ISSN 0361-7688, Programming and Computer Software, 2008, Vol. 34, No. 4, pp. 216–224. © Pleiades Publishing, Ltd., 2008.
Original Russian Text © D.V. Koznov, K.Yu. Romanovsky, 2008, published in Programmirovanie, 2008, Vol. 34, No. 4.

DocLine: A Method for Software Product Lines Documentation Development

D. V. Koznov and K. Yu. Romanovsky
Department of Mathematics and Mechanics, St. Petersburg State University, Universitetskii pr. 28, Staryi Petergof, St. Petersburg, 198504 Russia
e-mail: dim@dk12687.spb.edu, kromanovsky@yandex.ru

Received November 13, 2007

Abstract—The DocLine method designed for developing documentation for software product lines is presented. The method makes it possible to reuse document fragments with adaptation to a particular usage context. The method provides the Documentation Reuse Language (DRL) that has a graphical part (for designing the structure of documentation packages) and a text part (for implementing the documentation). It also describes a process for developing documentation and a toolset architecture based on the DSM approach and Eclipse GMF technology.

DOI: 10.1134/S0361768808040051

INTRODUCTION

The development of electronic documentation is a vast practical domain that covers the principles of development, maintenance, and integrity control of large document packages (for example, requirements specifications for large software systems, set of standards of a bank or large corporation, legal code), dedicated languages and tools, translation into various languages, and so on.

ACM SIGDOC[1] is one of the most active communities in the field of developing technical documentation. It organizes numerous conferences devoted to this topic. Many tools and methods for developing electronic documentation are available. Simple documentation is usually developed using general-purpose text processors, such as Microsoft Word. For complex documentation, special tools, such as FrameMaker [1] (the publishing software produced by Adobe Systems Incorporated), DocBook [2] (open source standard for developing Unix/Linux applications documentation), DITA [3] (a method for developing complex modular documentation adopted by IBM), and others are used.

A wide class of technical documents is user documentation for software. The specific features of this kind of documentation are determined by such properties of software as structural complexity, variability, existence of multiple versions, localizations, and multiple formats (user guide, support site, help system). Large document packages contain many almost identical text fragments that are used in various contexts and documents.

Reuse of various assets in software development is a rapidly developing field. The effort of software engineers and scientists is presently focused on software product lines. This paradigm proposes to use a unified development procedure for the set of software products that have some common features; this methodology is based on reusing assets in the framework of well-defined procedures; it also assumes a common marketing strategy.

Unfortunately, the development of user documentation is not distinguished as a special task in the framework of product line development, and no proper tools and methods are available. Such technologies as FrameMaker, DocBook, and DITA have some reuse capabilities. However, they do not support adaptability; that is the reused text fragments must be identical in all the contexts where they are used. This fact considerably restricts the reuse of text fragments. Bassett's technology [5] makes it possible to reuse arbitrary information represented in the electronic form and it supports adaptability. However, it has never been used for managing the reuse of documentation. Finally, there is a very promising approach to reuse based on visual modeling: this is the feature diagram method [6]. However, this method has never been used for managing the reuse of documentation.

In [7], the basic ideas of DocLine are described. This method integrates various reuse techniques into a unified technology for developing user documentation of software product lines. The Documentation Reuse Language (DRL), which is a part of DocLine, is thoroughly described in [8]. This language supports adaptive documentation reuse; it consists of two parts—graphical representation (DRL/GR) and text represen-

[1] Association for Computer Machinery's Special Interest Group on the Design of Communication.

tation (DRL/PR). The graphical representation is used to design the documentation package taking into account reusable assets, and the text representation provides for an XML implementation of reuse and for specifying markup of documents. In [9], an evaluation of DocLine as applied to developing documentation for a product line of broadcasting control systems is presented.

In this paper, we present a systematic description of the DocLine method; we distinguish three constituent parts—language, development process, and architecture of the development tools. A new feature—semantic links between document modules—is introduced into the language. Two schemes of development process are proposed—the ideal one (top-down) and the realistic (bottom-up). We also discuss documentation refactoring. Architecture of the software implementation of DocLine in Eclipse is proposed, which uses the DSM approach and the GMF technology. Integration with DocBook and multilevel diagnostics of the documentation correctness are implemented.

1. REVIEW

In this section, we consider the product line paradigm, which is the context of our study, and the technologies for developing technical documentation that include means for the reuse of parts of documents; these are DITA, DocBook, and FrameMaker. We also briefly describe the following methods and technologies used in our approach:

• Bassett's frame technology [5] designed for the adaptive reuse of electronic information of arbitrary nature;

• the approach to visual modeling of variant properties of software product lines called feature diagrams [10];

• the DSM approach that makes it possible to rapidly design and implement domain-oriented visual modeling tools [11] and the particular DSM technology called Eclipse GMF.

