AOM Roles Metadata Mapping

PATRICIA MEGUMI MATSUMOTO, Instituto Tecnológico de Aeronáutica
EDUARDO GUERRA, Instituto Tecnológico de Aeronáutica
HUGO SERENO FERREIRA, Faculdade de Engenharia, Universidade do Porto
ADEMAR AGUIAR, Faculdade de Engenharia, Universidade do Porto
FILIPE FIGUEIREDO CORREIA, Faculdade de Engenharia, Universidade do Porto
JOSEPH WILLIAM YODER, Refactory, Inc

The Adaptive Object Model (AOM) is a meta-architectural pattern that provides great flexibility for the applications by representing classes, attributes, relationships and behaviors as metadata. AOM applications have common needs, such as persistence mechanisms and GUIs, which could be provided by AOM frameworks. However, there are AOM applications that are domain-specific, which makes it hard to develop a generic AOM framework that can be used with those applications. This work presents a pattern for mapping domain-specific classes of AOM applications into a generic AOM class structure by using metadata that represent AOM roles. This generic AOM class structure can be referred by AOM frameworks in order to provide generic solutions for common needs of AOM domain-specific applications.

Categories and Subject Descriptors: **D.1.5 [Programming Techniques]**: Object-oriented Programming; **D.2.11 [Software Architectures]**: Patterns

General Terms: Adaptive Object Model

Additional Key Words and Phrases: Adaptive Object Mode roles, metadata mapping, reflection

1. INTRODUCTION

The Adaptive Object Model (AOM) consists in a meta-architectural pattern that represents classes, attributes, relationships and behaviors as metadata (Yoder et al. 2001, Yoder and Johnson 2002). This model provides great flexibility for applications, allowing relationships, attributes and behaviors to be changed in runtime by end users, thus improving time-to-market for systems whose business rules are constantly changing.

The AOM architectures are usually made up of several smaller patterns (Yoder et al. 2001, Yoder and Johnson 2002), such as TYPE OBJECT, PROPERTY LIST, TYPE SQUARE, COMPOSITE, STRATEGY, RULE OBJECT and ACCOUNTABILITY. Fig. 1 depicts the basic design of an AOM.

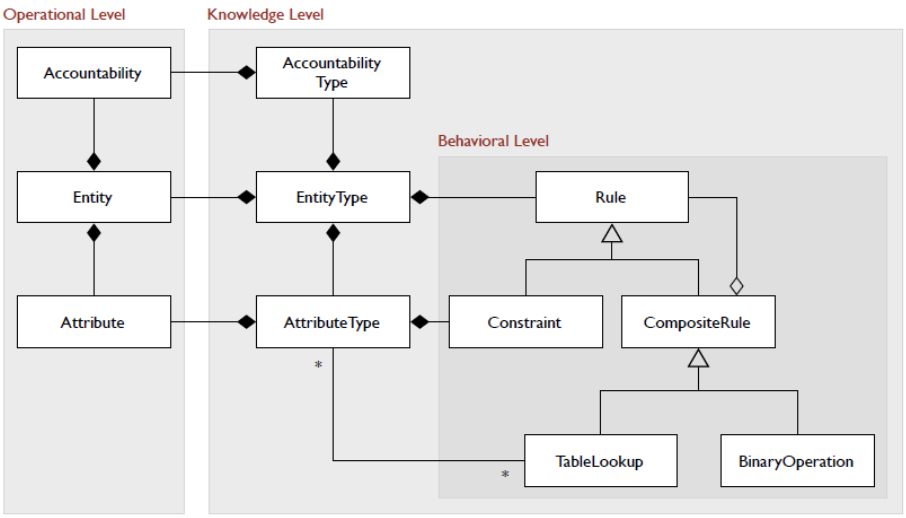

Fig. 1. AOM core architecture and design, adapted from (Yoder et al. 2001)

Besides the patterns mentioned above, there are several other patterns that are used when developing an AOM application. These AOM-related patterns form a pattern language that is currently divided in the following categories (Welicki et al. 2007b, Ferreira et al. 2010a):

- **Core:** includes the core patterns that are present in the basic implementations of AOMs.
- **Creational:** includes patterns that are used for creating runtime instances of AOMs.
- **Behavioral:** includes patterns that are used for dynamically adding, removing or modifying behavior to the AOMs.
- **GUI:** includes patterns that are used for presenting AOMs to end-users in applications.
- **Process:** includes patterns that are used in the process of creating AOMs, establishing guidelines for evolving frameworks and boundaries to avoid going up to the meta-levels far beyond than necessary.
- **Instrumental (Miscellaneous):** includes patterns that help on the instrumentation of AOMs. They also help to provide guidelines for non-functional requirements, such as performance and auditability.

