Composition Approaches Summary

Sherif Yacoub
Hewlett-Packard Laboratories
1501 Page Mill Rd., MS1126
Palo Alto, CA 94304, USA
+1 650 857 2765
sherif_yacoub@hp.com

Hany Ammar
Department of CSEE,
West Virginia University
Morgantown, WV26506, USA
+1 304 293 0405 x2514
ammar@csee.wvu.edu

Related Work
Several successful experiences have reported on the advantages of using patterns in designing applications. For instance, Srinivasan et al. [26] discuss their experience in using design patterns to develop highly interactive software systems for speech recognition. Garlow et al. [9] describe another experience in applying analysis, architecture, and design patterns to design and implement OO cellular communication software. These experiences, among others, do not follow a systematic method to develop applications using patterns. Systematic development using patterns utilizes a composition mechanism to glue patterns together at the design level. Generally, we categorize composition mechanisms as behavioral and structural compositions.

1.1 Behavioral Composition
Behavioral composition approaches are concerned with objects as elements that play several roles in various patterns. These approaches are also known in the OO literature as interaction-oriented or responsibility-driven design [29]. Our composition approach uses notation and composition techniques that are based on the structure of a pattern, which is a different line of research. For the sake of completeness, we summarize behavioral approaches in modeling and composing patterns and identify their limitations.

1.1.1 Object Oriented Role Analysis and Software Synthesis
Reenskaug [19,20] developed the Object Oriented Role Analysis and Software Synthesis method (OORASS, later called OOram). The method uses a role model that abstracts the traditional object model. The notion of roles focuses on the responsibilities of an object within the overall group of objects. "A role is an architectural representation of the objects occupying the corresponding positions in the object system" [20]. In role modeling, we suppress irrelevant objects, aspects, and details and generalize object identities. In role synthesis, a new role model is derived that is every role in the composition of one or more roles. The derived role combines several roles to be treated as a single role.

Role models used by Reenskaug differ from object structures defined as design patterns in the pattern community. Modeling a design pattern using roles integrates both the static and dynamic aspects of the pattern in one diagram. The Unified Modeling Language (UML) avoids such integration. Static models are used for composition and interface purposes while dynamic models are used for behavioral analysis and interaction purposes. Therefore, it is always better to keep separate models for each while maintaining their relationship and consistencies. Moreover, the model synthesis approach in OOram suffers from a traceability problem. The diagrams of
synthesized role models are overloaded with directed arrows synthesizing the roles from several role models. A tabular presentation of role models is suggested in [20], however a tabular representation is less illustrative than visual models.

1.1.2 Composing Design Patterns using Roles

Riehle [21] uses role diagrams for pattern composition. The role-based approach to document a pattern considers a pattern as a collaboration of objects. Riehle focuses mainly on developing composite patterns, which are compositions of patterns whose integration shows a synergy that makes the composition more than just the sum of its parts. Riehle introduces the concept of a role relationship matrix. A class may play multiple roles in several patterns. This approach integrates patterns based on the roles of their classes, which is mainly based on interaction between pattern participants.

The main drawbacks in modeling composite design patterns as composition of role diagrams are:

- "Choosing role diagrams as the primary means for describing patterns ignores class inheritance based patterns for which further techniques have to be developed" [21]. Our approach deals with patterns as structures of related classes and addresses inheritance issues.

- Some patterns tend to be more complex when represented by role diagrams, specifically patterns with recursive structure like Chain of Responsibility pattern [21].

- There is no consensus on a unique role diagram for a design pattern. For instance the disagreement between the representation of Chain of Responsibility and Composite patterns [20,22]. Patterns are usually documented using class diagrams or object collaboration diagrams. Pattern authors do not use role models to document patterns.

1.1.3 Contracts

Helm et. al. [10] describe another form of behavioral composition using contracts. Contracts describe a formal semantics of behavioral composition of objects. Contracts focus on collaboration of several objects. They can possibly be used as a behavioral composition technique to compose design patterns. Gluing patterns together using contracts is a topic of current research work [36].

1.1.4 Superimposition

The approach by Jan Bosch [5] uses design patterns and frameworks as architectural fragments. Each fragment is composed of roles and components that are merged with other roles to produce application designs. The approach uses a superimposition mechanism that allows a class to play several roles in architecture fragments. The approach is useful for combining patterns and frameworks. It assumes the existence of application-specific classes to which we want to assign roles (i.e. merging process). It is a behavioral composition approach of roles. Our approach is a structural composition approach that uses pattern-level views. These views allow tracing pattern participants to classes that are merged at lower design level.
1.1.5 Role/Type/Class modeling

Lauder et. al. [14] take a visual specification approach to design patterns. They utilize constraint diagrams that are developed by Kent [12] together with UML diagrams. Their work is concerned with purifying design patterns from textual details that cause ambiguity. They propose three-model representation of a pattern, "role, type, and class" representations. This approach is more concerned with visualizing individual patterns than composing patterns. Their technique seems visually complex even for simple patterns as Observer pattern [Figures in 17].

