides, and also a considerable number of variant texts. The right choice of a User Guide simplifies template creation.

Terms "typical" and "most similar" are not easy to formalize, and our current approach to identifying a typical document is

rather informal (we hint at better approaches to identifying typical documents and creating templates in Section VIII on Future Work): We run MS Word's *Compare* function on existing User Guides. Differences highlighted by MS Word are candidates for variation points in a template. Of course, we should add more variation points to accommodate variant text found in yet other sections and in User Guides for yet other PCEs. Figure 4 shows common (normal font) and variant (shaded font) parts in Staff and Project sections.

Suppose we observe that sections for Staff, Project, and Task are similar to each other. We could choose to create a Section-Template first. The advantage of creating Section-Template is that Sections recur (and therefore Section-Template can be reused) within a User Guide for one PCE, as well as across User Guides for different PCEs.

At each variation point in a template we define a default value which DME uses when generating custom documents unless the user overrides the default values. DME function "Toggle View" toggles views between variation point names and their default values. Figure 5 shows a Section-Template with variation points highlighted by DME in different colors.

To convert Staff Section into a Section-Template, we position cursor on fragments highlighted by MS Word as different and click on suitable DME button to turn variant text into a template parameter – a variation point at which template can be customized. For example, we turn 'Staff' into parameter sectionName (VT1), and then qualify other document variant fragments as selection (VT2), extra fragment (VT3) or almost common fragment (VT4). Each of the above actions creates a variation point that DME highlights in different color, depending on its type. DME propagates variation points across a document using Find-Replace buttons.

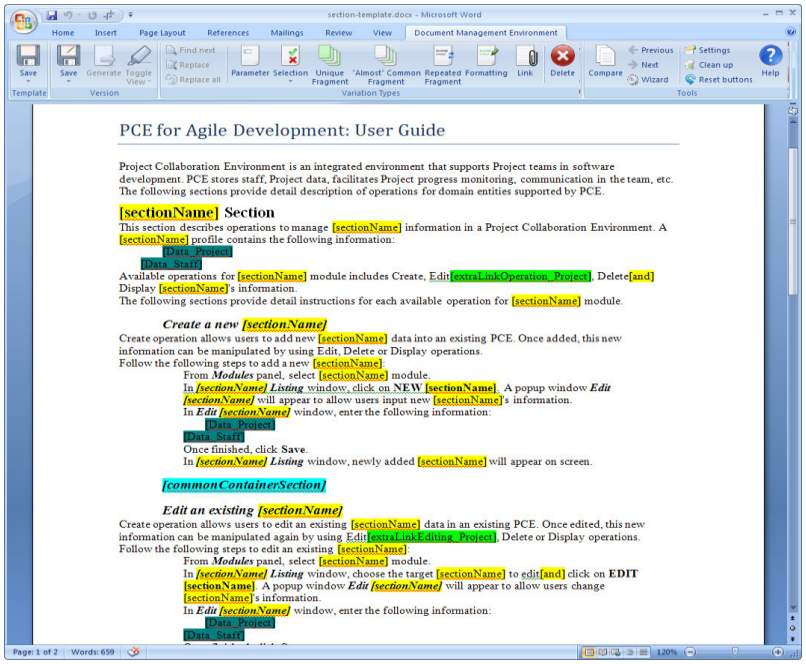

Figure 5. Section-Template created with DME

B. *Creating new User Guides from templates*

A Document Developer can customize templates to create User Guides. This is done by overriding default values at template variation points. Whenever this happens, DME automatically propagates new values to all the other relevant variation points in the current template and all templates linked to it. DME rules for propagating values across documents are carefully designed to maximize template reuse and to simplify template customization. DME function "Toggle View" toggles between template view (showing variation point names) and custom document view.

Still, template customization can be tedious. Sometimes, it may be better to start with an existing User Guide that is most similar to the one we want to create. We can now ask DME to show this document in a template view. This will show all the variation points in the document, with values assigned to them during document creation. We can accept values that suit us (no action required for that), and override the remaining values.

C. *Extra DME features*

Document fragments that recur many times should be reusable, after suitable customizations. Domain Engineer should place such reusable fragments in separate templates for inclusion with 'Link' DME button and function. DME traverses all templates linked together to compose a custom document during document generation. Any customizations are propagated to linked templates, making it possible to

consistently instantiate templates in many different ways, depending on the context.

The ability to Link templates and to propagate customizations via links during document generation is critical for scaling the DME approach. Often, many inter-related documents (e.g., a User Guide and Technical Manual) need be customized in sync one with another. Similar document fragments may spread through such documents, even though each document may be derived from different master templates. Common fragments can then be customized and included in variant forms in these documents via 'Link" connection.

