PANEL: Objects and Domain Engineering

Sanjiv Gossain, *Cambridge Technology Partners (moderator)*
Don Batory, *University of Texas at Austin*
Hassan Gomaa, *George Mason University*
Mitch Lubars, *Scientific and Engineering Software Inc.*
Christopher Pidgeon, *Cambridge Technology Partners*
Ed Seidewitz, *NASA Goddard Space Flight Center*

Objects are said by many to be a natural way of looking at the world. Object-oriented analysis, with its objects, relationships, and scenarios are purported to be a good way of modelling a domain.

Domain engineering is a field that has been emerging as a key component of any domain-specific reuse strategy. Domain engineering is a means of analysing and modelling a problem domain with a view to reusing the concepts of that domain across multiple software systems. There are domain analysis approaches currently available that are not based on object-oriented techniques, yet an object-oriented approach would seem to some to be the most suitable for domain engineering.

Panellists will be asked to use the following questions as a guideline in addressing how object technology may be used in domain engineering:

1. Can domain analysis and modelling be achieved without objects?
2. What aspects of object-oriented technology are best suited for domain engineering?
3. Are there any object analysis methods especially suited to domain engineering?
4. What limitations do objects impose when analysing and modelling domains (rules, representation of domain semantics, etc.)?
5. What is missing from object-oriented concepts that is required for effective domain engineering?
6. Based upon their practical experiences, what are the critical success factors in object-oriented domain engineering?

Extracts from the position papers of the panellists are presented here.

Mitch Lubars

Object-oriented analysis methods provide several useful mechanisms for supporting the analysis and expression of domain models. In particular, superclasses can be used to express the domain's static commonalities, while subclasses can express the domain's variabilities. Mix-ins can further help separate the different orthogonal variabilities for a given class of objects. Polymorphism and frameworks are especially valuable to highlight the behavioral commonalities and variabilities for domain objects, but they have to be carefully structured to express the interfaces and implementations that are common, while highlighting the variable methods to be chosen or customized.

One important addition for object-oriented methods to support domain analysis is the ability to guide developers in choosing different sets of mutually consistent classes, mix-ins, and methods. These can be described using issue-based or feature-oriented approaches, so that sets of object-oriented variabilities can be linked to specific features or user requirements. That kind of information can then be used by developers as a guide for how to select and customize variabilities when constructing new systems in the application domain.

Dr. Mitchell D. Lubars is a senior software engineer with Scientific and Engineering Software, Inc. His current work focuses on supporting automated code generation from SES' Objectbench CASE tool. Dr. Lubars recieved the A.B. degree in biology from Cornell university, in 1977. He recived the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana Champaign, in 1980 and 1986 respectively.

Hassan Gomaa

At George Mason University, a project is underway to support software engineering lifecycles, methods, and environments to support software reuse at the requirements and design phases of the software lifecycle, in addition to the coding phase. A reuse-oriented software lifecycle, the Evolutionary Domain

Lifecycle, has been proposed, which is a highly iterative lifecycle that takes an application domain perspective allowing the development of families of systems. Current emphasis is on the domain analysis and specification phase of the EDLC for developing an application domain model, which captures the similarities and variations of the domain.

The EDLC model incorporates two related sub-life cycles, domain modeling and target system generation. Domain modeling deals with developing the reusable requirements and domain specific software architecture for a family of systems, while target system generation deals with generating target systems from the domain model.

In a domain model, an application domain is represented by means of multiple views: aggregation hierarchy, object communication diagrams, state transition diagrams, generalization / specialization hierarchy, and feature/object dependencies.

This last view relates the end-user's perspective of the domain, namely the features supported by the domain, to the object types in the domain model. It shows for each feature (domain requirement) the object types required to support the feature.

A proof-of-concept experiment has also been carried out to develop a prototype domain modeling environment, which consists of an integrated set of software tools that support domain modeling and target system requirements elicitation.

The domain modeling environment has been used for modeling several different application domains. In addition to NASA's Payload Operations Control Center (POCC) and Transportable Payload Operations Control Center (POCC) domains, two other application domains, a manufacturing domain and a banking federation domain have been modeled. This demonstrates that the environment is indeed domain independent.

We are currently investigating extending the environment to support the design and implementation phases of the EDLC. In particular, we are investigating interfacing the domain modeling environment to the Regis distributed configuration environment.

Hassan Gomaa is a Professor in the Department of Information and Software Systems Engineering at George Mason University, Fairfax, Virginia. He has over 25 years experience in software engineering, both in industry and academia, and has published over 70 technical papers. He received his B.Sc.(Eng.) in Electrical Engineering from University College, London and his DIC and Ph.D. in Computer Science from Imperial College, London.

Christopher Pidgeon

In an engagement with the Michigan Department of Transportation to build a suite of transportation planning decision support tools, Cambridge Technology Partners used a framework to map from the referent system to a conceptual system to an object system in order to: 1) understand the transportation planning problems and 2) plot a path to an integrated collection of solutions to some of those problems.

