Adaptive Object-Model Architecture

“How to Build Systems that can Dynamically Adapt to new Business Requirements”

www.adaptiveobjectmodel.com
www.refactory.com

By: Federico Balaguer and Joseph W. Yoder

Copyright 2001 by Federico Balaguer and Joseph W. Yoder

Presenters

• Federico Balaguer
  – e-mail: balaguer@cs.uiuc.edu

• Joseph Yoder
  – e-mail: joeyoder@joeyoder.com
  – www: http://www.joeyoder.com

Copyright 2001 by Federico Balaguer and Joseph W. Yoder
Table of Contents

• General Problem
• General Solution
• Architectural Elements of AOM
• An Example in the Medical Domain
• Implementation Issues
• Advantages of AOM

Table of Contents (cont.)

• Disadvantages of AOM
• Other Alternatives
• When AOM is the best solution?
• When AOM is not the solution?
• Summary
Meta Collaborators

- Ali Arsanjani
- Krzysztof Czarnecki
- Martine Devos
- Brian Foote
- Martin Fowler
- Ralph Johnson
- Jeff Oaks
- Nicolas Revault
- Dirk Riehle
- Reza Razavi
- Michel Tilman
- Others…

General Problem

- Requirements change within applications’ domain.
- Business Rules are changing rapidly.
- Applications have to quickly adapt to new business requirements.
- Changing the application is costly, it generally includes code and data-storage.
- There are cycles of: build-compile-release.
General Solution

- Create an object design (meta-model) that describes the domain objects which includes attributes, relationships, and business rules as instances rather than classes.
- The domain objects are instantiated through a description given by the user or domain expert.
- Each new requirement is satisfied by creating a new description and a new instantiation.

Introduction of Metadata and Adaptive (Active) Object-Models

"Anything you can do, I can do Meta"

Metadata: If something is going to vary in a predictable way, store the description of the variation in a database so that it is easy to change….Ralph Johnson

"Meta is Beta"
Adaptive Object-Model

– An ADAPTIVE OBJECT-MODEL is an object model that provides “meta” information about itself so that it can be changed at runtime
  • explicit object model that it interprets at run-time
  • change the object model, system changes its behavior
– ADAPTIVE OBJECT-MODELS usually arise as domain-specific frameworks
– Business rules are stored as descriptive (meta) information in ADAPTIVE OBJECT-MODELS

Architectural Elements of AOM

• Metadata
• TypeObject
• Properties
• Type Square

• Entity-Relationship
• Strategy/RuleObjects
• Interpreters/Builders
• Editors/GUIs

Sometimes called a "reflective architecture“ or a "meta-architecture".
TypeObject: Context

- Example: The immunization program manages the provision of vaccine units for Flu, Mumps, Hepatitis-B and Polio. For each vaccine the system stores information about the recommended dose, storage requirements, illness, etc. Each vaccine unit (dose) handles due date, id, production plant. The program can have more than one vaccine for a particular disease.

- There are many instances of each particular kind of relevant element in the domain.
TypeObject: Problem

- Making a class representing each copy with the information about vaccine and vaccine unit will create structural and data duplication.
- Making classes representing each vaccine will make any system hard to maintain; each time a new vaccine is added the system should be updated with a new class and the database updated (new releases).

TypeObject: Solution

PLoPD3 - Johnson and Woolf

<table>
<thead>
<tr>
<th>VaccineUnit</th>
<th>-element</th>
<th>-type</th>
</tr>
</thead>
<tbody>
<tr>
<td>+identification[1] : Long</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>+dueDate[1] : Date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+source[1] : String</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vaccine</th>
<th>+name[1] : String</th>
</tr>
</thead>
<tbody>
<tr>
<td>+disease[1..*] : String</td>
<td></td>
</tr>
</tbody>
</table>
Properties: Context

- Example: The immunization program often requires new types of vaccinations, some of those are new for the system and have different attributes that need to be entered. This extra information can include contact information, overdose precautions and symptoms for example.

- *Instances of a given class might have different attributes and may vary at runtime.*

Properties: Problem

- Making subclasses based on their attributes makes the system static in nature. Each time a new attribute is required, a new class is created, and the system updated.
- The model will have a prolific hierarchy of classes representing the same domain abstraction or many versions that need to be released to represent the differences.
Properties: Solution

PLoP98 - Foote and Yoder

VaccineUnit
+identification[1] : Long
+dueDate[1] : Date
+source[1] : String

1
-newInstance

* -variables

Property
+name : String
+value : Object

Example: The dynamic properties added to instances of VaccineUnit have their own types. The system has to be consistent, instances linked to a given Vaccine should have the same properties.