1.2 Structural Composition

Structural composition approaches build a design by gluing pattern structures that are modeled as class diagrams. Structural composition focuses more on the actual realization of the design rather than abstractions as role models. Behavioral composition techniques such as roles [20,21] leave several choices to the designer with less guidelines on how to continue to the class design phase. We advocate a structural composition approach with pattern class diagrams. In the following, we summarize composition approaches that use the structure of a pattern.

1.2.1 Software Composition using Design Components

Keller and Schauer [11,23] address the problem of software composition at the design level using design components. Their approach and ours share the same objective of creating software designs that are based on well-defined and proven design patterns packaged into tangible, customizable, and composable design components. Keller defines the evolution of a pattern in a design composition environment along four dimensions: concreteness, specificity, scope, and revision. In addition, Keller also identifies limitations in modeling a design component in UML. Patterns in UML are defined as dotted ellipses (collaborations); he proposes extensions to UML package diagrams.

The structural composition approach proposed by Keller does not define a design process to compose patterns together and does not define a notion for pattern interfaces. Keller uses a package diagram to represent a design pattern as a component. Package representation has several syntax and semantic limitations as discussed in [33]. The approach in [11], though concerned with a similar problem as ours, focuses on what needs to be placed on a pattern as a design component and little about the composition procedure.

Another major concern about this approach is that it integrates two patterns to form a part of the design, then it integrates another pattern to the design and so on. This approach loses some benefits of a design component. First, the abstract view of the design is lost; there is no visual view to reflect the high level structure of the application in terms of patterns. Second, the approach does not use interfaces, which is expected by the user of a component to glue pattern structures together. The pattern visualization tool supporting Keller's approach does not support distinct pattern views that are traceable to class diagrams. Instead, the tool supports covering a class diagram with boundaries around pattern participants. Such a technique, also used in [17], is cumbersome and does not support a high-level view of the design.
1.2.2 Component-Based Frameworks using patterns

Larsen [13] takes a structural approach to glue patterns by mapping the participants of a pattern into implementation components. A component can implement many patterns and frameworks, and a pattern or a framework can be implemented across many components. The focus of his work is designing and delivering frameworks using the concepts of patterns at the design phase and components at the implementation phase. At the design phase, Larsen uses UML interface classes to model interfaces for patterns. The pattern solution structure, as a class diagram, is thus augmented with additional classes as interfaces. These class diagrams of patterns (with additional interface classes) are used to define the framework in one class diagram model. At the delivery phase of a framework, Larsen uses UML component diagrams to represent the framework as composition of physical components that are not a one-to-one mapping to patterns. Our approach shares the same concept of defining interfaces for patterns. However, Larsen's approach is different from ours in several aspects:

- Larsen uses class diagrams to represent the composition of patterns' classes. He shows a class diagram for each pattern (with their interface classes added) and then a class diagram for the framework. Our approach uses hierarchical logical models that hide details that are not utilized directly at a given design level abstraction.

- Larsen's approach does not support hierarchical views of the design. In our approach we use a pattern as a black box and then zoom into internal details. Larsen uses class diagrams only.

- The traceability problem is not solved in Larsen's approach. Pattern participants are lost in the framework design diagrams. Our models can trace pattern participants bottom-up and top-down as discussed later.

1.2.3 Composition using Macro Components

Xavier Castellani and Stephan Y. Liao [37] propose an application development process that focuses on the reuse of object-oriented application design. Their work presents a process that allows the system designer to create generic applications and reuse them in other application designs in the same problem domain. The approach starts with an existing application then abstracts design macro-components through the abstraction of application specifics. However, the authors use a general definition of macro-components (frameworks or patterns) which allows any group of related classes to be considered a pattern. An application, in the context of their development process, is split into macro-components which are filled later on with classes.

1.2.4 The Catalysis approach

D'Souza et. al. [38,39] define a component-based approach to develop software that is heavily based on interfaces at both the design and implementation level. The approach is called Catalysis which is used to build object and component-based systems using UML and some extensions made to UML constructs. At the design phases, D'Souza advocates using frameworks as building components where a framework is "a pattern of model or code that can be applied to different problems" and object oriented frameworks are "collaborations with a default, skeletal implementation." We take a similar approach for designing software as composition of constructional
design patterns, however we distinguish patterns from frameworks. Our approach addresses a more specific problem (composition of patterns) and can be used to build such frameworks. However, D'Souza's approach addressed other issues in developing software such as composing physical components, distribution of components, business driven solutions, etc.

REFERENCES