We presented DME as an interactive tool in which Document Developer enters customizations via DME user interface. However, it is also possible to import customization data from a file, database or from other tools that understand document variability.

VI. Comments on Two Implementations of a Text Manipulation Mechanism in DME

A key question now is how to implement text manipulation operations corresponding to seven Variation Types described in Section IV. We implemented DME's internal text manipulation mechanism in two ways, using MS Word Content Control API, and a general-purpose variability management method and tool ART (Adaptive Reuse Technique, http://art-processor.org/).

Since *Microsoft Office 2007*, Microsoft introduced XML-based file format *Office OpenXML* for MS Office documents. Developers can programmatically manipulate documents via APIs, and enhance MS Word with new functions (Word add-ins). Content Control API released by Microsoft provided a convenient set of operations for text manipulation for our purpose. Content Control API allowed us to treat document fragments as objects, and associate tags and other meta-data with them. Content Control was giving us good control over variation points in an MS Word document. Protecting the text contained at variation points from accidental changes was not a problem either. For better performance, we implemented Document Variability Management (DVM) engine in C#, using Visual Studio Tools for Office 4.0. DVM engine provided us with text manipulation primitives sufficient for implementing DME functions. It took five person-months to develop DME as an MS Word add-in. This effort also included brainstorming and formalizing DME requirements.

The reason why we considered yet another method to handle text manipulation operations in DME was to demonstrate that our proposed approach to document management could be applied in any text editor that allows users to access its internal textual representation of a document under editing. ART is a general-purpose variability management technique that works with any information represented in textual form. Document parts are instrumented with ART commands to form highly parameterized, adaptable templates. Each of the seven Variation Types discussed in the last section could be handled with proper combination of ART commands (the reader will find details in Section VII.A). This was not surprising as ART was designed to handle much more complex variability situations.

We kept ART commands in comments embedded at designated variation points in a document. Using OpenXML API, we could extract the document text and pass it to ART Processor for executing commands. The actual variability processing with ART was completely hidden from DME users. It took six person-months to develop DME prototype in ART. This effort included the time to learn ART.

We concluded that both MS Word Content Control and ART were viable strategies for text manipulation required in reuse-based document management.

VII. Managing Document Variability in ART

Here are general rules: ART templates contain document text parameterized with ART commands. ART Processor reads templates, and outputs custom documents. ART commands are interpreted, while text is emitted to the output as is. The processing sequence is defined by ART commands. Required customizations related to the seven Variation Types are defined in the specification file called SPC which is also a start point for processing.

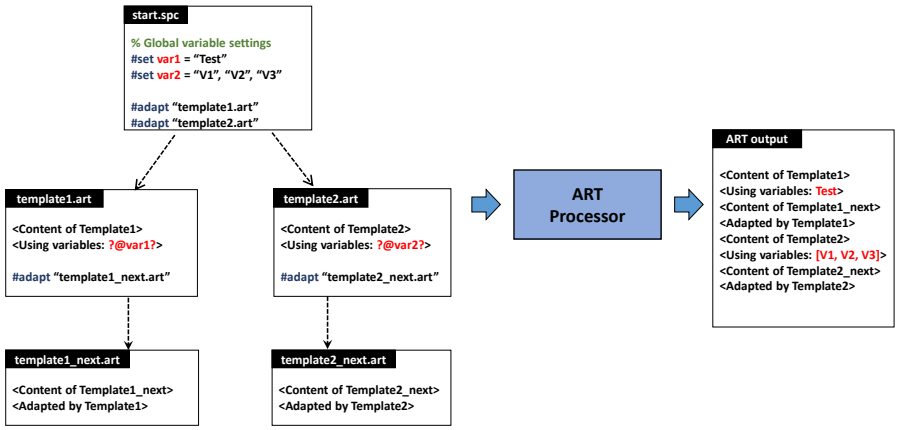

Figure 6. Processing of ART templates

ART commands in SPC, and in each subsequently processed ART template, are processed in the sequence in which they appear. When ART Processor encounters **#adapt** *f2* command in template file *f1*, it suspends processing of file *f1* and starts processing file *f2*. Once processing of file *f2* is completed, ART Processor resumes processing of file *f1* just after **#adapt** command. In that way the processing ends when the Processor reaches the end of the SPC.

Figure 6 illustrates the sequence of processing ART template files. On the right-hand-side side of the figure we see the output emitted by ART Processor after processing the files starting from SPC file and moving along the **#adapt** links.

