MODDO - A TAILORED DOCUMENTATION SYSTEM FOR MODEL-DRIVEN SOFTWARE DEVELOPMENT

Matthias Heinrich
SAP AG, SAP Research, Dresden, Germany
matthias.heinrich@sap.com

Antje Boehm-Peters
SAP AG, SAP Research, Dresden, Germany
antje.boehm-peters@sap.com

Martin Knechtel
SAP AG, SAP Research, Dresden, Germany
martin.knechtel@sap.com

ABSTRACT

In the last decade Model-Driven Software Development (MDSD) has become an established software engineering discipline. The new approach dramatically changed the entire software development lifecycle. However, the documentation process was not adapted and stuck with the old paradigms. In this paper, we propose a model-driven documentation system which is adapted to the MDSD lifecycle and therefore exploits synergies coming along with the alignment of software development and software documentation. Furthermore, the proposed documentation system builds upon the Internet as their major provisioning platform.

KEYWORDS

Software Documentation, Model-Driven Software Development

1. INTRODUCTION

Delivering high-quality software includes providing software documentation. Despite the close coupling between software development and software documentation, both processes used to be isolated. A first step towards weaving the documentation process into the development process was pioneered by Donald E. Knuth who introduced the so called *literate programming* [Knuth 1992]. The objective of literate programming is to combine source code with human-readable documentation within one file. Nowadays, this technique is applied in many mainstream programming languages (e.g. Java, C#).

Moving from traditional software development to MDSD made the literate programming technique obsolete since models themselves serve as technical documentation [Kleppe et al. 2003]. However, user and marketing documentation are still handcrafted in parallel to the software development lifecycle.

Therefore, we enhance MDSD in order to support the creation of all types of documentation. That is achieved by the Model-Driven Documentation (MoDDo) System which extracts relevant information from models specifying the software system.

The paper is structured as follows. In section 2 we briefly discuss related work. Section 3 outlines the adapted documentation process. Moreover, we present the architecture and implementation of the MoDDo System. Finally, we draw the conclusion in Section 4.

2. RELATED WORK

Most papers on automated documentation generation refer to tools for reverse engineering. In this approach software documentation for APIs, source code or even architecture [Deursen and Kuipers 1999; Riva and Yang 2002] is done based on already compiled software components.

Another approach is to document software directly during implementation phase, e.g. literate programming. Literate programming is the combination of documentation and coding in a fashion suitable for reading by persons. It mainly concentrates on technical documentation aspects, such as APIs, source code documentation and software component relationships whereas high-level process descriptions are not covered [Knuth 1992; Mall 2008].

A more software-generation-oriented approach for documentation generation has already been defined in the early eighties. SODOS (software documentation support environment) supports information generation at the specification and development time of software and aims to generate the necessary information without significantly adding to the documentation burden of the system developer [Horowitz and Williamson 1986]. The system resulting from this approach is based on a Database Management System (DBMS) and an object-based model of the software lifecycle but does not cover MDSD.

The World Wide Web Consortium provides a standard to transform XML documents [W3C 2006] into XML documents, called Extensible Stylesheet Language Transformations (XSLT) [W3C 2007]. Since models used in MDSD can be serialized as XML, XSLT is one viable transformation mechanism to retrieve information from models. However, XSLT lacks capabilities regarding constraint expressions, language exits and it is not incorporated into the MDSD.

3. THE MODDO SYSTEM – ARCHITECTURE AND IMPLEMENTATION

In order to benefit from MDSD in terms of documentation support, we introduce an aligned documentation process and the MoDDo System, a system adopting this process. The documentation process is illustrated in Figure 1 and aims at supporting the generation of all kinds of documentation (e.g. technical, end-user, and marketing documentation). Before presenting the MoDDo System, the dedicated documentation process will be described in detail.

Figure 1 – Documentation Process adapted to MDSD

The first step of the documentation process *Create Models* is also part of the software development process. It is actually the crucial part in MDSD since the entire software system is ideally specified by the use of models. From these models a non-intrusive mechanism is used to derive information. Extracted information is stored in atomic documentation entities, so called *Information Objects (IOs)*. The relation input model − extracted IO is one to many, so each input model is typically decomposed to dozens of IOs. Managing content in self-contained IOs promotes the content reuse. After generating IOs technical writers need to enrich the extracted information. Enriching IOs is the process of transforming generated information (e.g. bullet point lists) into continuous text. Enriched IOs need to be assembled in distinct document types (e.g. tutorial, handbook, etc.). The documentation process concludes in transforming the document type to the target document format XHTML.

