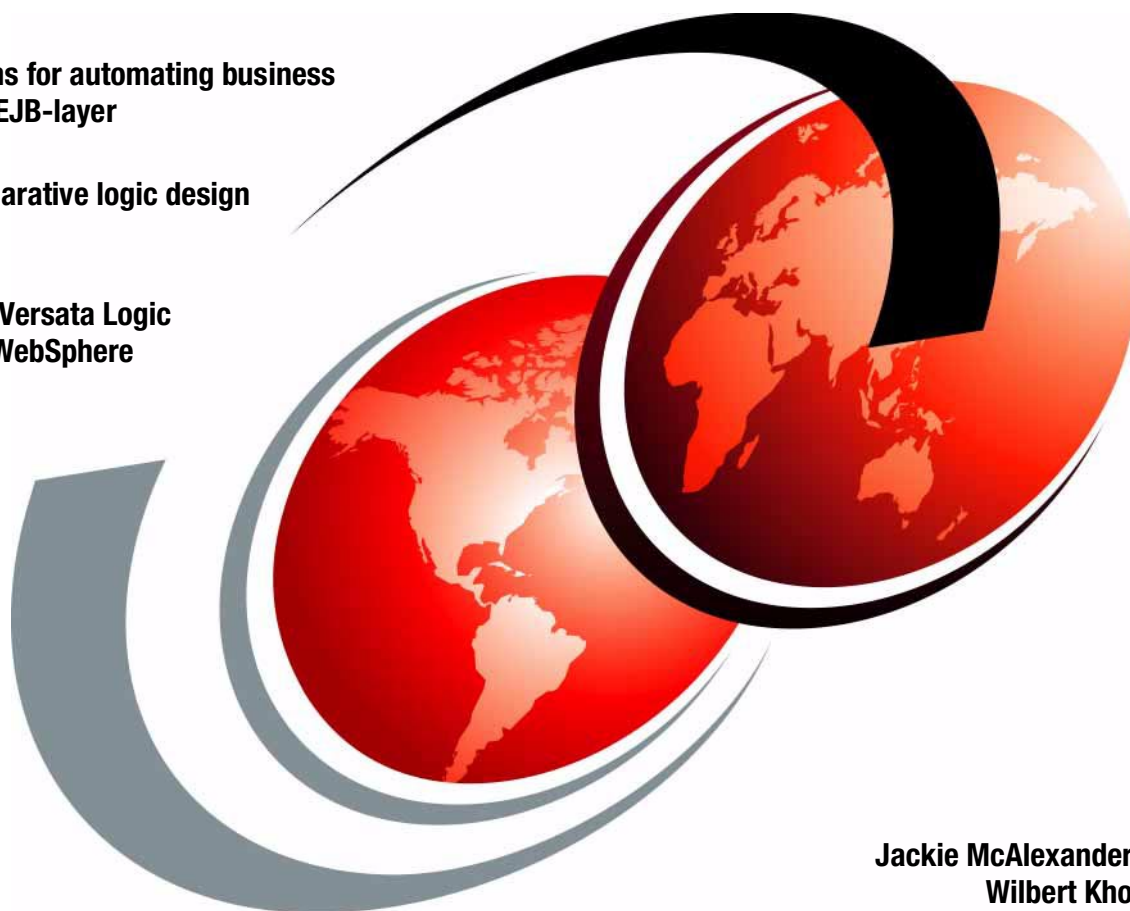


# Application Development Using Versata Business Logic Automation for WebSphere

Learn options for automating business logic in the EJB-layer

Explore declarative logic design using rules

Understand Versata Logic services in WebSphere



Jackie McAlexander  
Wilbert Kho

**Redbooks**





International Technical Support Organization

**Application Development Using Versata Business  
Logic Automation for WebSphere, Patterns for  
e-Business Series**

November 2001

**Take Note!** Before using this information and the product it supports, be sure to read the general information in “Special notices” on page 89.

### **First Edition (November 2001)**

This edition applies to the Versta Business Logic Server x.x and IBM WebSphere Enterprise Server x.x for use with the on Windows NT 4.0/2000 operating system.

This document created or updated on September 20, 2001.

Comments may be addressed to:  
IBM Corporation, International Technical Support Organization  
Dept. 5KNA Building 80-E2  
650 Harry Road  
San Jose, California 95120-6099

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2001. All rights reserved.

Note to U.S Government Users – Documentation related to restricted rights – Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

# Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition may also include minor corrections and editorial changes that are not identified.

Summary of Changes  
for SG24-6510-00  
for Application Development Using Versata Business Logic Automation for  
WebSphere  
as created or updated on September 20, 2001.

## Edition, month year

This revision reflects the addition, deletion, or modification of new and changed information described below.

### **New information**

- ▶
- ▶

### **Changed information**

- ▶
- ▶



# Contents

<b>Summary of changes</b> .....	iii
Edition, month year .....	iii
<b>Figures</b> .....	ix
<b>Tables</b> .....	xi
<b>Preface</b> .....	xiii
The team that wrote this redbook .....	xiv
Special notice .....	xiv
IBM trademarks .....	xiv
Comments welcome .....	xiv
<b>Chapter 1. Overview of the Self-Service Pattern - Topology 1</b> .....	17
<b>Chapter 2. Trade Application Overview</b> .....	19
2.1 Trade Application Functionality .....	20
2.1.1 The Trade Client Design Using MVC .....	21
2.1.2 Multiple Runtime Modes .....	22
2.2 Versata - A New Option .....	23
2.3 Details of the Trade EJB Implementation .....	23
2.3.1 Database Schema .....	24
2.3.2 Container Managed EJBs .....	24
2.3.3 Use of Copy Helper Access Beans .....	25
2.3.4 The Basic Business Logic in Trade .....	25
2.4 Potential Enhancements to Trade Business Logic .....	26
<b>Chapter 3. Business Logic Automation Using Rules</b> .....	27
3.1 Design and Runtime Environment .....	30
3.2 Transaction Rules - the Basis for Automated Business Logic .....	31
3.2.1 Example of a Rule .....	31
3.2.2 Characteristics of Rules .....	32
3.2.3 Rules and EJB Domains .....	34
3.2.4 Classification of Declarative Logic (Rules) .....	35
3.3 Business Uses of Rules .....	41
3.4 What the Versata Logic Suite Is Not .....	46
<b>Chapter 4. Architecture of the Versata Logic Server within WebSphere</b> ..	49
4.1 Most Frequent Question - What is it really? .....	50
4.2 Managing the Versata Logic Server in WebSphere .....	51

4.3	Versata Business Objects	52
4.4	Versata Logic Server Classes	54
4.4.1	Persistence as a Layer in the Server MVC	55
4.4.2	Rule-enabled Objects as WebSphere Components	56
4.4.3	ResultSet access and Just-in-time Object instantiation	57
4.5	Look to the Future - EJB 2.0 and JCA	61
4.5.1	EJB 2.0 --- Container Managed Relationships (CMR)	62
4.5.2	EJB 2.0 --- Local Interfaces	63
4.5.3	Java Connector Architecture (JCA)	63
4.5.4	Other J2EE standards used by the Logic Server	64
4.5.5	Recap of the Versata Logic Services	65
	<b>Chapter 5. Beginning Rules-based Development</b>	69
	<b>Chapter 6. Rules to Implement Core Trade Application Transactions</b>	71
	<b>Chapter 7. Automating the Presentation Layer with Versata</b>	73
	<b>Chapter 8. Deploying Business Objects and Client Applications</b>	75
	<b>Chapter 9. Enhancing Business Logic with Rules</b>	77
	<b>Chapter 10. Accessing the Logic Server through non-Versata Clients</b>	79
	<b>Chapter 11. Beyond Topology 1 and Conclusions</b>	81
	<b>Appendix A. Additional material</b>	83
	Locating the Web material	83
	Using the Web material	83
	System requirements for downloading the Web material	84
	How to use the Web material	84
	Using the CD-ROM or diskette	84
	System requirements for using the CD-ROM or diskette	84
	How to use the CD-ROM or diskette	85
	<b>Related publications</b>	87
	IBM Redbooks	87
	Other resources	87
	Referenced Web sites	87
	How to get IBM Redbooks	88
	IBM Redbooks collections	88
	<b>Special notices</b>	89
	<b>Abbreviations and acronyms</b>	91



