Developing Object-Oriented Frameworks Using Domain Models

MEHMET AKSIT

University of Twente, Department of Computer Science, P.O. Box 217, 7500 AE, Enschede, The Netherlands <aksit@cs.utwente.nl>

FRANCESCO MARCELLONI

Università di Pisa, Dipartimento di Ingegneria della Informazione, Via Diotisalvi, 2 – 56156 Pisa, Italy <france@iet.unipi.it>

BEDIR TEKINERDOGAN

University of Twente, Department of Computer Science, P.O. Box 217, 7500 AE, Enschede, The Netherlands <bedir@cs.utwente.nl>

Abstract

In this paper we present an integrated approach to model the domain knowledge related to a framework and to map the identified domain models into object-oriented concepts. We applied this approach to three pilot projects. We discuss the problems we encountered in mapping domain models into object-oriented frameworks. Our experience indicates that deriving a framework from the related domain knowledge reduces the amount of framework refinement time considerably.

Introduction

Although a large number of successful frameworks have been developed during the last several years, designing a high-quality framework is still a difficult task [Schmidt and Fayad 1997]. Existing framework development practices span a considerable amount of refinement time, and it is worthwhile to shorten this time. We consider modeling domain knowledge as an essential step to achieve this objective.

In this paper, we present an integrated approach to model the domain knowledge related to a framework and to map the identified domain models into object-oriented concepts. We evaluate this approach based on our experience in developing three frameworks: an atomic transaction framework for a distributed car dealer management system [Tekinerdogan 1994], an image processing framework for the analysis of the human heart [Vuijst 1994], and a fuzzy logic reasoning framework for supporting the formalization of the object-oriented development process [Broekhuizen 1996]. We conclude the paper with a discussion about the encountered problems in mapping domain models into object-oriented concepts.

The Approach

We first model the top-level structure of frameworks using the so-called *knowledge graphs* [Bakker 1987]. The vertices and edges of a knowledge graph correspond to domain concepts and relations, respectively. Finding the top-level knowledge graph of a framework requires searching the related literature and identifying the commonalities among various publications. For example, the atomic transaction knowledge graph shown in Figure 1 resulted by analyzing and comparing a considerable number of textbooks and articles written on atomic transactions (e.g. [Bernstein

1987]). Here, the node *TransactionManager* provides mechanisms for starting and terminating the transaction. The node *PolicyManager* determines the strategy for optimizing the transaction behavior. The node *DataManager* controls the access to the *DataObject*, and includes the nodes *Scheduler* and *RecoveryManager*. The node *Scheduler* orders the incoming messages to achieve serializability. *Scheduler* may include deadlock avoidance and/or detection mechanisms. The node *RecoveryManager* keeps track of changes to the data object to recover from failures.

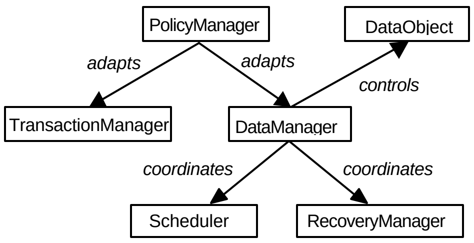

Figure 1. The top-level knowledge graph of an atomic transaction system.

Second, we refine each node within a top-level knowledge graph into an acyclic sub-knowledge graph called *knowledge domain*. The nodes in a knowledge domain correspond to a particular specialization in the domain and the relations typically represent generalization-specialization relations. For example, the node *Scheduler* corresponds to the knowledge domain shown in Figure 2. The node *Scheduler* represents the common characteristics of all schedulers. Specializations of *Scheduler* define various mechanisms to preserve consistent access to the data object.

