PERSISTENCE®

OBJECT-TO-RELATIONAL

APPLICATION DEVELOPMENT SYSTEMS

AN INTRODUCTORY GUIDE

Object-to-Relational Application Development Systems
An Introductory Guide

Edited by Emily Brower
Written by Christopher Keene

Persistence Software, Inc.
1700 Amphlett Boulevard, Suite 250
San Mateo, California 94402
Tel: +1 415-372-3600
Fax: +1 415-341-8432
www.persistence.com

CONTENTS

CONTENTS

About This Book

This book is a guide for corporate managers and developers working on large-scale object-based systems which require the integration of relational database technology—in other words, virtually every large-scale object development project.

"Persistence is straddling two fast horses. On one side, interest in object-oriented programming is growing and on the other, the market for relational databases is growing. That's not going to top off soon."
Steve McClure, Analyst, IDC

The book provides an overview of market and technical trends creating the need for object-to-relational development systems and explains, in detail, the features and benefits of the Persistence solution.

The first five chapters are primarily market-focused. In these chapters, readers will find answers to such questions as: why is object-to-relational development such an important issue today? what market trends are pushing companies towards mixing the two technologies? what exactly is the technical problem caused by object-to-relational applications and how does Persistence solve that problem? what are other approaches to the problem?

Chapters 6 and 7 provide technical information on the Persistence solution. They address questions such as: What is Persistence, exactly? technically, how does it work? how difficult is it to build an application with Persistence?

CHAPTER 1

Why Object-to-Relational?

Time is speeding up. In today's marketplace, everything is happening faster. Business cycles are accelerating as new market developments and technologies appear on the scene at breakneck pace. To maintain a competitive edge in this fast-forward environment, information-intensive industries, such as telecommunications, financial services and transportation, are building more complex custom applications more frequently. In this chapter, we examine how market pressures translate to application development requirements and explain how object-to-relational systems provides the right technology mix to meet these current development needs.

Market Problem: Business-Critical Application Development

Accelerating business cycles require fast application development. Business re-engineering requires tools that build applications which mirror business processes and adapt easily to change. Development tools for today's business environment must speed time-to-market and adapt easily to change.

Information is also gaining value in today's marketplace. To redefine business processes, corporations are consolidating business data to improve access. In industry parlance, this strategy is called data warehousing. As the intrinsic value of this information grows, so does the need to protect it, to ensure its integrity and validate its accuracy. Application development environments must include tools and features to ensure data integrity. Development tools must also support fast data access so that custom applications can process large volumes of data.

In addition to performance, flexibility and reliability, application development tools must work with existing technologies and legacy systems and provide a quantifiable return-on-investment. Few companies can afford to throw out existing technical infrastructures for new applications, no matter how great the benefits.

Figure 1-1. shows how today's business environment is driving technology requirements.

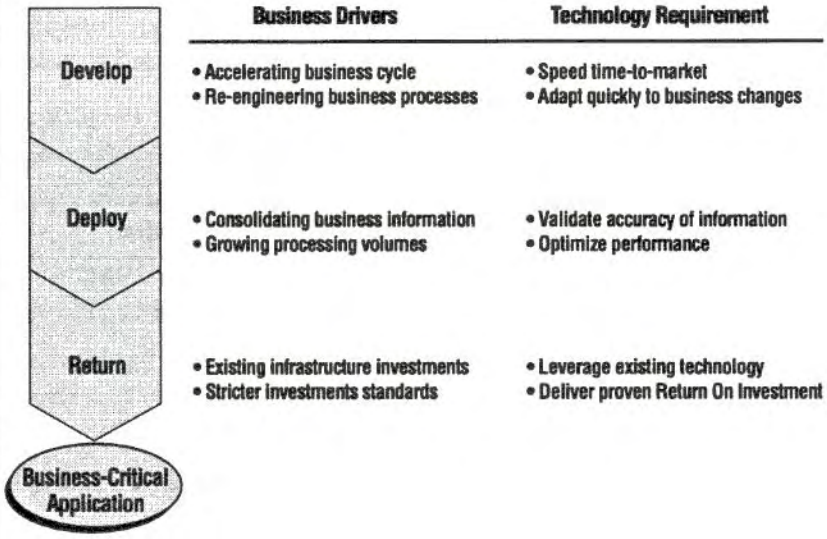

Figure 1-1. Requirements for business-critical applications.