1.2. Software Product Lines

Ad hoc reuse of program code did not receive widespread use in software engineering. Components that can be bought and employed as ready-to-use blocks are not widespread. The amount of unique code in software development is very large, which restrains software production and increases its labor content and risks. It became clear that code reuse must be thoroughly prepared.

The product line paradigm narrows the reuse context by restricting it to a group of similar projects developed by the same company. Moreover, the code reuse must be thoroughly prepared and planned. The development process is divided into two subprocesses—the development and maintenance of reused assets, and the development of target systems based on those assets.[2]

There are several models of software product lines development. The classical heavyweight top-down procedure [15] assumes that a thorough analysis of the reuse possibilities is performed at the initial stage of the product line development, the architecture of the line and of the individual products is designed, and the shared assets are developed. The development of the particular products is started only after the first stage is completed. Ideally, the development of a particular product is reduced to the choice and configuration of the shared assets [16].

The lightweight bottom-up procedure [17] recommends starting the development with a particular product. Then, when the second product is developed, the shared assets are distinguished in the first product and used to develop the second product and to refactor the first one. The main distinction from the heavyweight approach is that the shared assets are created only when they are really needed for more than one product.

1.2. Documentation Development Tools

DocBook is the technology for developing documentation proposed in 1991 by the companies HaL Computer Systems and O'Reilly&Associates. The main idea underlying this technology is to separate the contents of a document and its formatting, which makes it possible to create a unified representation of a document (single source) and then automatically generate documentation in various formats, for example, PDF, help systems (HTMLHelp or WinHelp), and electronic documentation (HTML) [18].

DocBook includes a language that enables one mark up documents and format them. The modern version is XML-based; the schema description is open, standardized by OASIS, and available in several formats—DTD, XML Schema, and Relax NG.[3]

There are a number of tools that support the development of DocBook documents. First of all, this is a package of XSLT transformations that makes it possible to generate documents in the HTML, HTMLHelp, FO, PDF, RTF, and some other formats [19]. Many commercial XML editors (for example, XML Spy and Oxygen) support editing DocBook documents.

[2] The idea of the joint development of a group of software products was proposed by Parnas as early as in 1976 [12]. Presently, there are two research centers that study the methods of developing software product lines. These are the Software Engineering Institute (SEI) at the Carnegie Mellon University [13] and the European Software Institute [14]. There are hundreds of publications devoted to this topic, and multiple conferences are organized.

[3] Earlier, DocBook used the Standard General Markup Language (SGML) developed in the 1980s with the aid of ANSI. Nowadays, SGML gives way to XML, which is adopted in DocBook.

Presently, DocBook is widely used for developing the documentation for UNIX-like operating systems (FreeBSD, Linux) and in the open source community.

DITA Technology was proposed by IBM in 2001 for developing modular technical documentation. In DITA, the documentation is represented in the form of a set of independent topics that can be assigned a type. Thus, the reuse of large logically complete text fragments is supported in various contexts. Such kind of reuse will be called *large-block* reuse.

The document formatting language used in DITA is also based on XML (as in DocBook); it allows one not only to describe topics, but also completely define the text format. Also, DITA allows one to organize the reuse of small text fragments such as words, phrases, terms, phrase fragments, and so on (such kind of reuse will be called *fine-grained* reuse).

The DITA development tools include a package of transformations from the DITA format to various output formats. The DITA format is standardized by OASIS and is supported by many XML editors (for example, XML Spy, Oxygen, Frame Maker, and others). The standard DITA tools are experimental; they are inferior to the DocBook tools in what concerns document formatting. Nevertheless, DITA is successfully used in some large companies (for example, IBM) to prepare bulky document packages containing large amounts of monotonous information.

FrameMaker is produced by Adobe [20]; it is a universal WYSIWIG[4] word processor. Due to its stability in working with large document packages and rich support of document markup, FrameMaker became a de facto standard in the development of technical documentation. FrameMaker supports structural document editing based on an analog of XML schemata using a built-in language. DocBook, DITA, and other XML markup languages can be used as well. FrameMaker supports extensions, and many commercial add-ins are available; for example, Adobe Robohelp [21], which makes it possible to represent the documentation in the WinHelp and HTMLHelp formats. FrameMaker has powerful means for reuse; however, it does not support adaptive reuse.

1.3. Related Approaches

Bassett's technology [5] is a universal approach to the reuse of arbitrary information. It was developed by Bassett in the University of York in Toronto (Canada) in the 1980s. Bassett analyzed various approaches to information reuse and their practical application. He concluded that the reused modules should be different when used in different contexts. This is what is called *adaptive* reuse. Information is represented in the form of an archetype and a set of deltas. An *archetype* is a model or template that integrates common features in different information objects. *Deltas* represent differences in the objects.

Bassett's technology allows one to reuse arbitrary data even if their format does not allow for such a possibility. For that purpose, a special labeling is performed on top of the native data structure that makes it possible to define archetypes, which are called frames in this case, and describe the modifications required in various usage contexts (deltas).

