Enterprise business application product line as a model driven software factory

Vinay Kulkarni
Tata Research Development and Design Centre, Pune, INDIA
+91 20 56086301
vinay.vkulkarni@tcs.com

Sreedhar Reddy
Tata Research Development and Design Centre, Pune, INDIA
+91 20 56086302
sreedhar.reddy@tcs.com

ABSTRACT

Enterprise business applications are critical to the smooth operation of modern businesses. They need to perform and scale up to the ever-increasing demands of modern businesses on IT, and are implemented using distributed architectures. These applications tend to have a long life during which they need to quickly respond to changing business rules, business processes and technology platforms. No two businesses are exactly alike even in the same business domain. This calls for an enterprise business application to be specialized for the needs of a specific business. Product line architectures that organize systems into well-defined core and variable parts have been proposed to address this need. Traditional code based development approaches do not provide the right kind of abstractions to support product lines. We propose a model driven development approach that enables a system to be specified in terms of composable units along the required dimensions of variations of a product line. Such a platform-independent specification can be retargeted to technology platforms of choice using model-based code generators. We propose a software factory for an enterprise business application product line wherein a set of purpose-specific tools generated from their specifications support a purpose-specific development process. We describe our experience in building and using such a software factory.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: D 2.9 Management, D 2.10 Design and D 2.13 Reusable software.

General Terms

Management, Design, Languages.

Keywords

Software factories, product lines, model driven development, separation of concerns, aspect oriented programming

1. INTRODUCTION

Modern businesses rely on enterprise business applications for their existence and smooth operation. During their lifetime, enterprise business applications need to quickly respond to changes in business rules, business processes and technology platforms. To have a better handle on scale-up and performance issues, modern enterprise applications are typically implemented as distributed systems. Faced with the task of developing large and complex applications, industrial practice uses a combination of non-formal notations and methods for implementation. Different notations are used to specify the properties of different aspects of an application and these specifications are transformed into their corresponding implementations through the steps of a development process. The development process relies heavily on manual verification to ensure the different pieces integrate into a consistent whole. This is an expensive and error-prone process demanding large teams with broad-ranging expertise in business domain, architecture and technology platforms such as presentation managers, programming languages, databases, middleware etc.

Model-driven development approach addresses this problem by providing a set of modeling notations for specifying different layers of a system namely user interface, application functionality and database in a platform independent manner [8]. A set of code generators then transforms these models into platform-specific implementations. Models, being at a higher level of abstraction, are easier to understand and verify for properties of interest. Model based code generation incorporating proven design and architectural patterns results in significant gains in productivity and uniformly high quality. This approach can be used to retarget product lines to multiple technology platforms.

No two businesses are exactly alike even in the same business domain. This calls for an enterprise business application to be specialized for the needs of a specific business. Product line architectures that organize systems into well-defined core and variable parts have been proposed to address this need the central idea being products within a product-line are differentiated by features [5, 2]. Producing a specific product variant can be seen as a stepwise refinement process wherein a common abstract model is refined to inject product-specific factors [1]. Feature commonalities can be captured as reusable patterns from which specific variants can be instantiated through suitable parameterization. A tool driven software factory can provide the necessary machinery to assemble the instantiated patterns [3]. Multi-dimensional separation of concerns approach addresses this need through decomposition of a system along multiple dimensions of interest [11]. Aspect oriented programming provides support for this approach only at programming language level where the same base language is used for specifying the different aspects of the system [6]. However, one would like to use purpose-specific languages to specify various aspects wherever possible. The richer abstractions provided by such higher level domain specific languages lead to ease of understanding and analysis, and a possibility of code generation.

We propose a model driven development approach that enables a system to be specified in terms of composable units along the required dimensions of variations of a product line. Such a platform-independent specification is retargeted to technology platforms of choice using model-based code generators. Typically, enterprise business applications tend to vary along five dimensions namely, functionality (F), business process (P), architecture (A), design strategies (D) and technology platform (T). A model based code generator encodes specific choices along A, D and T dimensions. We propose a software factory for an enterprise business application product line wherein a set of product line variant specific model based code generators are generated from their specifications. We describe our experience in building and using such a software factory.

