

GOOD: Meta-modeling and Grammar-Oriented Object Design

Ali Arsanjani

Object-Oriented Technologies, Inc.
And Maharishi University of Management

Submitted to the OOPSLA Workshop on Meta-Data and Dynamic Object Models, 1998

Abstract

Meta-modeling with grammars facilitates the creation of dynamically configurable and interacting software components which embody the highly volatile business rules of today's fast-paced business systems. The dynamic aspects of pluggable business components can be modeled and realized through grammar-oriented object design. We employ the concepts of grammar-oriented object design, reuse-levels, valid object interaction sequences and "objects handling their own manners" to help create such meta-models.

Introduction

Experience with flexible, extensible and distributed application architectures has taught us the convenience of grouping classes/objects in levels of abstractions that we refer to as "reuse levels". These are both concrete (Base Class, Inheritance Hierarchy, Aggregation Hierarchy, Cluster, Framework) and abstract (Pattern, Generic Architecture, Methodology Application Knowledge, Environment Inter-action Meta-Knowledge and Technology-Transfer Knowledge.)

For each reuse-level – typically a cluster, we create a meta-model based on the roles objects take on when collaborating within the cluster to fulfill a business intent. The Valid Object Interaction Sequences (VOIS) and Typical Object Interaction Sequences (TOIS) of the cluster of objects can be defined through a context-free grammar. This grammar can be interpreted at run-time to yield a system whose behavior is dynamically configurable; its inter-object relationships can be conveniently meta-modeled for potential run-time customizations based on dynamically changing business needs and new opportunities recognized by "non-programming" users.

Meta-modeling simplifies complex object models. Objects should handle their own manners and exist at various reuse-levels. At each level, the important reuse-level object's Valid Object Interaction Sequences and Manners, needs to be captured as a grammar of a language abstracting the problem domain. This grammar can be written in a variety of representations, yielding the concept of "grammar-oriented object design" (GOOD) in which the "laws governing" the application domain are captured in a language via the grammar. Parsers can be written for these grammars to act as active object models which are not only representations of possible (run-time) versions of an object model but that of their dynamic interactions as well.

Below, I will try to specify some of the epistemological factors that stem from practical system development and try to model the way I have been modeling systems. I am deliberately concentrating on "dynamic" rather than "static" aspects such as Property List or Intelligent Variable [Foote and Yoder 98].

Units of Coherence

The Way of Objects is to partition systems that are inherently complex into units of coherence. The criteria for more coherence are the correct selection and support for an abstract state through behavior related to that state space and the transformations that are possible within itself. But these transformations, these laws of Nature governing the behavior of an object system can be traced to simpler laws of behavior and valid interaction sequences that form a domain language of their own. These smaller units of coherence, these classes, represent a set of objects have the same manners: they observe the transformations of their state through correct behavior; they accept messages which leave the intent behind creating this unit of coherence within the business domain, intact. This is the integrity of the natural law of a set of interacting objects that have been created to enforce and fulfill a set of domain rules, a “domain grammar”.

Objects have Manners

Objects can manage their own behavior through ‘manners’. These are the rules along which they accept messages from other objects in order to alter their state space towards the fulfillment of a collective domain goal or intent. Thus, the concept of a valid object interaction sequence represented as a domain grammar, appears to be a convenient conceptual design artifact. The latter represents the set of all possible ways in which the objects in this cluster *should* interact in order to *fulfill* the domain goals of their cluster. These domain goals cover a wide spectrum of granularity: from “the smallest of the small to the greatest of the great”. This ranges from: being able to calculate the price of financial assets and derivatives, to balance a ledger, to control a part of a production line within a factory, to monitor the level of chemical substances in a patient’s bloodstream, etc.

Higher Levels of Coherence

The need to partition systems of interacting objects, which are units of coherence, leads us to create higher levels of architectural coherence. These levels of abstraction start with objects and clusters (sub-systems, categories, packages) which are *not* merely namespaces for grouping together objects, but which carry an identity, manners, state and behavior of themselves. Thus, an individual, family, community, society, city, nation and world are different levels of coherence and partitioning which are much more than the sum of their parts and carry their own unique and intricate, stochastic and collective identity, state, behavior and manners (rules of behavior within the cluster or group).

Reuse-Levels

Objects and classes are reused. Clusters of objects are found to be reusable. Combining inheritance hierarchies with aggregation hierarchies with clusters can result in domain and application frameworks. There are also more abstract levels of reuse such as patterns, generic architectures, environment interaction meta-knowledge, and technology/process transfer knowledge.

The concrete levels of reuse consist of base class, inheritance hierarchy, aggregation hierarchy, cluster, framework (these are the concrete reuse levels); patterns, generic architectures, environment interaction meta-knowledge, technology/process transfer knowledge are the abstract reuse levels.

