The CarrierWave Pattern Language

Draft 0.1

September 16, 2002

Chris K. Wensel

Introduction

This document is intended to present the major patterns in the CarrierWave architecture. Some patterns are only restated here with links or references to their original source. And a couple of them aren't necessarily new but there aren't any referenceable resources.

This document only identifies the patterns used by CarrierWave, not how CarrierWave applies them in practice.

Base Patterns

Name: Graph Plan

Intent

Provides a means to describe the structure of an object graph, or the closure of a sub-graph contained in a larger object graph while remaining independent of the object graph implementation.

Also Known As

Graph Model

Reference

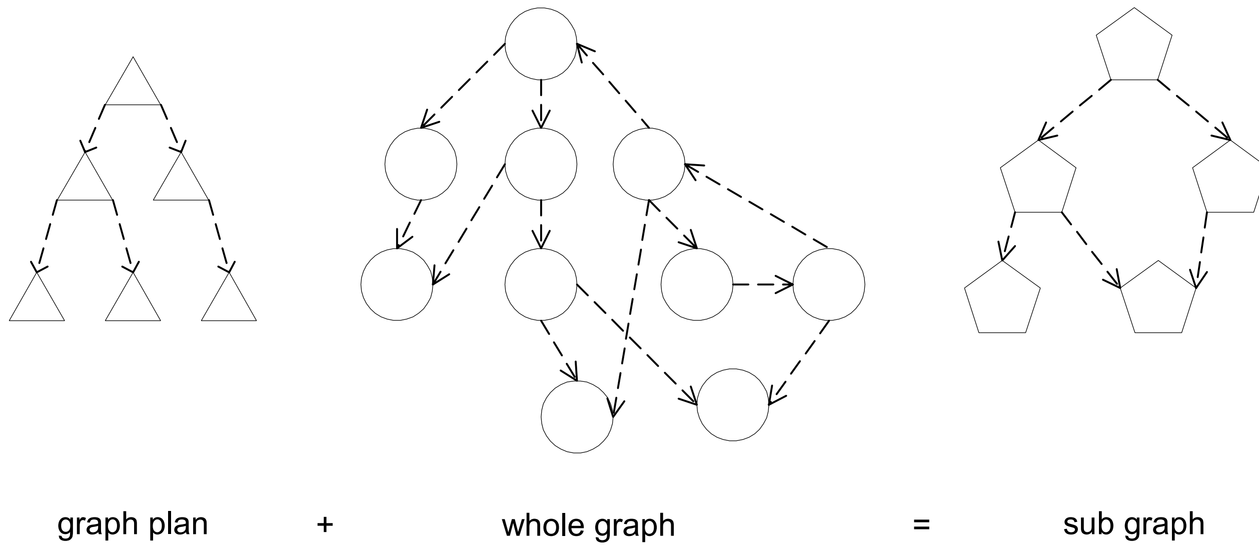
Motivation

When two or more semantically equivalent object graphs having different implementations exist, it may be necessary to have an implementation independent representation of the graphs that that they can be validated as representing the same 'type', or so that common functions can be applied to them, without coupling the functions to the graph implementations.

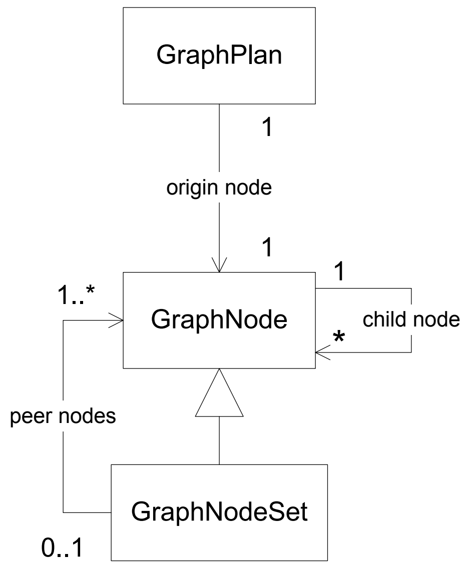
Note that semantic equivalence of two graphs says they represent the same idiom. For example, a "bill of sale" would represent a common idiom. A "bill of sale" object graph could have objects that represent the line items, buyer, supplier, etc. One implementation of this "bill of sale" would be used by internal business logic in an application server; another implementation could be used for the externalization of the "bill of sale" for persistence or integration purposes. A graph plan that recognizes both implementations could be used to validate the correctness of a given "bill of sale" or to prototype a new "blank" "bill of sale".

Applicability

Structure



Above is a conceptual diagram showing how graph plans are simply trees used to describe portions of whole graphs.



Above is a class diagram introducing the GraphNode, GraphNodeSet, and GraphPlan classes and their relationships.

Consequences

The Graph Plan pattern provides for a kind of polymorphic behavior over idioms as object graphs, or over the elements of the graphs, disregarding implementation of those elements.

Implementation

The GraphPlan is purely optional, but it would allow trees of GraphNode instances to be named (versioned), persisted, and shared across components of the architecture and between threads concurrently.

The GraphNode holds the type name of the node and the edge name referencing the node. The type name may be a "common type" name shared between all object graph implementations, thus satisfying one of the needs for providing External Polymorphism.

