Integrating Domain Specific Modeling into the Production Method of a Software Product Line

Gary J. Chastek
Software Engineering Institute

John D. McGregor
Clemson University

Introduction

This paper describes a new context for the application of domain-specific languages (DSL) and modeling techniques: that of product production in a software product line (SPL). It briefly describes SPLs and how their products are produced. It then describes the role of DSLs and methods engineering in SPL product production. Specifically, we combine meta-modeling and other domain specific language practices together with the "method fragment and assembly" approach of method engineering. Finally, this approach is illustrated with an example SPL.

Product Production in a Software Product Line

The standard definition of a software product line is: "a set of software-intensive systems sharing a common, managed set of features that satisfy specific needs of a particular market or mission, and that are developed from a common set of core assets in a prescribed way." [Clements 02] Informally, core assets are created to be reused in the production of multiple products, thereby reducing the cost of, and time to produce, an individual product.

Product production in an SPL consists of

- A production strategy that describes how product production will contribute to the achievement of the business goals of the product line.
- A production method that specifies a complete set of processes, models, and technologies to be utilized in product production and that satisfies the production strategy
- A production plan that documents how a product developer will use the production method to produce a product from the available core assets [Clements 02], [Chastek 02].

The production method bridges the gap between the goal-oriented production strategy and the activity-oriented production plan. Method engineering views software development as the complete collection of processes, models, and technologies needed to develop a product and treats it as a single abstraction. The production method is engineered to achieve the goals specified in the strategy, and accomplishes this by integrating processes, models and technologies that share a common perspective on software development, e.g. an object-oriented software development method.

For the core asset developer, the production method ensures that core asset development is coordinated so that the resulting core assets work together effectively. The “attached process” created as part of a core asset describes how to use that asset when producing a product; it is a “fragment” of a product production process. For example, the software architecture is accompanied by an attached process fragment that describes how the architecture is specialized to provide the structure for a specific product. The production method defines the form these attached processes should take.

For the SPL product developer, the production method describes how to assemble the product production process by integrating the process fragments that accompany the selected core assets. The production method provides a template production plan which is instantiated by weaving in the needed attached process fragments. The method also provides the tools for using the plan to produce the desired product.

The contribution of our research is a framework for using method engineering to create the production method for a product line. In this paper we describe how the specific assets related to domain specific models and languages can be integrated into a software product line production method.

Goal-driven Method Engineering for Product Production

Using our tailoring of method engineering, the development of the product production method for a software product line begins with the development of a product production strategy based on the business goals for that product line [McGregor 04]. That strategy then guides the development of the core assets and their attached processes. The attached processes from the core assets selected to develop a product are then assembled into the product production method. Using the techniques of method engineering ensures a complete, correct, and consistent method definition.

The production method needs to be described in terms that will be easy for product developers to understand and for future production plan developers to reuse. The generic production plan is described with sufficient variation points that it applies to all of the products in the product line. The product-specific production plan is created by resolving the variation points using the attached processes from the core assets that are selected for the particular product. There are a number of techniques for doing this [Brinkkemper 99]. The Object Management Group (OMG) has published a Software Process Engineering Meta-model (SPEM) that has the basic elements for such a description [OMG 02]. In Figure 1 we provide a portion of the OMG standard SPEM model extended for a product line. We use a blend of these two approaches for our method descriptions.

The generic production plan is a partial instantiation of the production method. The generic plan defines the philosophy of the method and provides an overall structure including variation points at which specific fragments will be inserted. The generic production plan describes conceptually the content required of method fragments and defines a format for the fragment descriptions. It places constraints on the types of activities that can be inserted into the template method at each insertion point.

The product-specific production plan is the full instantiation of the production method. It contains production information for a specific product and completely defines the development method for the specific product. The technical content for each step in the method is also completely determined.