Meta-Modeling

We find it convenient to distinguish two types of meta-models: static and dynamic. A model of an object model (class diagram) is a static meta-model. A model of valid object interaction sequences (VOIS) is a dynamic meta-model. This model of the dynamics of message-passing constitutes a grammar-oriented object design: the reuse-level containing its own unit of coherence (objects, inheritance hierarchies, aggregation hierarchies, clusters, frameworks, etc.). Models of valid object interaction sequences (VOIS) can be captured in a grammar.

Grammar-oriented Object Design

Therefore, separate all the collaborations for a single method (“collaboration interface”) from the objects that actually will collaborate to fulfill the intent behind the collaboration (“collaboration implementation”). Construct a meta-model for this and represent it as a grammar. Provide a parser for the cluster owning the collaboration. The cluster will interpret the grammar at run-time based on user configuration and the input stream of messages and will dynamically bind the collaborators to the collaborations. This is the essence of grammar-oriented object design.

The following is an observation about architecting scalable and extensible designs cast in pattern format.

“Meta-Model of Design”

Intent

Provide a road-map for the way you are going to solve a design problem by modeling the solution (the model) instead of just providing the solution. (By creating and representing a model of the solution that will be more extensible, simpler and more scalable, rather than just creating the single solution itself.)

Motivation

<This section is necessary to provide insight for people new to the pattern or those seeking new insights and trade-off mechanisms. It rests on the inductive faculty of the human mind to learn by mentoring. Caveat: Below, there are some dangling references to “fledgling patterns for implementing business rules” without there being an explicit definition for them in this text. I chose to mention them, even though they are not complete and are in the process of being better defined as part of a work-in-progress on a pattern language for business rules. >

Context

The fledgling pattern, Rules as First-Class Citizens, entails that for every reuse-level (e.g., object or cluster) you have a set of rules that are the object’s ‘manners’. This provides a mechanism for showing how the object should and actually does accomplish its purpose. This is achieved not by merely accepting ad-hoc messages from any object that calls it; but “reflectively”, by accepting a set of messages which incrementally change the state of objects, culminating in the accomplishment of the business objective that the cluster(or in general, the reuse level) was designed to achieve.

The meta-model is represented by a context-free grammar defining the valid object interaction sequences for that domain.

Forces

- Rules can be implemented in various ways depending on scalability and many other factors. How are rules to be implemented?
 - Boolean Methods (another fledgling pattern) suggests implementing a rule as a simple method that returns a boolean value after checking a condition.
 - Strategy can be used to do the same job, dynamically and more flexibly.
 - Rules-as-Objects encourages us to reify rules. Meta-modeling allows us to model this modeling of rule classification as being composed of at least an Assessor and an Action. The Assessor is the passive counterpart of the Command pattern: Command actually *does* something more (Action); whereas Assessor encapsulates evaluation and assessment of conditions (simple or compound). An Assessor may delegate assessment to an Interpreter necessary.
 - Dynamic interpretation of rules
- Implementation considerations encourage us to think in terms of economy: e.g., no-one wants to have Rules-As-Objects for *all* situations:

- Implementation considerations might include deciding whether you need a Simple Rule, not a Compound Rule. Explanation: the Compound Rule is a Composite of related Assessor/Action Composites. Actions can be implemented as Strategy, Command or with the help of Template Methods when there is a family of actions.
- When implementing rules, we may not want to implement the actual Action in the action object itself.
 - We may want to outline a Template Method of a generic *sequence of actions* that should occur after evaluating an Assessor. But who should (implements the Template Method deferred methods? We can have a GUI make a tabbed pane read-only and then access something from the database and set another user-interface with a value. Not all of these three actions are performed within the confines of the Action. The Action provides an interface that the GUI will implement to fulfill an action-step request (or message-send).

Solution

Model the possible solutions in a meta-model. Reify each solution as a class relating it to the other solutions in a meta-model. Allow configurable rule representations (pay as you go) by implementing parts of the meta-model.

Implementation Considerations

<one will be mentioned here for this discussion>

Write a grammar to represent the manners of the important clusters. Make this grammar available to “users” of the system at run-time, so they will be able to customize system behavior by changing the grammar either with a visual tool, or with a script editor.

References

- [Foote and Yoder 98]
Metadata and Active Object-Models.
- [Riehle 97]
Role Object.
- [Gamma, Helm, Johnson, Vlissides 95]
Design Patterns: Elements of Reusable Object-Oriented Software.
- [Premerlani95]
Meta-modeling. {Personal correspondences}
- [Blaha, Premerlani 98]
Object-oriented Modeling and Design for Database Applications.
- [Fowler 97]
Analysis Patterns.
- [Buschmann+97]
Pattern-Oriented Software Architecture.
- [Wolf and Johnson97]
Type Object.