TALISMAN MDE Framework: An Architecture for Intelligent Model-Driven Engineering

Vicente García-Díaz[1], Jose Barranquero Tolosa[1], B. Cristina Pelayo G-Bustelo[1], Elías Palacios-González[1], Óscar Sanjuan-Martínez, and Rubén González Crespo[2]

[1] University of Oviedo, Asturias, Spain
{garciavicente,UO174160,crispelayo,palacioselias, osanjuan}@uniovi.es
[2] UPSAM, Madrid, Spain
ruben.gonzalez@upsam.net

Abstract. Models are becoming first-class artifacts in Software Engineering because they provide better productivity and quality. In this paper we present a framework for developing all kinds of applications, mainly by following the best practices of the two main approaches to Model-Driven Engineering (MDE). On one hand is MDA (Model-Driven Architecture), proposed by the OMG (Object Management Group) and on the other hand are the Software Factories, proposed by Microsoft. Both approaches have their pros and cons and that is why we want to mix them into a single framework by selecting the best that each of them can give us. TALISMAN MDE Framework is based on the opinion of recognized experts in MDE and on the lessons learned in our previous work, TALISMAN MDA.

Keywords: MDA, MDD, MDE, DSL, Software Factory, Model, TALISMAN.

1 Introduction

In this paper we present TALISMAN MDE Framework, which is a framework for developing all kinds of applications using the foundations of MDE (Model-Driven Engineering). The main objective of this work is that developers will not have to know theoretical aspects related to MDE and only have to focus on tools that are provided by our framework to develop software with better quality in less time. That is, we have created a framework that allows working with real tools that facilitate software development under the MDE guidance.

Both MDA (Model-Driven Architecture) [1] and the Microsoft Software Factories [2] are two MDE initiatives that have been widely studied in the recent years and therefore we will only do a very brief introduction about them.

The paper is structured as follows: in Section 2 we show the uncertainty regarding which is the MDE approach that will have more future and the lack of agreement that there is at present. Section 3 describes the architecture and key features of TALISMAN MDE Framework. In Section 4 we check that our work complies with the requirements that should have a MDE approximation and finally Section 5 shows the conclusions and future work.

S. Omatu et al. (Eds.): IWANN 2009, Part II, LNCS 5518, pp. 299–306, 2009.

1.1 Model-Driven Architecture

From the OMG (Object Management Group) point of view, MDA is the most important approach, at least in number of publications relating to MDE. It is based on an architecture of four layers in which the highest level (M3) is the meta-metamodel, occupied by the MOF (Meta-Object Facility) language that defines itself. At the level below (M2) there are the instances of the meta-metamodel, which is to say, the metamodels. The most representative of all MDA metamodels is UML (Unified Modeling Language). Going down a level (M1) would be the models, instances of the metamodels [3]. Using transformations, as automated as possible, the elements of M1 level are transformed between different views (CIM, PIM, PSM and ISM) in order to obtain a runtime implementation. This approach seeks above all to increase portability and interoperability.

1.2 Software Factories

Software Factories are the approach of Microsoft to MDE. It is important to note that it is a concept and it is not related to any particular technology but it is also true that Microsoft, besides the concept, offers technology to facilitate creating Software Factories. It is based on four fundamental pillars that together can increase the productivity of development: (1) Development of product lines, (2) Architecture Frameworks, (3) Model-driven development and (4) Guidance in context.

2 Different Points of View

There are a lot of opinions from leading experts on MDE:

Martin Fowler, in one of the most widely read books about UML [4] has distinguished between three main uses of UML: *UMLAsSketch, UMLAsProgrammingLanguage* and *UMLAsBlueprint*. In the latter sense, UML is used as a first-hand artefact that will be transformed automatically or semi-automatically to source code. This means that the UML model should be sufficiently rich to make the transformation complete, but he said that reality shows that it is not, as for example a UML class diagram does not support static fields.

Steve Cook [5] said that MDA has some parallels with the promotion and subsequent failure of the CASE tools (Computer-Aided Software Engineering) in the 80s and it is likely that MDA may fail unless it is supported by properly combined models, patterns, frameworks and code within an agile development process, because if not, MDA is not really an architecture.