Solution: Object-to-Relational Development Systems

An object-to-relational development system provides all the features required by today's market place.

With object technology, developers can build applications that correspond more closely to the actual operations of a business. An order entry system, for example, can be constructed using objects called commercial customers and accounts. The commercial customer object can, in turn, inherit capabilities from a more general customer object that contains one or more accounts. In addition, object-oriented applications are easy to maintain and update because object development minimizes development time.

Relational database technology provides the other half of the equation. It provides the best solution for applications which use tabular data accessed by multiple applications with different

data schemas. The rich transaction and query and environment of relational databases ensure data integrity. Relational technology is an industry standard and scales to large applications that require high-performance. And, most importantly, a large percentage of business data is stored in relational databases.

Figure 1-2. shows how technology requirements are leading companies to select object-to-relational technology.

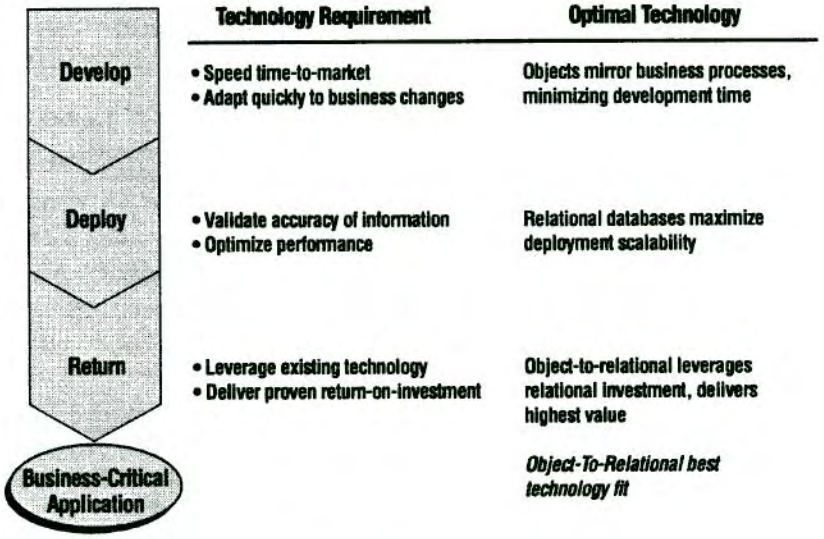

Figure 1-2. Object-to-relational technology meets today's development requirements.

Persistence is the market-leading provider of object-to-relational technology. The Persistence object-to-relational solution provides all the benefits of object-to-relational technology. It supports fast application development, application flexibility, data integrity, application performance, platform independence and a high return-on-investment. The following chapter describes in more detail the required technical features of an object-to-relational development system that provide access to these benefits.

CHAPTER 2

Object-to-Relational Requirements

The previous chapter described the benefits that object-to-relational technology provides for today's competitive business environment. Though they provide the right technical solution, integrating object and relational technology is like mixing apples and oranges. The two technologies are fundamentally different in ways that create complex technical challenges for developers. This chapter describes these technical challenges and outlines the required features of an object-to-relational development environment.

Speed Time-to-Market

Objects have explicit structure and relational data has implicit structure. These conflicting structures create a mapping mismatch. To overcome this mismatch, the explicit structure of an application's objects must be mapped to the database's tables and rows—no mean feat. For programmers, mapping relational data to object brings up complex development questions such as: what columns are foreign keys? what tables do foreign keys refer to? and what is the cardinality of relationships?