**Index** ..... 93



# Figures

2-1	Trade User's Home Page	21
2-2	To be inserted	22
2-3	To be inserted	22
2-4	To be inserted	23
2-5	To be inserted	23
2-6	To be inserted	24
2-7	To be inserted	24
2-8	Insert ServerComponents.jpg ?	25
2-9	Insert WebComponents.jpg ?	25
3-1	The rule specifying the value of ACTIVE_HOLDING	32
3-2	Based on GUIDE Business Rule Project	36
3-3	Validation from a cached list of values	42
3-4	Account relationships and referential integrity	44
4-1	Versata Logic Server Components Created by Installation	50
4-2	Versata Logic Server EJBs and Servlet Engine in WebSphere	51
4-3	Firewall and Hot Backup Configuration	52
4-4	Data Objects (left) in the Versata Studio. Account Attributes and rules (right)	53
4-5	Query Object in Trade shows a Profit_Loss attribute calculated at runtime	54
4-6	The MVC Architecture of the Versata Logic Server	55
4-7	Business Object Deployed as Local Classes with EJB faces	57
4-8	Value-based Access Through Versata Client Libraries	59
4-9	Standards used for WebSphere version 3.5	64



# Tables



# Preface

Patterns for e-business are a group of proven, reusable assets that can help speed the process of developing applications. This redbook demonstrates a method of developing and managing the business logic in the "self-service business pattern" (formerly known as the user-to-business pattern). The book describes the process of developing a stock trading application, based on the IBM "Trade" benchmark, using business logic rules to automate the construction and interaction of the transactional (EJB) components. It demonstrates substantially enhancing the business logic of the application through rule changes. Two methods of constructing the presentation layer of the application are examined. The first uses Versata presentation automation techniques. The second adopts the Model-View-Controller (MVC) framework of the existing IBM Trade application. The redbook demonstrates how to use the JSP's, servlets and Java beans of the existing Trade application to interface to the EJB-based business logic and explains the role of the runtime Versata logic services installed into the WebSphere Application Server. This redbook is organized into four parts. Part 1 presents background on Topology One of the IBM self-service pattern, which describes situations where users interact with a business applications to view or update data. Part 2 presents the Trade application as an example of this topology. Part 3 examines constructing and enhancing the Trade application functionality using business logic rules. Part 4 examines porting the existing Trade presentation layer to the new business logic layer. At the conclusion of the redbook the reader should understand how to develop and manage the business logic layer of a WebSphere application using rules-based business logic automation.

## The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at Versata Incorporated and IBM.

**Jackie McAlexander** is a Senior Marketing Manager for Versata Incorporated.

**Wilbert Kho** is a Worldwide WebSphere / Versata Technical Sales Specialist for IBM.

Thanks to the following people for their contributions to this project:

Joe DeCarlo, Manager  
International Technical Support Organization, San Jose Center


## Special notice

This publication is intended to help application developers need to learn how to use Versata Business Logic Automation for WebSphere. The information in this publication is not intended as the specification of any programming interfaces that are provided by Versata ann WebSphere. See the PUBLICATIONS section of the IBM Programming Announcement for WebSphere for more information about what publications are considered to be product documentation.

## IBM trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

e (logo)®  
IBM ®  
Versata

Redbooks  
Redbooks Logo 

## Comments welcome

Your comments are important to us!

We want our IBM Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:

[ibm.com/redbooks](http://ibm.com/redbooks)



- ▶ Send your comments in an Internet note to:  
[redbook@us.ibm.com](mailto:redbook@us.ibm.com)
- ▶ Mail your comments to the address on page ii.





# Overview of the Self-Service Pattern - Topology 1





# 2

## Trade Application Overview

Understanding application topology 1 gives us a starting point for understanding the Trade application, an IBM created benchmark available at [http://www-4.ibm.com/software/webservers/appserver/wpbs\\_download.html](http://www-4.ibm.com/software/webservers/appserver/wpbs_download.html). The business logic in Trade will be automated using rules as a part of the activities for this redbook.

Trade is an end-to-end web application modeled after an online brokerage. Trade leverages J2EE components such as servlets, JSPs, EJBs, and JDBC to provide a set of user services such as login/logout, stock quotes, buy, sell, account details, etc., through a standards based HTTP protocol.

## 2.1 Trade Application Functionality

As illustrated on the Home Page below (Figure 2-1), Trade provides HTML buttons to initiate the following functions:

- ▶ Register a new user to establish an account and profile
- ▶ Login to the application. The user name and password are validated from a database table and a session is created.
- ▶ View the Home Page. Page is displayed with the user's current balance and current market conditions.
- ▶ Review and update an account. Allows the user to view and modify his profile.
- ▶ Obtain security quotes and purchase stock. Each purchase establishes a new portfolio "holding" for the user.
- ▶ View the portfolio. Page is displayed with all of the user's holdings. Holdings may be sold from the portfolio. Holdings must be sold in their entirety (i.e. can't sell 10 shares from a 100 share holding.)
- ▶ Log-off from the Trade application and close the user session



Figure 2-1 Trade User's Home Page

Although Trade was primarily designed to test specific aspects of WebSphere performance, it has a well-designed Model-View-Controller architecture that is very useful as a case study of J2EE programming techniques.

### 2.1.1 The Trade Client Design Using MVC

As described in the Self-service Pattern Redbook [ReferenceHere], the Model-View-Controller (MVC) architecture is a common way of partitioning applications for maximum flexibility, maintainability, and reuse.

In MVC, a Model component represents an application object that implements the application data and business logic. A View component is responsible for formatting the application results and dynamic page construction. The Controller component is responsible for receiving client requests, invoking the appropriate business logic, and based on the results, selecting the appropriate view to be presented to the user.

**Figure 2-2 To be inserted**

In the Trade application, the Controller is implemented as a servlet and several Java classes. The main controller servlet, TradeAppServlet, provides a standard Web interface to Trade users. It maps user input, such as requests to "Buy" or "Update Account", to actions handled by an event handler class, TradeServletAction. When an action is sent to TradeServletAction, it invokes the appropriate method from an implementation class, TradeAction, and matches the output of the action to its specific JSP.

In the Trade client, JSPs provide the Views. The TradeAction class also interfaces to the Model, which, as we shall see below, can be configured in a number of ways. This four-step process, while seeming a little more complex than normal, allows for maximum flexibility when configuring the runtime behavior of the Trade application.

**Figure 2-3 To be inserted**

## 2.1.2 Multiple Runtime Modes

One of the most interesting features of Trade is the ability to choose, through set-up parameters, a runtime "mode" of the application. One mode of operation implements business logic and transactions (the Model in the MVC architecture) in the client-tier using simple Java classes. In this mode, data is accessed directly from the database through JDBC.

Two other modes implement business logic and transactions in the EJB-tier, using a stateless session EJB, TradeBean, to implement most of the business methods. Database access is implemented using an entity bean for each of the database tables. Persistence is managed through the WebSphere EJB container using container managed persistence (CMP). The two EJB modes differ in the way that they access the Trade session and entity EJBs. One mode uses optimized EJB access beans, generated by VisualAge for Java. The other uses direct, remote, EJB access from the client.

The purpose of multiple Trade modes is to compare the performance of JDBC versus both forms of EJB access. Readers can also examine each implementation to judge its relative ease-of-use.



In the application documentation, Trade developers discuss the tradeoffs between JDBC and EJB access. They note that well written JDBC code will generally outperform the EJB equivalent, while JDBC may be significantly more complex to develop and maintain. Most of the JDBC complexity comes from the need to explicitly manage transactions and difficulty with logic re-use when business logic is placed in the client-tier.

In summary, data for the Trade application is kept in a relational database and Trade provides three, distinct, paths to the data:

- ▶ JDBC to database access using direct JDBC calls from the Web tier (no EJBs)
- ▶ EJB to DB access using EJB container managed persistence leveraging IBM VisualAge for Java EJB Access Beans for EJB access.
- ▶ EJB to DB access using direct EJB access with leveraging EJB Access Beans

*Figure 2-4 To be inserted*

## 2.2 Versata - A New Option

The reason for choosing the Trade application as a starting point for this Redbook is the flexibility provided by the MVC architecture. Because of this, we can easily implement a fourth and fifth option -- accessing the database through business objects managed by the Versata Logic Server. As we shall see in Chapter 4, these objects contain business logic designed through declarative rules. The rule-enabled objects are installed into a WebSphere EJB container where they execute using runtime services provided by the Versata Logic Server.

Versata rule-enabled EJBs can be accessed from the Trade client, through their EJB interfaces and through the Versata Logic Server client libraries. The client library option is most closely examined for its potential to bridge the EJB-JDBC gap, encapsulating business logic in the EJB tier while providing higher-speed access from client applications.

*Figure 2-5 To be inserted*

## 2.3 Details of the Trade EJB Implementation

We finish this chapter by examining the Trade EJB implementation in more detail. This will allow us to understand the Versata implementation more easily.

## 2.3.1 Database Schema

The database schema to support Trade is straightforward. There are five database tables.

- ▶ Account - holds the user ID and current balance
- ▶ Profile - provides more information about the user such as address and e-mail
- ▶ Quote - holds stock symbols, and descriptions
- ▶ Holding - holds the result of a Buy operation. A holding row contains the user ID, the stock symbol, quantity purchased and the purchase price of the stock (per share).
- ▶ Registry - holds the user ID, password and status. Authentication and authorization in Trade is done through the application. This is a departure from a normal Topology 1 design that relies on WebSphere to authenticate of users.

