JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2004**

Vol. 3, No. 7, July-August 2004

Domain *

John D. McGregor, Clemson University and Luminary Software LLC, U.S.A.

Abstract

Domain plays an increasingly important role in our business. In fact, it is our business. The wildcard in the title indicates that there are several domain-related issues in software engineering. This time in Strategic Software Engineering I want to explore some of the implications of the increased recognition of the role of domain in software engineering. I will contrast a domain-based approach to a requirements-based approach and present a high-level domain-driven development process.

1 INTRODUCTION

Domain analysis, domain engineering, domain-specific languages, and many other wildcard matches apply to the term ***domain***. While some of the software we produce, such as debuggers and compilers, manipulates other software most of our software manipulates something else. The *something else* is the area in which we do business, our domain.

Domain may be thought of in several different ways. Many of the technical definitions view a domain as the subject for a family of programs, for example telephone call switching systems. Yet literature in fields other than programming uses the term domain as well. The definition that I like the best states that a domain is *a body of knowledge*. I like this definition because it decouples the real-world domain from software-based implementations of the domain. I also like it because it still leaves room for interpretation.

Domain is *what* a piece of software is about. At one level of detail it may be banking transactions or, at another level, it may be user interface controls. As we separate the implementation from the specification, the domain becomes the central focus of the specification. Product development is a effort to identify and separate the many domains involved in solving a problem, describe the domains in models and languages, and integrate them into a product.

Domains have a lifecycle that we can use to guide forward-looking decisions about products. Domains are initially ill-defined and rapidly changing. The software products that implement the domain are handcrafted for the individual business process of a specific enterprise. Users of these products must have an understanding of the business

Cite this column as follows: John D. McGregor: “Domain *”, in *Journal of Object Technology*, vol. 3, no. 7, July-August 2004, pp. 71-81. http://www.jot.fm/issues/issue_2004_07/column6

process independent of the product, think bank tellers. As the domain matures, we build software products that use standardized processes and these products can be sold to multiple customers with little or no modification. Eventually the domain is sufficiently stable for us to identify patterns and to abstract these to standardize product elements. Users can now be customers of the former trained users, think ATM users. The domain becomes sufficiently standardized that even end users can write scripts that embody frequent operations, think intelligent agents. Successful products enter a domain's market at just the correct point in the lifecycle.

Domain knowledge is a commodity. Software architects with extensive knowledge of telecommunications will be offered more by a telecommunications company than a person with just as much experience creating architectures in some other domain. The products of many companies implement a specific piece of domain functionality, such as an encryption algorithm, rather than being an end user product. A number of consortia have developed domain models available only to paying members.

The thesis of this article is: taking a domain-based approach provides a context for software development that creates synergy with other business activities of the company producing strategically significant results. Much has been written about focusing on the "core business." In fact, this has been one of the driving forces for outsourcing. The company focuses on what it does best and pays for others to do supporting activities. Domain-based development allows the software development group to focus on the core business of the organization just as the rest of the product development group does.

I will discuss a domain engineering approach to software development that seeks to make software development an integral part of strategic planning and product production within the organization.

2 REQUIREMENTS-BASED AND DOMAIN-BASED APPROACHES

One software development firm advertises, "We build exactly what you want." That sounds very good to a project manager who has a tight deadline for her project, but it is very bad for the organization that needs to explore opportunities for new or improved products. In a requirements-based approach to development, a team analyzes requirements to understand the specific product to be built. A team, perhaps a different one, then organizes to build the product. Communication between these teams and others in the development effort is in terms of the specifics of this product. The team may approach this as a completely independent effort from anything done before or they may try to reuse code from previous products. The structures created in this type of environment usually are implementation focused and are very difficult to use even in a slightly different context.