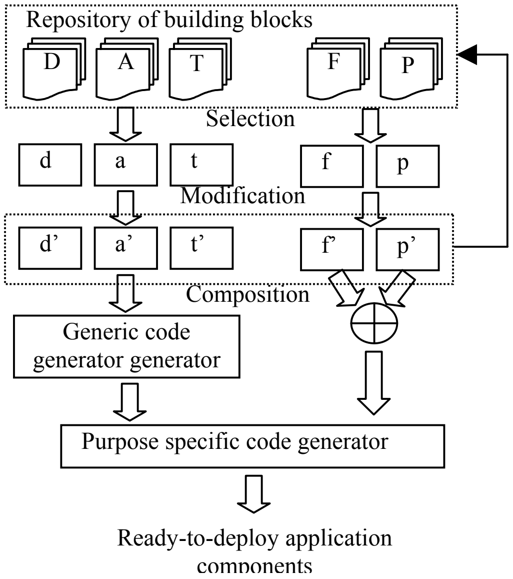
Repository of building blocks
D
A
T
F
P
Selection
d
a
t
f
p
Modification
d'
a'
t'
f'
p'
Composition
Generic code generator generator
Purpose specific code generator
Ready-to-deploy application components

Fig. 1. A model driven software factory

2. Solution approach

Figure 1 shows the proposed model driven software factory for enterprise business application product lines.

A product line is organized as a repository of composable building blocks structured along the different dimensions of variation. A specific product line variant is derived as a composition of such building blocks of interest along these dimensions. The derivation process begins by matching the requirements of the desired variant against the repository to select closest matching building blocks. A gap analysis then identifies the necessary modifications and adaptations to the candidate building blocks, if any. It may also lead to development of new building blocks. A purpose-specific code generator is then generated from these modified building blocks along A, D and T dimensions. The functionality and process building blocks are then composed to yield an integrated specification. The purpose-specific code generator translates this specification into a technology platform specific implementation incorporating the selected design and architectural patterns.

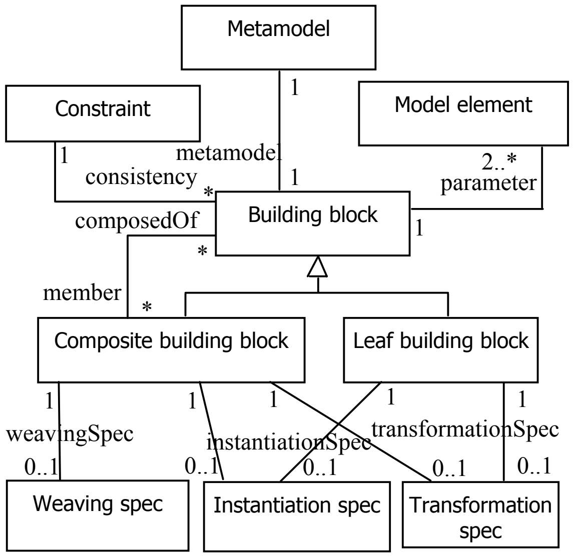

Fig. 2. Building block meta model

In our approach, an application is specified as a hierarchical composition of building blocks of interest along the dimensions of variation. A building block encapsulates reusable functionality along a dimension of variation. A building block can be seen as a specification of an aspect expressed in terms of a language specified by an associated meta model. Figure 2 shows the meta model of building block itself. On instantiation, a building block brings along a set of model elements that conforms to the meta model and associated constraints. Building blocks are of two kinds: *leaf building block* and *composite building block*. The instantiation specification of a leaf building block specifies how to stamp out aspect-specific model elements. The transformation specification specifies how the model is transformed into code. The instantiation specification of a composite building block specifies how model elements constructed in member building blocks are merged (woven) together. We have found *merge by name* scheme of model merging sufficient for our purposes. Weaving specification of a composite building block specifies how the code generated by its member building blocks is woven together. We have found a code weaving specification language along the linesof Hyper/J [4] sufficient for our purposes. We have used a model-to-text transformation language called SpecL for model transformations [10].

The process of aspect composition is realized through a post-order traversal of the building block hierarchy in three sequential steps namely *Instantiation*, *Transformation* and *Weaving*. The instantiation step stamps out models and merges them. The transformation step transforms models into code and generates weaving specifications for composing the generated code. The weaving step composes the generated code fragments by processing the weaving specifications.