*Figure 2-6 To be inserted*

## 2.3.2 Container Managed EJBs

There are five container managed entity EJBs that correspond to each of the Trade database tables. Each EJB implements the get and set methods for its own attributes. In addition, each entity EJB provides a `findByPrimaryKey()` method that allows the database row corresponding to the object to be retrieved by its unique index.

EJBs that need to support more complex queries use "Finder Helper" classes for each potential query structure. An example is the `HoldingBeanFinderHelper` class, which implements methods to find holdings by user id, find the maximum holding index number and so on.

Each of the five entity EJBs have a corresponding Access Bean "wrapper", generated by VisualAge for Java. When accessing an entity EJB directly from the client, these access bean wrappers reduce the complexity of remote access through a simplified client API and improve EJB performance through a number of techniques.

In general, however, entity EJBs are not directly manipulated by the Trade client, even when operating in EJB mode. Instead, operations go through the session bean, `TradeBean`, through its access bean wrapper, the `TradeAccessBean`.

An example of end-to-end processing follows is illustrated below.

*Figure 2-7 To be inserted*

### 2.3.3 Use of Copy Helper Access Beans

Some business functions, however, bypass the TradeBean. Business functions that update several items of client data at the same time need a more efficient way of setting EJB attributes. This is the case, for instance, when a user updates his personal profile.

Profiles are maintained by the doAccountUpdate() method. To optimize doAccountUpdate(), the TradeAction class directly calls the ProfileAccessBean. The ProfileAccessBean is a VisualAge for Java copy helper access bean. This bean provides optimizations designed specifically for updates. It allows individual attributes of an object to be updated in the client tier before committing the entire updated row in a single remote call.

As we can see from the diagrams below, to optimize performance, Trade designers have employed a sophisticated mix of session EJBs, entity EJBs, plus wrapper and helper classes for their business logic. This is in addition to the various JSP, servlets, classes and other components used for the MVC architecture of the client tier.

*Figure 2-8 Insert ServerComponents.jpg ?*

*Figure 2-9 Insert WebComponents.jpg ?*

### 2.3.4 The Basic Business Logic in Trade

Compared to the operations of a real life brokerage application, the business logic in the basic Trade application is minimal. Of course, this is because Trade was designed for performance testing, and not as a fully operational business application.

If we consider that business logic is the data processing that must occur to carry out organizational functions and policies, we can summarize the main, transactional business logic in Trade.

For the buy() method, whose signature is "public double buy(String userID, String symbol, double quantity)"

- ▶ Find the account for the current userID, error check if no account or non-existing account
- ▶ Find the quote price for the stock symbol, error check if no symbol or non-existing symbol
- ▶ Multiply the quote price by the quantity
- ▶ Create a holding object with for the specified user with the amount, stock and quantity

- ▶ Within the same transaction, debit the user's account balance by the amount of the transaction. (The debit function will need to be added to the Account EJB too supplement the default get and set methods.)

For the sell method, the logic is just similar but reversed, although more sophisticated coding may be needed to find the correct holding to sell.

## 2.4 Potential Enhancements to Trade Business Logic

The remainder of this Redbook will examine, not just how the initial business functionality in Trade can be automated using the Versata Logic Server, but examines how the application might be enhanced to meet more real life requirements. Such enhancements will include:

- ▶ Allowing users to sell partial holdings. This will include adding a transaction table to save the buy and sell operations for each holding
- ▶ Controlling "margin selling". Checking balances and margin rules before sell operations
- ▶ Much more extensive checking and validations in the EJB tier. The current Trade application implements most data validation and error checking in the client tier (in the TradeAction class). Placing field length, data type, and other validations in the business logic will allow it to be shared among multiple applications.
- ▶ Adding a flexible commission system that can maintained by business managers.

We will examine the ease of automating this functionality with high level declarative rules.



Chapter 3.

## Business Logic Automation Using Rules

The Versata Logic Suite is a business logic automation engine and design studio that captures business logic rules and executes them as J2EE components running within WebSphere. It is designed to simplify WebSphere application development and to make J2EE systems more general, flexible, understandable, and reusable.

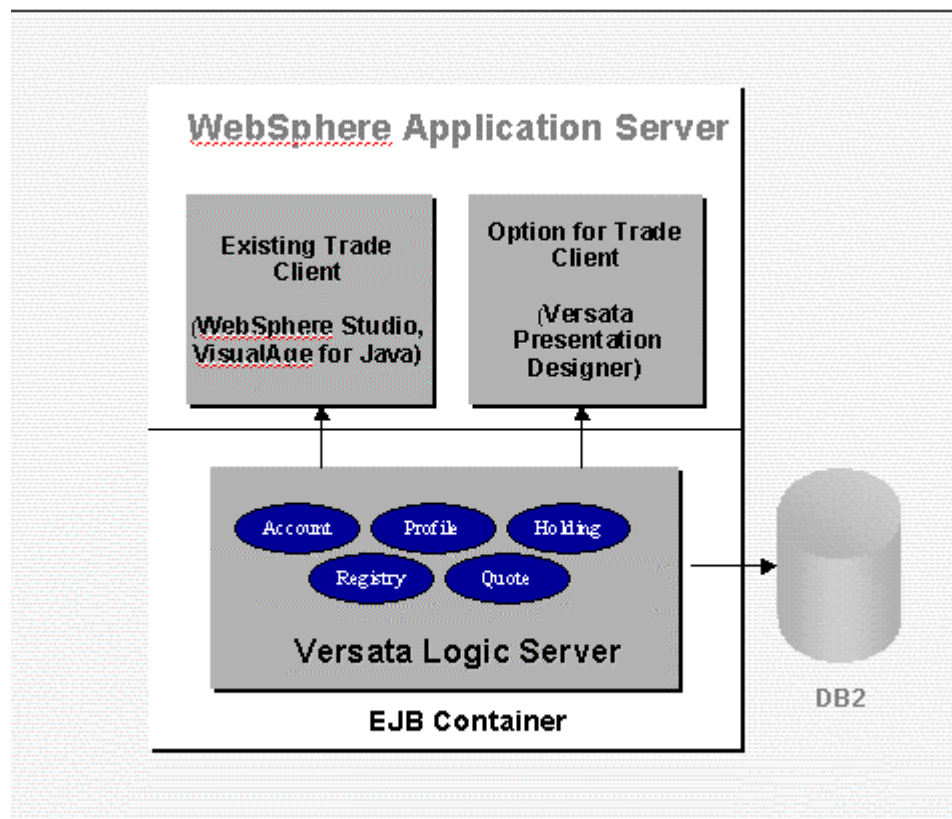
Versata defines the process of creating and executing systems directly from their high level rule specifications as automation. Automation allows the developer to specify “what” the logic should do, not “how” it is to be implemented.

Because Versata rules are easier to understand, write, change and debug than the hand-coded programs they replace, software systems can be built faster and with more consistent quality. In addition, the time and quality benefits should increase over the entire software life cycle, since new requirements can usually be handled by simple rule changes and additions. These changes are applied consistently during re-automation by Versata and typically do not introduce new bugs or errors --- the downfall of most software maintenance efforts.

There are two scenarios for automating WebSphere applications using the Versata Logic Suite.

In the first scenario, the Versata Logic Suite is used to automate the business logic of a WebSphere system and is also used to create the Versata-automated client application. This scenario is examined in Chapter 6, which introduces the Versata Presentation Server (part of the Versata Logic Suite).

In the second scenario, the Versata Logic Suite is used to create the transactional business logic (EJB tier) of a WebSphere application while non-Versata components are integrated to provide the presentation logic (Web tier). This scenario is examined in Chapter 10, where the existing Trade client, built with WebSphere Studio and VisualAge for Java, is integrated with Versata's rule based business logic.