The system needs to handle types on dynamic properties and you want to ensure that the types of properties are correct.
TypeSquare: Problem

- Fixing the property types information in each Vaccine forces the system to be changed each time a property is added or changed on its type.
- The system still need to be able to entirely define new high level Types (Vaccine).
- All Vaccine Types need to ensure only legal types for their properties are allowed.

TypeSquare: Solution
**TypeSquare: Instance Diagram**

```
<table>
<thead>
<tr>
<th>aDose: VaccineUnit</th>
<th>aFluVaccine: Vaccine</th>
</tr>
</thead>
<tbody>
<tr>
<td>identification: Long = 12341111</td>
<td>name: String = Flu Fall 2001</td>
</tr>
<tr>
<td>dueDate: Date = 12/12/01</td>
<td>disease: String = Flu</td>
</tr>
<tr>
<td>source: String = Chicago State Lab</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>aProperty: Property</th>
<th>aPropertyType: PropertyType</th>
</tr>
</thead>
<tbody>
<tr>
<td>value: Object = 10/02/01</td>
<td>name: String = dateProduced</td>
</tr>
<tr>
<td></td>
<td>primType: Object = Date</td>
</tr>
</tbody>
</table>
```

---

**Entity-Relationship: Context**

- Example: Barb is a registered practitioner for Immunizations at General Hospital since 1990. Barb is also the contact person of a kindergarten where immunization vaccine units are provided.

- *The application needs to expose the relationships between domain elements (entities). Each relationship gives roles to the participants. Roles can change and the system needs to know what current roles are being played and are available.*
Entity-Relationship: Problem

- Parties from the domain can have multiple relationships between them and the system may need to keep track of the relationships including time and other relevant business rules.
- Making instances of class Nurse and class ContactOnSite will create duplication and it will be hard to maintain the system when a Nurse becomes a ContactPerson and vice-versa.

Entity-Relationship: Solution

Accountability from Analysis Patterns – Fowler 1997

```
ImmunizationSite
+name : String
+address : String

1
* -owner
  * -role

Accountability
  * -role
  * -party

Person
```
Strategy: Context

- Some Vaccine types need special care about conditions of the environment where they are used (temp, light, air).
- The system should implement different algorithms that determine if a given Vaccine can be properly handle in a given immunization site.
- The model has to implements a defined set of interchangeable algorithms that customize the behavior of the system.
Strategy: Problem

- Making methods that implement the different algorithm in class Vaccine or VaccineUnit is would require a large case-statement and would be impractical to maintain.
- Different Vaccines can share the same algorithm.
- Vaccines for the same disease can have different algorithm depending upon context.

Strategy: Solution

Design Patterns - GOF95

![Diagram of Vaccination Pattern]

*PractitionerCertificationConstraint*  *EnvironmentConstraint*
Putting It All Together

Classes with Attributes

Behavior

RuleObject: Configurable Strategies

Copyright 2001 by Federico Balaguer and Joseph W. Yoder
Interpreters / Builders: Context

- Example: Developers successfully finish programming the classes of the system, but the system is useless. It has no definition of types of vaccine, immunization location, vaccine units…

- *Adaptive Object-Models need to implement ways for describing the types of entities, properties, and relationships as well as ways to create them from this description.*

Interpreters / Builders: Problem

- Creating methods in each class for instantiating the required metadata defeats the purpose of taking the AOM approach.
- The system has to be able to read the metadata any time, and configure itself.
- The metadata is based on the knowledge of domain experts rather than developers.
Interpreters / Builders: Solution

Example: Observations
Example: Medical Observations

Infants, Mothers and Doctors...