The pattern presented in this paper describes how the core structure of domain-specific AOM applications can be mapped to a generic model, allowing generic AOM frameworks to be used with domain-specific applications. This pattern can be categorized in the **Instrumental** group of the AOM pattern language.

In Section 2, the AOM ROLES METADATA MAPPING pattern is presented, using a pattern format that includes the *Context*, *Problem*, *Forces*, *Solution*, *Example*, *Implementation*, *Related Patterns* and *Known Uses* sub-sections.

2. AOM ROLES METADATA MAPPING

2.1 Context

Although AOMs provide great flexibility, the complexity of the applications can increase considerably. The development of AOM applications usually takes a bottom-up approach by gradually adding adaptability to the system only where it is necessary. Even though it is possible to create generic AOM applications using patterns such as EVERYTHING IS A THING and CLOSING THE ROOF (Ferreira et al. 2010b), this bottom-up approach can lead to the development of AOM applications that are domain-specific.

Regardless of this fact, there are common requirements that usually have to be implemented when the AOM architecture is used: persistence, user interface, version control, security, end user development support tools, etc. These requirements could be implemented in generic AOM frameworks for easing the AOM development process, but to be applicable to domain-specific applications these generic frameworks would need a way to identify the roles played by the applications' classes and attributes in the AOM architecture. Without this mechanism, each domain-specific application would need to implement its specific solution for each of the mentioned requirements and these solutions would not be applicable to other applications.

In order to illustrate this issue, two systems that were modeled using AOM are considered: the Illinois Department of Public Health (IDPH) Medical Domain Framework (Yoder et al. 2001, Yoder and Johnson 2002) and a banking system for handling customer accounts (Riehle et al. 2000).

The IDPH Medical Domain Framework was developed in order to manage common information that was shared among the IDPH. This common information consists in observations made about people and relationships between people and organizations. Examples of these observations are the blood pressure, eye color, height and weight.

In order to avoid the need for development and recompilation of the system whenever a business rule changed or a new type of observation was added, the application was developed using AOM. The resulting system model is depicted in Fig. 2. The design considers situations in which one observation is composed by other observations and also considers different types of observations (range values and discrete values).

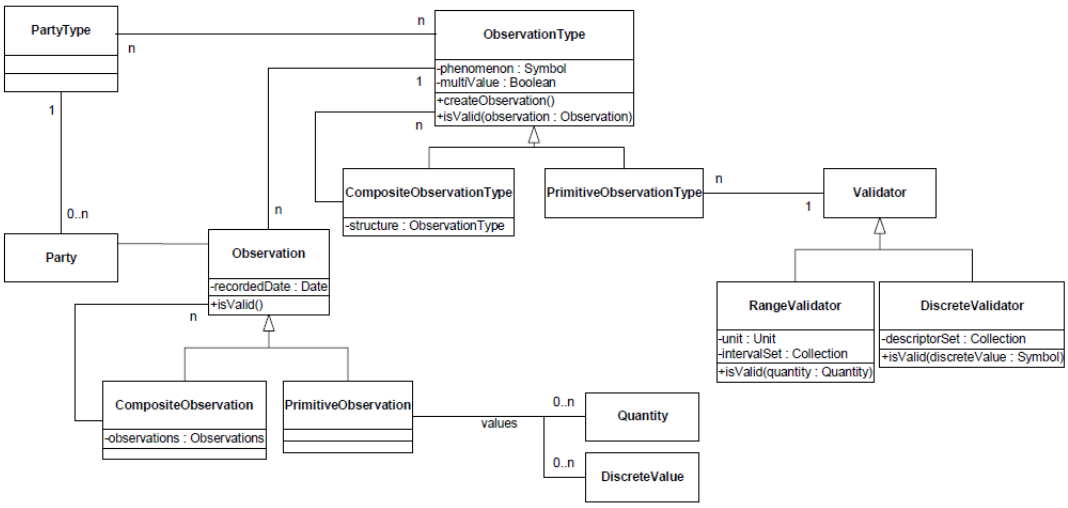

Fig. 2. IDPH Medical Framework design (Yoder et al. 2001)

