

White Paper: Persistent Data Development Tools Validate the Model Driven Architecture Approach

By Ruth Stento, Manager of Technical Outreach, ObjectStore



Table of Contents

Executive Summary	3
Why MDA Makes Sense	3
Applying MDA to the Persistent Data Model	4
The Role of Tools for Persistent Data Development.....	4
Achieving a Model-Driven Approach for Persistent Data Development	6
Conclusion	9

Executive Summary

The Object Management Group™ (OMG™) promotes Model Driven Architecture® (MDA®) as a platform-independent approach to developing applications. MDA separates business logic from platform technology to facilitate better resource management and improve technical results. One of the primary benefits is the ability to define a platform-independent model, using tools to generate application code that hides platform implementation details from developers.

A variety of tools are available for defining the persistent portion of the model—although they may not fully support MDA standards at this point. Products supporting the full application lifecycle allow architects and developers to define the model, specify object-relational (O-R) mapping, and generate code for persistent objects. For example, tools integrated with the Eclipse open-source IDE support development, compilation, and testing within the same easy-to-use environment. Enterprise development teams have validated the effectiveness of tools by using them to build impressive systems with high-volume transactional throughput, low response times, and high availability.

Why MDA Makes Sense

IT managers seek to control costs and ensure successful results for complex system development projects. Challenges include integrating legacy data and data from silos into accessible forms and choosing from the proliferation of hardware and software platforms for development and deployment—each of which offers its own set of benefits and trade-offs.

OMG's Model Driven Architecture proposes standards for system analysis and development that can alleviate these problems. MDA separates the concerns of application functionality and behavior from the technology in which it will be developed and deployed. The underlying implementation can change without requiring changes to the system that is exposed to the end user.

MDA accomplishes this loose coupling with several modeling phases,^a including the following:

- The Computation Independent Model (CIM) describes the domain requirements—what the end user needs—without trying to solve how the requirements will be achieved.
- The Platform Independent Model (PIM) specifies the parts and services a system would need to satisfy the CIM.
- The Platform Specific Model (PSM) details how a particular platform supports the PIM.

Enterprise architects have applied these approaches in limited areas, but the tools do not yet exist to realize the full promise of MDA. This paper makes the case that MDA principles have already been validated with respect to persistent data access in enterprise applications.

Applying MDA to the Persistent Data Model

Sophisticated tools are available today for creating flexible models such as the CIM and PIM. However, for the persistent part of the data model, breakdowns can occur during PSM definition and even further, at application implementation. Success factors at these phases include: choosing an appropriate technology for implementing data access and using efficient mapping to the physical database.

The data access layer has similar requirements for most applications but project managers acting independently tend to re-invent the wheel and implement data access in a slightly different fashion. Having each group implement their own solution can reduce the benefits derived from a model-driven approach. Inefficient code usually creates a performance bottleneck between the server tier and database and limits scalability.

The good news is that the object-relational impedance mismatch is a problem domain well-suited to the MDA approach. While the vast majority of data is stored in relational database management systems (RDBMSs) or legacy systems, new development most often uses object-oriented languages such as C++, Java, and C#, on platforms including J2EE and .NET.

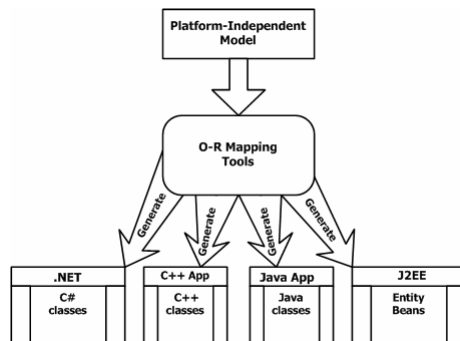
Applying MDA principles such as up-front design of the data model and reducing interdependencies between the model, application code, and the database result in the best outcome—reusable components that contribute to a scalable, flexible system.