The GraphNodeSet allows for substitutable nodes in the graph (sub-classes), of course all the peer nodes owned by the GraphNodeSet must have a common type the GraphNodeSet represents at default.

Known Uses

Grove Plan

Related Patterns

External Polymorphism

Depends

Name: External Polymorphism

Intent

Provides a means to treat different object implementations that are type equivalent, polymorphically.

Also Known As

Reference

External Polymorphism

Motivation

Applicability

Structure

Consequences

Implementation

Known Uses

Related Patterns

Depends

Name: Instance Reference

Intent

Provides a means to encapsulate the identity of a given entity without coupling to its implementation, especially for the case where an entity may have more than one implementation.

Also Known As

Reference

Instance Reference – CORBA Design Patterns

Motivation

Applicability

Structure

Consequences

Implementation

Known Uses

CORBA Inter-ORB Reference (IOR)

Related Patterns

Depends

Name: Value Object

Intent

A special case of the Data Transfer Object pattern that requires type equivalence between a Data Transfer Object and the business object its structure and values are sourced from.

Also Known As

Reference

Motivation

This pattern supports cases where the business object model and type hierarchy must be visible to client applications. It has specific support for representing both the dependent and literal values managed by a first class business object, and the relationships between first class business objects.

Note that this is contrary to the position of providing use-case specific struct types coupled to façade interfaces. The goal is to mirror the semantic structure of the server side business objects in the client applications, and put the onus of narrowing any sub sets of the object model into applicable views, which fulfill the necessary use-cases.

Applicability

Structure

Consequences

Implementation

Known Uses

Related Patterns

Data Transfer Object
Transfer Object
Horizontal-Vertical-Metadata – CORBA Design Patterns

Depends

Name: Value Graph

Intent

Provides a means for a set of Value Objects to be managed as a single object graph within a namespace.

Also Known As

Reference

Motivation

Where large numbers of objects and their relationships need to be represented and managed in a consistent fashion on client applications.

Considering that Value Objects have only state, and no business logic, the Value Graph pattern provides for a simple set of behavior used to maintain the integrity of the Value Object relationships and their namespace.

Applicability

Structure

Consequences

Implementation

Known Uses

Related Patterns

Depends

Value Object
Graph Plan

Name: Graph Visitor

Intent

Provides a means for an object graph to be traversed, without elements of the graph becoming coupled to the traversal mechanism or structure defining the traversal path.

Also Known As

Reference

Motivation

When working with object graphs that resemble a prototypical structure, it may be necessary to traverse the graph using the structure as a guide so as to remain independent of the specific graph structure or implementation.

The result would allow functions to by dynamically applied to the graph elements without directly adding new functionality to the underlying implementation or unnecessarily coupling the applied functions to the graph implementation.

Applicability

Structure

Consequences

Implementation

Known Uses

Related Patterns

Graph Plan

Depends

External Polymorphism

Name: Graph Prototype

Intent

Provides a means to source new graph instances from a default graph or structure.

Also Known As

Reference

Motivation

When consistently using commonly structured object graphs, it may prove beneficial to instantiate and source the graphs from a common prototype or (annotated) graph structure description.

Considering the case a “bill of sale” may have one or more “line item” sub-graphs, an idiom or structure description could be defined that provides the necessary elements that would make up a “line item”.

If the description is stored as meta-data, and not hardwired into the system, a “line item” definition could change over time, possibly without interrupting the system.

Applicability

Structure

Consequences

Implementation

Known Uses

Related Patterns

Depends

Architecture Patterns

Name: Consumer Defines the Model Architecture

Intent

Defines an architecture philosophy where servers implementing the business logic for a problem domain, only solve the problem domain, and not the narrower client scenarios associated with page views or other client specific use-cases.

Also Known As

Reference

Motivation

In the same sense a RDBMS server provides data sets sourced from the information model used to structure the persisted values, the "business engine" of an application server should only be concerned with the solution object model and their related semantics and logic. And the client applications should be responsible for defining the structure or closure of the model for a given view.

The converse of this is represented in the trend where application developers create façade interfaces and struct like value objects to fulfill the requirements of a given client application use-case. Resulting in coupling to a given use-case on both the client and server aspects of the architecture.

Applicability

Structure

Consequences

Implementation

Known Uses

Related Patterns

Model-View-Controller

Depends

Horizontal-Vertical-Metadata – CORBA Design Patterns

Name: Horizontal-Vertical-Metadata

Intent

Defines an architecture philosophy resulting in interoperable, flexible, and reusable applications.

Also Known As

Shared Semantics Architecture

Reference

Horizontal-Vertical-Metadata – CORBA Design Patterns

Motivation

Applicability

Structure

Consequences

Implementation

Known Uses

Related Patterns

Depends

Name: Repository Architecture

Intent

Provides a uniform system for shared access to common used resources.

Also Known As

Reference

Repository Architecture – CORBA Design Patterns

Motivation

Applicability

Structure

Consequences

Implementation

Known Uses

Related Patterns

Depends

References

T. Mowbray, R Malveau, CORBA Design Patterns, John Wiley and Sons, Inc, Canada, 1997

E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995