The example given in (Riehle et al. 2000) consists in a banking system for handling customer accounts. The fact that the number of types of accounts in the bank can increase significantly is taken into consideration and in order to avoid a subclass and attributes explosion the TYPE SQUARE pattern is used. The basic design for the system is shown in Fig. 3.

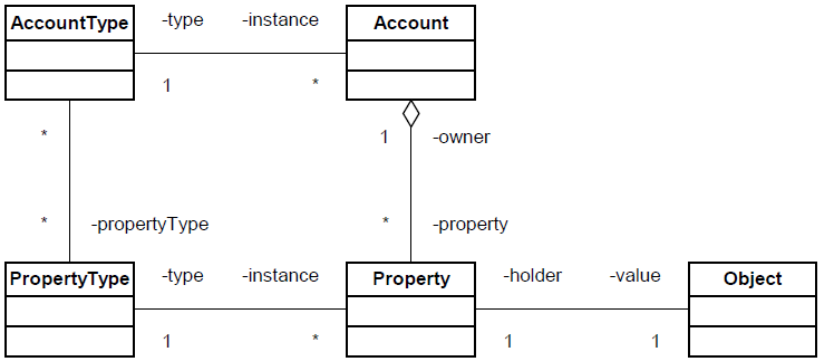

Fig. 3. Basic design for the banking system (Riehle et al. 2000)

Solutions for other kinds of concerns for the system could be described, but this simple model is sufficient for the purpose of this pattern.

There are some similarities between the structures used in the systems described, such as the usage of the TYPE SQUARE pattern. Despite these similarities, both present concerns like a persistence mechanism, a GUI, a version control for the object model and support tools for allowing end user development in the systems.

Although the systems share these common requirements, a solution developed for IDPH cannot be used for the banking system and vice versa, because each application is focused on solving the problems in their specific domains. In this context, if there were a mechanism to identify the roles played by the elements (classes and attributes) in these systems considering the AOM architecture, some common requirements implementations could be handled by an AOM framework, easing the development of these systems.

2.2 Problem

How to map a domain-specific AOM application model to a generic AOM structure so that generic frameworks can be applied to applications in different domains?

2.3 Forces

- AOM applications are usually built with a bottom-up approach by adding adaptability only where it is necessary. Therefore, although generic AOM applications can be developed, there are AOM applications that only solve specific problems in a specific to a domain
- AOMs provide flexible solutions, but their implementations are complex. If code can be reused among applications the implementation process could be simplified
- The implementation of some requirements, such as persistence, user interface, version control and support tools, may be similar among the applications using AOM, although they are usually coupled with domain-specific AOM implementations
- The implementation of some requirements does not require the knowledge of the domain of the AOM applications – there is only the need of knowing the AOM roles (e.g. Entity, Entity Type, Property, Property Type, etc) played by the classes, methods and attributes in the domain-specific application

2.4 Solution

Use metadata to identify the roles played by classes in the domain-specific AOM applications mapping them to a generic AOM class structure that encapsulates the domain-specific solution.

The use of metadata resources, such as annotations (Java), custom attributes (.NET) or even XML configuration files, allow AOM roles of domain-specific application classes to be identified in runtime. This information is used by classes that participate in a generic AOM class model that serves as an ADAPTER (Gamma 1998) for the domain-specific class model.

The AOM frameworks should refer to the generic AOM classes in their implementation instead of the application specific structure. Because these classes serve as ADAPTERS for the domain-specific AOM application classes, the solutions developed by the frameworks are automatically applied to the adapted model. This way, these frameworks are decoupled and can be reused in AOM applications in different domains.

Fig. 4 shows the solution for a simple AOM architecture. On the left side of the figure the domain-specific AOM application is depicted. The classes that play an AOM role in this application are annotated so that the generic AOM classes (depicted on the right side) can identify these roles and adapt the domain-specific classes. The AOM frameworks use the generic AOM structure for allowing their solutions to work for applications in different domains without the need to know these domains.

