Contents lists available at ScienceDirect
Future Generation Computer Systems
journal homepage: www.elsevier.com/locate/fgcs

Preserving architectural pattern composition information through explicit merging operators

M.T.T. That*, S. Sadou, F. Oquendo, I. Borne
Université de Bretagne Sud, IRISA, Vannes, France

HIGHLIGHTS

- We give pattern merging operators first-class status.
- We enable the traceability and the reconstructability of architectural patterns.
- We implemented the composition-centered architectural pattern description language.
- We conducted an empirical study on existing architectural patterns.

ARTICLE INFO

Article history:
Received 1 November 2013
Received in revised form
3 June 2014
Accepted 2 September 2014
Available online xxxx

Keywords:
Architectural pattern
Pattern composition
Model driven engineering

ABSTRACT

Composable software systems have been proved to support the adaptation to new requirements thanks to their flexibility. A typical method of composable software development is to select and combine a number of patterns that address the expected quality requirements. Therefore, pattern composition has become a crucial aspect during software design. One of the shortcomings of existing work about pattern composition is the vaporization of composition information which leads to the problem of traceability and reconstructability of patterns. In this paper we propose to give first-class status to pattern merging operators to facilitate the preservation of composition information. The approach is tool-supported and an empirical study has also been conducted to highlight its effectiveness. By applying the approach on the composition of a set of formalized architectural patterns, including their variants, we have shown that composed patterns have become traceable and reconstructable.

1. Introduction

A key issue in the design of any software system is the software architecture. It consists of the fundamental organization of the system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution [1]. Software architecture gives a basis for analysis of software systems before the system has been built and thus, helps manage risk and reduces cost during the software development. A good software architecture produces not only a system that works properly (the functional requirements) but also a system that meets non-functional requirements such as maintainability, exchangeability and reusability. [2]. Patterns address this important objective of software architecture by allowing the construction of specific software architectures with well-defined properties.

Indeed, some architecture description languages (ADL) such as Wright and ACME [3,4] support the construction of architectures by using predefined architectural patterns. Thus, the construction of architectures is simplified (reuse of the whole pattern structure) and equipped with proven solutions for well-known needs. Further, in real world architectures recurring problems are complex and their solutions can be represented by patterns in complex forms that require the combination and reuse of other existing architectural patterns [5]. The combined patterns on one hand, handle the increased complexity of the architecture and on the other hand, capture the properties of participating patterns. For instance, a system might use a pipe and filter pattern to process data but writes the result to a shared database. Thus, this system uses a pattern which is the combination of pipe and filter and shared repository patterns. In the literature, current support for pattern composition consists in fact of using merging operators that are not part of the pattern language [6–9]. Thus, once the architectural solution is achieved there is no means to know that it is a result of a composition of patterns. This is caused by what we call the vaporization of composition information. This prevents the traceability

* Corresponding author. Tel.: +33 0608214462.
E-mail addresses: minh-tu.ton-that@irisa.fr, minhtutonthat@gmail.com (M.T.T. That), Salah.Sadou@irisa.fr (S. Sadou), Flavio.Oquendo@irisa.fr (F. Oquendo), Isabelle.Borne@irisa.fr (I. Borne).

http://dx.doi.org/10.1016/j.future.2014.09.002
0167-739X/© 2014 Elsevier B.V. All rights reserved.

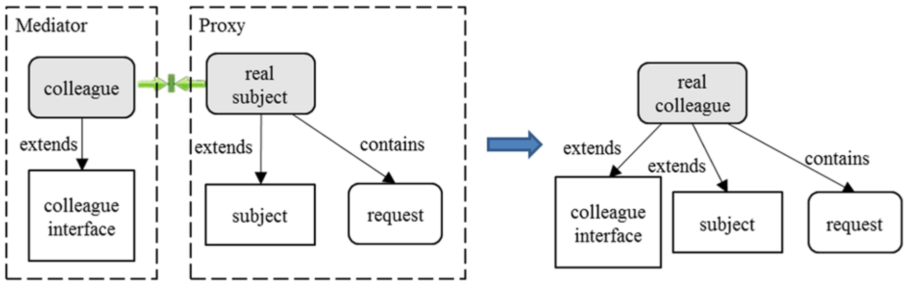

Fig. 1. Overlapping composition of Mediator pattern and Proxy pattern.