There are several variants of the markup language implementing the frame method. All these variants have a common set of basic constructs—frame (archetype); extension points, where the archetype may be modified); and frame adaptation (delta). The latest implementation of the frame language is the project of the XVCL language developed at the University of Singapore [22] from the beginning of the 2000s. This implementation assumes the use of XML as a basis for the markup language. XVCL is designed for organizing a platform-independent reuse in software applications. The most significant results were achieved for Java applications [23]. XVCL was also used for the reuse of UML specifications [24].

A well-known implementation of Bassett's technology is Netron Fusion produced by Netron Inc. This product designed for restructuring COBOL programs appeared in the 1980s and was a considerable commercial success.[5]

Feature Diagrams were proposed in 1990 as a part of the Feature-Oriented Domain Analysis (FODA) technology [6] by a group of researchers in the Software Engineering Institute (Carnegie Mellon University) headed by Kyo Kang. The main purpose of such diagrams is to formalize and visually represent the archetypes and deltas in software product lines. The key concept is the feature, which is a feature of the system recognized by the user or system developer. All the functional capabilities are represented as a set of features that are related to each other by usage relations. A usage relation may be optional, which means that it holds only in some of the members of the product line. Relations may be joined into groups; moreover, additional constraints may be imposed on the groups; for example, it may be required that each end product may include not more than one relation from a certain group.

An example of the feature diagram is shown in Fig. 1. By the Unitel telephone set, we mean a family of devices that can support various functions. All such devices are similar, but they have some differences. This diagram shows that all telephone sets must support incoming calls or outgoing calls (or both), may have

[4] What You See Is What You Get type of editors that allow one to see the ultimate representation of a document in the course of editing.

[5] Netron [25] is a Canadian company producing tools for organizing reuse in the course of software development. The successful application of these tools is partly caused by the fact that COBOL does not include means for code structuring. A common practice in COBOL programming was the development of long modules (thousands of code lines) with an intricate structure.

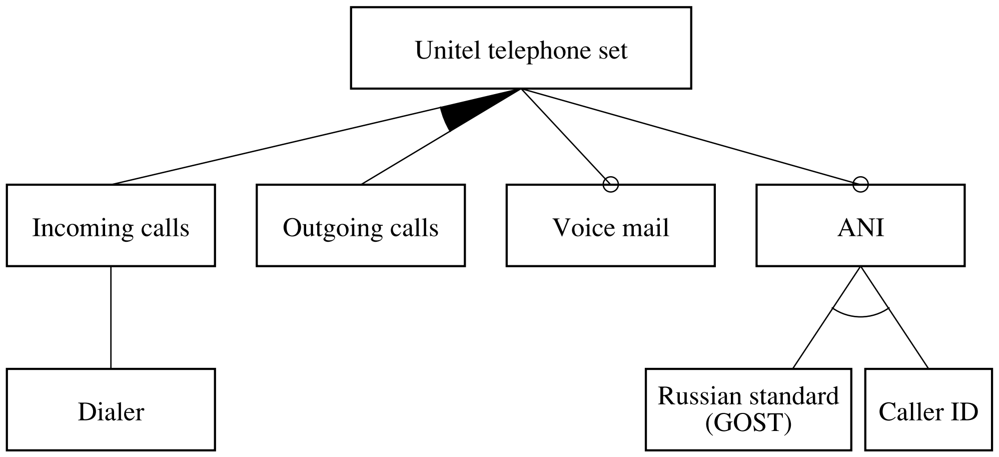

Fig. 1. Example of a feature diagram.

voice mail and automatic number identification (ANI); in turn, the latter may adhere to one of two standards—the Russian State Standard (GOST) or the international standard Caller ID.

Several implementations of feature diagrams are available;[6] however, all of them are research projects.

DSM Approach and Eclipse GMF Technology. The Domain-Specific Modeling (DSM) is an approach that appeared in the end of the 1990s; presently, it is developed and supported by Microsoft, IBM, Borland, and others. The main idea of this approach is to restrict the application domain (up to the specific features of a particular software project) and thus increase the abstraction level of visual languages [28].

Presently, various technologies that support the DSM are rapidly developing. One of them is the Graphical Modeling Framework (GMF). It is included in the Eclipse platform and is developed by IBM, Borland, HP, BEA, and Red Hat. Eclipse GFM enables one, given a metamodel of a language and a set of its descriptions, to generate a repository, graphical editors, and saving and restoring procedures for models and diagrams. The tools thus generated are integrated into the Eclipse platform, which provides a conventional user interface and the ability to interact with many free and commercial plug-ins.

1.4. Conclusions

There is a number of well-developed approaches to documentation design; a part of them supports the reuse of text fragments (DITA, DocBook, and FrameMaker). A general method for adaptive reuse (Bassett's technology) is available; however, it is not adapted for the development of documentation. Also, there are techniques for modeling the common and different features of systems that can be applied for the development of documentation (feature diagrams). However, no unified methods for documentation development are available that could support designing complex documentation and support adaptive reuse. These aspects are the key ones for developing the documentation of software product lines.

