

Architecture and Design of Adaptive Object-Models

Joseph W. Yoder

Software Architecture Group –
Department of Computer Science
Univ. Of Illinois at Urbana-Champaign
Urbana, IL 61801

yoder@refactory.com

Federico Balaguer

Software Architecture Group –
Department of Computer Science
Univ. Of Illinois at Urbana-Champaign
Urbana, IL 61801

balaguer@cs.uiuc.edu

Ralph Johnson

Software Architecture Group –
Department of Computer Science
Univ. Of Illinois at Urbana-Champaign
Urbana, IL 61801

Johnson@cs.uiuc.edu

ABSTRACT

Many object-oriented information systems share an architectural style that emphasizes flexibility and dynamically configurable. Business rules are stored in a database instead of in code. The object model that the user cares about is part of the database, and the object model of the code is just an interpreter of the users' object model. We call these systems "Adaptive Object-Models", because the users' object model is interpreted at runtime and can be changed with immediate (but controlled) effects on the system interpreting it. This paper describes the Adaptive Object-Model architecture along with its strengths and weaknesses.

Keywords

Adaptive Object-Model, Runtime Object Model, Metadata, Reflection, Meta-Architecture, Dynamic Systems, TypeObject, Properties, Strategies, RuleObjects, Changing Business Rules.

1. INTRODUCTION

Architectures that can dynamically adapt at runtime to new user requirements are sometimes called a "reflective architecture" or a "meta-architecture". This paper focuses on a particular kind of reflective architecture that has been given many names. It was called the "Type Instance pattern" in a tutorial at OOPSLA'95 [3]. This paper calls it the "Adaptive Object-Model (AOM) architecture". Most of the systems we have seen with an Adaptive Object-Model are business systems that manage products of some sort and are extended to add new products, and we have called it "User Defined Product architecture" [5]. These systems have also been called "Active Object-Models" [1] and "Dynamic Object Models" [8]. Martin Fowler's "knowledge-level" is usually just another name for an Adaptive Object-Model [2].

An Adaptive Object-Model is a system that represents classes, attributes, and relationships as *metadata*. It is a model based on instances rather than classes. Users change the *metadata* (object model) to reflect changes in the domain. These changes modify the system's behavior. In other words, it stores its *Object-Model* in a database and interprets it. Consequently, the object model is active, when you change it, the system changes immediately.

2. ARCHITECTURAL STYLE OF AOMS

Adaptive Object-Models provide an alternative to traditional object-oriented design. Traditional object-oriented design generates classes for the different types of business entities and associates attributes and methods with them. The classes model the business, so a change in the business causes a change to the code and leads to a new version of the application. An Adaptive Object-Model does not model these business entities as classes. Rather, they are modeled by descriptions that are interpreted at run-time. Thus, whenever a business change is needed, these descriptions are changed which are then immediately reflected in the running application.

Adaptive Object-Model architectures are made up of several smaller patterns. *TypeObject* [6] separates an Entity from an EntityType. Entities have Attributes, which are implemented with *Properties* [1], and the *TypeObject* pattern is used a second time to separate Attributes from AttributeTypes. The *Strategy* pattern [4] is often used to define the behavior of an EntityType. As is common in Entity-Relationship modeling, an Adaptive Object-Model usually separates attributes from relationships. Finally, there is usually an interface for non-programmers to define new EntityTypes. Applying the *TypeObject* pattern twice for both Entities and Properties as shown in Figure 1 (*TypeSquare*) is a common theme in many Adaptive Object-Models architectures.

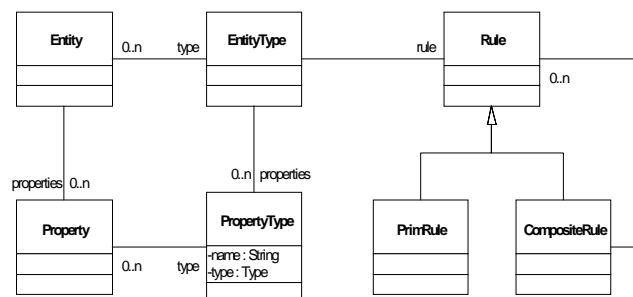


Figure 1 - Type Square

These adaptable systems are usually built from applying one or more of the above patterns in conjunction with other design patterns such as *Composite*, *Interpreter*, and *Builder* [4]. *Composite* is used for either building dynamic tree structure types or rules. For example, if your entities need to be composed in a dynamic tree like structure, the *Composite* pattern is applied. *Builders* and *Interpreters* are commonly used for building the structures from the meta-model or interpreting the results.

These are just patterns; they are not a framework for building Adaptive Object-Models. Every Adaptive Object-Model is a framework of a sort but there is currently no generic framework for building them. A generic framework for building the *TypeObjects*, *Properties*, and their respective relationships could probably be built, but these are fairly easy to define and the hard work is generally associated with rules described by the business language. This is something that is usually domain-specific and varies quite a bit.

