Supporting Multi-Dimensional Separation of Design Concerns

Robert France, Geri Georg, Indrakshi Ray

Abstract

The development of dependable software requires developers to address multiple, interrelated, design concerns, for example, security, fault tolerance, and safety concerns. As the complexity of software increases there is a need to develop mechanisms that allow developers to isolate realizations of design concerns during design and then compose them to obtain a comprehensive design. Such mechanisms can facilitate separation of concerns along multiple dimensions and can help manage the complexity inherent in creating and evolving highly-dependable software. In this paper we discuss some of the issues and challenges that should be addressed in developing mechanisms that support multi-dimensional separation of design concerns (MDSOC). We also outline a UML-based approach we are currently developing to support MDSOC.

1 Introduction

There is a growing awareness that the manner in which software is developed can have a significant impact on the dependability of a system (e.g., see [3]). Developers of dependable software need to address multiple concerns such as security, performance, fault-tolerance, resource distribution, and safety concerns, in addition to other functional concerns. The concerns that are of interest in this paper are those that impact software design. For example, addressing a security concern that requires invokers of operations to be authenticated requires a design that includes functionality that intercepts invocations and authenticates invokers. When a concern is addressed in a design we say that the concern is realized in the design. A realization of a concern is a description of how the concern is addressed in a design. The manner in which a concern is realized in a design can impact how other concerns are realized in order to meet software quality goals. The need to balance multiple, possibly competing design concerns against each other contributes to the inherent complexity of developing dependable systems.

In traditional design approaches, a design is expressed in terms of a structure of design units: in structured design the unit is a procedural abstraction, in object-oriented design the unit is a class or object. Each design unit localizes and encapsulates information around a set of highly-related concepts, thus easing software evolution: the information hidden in a unit can be changed without impacting other design units. In the traditional design approach, the breakdown of a system solution into system units is based on a fixed set of functional concerns. Addressing a concern that is not used to determine the design structure can result in the spreading of information pertaining to the concern across design units. We refer to these concerns as pervasive or cross-cutting concerns \(^1\). This can contribute to the difficulty of creating and evolving dependable systems (in general, complex systems) for the following reasons:

1. Changes related to these concerns (e.g., changes to security policies) can be problematic because the information is not localized or encapsulated in a design unit.

2. The distribution of information can make ensuring and determining that a pervasive concern is consistently addressed across the design units difficult. The problem is made more difficult if different parts of the design are developed by different teams of developers.

3. Understanding the interactions across pervasive concerns and determining the tradeoffs that are needed to balance concerns requires that developers maintain mental models of how the pervasive concerns are to be realized as they develop design units. The number and the variety of interactions across the realizations of concerns can lead to developers producing designs that fail to meet key goals.

\(^1\)It is important to note that the terms pervasive and cross-cutting refer to how a concern is realized in a design.
Development approaches that support the separation of concerns software engineering principle are considered to be effective at tackling the complexity of developing software \cite{9}. In this paper we argue that modeling techniques that provide mechanisms supporting multi-dimensional separation of concerns (MDSoC) can ease the cognitive burden of creating and evolving dependable software. Attempts at providing support for separation of concerns at the programming level has given rise to aspect-oriented and subject-oriented programming \cite{10, 11, 12}. In aspect-oriented programming (AOP) an aspect is an implementation or design concern that cross-cuts the primary functional decomposition of a program. Researchers have also started to develop techniques and mechanisms that provide support for multi-dimensional separation of concerns in the design phase of software development (e.g., see \cite{1, 2, 14, 17}). The subject-oriented approach of Clarke et al. \cite{1, 2} provides support for separating design concerns in terms of the requirements they realize.

In this paper we describe an aspect-oriented modeling (AOM) approach that supports MDSoC. The approach allow developers to represent realizations of concerns that cross-cut the elements of design models as design aspects. A design aspect provides a design view that contains only information about a concern realization. Note that a design aspect is not a model of a programming aspect (e.g., an aspect in AspectJ). A design aspect may or may not be implemented by an aspect in a AOP language. Henceforth, we will refer to a design aspect simply as an aspect. A design in the AOM approach consists of (1) a primary model that realizes a set of functional concerns, and (2) a set of aspects that model realizations of concerns not (fully) addressed in the primary model. In section 2 we discuss some of the issues and challenges that must be addressed when developing support for multi-dimensional separation of concerns. In section 3 we give an overview of an AOM approach we are developing. The description of the approach in this position paper is sketchy, but we provide references to other work that contain more detail.