To implement DocLine, we chose the open-source available technologies DocBook (to mark up texts and publication of documents in target formats) and Eclipse GMF (for implementing graphical tools for designing the structure of complex documentation packages). Thus, we do not have to develop the technology from scratch; we rather focus on the main thing—the support of adaptive reuse of documentation.

2. DOCUMENTATION REUSE LANGUAGE (DRL)

The Documentation Reuse Language (DRL), as well as the Specification and Description Language (SDL) [29], includes two notations—the graphical and the text ones. By analogy with SDL, they are called GR (Graphic Representation) and PR (Phrase Representation). A scheme of DRL is shown in Fig. 2.

The documentation design tools (DRL/GR) are intended for specifying the documentation structure—description of the product line structure, types of documents, their links with particular products, and for designing the large-block reuse.

The reuse development tools (DRL/PR) allow one to specify various aspects of the reuse of text fragments in detail.

The documentation development tools allow one to format and mark up documents.

2.1. Documentation Design Means

DRL/GR includes the following types of diagrams for designing documentation:

[6] For example, Feature Modeling Plug-in [26] and XFeature [27].

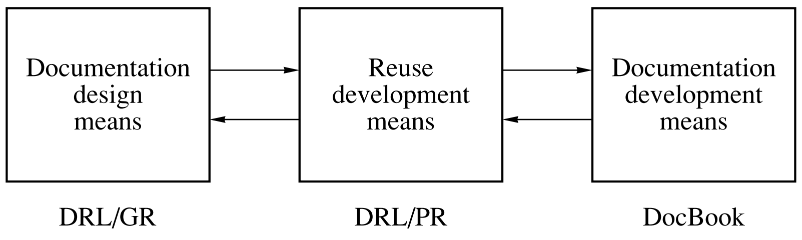

Fig. 2. Scheme of the DRL.

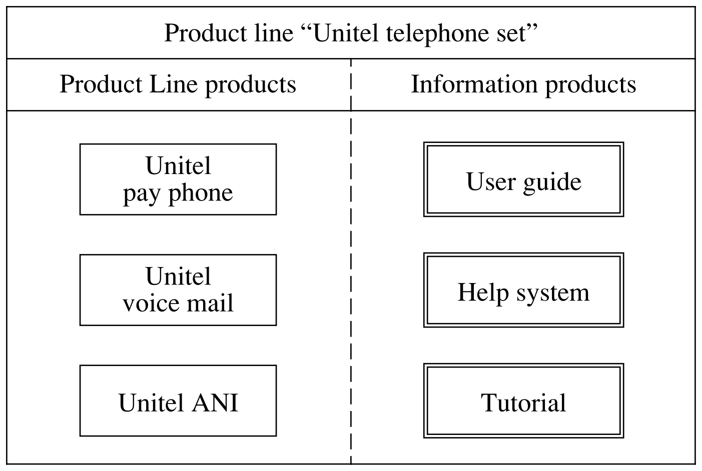

Fig. 3. Main diagram.

• The main diagram represents the list of products in the product line and the structure of the documentation of a typical product.

• The variability diagram represents the set of reused documentation fragments and rules for composing documents from them.

• The product diagram adapts the variability diagram for a particular product.

An example of the main diagram is shown in Fig. 3. In the left panel, it represents the product line called *Unitel telephone set,* which includes three types of devices—Unitel pay phone (which supports only outgoing calls), Unitel voice mail (a home telephone set that has the voice mail feature), and Unitel ANI (a home telephone set that has the ANI feature). The typical documentation package for this line is represented in the right panel of the diagram. It includes a user guide, a help system, and a tutorial. The parts of this package are called *information products*; actually, they are templates for target documents. Each particular product in the line can include certain information products, which is denoted by a line connecting the product and the information product.

An example of the variability diagram is shown in Fig. 4. This figure shows the structure of the user guide. This information product consists of the following parts called *information elements* (IE)—outgoing calls module documentation, incoming calls module documentation, voice mail module documentation, and ANI module documentation, which can be decomposed further. An IE is an isolated fragment of the documentation prepared for reuse; it may be a syntactically isolated module (such as a chapter or section) or a plain text fragment. IEs are the basic units for large-block reuse.

Links in variability diagrams indicate the hierarchy of catalog aggregation of IEs. Here, we use a modified feature diagram notation [10].

In addition to hierarchical links, variability diagrams may include semantic links between arbitrary IEs. Semantic links have no exact semantic meaning; they are used for specifying an arbitrary relation between two IEs. The particular interpretation of a semantic link depends on the situation. Such links are convenient for maintaining documents—they indicate that two IEs contain the same information that is not formally described. Therefore, if one IE is changed, the other must be changed as well.

An example of the variability diagram is shown in Fig. 5. The information product called *User Guide* is adapted for the product *Unitel pay phone* (Fig. 5a), and for the product *Unitel voice mail* (Fig. 5b).