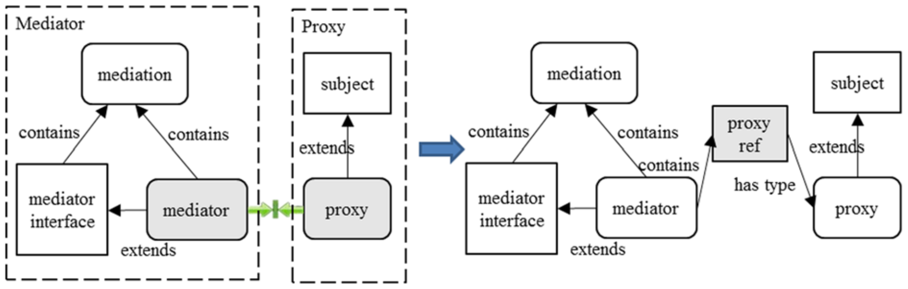

Fig. 2. Conjunction composition of Mediator pattern and Proxy pattern.

as well as the reconstructability of patterns which are essential for software evolution.

For addressing these open issues, we propose to reserve first-class citizenship for pattern merging operators. Throughout this paper, we show that being able to store and manipulate merging operators is crucial in the context of pattern construction. The idea is implemented in an architectural pattern description language, called COMLAN (Composition-Centered Architectural Pattern Description Language). The language provides a proper description of pattern that supports composition operations and a two-step pattern design process that helps to preserve pattern composition information. The idea is applicable in both architectural patterns and design patterns. However, in this work we focus our approach as well as its evaluation on architectural patterns. The remainder of this paper is organized as follows: Section 2 presents the state-of-the-art for pattern composition and pattern description. Section 3 points out the open issues through examples. Section 4 introduces the general approach to address the identified problems and goes into details of the pattern description language. Section 5 describes the pattern refinement step. Section 6 gives implementation information. Section 7 describes the validation of the proposed approach using empirical studies. Finally, Section 8 concludes the paper.

2. State of the art

As described in the previous section, the research problem we deal with is the vaporization of pattern composition information. This problem directly concerns two research areas: pattern composition and pattern description in ADLs. In the following we will elaborate the state of the art of these domains.

2.1. Pattern composition

There are mainly two branches of work on the composition of patterns. The first including [6–8] proposes to combine patterns at the pattern level which means that patterns are composed before being initialized in the architectural model. These approaches support two types of pattern element composition. The first type consists of creating a totally new element which is the product of the unification of participating elements. Regardless of the different terminologies used in [6] (conservative composition), in [8] (unification) or in [7] (overlapping), the same idea is that the combined element will have all the characteristics of participating elements, and these will no more be present in the combined structure. An example taken from [8] is the composition of the Mediator pattern and the Proxy pattern as shown in Fig. 1. The composition takes place between the Colleague class of the Mediator pattern and the Real subject class of the Proxy pattern. In its original pattern, the Colleague class extends the Colleague Interface. Similarly, the Real subject class extends the Subject class and contains the Request method. In the combined pattern, the Real Colleague class, which is the product of the composition of Colleague class and Real subject class, inherits all the features of its constituent classes. More specifically, it is a Real subject that can communicate with other Colleagues of a Mediator structure.

The second type implies that the participating elements in the composition keep their own identity, no new structure is formed because of the composition. Instead, a link element is added to connect participating elements. This composition is called combinative composition in [6] or conjunction in [8]. The example of the composition of the Mediator pattern and the Proxy pattern is retaken to illustrate this type of composition. As shown in Fig. 2, the composition takes place between the Mediator class of the Mediator pattern and the Proxy class of the Proxy pattern. The result of this composition is an added Proxy Reference which connects the Mediator and the Proxy.

On the contrary to the approaches above, in [9], Deiters et al. propose to compose pattern at instance level. An architecture entity can at the same time play roles from different architectural building blocks which in fact represent architectural patterns. As a result, the affection of different architectural building blocks to an architecture entity is not only an instantiation but also a composition.

In another work [10], Jing et al. propose a UML profile to attach pattern-related information on merged elements in composed patterns. Fig. 3 is an example taken from [10]. The Business Delegate pattern is composed with the Adapter pattern by overlapping the