Model-Driven Software Development Activities

The Process View of an MDSD Project

Author: Jorn Bettin

Version 0.1

May 2004

Copyright © 2003, 2004 SoftMetaWare Ltd.

SoftMetaWare is a trademark of SoftMetaWare Ltd.

All other trademarks are the property of their respective owners.

SoftMetaWare

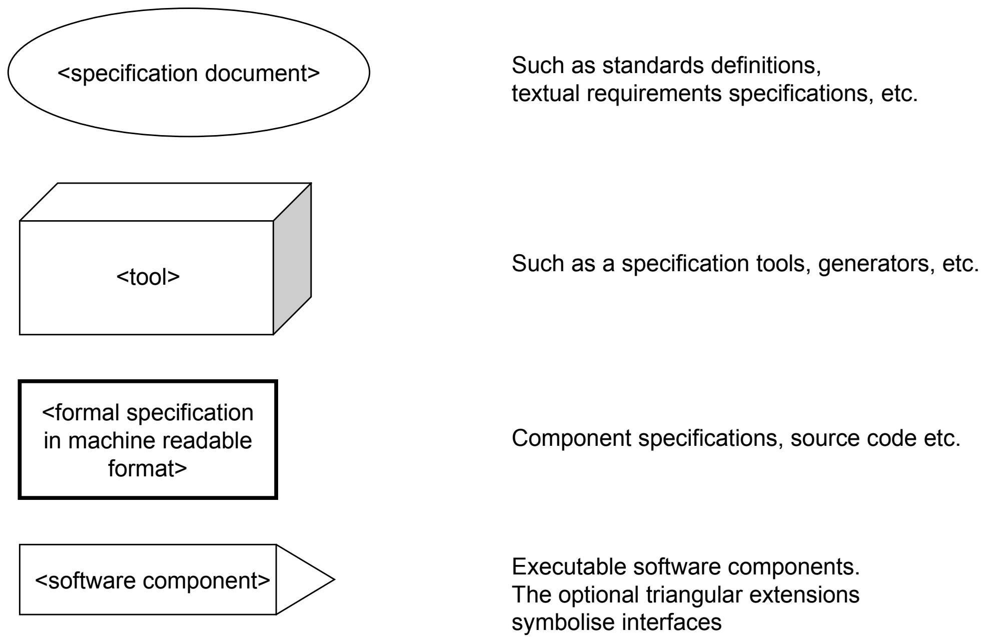
1 Introduction

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

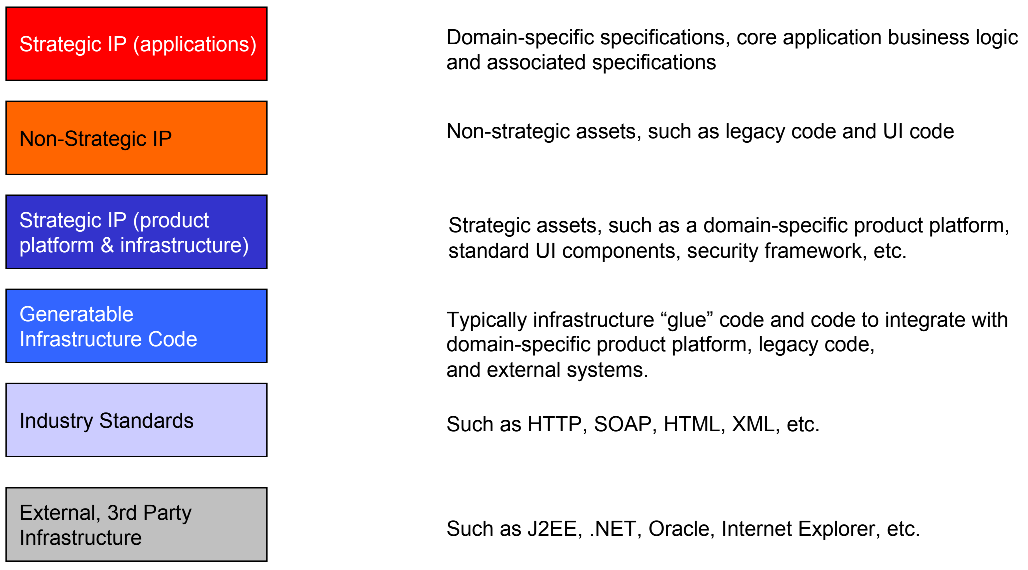
The best way of describing a high-level view of a process is in diagrammatic form. Besides UML (Unified Modeling Language) activity diagrams we make use of somewhat less formal diagrams as appropriate, making use of the following notation:



<specification document>
Such as standards definitions, textual requirements specifications, etc.
<tool>
Such as a specification tools, generators, etc.
<formal specification in machine readable format>
Component specifications, source code etc.
<software component>
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.

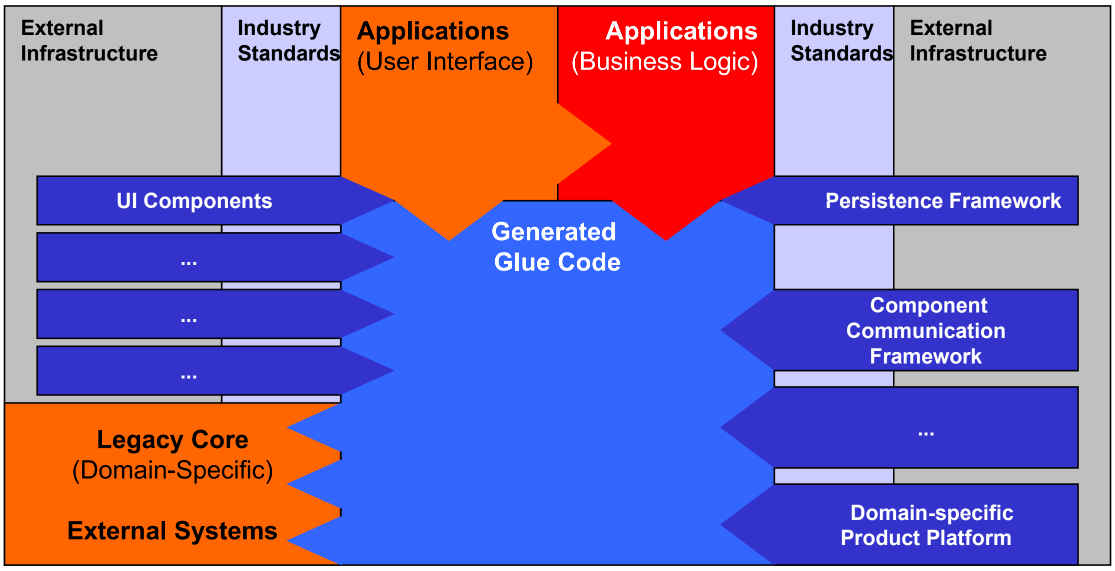


Figure 3 *Characteristics of Model-Driven Software*

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.