The described process leads to the MoDDo architecture represented in Figure 2.

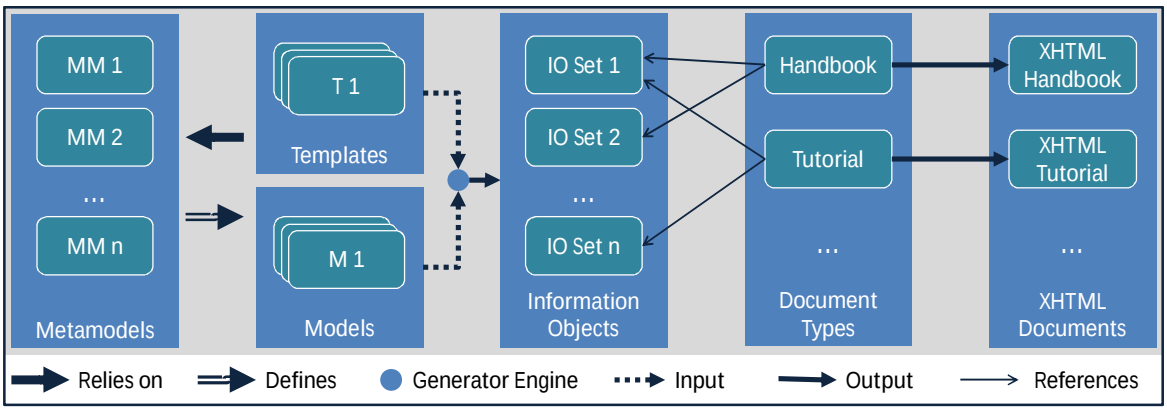

Figure 2 – MoDDo Architecture

The MDSD approach dictates that each model is based on a metamodel. The metamodel defines a language or a vocabulary that can be used to design models. In order to extract information from models, templates are introduced. These templates are defining a mapping between language elements (defined via a metamodel) and IOs. Since templates are referring to a metamodel and not to concrete models, they provide a generic mapping definition applicable to all models. To actually execute the mapping defined within templates a generator engine is utilized. Once IOs are created, an automatic linkage between static document types and IO Sets will be carried out. A final transformation generates XHTML documents using the document type description.

The implementation of the explained MoDDo architecture splits up into 3 pillars depicted in Figure 3.

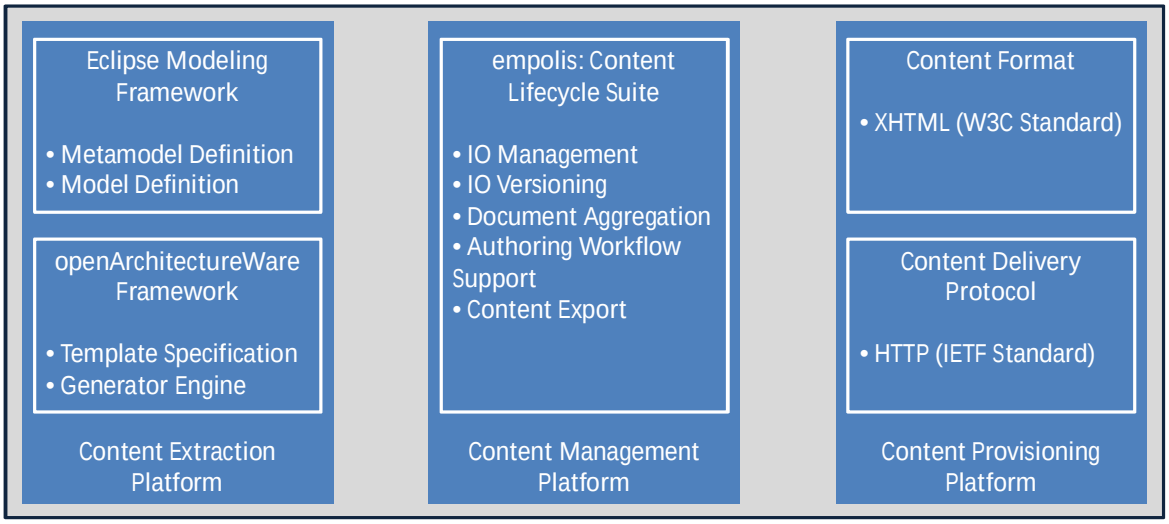

Figure 3 – MoDDo Implementation

While the *Content Extraction Platform* takes care of deriving information from existing models, the *Content Management Platform* governs the gathered information. The third pillar – the *Content Provisioning Platform* – makes all documentation available to the outside world.