The generic model depicted in Fig. 4 could also include other roles, such as Accountability, Accountability Type and Rules, which will not be mentioned in this paper for simplicity (the concept for them is analogous to the Entity, Entity Type, Property and Property Type roles). The incorporation of these roles is made by creating new types of metadata.

Although Fig. 4 shows the generic model as if its structure were similar to an ordinary AOM architecture, this diagram only shows what is made public for the AOM frameworks. In the actual implementation of the pattern each generic class contains a reference to an object that corresponds to an instance of a domain-specific class that plays the same AOM role. This object is used for invoking methods through reflection in the adapter classes. Refer to the *Implementation* section for more details.

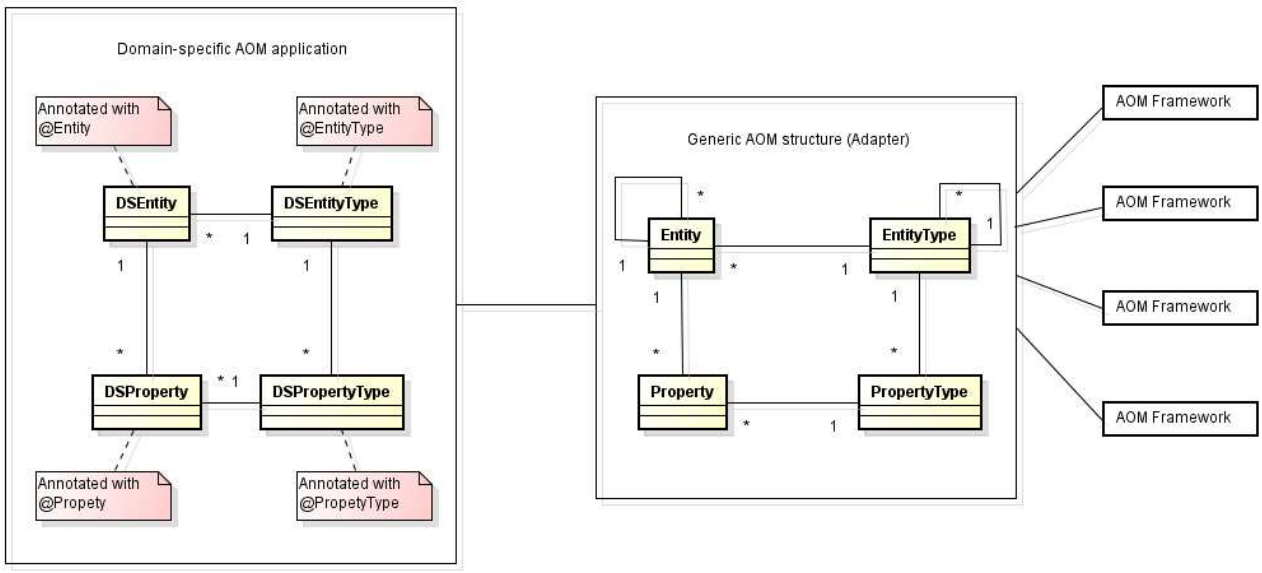

Fig. 4. Generalization of the domain-specific AOM application

Some examples of AOM role annotations are shown below (the examples given in this paper are based on annotations, but role representations are analogous for .NET custom attributes and for XML configuration files):

- @EntityType: used to indicate that a class plays an Entity Type role in the TYPE SQUARE pattern
- @Entity: used to indicate that a class plays an Entity role in the TYPE SQUARE pattern
- @PropertyType: used to indicate that a class plays a Property Type role in the TYPE SQUARE pattern
- @EntityProperty: used to indicate that a class plays a Property role in the TYPE SQUARE pattern

Other kinds of annotations or variations of the above annotations can be created in order to identify methods and fields that contain AOM role information in the generic classes.

