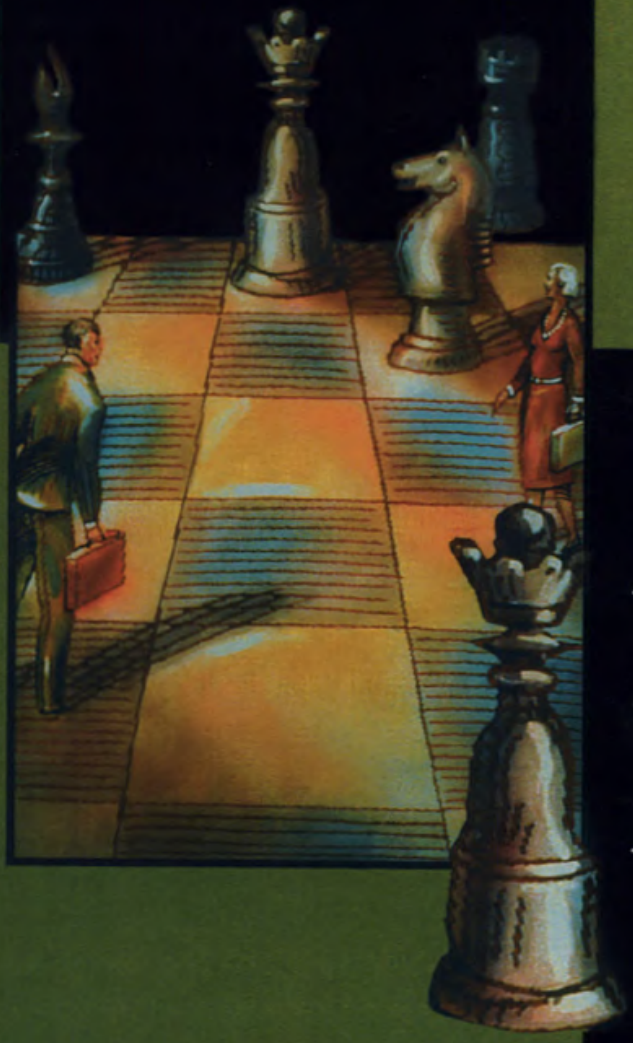
WHAT Not HOW



The Business Rules Approach to Application Development

C. J. Date

Copyrighted material

WHAT Not *HOW*

The Business Rules Approach to Application Development

C. J. DATE

ADDISON–WESLEY

Boston • San Francisco • New York • Toronto • Montreal
London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Copyrighted material

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and we were aware of a trademark claim, the designations have been printed in initial caps or all caps.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers discounts on this book when ordered in quantity for special sales. For more information, please contact:

Pearson Education Corporate Sales Division
One Lake Street
Upper Saddle River, NJ 07458
(800) 382-3419
corpsales@pearsontechgroup.com

Visit AW on the Web: www.awl.com/cseng/

Library of Congress Control Number:

00-131706

Copyright © 2000 by Addison-Wesley

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher. Printed in the United States of America. Published simultaneously in Canada.

Text printed on recycled and acid-free paper.

ISBN 0201708507

3 4 5 6 7 8 MA 03 02 01 00

3rd Printing September 2000

Copyrighted material

Contents

Copyrighted material

Preface

An exciting new technology called **business rules** is beginning to have a major positive impact on the IT industry—more precisely, on the way we develop and maintain computer applications. The aim of this book is to explain what this new technology is all about, and why you should care.

I must make it very clear right away that the book is not impartial. I'm very enthusiastic about business rules!—and I hope you will be too, when you've finished reading. In other words, this is definitely a book with an attitude: It explicitly champions the business rules idea, and it describes and explains what in my opinion are the merits and benefits of that idea. Why am I so enthusiastic? For two reasons: because of what the technology can do, and because it is so squarely in the spirit of "the original relational vision." And the book's structure reflects these two points. To be more specific, Part I is an overview of what business rule technology consists of and what it makes possible, and Part II then takes another look at the ideas presented in Part I and considers them from an explicitly relational point of view.