- **Person**
  - `name : String`
  - `address : String`
  - `phone : String`

- **Infant**
  - `gestationalAge : Number`

- **Mother**

- **Doctor**

- **LabTechnician**
Example: Medical Observations

First model from analysis

Observations: TypeObject
Observations: Properties

Observations: TypeSquare
Observations: Strategy

Medical Observations
Party and Accountability

Example

Sue Smith

John Smith

Sue is the mother of John
Party, PartyType and Accountability

Sue is the mother of John

PartyType: Metadata-Editors
Accountability: Metadata-Editors

Observation: Metadata-Editors
Observation: Metadata-Editors

Primitive Observation Type Editor

Composite Observation Type Editor
Implementation Issues

- Persisting the AOM
- Keeping Consistency
- Metamodel and GUI
- Managing Releases
- Precautions

Persisting AOM

- The metadata is expressed with objects, those objects can be mapped to relational databases as well as object-oriented databases
- There is an increasing interest on persisting metadata as XML/XMI files
Keeping consistency (versions)

- It is important keeping consistency within the metamodel while changing instances of TypeObject or other object associated with them.
- Example: changing the legal range of a Validator can make existing observations invalid.
- May have to keep version of the metadata available and apply the rules based upon the timeframe the rule applies.

Metamodel and GUI

- The metadata can simplify building user interfaces. Special GUI components can be developed for using the metadata.
- Example: The Observation model includes widgets that display list of values from the DiscreteValidators and also EntryBoxes that use RangeValidator.
- A Mediator and Adaptor layer was developed for managing the interactions between the domain objects and the GUIs.
Metamodel and GUI

OBSERVATION
DEMO
Managing releases

- The system has releases because of changes in the metadata not only the code.
- Changes in the metadata should be checked by running test cases. Use of testing tools is recommended.
- Versions of the metadata has to be kept.
- May have effective dates for the rules which are represented by the metadata.

Precautions

Avoid using the metadata for storing:
- Error and warning messages to the user.
- Relationships between classes of the model (example: ObservationType-Validator)
- Variables that are inherent to the design (example: RangeValidator::unit)
- Over design…
Advantages of AOM

- Systems can more easily be adapted to domain changes.
- Changes do not require recompiling the system.
- Power Users can change the business rules.
- Shorter time-to-market.
- Smaller in terms of classes so can be easier to maintain by experts.

Disadvantages of AOM

- Developing AOM is expensive.  
  (higher startup costs)
- Can be hard to understand and maintain.  
  (user-model and meta-model)
- It requires skilled human resources.
- Can have poor performance.
- It demands having infrastructure for storing, building, interpreting metadata (special support tools, editors, etc).
Other Approaches and Technologies

- Black-box Frameworks
- Code Generators
- Metamodeling Techniques
- Table-driven Systems
- Generative Techniques

Black-box frameworks

- These frameworks are instantiated by means of parameterization and object creation.
- They don’t need to have a meta-level.
- They don’t need to have interpreters and builders.

*AOMs can be Black-box frameworks but don’t have to be*
Code Generators

- It provides infrastructure for transforming descriptions of a system into code.
- Descriptions are based on provided primitive structures or elements.
- Code generators produce either executable-code or source-code.

Metamodeling techniques

- It focuses on manipulating the model and metamodel behind a system as well as supporting valid transformations between representations.
- The attention is on the meta-model, or a model for generating a model, rather than the final application that will reflect the business requirements.
Table-driven Systems

- Business rules can be parameterized and stored in a database.
- The running system can either interpret these rules from a database table or the appropriate function can be called with the differing values from the database.
- Sometimes these are implemented with triggers and stored procedures.

When AOM is the best solution?

- Need for flexibility
- High pace of business change
- Need for experimentation
- Need to empower user
Reasons for Failure of AOMs

- Inadequate bridge between business and technology level
- Communication: different universe of discourse
- Unclear operations and deployment structure
- High availability/runtime evolvability
- Takes too long to develop
- Security

Process for Developing AOMs

- Developed Iteratively and Incrementally.
- Get Customer Feedback early and often.
- Add flexibility only when and where needed.
- Provide Test Cases and Suites for both the Object-Model and the Meta-Model.
- Develop Support Tools and Editors for manipulating the metadata.

Very similar to Evolving Frameworks
by Roberts and Johnson PLoPD3
Summary

- Adaptive Object-Models can take time to develop -- but the payoff can be enormous!
- Adaptive Object-Models work based upon domain expert knowledge.
- Adaptive Object-Model architectural style exposes the elements of the domain and business rules.

Summary

- Applying well-known design principles such as TypeObject, Properties, and Strategies works well for developing AOMs.
- Documenting the design in terms of well know design principles can make AOMs easier to understand and maintain.
Where to Find More Information

- http://www.adaptiveobjectmodel.com
- http://www.joeyoder.com/papers/patterns
- http://hillside.net
- http://www.refactory.com

That’s All