2 The Domain Engineering Process

The definition of MDSD rest on the foundation of software product line engineering principles [WL 1999]. This means that an MDSD process can be described in terms of a *Domain Engineering* workflow and an *Application Engineering* workflow. The steps shown in figure 4 describe the Domain Engineering workflow, which is the responsibility of a Domain Engineering Team within the software development organization.

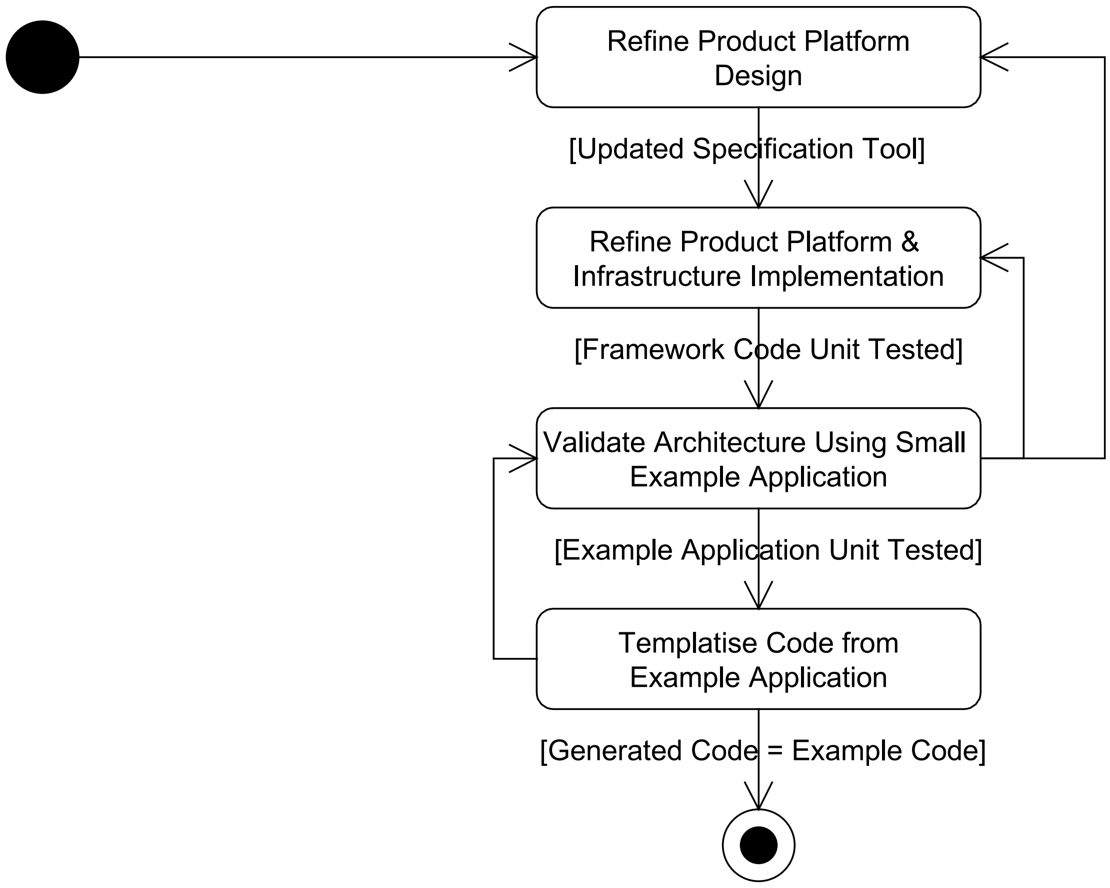

Refine Product Platform Design
[Updated Specification Tool]
Refine Product Platform & Infrastructure Implementation
[Framework Code Unit Tested]
Validate Architecture Using Small Example Application
[Example Application Unit Tested]
Templatise Code from Example Application
[Generated Code = Example Code]

Figure 4 *Domain Engineering workflow*

The initial step towards product-line engineering is infrastructure standardization, so the term *Infrastructure Engineering* is appropriate until the focus of work shifts to the domain-specific product platform .

In order to get a good idea of the domain engineering workflow, we need to drill down to the next level of detail.

2.1 Refine Product Platform Design

The first step of defining or refining a product platform consists of designing the meta model of a domain-specific application specification tool - in more technical terms this is called designing a meta model of a domain-specific language.