3. Discussion

During the past 10 years we have developed several business-critical enterprise solutions for a variety of business verticals like banking, financial services and insurance [7]. We are in the process of organizing these purpose-specific enterprise solutions in the form of vertical-specific product lines. The proposed approach provides technology and process related infrastructure to support such product lines as a software factory. We are in the process of defining the required domain specific languages to specify building blocks along the required dimensions for each product line.

The process of deriving a specific product variant begins by identifying the business process flows of interest. This leads to identification of functions required to implement these flows. A keyword based search identifies functionality and process building blocks available in the repository. A manual comparison of the desired business process flows and business functions with existing process flows and their implementations identifies the functionality gap. Non-functional requirements like performance, throughput, architecture and technology platform are the basis for identifying D, A and T building blocks. A simple keyword based search mechanism is provided for selecting suitable D, A and T building blocks. These building blocks are composed to quickly realize implementation of model based code generators that impart the desired non-functional characteristics to the business functionality under consideration. If found unsuitable, one goes back to select a different set of D, A and T building blocks from the repository or modifies the existing ones suitably.

We have realized the factory vision shown in figure 1 only in parts by being able to address the D, A and T dimensions of variation by aspect-oriented restructuring of our MDD toolset facilitating easy customization of the code generators. We decomposed the code generators into well-defined self-contained building blocks such as model to java, object-relational map, auditing, concurrency management, error handling, message handling strategies like synchronous, asynchronous, queue-based etc.

Our MDD toolset [9] translates a model (M_u) that is an instance of a unified meta model (MM_u) to various software artefacts like Java code, JDBC code, JSP code and a variety of configuration specifications in XML as shown in figure 3. Limiting aspect weaving only to code level artefacts would necessitate specialized weavers for Java, JDBC, JSP, XML etc. each having separate join point models. Also, this approach would necessitate some commonality over these join point models so as to have an integrated Java application. With increased number of software artefacts to be produced the approach becomes increasingly complex as essentially it amounts to building aspect infrastructure for each such artefact. We address this problem by specifying aspect weaving at the unified meta model level and performing it at the model level whenever possible. Unified meta model enables specification of relationships between the various (sub) modeling languages. A reflexive meta modeling framework provides the necessary infrastructure to define and integrate the various modeling languages of interest and a meta model aware model transformation framework provides the necessary technology to address model weaving requirements. Performing aspect weaving at the model level also, whenever possible, results in reuse of model based code generators such as model-to-Java, model-to-JDBC, model-to-JSP and model-to-XML as these code generators are specified at the unified meta model level.

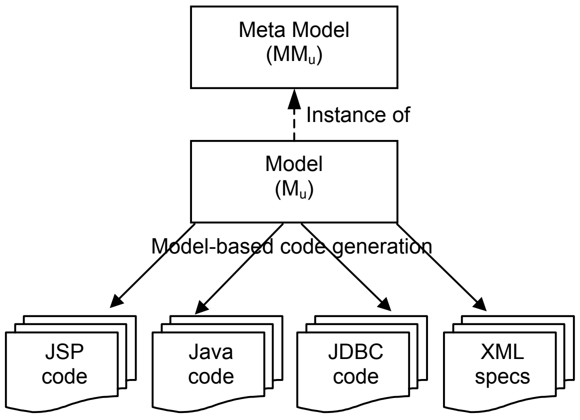

Fig. 3. Model driven development

We envisage several issues in decomposing a product line in terms of building blocks along F and P dimensions.

It is not clear which facets of a system deserve to be treated as aspects. There is a need to identify which of these aspects need to be separately specified. For instance, it is not clear how to cleanly separate the performance aspect from functionality. There is a need to investigate how these aspects can be modeled and what the right kind of abstractions for modeling them are to satisfy the various 'ities' like maintainability, reusability etc. For instance, how does one model a design for better maintainability?

Aspects may overlap each other. This may introduce a dependency on the order of their weaving. In such cases, how does one ensure that properties of all aspects hold after their weaving? An aspect specification may exist partly in model form and partly in code form. What's the right approach to integrate such aspects into the aspect modeling framework? A system is organized as a set of independently specified aspects. The knowledge of weaving an aspect is hidden inside the transformation. This gives rise to the issue of traceability from an aspect to the final implementation. It is not clear how to compute the impact of a change in an aspect on the final implementation of the system. This information would be critical for 'what if analysis', estimating testing efforts, managing releases etc.