In most development efforts, over half of the faults are related to incorrect requirements or to misunderstanding the requirements. Ultimately no development effort can be successful unless what is to be built is clearly and correctly understood. One of the

improvements in requirements writing was Jacobson's use case concept. Use cases were seen as an improvement because of

- the ability to structure the use case model and show explicitly the relationships among the use cases. This is an improvement over a bulleted list of short phrases or a text document full of *shall*s.
- the use of scenarios in the use case. A good scenario helps you understand the problem in the context of its domain. Of course these short stories can be just as unilluminating as the bulleted list if they digress into details of the user interface or specific implementations.

Use cases are part of the move toward domain-based development.

One of my early explorations into object-orientation came when I was working on a project to develop an automatic programming system that would help physicists solve systems of partial differential equations. We were trying to understand the problem and creating a working model of the concepts involved in the problem seemed a useful approach. This domain-based approach provided an effective basis for exploration and learning. We eventually cast the results of the domain analysis in a domain model and a library of classes, which was our equivalent to a domain-specific language.

We then worked with several development teams to build products from the domain infrastructure. A number of problems had to be solved while building these products, but they involved issues of implementation and meeting performance objectives. Throughout the effort we were able to talk with the scientists who would use the products and they were able to understand what we were doing since it was stated in domain terms.

The domain-based approach provided our extended team with a common basis for understanding. It provided concepts, vocabulary, and relationships among the concepts. These artifacts were easily reused on the development of several products in the same domain.

To an extent the domain-based approach can be thought of as subsuming the requirements-based approach. In a domain-based approach to development, a team analyzes a broader domain before analyzing the requirements for a specific product. The FODA technique used in software product lines is an example [Kang90] of a technique that provides a natural transition from domain modeling to requirements analysis. In short, domain provides a context for asking *what a product should be* while a requirements-based process asks *what a product should do*.

Domain-based development enhances communication between business and development personnel promoting reuse across products and in some cases even reuse across organizations. The Object Management Group's Catalog of OMG Domain Specifications currently contains specifications for over 30 domains [OMG04]. Each specification provides a set of interfaces that describe the concepts of the domain. These specifications allow a vendor to create products that can interact with other products within the same domain without direct interaction among the vendors as the products are built.

Brian Foote [Foote00] discovered a useful pattern, "Deploy People Along the Grain of the Domain." This pattern takes advantage of the fact that as we have more experience with a particular piece of functionality, we get more insights into how to improve it. Sounds obvious, but traditional deployment of personnel develops expertise in a portion of the process, such as analysis, as opposed to gaining experience in some slice of the domain from analysis through implementation. This works as long as each new project is disjoint from previous ones. Brian's pattern is most effective when there is a longer term vision beyond the current product.

3 DOMAIN ENGINEERING

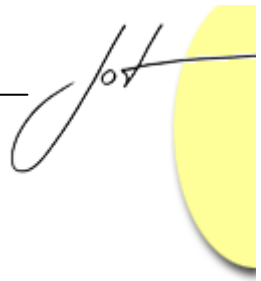
Some in the software product line community have adopted the term *domain engineering* to refer to the various activities related to the domain and the development of reusable assets[1]. In fact, the term is used to refer to all of the activities that are related to the product line as a whole as opposed to those activities related to individual products. This terminology is chosen in recognition that reuse-oriented tasks such as building the business case and identifying the scope of the product line are inherently domain-related.

Domain analysis is perhaps the most mature component of domain engineering. Prieto-Diaz [Prieto-Diaz87] provided some of the early work in this area along with Kang [Kang90]. Hewlett-Packard has applied domain analysis to enhance the reusability of software elements [Cornwell96]. The idea is to understand the concepts in the domain and then to translate those into objects in working programs. The domain objects form the core of most good object-oriented software. That is not to say that the domain objects define the software architecture, more later.

Czarnecki [Czarnecki00] describes a domain-based library development approach that goes beyond traditional domain analysis. The approach takes the software-specific definition of domain I mentioned above. That is, the domain is defined by studying applications that share common requirements or features. They build one model on the abstract data types (ADT) that are used in the implementations of the applications. They build a feature model using Feature Analysis to capture the features provided by the applications. **Figure 1** lists the steps in the process.