Michael Guttman said [6] that the OMG standards are not the only MDE alternative but it is the best solution when looking for interoperability between different tools and platforms. In addition, there are many people who are researching these standards and there are already many cases of success in their use.

Dave Thomas [7] said that both MDA code generators and UML used in moderation, are useful and can help, but are far from what many have expected. He believes that UML is valid for many projects related to information technology and telecommunications but fails in other domains.

Steve Cook and Stuart Kent [8] again criticize UMLAsBlueprint for two reasons:

- UML is very large and the APIs that are offered to work with UML models are very difficult to use.
- UML is not domain specific and despite the UML Profiles, is not enough to save the conceptual differences between the domain and UML.

They propose the creation of domain specific IDEs (Integrated Development Environments) because they have the advantages of model-driven development and domain-specific languages.

Steven Kelly and Jukka-Pekka Tolvanen [9] criticize MDA and are in favour of a solution using DSLs (Domain-Specific Languages). According to them, UML is too general and does not serve to generate practical solutions through models transformations.

3 TALISMAN MDE Framework Architecture

The fact is that MDE experts do not agree with what is the best approach. However, based on the specifications and on their statements, they agree that the main goal of Software Factories is to seek the highest possible productivity by providing tools that focus on a specific platform, and that the fundamental advantage of MDA is the improvement of interoperability and portability, while not having in mind the Microsoft proposal. Hence, our goal is none other than merging the best of the two main MDE approaches in a framework. It consists of different components that together, they create a framework that provides intelligence to the development environment to facilitate the creation of applications. The architecture of TALISMAN MDE Framework (from now TMF) can be seen in Fig. 1. We will explain it now:

3.1 TALISMAN Selector

The goal of TALISMAN Selector is to avoid repeating the task of having to create from scratch the various projects and the code needed to connect them through different points. The majority of software solutions of a certain level of complexity are formed by more than one project. The goal is to provide a DSL with a graphical interface through which to add, delete, and update the basic configuration of any project that is included within a single software solution that is simple and quick. In addition, it will always have synchronized the graphic designer of TALISMAN Selector with the source code for various projects and it will serve as a tool to improve refactoring with respect to traditional IDEs since it is easier to locate the items using a graphic designer than by searching in the source code.

The closest thing to TALISMAN that exists today is a designer that is built into Visual Studio Team Edition for Software Architects, in the so-called Distributed System Design Suite. Its name is Application Designer but it has several disadvantages such as that it will not create all the components of the projects to work together under a single architectural solution because it creates very generic applications that you cannot use in free versions of Visual Studio.

To create TALISMAN Selector, we use the DSL Tools, a free of charge tool for creating DSLs. Rather than create one from scratch, we studied the possibility of