I should also make it clear that the book grew out of the script for a live presentation. As a consequence, the style is a good deal chattier than my usual writing, and the tone is possibly a little shrill on occasion . . . In the interests of full

Copyrighted material

disclosure, I must also explain that the presentation and the book were both produced under an agreement with Versata Inc. (formerly known as Vision Software Tools Inc.), a company that has a business rules product to sell. However, the book is not about Versata specifically, nor is it about any other specific company or product; rather, it's about business rule technology in general. What's more, "the views expressed are my own"; they're not necessarily endorsed by Versata, nor by any other vendor. Equally, I don't mean to suggest that all of the features we're going to be examining can be found in all of the commercially available products (or in some cases, perhaps, in any of them!). The book describes how business rule systems work in *general* and in *principle;* it doesn't necessarily correspond exactly to the way any given product works in practice.

Who should read this book: Part I of the book is meant to introduce business rule technology to the widest possible audience. It's deliberately not very technical; in fact, it's intended primarily as a "manager's guide" to the subject, though I do believe that technologists, especially people concerned with developing databases and applications in the traditional way, should benefit from it as well. All you need in order to understand Part I is a basic knowledge of what databases and applications are all about, together with a broad idea of what's involved in the traditional approach to developing such databases and applications.

Part II of the book is a little more technical in nature, but not very much so; the primary target audience is still basically as for Part I, and in any case most of the technical background required to understand the overall message is explained in the text itself.

How to read the book: Part I is meant to be read in sequence as written and in its entirety; skipping chapters or reading them in a different order is not recommended, at least not on a first reading. Part II can be skipped if you like, but if you do read it then I would strongly suggest, again,

Copyrighted material

that you read it in sequence and in its entirety, at least on a first reading. Of course, the book is quite short, and you could probably read the whole thing in a single sitting if you felt like it.

Acknowledgments: First of all, I'd like to thank the people at Versata (especially Mike DeVries and Val Huber) for supporting me in the writing of this book, and Gary Morgenthaler for suggesting that I write it in the first place. Second, I'd like to thank Manish Chandra, Paula Hawthorn, Keri Anderson Healy, Vi Ma, Rahul Patel, and Ron Ross for help with technical questions and the like during the preparation of the manuscript. Ron Ross in particular deserves a special mention for his persistence over the years in trying to make me buckle down and get my thoughts on this subject into some kind of order . . . Thanks, Ron! Third, I'm very grateful to my reviewers Hugh Darwen, Val Huber, Paul Irvine, Haim Kilov, Rahul Patel, Ron Ross, and David Wendelken for their generally enthusiastic and helpful comments on earlier drafts of the manuscript. Finally, I'm grateful, as always, to everyone at Addison-Wesley (especially Paul Becker and Ross Venables) for their encouragement and support throughout this project, and to my editor Elydia Davis for her usual sterling job.

C. J. Date
Healdsburg, California
2000

Copyrighted material

PART

I

Overview

This first part of the book is intended for anyone who wants to gain an understanding of what business rules are all about at an overview kind of level. It's deliberately not very deep, technically speaking. It consists of eight short chapters, as follows:

1. What's the Problem?
2. Business Rules Are the Solution!
3. Presentation Rules
4. Database and Application Rules
5. The Data Model
6. Potential Advantages
7. Potential Disadvantages
8. Summary of Part I

The material is written on the assumption that you have at least a basic knowledge of what databases and applications are, and that you also have some general idea of what's involved in the traditional approach to developing such databases and applications.

Copyrighted material

What's the Problem?

1

When they first began to appear, in the early 1950s or so, computers were very hard to use—they required very specialized skills, and you really had to be a computer technician in order to use them at all; originally, in fact, you probably had to be a hardware engineer. Over time, however, computer systems have become much more "user-friendly" and easy to use, thanks to a continual *raising of the level of abstraction:* so much so, in fact, that now you can use a computer effectively even if you have almost no knowledge of how its internals work at all (much as you can drive a car effectively even if you don't know what goes on under the hood). Here are a few familiar examples of that "raising of the level of abstraction" that have taken place over the years:

- 1GLs ⇒ 2GLs ⇒ 3GLs ⇒ 4GLs

  Programming languages have evolved through several "generations," from first generation languages (1GLs) to at least a fourth generation (4GLs). Just to remind you: 1GLs were machine languages; 2GLs were assembler languages; 3GLs were the so-called "high-level" languages (COBOL, Fortran, and the rest); 4GLs were various proprietary languages, such as FOCUS from

Copyrighted material

Information Builders, Inc. Some people regard SQL as a 4GL [16].*

- Sequential files ⇒ indexed (ISAM) files ⇒ hierarchic and network databases ⇒ SQL tables

  Over the years, more and more of the details of storing and managing data have been taken over by the system; in a word, they've been *automated*. Nowadays, it's the system, not the user, that's responsible for finding data as and when required (and finding it fast); it's the system, not the user, that's responsible for recovering data in the event of failure; it's the system, not the user, that's responsible for protecting data from concurrent update; and so on.