2.5 Example

The situation described in the *Context* section considers two systems, modeled using AOM, which solve different problems in different domains. Following the solution presented above, the *PartyType* and *AccountType* classes can be annotated with @EntityType; the *Party* and *Account* classes can be annotated with @Entity; and so on. Fig. 5 shows simple implementations of the *Party* and *Account* classes with AOM roles annotations being used.

```
@Entity
public class Party {

  @EntityType
  private PartyType partyType;

  @EntityProperty
  private List<Observation> observations;

  public PartyType getPartyType() {
    return partyType;
  }

  public void setPartyType(PartyType partyType)
  {
    this.partyType = partyType;
  }
```

```
@Entity
public class Account {

  @EntityType
  private AccountType accountType;

  @EntityProperty
  private List<Property> properties;

  public AccountType getAccountType() {
    return accountType;
  }

  public void setAccountType(AccountType
                                accountType) {
    this.accountType = accountType;
  }
```

```
  public List<Observation> getObservations() {
    return observations;
  }

  public void addObservation(Observation
                             observation) {
    observations.add(observation);
  }
}
```

```
  public List<Property> getProperties() {
    return properties;
  }

  public void addProperty(Property property) {
    properties.add(property);
  }

}
```

Fig. 5. Simple implementation of the Party and Account classes that shows AOM role annotations being used

An AOM framework that handles a common requirement in AOM systems, such as persistence, can refer to the generic AOM class structure. These classes will adapt the classes in the IDPH and the banking systems according to a configuration, making the solution implemented by the AOM framework applicable to both systems, even though the framework does not know any of the domains.

Fig. 6 shows an example for the implementation of the Entity class in the AOM generic model. This class would be an ADAPTER for the classes annotated with @Entity. The implementation of the classes representing the other AOM roles would be analogous to the implementation shown below.

```
public class Entity {

  // Attribute to store the instance of the domain-specific class
  private Object dsEntity;

  // Method for getting the Entity Type
  private Method getEntityTypeMethod;

  // ... Other attributes, such as method for getting Properties

  private Entity () {}

  public static Entity createEntity (String entityClass)
  {
     // Exception handling code was ommitted

     Entity entity = new Entity();
     Class entityClazz = Class.forName(entityClass);
     entity.setDsEntity(entityClazz.newInstance());
     Field[] fields = entityClazz.getDeclaredFields();
     for (Field f : fields)
     {
        EntityType entityTypeAnnotation = f.getAnnotation(EntityType.class);

       if (entityTypeAnnotation != null)
       {
           // Identifying the method for getting the Entity Type
           String fieldName = Utils.firstLetterInUppercase(f.getName());
           String getEntityTypeMethod = "get" + fieldName;

           Method getMethod = entityClazz.getMethod(getEntityTypeMethod);
           if (getMethod != null)
                entity.setGetEntityTypeMethod(getMethod);

       }

       // ...
     }
     return entity;
  }

  // Method used by the AOM frameworks
  public EntityType getEntityType(){

    // Ommitted exception handling code
    // The getEntityType method returns an EntityType class instance that
```

```
    // corresponds to the domain-specific object returned by the
    // getEntityTypeMethod. It guarantees that there is a one-to-one
    // relationship between instances in the generic and domain-specific
    // models
    return EntityType.getEntityType(getEntityTypeMethod.invoke(dsEntity));
  }

  // ...

}
```

Fig. 6. Example of code for the Entity class in the generic structure

This solution eases the process of AOM application development, once it allows generic solutions for the AOM architecture to be adapted to the domain-specific applications.

2.6 Implementation

The implementation of this pattern demands use of reflection mechanisms to read the class metadata and invoke the identified methods. The following aspects should be taken into consideration when implementing this pattern:

- Each class in the AOM generic model should have an attribute to store the instance of its corresponding class in the domain-specific application. This instance should be used for invoking methods in the adapted classes;
- There must be a way to identify methods for getting and setting objects in the domain-specific classes so that relationships between classes in the model (e.g. an Entity class must contain a reference to an Entity Type class) can be adapted by the generic classes. This identification mechanism can be done through specific annotations or through name conventions for methods. For instance, if the Entity class contains a field named *type*, which contains the reference for its Entity Type class, the method that returns this field value can be considered as being *getType* and a method that sets this value as being *setType*;
- There should be a one-to-one relationship between an instance in the generic model and an instance in the domain-specific model. In order to control those instances, a dictionary may be needed;
- The use of annotations or custom attributes for AOM roles identification eases the development process when compared to configurations through XML files. However, these annotations must be added into the code in the AOM domain-specific applications. XML configuration files with metadata for identifying AOM roles can be useful if AOM framework solutions are to be applied to released AOM applications that had not been annotated during the development process

2.7 Consequences

(+) The roles played by classes in a domain-specific application modeled using AOM can be easily identified by automated tools.