Weiss et al [Weiss99] provide one approach to a complete product development approach: *f*amily-oriented *a*bstraction, *s*pecification, *t*ranslation (FAST). The FAST approach to product line development focuses on doing sufficient analysis and design to create a domain-specific language. Product builders specify the desired product by writing a program in the language and the product is produced automatically using program transformations.

[1] Those who use the term domain engineering for the product line level also use application engineering as the product level organization. These terms are roughly equivalent to the core asset builder and product builder terms used in the Software Engineering Institute's product line model, but there are differences [Clements01].



Domain Analysis
- Domain Definition
- Domain Modeling

Domain Design
- Architecture creation
- Specification of domain-specific language

Domain Implementation
- Implementation of domain-specific language
- Implementation of components

Figure 1 - Domain Engineering Method For Algorithmic Libraries

I want to put the activities in **Figure 1** into the context of a "complete" development process, a portion of which is shown in **Figure 3**. The initial step is to identify the set of domains that completely encompass the problem being solved. This implements the "separation of concerns" pattern. The work of Tarr and Ossher [Tarr99] and others at IBM has resulted in a concern manipulation environment (CME), implemented as an Eclipse plug-in. The CME supports, among other things, tracking the elements that constitute a single domain but that are distributed over several physical assets such as files or class definitions. In **Figure 2** I show a query, lower left window, in the CME that searches for all classes whose name includes the word "Array". This finds the core domain classes for my application. In the visualizer, middel window on the right, the blue bar near the top of some of the rectangles identifies the line of code that contains the class name definition. The visualizer is also showing the relative size of each class file by using different size rectangles.

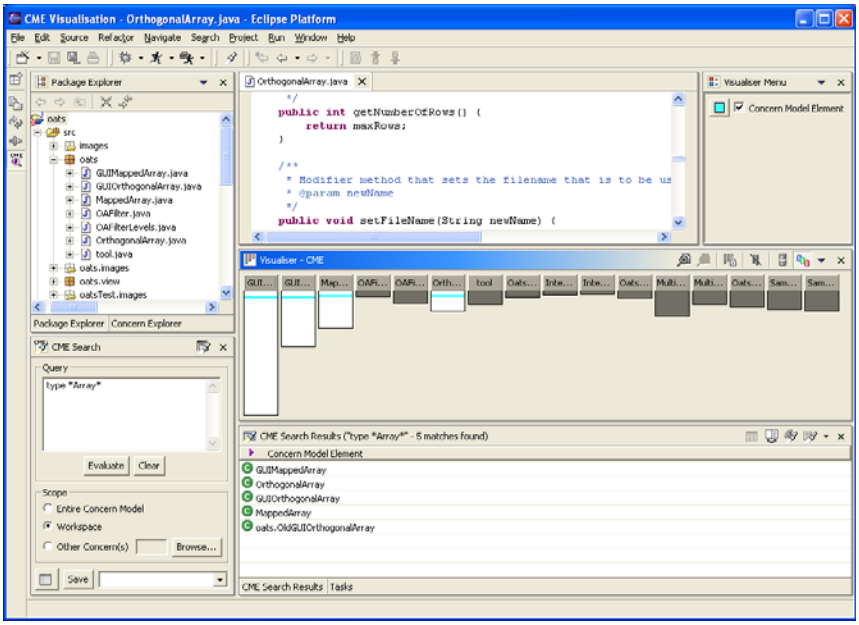


Figure 2 - CME operation

Some of the domains deal with the business concepts involved in the problem. Others are implementation domains such as the windowing functionality. Typically, some of the domains will have already been analyzed and a domain-specific language defined, perhaps as a set of classes such as the javax.swing.* packages in Java. Where necessary, a domain is engineered following the process shown in **Figure 1**.