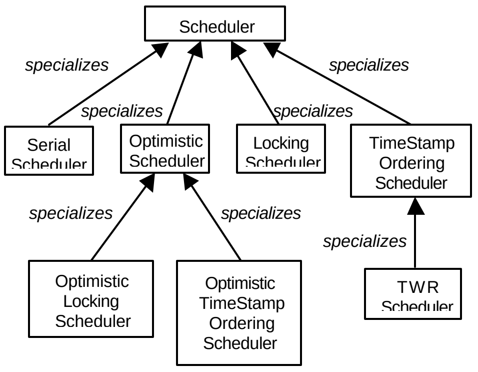

Figure 2. The knowledge domain corresponding to Scheduler node.

Third, we identify which nodes in a knowledge domain can be included together in the top-level knowledge graph. This is needed because specializations from different domains may enforce constraints on each other. For example, some specializations of *Scheduler* and *RecoveryManager* may exclude each other [Bernstein 1987]. The set of semantically correct alternatives defines here the *adaptability space*.
Fourth, to verify whether the knowledge domains model the relevant knowledge, we match them against the *use cases* identified from the user requirements. If necessary, the domain analysis process is iterated to tailor domain models to user requirements.
Finally, we map knowledge domains into object-oriented concepts. We select the nodes of each knowledge domain, which are considered relevant by the user. We define a path from the selected nodes of a knowledge domain to the most abstract node in the domain. For example, if the user requires *Optimistic Locking Scheduler*, the nodes *Optimistic Scheduler* and *Scheduler* must be selected as well. Typically, a framework user requires a set of alternative nodes, which correspond to multiple paths connected to the same node. These paths are used to identify the *hot spots* in the framework [Pree 1994]. Note that the different possible alternatives of a hot spot must meet the adaptability space constraints. We try to realize a one-to-one mapping of the selected nodes and the corresponding relations into the object-oriented concepts. Hot spots are generally implemented using hook methods and hook classes [Fayad and Schmidt 1997]. We refer to design pattern catalogs when appropriate [Schmidt et al. 1996].

Evaluation of Our Approach

We extensively tested these frameworks from the perspective of robustness and adaptability. In addition, we asked students to apply and, if possible, extend the frameworks. For example, in [Visser 1994], students modified the knowledge domain shown in Figure 2 by a hierarchical locking scheduler which had not been considered before. The architecture of the framework, however, was not affected by this change. We concluded that knowledge graphs provide stable foundations for frameworks because knowledge graphs are derived from well-established concepts characterizing the domains. Further, knowledge graphs help us identify the adaptable part of the frameworks. The overall development time of the frameworks was considerable less than the ones presented in the literature. In [Roberts and Johnson 1996], for example, the overall time is much longer because a framework is defined when a sufficient knowledge is gained after a series of implementations. The time spent during the refinement process is longer than 50 percent of the overall development time. In our approach the domain analysis and the framework refinement time took 35 and 25 percent of the overall development time, respectively.

The success of our approach depends on two factors. First, the domain knowledge has to be characterized by well-established concepts. Second, it must be possible to map knowledge graphs into object-oriented modeling concepts directly. Otherwise, software engineers may be forced to represent some elements of knowledge graphs in the operations of objects instead of providing explicit representations. This reduces adaptability and reusability of frameworks.

Defining Knowledge Graphs

The transaction framework was derived from publications on transaction systems [Bernstein et al. 1987]. The image-processing framework was based on the principles of image algebra [Ritter et al. 1987]. The fuzzy-logic framework was derived from fuzzy-logic theory [Zadeh 1973]. In all these pilot projects, the related domain knowledge is based on established theories, which allowed us to derive knowledge graphs using a reasonable effort.

Mapping Knowledge Models into Objects

We experienced the following problems because not all the elements of the knowledge models could be directly mapped into object-oriented concepts.

Difficulties in expressing knowledge specializations using class inheritance: We observed that the generalization-specialization hierarchies as defined in the knowledge domains cannot always be mapped directly to the object-oriented inheritance hierarchies. Generally object-oriented inheritance semantics are defined as inheritance of methods and instance variables, and this cannot always represent inheritance of knowledge domain specifications [Aksit and Bergmans 1992][1].