(+) Frameworks developed based on the general AOM structure can be reused for domain-specific AOM classes configured with metadata.

(+) The identification of the roles played by classes in AOM modeled applications can serve as a documenting purpose, improving the code understandability.

(-) This pattern requires the use of reflection to identify AOM roles played by the classes in the domain-specific applications, which can reduce the system performance.

(-) The frameworks are restricted to the roles defined by the metadata supported by the general structure.

2.8 Related Patterns

This pattern uses the ADAPTER (Gamma et al. 1998) pattern when it provides the AOM generic structure for the AOM frameworks, delegating the operations to the domain-specific AOM applications.

This pattern uses the ENTITY MAPPING (Guerra et al. 2010) by using metadata to map the AOM domain-specific application classes to the generic AOM structure.

This pattern can be used to implement other patterns of the AOM pattern language, such as the PROPERTY RENDERER (Welicki et al. 2007a).

The mechanism for handling metadata can use the patterns of the pattern language for metadata-based frameworks (Guerra et al. 2009) in order to provide a flexible implementation.

2.9 Known Uses

Oghma (Correia and Ferreira 2008) is an AOM framework that uses this pattern, implementing it with .NET custom attributes.

The AOM Role Mapper (AOM Role Mapper Project) is a project under development that implements the concepts presented in this pattern in Java, using annotations.

REFERENCES

AOM Role Mapper Project. Available at: <https://sourceforge.net/projects/aomrolemapper>. Accessed in: 2011-05-21.

Correia, F. F. and Ferreira, H. S. 2008. Trends on Adaptive Object Models Research. In *Proceedings of the Doctoral Symposium on Informatics Engineering 2008*, Porto, Portugal.

Ferreira, H. S., Correia, F. F., Aguiar, A. and Faria, J. P. 2010. Adaptive Object-Models: a Research Roadmap. *International Journal on Advances in Software*, 3, 70—89.

Ferreira, H. S., Correia, F. F., Yoder, J. W. and Aguiar, A. 2010. Core Patterns of Object-Oriented Meta-Architectures. In *Proceedings of the 17th Conference on Pattern Languages of Programs (PLoP2010)*, Reno, Nevada. USA.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. 1998. Design Patterns: Elements of Reusable Object Oriented Software. *Addison-Wesley*.

Guerra, E., Souza, J. T. and Fernandes, C. 2009. A Pattern Language for Metadata-based Frameworks. In *Proceedings of the 16th Conference on Pattern Languages of Programs (PLoP2009)*, Chicago, Illinois. USA.

Guerra, E., Fernandes, C. and Silveira, F. F. 2010. Architectural Patterns for Metadata-based Frameworks Usage. In *Proceedings of the 17th Conference on Pattern Languages of Programs (PLoP2010)*, Reno, Nevada. USA.

Riehle, D., Tilman, M. and Johnson, R. 2000. Dynamic object model. In *Proceedings of the 2000 Conference on Pattern Languages of Programs (PLoP 2000)*, Washington University Department of Computer Science.

Welicki, L, Yoder, J. W. and Wirfs-Brock, R. 2007. Rendering Patterns for Adaptive Object Models. In *Proceedings of the 14th Pattern Language of Programs Conference (PLoP 2007)*, Monticello, Illinois, USA.

Welicki, L., Yoder, J. W., Wirfs-Brock, R. and Johnson, R. E. 2007. Towards a pattern language for adaptive object models. *Companion of the ACM SIGPLAN Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA 2007)*, Montreal, Canada.

Yoder, J. W., Balaguer, F. and Johnson, R. 2001. Architecture and Design of Adaptive Object-Models. In *Proceedings of the ACM SIGPLAN Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA 2001)*, Tampa, Florida, USA.

Yoder, J. W. and Johnson, R. 2002. The Adaptive Object-Model Architectural Style. *IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture: System Design, Development and Maintenance (WICSA 2002)*, Montréal, Québec, Canada.