Adaptive Object-Models[1]

Joseph W. Yoder
University of Illinois
joeyoder@joeyoder.com

Reza Razavi
University of Paris 6 (LIP6)
razavi@acm.org

1. Abstract

The unrelenting pace of change that confronts contemporary software developers compels them to make their applications more configurable, flexible, and adaptable. A possible way to meet such requirements is to use an *Adaptive Object-Model (AOM)*. This poster describes common architectures for adaptive object-models and summarizes the results from our ECOOP 2000 workshop [9].

2. Need for Adaptable Architectures

The era where business rules are buried in code is coming to an end. Today, users themselves may seek to dynamically change their business rules. Customers require systems that more easily adapt to changing business needs, meet their unique requirements, and scale to large and small installations.

On the other hand, the same technique is adequate for the slightly different purpose of producing a whole line of software products: of course, a line of products may be obtained by variously instantiating an abstract model, but also by adapting a given initial system to various requirements that appear simultaneously instead of evolving in time.

3. The *Adaptive Object-Model* Approach

Early solutions that have been developed in order to design flexible implementation of business rules was provided by black-box frameworks [2]. A more recent approach to meet such requirements is to use an *Adaptive Object-Model* [1], where the object representation of the domain under study has itself an explicit object model (albeit partial) that is interpreted at run-time. Such an object model can be changed with immediate (but controlled) effect on the system interpreting and running it.

Objects have states and respond to events by changing state. The *Adaptive Object-Model* defines the objects, their states, the events, and the conditions under which an object changes state. If you change the object model, the system changes its behavior. For example, such a feature makes it easy to integrate a workflow mechanism, which proves useful in many systems.

Adaptive Object-Models lets to confront successfully the need for change by casting information like business rules as data rather than code. In this way, it is subject to change at runtime. Using objects to model such data and coupling an interpretation mechanism to that structure, we obtain a domain-specific language, which allows users themselves to change the system following the evolution of their business.

Metadata is then often used in adaptive object-models to describe the object model itself. When runtime descriptions of these objects are available, users can directly manipulate these objects. Since the system can interpret the metadata to build and manipulate these runtime descriptions, it is easy to add new objects to the adaptive object-model, and make them immediately available to users.

This approach has been validated by several successful industrial projects [6,7,8].

4. Observation Framework

The Observation framework [8] is an adaptive object-model architecture for dynamically describing different types of phenomenon over a given period of time; this concept is widely applied in a domain related to tests, samples, and measurements. Observations play a large role in the medical domain because they make it possible to associate specific conditions and measurements with people at a given point in time. Some typical medical observations are eye color, blood pressure, height and weight. Medical observations along with their business rules can be described and stored in a database so that any new types of observations or valid values can be made available to the system without writing new code.

5. *AOMs* and End-User Programming

One way to cope with the rapid need to change rules as a result of changes in user requirements is to empower domain experts with adequate tools to create, customize, specialize and extend their software applications.

From the end users computing perspective, a domain expert can be characterized as a non-programmer person who has computational needs and wants to make serious use of the computers. If s/he is provided with an adequate tool, s/he can change, extend and tailor its applications to meet the demands of local conditions.

Adaptive Object-Model architectural style is particularly adequate for building end-user programming systems. By providing design patterns to create dynamically adaptable systems, they offer an appropriate foundation to the problem of facilitating programming for 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 by two industrial projects.

[1] Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
OOPSLA 2000 Companion Minneapolis, Minnesota
(c) Copyright ACM 2000 1-58113-307-3/00/10...$5.00

6. Other *AOM* architectures

6.1 Domain Model Engine

The *Domain Model Engine* [4] is an Adaptive Object-Model proposed by R. Johnson and D. Manolescu as a common infrastructure for product and process models [MJ98a]. The *product model* provides a framework for defining, by composition, new domain object models at runtime. The process model is designed to support a large number of rapidly changing business rules. Its application for building a process model for workflow applications has led to micro-workflow framework.

6.2 Micro-workflow

Micro-workflow [3] is an object-oriented framework for adaptive workflow systems that tackles the workflow problem at the object-level. It's a specialized interpreter for application processes (business, administrative, scientific, etc.). It supports ad-hoc processes and dynamic process models. "Micro" states for the fact that micro-workflow involves small-scale processes that execute within applications. As a framework, its features are accessible to developers and can be specialized to cover specific requirements. The key characteristic of micro-workflow are: small kernel that provides basic features; advanced features are provided as plugins. Therefore, software developers can pick and choose the features they need.