In the fuzzy-logic reasoning framework, for example, the language-based specifications of linguistic variables require a grammar specification for parsing. In the generalization-specification hierarchy of the knowledge domain *Linguistic Variable*, new linguistic variables are added in specialization nodes. This corresponds to the extension of the grammar rules. It is not possible to map this grammar-based hierarchy directly to a class-inheritance hierarchy.

Nevertheless, the problem of representing a certain generalization-specialization hierarchy of a knowledge domain can be solved by defining a dedicated function in the application framework. In [Aksit et al. 1990], for example, it is shown that a dedicated *grammar inheritance* mechanism can be defined as a structural organisation of grammar rules. In this organisation, a grammar inherits rules from *super-grammars* and/or may have its own rules inherited by *sub-grammars*.

***Architectural constraints*:** Nodes from different knowledge domains may not be composed arbitrarily. This implies that whenever the composition is changed, the consistency of the new composition must be checked. The enforcement of constraints on composition is typically achieved through type-checking mechanisms: by specifying a particular type for each of the components, we can ensure that only specializations of that type will be used as components. However, this is not always sufficient; a more powerful type checking mechanism may be needed because several complex rules may determine the architectural constraints.

[1] Note that it is usually possible to implement an object-oriented application that provides correspondence to a domain knowledge hierarchy. However, this may require the creation of additional structures and interactions because a one-to-one mapping is impossible.

For example, complex architectural constraints occur in the image processing framework. In this framework, value and coordinate sets must be homogeneous, ordering of elements in sets is restricted, and algebraic operations impose type compatibility among the elements of the sets. To manage these problems, we have adopted reflective processing techniques [Aksit et al. 1993]. Consider for example class *Image* as shown by Figure 3. *Image* consists of coordinate and value sets. The messages sent to an image are intercepted by an instance of *MetaFilter*. This object converts the received messages into objects and passes them to an instance of *ConstraintChecker*. *ConstraintChecker* accesses the attributes of the message object. If the attributes have the correct values, the message object is converted back to an execution.

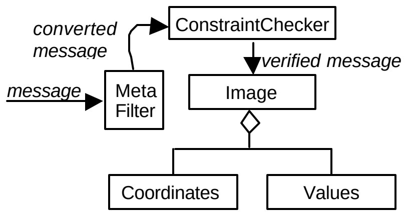

Figure 3. Enforcing constraints using reflective processing techniques.

***Dynamically Changing Implementations*:** Sometimes, the implementation of an object is not fixed but may change at execution time. Implementation improvements, for instance, may be needed for optimizing speed and space performance of objects. We experienced the need for dynamically changing implementations in all the pilot projects. The Bridge and Strategy [Gamma et al. 1995] patterns can be used for this purpose. These patterns are based on message forwarding principle; interface classes forward messages to the encapsulated implementation classes. Interface classes, therefore, must declare all the forwarded methods explicitly. This can be a tedious and error-prone task. Further, the precise set of methods and their arguments have to be fixed when an interface class is defined.

A more flexible alternative to these patterns is the delegation mechanism [Lieberman 1986]. Delegation can express dynamic implementations through delegating requests to implementation objects. The delegation mechanism, therefore, eliminates the need for declaring all the methods explicitly and supports the evolution of implementation objects. The conventional delegation mechanism, however, cannot adequately support a conditional delegation.

While building the transaction framework, for example, we found it necessary to dynamically change the implementation of the scheduler object based on certain conditions such as the state of network contention. To solve this problem, we defined our own delegation mechanism using the so-called Dispatch filter [Aksit et al. 1993]. The messages sent to a scheduler are intercepted by an instance of *DispatchFilter* which delegates the received message to the corresponding object based on the value of a condition.

