Framework Description Using Concern-Specific Design Patterns Composition

António Rito Silva, Francisco Assis Rosa and Teresa Gonçalves
INESC/Technical University of Lisbon

This paper proposes an approach to object-oriented framework description. Frameworks are described using concern-specific design patterns and their composition. The frameworks understandability is achieved by the separate description of concern variations and by the description of inter-concern combinations of those variations. Both descriptions are independent of implementation details. The technical innovations are based on using design patterns for describing concern variations and on using design patterns compositions for describing variations compositions.

General Terms: Object-Oriented Programming

Additional Key Words and Phrases: Object-Oriented Frameworks, Object-Oriented Design Patterns, Object-Oriented Composition

1. INTRODUCTION

This paper proposes the use of concern-specific design patterns composition to describe object-oriented frameworks [Johnson and Foote 1988; Fayad and Schmidt 1997]. Object-oriented frameworks are designed, after a domain analysis, by experts in a particular domain. As result of this activity, a large set of classes and interactions is designed. Framework users may be non-experts in a particular domain. However, they have to be able to understand the framework so they can use it. A framework is usually described by its code; its classes are written in a particular programming language and are biased by implementation details. This makes framework understanding even harder.

The principal contribution is the capture of domain knowledge in the framework description by describing concerns and their composition. Moreover, it relates domain knowledge to solution space by associating concerns with design patterns and concerns composition with design patterns composition.

Our approach extends previous proposals for describing frameworks independently of implementation details [Johnson 1992] and based on design patterns [Gamma et al. 1995; Schmidt et al. 1996]. In particular it extends Riehle's proposal of

Address: Rua Alves Redol n°9, 1000 Lisboa, PORTUGAL

composite design pattern [Riehle 1997] with concern issues as concern variations.

The DASCo (Development of Distributed Applications with Separation of Concerns) [DASCo] approach identifies in the solution space a set of concerns. A constructive stepwise development process is defined by establishing solutions for each concern and composing concerns solutions. Concern-specific design solutions have been described as design patterns and their implementation is a set of reusable and extensible components. Our design patterns decouple concern-specific issues from object functionality and abstract concern variations. For instance: the *distributed proxy* design pattern [Silva et al. 1998] abstracts argument passing variations as passing by reference and passing by copy; the *object synchronization* design pattern [Silva 1997] abstracts object synchronization variations as readers/writers and producer/consumer; the *object recovery* design pattern [Silva et al. 1997] abstracts object recovery variations as deferred-update and update-in-place; the *object concurrency* design pattern abstracts variations as single-threaded object and multi-threaded object; the *naming design* pattern [Silva et al. 1998] abstracts naming variations as universal and local names; the *functionality configuration* design pattern [Rosa and Silva 1998] abstracts variations as flat components and hierarchical components; the *partitioning configuration* design pattern [Rosa and Silva 1998] abstracts variations as static and dynamic application partitioning configuration; the *object replication* design pattern [Goncalves and Silva 1997] abstracts object replication variations as primary-copy and read-one-write-all.

To cope with the framework understandability problem DASCo separates framework description from framework code. Framework description is made of concern-specific descriptions and concern composition descriptions. The former describes the framework components while the latter describes their possible combinations. In terms of DASCo, these descriptions present the required characteristics of concern components and composed components to ensure some of the framework properties.

2. CONCERN-SPECIFIC DESCRIPTION

For each distribution concern a DASCo framework component is constructed. Concern components are responsible for offering a solution for a minimal problem (concern). The solution provided by a concern component is orthogonal to those provided by other concern components.

Concern-specific components are described as design patterns using a modification of the format in [Gamma et al. 1995]. This modification addresses the component's expressive power which abstracts several variations of concern solution and provides framework extensibility.

The following items are modified or added to the design pattern format [Gamma et al. 1995]:

—**Concern Analysis.** In this new item different solutions for the concern are analyzed and synthesized. The analysis addresses existing systems and research results. The synthesis should emphasize the existing variations of the concern solution.

—**Forces**. This item should include the **extensibility** force. This force requires the solution to abstract a large number of the concern variations described in the

Concern Analysis item.

—**Structure and Participants.** The participants should describe the abstractions necessary to resolve the extensibility force. Moreover, the description should be independent of implementation details. The role concept [Reenskaug et al. 1996] can be used instead of class and object concepts to avoid implementation bias. For instance, decisions in terms of using inheritance or delegation, if not relevant to achieve expressive power, should be avoided and delayed until framework implementation.

—**Implementation.** This item describes how the abstractions defined in the structure item are customized to support the concern variations.

Two concern components are considered as examples: functionality configuration and partitioning configuration. A short description is presented below. For a complete one see [Rosa and Silva 1998].

—**Functionality Configuration** separates the configuration of a component-based application functionality from the functionality implementation of its components. It defines components, which encapsulate the functionality, component interfaces, which constitute the points through which components communicate, and component configuration, which describes component creation and deletion and well as components connection and disconnection, using component interfaces. Two variations, relevant for the example, where identified by concern analysis:*static functionality configuration* and *dynamic functionality configuration*. The latter allows change of connections between components at run-time, while the former restricts it to component creation-time.

—**Partitioning Configuration** presents an abstraction to the configuration of applications partitioned into nodes. It decouples the partitioning configuration from the functionality of the elements belonging to the application. In order to cooperate nodes should be configured. Node configuration can be static or dynamic. *Static partitioning configuration* corresponds to the deployment of functional entities on nodes at node creation time. *Dynamic partitioning configuration* should allow the migration of these functional entities within nodes during execution.

3. CONCERN COMPOSITION DESCRIPTION

Concern components are combined to provide solution to multiple concerns in an application. Concern composition results in the combination of the concerns variations. However, as not all combinations of concern variations are possible, concern composition results in a synergy where the composition is more than the sum of the parts.

