The Journal of Systems and Software 83 (2010) 1179–1191

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

TALISMAN MDE: Mixing MDE principles

Vicente García-Díaz *, Héctor Fernández-Fernández, Elías Palacios-González, B. Cristina Pelayo G-Bustelo, Oscar Sanjuán-Martínez, Juan Manuel Cueva Lovelle

University of Oviedo, Department of Computer Science, Sciences Building, C/Calvo Sotelo s/n, 33007 Oviedo, Asturias, Spain

ARTICLE INFO

Article history:
Received 27 April 2009
Received in revised form 6 January 2010
Accepted 6 January 2010
Available online 13 January 2010

Keywords:
MDA
Software factory
TALISMAN
MDE
TMDE
Model-Driven

ABSTRACT

The Model-Driven Engineering approach is progressively gaining popularity in the software engineering community as it raises the level of abstraction in software development. In TALISMAN MDE framework, we combine the principles of the two most important initiatives, Model-Driven Architecture and Software Factories. Both have their pros and cons, and we select the best from each in TALISMAN MDE. To show the advantages of TALISMAN MDE, we have developed a systems generator and used it to create applications for controlling food traceability. The applications are being used in dairies with different manufacturing processes, using software developed specifically for each dairy by working only with models, without additional programming.

1. Introduction

The Model-Driven Engineering (MDE) is an increasingly popular approach to software engineering that tries to achieve the generation of application artifacts either automatically or semi-automatically. MDE is a generic term that refers to different initiatives like the Software Factories (SFs) (Greenfield, 2004), proposed by Microsoft or the Model-Driven Architecture (MDA) (Miller et al., 2003), proposed by the Object Management Group (OMG) (OMG, 2008), which are providing further impetus to MDE, at least in terms of the number of publications. There are some others attempts like the Architecture-Centric Model-Driven Software Development (AC-MDSD) (Völter and Stahl, 2006) that should also be taken into account, but in practice they all have the same underlying ideas.

Some proposals, such as SFs, focus primarily on productivity, while others, such as MDA, focus on interoperability, portability and reusability. The contribution of this paper is to present the TALISMAN MDE (TMDE) proposal, whose main novelty is the use of a mixture of principles to achieve maximum productivity with maximum possible interoperability, portability and reusability.

Since we know of no system to build food traceability applications using any MDE initiative, we show TMDE via a case study on developing software for food traceability via a systems generator developed following the TMDE principles, thus obviating the need to build specific software for each customer, as the models are used to build software automatically. It is important to note that the ideas underlying TMDE are not tied to any specific technology, but could just as easily be applied for any type of software domain.

The remainder of this paper is structured as follows: first, we present an overview of SFs and MDA, along with their assessments by experts. In Section 2 we describe the main concepts of TMDE. In Section 3 we define food traceability and the origin of our case study. Section 4 shows the real case study and finally, in Section 5 we indicate our conclusions and possible future work.

1.1. Software factories principles

In Greenfield et al. (2004), after comparing software engineering with other fields such as civil engineering, the authors have identified the following problems with traditional software development methods: (1) one-off development, (2) monolithic systems and increasing systems complexity, (3) process immaturity and (4) rapidly growing demand for software systems. To try to solve these problems, Microsoft has proposed the concept of the Software Factory, being its main objective the development of product lines. A software product line (SPL) is a set of software-intensive systems that share a common, managed set of features that satisfy the needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way (Clements and Northrop, 2002). As defined in Greenfield et al.

* Corresponding author.
E-mail addresses: garciavicente@uniovi.es, vicegd@gmail.com (V. García-Díaz), UO887@uniovi.es (H. Fernández-Fernández), palacioselias@uniovi.es (E. Palacios-González), crispelayo@uniovi.es (B. Cristina Pelayo G-Bustelo), osanjuan@uniovi.es (O. Sanjuán-Martínez), cueva@uniovi.es (J.M.C. Lovelle).

0164-1212/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2010.01.010

(2004), a SF is a software product line that configures extensible tools, processes and content using a software factory template based on a software factory schema to automate the development and maintenance of variants of an archetypical product by adapting, assembling and configuring framework-based components.

To explain this statement, we have to define the four main concepts related to SFs (Lenz and Wienands, 2006).

- Architecture frameworks implement common features of a system and provide extension points where these components can be integrated and extended.
- Product line Development should only attempt to cover a specific domain or market segment, without attempting to cover all the possible domains.
- Model-Driven Development is the closest concept to MDA, and is closely related to Domain-Specific Languages (DSLs) (Mernik et al., 2005).
- Guidance in Context states that the SFs should include facilities such as code samples, how-to help pages, articles and so on.