The tricky part comes in the final block in **Figure 3**. The separate streams of design must be integrated. This is the role of the software architecture. The structures of the domain models represent semantic relationships among elements but do not necessarily represent the best structure for an executing system. The software architecture determines execution time relationships among elements from several domains and among elements within a single domain. A number of architectural patterns are used for this integration including the aspect pattern that cuts across elements and the plug-in pattern used by Eclipse. In the Eclipse open source project a number of domains, such as visual editing and entity modeling, are being investigated and designed concurrently. They are integrated using a registry that lists all plug-ins located by an instance of Eclipse and providing services to allow one plug-in to find the other plug-ins upon which it depends.

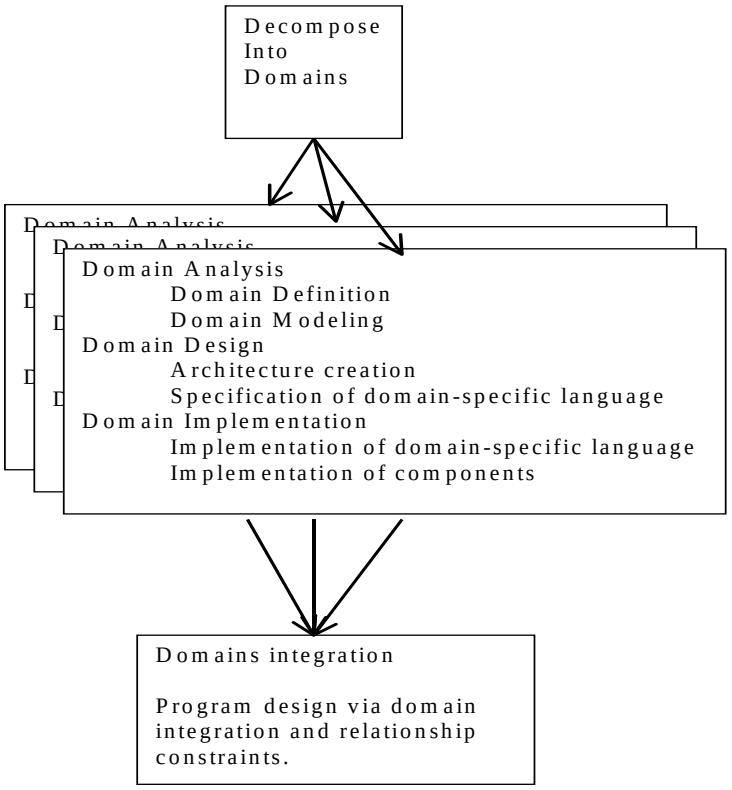


Figure 3 – Domain-driven Development

4 DOMAIN STRATEGIES

Technical domain engineering processes are usually presented in isolation. In order for me to fulfill my earlier promise, I need to tie these processes into the broader enterprise level strategizing process. Corporate planning and strategy development interacts with the two levels of engineering in the product line as shown in Figure 4. For the purposes of this discussion I am ignoring tactical management activities.