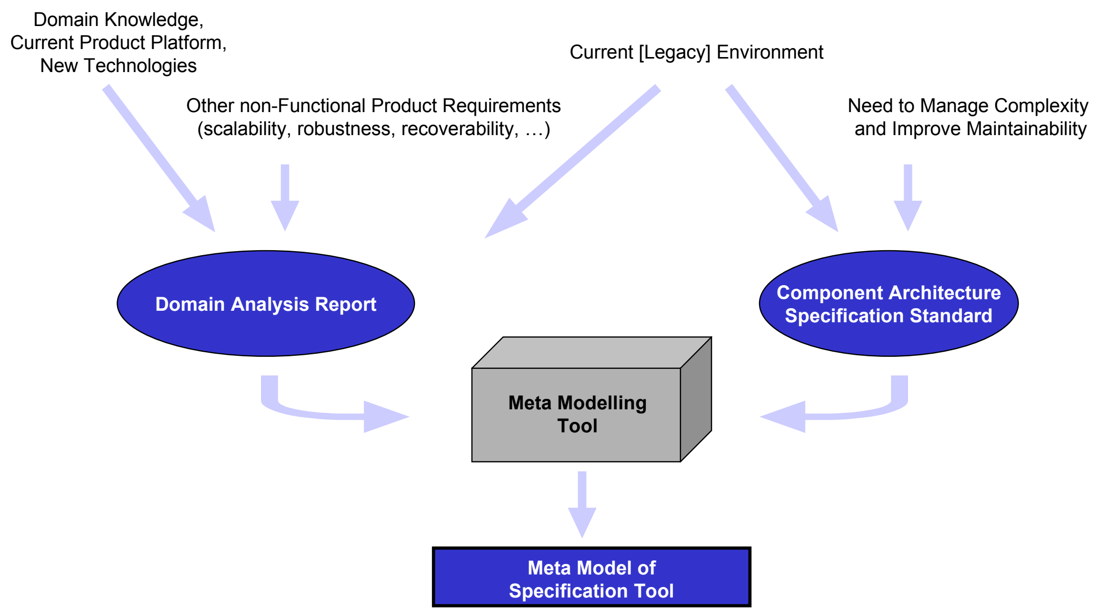

Domain Knowledge,
Current Product Platform,
New Technologies
Current [Legacy] Environment
Other non-Functional Product Requirements
(scalability, robustness, recoverability, …)
Need to Manage Complexity
and Improve Maintainability
Domain Analysis Report
Component Architecture
Specification Standard
Meta Modelling
Tool
Meta Model of
Specification Tool

Figure 5 *Designing the Meta Model of a Domain-Specific Application Specification Tool*

Figure 5 shows example inputs into the meta modeling process. The next step consists of using a specification tool generator to automatically generate a domain-specific application specification tool. This may sound esoteric to some software architects and software developers, but this is a well established process in software product line engineering, and today there are several industrial-strength Open Source tools [Eclipse GMT], [GenFW] available that support this process.

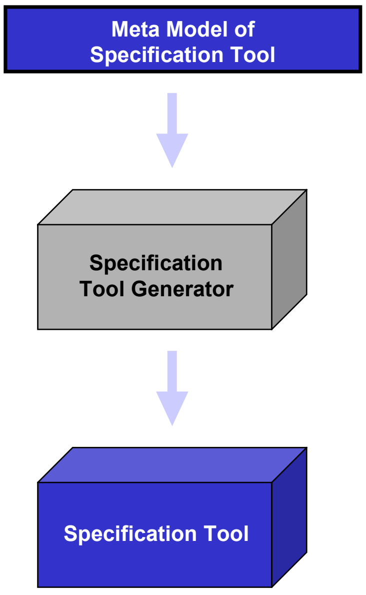


Figure 6 *Automatic Generation of a Specification Tool*

The first version of the generated specification tool may be very basic, but it will be sufficient for practical validation of the meta model, which is performed simply by capturing relevant specifications for a concrete application using the generated tool.

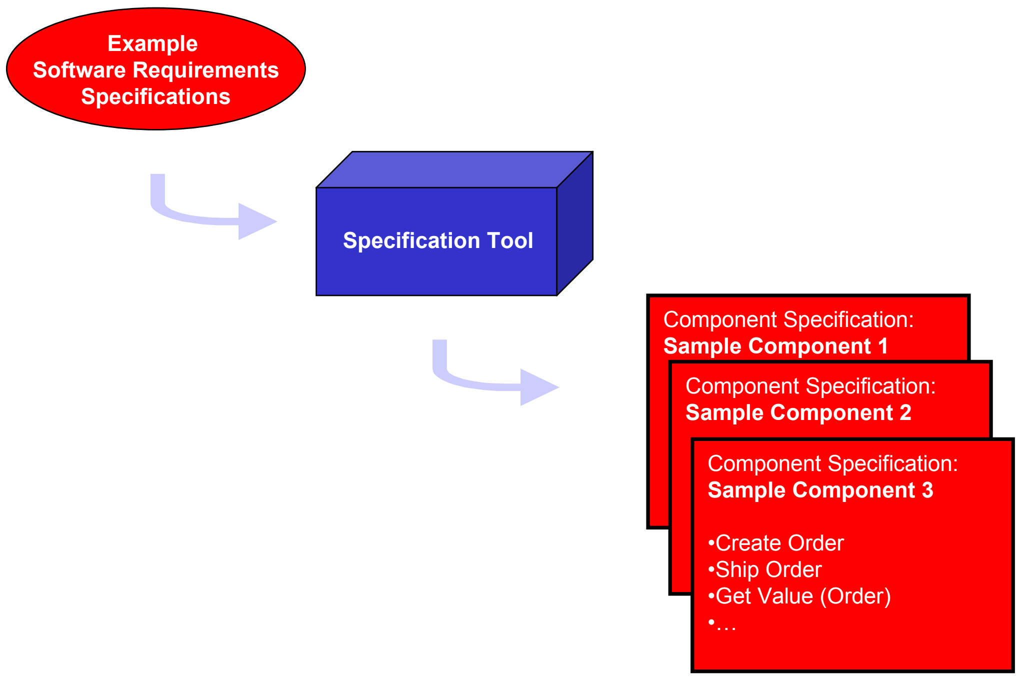


Figure 7 *Validation of the Generated Specification Tool*

The results of the validation may lead to some tailoring of the specification tool generator, which usually amounts to code changes in a set of code generation templates, and which contrary to some popular myths and misconceptions about model-driven approaches (see section *Myths about Model-Driven Approaches* [Bettin 2004a]) does not consume a significant amount of time and effort.

2.2 Refine Product Platform & Infrastructure Implementation

In Model-Driven Software domain-specific frameworks and the binding to concrete implementation technologies are part of an application or *product platform*. The development of domain-specific meta models and the development of domain-specific frameworks go hand in hand. The generative approach within the MDSD paradigm enables automatic generation of framework completion code (sometimes also called glue code), and thereby represents a large step forward in framework design. Without the assistance of model-driven generators, framework designers have to provide extensive documentation to **promote** correct use of the framework, and the very nature of frameworks means that users are forced to learn a significant amount about the internal workings of a framework. MDSD enables framework designers to **enforce** correct use of a framework, and shields framework users from the technical details of a framework that pertain to the solution space rather than the problem space. Figure 8 shows that the development of domain-specific frameworks involves integration with relevant industry standards and external infrastructure implementation. The quality of a domain-specific framework is dependent on the level of insulation that the framework provides between applications and external implementation technologies that are subject to technology churn.