**Use Scenario 1 & 2 - Two options for Client Applications**

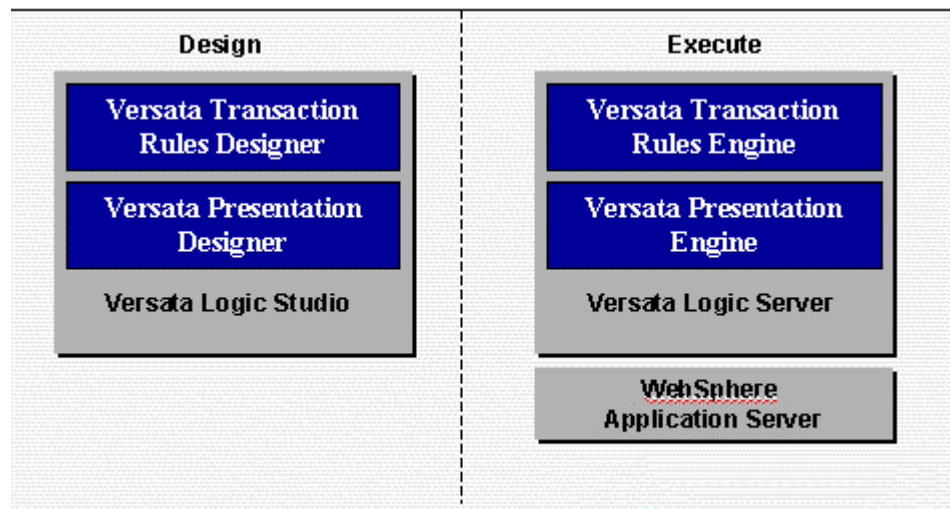
**Note:** In either client scenario, VisualAge for Java can have an important role in a Versata-automated system. For instance to, integrate Versata business logic to enterprise resources, Versata rules can call Visual Age for Java components such as Enterprise Access Beans. This ability allows new, rule-based logic to interact with legacy applications, MQSeries middleware, and other specialized EJB's. In addition, VisualAge for Java can be used as the Java debugging environment for the Versata Logic Server. These, and other potential integration points, are explored in Chapters 10 and 11.

## 3.1 Design and Runtime Environment

The Versata Logic Suite includes a design environment, the Versata Logic Studio, and a runtime environment, the Versata Logic Server.

The Versata Logic Studio includes a transaction rules designer to specify business logic, and an optional presentation designer to specify complete HTML or Java client applications. The Versata Logic Studio runs as a desktop application.

The Versata Logic Server includes a transaction rules engine to execute business logic, and an optional presentation engine to execute client application logic. The Versata Logic Server installs into an EJB container provided by the WebSphere application server. A development copy of the Logic Server is provided with the Versata Logic Studio for testing.



**The Design and Execute Environments**

As noted above, the Versata Transaction Rules Designer and Engine can be used with or without the Versata Presentation Designer and Engine. Both use a high level development approach.

The remainder of this chapter examines some characteristics of Versata rules created in the Transaction Rules Designer.



## 3.2 Transaction Rules - the Basis for Automated Business Logic

In the software industry, the term "rule" has been used to describe several technologies. Within messaging systems such as IBM MQSeries, rules are used to direct the routing of messages. For web personalization, such as that provided by modules in the WebSphere Application Server, rules are used to control the presentation of content. Finally, within Artificial Intelligence (AI)-like inference engines, rules are used to leverage "expert knowledge" to solve a specific business problem.

For clarity, we will distinguish Versata Logic Server rules from other rule technologies.

**Important:** Versata Logic Server rules are high level, unordered assertions about data used to formulate and direct the transactions within a J2EE application server.

Unlike messaging systems, they do actual logic execution. Unlike personalization add-ons, they carry out the core business functions of a WebSphere application. And, unlike inference engines, they are not situated "outside" of the system to provide inferred conclusions.

### 3.2.1 Example of a Rule

Having said that Versata rules are assertions about data, here is an example of an assertion:

*Example 3-1*

---

For an object Account,  
the value of the attribute ActiveHolding  
is  
the number of Holding objects associated with this Account where the  
QuantityOnHand of the Holding is greater than zero.

---

Here is the picture of the rule within the Versata Studio.

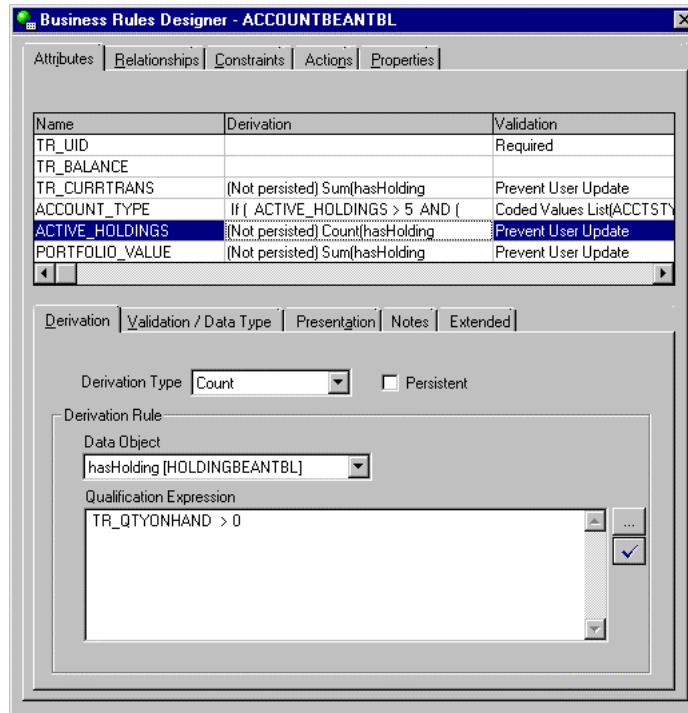


Figure 3-1 The rule specifying the value of ACTIVE\_HOLDING

The transactional effect of this assertion is that, when the QuantityOnHand of a Holding is changed (when buying or selling a stock, for instance), the ActiveHoldings count in the Account object will be automatically examined. If a new Holding is added, or if the shares of a Holding drop to zero, it will be adjusted up or down as needed.

**Note:** The number of ActiveHoldings will be used as part of a personalized commission calculation in the rule-enhanced Trade application

### 3.2.2 Characteristics of Rules

Here are some important points to notice about the rule:

- **It implements logic for multiple objects.** From what we have seen, the rule affects at least two entities -- Account and Holding. In fact, the rule will also involve a third entity - a new Transaction object. Logic in the third object will be automatically created because Holding has a rule to derive its QuantityOnHand from the Transaction entity.

- ▶ **It is declarative.** This means that the designer did not specify how the system is to get this result. The developer did not need to decide whether the "count" function resides in the Holding EJB, or the Account EJB or a third, session EJB. He did not need to implement arrays of Holdings, counters, or cross-object get and set methods.
- ▶ **It is transaction independent.** There are several transactions that affect the on-hand quantity of a stock in a holding. A buy transaction will increase the quantity; A sell transaction will decrease it. In addition, new transactions may be added in the future (to convert options to stock, for instance.) These all may affect the on-hand quantity of a stock. The important thing is that the rule to re-calculate ActiveHoldings will be applied automatically, to any transaction that requires it. No developer analysis is needed to determine when to apply the rule.
- ▶ **It is unordered.** Although a specific sequence of operations will be implemented (Holding will be updated before Account), the developer need not worry about the ordering of operations. A rules compiler in the Versata Logic Server will unravel the dependency between objects and sequence operations correctly.
- ▶ **It will be automatically optimized for runtime.** As we saw in the original Trade application, performance is frequently a concern when utilizing EJB remote interfaces and container managed persistence. The Versata Logic Server provides services within the WebSphere Application Server to overcome many of these concerns.

For instance, the Versata Logic Server maintains a transaction cache for cross-entity interactions inside of WebSphere. This means that the appropriate Transaction, Holding and Account objects will be brought into the Logic Server cache in a single read from the database. The entire rule chain will execute from this cache, speeding execution while guaranteeing data integrity.

In addition, all the interactions between Transactions, Holdings, and Accounts will be done with simple Java method calls, rather than with expensive EJB access. Current EJB containers (those adhering to the EJB 1.1 specification) impose significant overhead on EJB calls, even between beans using the same Virtual Machine. This overhead includes RMI call-by-value semantics, permission checks, and transaction control. To avoid this overhead, the Versata Logic Server implements business object rules in lightweight Java class "helpers" --- one for each entity EJB. Thus, all Versata rule processing will use simple Java method calls, greatly improving the performance of rule processing versus hand-coded EJB transactions.

**Note:** EJB 2.0 compliant servers are expected to adopt this approach to local object access.

Chapter 4 details the runtime architecture of the Versata Logic Server within WebSphere and explains these performance features in more detail.

### 3.2.3 Rules and EJB Domains

From the example above, we can make these additional observations about Versata rules.

The first observation is that rules apply to data, specifically they apply to data when it changes. From this we can see that, in applications where persistent data does not change, this type of rule will not be very useful.

**Note:** This characteristic is one of the main differences between "inference rule engines" and "transactional rule engines". Inference engines collect their data and make decisions independent of transactions. The result of the decision may or may not be used in a transaction. With most inference engines, the transaction will be designed and coded by the developer using traditional coding methods.

Transactional rules engines, on the other hand, drive transaction logic. Automated transactions proceed (or don't proceed) because of rules. These automated transactions may be customized with hand-coded logic, but the bulk of the calculations, validations and evaluations come directly from the executing rule.