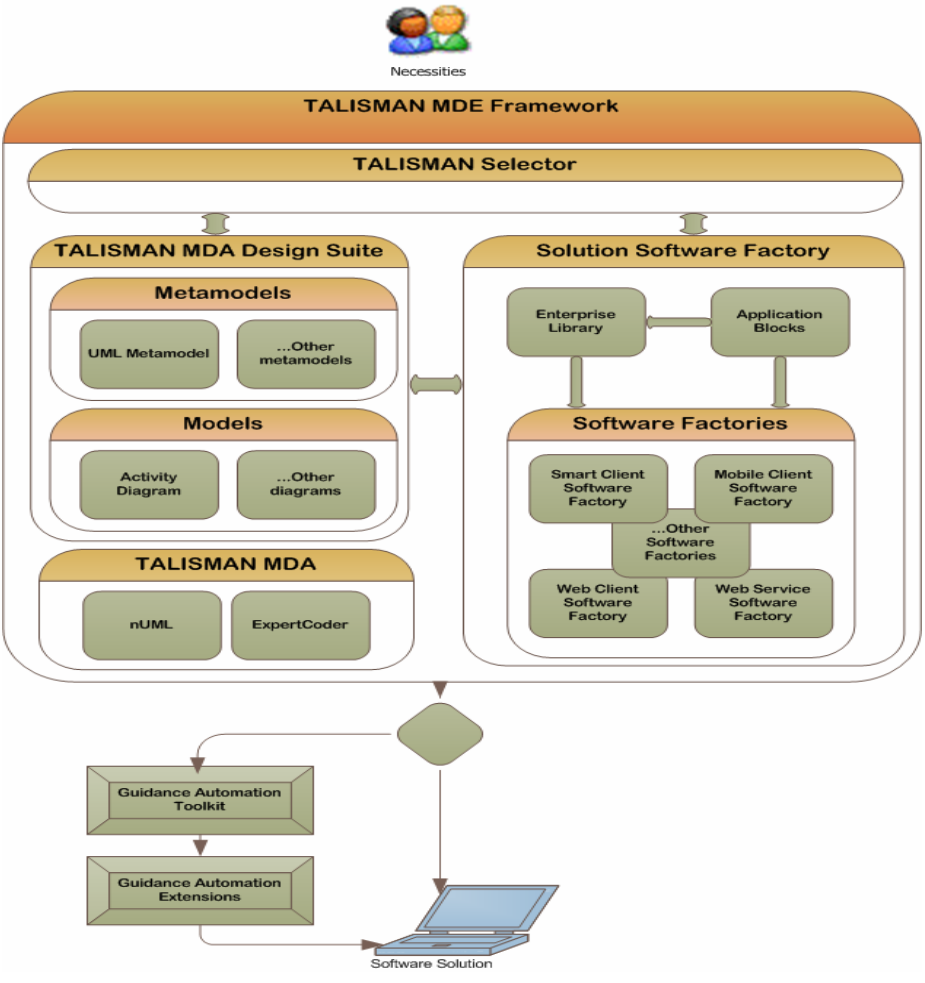

Fig. 1. TALISMAN MDE Framework. General architecture with its main components.

changing the Application Designer using the DSL Tools and thus achieve the functionality we just explained. But the reality is that Visual Studio designers have been created with a tool that has evolved in parallel with the DSL Tools and this fact has been crucial because Microsoft is currently promoting the DSL Tools. Fig. 2 shows an initial prototype of TALISMAN Selector.

3.2 Solution Software Factory

There are several Open Source Software Factories created by third parties in order to increase productivity in the development of a domain-specific application. They provide a set of guides and helpers for application developers, including frameworks, architectures, examples, reusable code, patterns, best practices and a set of automated tasks to use with an integrated IDE that improve the development experience.

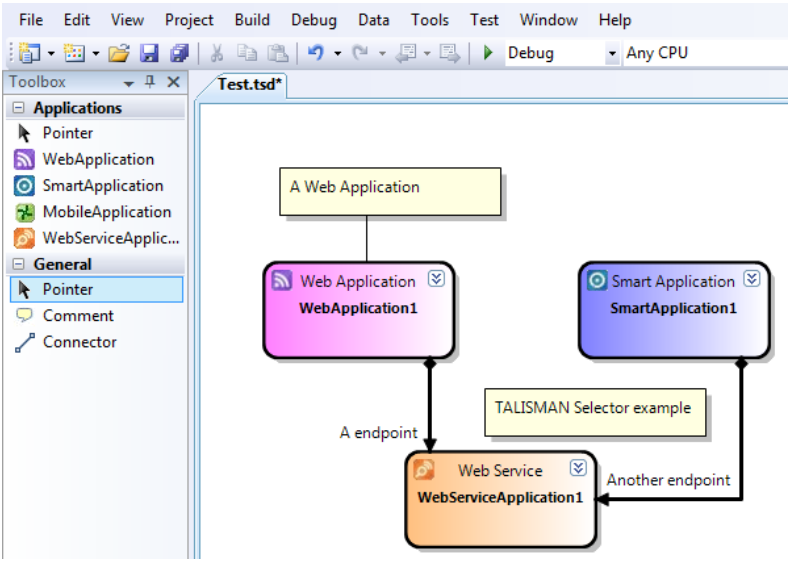

Fig. 2. TALISMAN Selector. A DSL with a graphical interface through which to add, delete, and update the basic configuration of any project that is included within a single software solution.