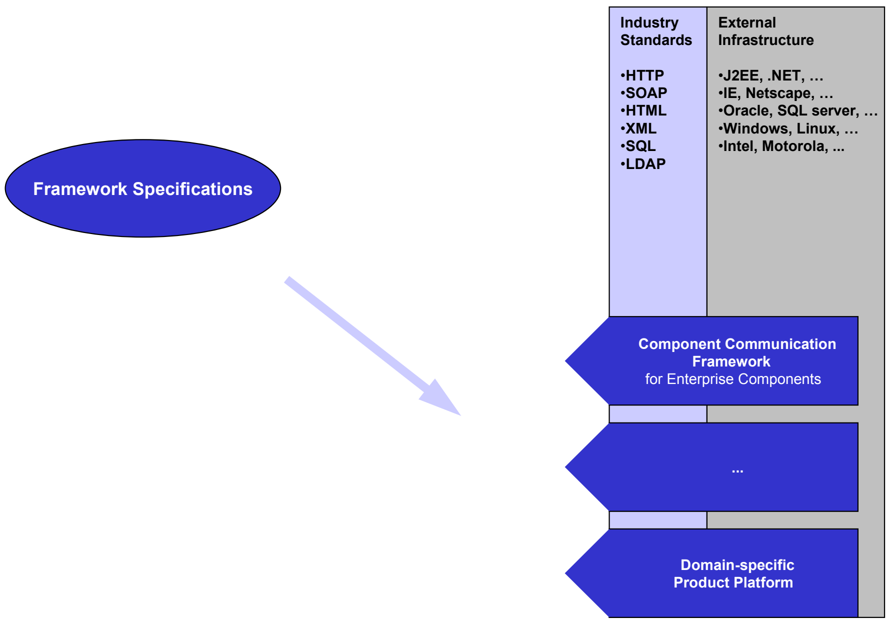
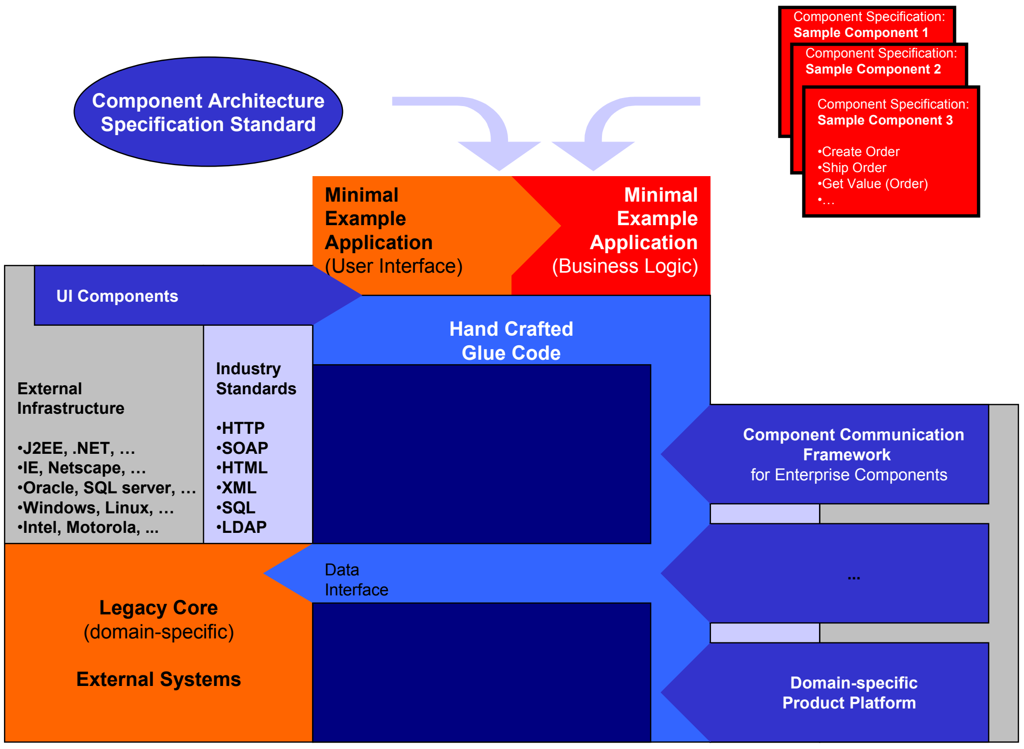


Figure 8 *Development of Domain-Specific Frameworks*

2.3 Validate the Architecture using an Example Application

The development of high-quality domain-specific frameworks does not happen in a vacuum, but in the context of the experience of building more than one application of a specific type. Framework functionality is typically extracted out of a working *reference application*. The first version of the framework is then tested and validated by rebuilding the reference application using the newly developed framework.



Component Specification: Sample Component 1
Component Specification: Sample Component 2
Component Specification: Sample Component 3
•Create Order
•Ship Order
•Get Value (Order)
•…
Component Architecture Specification Standard
Minimal Example Application (User Interface)
Minimal Example Application (Business Logic)
UI Components
Hand Crafted Glue Code
External Infrastructure
•J2EE, .NET, …
•IE, Netscape, …
•Oracle, SQL server, …
•Windows, Linux, …
•Intel, Motorola, …
Industry Standards
•HTTP
•SOAP
•HTML
•XML
•SQL
•LDAP
Component Communication Framework for Enterprise Components
…
Data Interface
Legacy Core (domain-specific)
External Systems
Domain-specific Product Platform

Figure 9 *Validating the Architecture of Domain-Specific Frameworks*

2.4 Templatize Framework Completion Code

As has already been hinted at, once appropriate domain-specific frameworks are in existence, the required framework completion code can be extracted from the reference application, and encapsulated in the form of transformations, which in the case of a template language based code generator amount to code templates.