The second thing we observe about Versata logic rules is that they govern the interactions of a set of business objects (and their attributes) in a J2EE application. Here we will coin a new phrase - this set can be thought of as a domain of EJBs, where the domain is a collection of entity-type EJBs that will be needed to carry out a series of common business functions.

The specification of this domain usually begins with an object model (or a database schema.). In fact, the Versata Logic Suite, has utilities to import both object models (from Rational Rose, for instance) and schemas (from relational databases). These jump-start the definition of the domain. The remainder of the domain specification is done through rules.

**Note:** For those familiar with software engineering domain analysis, a repository of Versata business objects and rules capture both the commonalities and variabilities of a set of applications to be deployed in the enterprise.

The complete specification for all of the domain objects and their interactions is stored as "metadata". This metadata is kept in an XML-formatted repository and is available, at design time, to the Versata Logic Server and other applications accessing objects in the domain.

What remains then is to classify the types of rules that can control the behavior of a domain of EJBs.

### 3.2.4 Classification of Declarative Logic (Rules)

In the business rules community, there has been considerable analysis of the types of rules needed to specify the logic of business functions. As early as 1995, the GUIDE Business Rule Project established a vocabulary and taxonomy (classification) of rules on which Versata's definitions are largely based.

**Tip:** A paper describing this is available at [http://www.businessrulesgroup.org/first\\_paper/](http://www.businessrulesgroup.org/first_paper/)

**Note:** Because the Project did not tie their vocabulary to a specific technology, we have narrowed their definitions to make them more relevant to Java developers. For instance, a business “term” is narrowed to mean a Java class, entity, or business object (all synonymous) and its fields or attributes (also synonymous).

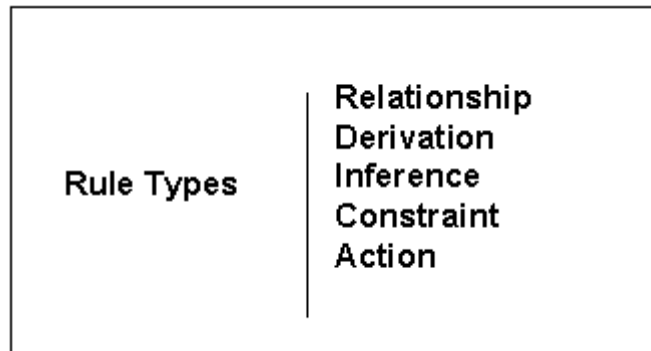


Figure 3-2 Based on GUIDE Business Rule Project

- ▶ Relationship rules specify the association between entities. Rather than controlling transaction behavior itself, they *enable* other rules to be designed and enforced.
- ▶ Derivation rules are algorithms that derive attributes from other attributes. In Versata, derived attributes can be persisted or non-persisted (in which case they represent runtime only values).
- ▶ Inference rules use one or more truths to arrive at a new truth. The new truth usually derives an attribute.
- ▶ Constraint rules specify the policies of a business. They govern under what conditions operations can proceed.
- ▶ Action rules initiate another business event, message, or activity based on some condition.

Here are some examples from the Trade application.

### Relationship Example

- ▶ Relationships typically imply a parent-child association between entities. Through the use of intermediate entities, they can also be used for more

complex relationships. An example of a parent-child association in the Trade application is a rule that says,

**“An Account Has Holdings.”**

As we will see, the Versata Logic Server automates can almost all operations between related entities. For instance, an Account can automatically count and "sum" all of its Holdings. Holdings can automatically check for sufficient funds in the related Account.

Furthermore, as they are entered, Holdings can automatically check to ensure they are associated with valid Accounts. These are only some object behaviors enabled by relationships

### **Derivation Example**

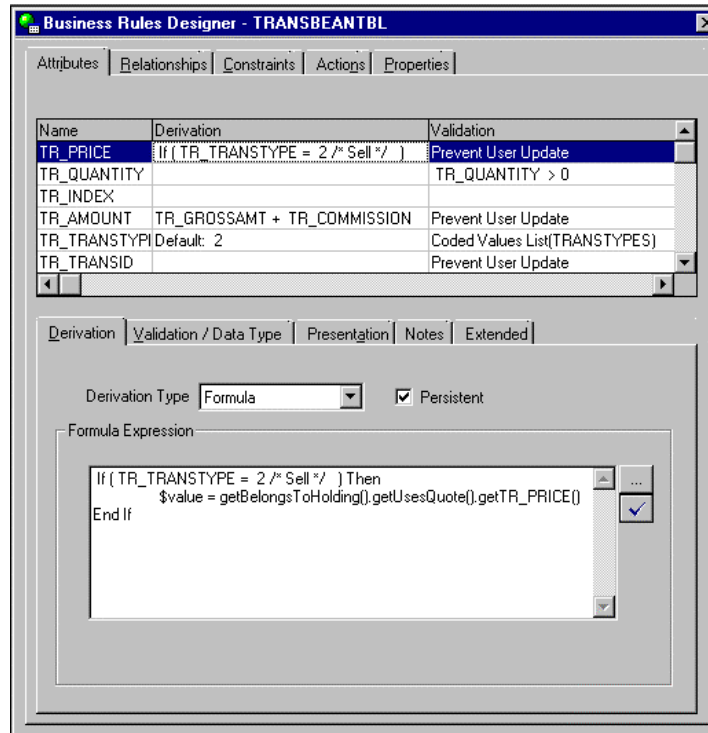
- ▶ Derivations are *computational*, which means that the value of that attribute is arrive at by a formula. An example is a rule that says, for a Holding, the

**"QuantityOnHand = QuantityPurchased - QuantitySold".**

In addition, computational derivations can specify sophisticated qualifications and can navigate to other objects to use their values in computations. An example of this is a rule that says, for a sales Transaction,

**"The Price used to calculate the Amount of the Transaction is the Price (from the Quote Object) of the associated Stock (specified in the Holding Object)"**

From the Versata Logic Studio, the rule looks like this:



## Inference Example

- Inference rules can be thought of as a specialized type of derivation, where an attribute is derived from a truth. An example is a rule that says, for an Account,

**"if Balance is greater than \$500K, then the AccountType is 'wholesale'."**

Like computational derivation, inference derivations can chain together evaluations and calculations from several entities. It is possible, for instance, for a rule to say that, for an Account,

**"An AccountType is "wholesales" if the Account Balance is greater than \$100K [Account entity], AND the number of ActiveHoldings is greater than 20 [Holding entity], AND the average TransactionSize is > \$10K [Transaction Entity]."**



## Constraint Example

- ▶ Constraints define legal states in the system. For transactional systems they define legal values of data that are allowed to exist. A rule to restrict (constraint) margin selling in the Trade application is one such state. It says, on Account

### **"Balance can't be less than zero"**

This simple-sounding constraint rule will automatically included in any rule chain that can potentially affect the Account Balance. For instance, here's a "Buy" rule chain, seen in more detail in Chapter 9, which will be constrained by the margin rule above.

When buying a stock the following happens:

- 1) First the Logic Server begins to create a new Holding for this Symbol, Date and Quantity
- 2) It next finds the stock's price from the associated Quote
- 3) It then begins to insert an initial "Buy" Transaction object
- 4) Next, it calculates the cost of the stock [multiply Price times Quantity]
- 5) Then it finds the Commission [by checking the Account entity to see whether the AccountType is a wholesale or retail account, checking the current CommissionRate for that Account Type and multiplying the CommissionRate times the TransactionAmount]
- 6) Then it calculates the total Transaction Amount
- 7) Next, it begins to Update the Transaction Count for this Holding
- 8) Then it begins to debit the Account Balance with the Transaction Amount
- 9) Finally, it encounters the constraint
- 10) If the constraint is violated (if the new Balance will be less than zero)
- 11) It unwinds the entire operation [rolls back changes to the Holding, Transaction, and Account objects.

This example reinforces three important characteristics about rules.

First, notice that we did not have to write a "Buy" method to perform this chain of events. A simple change [insertion] to a Holding object began the process. The Logic Server understood all of the objects involved. It understood all of the rules and all of the cross-logic between the object attributes. It understood all of the dependencies, including which was the "inner-most" and "outer most" object. It understood how to optimize the chain of events, including caching the Account, Holding, Transaction and Quote objects during the transaction because their attributes were needed several times. Finally, it understood where to put transaction boundaries and how to deploy the transaction into the WebSphere application server.

Next, notice that the rule is not tied to any particular transaction. Instead, it applies to the "state" of the system. The rule would apply also if the user tried to withdraw money from his account - his would be prevented from withdrawing more funds than he had. The Logic Server is responsible for maintaining the declared state and the Logic Server is responsible for identifying all the possible transactions that affect that state.

Finally notice that the declaration of rules was completely unordered. We actually created this rule to constrain margin selling before we even knew how the Balance would be derived. Similarly, declared how Commission was derived before worrying about how the stock price was going to be found.