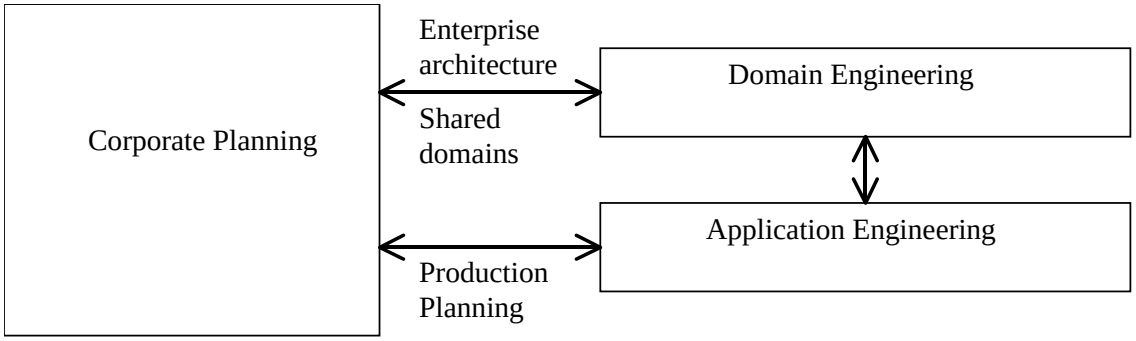


Figure 4 - Strategic connections

The product line organization is constrained by the enterprise architecture. An enterprise architecture is used to assure that various facets of the enterprise use compatible, if not the same, technologies. Even companies that do not have an enterprise architecture often have corporate standards for look and feel of user interfaces and standard definitions for certain key concepts.

The fundamental strategy of an enterprise is to address a specified set of domains; the markets that the enterprises targets with products. A product line organization probably will not be the only business unit within the enterprise to address a certain market. Domain information will be available to the product line organization. There will at least be domain experts and there may also be domain assets that either can be used in the product line or perhaps can flavor the development of new assets.

The product line organization may be constrained by the production techniques used at the corporate level. There are several reasons for this constraint. There are often assets that the enterprise owns that can be used in the product line, but these assets have already been engineered to be used in particular ways. For example, the enterprise may have invested in developing a program generator and a set of templates. The product line staff have experience using the generator, gained on numerous previous projects, that must be leveraged to full advantage. Adopting a totally new production process would be counter productive.

Several early adopters of the product line approach have specifically sought benefits related to the domain of application. They have experienced improved competitive

advantage, the ability to address niche markets within their domains of interest through customization of domain assets, and the ability to respond to unanticipated opportunities through that same customization.

5 EXAMPLE

In my first column I introduced an example product line of arcade games. The Arcade Game Maker (AGM) is the fictional operator of that product line. I want to consider the role of domain in that product line.

Figure 6 shows a domain model for the Game domain used in the product line. The classes in the class diagram form a domain-specific language in a narrow sense of the term. I can write lines of code such as shown in **Figure 5**. In these lines the object types and the method names are all domain terminology. In languages such as Java and C#, the strong type checking ensures some degree of compliance with the domain.

```
StationarySprite ss1 = new StationarySprite();
StationarySprite ss2 = new StationarySprite();
ss1.hit(ss2);
```

Figure 5 - Domain code example

AGM is building nine products within this domain. In fact, AGM already has built a number of products in this domain. The staff assigned to the product line organization bring with them expertise in the domain even before the model is developed.

The domain objects in this product line contain non-domain functionality. The architecture is a model-view-controller pattern, but the decision was made to implement portions of the view in the same classes as the model since there would only ever be a single view. The domain model in **Figure 6** shows only methods related to the domain. The architecture-specific methods were added after the domain had been modeled and understood.

The concern manipulation environment was not available when we developed this example product line. It would have been a useful tool for tracking the portions of each class related to domain and the portion related to application-specific details.