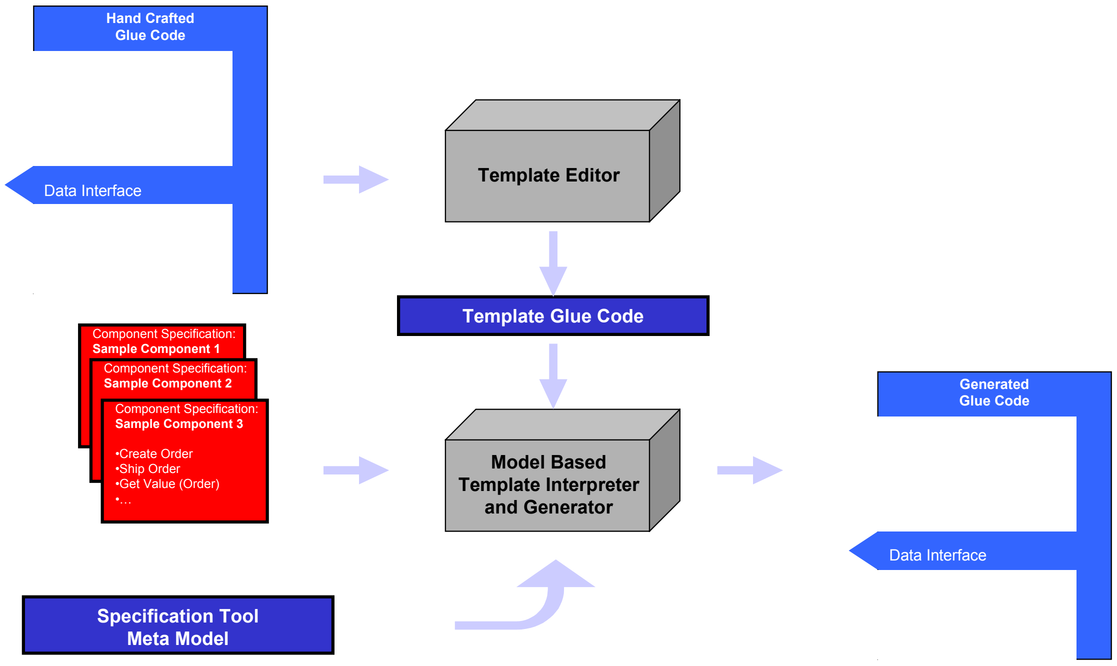

Hand Crafted Glue Code
Data Interface
Template Editor
Template Glue Code
Component Specification: Sample Component 1
Component Specification: Sample Component 2
Component Specification: Sample Component 3
•Create Order
•Ship Order
•Get Value (Order)
•…
Model Based Template Interpreter and Generator
Generated Glue Code
Data Interface
Specification Tool Meta Model

Figure 10 *"templatization" of Framework Completion Code*

Building an a product platform, validating the platform using a reference application, and templatizing framework completion code amounts to the EXTRACT THE INFRASTRUCTURE pattern [VB 2004], [Bettin 2004a] in MDSD. The next step in the process is a second round of validating the architecture, by generating the reference application from an appropriate model to ensure that the code templates accurately generalize the required framework completion code.

At this point we have:

- Relevant domain-specific meta models and frameworks
- A domain-specific application specification tool
- Transformations or code templates that together with relevant domain-specific meta models serve as the configuration for a model-driven generator that is capable of generating the required framework completion code

A last step before large scale deployment of the resulting domain-specific Application Engineering process usually consists of modeling a test application that is more broader than the original hand-crafted reference application, in order to provide additional test coverage for scenarios not covered by the original reference application.

3 The Application Engineering Process

In Product-Line Engineering terminology applying the results of Domain Engineering and automation to build applications constitutes the *Application Engineering* workflow, which is the responsibility of the Function Development Teams within a software development organization. On the highest level the custom-built domain-specific application engineering process follows the steps shown in figure 11.

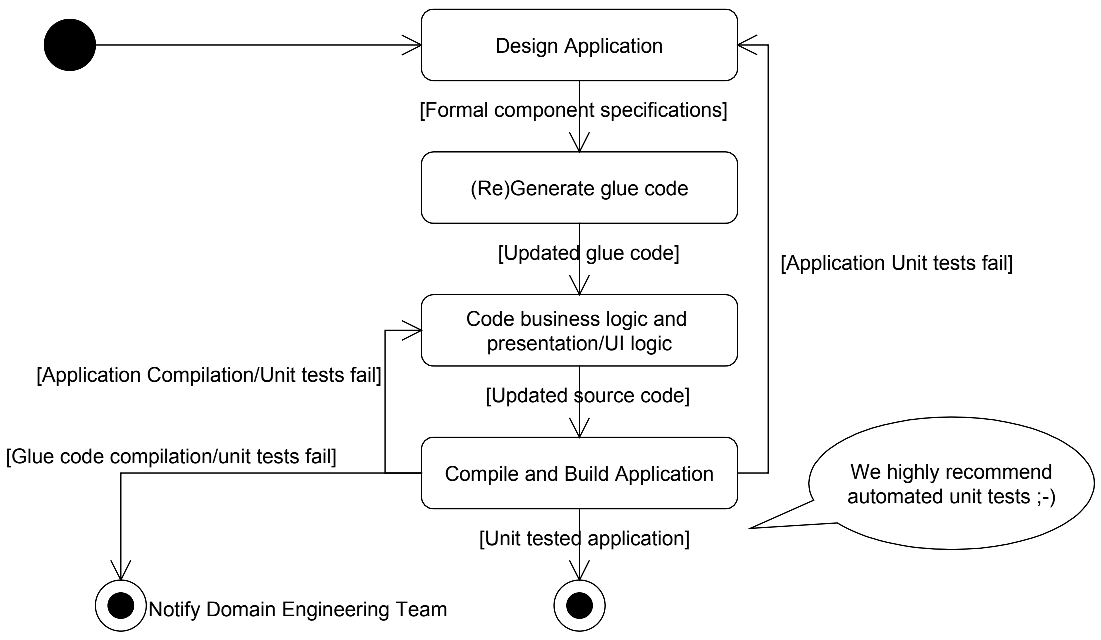

Design Application
[Formal component specifications]
(Re)Generate glue code
[Updated glue code]
[Application Unit tests fail]
Code business logic and presentation/UI logic
[Application Compilation/Unit tests fail]
[Updated source code]
[Glue code compilation/unit tests fail]
Compile and Build Application
We highly recommend automated unit tests ;-)
[Unit tested application]
Notify Domain Engineering Team

Figure 11 *Application Engineering Process*

3.1 Application Design

At *application design-time* the application engineers translate informal software requirements into formal specifications using the domain-specific application specification tool(s).