This is "what" not "how" that Chris Date talks about when categorizing declarative logic rules.

There is another type of constraint rules --- a transition constraint.

- ▶ Transition constraints define legal transitions, or changes from one state to another. An example is a rule that says,

**"Old QuantitySold cannot be more than New QuantitySold"** (In other words, in the Trade application, it is not possible to "un-sell" a stock)

During rules processing, the Logic Server performs maintains old and new values of all attributes. This allows rules to easily check and constrain transitions between object states.

## Action Example

- ▶ The final type of rule are action rules, more completely called an Event/Condition/Action. The *Event* is the operation and entity being watched by the Logic Server. The *Condition* must be met in order to proceed. The *Action* is the side-effect that should of an condition being met: An example of this is a rule that says,

**"When adding a new Transaction, if the TransactionType is "sell", credit the related Holding's related Account Balance".**

The Logic Server will take an action when all other rules leading up this event/condition have evaluated "true".

### 3.3 Business Uses of Rules

It is also helpful to classify the business use of rules. Although there are many more, here are six business uses of transactional rules.

1. Rules can automate data validation. Data and transactions are more accurate. Validation coding is eliminated.
2. Rules can preserve the association between related objects when they change. System consistency is assured. Coding to check and maintain relationships is eliminated.
3. Rules can automatically synchronize related attributes in different objects. Complex transactions are always implemented correctly. Performance is optimized.
4. Rules can enforce operational policies and respond to change when policies change. Logic is defined in well-understood declarations. Policies are enforced uniformly.
5. Rules can identify interesting data. Flagged data can be used for personalization or cross-marketing.
6. Rules can initiate asynchronous events. Events integrate the rule-based system with external applications. Events can also initiate exception processing.

Details and examples of the EJB hand coding code they replace follow.

#### **1. Rules can automate data validation. Data and transactions are more accurate. Validation coding is eliminated.**

A fundamental use of rules in the Versata Logic Server is for data validation. Validation rules are a type of constraint.

Although validation is frequently overlooked when estimating the amount of business logic in an application, it's design and development consumes costly programming resources. Moreover, validation (and related error handling) code is usually sprinkled throughout various client-tier and logic-tier components. This makes logic difficult to re-use and maintain.

The Trade application, for example, checks user input first in the TradeAppServlet, validates the parameters in the TradeAction class, and finally confirms them in the Trade session EJB. If the Trade application were to do more extensive validations, or if the datatypes in the entity EJB's were to change, many components would require maintenance.

Versata rules allow data validations to be attached to objects directly, as part of their metadata. This is illustrated below, where the Account Type (wholesale, retail or new) is validated from a list of values. (To optimize performance, Versata Logic services can cache this list in the Web-tier of the application server and share it among users.)

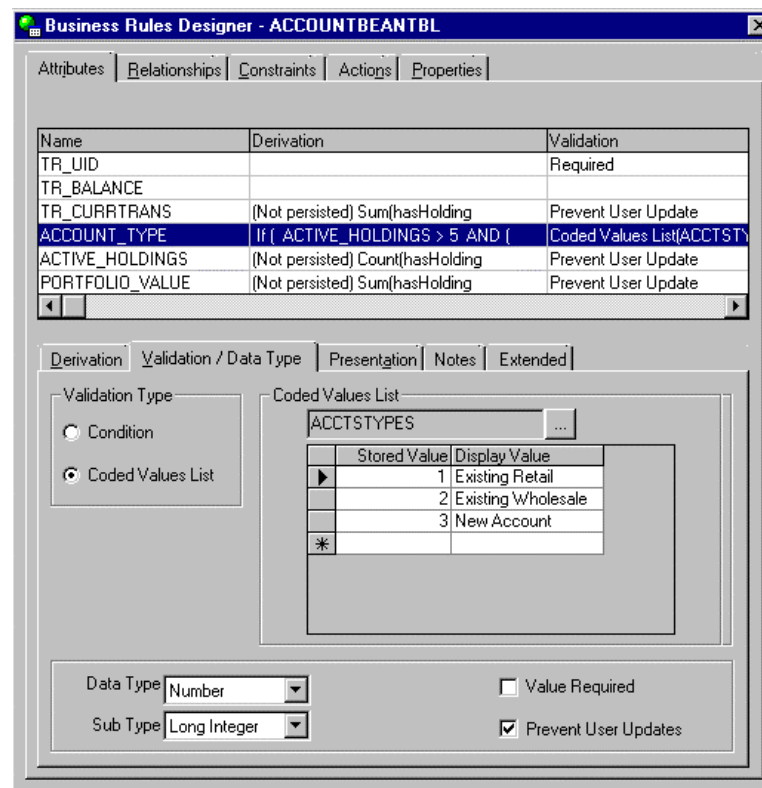


Figure 3-3 Validation from a cached list of values

Other validations for the Account\_Type attribute shown here are:

- ▶ The User must enter a value

- ▶ The value will be translated, using the validation list, to a Java long integer (this is stored in the database).

Using rules to specify attribute validations with rules has several benefits. First, validations, like all rules, will be applied consistently across all applications. This increases system integrity.

In addition, validation metadata, like other rule-based metadata, can be accessed through object methods. This makes it possible for a client components to derive their behavior from business entity metadata. This extends the use of the "Model" in the Model-View-Controller architecture and could allow client components to adapt automatically to changes in business logic.

**2. Rules can preserve the association between related objects when they change. System consistency is assured. Coding to check and maintain relationships is eliminated.**

Referential integrity refers to the need to maintain consistency between one business object and all of the other objects that refer to it. For example, in the Trade application, where there is a relationship between an Account and its Holdings, it would not be good business policy to allow an Account to be deleted if it had active Holdings. Similarly, the business undoubtedly has a policy ensuring that Holdings are created only if the user has a valid Account.

In applications where all data is kept in the same relational database, enforcing referential integrity can be left to the database management system. If, however, there is the potential for data to come from more than one source, integrity must be assured through application code. In J2EE applications, the place for this code is in EJBs (Whether it should be placed in the parent entity EJB or the child entity EJB, or in a third, session EBJ is frequently debated.)

Versata business logic rules provide an easy and consistent way to enforce relationships and integrity, even between objects in different databases or legacy applications.

The screen below shows a Trade referential integrity rule specified in the Versata Studio. Here, the Account entity has two relationships defined: one to the Holding entity and one to the Profile entity. Versata rules specify how the relationship between Account and Holding should be formed (by userID) and specifies the types of referential integrity that will be enforced.

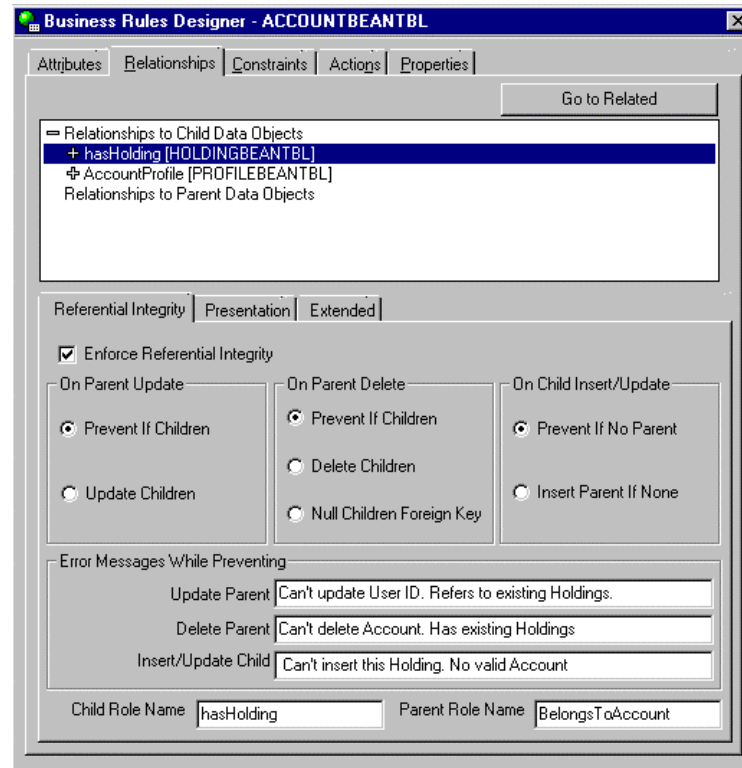


Figure 3-4 Account relationships and referential integrity

Versata rules can enforce very sophisticated integrity. For instance, to perform automatic clean-up, a rule could be define that, when deleting an Account, the user's Profile should be deleted as well.

Enforcing referential integrity in EJB's greatly increases integrity of data entered. Automating this behavior with rules improves the functionality of systems without tedious hand-coding.

