Abstract

We would like to understand the interests of our target audience. Please register at **www.softmetaware.com/whitepapers.html** to provide us with some information about yourself, and to obtain access to the full content of all SoftMetaWare white papers.

Model-Driven Software Development Activities

The Process View of an MDSD Project

Author: Jorn Bettin

Version 0.1

May 2004

SoftMetaWare

1 Introduction

The Model-Driven Software Development (MDSD) [Bettin 2004a] paradigm is intentionally not prescriptive about most micro-level activities in the software development process. This enables a model-driven approach to be used in conjunction with a range of agile techniques, and with one of several methodologies for software product line engineering. At a macro-level however, the process and activities in Model-Driven Software Development are quite different from the activities in traditional iterative software development. Hence this article defines the essential macro-level **software modeling and development activities** that are characteristic of the MDSD paradigm.

Note that this article does not cover "non-development" activities such as domain analysis, requirements management, software supply chain design, and project management. A down-to-earth description of the essence of domain analysis is provided in [Cleaveland 2001], and software supply chain design for MDSD is described in [Bettin 2004b]. Further topics are covered as patterns in [Bettin 2004a] and [VB 2004].

1.1 Notation and Terminology

The best way of describing a high-level view of a process is in diagrammatic form. Besides UML (Unified Modeling Language) activity diagrams we make use of somewhat less formal diagrams as appropriate, making use of the following notation:

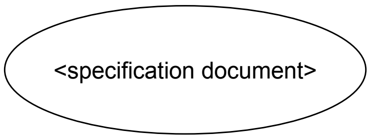
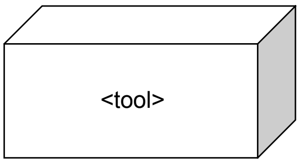
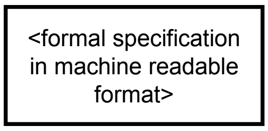
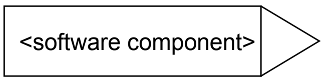
	Such as standards definitions, textual requirements specifications, etc.
	Such as a specification tools, generators, etc.
	Component specifications, source code etc.
	Executable software components. The optional triangular extensions symbolise interfaces

Figure 1 *Notation - Shapes*

We use the following color-coding scheme to differentiate different types of intellectual property, ranging from strategic proprietary IP to open industry standards.

🟥 Strategic IP (applications)	Domain-specific specifications, core application business logic and associated specifications
🟧 Non-Strategic IP	Non-strategic assets, such as legacy code and UI code
🟦 Strategic IP (product platform & infrastructure)	Strategic assets, such as a domain-specific product platform, standard UI components, security framework, etc.
🟦 Generatable Infrastructure Code	Typically infrastructure "glue" code and code to integrate with domain-specific product platform, legacy code, and external systems.
🟪 Industry Standards	Such as HTTP, SOAP, HTML, XML, etc.
⬜ External, 3rd Party Infrastructure	Such as J2EE, .NET, Oracle, Internet Explorer, etc.

Figure 2 *Notation - Colors*

1.2 Characteristics of Model-Driven Software

In MDSD we use the term *Model-Driven Software* to refer to software that is developed using a model-driven approach and generative techniques, as the quality attributes (technical consistency, consistency of user interface, maintainability, ...) of software developed and maintained that way are different from the quality attributes of "ordinary" software. Model-Driven Software can be constructed using endless possible combinations of implementation technologies, and leveraging numerous design patterns in various ways, which is why the term "Model-Driven Architecture®" that was introduced by the Object Management Group is probably not the best choice of words.

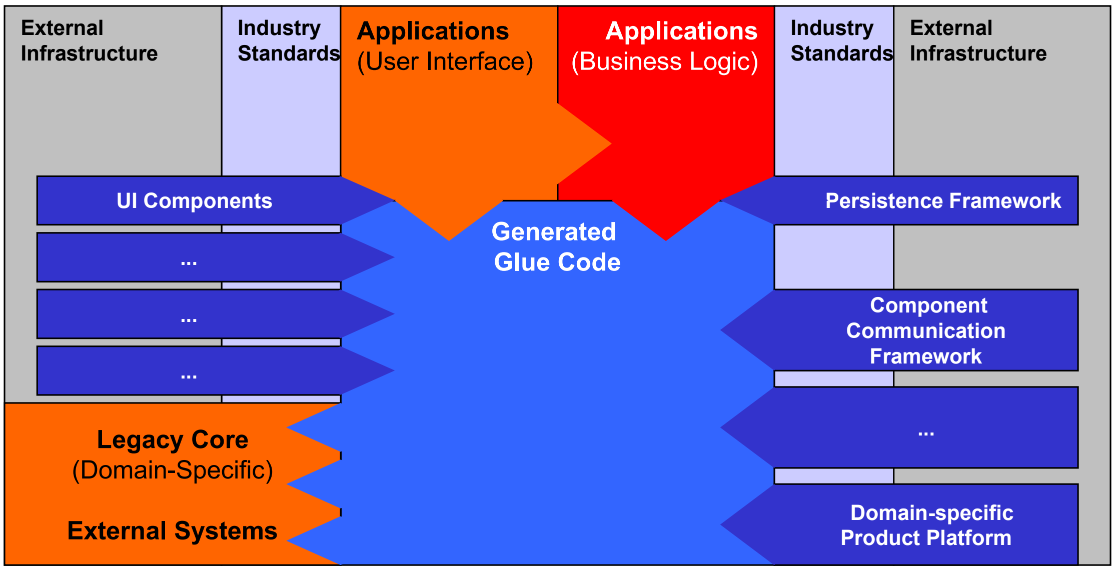


Figure 3 *Characteristics of Model-Driven Software*

In Model-Driven Software Development and in the design of Model-Driven Software we

- Leverage existing skill sets by capturing domain-specific knowledge (IP) in a human and machine readable format
- Increase maintainability of code base by insulating strategic IP from implementation technology churn
- Use automation to achieve organizational agility, reduce software development costs, and decrease time-to-market

Figure 3 shows that the difference between architecture-centric MDSD and MDSD with a rich domain-specific product platform (ARCHITECTURE-CENTRIC MDSD pattern in [Bettin 2004a], [VB 2004]) is small: In architecture-centric MDSD there is no "domain-specific product platform" and the size of the code base that needs to be manually maintained is larger − all domain-specific business logic needs to be hand-crafted, but otherwise the overall picture in terms of leveraging industry standards and using external infrastructure is unchanged. The development of a domain-specific product platform is an incremental process once architecture-centric MDSD is established.