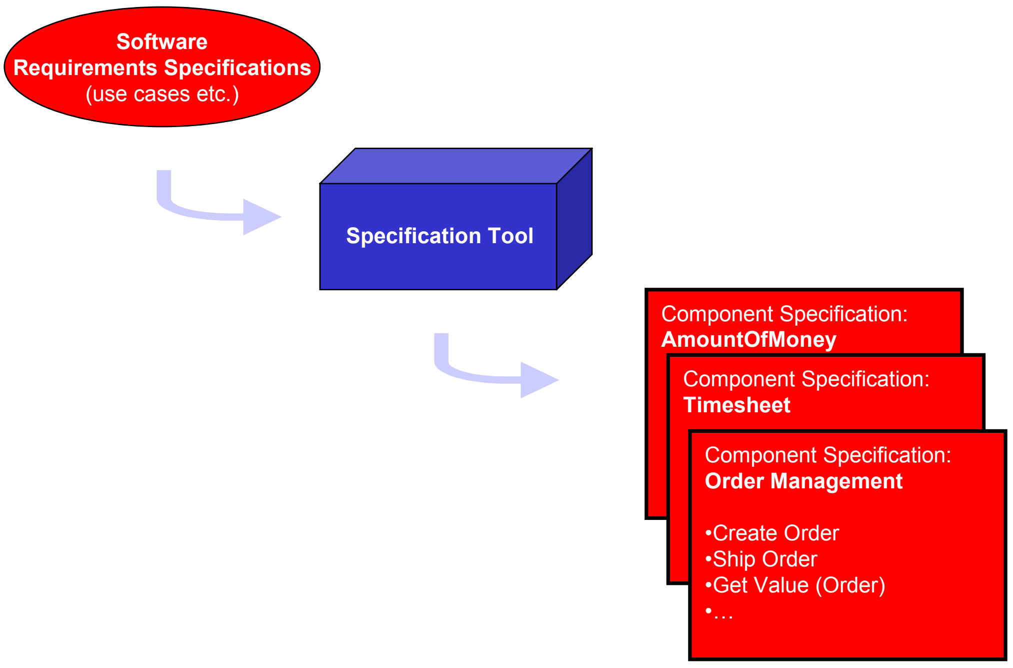

Software
Requirements Specifications
(use cases etc.)
Specification Tool
Component Specification: AmountOfMoney
Component Specification: Timesheet
Component Specification: Order Management
•Create Order
•Ship Order
•Get Value (Order)
•…

Figure 12 *Application Design-Time*

3.2 Application Generation

At *application generation-time* the application engineers use the domain-specific generators to generate framework completion code. The result is a prototype application that is functional except for those bits that are application-specific, and that are not covered by the functionality of the domain-specific framework(s).

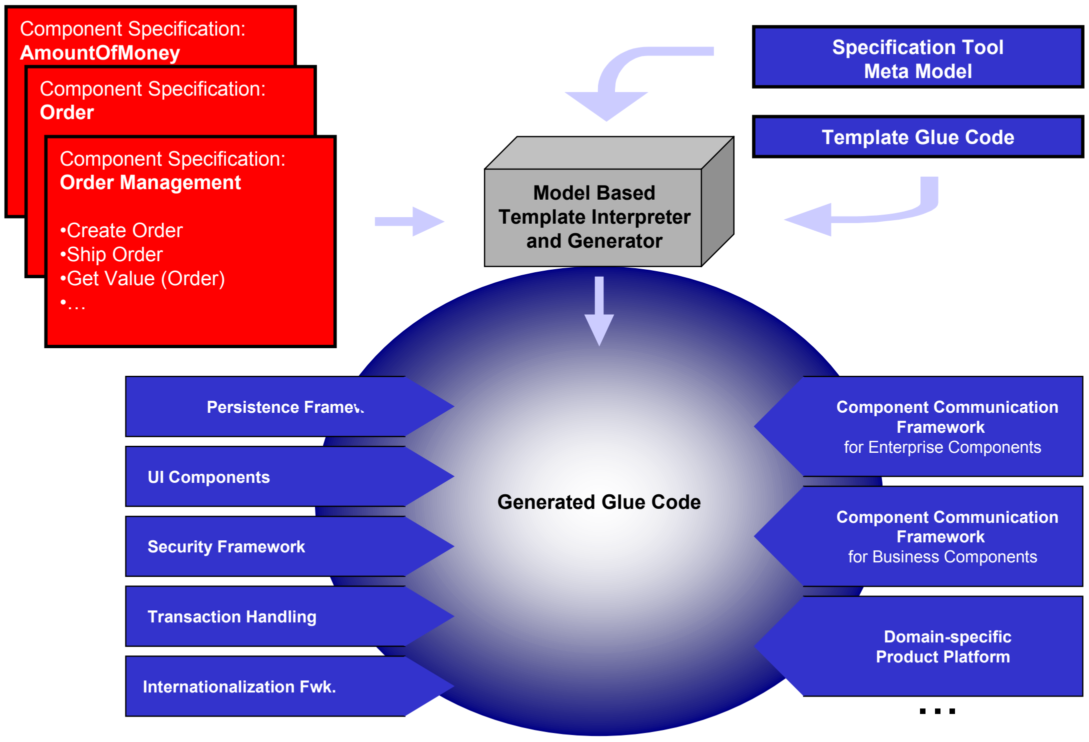
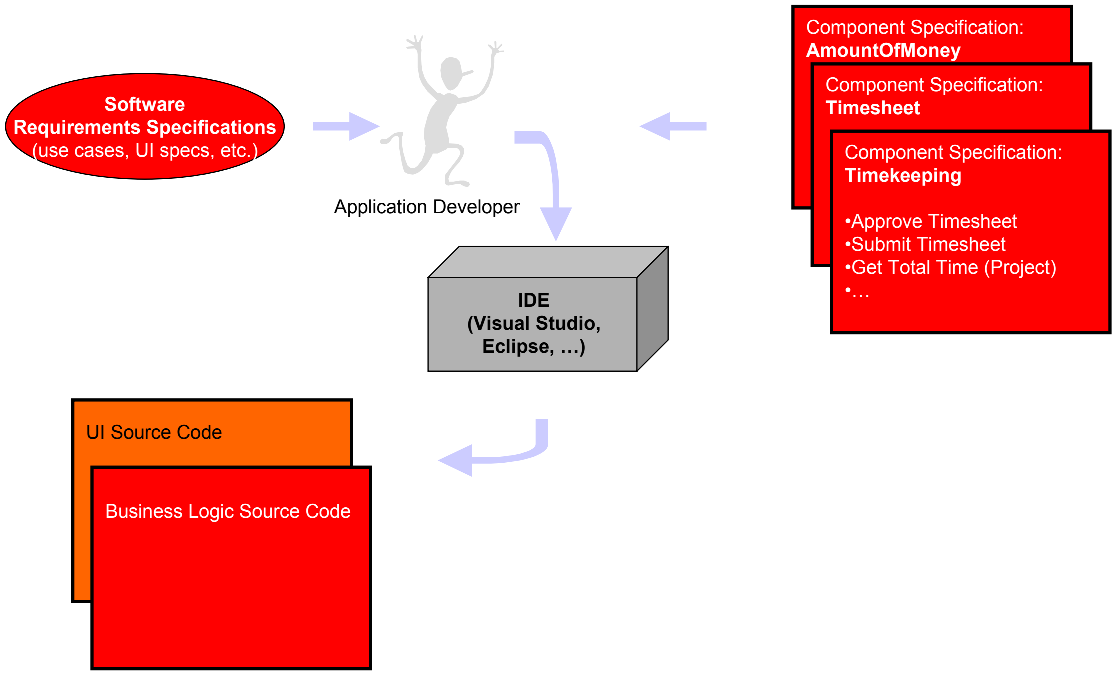