2 Developing Support for MDSoC: Issues and Challenges

An AOM approach should provide mechanism for (1) isolating information about pervasive design concerns in aspects, (2) composing aspects, and (3) evaluating composed models and addressing problems arising from emergent behaviors. An AOM approach can be beneficial for the following reasons:

- Understanding and communicating pervasive dependability concerns is easier if the information about the concerns is localized in one place.

- High quality experiences related to realizing concerns in a design can be explicitly captured in aspects and strategies for composing (weaving) these models with other design models. This enables reuse of expert experiences within and across projects.

- Changes to the realization of a concern (e.g., changes to a security mechanism) are made in one place (the aspect) and effected by composing the changed aspect with a primary model.

- The impact of a concern on a primary design structure can be analyzed by composing the aspect with the primary model and evaluating the result. Analysis tools can be developed to support the identification of conflicts and undesirable emergent behaviors. Analysis results can be used as the basis for tradeoff decisions. Tool-supported composition and evaluation can also enable investigation of alternative ways to realize concerns.

Realizing the above benefits requires addressing the following issues and challenges. Similar issues may exist for AOP, however, we believe that the higher level of abstraction supported by modeling necessitates addressing these issues using mechanisms that are more flexible than those found at the programming level.

- What types of pervasive concerns can be effectively isolated and represented as aspects? The challenge is to develop AOM techniques that (1) produce understandable forms of aspects that each capture only the information pertaining to the realization of a concern in the design, and (2) facilitate tool-supported composition of aspects. The types of technology-specific dependability mechanisms (e.g., security protocols) that can be represented by aspects must also be determined. This allows separation of technology-specific concerns from technology-independent concerns. This type of separation of concerns is a cornerstone
of the Object Management Group’s (OMG’s) Model Driven Architecture (MDA) initiative. Not all concern realizations should be expressed in aspects. Determining what concerns should be isolated in aspects and what concerns should be embedded in the primary model is an important activity. The evolvability of a system is enhanced if the pervasive concerns that are likely to change often are isolated in concern models. Understandability is also enhanced if complex pervasive concerns are isolated in aspects. On the other hand, managing many aspects can be problematic.

- How should composition be performed? What strategies are appropriate when multiple, interacting, realizations of concerns are involved? A system may be required to satisfy a number of sometimes conflicting quality goals. Furthermore, realizations of concerns can interact in ways that can compromise achievement of goals associated with the concern (we refer to this as interference). In these situations, the system developer may have to make trade-offs that determine the degree to which quality goals are met. The challenge is to (1) develop techniques for capturing and representing experience related to resolving conflicts and minimizing interference, and (2) use the captured experience to guide model composition.

- What kind of evaluation is needed to identify undesirable emergent behaviors arising from the composition of aspects?

In the next section we outline how the AOM approach we are developing addresses some of the above issues.

3 An AOM Approach

The AOM approach that we are developing supports the (1) creation of aspects that isolate information related to how a pervasive concern is addressed in a design, (2) composition of aspects, and (3) the evaluation of composed models. In our work, a design model consists of a primary model that describes a realization of a set of functional concerns, and a set of aspects. Each aspect describes a realization of a pervasive concern.

The UML is used to represent both the primary and concerns model. A UML model of a system consists of a set of UML diagrams, where each diagram describes a view of the system, e.g., a Class Diagram describes the static structural view of a system. A UML model consists of a variety of diagrams, but the model is organized around a primary design structure that is reflected in a UML Static Structural Diagram. The primary design structure is the basis for relating concepts across different UML diagrams in a model.

From a UML modeling perspective, a pervasive concern is one whose realization requires distributing information across the primary design structure, that is, the information is spread across elements found in the UML diagrams of a design model. For example, Fig. 1 shows a UML model that consists of diagrams describing the system from three perspectives (Activity, Class Diagram, and Interaction views), and two concerns that cross-cut the elements in each view (the elements enclosed in the ovals contain information pertaining to the concerns).

3.1 Representing Aspects

We define two types of aspects in our approach: context-specific concern models and generic aspects. A context-specific aspect describes an application-specific realization of a concern. A generic aspect is a design pattern that defines a family of concern realizations. Tailoring a generic concern model to a particular application results in a context-specific aspect.