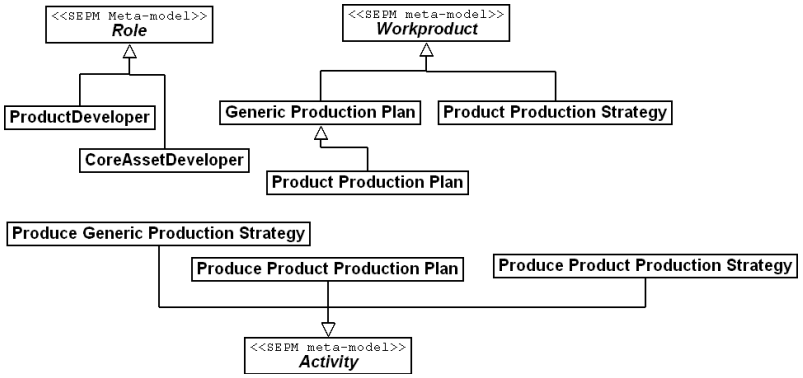

Figure 1 - Portion of SPEM for Product Lines

Domain-based Strategies and Production Methods

A domain-specific modeling approach to product production is clearly what we are referring to as a method. It affects how virtually every phase in the development process is carried out. Domain-based approaches provide processes such as analyze-model-generate, models such as meta and concrete domain models, and techniques such as meta-modeling that are unique to the domain specific method. This approach creates uses and complements the domain definition. Each of these artifacts, the processes, tools, and techniques, is compatible with the others so that a coherent whole is achieved.

Domain specific modeling is a natural approach for a product line since a number of the steps are intended to address the similarities and differences among the various products. Domain analyses of various types are a part of many software product line methods. For example [Clements 02] defines a product line practice area for "Understanding Relevant Domains." The integration of domain-specific modeling and software product lines is not so much a matter of changing either approach as much as it is weaving the activities of the two approaches together into a comprehensive development process. Then the models and techniques associated with each approach are used as needed to support the activities.

Domain-specific modeling (DSM) is integrated into product production through the production method. The initial product line activities such as product line scoping provide constraints on the content of the analysis. Product line analysis [Chastek 01] already includes a feature-oriented domain analysis. Product line analysis also produces an object model. The current purpose and content of this model is broadened somewhat to accommodate domain-specific modeling. In particular, the analysis is modified to provide a conceptual domain analysis that provides the basic inputs for creating the domain-specific language.

The product line architecture activity provides the structure of the products but also imposes some requirements on the domain specific language. In particular, the architecture defines the types of runtime relationships and constraints that the language must be able to express. Certain views used to describe the architecture relate its structures to the constructs identified in the domain model. For example, the features are mapped to specific architectural elements.

The component set will be constructed to implement the underlying architecture but it will reflect the domain model as well. The domain modeler establishes a link

between those components and terms and expressions in the domain-specific language. These connections are often encoded in the program generator.

In a product line using domain specific modeling, the core asset builders need guidance on how to design their assets to be compatible with the domain orientation. The production method provides this guidance. For example, the production method provides rules that software assets should be decomposed based on specific domain concepts and have domain-related names. The production method includes techniques for structuring the architecture to correspond to the natural domain divisions and to reveal the constraints between these divisions.

The product developers must understand which domain concepts to manipulate and how to exploit the relationships among those concepts. For example, the product developer must be able to specify a product using the domain specific language. The production method provides techniques for mapping the domain concepts onto architectural elements and then on to code assets.

The production method describes software development activities that are necessary to produce the product in the product line context, such as choosing variants at variation points. The method describes activities that are necessary because of the domain orientation being used, such as domain or feature analysis. The method engineer integrates these activities and others together to arrive at a development method that satisfies the production goals and achieves the required product qualities.

The attached process fragments are integrated by considering a producer-consumer relationship between activities. For example, the domain modeling activity produces a model that provides a set of entities that represent distinct concepts. The activity of choosing variants at variation points uses those entities as the possible choices for variant values. This activity produces a completed specification of the product.

The method engineer seeks to produce a production method that is complete, consistent, and correct. The method is complete if all the actions necessary to produce a product are described in the production process and the techniques needed to carry out those actions are provided. The domain specific language's vocabulary must be sufficient for specifying any products within the product line's scope, including all variant values and necessary constraints. The method is consistent if actions at one stage of the method do not contradict actions at other stages. The domain concepts defined in the model and implemented in the language must be well-defined. Finally, the method is correct if it produces products that meet their quality goals and the method achieves the production goals. The code generation from the domain specific language must produce code that compiles and executes correctly.

Case Study

Our case study is based on an example product line. The fictional company Arcade Game Maker[1] (AGM) has a product line of three different games: Brickles, Pong, and Bowling (see Figure 2). Each game is available in three variants: a simple PC-based game, a wireless device version, and a conventional give-away version that can be customized to the company giving it away. Further, a number of platforms are supported either through different operating systems or, in the case of the wireless version, a wide range of processors.

[1] Temporarily available at www.cs.clemson.edu/~johnmc/productLines/example/frontPage.htm

The AGM product line was implemented in three increments. Each increment comprises one variant of each of the games, such as the wireless device implementations of Brickles, Pong, and Bowling. Since a grouping of three products is often the break-even point between the cost of product line and individual product development, this division was also useful for decision-making (e.g., what language to use).