The goal of SOSF (Solution Software Factory) is to bring together the best features of each of the most important Software Factories and be able to build a Software Factory, which serves to generate any type of software solution. The idea is to enable developers to create software solutions and add different projects, regardless of the type of project, with the aim of having integrated the solution in a single location and with a common software architecture. One of the features of the previous Software Factories is that all are constructed independently of each other and this leads to architectural inconsistencies that might be accentuated over time.

Fig. 3 shows the *solution explorer* aspect when a solution is created using SOSF. As our goal is to create a modular framework so that their characteristics can be built and distributed separately, SOSF can be distributed for use regardless of the remaining components of TMF. Despite this, it is advisable to use it with TALISMAN selector in order to obtain a greater degree of automation because it is the component that is responsible for maintaining the synchronization between the different projects of the solution.

3.3 TALISMAN MDA

TALISMAN MDA is the subsystem that is responsible for all aspects related to the MDA approach within TMF. It may be used in isolation because the software is highly modular and loosely coupling. The main components are two Open Source projects in constant evolution:

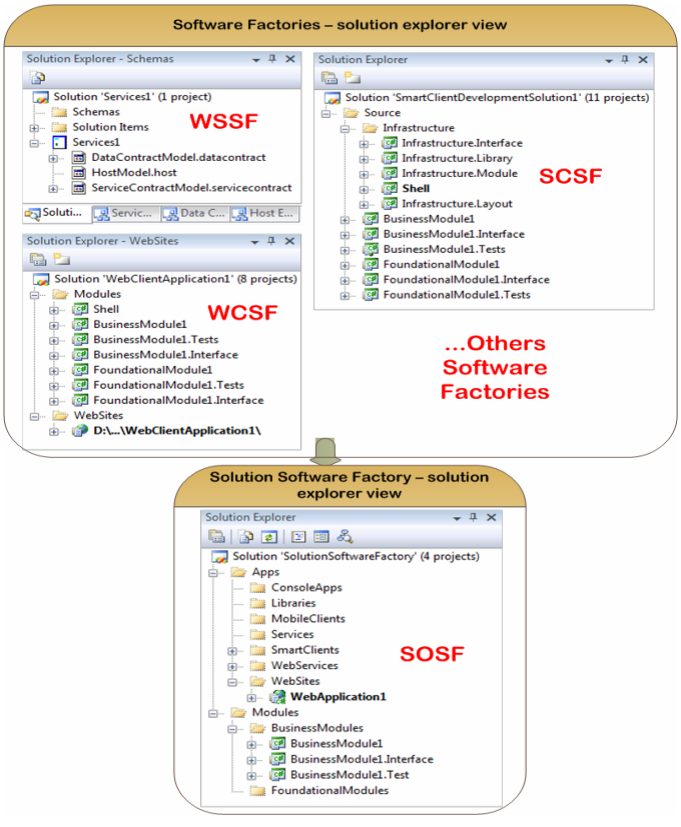

Fig. 3. Solution Software Factory. Appearance of solution explorer while creating a solution with SOSF.

- nUML [10]. It is a very powerful set of libraries that allows working with MOF 2.0, UML 2.0 and XMI 2.1. With nUML we offer the necessary tools to work with the standards proposed by the OMG in applications that are developed with TMF.
- ExpertCoder [11]. It is a set of libraries that are used to write code generators based on expert systems, creating a set of rules and determining the precedence that exists between them.

3.4 TALISMAN MDA Designer Suite

The *Class Designer* that is built into Visual Studio maintains synchronization between the code of the software solutions and UML class diagrams, so that changes in either side are reflected in the other side. Based on this fact, we want to implement key parts of the UML metamodel with the help of the DSL Tools. We believe it would be very helpful because at any moment of development one can have an overview of the UML diagrams and even could use diagrams to modify the code without the need of other external tools and with the assurance that both sides are synchronized at all

times. Initially, it will start with the UML metamodel to create UML diagrams and synchronize them with the source code, but the idea is to add support for other metamodels such as the ODM, an OMG metamodel for working with OWL ontologies.

