Position Paper for Multi-Dimensional Separation of Concerns in Object-Oriented Systems Workshop --- OOPSLA 99

# Motivation for Enabling Separation of Concerns in Software Product Lines

Michael Grier
Raytheon Systems Company
Aurora, Colorado

Software product line approaches are becoming common for complex systems through technical enablers such as object-oriented (OO) technologies and component-based development.  An example of this application is the Control Channel Toolkit (CCT), a component architecture for satellite command and control (C&C) systems being developed by Raytheon Systems Company for government and commercial use.  This position paper is based on the experience of developing and using the CCT, and how the underlying object technologies have constrained its reuse and evolvability.

The CCT is designed to meet the requirements of a broad range of missions and to adapt to most operations concepts.  Among its architecture requirements is the need to be based on open standards and independent of proprietary technologies.  Though OO programming techniques have been essential to meeting these goals and requirements, their use has imposed many constraints on the reuse of the CCT architecture and assets.  These constraints significantly effect the reuse and sustainability of the CCT as a product line.

OO technologies allow for significant flexibility in separating concerns, and much framework design is in fact an example of this use.  The issues of constraints as a result of their use in product lines arise in several forms.  The first arises because implementing a separation of concern in OO typically means establishing an OO model for the concern, for example, a framework for database access.  Though implementations of the framework may vary through dynamic polymorphism, the framework's model itself is rigid.  This means that the product line design is resistant to change if the modeled separation of concern changes, an event often encountered when targeting product lines to new business opportunities.

A second constraint arises in the implementation of variation points in the product line components.  Variation points represent identified places in the requirements, architecture and design that need to support modifiability by application engineers.  The goal of a variation point is to allow components to be easily tailored for specific systems without modifying the component baseline (that is, the tailoring needs to be non-invasive).  The design of variation points must balance the complexity (and hence usability) of their resultant APIs against their flexibility in meeting future requirements.  In many cases the design can be resolved to a simple function callback, others require complex frameworks.  In all cases explicit assumptions must be made regarding what data and implementation aspects must be made visible to application engineers.  As in the case of using the model to separate concerns, those assumptions become captured and fixed in the design model, or require use of more flexible data structures (for example, property lists and "anys") that can have significant performance impacts. In short, variation points are required to be composed using the underlying OO technology, which requires fixing many aspects of their design and implementation at the compilation of the component, hence rendering them less flexible to application engineers.

A third constraint arises when positioning facades to wrap technologies or products that are likely to change (databases, displays, middleware). Changes in these technologies can occur from system to system because of licensing costs, performance characteristics, or even local standards for individual end systems. Typically a lowest-common denominator approach is taken for the API to insulate the client from the implementation. This follows the wrapper pattern where the client's use of the API is determined at compile time but the implementation is selected at run time. In general, use of the API is not central to the client's role in the system, but the API needs to be local to the business logic implemented by the client. In this scenario clients are not able to make use of the explicit API of the product because the composition of application occurs at the object level, and a tractable object design is required. (That is, factoring out local use of the services would unduly burden or complicate the design.) The solution then is to position an abstract wrapper for the product. However, not using the explicit API of the product limits its use, in particular the use of proprietary features which may provide a key discriminator for selection the end system by customers. These features may also have special synergies with other technologies which are also wrapped, and hence typically difficult to compose effectively together.

Multi-dimensional separation of concerns offer an attractive approach to these constraints in product line development because they offer more alternatives as to how, when and where components get instantiated in the application engineering environment. Non object-based compositional models (for example, aspect-oriented and subject-oriented programming) may permit a product line to move the time when a feature gets bound to a component implementation from the component engineering phase to the application engineering phase, and with sufficient levels of automation to still enable low recurring costs.

In a broad sense, proper engineering support for managing concern spaces could enable a product line organization to simultaneously increase the systematic reuse of software assets (reducing cost) while offering a wider array of features to customers (increasing value).

Key areas of interest for our applications of these technologies include variation point design, enabling the use of multiple underlying technologies (for example, either message oriented middleware or remote procedure call middleware), and how the availability of techniques and tools that support greater separation of concerns affects the structure of product line organizations.