The illustrated MoDDo implementation realizes the documentation process described previously. A walkthrough will explain the used frameworks. Before specifying the software system, metamodels based on the Eclipse Modeling Framework (EMF) [Budinsky et al. 2003] have to be created. Once the metamodels are in place, a software engineer will design models in order to describe each aspect of the software system. Those steps are essential for the software development process as well as for the documentation process.

Now templates using Xpand [Xpand 2008], a code generation language that comes with the openArchitectureWare Framework [oAW 2008], have to be defined. That is where the documentation process becomes independent. After the content extraction is triggered, a generator engine bundled within the openArchitectureWare Framework takes all templates and referenced models in order to create IOs. IOs are XML Documents satisfying an XML Schema, which comes with the empolis:Content Lifecycle Suite (e:CLS) [e:CLS 2008]. Our generated IOs contain raw information typically represented by bullet point lists. A technical writer needs to refine the information contained in an IO using a dedicated e:CLS Client. Besides modifying IOs, the e:CLS Client is also capable of setting up authoring workflows, retrieving different versions of IOs and aggregating IOs in various documents. The next step following IO aggregation is the content export step. XSL Transformations operate on IOs in order to generate XHTML documents. Finalized XHTML documents are to be published on the web.

4. CONCLUSION

In this paper, we described an approach to unify MDSD with software documentation. Therefore, we identified the characteristic steps of an MDSD-tailored documentation process and realized the defined process in the MoDDo System. The MoDDo System collects and preprocesses documentation-relevant information from models. The automatically retrieved information is consumed by technical writers who can thereby speed up their documentation tasks. Beyond reducing effort with respect to information search and aggregation, the MoDDo System also assures a more synchronized release schedule since software development and documentation development are closely aligned.

ACKNOWLEDGEMENT

The project was funded by means of the German Federal Ministry of Economy and Technology under the promotional reference “01MQ07012”. The authors take the responsibility for the contents.

REFERENCES

Budinsky, F. et al., 2003. *Eclipse Modeling Framework.* Addison-Wesley Professional, USA.

Deursen, A. v. and Kuipers, T., 1999. Building Documentation Generators. *Proceedings ICSM’99 (International Conference on Software Maintenance)*. Oxford, England, UK, pp. 40-49.

e:CLS 2008. Reference Manual - empolis:Content Lifecycle Suite (e:CLS Server). *Version 5.0 for e:CLS 3.2.2.*

Horowitz, E. and Williamson R. C., 1986. SODOS: a software documentation environment-its use. *IEEE Transactions on Software Engineering*, Vol 12, No. 11., pages 8 - 14

Kleppe, A. et al., 2003. *MDA Explained – The Model Driven Architecture: Practice and Promises.* Addison-Wesley, Boston, USA.

Knuth, D. E., 1992. *Literate Programming*. CSLI Lecture Notes. no. 27. Center for the Study of Language and Information, Stanford, California.

Mall, D. 2008. http://www.literateprogramming.com, retrieved July 17th, 2008.

oAW 2008. http://www.openarchitectureware.com/, retrieved July 20th, 2008.

Riva, C. and Yang, Y., 2002. Generation of architectural documentation using XML. *Proceedings of Ninth Working Conference on Reverse Engineering*. Richmond, Virginia, USA, pp. 161-169

W3C 2006. *Extensible Markup Language (XML) 1.1 (Second Edition)*. W3C Recommendation. World Wide Web Consortium. available at http://www.w3.org/TR/xml11/, retrieved July 23rd, 2008.

W3C 2007. *XSL Transformations (XSLT) Version 2.0*. W3C Recommendation. World Wide Web Consortium. available at http://www.w3.org/TR/xslt20/, retrieved July 23rd, 2008.

Xpand 2008. http://www.eclipse.org/gmt/oaw/doc/4.3/html/contents/core_reference.html#xpand_reference_introduction, retrieved July 20th, 2008.