- Specialized languages and interfaces: for example, RPG, SQL, QBE, "visual programming" (QBF, ABF), spreadsheets, . . .

  This one is more or less self-explanatory. However, I'd like to elaborate briefly on QBF and ABF, because they're directly relevant to some of the ideas we'll be examining in the next few chapters. QBF—*Query By Forms*—allows you to do database queries and updates† by making simple entries in a form on the screen. ABF—*Applications By Forms*—allows you to develop applications in the same kind of way, and those applications in turn also use on-screen forms as the interface with the user. *Note:* QBF and ABF both grew out of work originally done in the early 1980s at the University of California

*Throughout this book, numbers in square brackets refer to publications listed in the References and Bibliography section at the end of the book.

†In accordance with normal usage, this book uses the term "update" generically to include all three of the familiar operators INSERT, DELETE, and UPDATE.

Copyrighted material

at Berkeley [26]; they were first commercialized in the Ingres product, originally from Relational Technology Inc., now—under the name Ingres II—from Computer Associates International Inc.

Let me add a word on spreadsheets, too. Spreadsheets raised the level of abstraction, in their particular field of application, by getting away from writing programs *entirely* (nobody today would use Fortran to write an application to perform spreadsheet-style processing). There's a very direct parallel here with business rule systems, as we'll see.

In a nutshell, then, it should be clear that the historical trend has clearly always been away from *procedural* and toward *declarative*—that is, from **HOW** to **WHAT.** HOW means saying how, step by step, the work is to be *done;* WHAT just means saying what the work to be done *is.*

So why is this trend A Good Thing? The answer is, of course, that declarative (WHAT) means the system does the work, while procedural (HOW) means the user does it. In a nutshell:

Declarative is better than procedural!

So wouldn't it be nice if we could do *all* of our application development work declaratively? Such has indeed been a goal for many, many years (people have been talking about the possibility of **fully compilable and executable specifications** ever since the 1970s, if not earlier [28]). In other words, wouldn't it be great if we could simply *specify* our applications precisely, and get the system to *compile* those specifications into executable code?

Well, we're getting there. As we'll see.

The advantages are obvious: *Productivity,* of course—the work gets done much more easily and much more

Copyrighted material

quickly; and numerous subsidiary benefits follow, including in particular various kinds of *independence*. One familiar kind is *data* independence, which lets us make changes (for performance reasons, for example) to the way the data is physically stored on the disk, without having to make any corresponding changes in applications that use that data. And there are many other kinds of independence too, some of which we'll be looking at later in this book. The basic advantage in all cases is that they make applications immune to certain kinds of change (in particular, immune to certain kinds of *business* change). And that's a good thing, because—as we all know—the only thing that's constant in life is change.

But what exactly is an application? Obviously, it's the implementation of some business function—for example, "insert an order line item," "delete an order line item," "update the quantity on hand of some part." In general, an application involves three parts or components:

Applications have three components

1. **Presentation aspects**
2. **Database aspects**
3. **Aspects specific to the business function *per se***

Presentation aspects are the ones having to do with the end-user interface—displaying forms to the end user, accepting filled-out forms from the end user, displaying error messages, producing printed output, and so forth. Database aspects are the ones having to do with retrieving and updating database data in response to end-user requests and end-user entries on forms (they're the portions that interact with the database server, also known as the DBMS). Finally, "aspects specific to the business function *per se*" might be thought of as the application proper—they're the ones that specify the actual processing to be carried out in order to implement the business function, or in other words the ones that effectively implement the business's policies and practices.

Copyrighted material

That is, specify business processing declaratively, via business rules—and get the system to compile those rules into the necessary procedural (and executable) code. And just how we might actually be able to do this is the subject of the next few chapters.

Having set the scene, as it were, let me close this introductory chapter with a quote to support some of the things I've been saying. It's taken from an interview [15] with Val Huber of Versata Inc. (formerly known as Vision Software).