The most challenging aspects of this engagement included:

1. The roles of syntax and semantics in moving from the referent system to the intermediate domain concept map and ultimately to an implementation. We were really hobbled by language: Transportation experts used transportation speak (with several dialects e.g., roadway, bridge, safety, congestion, intermodal, public transit); object modelers used object speak. By introducing the intermediate level of domain concepts with a focus on the commonality of purpose for concepts in transportation planning, we were able construct a communication environment conducive to effective system definition and building.

2. Maintaining consistency of representations for elements of the knowledge triangle—concepts, symbols, and referents. Here the difficulty was two fold: the volume and diversity of concepts (intensional thoughts, ideas, senses) symbolized with symbols (words, icons) and referring to referents (extensional objects) quite often overwhelmed our representation schemes.

Christopher W. Pidgeon is a Senior Technical Consultant with Cambridge Technology Partners, an international professional services company specializing in the rapid delivery of open, distributed computing solutions to strategic business problems.

Dr. Pidgeon's research focus is practicable reusable software engineering.

Don Batory

Over the last ten years, our research has focussed on domain modeling methodologies and implementation techniques for creating software system generators. To us, a domain model is a component-based, parametric definition of a family of related systems. Every family member is defined by a unique composition of components. The similarities and differences among different family members are exposed by comparing the component compositions that define them. An implementation of such a model is a software system generator.

The modeling methods and implementation techniques that we have developed are expressed in the GenVoca model. The GenVoca approach to software system generation relies on standardized and layered decompositions of software systems. The building blocks of software systems are components, which are layers. Components both export and import "standardized" interfaces so that they act like software "legos". That is, target software systems are hierarchical compositions of plug-compatible components. From an object-oriented perspective, components are generally not individual classes, but are subsystems - i.e., suites of interrelated classes. Thus, a component's export interface typically consists of multiple classes and a component itself encapsulates the implementation of these classes. A realm is a library of plug-compatible components.

GenVoca realms and object-oriented frameworks are related, although the exact relationship is not yet fully understood. An essential aspect of framework design to express collaborations among abstract classes as "template" operations; it is not obvious where this counterpart lies in GenVoca. Our work has focussed on developing a clean abstract model of component composition; we do not yet know how the composability of frameworks relates to our model.

Don Batory is an Associate Professor in Computer Sciences at the University of Texas at Austin. He was a member of the ACM Software Systems Award Committee from 1989-1994. He was an Associate Editor of ACM Transactions on Database Systems from 1986 to 1992, and an Associate Editor of IEEE Database Engineering from 1981 to 1984. He was the Program Committee Chairman for the 3rd International Conference on Software Reuse (November, 1994).

Ed Seidewitz

For over five years, the Flight Dynamics Division (FDD) at NASA's Goddard Space Flight Center has been carrying out a detailed domain analysis effort and is now beginning to implement Generalized Support Software (GSS) based on this analysis.

Throughout this process there has been a continual tension between keeping the concepts as simple as possible and assuring that they are powerful enough to allow specification of domain functionality without undue complication. The core concepts of the model include the basic object-oriented principles of classes, objects and messages. Additional concepts have been added to this core only when not including the new concept would make it difficult or impossible to clearly specify some specific domain functionality under consideration.

Following our specification concepts, the current GSS specifications are defined with more detail and less ambiguity than typical FDD software specifications. This has had a positive impact on the development process, since class specifications are generally detailed enough to serve themselves as program design language for the implementors.

The specification concepts need to be updated to improve the description of how classes interact to support algorithms. For example, we need to improve the overview documentation in the specifications to explain these interactions.

The specification of dependencies between classes, together with the generalized design for dependencies, completely captures a functional architecture for GSS. This corresponds to a level of definition of system structure that was typically not reached until preliminary design in previous FDD systems. Thus, the development of an increment of the GSS library typically starts with the detailed design of classes, rather than system-level preliminary design.

The GSS project has always had developer involvement in the domain analysis process. This process may be further improved by increasing this involvement, perhaps even evolving towards a joint

analysis/implementation team. In particular, as more classes are implemented, the developers have a greater stake in making sure that new analysis work will not have negative effects on the existing class library. This issue is becoming particularly important as we begin to field GSS-based applications, while at the same time expanding the domain of applicability of GSS.

Ed Seidewitz has worked at Goddard for over ten years, both in spacecraft attitude analysis, system specification and software engineering. His current projects focus on software reuse, generalized systems and system engineering. He has been involved with Ada and object-oriented methods since 1985 and is a recognized expert in the field of object-oriented analysis and design. Mr. Seidewitz has two BS degrees from the Massachusetts Institute of Technology, one in Aeronautics and Astronautics and one in Computer Science and Engineering.