7. Summary of ECOOP'2000 Workshop

This workshop was held in June 2000 at Cannes in France. The participants focused on comparisons of *Adaptive Object-Model*'s approach with those of Reflection and Metamodeling. Position papers for all workshop participants can be found at http://www-poleia.lip6.fr/~razavi/aom/papers/.

The participants observed that the three domains share the same dimensions of abstractions. The top most level, called L3, represents *a language for describing languages*. The next level, L2, represents a domain specific language, derived by applying L3 to the application domain. The L1 level represents the specification for a particular software (system). Finally, the L0 level represents is an instance of that specification.

Type Cube is an example of this conceptual layering from *Adaptive Object-Model* domain. Applying *Type Cube* to an application domain delivers a domain specific language for that domain. This language allows experts to specify their applications, which are then instantly executable.

The Reflection community has also used similar approaches to provide adaptable programming languages. Many reflection techniques can be found in *Adaptive Object-Models.*

The Common Warehouse Metamodel (CWM), a recently adopted standard of the Object Management Group (OMG) for metadata interchange in the data warehouse and business intelligence environments, is naturally another an example from metamodeling, that has already adopted this four level architecture. In fact, the MOF ontology, meta-layer stack, and semantics is imposed on compliant metamodels.

CWM extends the OMG's standard MOF/UML/XMI metamodeling architecture with data warehousing and business intelligence domain concepts. CWM supports a model-driven approach to metadata interchange, in which object models representing shared metadata are constructed according to the specifications of the CWM metamodel. Tools agree on fundamental domain concepts (as defined by the CWM metamodel) and, therefore, are capable of understanding a wide range of models representing particular metadata instances.

The OMG architecture generally allows for the creation of metamodels whose instances readily align with (or reveal or expose) the fundamental patterns of Adaptive Object-Models. The CWM metamodel, by directly extending MOF/UML, drives support for Adaptive Object-Model patterns into the data warehousing and business intelligence domains, leading the way for a new generation of data warehousing and business intelligence tools that are dynamically configurable and highly adaptive to changing environments. Such tools would be driven by Adaptive Object-Models, with CWM serving as the foundational metamodel guiding the creation of those Adaptive Object-Models.

8. Conclusions

Relatively large industrial applications using A*daptive Object-Models* have been built successfully. They provide domain experts and analysts with tools to make evolve their applications as their business evolves.

Also, discussions during our workshop at ECOOP'2000 has shown that *Adaptive Object-Models* share the same abstraction levels with *Reflection* and the *Object Management Group* (OMG) metamoldeing architecture.

9. References

1. Brian Foote and Joseph 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. Don Roberts and Ralph Johnson. "Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks," *Pattern Languages of Program Design 3,* Robert Martin, Dirk Riehle, and Frank Buschmann, eds., Addison-Wesley, Reading, MA., 1997.
3. Dragos A. Manolescu and Ralph E. Johnson. Dynamic Object Model and Adaptive Workflow. Metadata and Active Object-Model Pattern Mining Workshop. OOPSLA'99, Denver, USA. http://www.uiuc.edu/ph/www/manolesc/Workflow.
4. Dragos-Anton Manolescu and Ralph E. Johnson. A proposal for a common infrastructure for process and product models. In OOPSLA Mid-year Workshop on Applied Object Technology for Implementing Lifecycle Process and Product Models, Denver, Colorado, July 1998.
5. John Poole - The Common Warehouse Metamodel as a Foundation for Active Object Models in the Data Warehouse Environment. Position paper to ECOOP'2000 workshop on Metadata and Active Object-Model Pattern Mining. June 2000, Cannes, France.
6. Ralph Johnson and Jeff Oaks. "The User-Defined Product Framework," URL: http://st-www.cs.uiuc.edu/users/johnson/papers/udp/
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. Joseph W. Yoder, Federico Balaguer , Ralph Johnson --- From Analysis to Design of the Observation Pattern. Metadata and Active Object-Model Pattern Mining Workshop. OOPSLA'99, Denver, USA. URL: www.joeyoder.com/Research/metadata/OOPSLA99
9. Joseph Yoder and Reza Razavi. "Metadata and Active Object-Models Workshops" *Collected papers from the ECOOP2000.* www.joeyoder.com/Research/metadata/ECOOP2000