Component Specification: AmountOfMoney
Component Specification: Order
Component Specification: Order Management
•Create Order
•Ship Order
•Get Value (Order)
•…
Specification Tool Meta Model
Template Glue Code
Model Based Template Interpreter and Generator
Generated Glue Code
Persistence Framew.
UI Components
Security Framework
Transaction Handling
Internationalization Fwk.
Component Communication Framework for Enterprise Components
Component Communication Framework for Business Components
Domain-specific Product Platform
…

Figure 13 *Application Generation-Time*

As more and more applications get built, feedback from application engineers can be used to increase the expressive power of the domain-specific [modeling] language(s) used to specify applications, and to leverage commonalities between applications to further raise the level of abstraction of specification models.

3.3 Application Development

At *application development-time* the application engineers manually code the application-specific functionality that is not automatically provided via generation and the domain-specific framework(s).



Software Requirements Specifications (use cases, UI specs, etc.)
Application Developer
Component Specification: AmountOfMoney
Component Specification: Timesheet
Component Specification: Timekeeping
•Approve Timesheet
•Submit Timesheet
•Get Total Time (Project)
•…
IDE (Visual Studio, Eclipse, …)
UI Source Code
Business Logic Source Code

Figure 14 *Application Development-Time*

3.4 Application Compilation

Manually created application functionality needs to be compiled very much in the same way as in "traditional" software development. The compilation is typically controlled by a specific tool component, which depending on the design of the concrete application engineering process, is invoked either from a standard off-the-shelf IDE, or from within a domain-specific application specification tool.

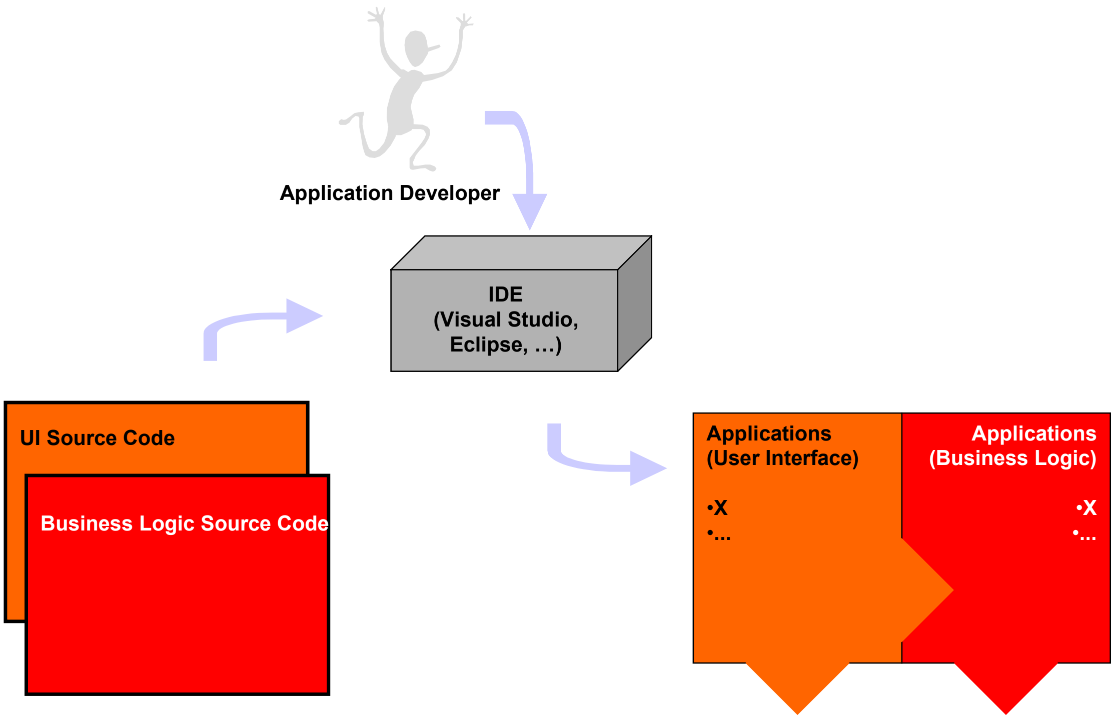

Application Developer
IDE (Visual Studio, Eclipse, ...)
UI Source Code
Business Logic Source Code
Applications (User Interface)
•X
•...
Applications (Business Logic)
•X
•...

Figure 15 *Application Compile-Time*

4 Using Model-Driven Software

Superficially the user should not notice any difference between model-driven software and software that is built using traditional software development techniques. At closer inspection, the user will typically notice a higher degree of consistency in the application of user interface standards and user-system interaction patterns. In fact, the relevant standards and interaction patterns can be agreed and validated with users in advance, when building the reference application.