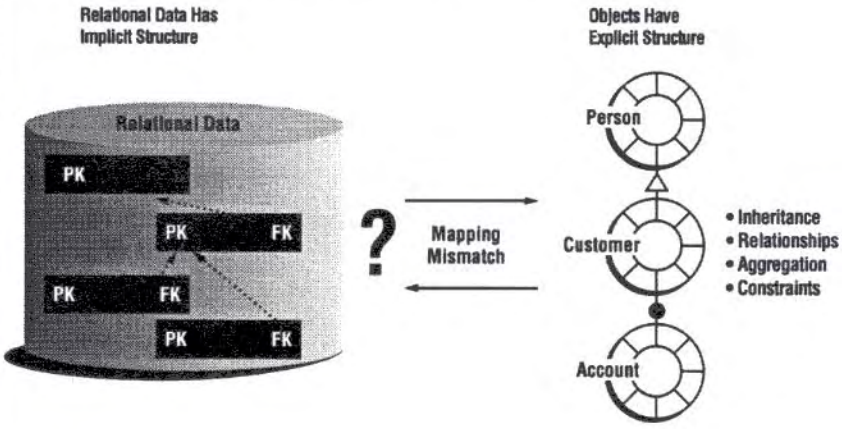

Figure 2-1. Mapping mismatch.

Companies that tackle object-to-relational programming by hand face a daunting challenge. A C++ developer interfacing object

classes to a database must complete a series of difficult and time-consuming tasks. These include mapping classes to relational tables; writing methods to create, read, update and delete instances; and managing foreign key information for aggregation—tasks that would take at least one week per class to implement. Any major change to the object model or database schema would necessitate reimplementing significant portions of the affected classes.

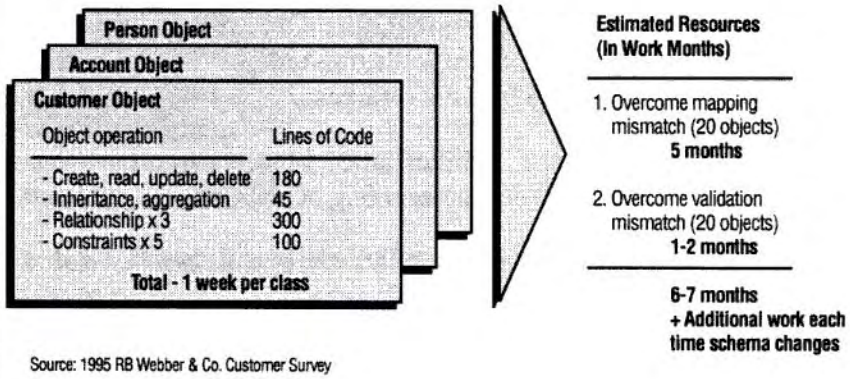

Figure 2-2. The cost of handcoding.

"Fusing a relational database to objects requires a tremendous amount of manual labor. We need tools to do this for us." Christine Comaford, columnist, PC Week

Hand-coding also presents other road blocks. With traditional languages, such as C or COBOL, developers routinely manage the mapping between application data and relational data. While this approach works with object-oriented applications, it is made more difficult by the greater modeling power of object languages. Hand-coded objects also lack the robustness to be revised by other developers.

Objects that access relational data must be:

- Complete. They must perform all logic data operations (e.g. create, read, update, delete), not just a subset of operations required by a particular project.

- Robust. They must provide high reliability in all applications, not just the original project environment.
- Universal. They must contain no hidden dependencies on a particular platform, database or compiler.

A well-designed object-to-relational development system creates objects that meet these requirements. It also reduces the complexity, and therefore the development time, of mapping objects to relational data. Objects created with object-to-relational development tools must map the following basic object constructs to a relational database:

- Inheritance: objects can inherit capabilities from other objects (e.g., Customer may inherit from Person).
- Associations: objects can be related to other objects (e.g., each Customer may have an association with a Company).
- Aggregation: objects can be composed of other objects (e.g., a Customer may contain Accounts).

Adapt Quickly to Change

Object-to-relational tools must be able to adapt quickly to changes in the object model or relational schema. In order to manage change, an object-to-relational system must be closely integrated with both relational technology as well as other development tools. For example, it must support data dictionaries so that when a data dictionary is changed the object model automatically changes as well. Object-to-relational systems must also be tightly integrated with CASE tools as well to support the generation of relational mappings when object models change.