Variables can be assigned values with **#set** command, and **?@name?** retrieves their value. Values of the variables propagate to the adapted files.

A. Document Variation Types in ART

In this section, we describe how we can express in ART the seven Variation Types discussed in Section IV.

VT 1. Parametric variations

ART variables handle parametric document variations. Command **#set SectionName = "Staff"** defines variable **SectionName** and initializes it to **"Staff"**. **?@SectionName?** refers to the value of that variable. ART Processor emits variable value to the output file (Figure 7).

```
#set SectionName = "Staff"
…
?@SectionName? Section
<Section template text>
```

Figure 7. Sample ART template tempSection.art

Typically, variables are set in SPC and ART Processor propagates their values down to all adapted files. Suppose there is a command **#set SectionName = "Project"** in SPC that adapts tempSection.art. Then, ART Processor would emit text "Project Section". Otherwise, should there be no **#set SectionName = …** command in SPC, ART Processor would emit text **"Staff Section"**, as a default value. This way of handling variable values propagation allows ART Processor to emit document variants from the same templates. This is also illustrated in Figure 6.

VT 2. Selection

ART command **#select - #option** works in a similar way as switch statements in programming languages. **#select** lets us select one of the many variant parts that should be included at a designated point in a document.

```
#select SectionName
      #option "Staff"
 <Extra section(s) for Staff>
      #endoption
      #option "Customer"
 <Extra section(s) for Customer>
      #endoption
 #endselect
```

In the above example, if the value of **SectionName** is **"Staff"**, ART Processor emits the content of the first **#option** to the output document; otherwise if the value of **SectionName** is **"Customer"**, ART Processor emits the content of the second **#option**.

VT 3. Extra Fragments

Extra fragments are managed by ART commands **#insert** into **#break.** ART Processor emits to the output document fragments or sections contained in **#insert** at points designated by matching **#break** in templates. Matching is done by names associated with **#insert** and **#break.** There are three variations of **#insert** that append, prepend or replace the content marked with matching **#break**.

SPC_Staff file:

```
#set SectionName = "Staff"
#adapt: "sectionTemplate.art"
     #insert: "additional_content"
          <Staff extra fragment content>
     #endinsert
#endadapt
```

sectionTemplate.art file

```
<template content>
#break "additional_content"
```

In the above example, ART Processor emits extra fragment for Staff when processing **#break "additional_content"**, ignoring any text contained in **#break**.

VT 4. "Almost common" fragment

Text in **#break** is a default output in case there is no matching **#insert** for a given **#break**. So "almost common" fragments are conveniently handled by making them **#break**'s defaults. In cases when the "almost common" fragment should be omitted (or replaced by other fragment), we place a suitable **#insert** matching the **#break** in SPC.

VT 5. Repeated section

ART command **#while** iterates over its body a predefined number of times emitting output accordingly.

```
#set SectionName = Staff,Project,Task
#while SectionName
     #adapt "PCE_UserGuide.art"
#endwhile
```

SectionName is a multi-value variable, and **#while** iterates over its values adapting template PCE_UserGuide.art each time in different way.

VT 6. Formatting variations

We can use ART variables or selection to handle formatting variations.

VT 7. Linked Document

If we wish to split large documents into parts, then we use command **#adapt** to compose the whole document from its parts.

In creating of ART templates, we could address any conceivable differences among documents with a single operation such as **#select** or **#insert**. The reason why we have

seven DME operations is to let the user intuitively think about document differences in terms of the seven Variation Types and map them to DME operations. Also, from our experience with using ART we know that by skillful use of these seven operations ART templates are much simpler than if we tried to address all the document Variation Types with a smaller number of operations.

B. *User Guide Template*

Document Architect creates templates using DME interface, for which she does not need to know ART or internal representation of Word documents. Document Architect's view of a template is shown in Figure 5. Figure 8 shows an internal representation of a template and SPC to create the *Staff User Guide* (Figure 4). Our templates and SPC are written in Rich Text Format (RTF) parameterized with ART commands. Consequently, ART Processor also emits RTF documents. The choice of the format is irrelevant to our ability to parameterize document with ART commends. However, RTF makes text manipulation easy, as the whole document including formatting is defined as a text file.

Document Developers define the required customizations via DME interface, from which DME generates an SPC. The internal representation in RTF instrumented with ART is not easy to read, but it is hidden from users, generated and manipulated by DME user interface operations.