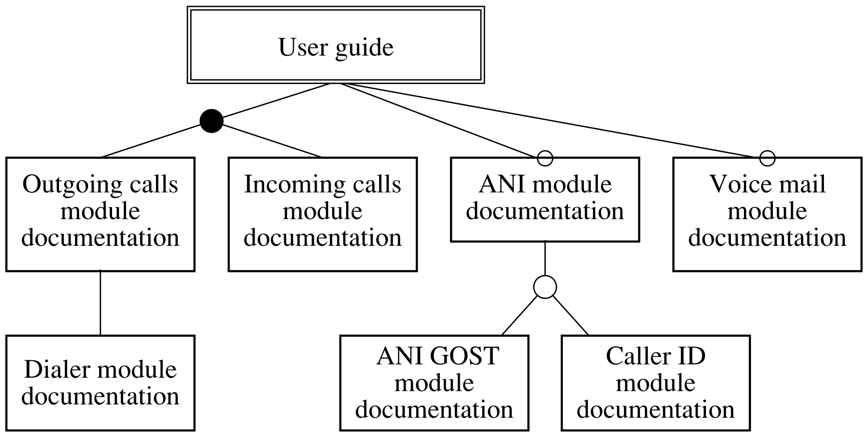
User guide
Outgoing calls module documentation
Incoming calls module documentation
ANI module documentation
Voice mail module documentation
Dialer module documentation
ANI GOST module documentation
Caller ID module documentation

Fig. 4. Variability diagram.

In these diagrams, all the uncertainties of the original variability diagram are resolved. For example, for the pay phone (Fig. 5a), only the documentation for the outgoing calls module is selected, and the documentation for the incoming calls module is discarded.

2.2. Reuse Development Means

DRL/GR allows one to create a draft of the large-block reuse, and it is specified in detail using DRL/PR. For example, DRL/GR only specifies where an IE is included and the IEs it contains, while DRL/PR defines the parameters and extension points. For that purpose, Bassett's frame technology is used. Actually, IEs play the role of archetypes. IEs are adapted using a special *adapter* construct. This construct differs from the similar construct used in the Bassett's frame technology in that it makes it possible to separate design and specialization; namely, at the design stage, it is only specified that one IE is included in another. At the specialization stage, an adapter is used to specify the values of the parameters and to manipulate extension points in the context of a particular inclusion (for different inclusions of the same IE, different adapters may be specified).

The fine-grained reuse is implemented in DRL using dictionaries and catalogs. A *dictionary* is a set of pairs (name, value). At an arbitrary point of any document, one can include a value from the dictionary by specifying its name. The dictionary can include such elements as the product name, version, the set of supported operating systems etc. A *catalog* is a set of tuples (name, attribute 1, attribute 2, …). The catalog is associated with a set of element representation templates describing the combinations of attributes when the catalog element is included in the text. A typical example of a catalog is a set of user interface instructions. Each instruction is described by its name, hint, icon, hot key, menu item, and so on. The adaptability of fine-grained reuse is achieved using the explicit specification of usage variants (the list of variants may be extended as required). A typical scenario of using the variability of the fine-grained reuse is the application of different grammar forms (gender, case, number and the like) of the same word or phrase. This is implemented using the catalog and listing the variants as attributes of a catalog element. Then, a set of representation templates is created for each grammar form.

Another adaptability mechanism for the fine-grained reuse is provided by product-dependent dictionaries and catalogs. In the documentation of a particular product, dictionaries and catalogs are defined that have the same names as those in the documentation of the entire line. When references to the dictionary or catalog elements are resolved, the dictionaries and catalogs of the current product are first searched; if no appropriate element is found, the search domain is extended to the documentation of the entire line (documentation of the other products in the line is ignored).

2.3. Documentation Development Tools

In DocLine, the utilitarian functionality, such as design of the text (chapters, sections, subsections, figures, etc.) is implemented by integrating with the DocBook technology. A text in DocBook may be included into any DRL construct. In other words, the original plain text is first marked up using DRL, and the resulting fragments are marked up using DocBook. When the document is published, the set of DRL descriptions is translated into the pure DocBook text; then, it is transformed into various target formats (HTML, PDF, etc.) using DocBook tools.