4 Requirements for a MDE Approximation

To demonstrate that our framework fulfils its purpose, we have checked eight possible reasons why a MDE approach may not be successful [12].

1. *Not met all the objectives of MDE*. All components of TMF are intended to increase the development productivity as it reduces the complexity by using an integrated environment with simple tools and contextualized guides.
2. *Used only in one dimension modeling (PIM and PSM)*. With TMF, in addition to the transformations between PIM and PSM, we use different DSLs with the aim of facilitating the development of solutions. TMF also can be used as a base framework for creating new Software Factories.
3. *It focuses on generating new artifacts*. The use of *partial classes* and the fact that DSLs are synchronized with the code they generate makes the spread of the changes automatic, avoiding the generation of new artifacts in every change.
4. *They are used for general purpose languages*. With tools like DSL Tools, you can create other DSLs, within the development domain. If you prefer the MDA approach, with TALISMAN MDA, UML Profiles can be created.
5. *They use different domain specific languages*. By using a mix of approaches, it offers the possibility of working with UML Profiles and DSLs generated with the same metamodel, which use two different meta-metamodels.
6. *Transformations between models that are not executables*. With TMF you do not need to worry about the changes because they are prefixed in advance and carried out on the fly.
7. *Unable to test the model*. With TMF, all the DSLs are synchronized with the code that is generated and this makes the model directly verifiable.
8. *There are not enough tools to support the process*. TMF is perfectly integrated into the Visual Studio IDE using all its designers and helpers.

5 Conclusions and Future Work

New approaches and new technologies emerge with the aim of increasing the level of automation with which the software is built and that is why models are reaching an increased level of importance for developing software.

Both MDA and the Software Factories, which uses new technologies, are examples of how research in Software Engineering is betting heavily on the use of models like they did in the past in other industrial sectors. Both approaches provoke feelings and opinions both positive and negative by many prominent personalities in MDE and therefore it is difficult to predict what will be the future even in the short term. We

focus on a framework that allows working with tools that facilitate software development based on the Model-Driven Engineering paradigm.

We are working in various parts of TMF with the goal of having a mature tool that helps to give further advances in developing software. Our main target is to finalize a development framework made up of independent modules working together to help software developers create higher quality software in less time, based on the mix of the best ideas of MDA and the Software Factories and that is why our future work plans to go deeper into TMF.

References

1. Model-Driven Architecture, http://www.omg.org/mda
2. Greenfield, J., Short, K., Cook, S., Kent, S.: Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools. Wiley, Chichester (2004)
3. Gasevic, D., Djuric, D., Devedzic, V.: Bridging MDA and OWL ontologies. Journal of Web Engineering 4, 118–143 (2005)
4. Fowler, M.: UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd edn. Pearson Education, London (2004)
5. Cook, S.: Domain-Specific Modeling and Model Driven Architecture. The MDA Journal: Model Driven Architecture Straight From The Masters. Meghan Kiffer Pr (2004)
6. Guttman, M.: A response to Steve Cook. The MDA Journal: Model Driven Architecture Straight From The Masters. Meghan Kiffer Pr (2004)
7. Thomas, D.: MDA: Revenge of the modelers or UML utopia? IEEE Software 21, 15–17 (2004)
8. Cook, S., Kent, S.: The domain-specific IDE. The European Journal for the Informatics Professional (UPGRADE) 9, 17–21 (2008)
9. Kelly, S., Tolvanen, J.-P.: Domain-Specific Modeling: Enabling full code generation. Wiley, Chichester (2008)
10. nUML, http://nUML.sourceforge.net
11. ExpertCoder, http://expertcoder.sourceforge.net
12. 8 reasons why model-driven approaches (will) fail, http://www.infoq.com/articles/8-reasons-why-MDE-fails