The Role of Tools for Persistent Data Development

In the spirit of MDA, tools should allow developers to focus on the problem domain rather than on the low-level data access infrastructure. O-R mapping tools offer a common API that can be used across many applications. These tools can insulate developers from needing to understand the specifics of multiple databases.

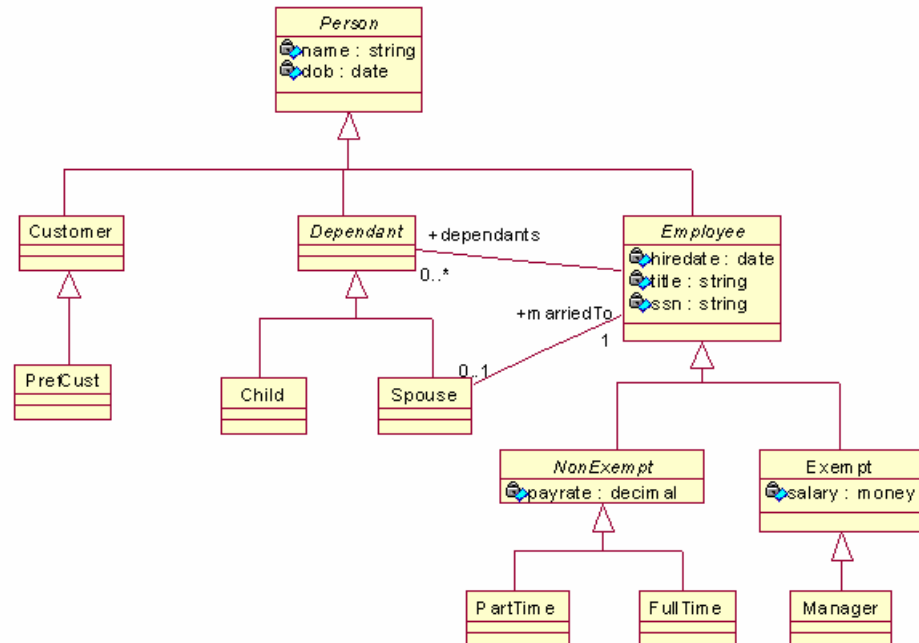
Some vendors offer a choice of interface by providing O-R mapping plug-ins that integrate with IDE's such as Eclipse or Visual Studio. In fact, some allow you to use the same model to generate code for C++, Java, .NET, and J2EE applications. Figure 1 illustrates how such O-R mapping tools abstract the model from the implementation.

Figure 1. Abstraction of Model from Implementation



To see how tools that generate O-R mapping code assist in model-driven development, consider a simple object hierarchy that allows an organization to model both customers and employees. Such an object model might look similar to the one shown in Figure 2. The model represents the structure and semantics of the code that will be generated, but is not expressed as explicit programming.

Figure 2. Simple Object Model



Realistically, mapping decisions must be constrained by the database schema and vice-versa. Many organizations have large quantities of legacy data that must be taken into account when designing new applications. Fortunately, there are several possible ways to map the inheritance structure in the model to database tables and still accommodate new and legacy data schemas:

- With horizontal mapping, each concrete class represents a distinct table. Horizontal mapping sometimes causes duplication in the layout of the database tables (i.e., inherited columns are duplicated).
- Union mapping (sometimes called filtered mapping) allows multiple classes to share a common table and thus reuse columns representing the common data.
- Vertical mapping uses tables at each level of the hierarchy. Such tables contain only the columns for the attributes defined at that level in the hierarchy. To relate the multiple tables required to instantiate an instance of a class requires a join of the appropriate tables through shared key values.

Database schema design is a science that has a large impact on how applications will perform at runtime, and is beyond the scope of this document. However, at the application level, tuning strategies such as caching, database connection pooling, and clustering are important to ensure that an elegant design performs and scales well.