**3. Rules can automatically synchronize related attributes in different objects. Complex transactions are always implemented correctly. Performance is optimized.**

One of the most common patterns coded into EJB logic is getting and setting the attributes in related objects within the scope of transactions. The IBM-version of the Trade application "Buy" function is a simple example. Given a UserID, stock Symbol and Quantity, the Buy operation:

- ▶ Finds the related stock from the Quote entity (the developer first defines a find method to do this)
- ▶ Checks that the Quote is valid and gets its price
- ▶ Find the related account for that UserID (the developer first defines a find method to do this)
- ▶ Checks that the account is valid and gets its balance
- ▶ Creates a holding and checks to see that this succeeds
- ▶ Debits the account balance (the developer first defines a debit method to do this)

(The Trade code notes that the logic to check that there are sufficient funds to buy the holding has not been implemented.)

As we see in Chapter 6, rules can be used to automate this transaction. The relationship rule between the Holding and Quote entity allow us to automatically get the price of a stock. The relationship between the Holding and Account allow us to automatically get and update the Account balance. And as we saw earlier, logic to check for sufficient funds can also be implemented with a simple constraint.

One of the most effective uses of Versata Logic Server rules is to automatically implement complex, cross-entity transaction logic. Rules assure that the logic is implemented correctly, and that it can be enhance with simple rule modifications.

#### **4. Rules can enforce operational policies and respond to change when policies change. Logic is defined in well-understood declarations. Policies are enforced uniformly.**

Often, organizations begin to look at rule-based systems when they need comply with government or industry regulations, especially when those regulations change. Constraint rules are useful for this purpose.

In the rule-based implementation of the Trade application, we will add a rule to control margin accounts. For each user, a margin limit will be calculated based on the current SEC regulation for that account type. Buy operations will be permitted or rejected by consulting the current SEC cut-off.

#### **5. Rules can identify interesting data. Flagged data can be used for personalization or cross-marketing.**

Although transactional rules are not primarily used for personalization or data mining, such rules do "watch" transactions as they flow through the J2EE application server.

For example, with a rule on the Account entity, a sales representative can be automatically notified when a new user, with an account balance greater than \$100K, is added to his territory. Or a user may set a flag on his holdings, to notify him when a stock drops more than 25%.

Since rules watch for changing data, any behavior initiated by a change in state can be a candidate for a rule.

**6. Rules can initiate asynchronous events. Events integrate the rule-based system with external applications. Events initiate exception processing.**

Although Trade is an entirely synchronous application (transactions are committed or rolled back immediately), most enterprise systems have some asynchronous operations.

For example, when a user updates his account information, a rule could create an XML formatted message and place it on a message queue to be picked up by the corporate CRM system. Or the system could start a workflow process, and advise an investment representative to contact this customer who had just deposited \$100K in his account.

Rules watch data changes and can initiate synchronous transactions or asynchronous events based on those changes.

## 3.4 What the Versata Logic Suite Is Not

This chapter concludes with a discussion of what transactional rules are not. This may help to clarify when and when not to use transactional rules for WebSphere applications.

As we discussed, the Versata Logic Server is not an inference rules engine. Typically inference (or decision support) engines sit outside of the transactional system. They may be used to build expert systems or produce input to transactional components, but they do not directly implement the transactions contained in components such as EJBs.

The Versata Logic Studio is also not a Case tool. Case tools produce models and code "stubs" which are then implemented and integrated. Versata rules are executable and they need no further development (although they can be customized, as we see in Chapter 9.)



The Versata Logic Suite is not a "4GL". Fourth-generation languages shortcut procedural programming but they are not declarative. A 4GL function will generally map, one-to-one, to a coded procedure. They do not unravel dependencies, sequence complex chains of operations and map to logic across many entities.

Finally, although the Versata Logic Suite does construct Java components, it is much more than a code generator. It uses high level specifications to create and directly execute applications. In each case, a specification (the "what") is input to the Versata Design Studio and is stored as exchangeable XML. From the specification, the system automatically parses, analyzes and creates the desired applications utilizing highly performing enterprise Java frameworks (the "how".)

As we see during the next chapters, which detail the rules-enhanced Trade application, the Versata approach is particularly well-suited to WebSphere applications with substantial business logic, where application requirements are evolving or where project time, costs or EJB development skills may be an issue.



## 4



# Architecture of the Versata Logic Server within WebSphere

In Chapters 2 and 3 we explained that the Versata Logic Server installs into the WebSphere application server to provide runtime services for rule-enabled business logic. This chapter answers common questions about the Logic Server Architecture within WebSphere, its utilization of WebSphere services and the exact nature of the business objects it creates and executes.

**Note:** At the time of this writing the Versata Logic Server was generally available on WebSphere version 3.5. A port was underway to WebSphere version 4.0. Terminology may differ slightly for WebSphere version 4.0

## 4.1 Most Frequent Question - What is it really?

The most frequently asked question about the Versata Logic Server is simply, "what is it really? In terms of J2EE components, what is it comprised of and how does it work?"

The Versata Logic Server is a loose term for a small set of sophisticated EJBs that run inside of the WebSphere Application Server. The EJBs provide libraries of Java classes (services) used by the business objects and applications created in the Versata Studio.

The two primary EJBs are the VLContext, used by the Versata Transaction Rules Engine, and the PLContext, used by the Versata Presentation Engine. When the Logic Server is installed, these stateful session EJBs are deployed automatically into a WebSphere EJB container.

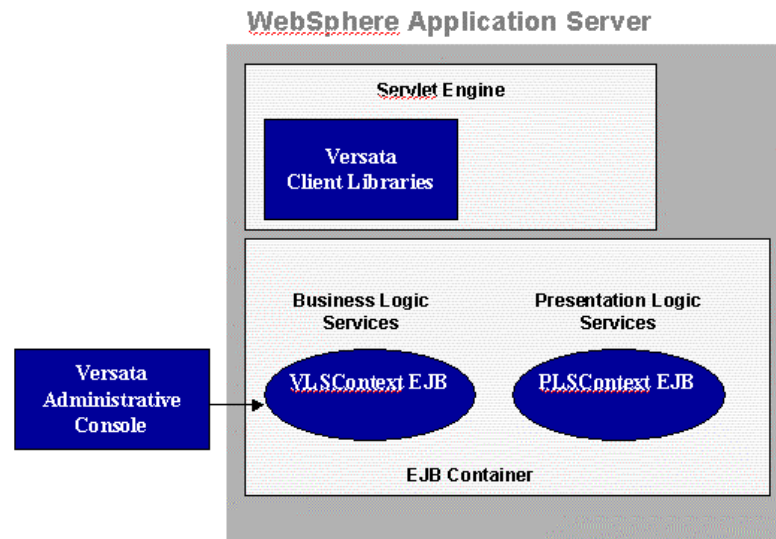


Figure 4-1 Versata Logic Server Components Created by Installation

To assist in managing the collection of Versata resources, an instance of the WebSphere Application Server, called VERSATA, is created to house the EJB container. A servlet engine is also configured to execute Versata application servlets. When a user connects to the Logic Server, a VLContext instance will be created and will be available for the duration of the session. When a user executes a Versata constructed application, a PLContext instance will also be created.

(WSConsole.gif goes here)

*Figure 4-2 Versata Logic Server EJBs and Servlet Engine in WebSphere*

## 4.2 Managing the Versata Logic Server in WebSphere

It is important to note that the Versata Logic Server has been designed to take advantage of the management and scalability strengths of the WebSphere Application Server. Here are some of the management characteristics of the Logic Server running within WebSphere.

- ▶ It is implemented with workload managed session EJBs. This allows multiple Versata Logic Servers to be replicated (cloned) and allows WebSphere to transparently distribute user load between Logic Servers, increasing throughput. Cloned copies of the Logic Server can be placed on the same physical system (vertical scaling), or on multiple distributed systems (horizontal scaling).
- ▶ It can use WebSphere security mechanisms. As we see in the next chapters, the Versata Logic Server controls access to its business objects using role-based authorization. Versata can use any WebSphere authentication mechanism to obtain validated user and role information. In this way, an organization can maintain a single directory of user data. Similarly, the Versata Logic Server can use sign-on information, passed from other applications, to grant access to business objects. This facilitates single user sign-on.
- ▶ It can be configured to provide redundancy. When configured with workload management software and WebSphere servlet redirection, service requests to a failed system can be directed to a Logic Server clone on a backup system. When the WebSphere administrative database and Versata role information is also replicated, this configuration eliminates a single point of failure.
- ▶ It can be configured to run behind a firewall. With WebSphere servlet redirection, the Versata Logic Server can be placed behind a firewall, as shown below.

