Domain Models in the Synthesis Methodology

Grady Campbell

The objective of the Synthesis project, beginning in 1990, has been to define and disseminate a viable methodology for reuse-driven software development. Our approach was to bound the problem by restricting our scope to domains that arise from and are directly traceable to an organization's business objectives. The resulting methodology is documented in a comprehensive guidebook (RSP 1993) for experienced engineers and managers and has been validated as effective by projects in Rockwell, Boeing, Martin-Marietta, and Lockheed, including the Boeing/Navy STARS demonstration project.

In this position paper, we define, from a perspective of the reuse-driven/domain-specific engineering of software-intensive systems, what a domain model is and its possible uses. For purposes of context, we view a reuse-driven engineering process as being comprised of two activities: domain engineering and application engineering. Application engineering is the creation and support of an application product (that is, a system and all associated work products) for a customer. Domain engineering is the creation and support of an organization's application engineering capability. We refer to the product of domain engineering as a domain by which we mean to denote a product family, corresponding to a set of systems that will solve similar problems for a targeted business-area market, and a (application engineering) process for the production of individual products for particular customers.

What is a Domain Model?

A model is a representation of a thing (its subject) that enables the determination of (approximate) answers to designated questions about the thing itself. Within the context of Synthesis, a domain denotes a set of systems that solve similar problems. Therefore, a domain model is any representation of a set of systems that enables the answering of designated questions about those systems or about the problems they solve.

A domain model should enable answers to questions about the requirements of systems within the domain, about the design of those systems, and about the implementation of those systems. Because a model by its nature is only an approximate representation of its subject, a domain model will not provide answers to all possible questions and the answers it can give may be only approximations of the true answer. A model is, however, acceptable if its answers are sufficiently accurate (not misleading) within explicit bounds and they are obtainable at much lower cost and effort than is otherwise obtainable (for example, by observing the subject itself).

In Synthesis, the subject that a domain model represents is a product family. A product family is described in terms of the requirements, design, and implementation of products in a domain and the variations that distinguish among those products. For flexibility, a product family can be represented informally, in terms of conventional forms of work products used by practicing engineers, or it can be represented in more formally-defined notations.

What are the Uses and Criteria of a Domain Model?

There are four potential uses of a domain model from our perspective: as a shared repository of the knowledge that underlies a domain, as a resource for educating engineers about the domain (that is, to impart domain knowledge), as a predictor of properties of systems within the domain, and as a formalization for generating a product. A domain model, for any of these uses, may be passive, enabling questions to be answered by inspection of represented knowledge, or active, responding to a set of prescribed questions whose answers are derivable from that knowledge.

The first potential use, as a repository of shared knowledge, is fundamental. Any domain model inherently serves this purpose, although it may only be implicit. For many organizations, this use alone is sufficient to justify the creation of a domain model. Although this use requires a disciplined process of knowledge acquisition, the domain model may be informally expressed. Communication among knowledgeable parties does not require formally-defined notations but does lend itself to specialized (concise or abbreviated) notations. This informal level alone is sufficient to enable effective reuse-driven software development.

The second potential use, as a resource for education, is directly derivative of the first use. This use, however, is not implicit to the first use in that the domain model may not be sufficiently tutorial. It may require some level of knowledge and experience before it can be understood. For tutorial use, the domain model must include not only expert-level shared knowledge about the domain but be elaborated with novice- and intermediate-level explanations, rationale, and examples.

The third potential use, as a predictor of systems' properties, is closest in spirit to the domain model as a model. For this use, there must be an understanding not only of the problems that included systems solve and how those systems are constructed but also why the systems are constructed that way and the trade-offs in alternative constructions. The difficulty of creating such a model depends in large part on the questions that application engineers need it to help them answer. Although a formalized notation is not necessarily incompatible with the preceding uses, it is essential to this use because many properties are derivative of and interact with other properties rather than being explicit in domain knowledge.

The fourth potential use, as a formalization for generating a product, requires the expression of domain knowledge in sufficient detail to define a transformation from perceived needs (that is, a problem) to a product (a corresponding solution). Some approaches to this, arising from research in artificial intelligence and software process modeling, propose a formally-defined notation and associated sets of domain-independent and domain-specific transforms intended to mimic the process that engineers follow to create software. The approach in Synthesis, arising from research in software engineering methods, is to express domain knowledge in a 'compiled' form, as a set of adaptable work products (after all, work products are in essence compiled domain knowledge). This use requires more formalized expression of domain knowledge at greater levels of explicit detail over greater breadth than any of the other uses.

A comprehensive domain model within the context of reuse-driven engineering must define the motivations, form, and content of a product family. This includes the form and content of all associated work products, encompassing requirements, design, and implementation viewpoints. It must be understandable by other domain experts and usable by application engineers as far as is required by the application engineering process to create acceptable products cost-effectively. An effective paradigm for domain engineering goes further, providing a framework for the concurrent engineering of a product family and an associated production process.

Methods for Domain Analysis

The Synthesis guidebook (RSP 1993) prescribes methods for domain engineering consonant with the objective of reuse-driven engineering. These methods guide domain engineers who are domain and software experts to organize and express domain knowledge in a form that enables a streamlined application engineering process. Domain engineering methods address management, domain definition, product family engineering, process engineering, and (application engineering) project support. Methods for developing a domain model fall within product family engineering and build

on methods for developing system/software requirements, design, and implementation models, extending them to consider the implications of variability across a family.

The Synthesis approach constrains a domain to a set of systems that meet similar needs. Systems that meet similar needs tend to be viewable as having well-constrained variabilities within a framework of substantial commonalities. By extracting the variabilities among the systems (and their associated work products) and formulating them as a set of deferred decisions representing the supported range of needs, it is viable to construct adaptable work products to which resolved decisions, based on particular needs, can be applied to create a tailored product. A product, and therefore the application engineering process, becomes a function of the deferred decisions that characterize the domain represented by a product family.

References

RSP 1993 — Reuse-Driven Software Processes Guidebook, SPC-92019-CMC, version 02.00.03. Herndon, Virginia: Software Productivity Consortium.

Acknowledgments

This material is based in part upon work sponsored by the Advanced Research Projects Agency under Grant # MDA972-92-J-1018. The content does not necessarily reflect the position or the policy of the U.S. Government, and no official endorsement should be inferred.