Therefore, a full-featured O-R mapping tool should include the following capabilities:

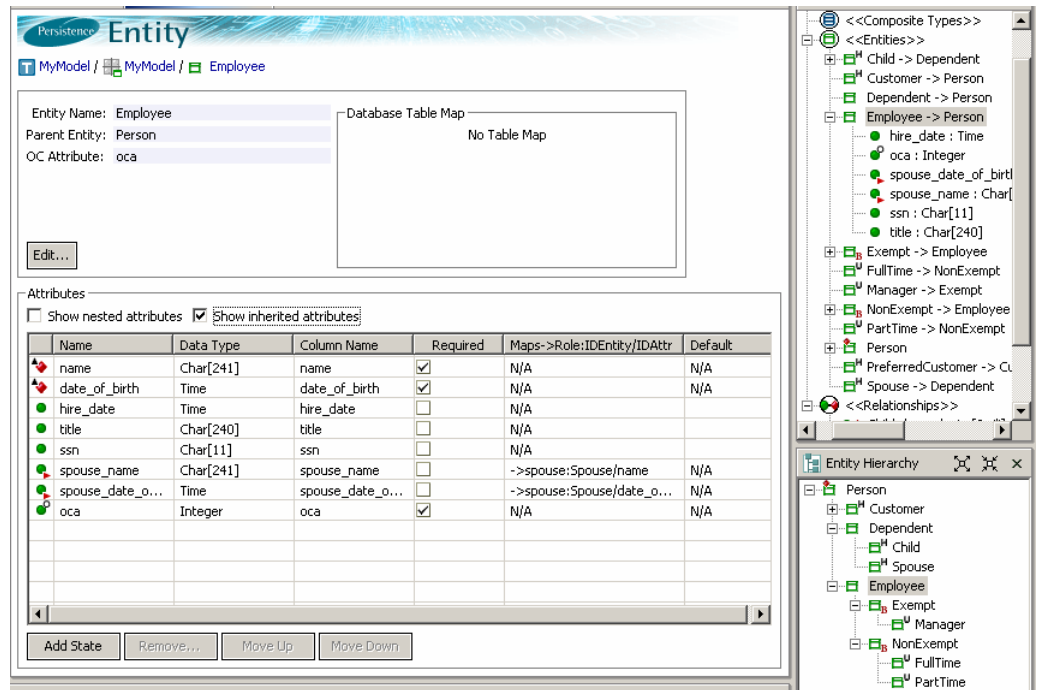
- Flexible mapping
- Code generation
- Platform independence
- Database independence
- Performance tuning

Achieving a Model-Driven Approach for Persistent Data Development

As mentioned previously, tools exist today that follow the principles of MDA with respect to the development of persistent data access. This section uses a tool that provides Eclipse plug-ins^b to illustrate how mapping concerns can be simplified, development efforts can be reduced, and performance and scalability can be ensured for enterprise applications.

Figure 3 shows the Employee object from our example model as it displays in the Eclipse Plug-Ins entity editor window. Notice the decorators that distinguish different types of attributes. For example, the Employee entity inherits the attributes of name and date_of_birth from Person. In this case, the combination of these two attributes will uniquely distinguish an Employee instance and will be mapped to key columns in the underlying database.

Figure 3. Definition of An Employee Entity in Eclipse Plug-Ins



The Plug-Ins for Eclipse support inheritance and relationships. Because of the object-relational mismatch, relationships that may be valid in an object model might not be possible when mapped to a relational schema. In this case, the Plug-Ins for Eclipse enforce constraints by only presenting you with valid choices. If you make a change in the model that invalidates a prior entity or relationship definition, the Plug-Ins warn you and present you with the appropriate editor to fix the problem.