The following production strategy was adopted for AGM: We will produce the initial products using a traditional iterative, incremental development process using a standard programming language, IDE, and available libraries. We will create domain-based assets, including a product line architecture and software components, for the initial products in a manner that will support a migration to automatic generation of the second and third increment products.

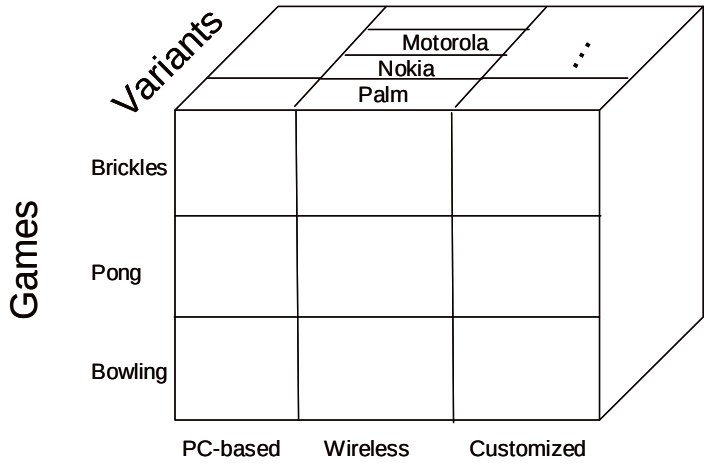

Figure 2 - The AGM Products Matrix

As part of the method specification we identified low-cost and simple as the qualities immediately required of the production method, with automatic to be added as the product line matured. The organization used goal-driven method engineering techniques to produce the production method from the stated production strategy. The resulting methods are briefly summarized in Table 1. The organization then produced a production plan from that production method. The product line organization is now ready to produce the third increment of products.

Table 1 – Development Method Summary

	Increment 1	**Increment 2**
Processes	RUP	RUP
	ADD	ADD
	ATAM	ATAM
Technologies	C#	Java
	Visual Studio	Eclipse
Methods	UML	UML
	Full set of architectural models Few design models	Full set of architectural models Full set of design models

The AGM product line organization could use any of several automation techniques. Their year of experience in producing game products has created many domain experts. The development staff is highly skilled in a wide range of technologies. A configurable software base in which a large code base is delivered and then configured is not a good choice since the wireless increment requires a small memory footprint for the products. Generation of the executable seems a reasonable approach since the software architecture binds all variation points at product definition time.

The team developing the production method for increment 3 (customized convention giveaways), decided to explore domain meta-modeling and eventually chose this as their approach to automation. They used MetaEdit+ as the modeling tool [Tolvanen 04]. In this approach, a meta-model is developed that essentially builds a modeling tool that implements a domain specific language. This tool supports the development of products within the domain described in the meta-model.

The AGM domain experts had already developed a couple of domain models. One was created using the Feature-Oriented Domain Analysis, which can be found in the product line artifacts [Kang 90]. The other was created using domain concepts, see Figure 3. The team developed a meta-model using the conceptual domain model and the basic architecture of the products.

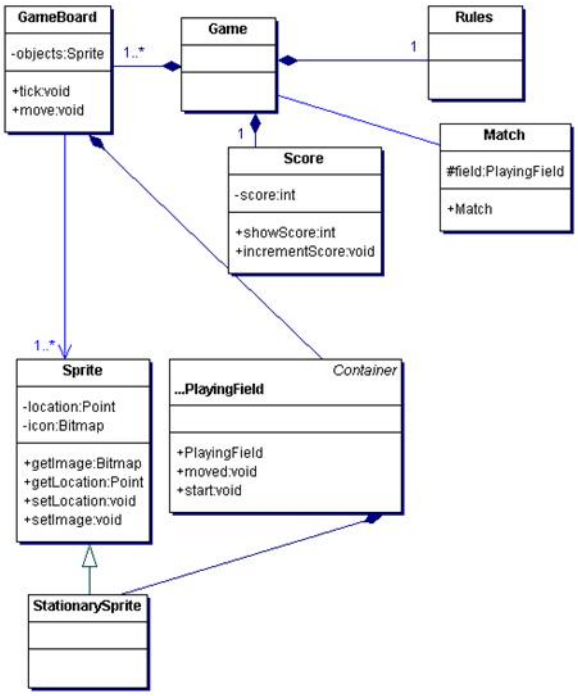

Figure 3 - Conceptual Domain Model

Figure 4 shows the wizard used to define a new Object type. In this case it is the GameBoard. The GameBoard contains two collections, one for MovableSprites and one for StationarySprites. The AGM team has used an object-oriented development method for many years. There were a number of false starts where modelers considered the meta-model to be the abstract classes that are the roots of inheritance

hierarchies. The proper separation of the modeling concepts and their implementations took several attempts to achieve.

