Dynamic Schemas in object database management systems (ODBMS)

Lenny Hoffman, Todd Stavish, Dr Nic Caine, Brian Clark
Objectivity, Inc.

Abstract

Object oriented languages support the development of systems to solve many of today's computing problems where both the processing and data is complex. Data can be complex both in type and relationships. Modeling the data as objects is a natural way to represent this real world data. ODBMSs are an easy way to persist application objects, avoiding the overhead of an object to relational mapping. ODBMSs persist objects of any degree of complexity that the object language can define. However through the lifetime of a system processing and data requirements change resulting in changes to the underlying object model. Managing these changes without having to take the application down is a key requirement in many of these systems. This paper discusses ways to manage these changes.

1. Introduction

Most object oriented languages are static, i.e. the class definitions are compiled into the application. If a class definition changes the application has to be taken down, rebuilt and re-deployed. In this paper we look at 2 different ways to avoid this rebuilding of the application.
Version 2.0 of the ODMG (Object Database Management Group) standard[1] included a meta-object interface for defining (create, read, update, delete classes) a schema in the schema repository dynamically, and an interface to create, read, update and delete instances of these classes without having the concrete class descriptions compiled in to the application.
We will also describe a way to implement a meta-schema (schema of schema) using standard schema and object modeling techniques.

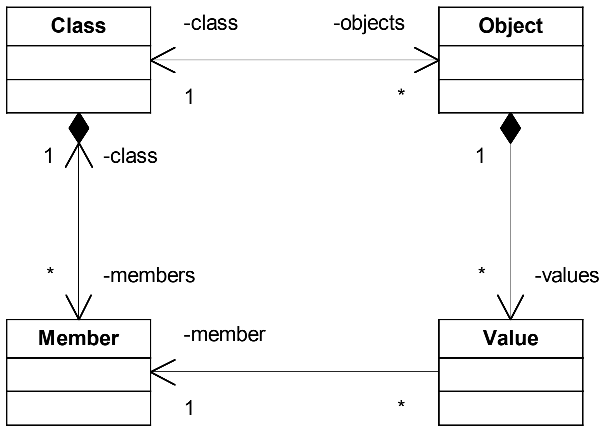
2. Application Programmer Interfaces

Some object databases have implemented the ODMG V2.0 meta-object interface. This provides APIs to dynamically create, read, update and delete class definitions in the schema repository from within the applications, and then from within the same application, or a different application, create, read, update and delete instances of those classes. The advantage of this approach is that the application does not need to be taken down to recompile the application to include the new or modified class definitions. Object conversion can also be achieved dynamically from within the application.
Objectivity/DB provides functionality for the dynamic access of both schema and object instances and separates the write and read functionality so that browsers can be written without having access to the write functionality.

3. *Meta Schema*

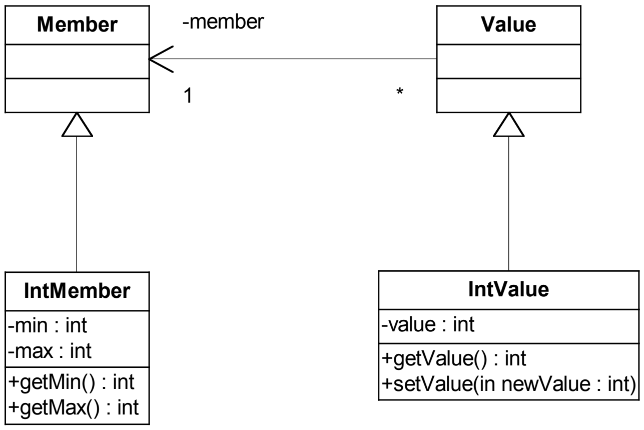
A great number of applications gain flexibility and power by working one level above normal persistent classes. Normal persistent classes in this context are ones that are defined through normal schema definition processes, and are statically tied to a corresponding language class definition. By working with instances that describe the "normal" persistent classes, applications are able to programmatically change schema, and do so without requiring changes in static language class definitions.

This technique is often termed a "meta-schema" and usually takes the following form:



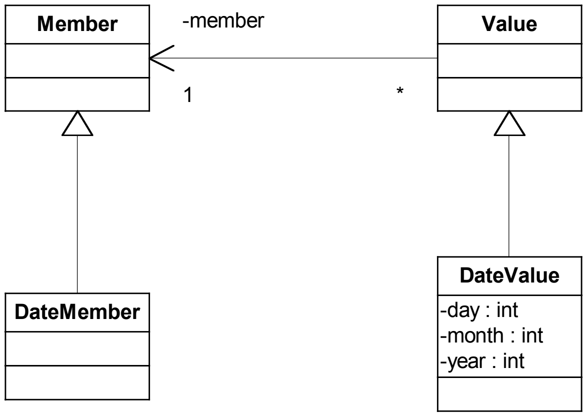
This is the schema as database sees it. Actual classes are described by creating instances of **Class** with varying numbers and types of **Member**s. An instance of programmatically generated class is creating by creating an instance of **Object** and populating it with **Value** instances that correspond to the class instances' members.

Along with the flexibility to generate classes and objects based on them at run time, the approach allows you to define specialized member types, member types not found in object-oriented programming languages such as C++ or Java. Take for example a range limited integer member. In Java and in C++, there is no built in way to associate range information with a primitive, but since we are defining our own member types, this is something we can easily do.



Our new range specifying member **IntMember** inherits from **Member**, and supports the definition of a minimum and maximum value allowable for corresponding **IntValue**s. **IntValue** inherits from **Value**, provides storage of an **Object** instance specific integer value, and in its *setValue* method checks *newValue* against the **IntMember** defined range.

Along with adding new capabilities to member types found in Java and C++, it also becomes a simple matter to add completely new member types. For example:



Here we have added a date as a first class member, not as a referenced or embedded member like we would have to do in Java or C++.

The real thrust of all this modeling flexibility is that you can tailor your class model to match your domain and its needed general capabilities.

In many cases, schema and data migration is a simple matter of propagating changes from the class to all of its instances – notice the bi-directionality of the object-class relationship specifically for this purpose. If such immediate propagation is not desired then the **Class** instance can be versioned (simply a new copy made) before being changed. Then Object instances can be selectively migrated by moving their class reference from the old class to the new and updating values as needed.

4. Summary

A number of Objectivity/DB users have implemented solutions using both API and meta-schema approaches. The API gives the best performance without having to re-compile the application. The schema and database is still dealing with concrete classes. The meta-schema approach provides the most flexibility at a small cost in performance. The meta-schema approach results in lots of small objects with lots of relationships, but that is what object databases are good at anyway.

5. Acknowledgements

The Technical Services Engineers at Objectivity who have helped our users implement these types of solutions.

6. References

[1]ODMG information in this document is based in whole or in part on material from *The Object Database Standard: ODMG 2.0*, edited by R.G.G. Cattell, and is reprinted with permission of Morgan Kaufmann Publishers. Copyright 1997 by Morgan Kaufmann Publishers.