3. PROCESS

The i d e a l p r o c e d u r e of developing the user documentation of a product line starts from a thorough analysis of reuse possibilities, designing the documentation structure, and describing the reused fragments. Then, when the shared documentation assets are created, they are used to generate the documentation for

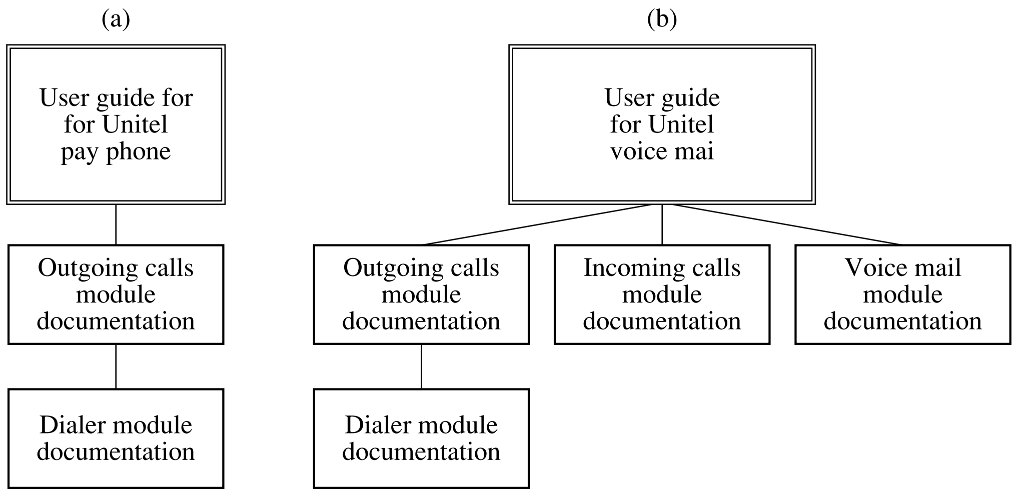
(a)
User guide for for Unitel pay phone
Outgoing calls module documentation
Dialer module documentation
(b)
User guide for Unitel voice mai
Outgoing calls module documentation
Incoming calls module documentation
Voice mail module documentation
Dialer module documentation

Fig. 5. Example of product diagrams.

particular products. Under such an approach, the first final result requires considerable preparatory work; for that reason, this approach is called heavyweight [15]. It can be used when the composition of the product line is known in advance and when there are sufficient resources (people, time, money, etc.) for a thorough analysis of the domain.

R e a l i s t i c p r o c e d u r e. In practice, many software development companies assign modest resources for documentation development. In this case and when the development starts with a single product (for example, in the framework of the lightweight procedure [17]), the scheme represented in Fig. 6 is more appropriate.

The documentation development begins with writing the documentation for the first product. Then, when the documentation for the subsequent products is developed, the possibilities of reuse are analyzed, the document package is designed, shared fragments are selected, and the documentation for the next product is developed using the shared assets.

D o c u m e n t a t i o n r e f a c t o r i n g. The creation of reused document fragments require that the documentation be refactored. In the case of the lightweight procedure, this is a must. Refactoring is also useful in the case of the heavyweight procedure. DocLine includes the following types of refactoring: extraction of information products, information elements, conditional block, dictionary elements, catalog elements, representation templates of the catalog elements, parameters, extension points, and adapters; renaming of information products, information elements, dictionary elements, catalog elements, and representation templates of the catalog elements.

4. DEVELOPMENT TOOLS

The proposed architecture of the development tools is illustrated in Fig. 7.

The graphical editor DRL/GR is a visual editor that enables one to design the documentation structure in terms of DRL/GR; the editor supports all the three types of DRL/GR diagrams. It is based on Eclipse/GMF.

DRL/PR is a text XML editor that supports editing texts written in DRL/PR, importing documents from external sources, and documentation refactoring.

The roundtrip development manager integrates the text and the graphical editors and maintains the integrity of the documentation. In fact, the automatic roundtrip procedure [30] is implemented.

The error diagnostics module checks the correctness of texts written in DRL and performs a multistep error diagnostics. Checking is performed in several steps. It is checked whether the text adheres to the DRL syntax; referential integrity and the correctness of the generated DocBook text are also checked.

The translation module compiles DRL texts into the DocBook format and checks the correctness of the generated DocBook texts at the request of the diagnostics module.

The generation module uses the tools provided by DocBook to produce target documents in the HTML and PDF formats.

The current state of the project is as follows. Presently, a version on the Java/Eclipse platform is implemented (the first version was developed in the Microsoft Visual Studio environment; the first version was used to develop the documentation for a family of television broadcasting control systems [9]). This version includes a graphical editor, an editor of the subset of XML (DRL/PR) used in the product, and a compiler into DocBook. This year, the technology is supposed to

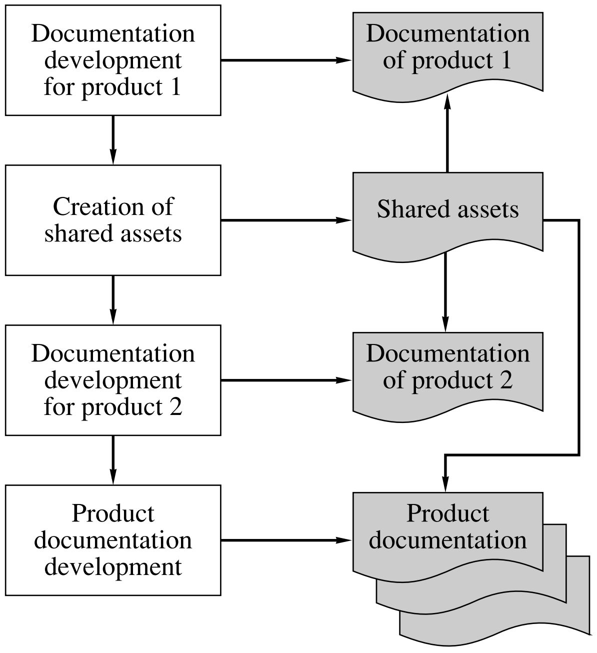
Documentation development for product 1
Documentation of product 1
Creation of shared assets
Shared assets
Documentation development for product 2
Documentation of product 2
Product documentation development
Product documentation