> **Years of experience with information system development have taught us two important lessons—it takes far too long to turn a relatively simple set of requirements into a system that meets user needs, and the cost of converting existing applications to new technologies is prohibitive . . . The factor underlying both of these problems is the amount of code it takes to build a system . . . If code is the problem, the only possible answer is to eliminate the coding by building systems directly from their specifications. That's what the rule-based approach does.**
>
> **—Val Huber**

As you can see, Huber is effectively suggesting, again, that what we should be aiming for is *compilable and executable specifications*. Observe in particular that he touches on two separate problems with the old-fashioned way of doing things:

- The time it takes to build applications in the first place;
- The difficulty of migrating existing applications to take advantage of new technologies as they become available (for example, moving a client/server application on to the Web).

I'll come back to both of these problems in Chapter 6.

Just to be definite, let's assume this database is an SQL database specifically. Then the boxes in the figure correspond to SQL tables, and the arrows correspond to foreign keys that relate those tables to one another, logically speaking. For example, there's a foreign key from the ORDER table to the CUSTOMER table, corresponding to the fact that every individual order must be placed by some customer.

By the way, if you'd rather think of CUSTOMER and the rest not as SQL tables as such but rather as *entity types*, well, that's fine; in some ways, in fact, it might be better to talk in such terms, since they're not so specific. Business rules aren't specific to SQL databases! But I'll stick to SQL, for the reason already given.

Observe now that:

- Each customer has many orders (but each order is from just one customer); each order has many line items (but each line item belongs to just one order); each part is involved in many line items (but each line item involves just one part).
- The foreign key relationships can be thought of, in part, as *existence dependencies:* An order can't exist unless the corresponding customer exists, an order line can't exist unless the corresponding order and part both exist.
- Those existence dependencies are business rules! Indeed, foreign key constraints in particular are an important special case of business rules in general, and I'll have quite a lot more to say about them in Part II of this book.

Here then, in outline, are SQL definitions—that is, CREATE TABLE statements—for the four tables in the customers and orders database:

Copyrighted material

SQL definition for customers and orders

```
CREATE TABLE CUSTOMER
       ( CUST# . . . ,
         ADDR . . . ,
         CREDIT_LIMIT . . . ,
          . . . ,
         PRIMARY KEY ( CUST# ) ) ;

CREATE TABLE ORDER
       ( ORDER# . . . ,
         CUST# . . . ,
         PAID . . . , ---------- yes or no
         SHIPPED . . . , ---------- yes or no
          . . . ,
         PRIMARY KEY ( ORDER# ) ,
         FOREIGN KEY ( CUST# ) REFERENCES CUSTOMER ) ;

CREATE TABLE LINE_ITEM
       ( ORDER# . . . ,
         LINE# . . . ,
         PART# . . . ,
         QTY_ORD . . . ,
         ORD_PRICE . . . , ---------- fixed at time of order
          . . . ,
         PRIMARY KEY ( ORDER#, LINE# ) ,
         FOREIGN KEY ( ORDER# ) REFERENCES ORDER ,
         FOREIGN KEY ( PART# ) REFERENCES PART ) ;

CREATE TABLE PART
       ( PART# . . . ,
         CURRENT_PRICE . . . ,
         QTY_ON_HAND . . . ,
         REORDER_LEVEL . . . ,
          . . . ,
         PRIMARY KEY ( PART# ) ) ;
```


Copyrighted material

Some points to note regarding these definitions:

- The CUSTOMER table includes a CREDIT_LIMIT column, with the obvious semantics.
- The ORDER table includes two yes/no columns, PAID and SHIPPED, that indicate whether the customer has paid for the order and whether the order has been shipped, respectively.
- The LINE_ITEM table includes the part number (PART#), the quantity ordered (QTY_ORD), and the corresponding order price (ORD_PRICE). The order price is locked in at the time the order is placed and doesn't change, even if the current price of the part does subsequently change.
- The PART table includes the current price, the quantity on hand, and the reorder level, all with the obvious semantics.
- Finally, note that the PRIMARY KEY and FOREIGN KEY clauses correspond to business rules (and of course they're stated declaratively). For example, it's a business rule that every customer has a unique customer number; as mentioned earlier, it's also a business rule that every order involves exactly one customer; and similarly for all of the other primary and foreign key declarations.

Now let's consider a typical business function involving this database—"insert line item," say. The way it works goes something like this:

- By clicking on a menu item or something of that nature, the end user asks for a form corresponding to the LINE_ITEM table to be displayed on the screen.

Copyrighted material

- You can think of that form as a form in the style of QBF (recall that we discussed QBF briefly in the previous chapter): It will include among other things fields corresponding to the customer number, the order number, the part number, and the quantity ordered.
- The end user fills in those fields appropriately—that is, he or she provides the necessary information regarding the new order line—and clicks on "enter" or "save" or something of that kind.