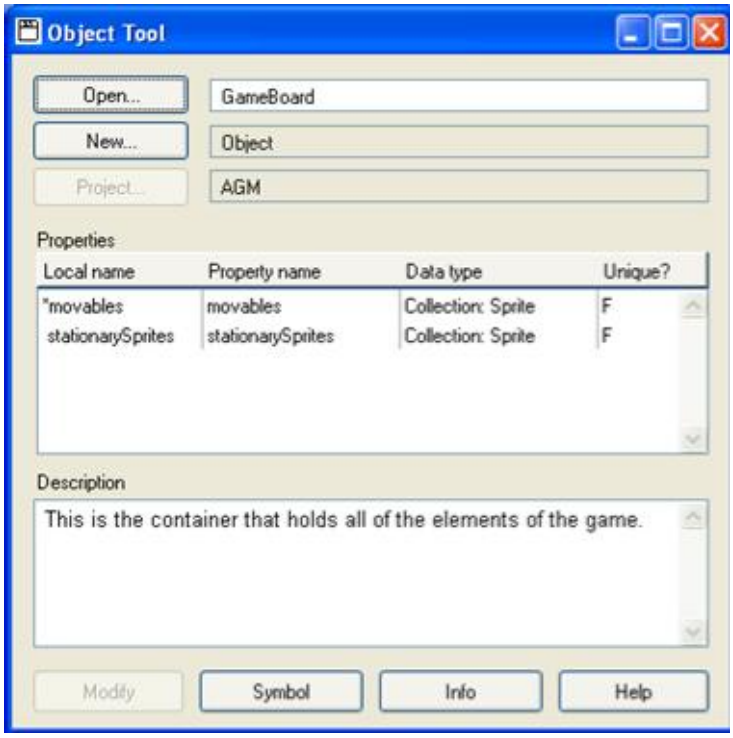

Figure 4 – Object Tool

The AGM team saw the need for a basic architecture around which the products are shaped. For the AGM product line, that architecture took the shape of a simulation loop. Each product follows the sequence of initialization, where the game elements are instantiated, simulation step, where the state and location of game elements are updated, and evaluation, where the states of game elements are checked to see if the score should be updated or whether the game is over. The last two steps are repeated until the game is terminated in some way. This basic architecture has a variation point between games that run continuously, such as Pong, and games that wait to be triggered by a user event, such as Bowling.

The asset developers associate each concept in the meta-model with a specific implementation. This is done using the report browser shown in 5. A series of reports are defined. A product will be produced by a top level report that calls the other reports necessary to produce the code needed for the product.

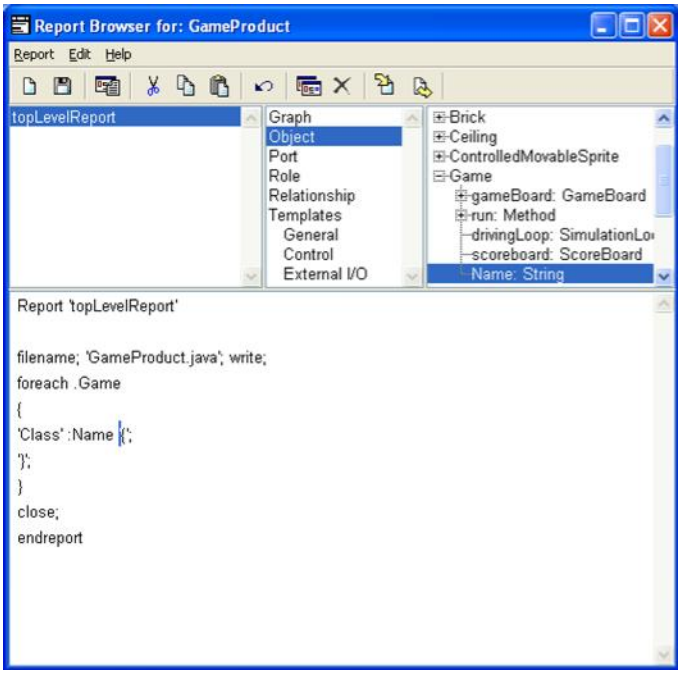

Figure 5 – Report Browser

The product builders use the dialog shown in Figure 6 to define a product. The product builder defines a new top-level graph. The graph links together instances of object types. The links enable interactions between the associated objects. The report associated with this graph will invoke the other reports associated with the members of the graph.