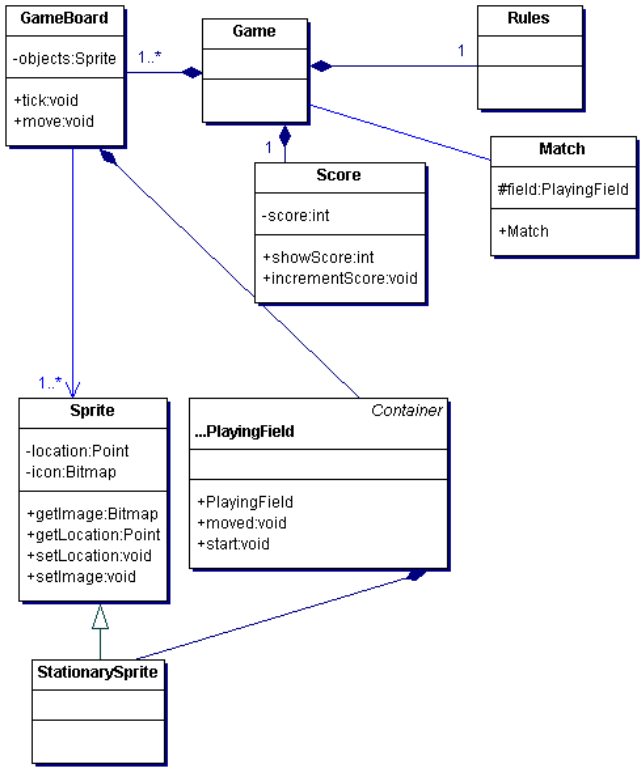


Figure 6 – Partial Domain Model

6 SUMMARY

I have illustrated some aspects of the domain-driven development approach to software-intensive product development. It is a useful perspective but it is not always the most appropriate approach.

A requirements orientation is appropriate when

- a single product is to be built as quickly as possible
- the main domain of the product is changing so rapidly that elements will not be reused
- a product is being built in an established domain in which the staff has extensive experience (this assumes that a domain-based approach was followed at sometime previously)

A domain orientation is appropriate when

- the implementation is not particularly specialized and the development staff can focus on the problem.

- the domain is not well understood by the development staff and experimentation with solutions will focus on domain algorithms rather than innovative implementations.
- There is an emphasis on reuse as in a software product line

I have participated in a number of projects using both approaches. My experience has shown that for a company that produces software-intensive products for specific markets the domain-based approach will make the most sense in the long run but only if the enterprise is sufficiently disciplined to take advantage. The industry's experience is much like my own. In recent, and so far unpublished, research on software product lines, over half of the industry representatives indicated that they are using domain-oriented techniques. Many of the recent trends and initiatives are heading in the direction of domain *.

REFERENCES

[Clements01] Paul Clements and Linda Northrop: *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2001.

[Cornwell96] Patricia Collins Cornwell: "HP Domain Analysis: Producing Useful Models for Reusable Software", *HP Journal*, August 1996.

[Czarnecki00] Krzysztof Czarnecki and Ulrich Eisenecker: *Generative Programming: Methods, Tools and Applications*, Addison-Wesley, 2000.

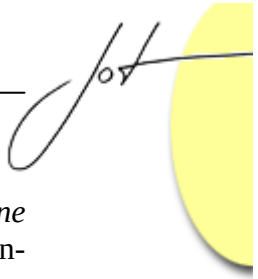
[Foote00] Brian Foote: "Deploy People Along the Grain of the Domain". *Seventh Conference on Pattern Languages of Programs*, Honolulu, Hawaii, 2000.

[Kang90] Kyo C. Kang; Sholom G. Cohen; James A. Hess; William E. Novak; and A. Spencer Peterson: *Feature-Oriented Domain Analysis Feasibility Study* (CMU/SEI-90-TR-21, ADA235785). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1990.

[OMG04] Object Management Group: Catalog of OMG Domain Specifications, http://www.omg.org/technology/documents/domain_spec_catalog.htm, 2004.

[Prieto-Diaz87] Reuben Prieto-Diaz: "Domain Analysis for Reusability", in *Proceedings of COMPSAC 87*, October 1987.

[Tarr99] Peri Tarr, Harold Ossher, William Harrison, S. M. Sutton, Jr. (1999). N "Degrees of Separation: Multi-Dimensional Separation of Concerns", *Proceedings of the 21st International Conference on Software Engineering*, May 1999.



[Weiss99] David M. Weiss and Chi TauRobert Lai: *Software Product Line Engineering: A Family-Based Software Development Process*, Addison-Wesley, 1999.

About the author

Dr. John D. McGregor is an associate professor of computer science at Clemson University and a partner in Luminary Software, a software engineering consulting firm. His research interests are software product lines and component-base software engineering. His latest book is *A Practical Guide to Testing Object-Oriented Software* (Addison-Wesley 2001). Contact him at johnmc@lumsoft.com.