Fig. 6. Lightweight documentation development process.

be stabilized and used in *LANIT-Terkom* to develop the documentation for a family of telephone stations.

All the technologies used in the project are open source ones (Eclipse, DocBook, and others). Correspondingly, the product is supposed to be distributed in the framework of the open source license agreement.

5. APPLICATION OF THE METHOD

Is the approach described above practical? The main issue is the use of XML technologies in documentation development. Indeed, technical writers often have no engineering training; hence, it is not an easy task to teach them to use modern documentation development techniques.

However, many technical writers master XML technologies.[7] The advantages of using XML in medium-size and large companies outweigh the cost of their adoption (for example, see [32]). DocBook and DITA, which are based on XML, are adopted and successfully used in dozens of companies and projects, for example, in IBM, PostgreSQL, Adobe, Sun, Cray Inc. [33], installation packages of Unix-like systems, graphical shells (GNOME, KDE) [34], and others.

DocLine is an XML technology and an extension of DocBook. Therefore, it is in the mainstream of the modern technologies for documentation development. Moreover, DocLine adds some useful features for adaptive reuse that require, in the case of pure DocBook, an advanced knowledge of XML or are not available at all.

[7] A discussion of using XML by technical writers can be found at the forum of PhiloSoft [31].

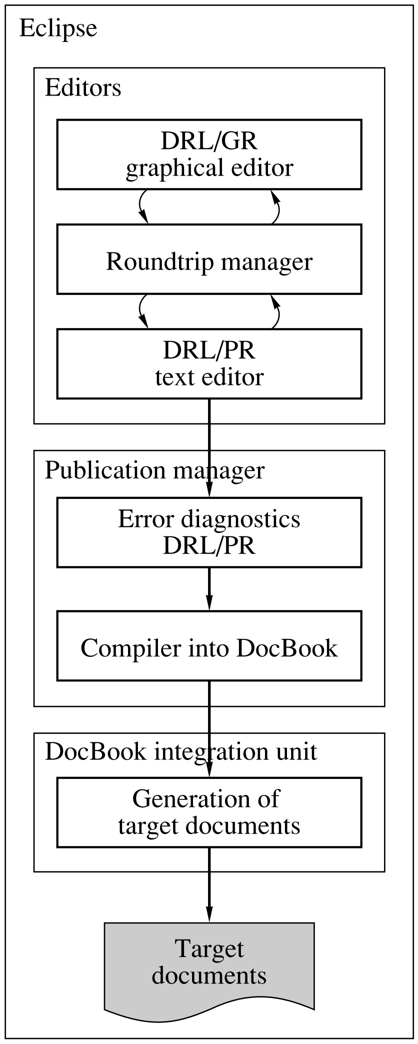

Fig. 7. Scheme of the DocLine tool.

The visual design language for complex document packages considerably facilitates the use of XML, and the roundtrip procedure applied for the joint development of diagrams and XML texts enables one to use diagrams later in the project development.

The advantages of DocLine are most clear in the following situations:

• The documentation parts of different products have much in common but the reused fragments have differences.

• Documents must be maintained; in particular, new versions of the documents are to be produced, and new documents for new products in the line must be developed.

DocLine was used to develop the documentation for a line of television broadcasting control systems [9]. The use of DocLine helped to reuse text fragments. However, it became clear that a convenient XML editor is needed to enable technical writers to work with XML.

6. CONCLUSIONS

DocLines is mainly designed for developing user documentation. However with minor adaptations, it can be used for documenting processes (for example, in the framework of the Capability Maturity Model) and for other document families.

We intend to develop DocLine in the direction of document refactoring and applications to real-life industrial-size projects.

ACKNOWLEDGMENTS

This work was supported by the Russian Foundation for Basic Research, project no. 08-01-00716-a and by the Intel Corporation.

We are grateful to the management of the System Programming and Information Science Laboratory, which was organized at the Department of Mathematics and Mechanics of the St. Petersburg State University with the support from the Intel Corporation, for their help in organizing training and research projects the results of which were used in this study.