Business rules in an object-oriented system are defined in terms of classes, attributes, behaviors, and relationships along with their respective rules defined in the business objects. AOMs provided an architecture style for defining these classes, attributes, behaviors, and relationships at runtime. These systems can evolve to be a very powerful *meta* language for defining business rules such as the form-based workflow-system described by [9].

3. AOMS AND END-USER PROGRAMMING

Applications have to be tailored many times to new requirements during its lifetime. Usually the process includes the redefinition and implementation of business rules. One way to cope with the need to change business rules is to empower domain experts with adequate tools for customizing, specializing, extending and even creating their software applications.

A domain expert can be characterized as a non-programmer person who has computational needs, has a profound knowledge of the application domain and is also able to understand the abstractions in which the system relies on. If she is provided with adequate tools, she can change, extend and tailor the application to meet the demands of local conditions.

Adaptive Object-Model architectural style is particularly adequate for building adaptable systems by means of domain-expert customization. Design patterns offer an appropriate foundation to the problem of facilitating customization and programming tools to end-users and domain-experts.

Type Cube [7] is a model for building end user programming systems. It provides guidelines for designing object-oriented applications, which can be extended and personalized by domain experts. It deals with dynamic definition of new entity types and behavior. *Type Cube* has been validated in two industrial projects.

4. CONSEQUENCES OF AOMS

Every architectural style has both advantages and disadvantages. The main advantage of the Adaptive Object-Model is ease of change. An Adaptive Object-Model is the appropriate solution if the system domain is constantly changing, or users need to dynamically configure and extend their applications. An Adaptive Object-Model can lead to a system that allows users to "program without programming". Alternatively, an Adaptive Object Model can evolve into a domain-specific language.

Turning a program into an Adaptive Object-Model usually makes it much smaller in the number of classes. Information that was encoded in the program is now encoded in the database. This new class structure doesn't change. Instead, changes to the specification lead to changes in the content of the database.

The primary disadvantage is that these systems can be hard to build and hard to understand. To understand them completely, developers must understand two object systems; the interpreter written in the object-oriented programming language and the Adaptive Object-Model that is interpreted. Classes do not

represent business abstractions because all information about the business is in the database.

Adaptive Object-Models generally need tools and support GUIs for defining the objects in your system. Adaptive Object-Models can also be slower since they are usually based upon interpreting the representation of your object model. However, our experience has shown that lack of understanding is a bigger problem than lack of speed.

5. CONCLUSIONS

Relatively large industrial applications using Adaptive Object-Models have successfully been built and deployed. They provide domain experts and analysts with tools to evolve their applications, as their business needs change. In fact, when these systems are successfully built and deployed, we have seen changes to the business model reflected in running systems an order of magnitude or two faster than before. This architectural style can be very useful in systems; specifically systems that emphasize flexibility and those that need to have their business rules dynamically configurable. This paper describes the architectural style of Adaptive Object-Models, including the process for developing them along with advantages and disadvantages in order to assist architects and developers to understand, develop, and maintain these types of systems.

6. ACKNOWLEDGMENTS

Our thanks to the many anonymous reviewers and members of the Software Architecture Group from The University of Illinois at Urbana-Champaign for their valuable comments.

7. REFERENCES

- [1] B. Foote and J. W. Yoder. "Metadata and Active Object-Models," *Collected papers from the PLoP '98 and EuroPLOP '98 Conference*, Technical Report #wucs-98-25, Dept. of Computer Science, Washington University, Sept 1998.
- [2] M. Fowler. *Analysis Patterns, Reusable Object Models*. Addison-Wesley. 1997.
- [3] E. Gamma, R. Helm, J. Vlissides, *Design Patterns Applied*, tutorial notes from OOPSLA'95.
- [4] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, MA, 1995.
- [5] R. E. Johnson and J. Oakes, *The User-Defined Product Framework*, 1998.
URL: <http://st.cs.uiuc.edu/pub/papers/frameworks/udp>
- [6] R. Johnson, B. Wolf. "Type Object". *Pattern Languages of Program Design 3*. Addison Wesley, 1998.
- [7] Reza Razavi. *Foundations of a Framework for Developing End User Programming Environments*. Position paper to ECOOP'2000 workshop on Metadata and Active Object-Model Pattern Mining. June 2000, Cannes, France.
- [8] D. Riehle, M. Tilman, R. Johnson. "Dynamic Object Model". *Proceedings of PLoP2000*. Technical Report #wucs-00-29, Dept. of Computer Science, Washington University Department of Computer Science, October 2000. URL: <http://jerry.cs.uiuc.edu/~plop/plop2k>.
- [9] M. Tilman, M. Devos. "A Reflective and Repository Based Framework". *Implementing Application Frameworks*, Wiley, 1999. On page(s) 29-64.