The "insert line item" application is now invoked and carries out the following tasks (among many others, of course):

A. It checks the customer's credit limit.

B. It computes the order total.

C. It determines whether the part needs to be reordered.

A., B., and C. here are business requirements that must be met in order to carry out the overall business function. Incidentally, there's an important point here that I'll come back to in a little while (when I discuss *reuse*): Those very same business requirements might also need to be met as part of certain other business functions—for example, "change line item" or "delete line item." But let's concentrate on the "insert line item" function for the time being.

For each of these three business requirements, then, the application developer will have to specify a corresponding set of business rules. In the case of the "check credit limit" requirement, for example, the rules might look something like this (of course, I'm simplifying the syntax considerably, for obvious reasons):*

*It's worth mentioning that—as will quickly become clear—the process of developing these rules precisely mirrors the "analysis interview" process: The analyst gets the business user (a) to state the business objectives and (b) to provide definitions of terms that get introduced in step (a).

Copyrighted material

not just talking about applications in the classical sense. I'll have a little more to say on this point in Chapter 8.

As you can see, then, these rules are fairly declarative (nonprocedural). **But they can be compiled into procedural code.** In other words, the rules are executable (loosely speaking). So we've specified the application in a purely declarative way—we haven't explicitly written any of the usual procedural code at all—and yet we've still wound up with running code: an application that can be executed on the machine.

By the way, the procedural code produced by the compiler isn't just executable code—it is (or should be) *optimized* code as well. That is, the rules compiler is (or should be), specifically, an optimizing compiler. I'll have more to say on this particular point in several subsequent chapters.

That's the end of the example for now (though we'll be coming back to it at several points in the next few chapters). To repeat, we've just built an application without writing any procedural code! And there are many immediately obvious advantages that accrue from this way of doing things. Here are some of them:

- First of all, of course, the declarative rules replace many pages of hand-written procedural code. Each of those rules could easily correspond to a couple of hundred lines of 3GL code! This is the source of the productivity benefit, of course.

- Next, the rules are applied and enforced—"fired," to use the jargon—on *all relevant updates*. I touched on this point before, when I observed that the very same business requirements might need to be met as part of several different business functions. To be more specific, although the rules we looked at in the example were specified as part of the process of building the application called "insert line item," *they're relevant to other applications too*. For example, the application called "delete line item"

Copyrighted material

Database Technology

"What I think Date has done is nothing less than to lay out the foundational concepts for the next generation of business logic servers based on predicate logic. Such a breakthrough should revolutionize application development in our industry—and take business rules to their fullest expression."

—Ronald G. Ross, Principal, Business Rule Solutions, LLC
Executive Editor, *DataToKnowledge Newsletter*

The way we build computer applications is about to change dramatically, thanks to a new development technology known as business rules. The key idea behind the technology is that we can build applications declaratively instead of procedurally—that is, we can simply state WHAT needs to be done instead of HOW to do what needs to be done. The advantages are obvious: ease and rapidity of initial development and subsequent maintenance, hardware and software platform independence, overall productivity, business adaptivity, and more.

What Not How: The Business Rules Approach to Application Development is a concise and accessible introduction to this new technology. It is written for both managers and technical professionals. The book consists of two parts: Part I presents a broad overview of what business rules are all about; Part II then revisits the ideas in Part I and shows how they fit squarely into the solid tradition of relational technology. Topics covered include:

- Presentation rules
- Database and application rules
- Building on the data model
- Potential advantages and disadvantages
- A new look at relational fundamentals
- Business rules and the relational model

Overall, the book provides a good grounding in an important new technology, one poised to transform the way we do business in the IT world.

C. J. Date is an independent author, lecturer, researcher, and consultant specializing in relational database systems, a field he helped pioneer. Among other projects, he was involved in technical planning for the IBM products SQL/DS and DB2. He is best known for his books, in particular, *An Introduction to Database Systems* (7th edition, Addison-Wesley, 2000), the standard text in the field, which has sold well over half a million copies worldwide. Mr. Date is widely acknowledged for his ability to explain complex technical material in a clear and understandable fashion.

http://www.aw.com/cseng/

Cover design by Karin Hansen
Cover illustration by Todd Davidson, Image Bank
♻ Text printed on recycled paper

ADDISON-WESLEY
Pearson Education

ISBN 0-201-70850-7
90000
9 780201 708509

Copyrighted material