The idea is not to create a system whereby we can create all kinds of applications automatically, as MDA aspires to do. Rather, SFs are focused on specific domains or families of software product lines.

Conceptually, a SF can be divided into two main phases, the creation of a schema and the creation of a template of that schema. The SF schema is a model that can be interpreted by humans and tools, which describes work products, workflows used to produce the work products and assets used in the enactment of the workflows, for a specific family of software products in a given domain (Greenfield et al., 2004). Therefore, the development process depends on the experience of development, because with a defective schema the SF will fail. Then, the SF template can be considered as an instance of the schema (Lenz and Wienands, 2006) that is usually integrated into a development environment (Fig. 1).

The solution consists of three parts: the first is a common architecture that is reused for all the solutions of the SF, because they all have the same basis. The second are schematic repetitive artifacts, which differ from one solution to another but show a similar predetermined pattern and therefore can be abstracted by using domain specific design patterns (Spinellis, 2001). The last part corresponds to the specific artifacts that could have the solutions and therefore must be added manually from the development environment.

1.2. Model-Driven Architecture principles

MDA (Miller et al., 2003) is a set of related standards specified by the OMG, based mainly on reducing the weight of the implementation by modeling the system using standards. This follows a similar approach to concepts that are popular in other engineering fields. For example, building a bridge requires a detailed plan, that is, a model.

The three primary goals of MDA are portability, interoperability and reusability through architectural separation of concerns (Miller et al., 2003). The idea is to start with models of a high level of abstraction (Computational Independent Model or CIM) that reflect the requirements of the system. Later, the model is subjected to a transformation that lowers the level of abstraction to a computer model but is still independent of the computer platform used (Platform Independent Model or PIM) which represent solutions at design level for the requirements of the CIM. The PIM can be transformed into one or more specific models for one or more desired technological platforms (Platform Specific Model or PSM). The last transformation is to convert the PSM into the final artifacts (Implementation Specific Model or ISM), that are ready to be used or to be refined and then used. OMG has other defined standards that serve as a basis for the definition of the MDA.

Fig. 2 shows the entire development process within MDA. As can be seen, a pure MDA process does not provide for the introduction of individual artifacts or the use of architecture frameworks.

1.3. Different points of view

There are several recognized experts in the field of MDE, each having their own opinions about MDA and SFs. Some of their most relevant statements, in chronological order, are the following:

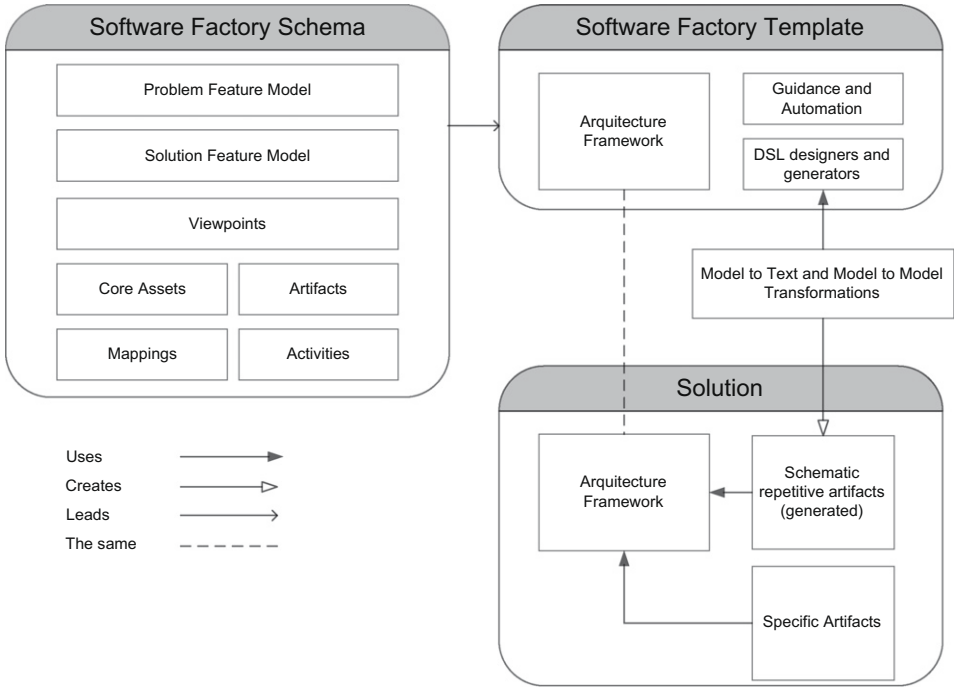

Fig. 1. Software Factories overview.