Concern composition is described extending concern-specific descriptions:

—**Structure and Participants.** Concern participants should be synthesized following the role diagram composition approach described in [Riehle 1997]. New roles can be added to support the composition add-on.

—**Implementation.** The allowable variations combinations are described by a table defining consistent combinations. When the composition generates a synergy, the table constrains the possible composition variations. It is also necessary to

Partitioning Variations \ Functionality Variations	Static Configuration	Dynamic Configuration
Static Configuration	compatible	compatible
Dynamic Configuration	incompatible	compatible

Fig. 1. Concern Components Variations Compatibility

describe how the composition participants are customized to support the variations by detailing the participants role diagram.

A short description of functionality and partitioning configuration composition is used to illustrate concern composition. Variation composition is emphasized, and so, concern participants synthesis and composition participants customization is not described. A complete description can be found in [Rosa and Silva 1998].

Figure 1 shows a table with consistent combinations of concern variations. Dynamic partitioning configuration requires dynamic functionality configuration, because component migration between nodes implies generation and destruction of components and therefore change of connections between components at run-time. However, it is possible to have a dynamic functionality configuration with static partitioning configuration because change of connections between components at run-time does not require creation and destruction of components in nodes during run-time.

Framework understandability is improved because framework users can verify, looking at the table in Figure 1, which are the possible combinations of concern variations. Moreover, from the implementation item users learn how each of the composition variations are implemented by the composition participants. For instance, consider a shared agenda application which has two kinds of components: `Agenda Manager`, containing users agenda data, and `Agenda Session`, providing users with the capability to consult and manipulate their agendas' data. Functionality configuration is static because when an `Agenda Session` component is created it connects to the `Agenda Manager` component. Components execute in different nodes and the `Agenda Manager` component can migrate between nodes while `Agenda Session` components are executing. So, dynamic partitioning configuration is required and according to the table, functionality configuration should also be dynamic. Note that, if `Agenda Manager` migration was not required, components would not need to allow connections and disconnections during run-time, which would permit an optimized implementation of component connections and, as a consequence, better communication performance.

4. FINAL REMARKS

The DASCo approach for framework description follows the previous work on framework description [Johnson 1992; Riehle 1997], enriching it with the separation of concerns perspective. The DASCo framework is based on component composition for concern composition and components specialization for concern extensibility.

The understandability problem is dealt with by separating descriptions which emphasize concerns variations and their combination.

This article presents a solution for the framework understandability problem. However, descriptions closer to implementation may be necessary to fully understand the framework. They describe how the components are implemented emphasizing language composition techniques, e.g. inheritance and delegation, and language specific constructs, e.g. C++ template classes. In [Silva 1998] the DASCo framework implementation is addressed.

DASCo framework is publicly available from the DASCo home page [DASCo].

REFERENCES

DASCo. Development of Distributed Applications with Separation of Concerns Project. DASCo Home Page URL: http://albertina.inesc.pt/~ars/dasco.

Fayad, M. and Schmidt, D. 1997. Theme issue on object-oriented application frameworks. *Communications of the ACM 40*, 10 (October).

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley.

Goncalves, T. and Silva, A. R. 1997. Passive Replicator: A Design Pattern for Object Replication. In *The* 2^{nd} *European Conference on Pattern Languages of Programming, EuroPLoP '97* (Kloster Irsee, Germany. Siemens Technical Report 120/SW1/FB, 1997, July 1997), pp. 165–178.

Johnson, R. E. 1992. Documenting Frameworks using Patterns. In *OOPSLA '92, Proceedings* (Vancouver, Canada, October 1992), pp. 63–76.

Johnson, R. E. and Foote, B. 1988. Designing Reusable Classes. *Journal of Object-Oriented Programming 1*, 2 (July), 22–25.

Reenskaug, T., Wold, P., and Lehne, O. 1996. *Working With Objects: The OOram Software Engineering Method*. Manning Publications Co.

Riehle, D. 1997. Composite Design Patterns. In *Conference on Object-Oriented Programming Systems, Languages and Applications, Proceedings* (Atlanta, Georgia, USA, October 1997), pp. 218–228.

Rosa, F. A. and Silva, A. R. 1998. Functionality and Partitioning Configuration: Design Patterns and Framework. In *Proceedings of the Fourth International Conference on Configurable Distributed Systems* (Annapolis, Maryland, USA, May 1998). IEEE.

Schmidt, D., Fayad, M., and Johnson, R. 1996. Theme issue on software patterns. *Communications of the ACM 39*, 10 (October).

Silva, A. R. 1997. A Quality Design Solution for Object Synchronization. In C. Lengauer, M. Griebl, and S. Gorlatch Eds., *Third International European Conference on Parallel Processing. Passau Germany, August 1997. Proceedings*, pp. 576–580. Springer-Verlag, LNCS 1300.

Silva, A. R. 1998. Development and Extension of Frameworks. In S. Zamir Ed., *Handbook of Object Technology*, Chapter 32. CRC Press.

Silva, A. R., Pereira, J., and Marques, J. A. 1997. Object Recovery Pattern. In R. Martin, D. Riehle, and F. Buschman Eds., *Pattern Languages of Program Design 3*, Chapter 15, pp. 261–276. Addison-Wesley.

Silva, A. R., Rosa, F. A., Goncalves, T., and Antunes, M. 1998. Distributed Proxy: Design Pattern and Framework. Technical Report RT/001/98-CDIL (February), INESC.

Silva, A. R., Sousa, P., and Antunes, M. 1998. Naming: Design Pattern and Framework. In *Proceedings of the 22nd Annual International Computer Software and Applications Conference* (Vienna, Austria, August 1998). IEEE.