Validate Object Data

Objects lack the validation capabilities of relational technology. They are designed to hide internal data from other objects, which simplifies programming by ensuring that an object can be sepa-

rated from its own information. At the same time separation between objects and information create significant integrity issues for objects accessing data.

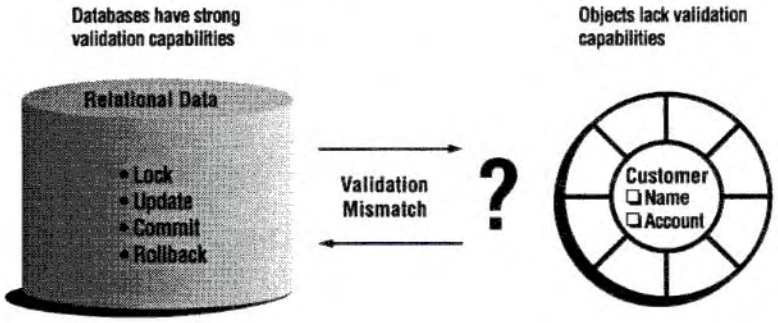

Figure 2-3. Validation mismatch.

Once relational data is placed in an object, developers must find ways to address the following issues:

- What locks are placed on that data in the database?
- Who is responsible for knowing if the data has been changed in the object?
- When are these changes written to the database?
- Where is the checking done upon transaction commit to ensure that all changes were made successfully?

If these questions are not adequately addressed, data can be easily corrupted. For example, data may not be locked appropriately in the database or changes to an object may not be written to the database. Or, if a transaction fails, associated objects may not be reset to their initial values.

Relational databases are designed to provide a high level of data integrity. Features such as data locking and updating ensure that the right data is saved in a database. A well-designed object-to-

relational system needs to be tightly integrated with these database features in order to provide data validation.

Optimize Application Performance

Fast, reliable performance is the life blood of telecommunications, transportation and financial services applications. To speed performance an object-to-relational system must provide access to advanced database features such as stored procedures, triggers and arrays.

"Object-relational enablers are critical because they allow developers to get the benefit of object-oriented tools without throwing out their relational databases."
Hugh Bishop, analyst, The Aberdeen Group

In addition, an object-to-relational system should also optimize the use of structured object information to enhance performance. For example, navigation of an object's hierarchical structure should be used whenever possible to avoid programming database joins, which slow application performance. By navigating data cached in an object, queries and updates can be performed much more efficiently. This provides a significant performance benefit because in-memory access is three orders of magnitude faster than disk access and costly network traffic is avoided.

Leverage Existing Technology

Object-to-relational systems should be portable across all platforms and databases in order to gain many of the benefits described above. Portability eliminates the risks associated with dependence on one platform or database—dependencies can significantly raise the total cost of a development project.

Provide High Return-on-Investment

Today's business climate places stringent restrictions on technology investment, and project managers are under pressure to show a return on their technology expenditures. Any development tool must provide a high return-on-investment (ROI), and object-to-relational systems are no exception. Through shortened development cycles and decreased maintenance needs, a well-designed object-to-relational system can provide a lot of bang for

the buck. Chapter 5 of this Guide provides more detail on the ROI of object-to-relational systems.

Persistence: an Object-to-Relational Development System

To summarize, an object-to-relational development system must provide the following features:

- Developer productivity to speed time-to-market.
- Adaptability for integrating business change.
- Information validation to ensure data integrity.
- Performance optimization of for fast performance of deployed application.
- Platform independence to leverage existing technology.
- High return-on-investment to justify expenditures.

The Persistence solution meets all these requirements. It provides tools that automatically generate object mapping to speed application development. This automatic data mapping provides flexibility: classes can be regenerated if the object model schema changes.

In addition, tight integration with relational database features allows Persistence applications to validate object data and optimize application performance. Persistence supports all major databases, operating platforms and development tools, providing complete technical independence. The combination of these features allows Persistence to provide a very high ROI. The next chapter provides a detailed explanation of the Persistence solution's features and benefits.