SPC:

```
#set SectionName = "Staff"
#adapt "PCE_UserGuide.art"
```

PCE_UserGuide.art:

```
{\rtf1\ansi\deff0 \fs20 {\fonttbl {\f0 Times New Roman;}}
{\pard\sb120\sa240 \qc \fs40 PCE for Agile Development: User Guide \par}
Project Collaboration Environment is an integrated environment that
supports Project teams in software development…
#select SectionName
    #option "Staff"
        #adapt: "sectionTemplate.art"
            #set extraLinkOperation = ""
            #insert "data"
Name: Full name of Staff\line
Office: Room number of Staff\line
…
            #endinsert
        #endadapt
    #endoption
    #option "Project"
        #adapt: " sectionTemplate.art"
            #set extraLinkOperation = ", Link with other projects"
            #insert "data"
Title: Project's title\line
Type: Type of project Internal or External\line
…
            #endinsert
            #insert "extra_actions"
                #adapt "Project_extra_actions.art"
            #endinsert
        #endadapt
    #endoption
#endselect
```

sectionTemplate.art:

```
This section describes operations to manage ?@SectionName? Information
in a Project Collaboration Environment.\line
A ?@SectionName? profile contains the following information:\par}
#break "data"\par}
Available operations for ?@SectionName? module includes Create,
Edit?@extraLinkOperation?, Delete,
and Display ?@SectionName?'s information.\line
The following sections provide detail instructions for each available
operation for ?@SectionName? module
Create a new ?@SectionName?}
```

Figure 8. RTF document output from ART code

VIII. FUTURE WORK

The approach to managing families of similar documents presented in this paper, as well as DME are a proof of concept. We applied DME to a small number of documents. We also did usability tests to evaluate if the approach can be easily communicated to others, and if the DME's user interface was simple and intuitive. We received mostly encouraging feedback from the evaluation. Some comments allowed us to refine the DME's user interface.

Still, much work needs be done before DME becomes a production quality tool (in terms of usability and reliability) that can be applied in real situations. Some issues, such as more intelligent user interface, may considerably improve usability of DME, but require further research, as we explain below.

Our current DME prototype implements functions related to all document Variation Types except VT6 – repeated section. We are clear about internal mechanism to manage reuse of repeated sections, but still unclear about how to let DME users specify and then instantiate repeated sections in an easy way.

DME described in the last section communicates with users in terms of variant document parts such as sentences or paragraphs. Such DME can provide effective assistance in managing documents in hands of technical staff, but it is too low level for non-technical staff.

DME usability can be enhanced by allowing a Document Architect to model document variability and map it to template variation points. Feature diagrams [3] commonly used in Software Product Line [1] research and practice might be used to model document variability. Feature diagrams explicate common and variant features in an intuitive, hierarchical form that can be comprehended by non-technical staff. Document Architect can create feature diagrams for a given document family based on understanding of commonalities and differences in subject documents. When creating a custom document, Document Developer would select required variant features from the feature diagram, and DME would automatically inject relevant customizations to a template. This can eliminate (or at least substantially reduce) the need for manual customizations.

Even higher-level of interaction can be achieved by letting DME users work on documents in terms of concepts of their application domain. Domain-specific languages and their generators can be implemented using Visual Studio's DSL Toolkit. Both feature diagram-based and domain-specific mode of communication between users and DME will be more intuitive than the mode of communication described in the previous section. DME enhanced with the above features will provide higher levels of automation for document management, and will be easy to use for non-technical staff.

Here is summary of functions that we plan to implement to further enhance usability of DME:

1) DME will display a summary of customizations that occurred at specific variation points in custom documents created so far.
2) Query-based analysis will allow users to selectively retrieve information from a customization history repository.
3) DME will have a flexible rights-control system to allow/disallow different classes or users to perform certain actions. User rights will be applied to control which parts are read-only.
4) DME will accept customization data from external sources such as databases, spreadsheets, data files, requirement management databases (such as DOORS), or already mentioned feature diagrams.
5) Assistance will be provided for analysis of similarities/differences in existing document variants. If many documents already exist, identification of document variability may become difficult just using MS Word's *Compare* function (described in Section A). Document analysis tool can compute editing distance similarity metrics to help Document Architects understand document variability, build a feature model, and identify a "typical" document, suitable for template creation. This will help Document Architect to create templates.
6) DME will support Variation Point Documentation (VPD). VPD will allow users to enter/read meta-data of variation points. VPD will contain information such as variation point name, description, possible values, suggested customizations, etc. User customization rights will be contained in VPD.