3.1.1 Context-Specific Aspects

In our AOM approach, a design consists of a primary model and a set of context-specific aspects (some of which may have been obtained from generic aspects). A context-specific aspect consists of UML diagrams that describe the realization from the perspectives supported by the UML (e.g., static structural and activity

---

perspectives). Composing the aspects with the primary model results in a merging of the diagrams to produce a more comprehensive view of the design [4, 8].

The approach to representing context-specific aspects is similar to that used in the work of Clarke et al. [2] in that the models are views that can be merged. Our approach differs, though, in that (1) we do not restrict concerns to realizations of requirements, and (2) more complex forms of composition, that take into account conflict resolution strategies, is possible through the use of weaving strategies.

### 3.1.2 Generic Aspects

We use a notation we developed for representing families of designs defined by patterns to express generic aspects. In our work, generic aspects are representations of patterns. Generic aspects are represented as structures of roles, called Role Models, where a Role Model defines a family of UML models [6, 5]. It is important to note that the notion of roles used in our work differs from the run-time object-based notion of roles used in OORAM (e.g., see [15, 16]). Our roles are played by instances of the UML metamodel classes (e.g., UML class and association constructs), not by run-time instances of application classifiers. For more information on Role Models see [13, 5].

The types of concern realizations that can be represented as generic aspects are those whose distributed forms have common structural and behavioral characteristics (i.e. can be described as patterns of structure and behavior). For example, a security concern that requires all accesses to operations to be logged can be represented as a pattern consisting of a service invoker role, an invokee role, an invoked operation role, and a logging entity role. The structural relationships between entities that play these roles and the interactions between these entities that are required to log operations can be captured by Role Models that characterize the static and dynamic models that represent solutions.

### 3.2 Weaving Aspects with a Primary Model

Weaving a (context-specific) aspect into a primary model involves merging aspect elements with primary model elements. Weaving results in a model with a set of properties that is a composition of properties defined in the aspects and the primary model.
The manner in which aspects are woven with a primary model can have an impact on the quality of the composed model. For example, in some cases the aspects have to be woven with the primary model in a particular order to ensure that desired qualities are present in the composed model. Weaving strategies are used to guide the manner in which two or more aspects are woven into a primary model. Given a quality goal, a weaving strategy utilizes knowledge about the design concerns arising from the goals to determine (1) the aspects to be composed and (2) a composition strategy (e.g., determining composition order and precedence relationships to resolve conflicts) that is likely to produce a design that meets the goal. In [7] we give examples of the types of information that can be used to define weaving strategies for security concerns, and provide an example of weaving in which different orderings give different results.

3.3 Evaluating Weaving Results

Evaluation of the model produced by weaving aspects with a primary model is carried out in order to (1) detect errors, and (2) determine whether the modeled systems meet desired goals. An analysis technique we are currently developing involves evaluating woven design models against test scenarios expressed in terms of diagrams such as UML Sequence Diagrams. The test scenarios can model desirable or undesirable behaviors.

Evaluating a woven model against test scenarios can be carried out statically or dynamically. Static analysis involves composing the (models of) test scenarios with the woven model. This allows one to use the weaving mechanisms for evaluating the woven models. If composing a test scenario representing a desirable dependability behavior with a woven model results in a model with conflicts then the woven model is faulty. If the result of composing a model of an undesirable test scenario with a woven model is a consistent model then the woven model is faulty.

Dynamic analysis is possible if the woven models are executable. Dynamic analysis involves executing woven models and determining whether the sequence of interactions modeled in test scenarios can occur during execution of the woven model. As an example, consider a situation in which realizations of security concerns have been woven with a primary model. In order to evaluate the woven model, test scenarios that model attacks that exploit vulnerabilities that the concerns address are developed. During evaluation, the woven model is executed against the test scenarios. If an attack succeeds then the weaving produced a flawed design.

4 Conclusion

Our research is aimed at providing modeling support for multi-dimensional separation of concerns. AOM techniques can (1) ease the cognitive burden of developing complex software that must fulfill multiple, sometimes competing, quality goals, (2) provide the basis for the development of rigorous tradeoff analysis techniques, and (3) ease the evolution of the software by providing support for localizing information about pervasive concerns and for composing models across the concern dimensions.

There are a number of outstanding issues that we are currently addressing, some of which are listed below:

- How can weaving strategies based on domain knowledge be expressed and incorporated into tools to support model composition?
- What support is required for tradeoff analyses?
- How can one systematically transform informal descriptions of dependability concerns to aspects? How can one validate an aspect?
- How does one support refinement or reification of design concepts in a multi-dimensional modeling framework?

References