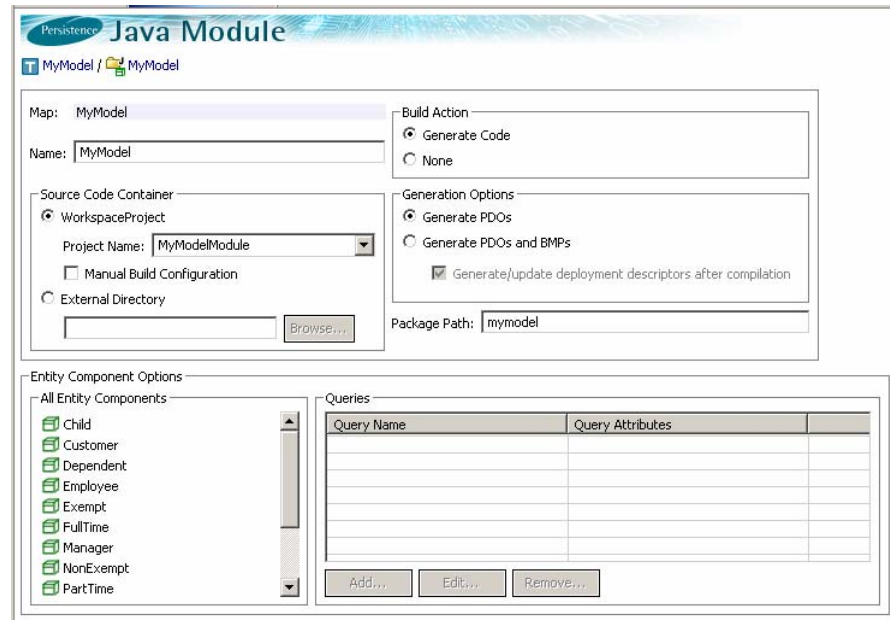
In the Plug-Ins interface, the specification of the entity mapping to a relational database for persistent objects is separate from the specification of the platform, which you define as a Module. In this way, you can generate persistent data objects for a Java application and generate BMP (bean-managed persistent) wrappers for use in a WebLogic or WebSphere application. From the command line, you can use the same model to generate code for C++ or .NET applications. Figure 4 shows the Plug-Ins Module editor where you choose code generation and build options. This is where you can specify optional queries that provide high-speed, efficient access to cached entities at runtime.

Because Plug-Ins for Eclipse generate all of the O-R mapping code for you, they support agile development practices. You can focus on one portion of your system model, iterate

to fine tune it, and then add the next part of the model and re-generate code. The generated classes have areas in which you can add application-specific customizations. The tools preserve these customizations in subsequent code generations.

A unique benefit of this tool is that the generated classes transparently take advantage of high-performance caching—no additional programming is required. You can define additional application-specific queries, which the Plug-Ins will generate.

Figure 4. Separation of Model from Code Generation and Build Options



The generated code supports indexing of cached objects to optimize in-memory access, and you can configure additional indices to match custom queries. In addition, applications running on different platforms can access the same data without compromising data integrity by using built-in distributed synchronization mechanisms.

Figure 5 shows some of the classes generated for the example object model. Note the following classes generated for the Employee entity: Employee.java, contains accessors for attributes and relationships; EmployeeHome.java is the factory class and contains a set of optimized queries; and EmployeeKey.java represents the primary key attributes.

Figure 5. Generated Classes



The following code snippet from Employee.java shows the signature of generated attribute and relationship getter and setter methods:

```
public interface Employee
    extends Person
{
    public abstract Date getHiredate() throws PDOException;
    public abstract void setHiredate(Date val) throws PDOException;
    public abstract String getTitle() throws PDOException;
    public abstract void setTitle(String val) throws PDOException;
    public abstract String getSsn() throws PDOException;
    public abstract void setSsn(String val) throws PDOException;
    public abstract String getTheSpouse_name() throws PDOException;
    public abstract String getTheSpouse_dob() throws PDOException;
    public abstract Spouse getTheSpouse() throws PDOException;
    public abstract void setTheSpouse(Spouse pdo) throws PDOException;
    public abstract Collection getDependents() throws PDOException;
    public abstract void addToDependents(Dependent pdo)
        throws PDOException;
    public abstract EntityState getState(int stateFlag)
        throws PDOException;
    public abstract void setState(EntityState state) throws PDOException;
}
```