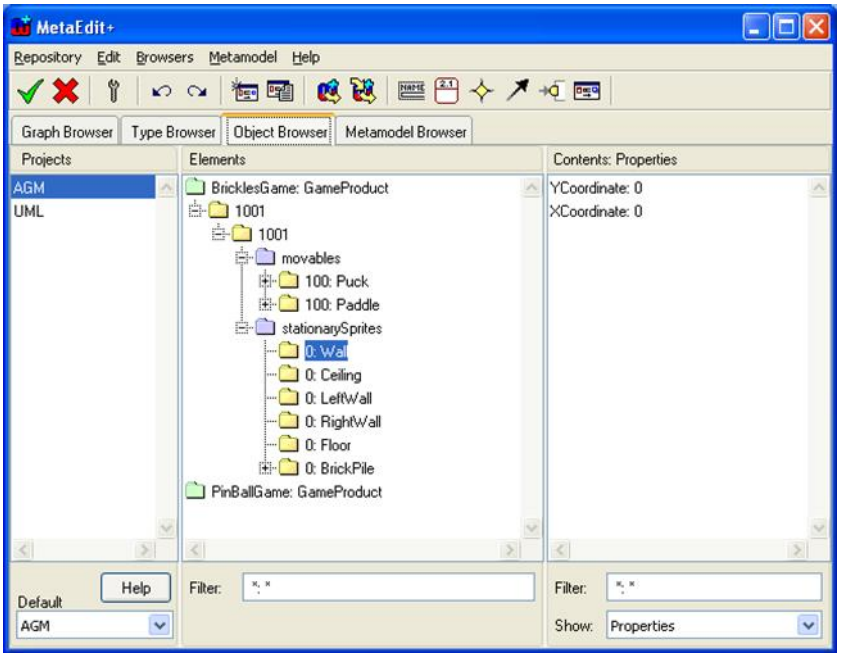

Figure 6 – Defining a Game Product

Conclusion

One approach to product production in a software product line is to create a domain-specific language tied, by a tool set, to a set of software components. We have

presented the notion of a production method that coordinates the processes, tools, and models used in SPL product production. The production method weaves these elements together so that the domain specific models and languages are an integral part of product production. The scope of a software product line makes the resources expended to develop the models and language a sound investment.

References

[Brinkkemper 99] Brinkkemper, Sjaak; Saeki, Motoshi; and Harmsen, Frank. Meta-Modeling Based Assembly Techniques for Situational Method Engineering. Information Systems. 24(3): 209 – 228, 1999.

[Chastek 01] Chastek, Gary; Donohoe, Patrick; & Kang, Kyo. Product Line Analysis: A Practical Introduction (CMU/SEI-2001-TR-001). WWW:<URL: http://www.sei.cmu.edu/publications/documents/01.reports/01tr001.html> (2001).

[Chastek 02a] Chastek, Gary & Mc Gregor, John. *Guidelines for Developing a Product Line Production Plan* (CMU/SEI-2002-TR-006). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. WWW: <URL: http://www.sei.cmu.edu/publications/documents/02.reports/02tr002.html>(2002).

[Chastek 02b] Chastek, Gary; Donohoe, Patrick; & McGregor, John D. Product Line Production Planning for the Home Integration System Example (CMU/SEI-2002-TN-029).

[Chastek 04] Chastek, Gary; Donohoe, Patrick; & McGregor, John D. A Study of Product Production in Software Product Lines (CMU/SEI-2004-TN-012).

[Chastek 05] Chastek, Gary; Donohoe, Patrick; & McGregor, John D. Applying Goal-Driven Method Engineering to Product Production in a Software Product Line (CMU/SEI-2005-TN-034).

[Clements 02] Clements, Paul & Northrop, Linda. *Software Product Lines: Practices and Patterns*. Boston, MA: Addison Wesley Longman, Inc., 2002.

[Kang 90] Kang, Kyo C.; Cohen, Sholom G.; Hess, James A.; Novak, William E.; & Peterson, A. Spencer. Feature-Oriented Domain Analysis Feasibility Study (CMU/SEI-90-TR-21, ADA235785). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1990.

[McGregor 04] Mcgregor John D. Factors in Engineering Strategically Significant Software Development Methods, Second Workshop on Method Engineering for Object-oriented and Component-based Development, October 2004.

[OMG 02] Object Management Group, Software Process Engineering Modeling, 2002.

[Tolvanen 04] Tolvanen, Juha-Pekka. Making model-based code generation work, Embedded Systems Europe, AUGUST/SEPTEMBER 2004.