IX. RELATED WORK

The presented approach has been inspired by research on software reuse. In Software Product Line (SPL) engineering [1], we manage a family of similar software products (e.g., financial products) from a common set of reusable software artifacts such as architecture shared by systems, source code components, documentation, test cases, etc. All SPL products are similar, but each one also differs from others in client-specific features. The impact of client-specific features shows as many changes that must be applied through code and documentation - a repetitive, time-consuming and error-prone process if done manually. Methods have been proposed to manage variability in software to address this problem, increasing productivity via software reuse, one of which is ART.

DME can be viewed as a template engine, a tool that generates custom output from templates and a data model. Templates represent the textual contents in parameterized form, while data model defines parameter settings. In DME, parameter settings can be either imported or the user can define them in the interactive session, via DME user interface. DME is unique in fine-granular level of customizations, and in providing template engine capability for MS Word.

Publishing tools such as Adobe FrameMaker™, DocBook™ and DITA™ generate documents and facilitate reuse of document fragments. However, these tools do not support customizations of reused fragments which is a key feature of DME approach.

Generation of documentation for Software Product Lines is addressed in [4][5][6][7]. Research tools [4][7] extend

DocBook with document variability management, while commercial tools [5][6] generate custom documents from variability models. pure:variants [6] allows one to include/exclude optional sections in MS Word documents based on selected features. DME supports optionality and yet other six document variation types (Section IV), and provides interactive means to manage document variability as well as importing of customization data.

Commercial tools implement various approaches to document generation. Many tools provide general means for document design; Q-Pulse stores document versions, provides facilities to track changes, but does not instantiate and propagate specific customizations of document templates; we do; Intelledox generates documents based on selected rules; Corticon focuses on management of companies' business rules/decisions as enterprise assets, and document generation in the context of supported business processes; Wizilegal supports end-user document creation via Web service; MS Word templates can be used to generate documents according to inputs from a database, Excel, XML or other data sources (data-driven document generation). The general goal of these tools is the same as ours – to improve productivity of some aspects of document management. However, the specific goals and capabilities of these tools differ from ours mainly in the granularity and the nature of document variability that is addressed. We have not identified a document management tool on the market that focuses on managing client-specific detailed differences among multiple document versions, which is the strength of our approach. We believe our approach complements rather than competes with existing documentation tools.

X. CONCLUSIONS

We presented a method and tool called DME for managing families of similar documents. Implemented as an MS Word add-in, DME extends the concept of MS Word templates to achieve documentation reuse with automated propagation of custom changes during custom document generation. DME supports template creation and instantiation (document generation), automated propagation of customizations across documents and ease of adoption due to seamless integration of DME into the usual document processing model (Figure 1) and MS Word. We presented two implementation strategies for handling text manipulation: The first one uses Content Control API and is specific to MS Word technology, and the second one applied general-purpose text manipulation method and tool ART.

We believe the ideas and technical approach to document management described in this paper could find applications in both software and non-software domains, where information reuse based on clear understanding of commonalties and differences among artifacts is important.

Presented here DME is a proof of concept. In future work, we will apply DME in real world projects, validate basic assumptions, and build domain-specific interfaces to enhance DME's usability.

ACKNOWLEDGEMENT

Authors thank Mr. Paul Bassett, the inventor of Frame Technology™ and a co-founder of Netron, Inc, for his generous contributions to our projects on ART and XVCL, his suggestion to work on the DME project, and insightful comments on this paper.

This study was supported by a grant S/WI/2/2013 from Bialystok University of Technology and founded from the resources for research by Ministry of Science and Higher Education.

REFERENCES

[1] Clements, P. and Northrop, L. *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2002

[2] Jarzabek, S. *Effective Software Maintenance and Evolution: Reused-based Approach*, CRC Press Taylor and Francis, 2007

[3] Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E. and Peterson, A.S., Feature-oriented domain analysis (FODA) feasibility study. *Technical Report CMU/SEI-90-TR-021*, SEI, Carnegie Mellon University, November 1990

[4] Koznov, D. and Romanovsky, K. "DocLine: A Method for Software Product Lines Documentation Development," *Programming and Comp. Soft.*, vol. 34, no. 4, pp. 216-224 (DOI: 10.1134/S0361768808040051)

[5] Krueger, C. "The BigLever Software Gears Unified Software Product Line Engineering," Proc. 12th Int Soft. Product Line Conf. Limerick, 2008, p. 353

[6] Pure:systems GmbH

[7] Rabiser, R., et al "A Flexible Approach for Generating Product-Specific Documents in Product Lines," *Proc. Int. Soft. Product Line Conf, SPLC'10*, Jeju, S. Korea, Sept. 2010, pp. 47-61 (DOI: 10.1007/978-3-642-15579-6_4)

[8] XVCL, XML-based Variant Configuration Language, a reuse method and tool, http://art-processor.org