Of course, the implementation is also generated. Developers simply write business objects that access the persistent data through these interfaces. To access an employee object attribute, the code becomes as straightforward as the following snippet:

```
// Create the key class
EmployeeKey eKey = new EmployeeKey(name, dob);
try {
    // Find the Employee instance
    Employee emp = EmployeeHome.findByPrimaryKey(eKey);
    // Get the current salary
    salary = emp.getSalary();
}
```

Conclusion

The introduction of this paper asserted that existing systems validate the MDA approach, at least with respect to the development of persistent data access. As supporting evidence, enterprise development teams using a full-featured O-R mapping tool have documented the following key benefits:

- **Increased developer productivity:** By using an O-R mapping tool that generated highly efficient data access code, a large telecommunications company saved nearly \$1.5 million on the development costs of a large-scale software project, and expects to realize more than \$500,000 in additional savings on maintenance and upgrades over the five-year life of the project.
- **Ability to easily integrate application components across platforms:** NetJets uses a COM-to-CORBA bridge to make calls between the J2EE services and the Windows-based clients.^c
- **Ability to switch the development platform or database with a minimum amount of effort:** VCG migrated the data access portion of a mature application (471 classes) from C++ to C# in three days.^d An OEM team switched from using PrimeBase to MySQL without re-writing any code.
- **High performance and scalability for a demanding real-time system:** A leading financial institution saw response times improve from 50 to 200% with a several-fold improvement in the number of concurrent users that could be served using the same hardware.

The model-driven approach to development offers IT professionals an optimal way to handle the increasing complexities of modern systems. With disciplined modeling and planning, development teams can move forward with confidence to implementation and testing.

Tools for O-R mapping and code generation offer significant benefits for persistent data access—at the system or the application level. They support model-driven development that separates the architecture from code generation and implementation. Automatic code generation reduces initial development efforts by 20 to 50% and continues to add value during deployment by reducing maintenance costs.

The built-in caching and synchronization features of code generated from some tools allow developers to focus on their business objectives and application-specific logic rather than manually implementing strategies to increase performance, scalability, and availability. Organizations that must support multiple platforms can use one common model to develop the data access layer for Java, C++, J2EE or .NET applications. At runtime, applications on disparate platforms can synchronize cached data to provide the greatest concurrency. Such tools are well-positioned to evolve and support the full MDA approach.

References

- ^a OMG, Miller J and Mukerji J editors, *MDA Guide Version 1.0.1*, June 2003.
- ^b Screen shots from ObjectStore® EdgeXtend™ Plug-Ins for Eclipse.
- ^c Seeley R, "NetJets Flies with Web service-free SOA," *Application Development Trends*, August 1, 2004.
- ^d Kearnes P, "Extend your Wings and Start the Migration," *dotnetdevelopersjournal.com*, December 2004.



www.objectstore.com

Worldwide and North American Headquarters

Progress Software Corporation, 14 Oak Park, Bedford, MA 01730 USA Tel: +1 781 280 4000 Fax: +1 781 280 4095

EMEA Headquarters

Progress Software Europe B.V., Schorpioenstraat 67, 3067 GG Rotterdam, The Netherlands Tel: +31 10 286 5700 Fax: +31 10 286 5777

UK Office

Progress Software Limited, 210 Bath Road, Slough, Berkshire England SL1 3XE Tel: +44 1753 216 300 Fax: +44 1753 216 301

German Office

Progress Software GmbH, Konrad-Adenauer-Str. 13, 50996 Köln, Germany Tel: +49-221-93-57-90 Fax: +49-221-93-57-978

© 2004 Progress Software Corporation. All rights reserved. Progress, ObjectStore, EdgeXtend, and Cache-Forward are trademarks or registered trademarks of Progress Software Corporation, or any of its affiliates or subsidiaries, in the U.S. and other countries. Any other trademarks or service marks contained herein are the property of their respective owners. Specifications subject to change without notice. Visit www.objectstore.com for more information.