REFERENCES

1. Margues, M., Single-sourcing with FrameMaker, *TECHWR-L Magazine Online*, http://www.techwr-1.com/techwhirl/magazine/technical/singlsourcing.html.
2. Walsh, N. and Muellner, L., DocBook: *The Definitive Guide*, O'Reilly, 1999.
3. Day, D., Priestley, M., and Schell, D.A., Introduction to the Darwin Information Typing Architecture—Toward Portable Technical Information, http://www-106.ibm.com/developerworks/xml/library/x-dita1/.
4. Clements, P. and Northrop, L., *Software Product Lines: Practices and Patterns*, Boston, Mass.: Addison-Wesley, 2002.
5. Bassett, P., *Framing Software Reuse—Lessons from Real World*, Prentice Hall, 1996.
6. Kang, K., Cohen, S., Hess, J., Novak, J., et al., Feature-Oriented Domain Analysis (FODA): Feasibility Study, *Techn. Report of Software Engineering Institute, Carnegie Mellon University*, Pittsburg, 1990, CMU/SEI-90-TR-21.
7. Romanovsky, K.Yu., A Method for Developing Documentation for Software Product Lines, *System Programming*, Terekhov, A.N. and Bulychev, D.Yu., Eds., issue 2, St. Petersburg: Izd. St. Perersburg. Gos. Univ., 2007, pp. 191–218.
8. Romanovsky, K.Yu. and Koznov, D.V., DRL: A Language for Designing and Developing Documentation for Software Product Lines, *Vestn. S. Peterb. Univ., Ser.* 10.2007, issue 4, pp. 110–122.
9. Koznov, D.V., Peregudov, A.F., Romanovsky, K.Yu., Kashin., A., and Timofeev, A., Experience in Using UML for Developing Technical Documentation, Terekhov, A.N. and Bulychev, D.Yu., Eds., issue 1, St. Petersburg: Izd. St. Perersburg. Gos. Univ., 2005, pp. 18–36.
10. Czarnecki, K. and Eisenecker, U., *Generative Programming: Methods, Tools, and Applications*, Reading, Mass.: Addison-Wesley, 2000.
11. Greenfield, J., Short, K., Cook, S., and Kent, S., *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, Indianapolis: Wiley, 2004.
12. Parnas, D. On the Design and Development of Program Families, *IEEE Trans. Software Eng.,* 1976, March, pp. 1–9.
13. Software Engineering Institute (SEI), http://www.sei.cmu.edu/plp.
14. European Software Institute (ESI), http://www.esi.es.
15. Krueger, C., Eliminating the Adoption Barrier, *IEEE Software*, 2002, July–August, pp. 29–31.
16. Krueger, C., New Methods in Software Product Line Practice, *Commun. ACM*, 2006, vol. 49, no. 12, pp. 37–40.
17. Clements, P., Being Proactive Pays Off, *IEEE Software*, 2002, July–August, pp. 28–31.
18. Rockley, A., Kostur, P., and Manning, S., *Managing Enterprise Content: A Unified Content Strategy*, New Riders, 2002.
19. DocBook Project, http://sourceforge.net/projects/docbook.
20. Adobe Frame Maker, http://www.adobe.com/products/framemaker/.
21. Adobe RoboHelp, http://www.adobe.com/products/robohelp/.
22. Jarzabek, S., Bassett, P., Zhang, H., and Zhang, W., XVCL: XML-based Variant Configuration Language, *Proc. of the 25th Int. Conf. on Software Engineering*, 2003, pp. 810–811.
23. Yang, J. and Jarzabek, S., Applying a Generative Technique for Enhanced Reuse on J2EE Platform, *4th Int. Conf. on Generative Programming and Component Enegineering, GPCE'05*, Tallinn, 2005, pp. 237–255.
24. Jarzabek, S. and Zhang, H., XML-based Method and Tool for Handling Variant Requirements in Domain Models, *Proc. of the 5th Int. Symp. on Requirements Engineering, RE'01* Toronto, 2001, IEEE Press, 2001, pp. 166–173.
25. Netron, http://www.netron.com/.
26. Feature Modeling Plug-in, https://sourceforge.net/projects/fmp/.
27. XFeature, http://www.pnp-software.com/XFeature/.
28. Preface, *J. Visual Lang. Comput.*, 2004, vol. 15, pp. 207–209.
29. ITU-T Recommendation Z.100: Specification and Description Language (SDL), 1999.
30. Assmann, U., Automatic Roundtrip Engineering, *Workshop on Software Composition*, Warsaw, 2003, *Electronic Notes Comput. Sci.*, 2003, vol. 82, no. 5, pp. 1–9.
31. PhiloSoft Forum, http://forum.philo-soft.ru/cgi-bin/forum/ikonboard.cgi.
32. Albing, B., Combining Human-Authored and Machine-Generated Software Product Documentation, *Proc. of the Professional Communication Conference, 2003*, IEEE Int. Volume, 2003, pp. 6–11.
33. List of companies using DITA, http://dita.xml.org/deployments.
34. List of Companies Using DocBook, http://wiki.docbook.org/topic/WhoUsesDocBook.