Conclusion

The main claim of this paper is that the framework refinement time may be reduced considerably by modeling the related domain knowledge explicitly. We proposed a domain-knowledge based approach and applied it to developing three frameworks. We discussed the problems encountered in mapping knowledge domains into object-oriented concepts and how these problems can be solved by extending the object-oriented models.

REFERENCES

AKSIT, M., MOSTERT, R. AND HAVERKORT, B. 1990. Compiler Generation Based on Grammar Inheritance, *Memoranda Informatica* 90-07, University of Twente.

AKSIT, M. AND BERGMANS, L. 1992. Obstacles in Object-Oriented Software Development. In *Proceedings OOPSLA '92*, ACM SIGPLAN Notices, 27, *10*, 341-358.

AKSIT, M., WAKITA, K., BOSCH, J., BERGMANS, L., AND YONEZAWA, A. 1993. Abstracting Object Interactions Using Composition-Filters. In *Proceedings of ECOOP'93 Workshop Object-Based Distributed Programming*, R. Guerraoui et al., Eds., LNCS 791, Springer-Verlag, 152-184.

BAKKER, R. 1987. Knowledge Graph: Representation and Structuring of Scientific Knowledge. *PhD Thesis*, University of Twente, Deptartment of Computer Science, The Netherlands.

BERNSTEIN, P.A., HADZILACOS, V. AND GOODMAN, N. 1987. *Concurrency control and recovery in Database Systems*. Addison-Wesley.

BROEKHUIZEN, P. 1996. FLUENT: A Fuzzy Logic User Environment for an OO Fuzzy Logic Reasoning Framework. *Msc thesis*, University of Twente, Deptartment of Computer Science, The Netherlands.

FAYAD, M. E. AND SCHMIDT, D. C. 1997. Object-Oriented Application Frameworks. *Communications of the ACM*, Vol. 40, No. 10, 32-38.

GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. 1995. *Design patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.

LIEBERMAN, H. 1986. Using Prototypical Objects to Implement Shared Behavior. In Proceedings of *OOPSLA '86*. ACM Sigplan Notices, 21, 11, 214-223.

PREE, W. 1994. *Design patterns for Object-Oriented Software Development*, Addison-Wesley, Reading, Mass.

RITTER, G. X., SHRADER-FRECHETTE M. A., AND WILSON, J. N. 1987. Image Algebra: A Rigorous and Translucent Way of Expressing All Image Processing Operations, in *Proceedings of the 1987 SPIE Tech. Symp. Southeast on Optics, Elec.Opt. and Sensors*, Orlando.

ROBERTS, D. AND JOHNSON, R. 1996. Evolving Frameworks: A pattern language for Developing Object-Oriented Frameworks, at URL: http://st-www.cs.uiuc/edu/users/droberts/evolve.html.

SCHMIDT, D. C., AND FAYAD, M. E. 1997. Lessons Learned Building Reusable OO Frameworks for Distributed Software. *Communications of the ACM*, Vol. 40, No. 10, 85-87.

SCHMIDT, D. C., FAYAD, M. E., AND JOHNSON, R. 1996. Software Patterns. *Communications of the ACM*, Vol. 39, No. 10.

TEKINERDOGAN, B. 1994. The Design of an Object-Oriented Framework for Atomic Transactions, *Msc thesis*, University of Twente, Department of Computer Science, The Netherlands.

VUIJST, C. 1994. Design of an Object-Oriented Framework for Image Algebra, *Msc thesis*, University of Twente, Department of Computer Science, The Netherlands.

VISSER, B.S., EVERS, M.J. AND VAN DEN ENDE, C.W. 1994. A multi-user software development environment framework in Smalltalk, *Design Project*, University of Twente, Department of Computer Science, The Netherlands.

ZADEH, L. A. 1973. Outline of a New Approach to the Analysis of Complex Systems and Decision Processes. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-3, No.1, 28-44.

.