Supporting separation of concerns for product lines using MDD raises several tooling issues. The modeling tool should be extensible to support new modeling languages. This is required to define new aspect models and relate them to the existing component models through model transformation mechanism. The model transformation tool should have adequate support for pattern matching and composition. It should provide support for incremental reconciliation of models. The performance of the tool should scale up to cater to the demands of enterprise class applications.

There should be tool support for intelligent debugging at aspect model level. This is significant because aspects are specified independent of each other and are woven together into the final implementation code. A bug detected at code level should be traceable back to the aspect specification.

There should be support, preferably tool-aided, for aspect-based testing. Since aspects are independently specified, it should be possible to specify test cases for an aspect independently and compose the test cases to arrive at the system level test cases.

4. Summary

In this paper, we presented an approach to support enterprise business application product lines as a software factory using model driven development techniques. We described a partial realization of this vision using multi-dimensional separation of concerns. We discussed several issues that need to be investigated to fully realize this vision. We also discussed some tooling issues.

Despite the many problems yet to be solved, we found that aspect-oriented restructuring of our MDD toolset has facilitated easy customization of the code generators and has resulted in increased reuse across their variants. Our MDD toolset has been used to develop several large enterprise class business applications for the past several years. These applications can be viewed as a set of vertical-specific product lines having toolset requirements that are *similar* but not exactly the same. Earlier, such a customization request meant opening up the implementation of the impacted tools that required expertise of all the tools to ensure the relevant changes are implemented in a consistent manner. Aspect-oriented restructuring has enabled us to organize the development team along two independent streams namely technology platform experts and design experts. A single design team can now service all the technology platform teams. Separation of design strategies has enabled leaner technology platform teams. Moreover, it has enabled our toolset itself to be organized as a product family wherein a tool variant can composed from design strategy and technology platform aspects of choice. Containment of change impact due to localization and increased reuse due to composability have led to quick turn around time for delivering a tool variant. Use of a higher-level model-aware transformation language has made maintenance and evolution of the product line easy.

We are working on addressing some of the open issues described earlier such as defining a process for identification, definition and extension of building blocks, and an infrastructure for defining, composing and processing domain specific languages of interest.

5. REFERENCES

[1] Don Batory, Jacob Neal Sarvela and Axel Rauschmayer, Scaling step-wise refinement, IEEE TSE, 2004

[2] K Czarnecki and U Eisenecker, Generative programming methods, tools and applications, Addison-Wesley, 2000.

[3] Jack Greenfield and Keith Short, Software factories: Assembling applications with patterns, models, frameworks and tools, Wiley, 2004.

[4] IBM research. Hyper/J: Multi-dimensional separation of concerns for Java. http://www.research.ibm.com/hyperspace/HyperJ/HyperJ.ht m

[5] K Kang, S Kohen, J Hess, W Novak and A Peterson, Feature-orientation domain analysis feasibility study, Technical Report, CMU/SEI-90TR-21, November 1990.

[6] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Longtier and John Irwin. Aspect oriented programming. ECOOP'97 LNCS 1241, pp 220-242. Springer-Verlag. June 1997.

[7] Vinay Kulkarni, Sreedhar Reddy: Model-Driven Development of Enterprise Applications. UML Satellite Activities 2004: 118-128

[8] Vinay Kulkarni, R. Venkatesh and Sreedhar Reddy. Generating enterprise applications from models. OOIS'02, LNCS 2426, pp 270-279. 2002.

[9] MasterCraft – Component-based Development Environment. Technical Documents. Tata Research Development and Design Centre. http://www.tata-mastercraft.com

[10] MOF Models to Text Transformation RFP http://www.omg.org/cgi-bin/doc?ad/05-05-15

[11] Peri Tarr, Harold Ossher, William Harrison and Stanley M. Suttom Jr. N Degrees of separation: Multi-dimensional separation of concerns. Proceedings of the International Conference on Software Engineering (ICSE'99) pp 107-119.