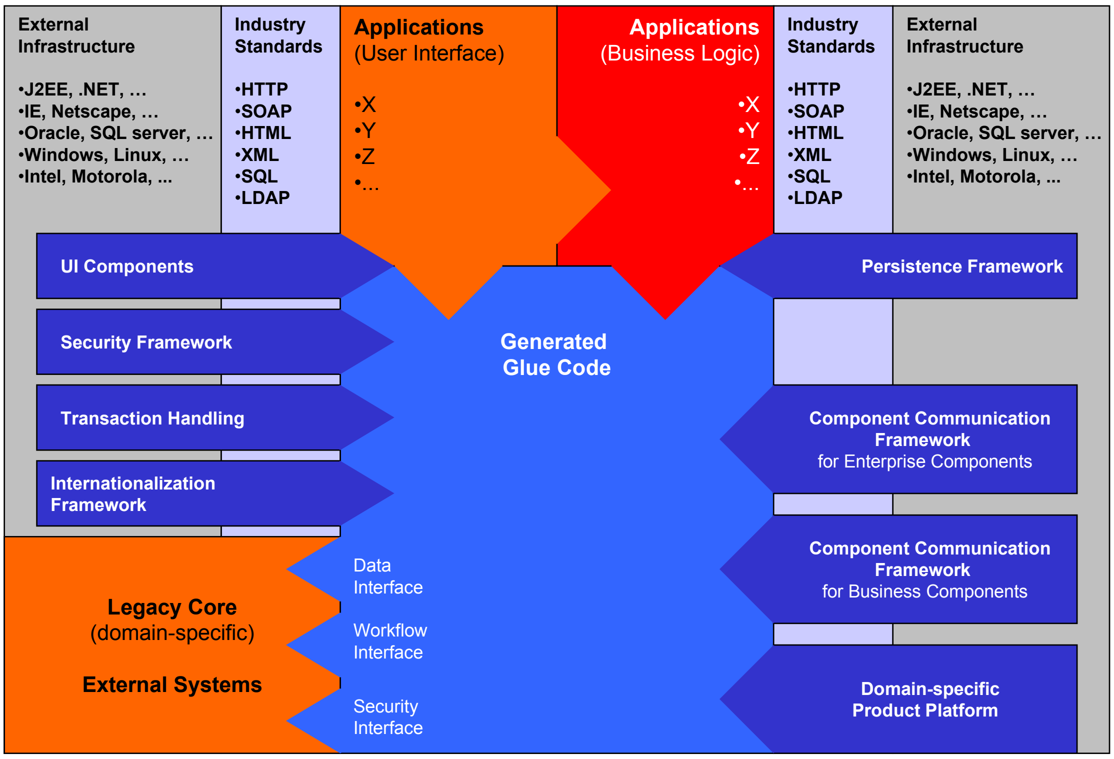


Figure 16 *Application Run-Time*

Figure 16 shows an illustration of a deployed model-driven software system. In terms of the implementation technologies used not really too different from software built with a different paradigm, but

- Separation of concerns is very pronounced
- Clean separation of different types of intellectual property in distinct parts of the code base
- Typically over 50% of the code is automatically generated from domain-specific models that are much smaller than the generated code

The result is a positive impact on application maintainability and application development productivity, and by implication on time-to-market.

5 Tool Support for MDSD

Figure 17 sketches the core tools that are needed to implement MDSD in addition to the usual set of software development tools.

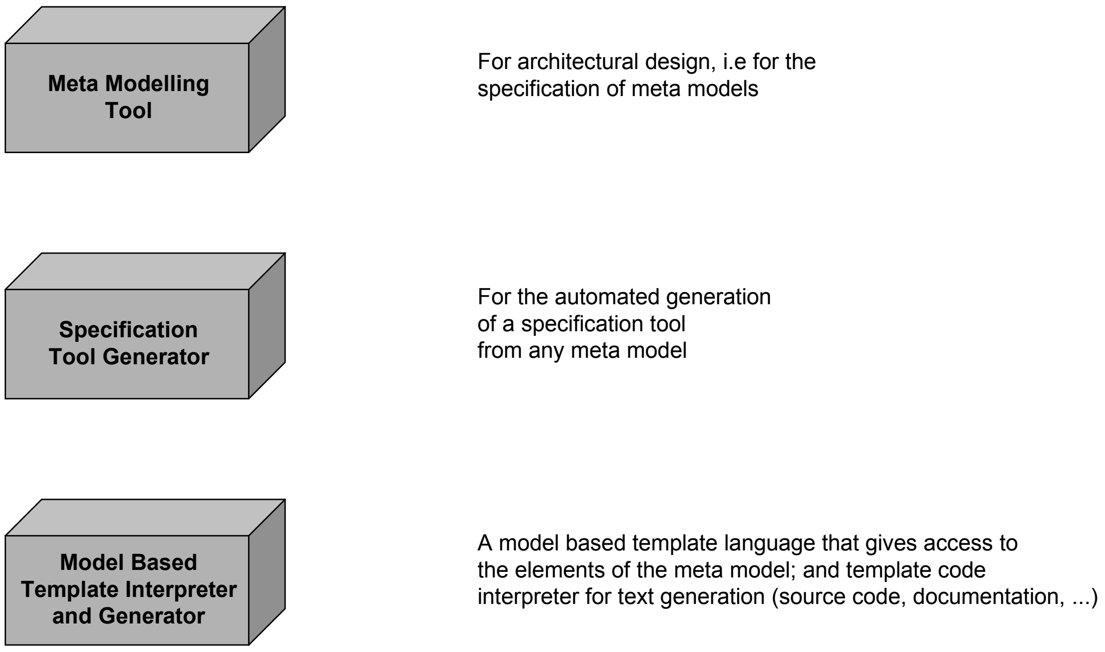

Meta Modelling Tool
For architectural design, i.e for the specification of meta models
Specification Tool Generator
For the automated generation of a specification tool from any meta model
Model Based Template Interpreter and Generator
A model based template language that gives access to the elements of the meta model; and template code interpreter for text generation (source code, documentation, ...)

Figure 17 *Required Tools for MDSD*

One example of a tool set that provides components for meta modeling, specification tool generation, and model-driven, template language based code generation is the FUUT-je tool which is part of the Generative Model Transformer Open Source MDA tool component platform [Eclipse GMT].

6 References

[Bettin 2004a]	Jorn Bettin, 2004, *Model-Driven Software Development: An emerging paradigm for industrialized software asset development,* http://www.softmetaware.com/mdsd-and-isad.pdf.
[Bettin 2004b]	Jorn Bettin, 2004, *Model-Driven Software Development Teams: Building a Software Supply Chain for Distributed Global Teams,* http://www.softmetaware.com/distributed-software-product-development.pdf.
[Cleaveland 2001]	Craig Cleaveland, 2001, *Program Generators with XML and Java*, Prentice Hall
[Eclipse GMT]	Generative Model Transformer project, http://www.eclipse.org/gmt/
[GenFW]	Sourceforge.net, *openArchitectureWare*, http://architecturware.sourceforge.net
[VB 2004]	Markus Voelter, Jorn Bettin, 2004*, Patterns for Model-Driven Software Development,* http://www.voelter.de/data/pub/MDDPatterns.pdf
[WL 1999]	D. M. Weiss, C.T.R. Lai, 1999, *Software Product Line Engineering, A Family-Based Software Development Process*, Addison-Wesley