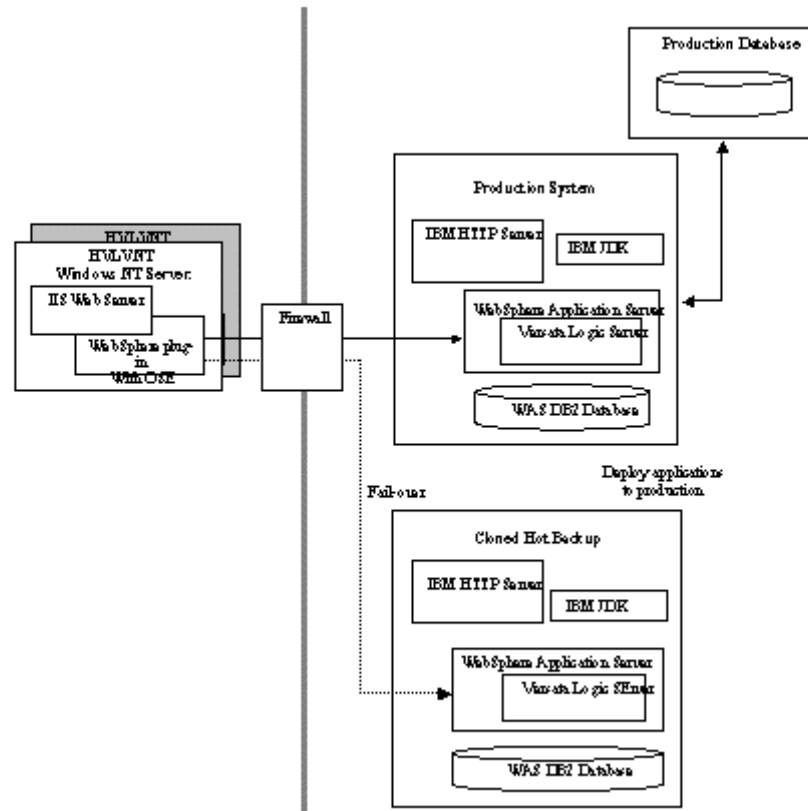


Figure 4-3 Firewall and Hot Backup Configuration

## 4.3 Versata Business Objects

Before beginning development, it is also useful to understand the construction of the business objects managed by the Versata Logic Server. As we explained in Chapter 3, rules attach to data. Specifically, rules attach to business objects and their attributes.

Business objects in Versata are application-independent components that represent data and encapsulate the logic, or rules, need to carry out business processes. There are two types of business objects used in Versata:

- ▶ Data objects map to entities physically persisted to disk. A data object contains the set of attributes (both persistent and derived virtual attributes) to

which rules are attached. Within WebSphere, data objects are deployed as entity EJBs.

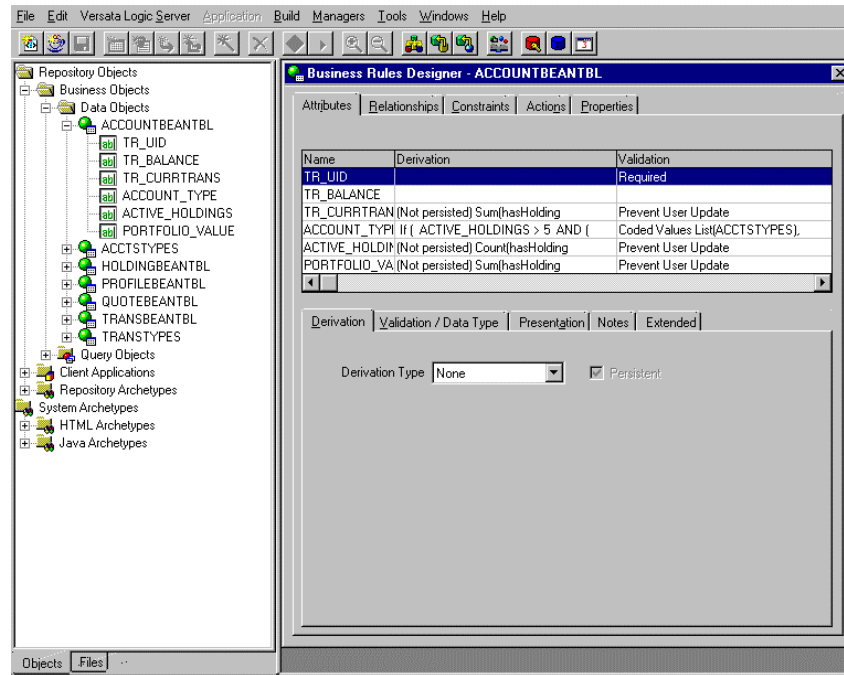


Figure 4-4 Data Objects (left) in the Versata Studio. Account Attributes and rules (right)

- ▶ Query objects represent "views" of joined or restricted data objects. A query object provides an abstracted, reusable view of one or more data objects that protect client components from changes to the underlying data objects. Query objects can also have "virtual" attributes calculated at runtime. Within WebSphere, query objects are deployed as session EJBs and implements the J2EE pattern of aggregate or compound entities.

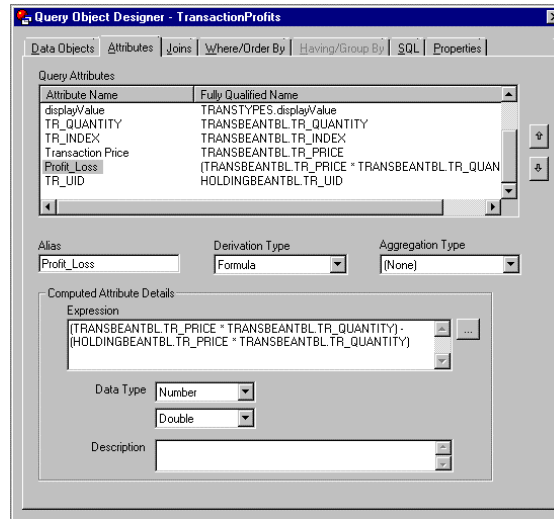


Figure 4-5 Query Object in Trade shows a Profit\_Loss attribute calculated at runtime

## 4.4 Versata Logic Server Classes

The Versata Logic Server class libraries provides both the construction framework and the runtime environment for business objects. Each business object will be created from a subclass of either a Versata DataObject or QueryObject Java class. In addition to normal EJB methods --create(), findByPrimaryKey() and so on --- these objects will inherit extensive methods to:

- ▶ Execute ad-hoc queries
- ▶ Communicate with related objects
- ▶ Listen for and process rule events
- ▶ Manage the transaction cache
- ▶ Form transactions
- ▶ Communicate with the Versata Logic Server persistence layer

The mechanism for business object transactions and persistence is particularly interesting and is explained below.



#### 4.4.1 Persistence as a Layer in the Server MVC

The Versata Logic Server has a layered architecture that can be thought of as its own, server-side "Model-View-Controller". The purpose of the Logic Server's MVC is to abstract business logic [the Controller] from the physical source of data [the Model], and to produce optimized "packets" of serialized data for display by the client-tier [View]. This combination provides performance, extensibility and portability.

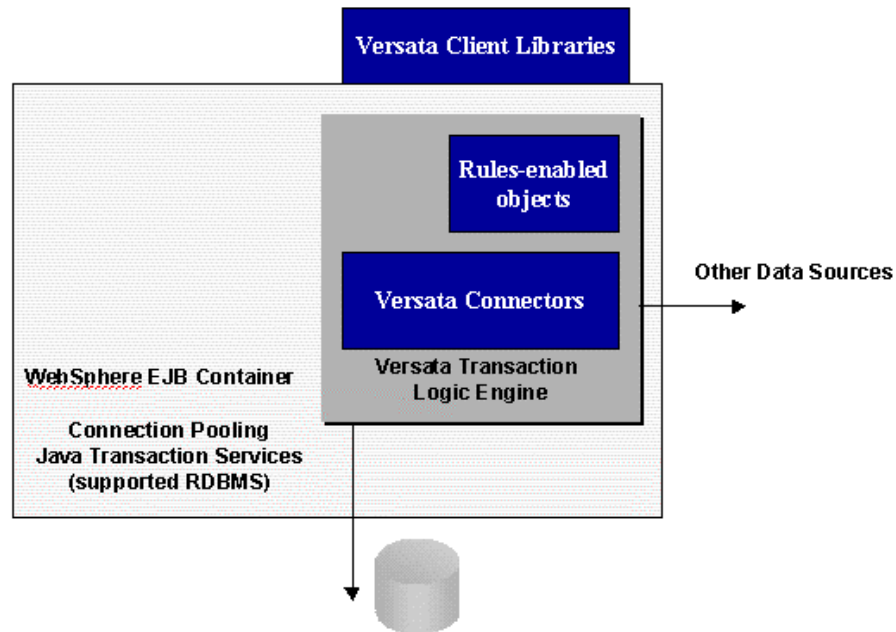


Figure 4-6 The MVC Architecture of the Versata Logic Server

The Logic Server's controller layer consists of the rule-enabled objects, the View layer consists of the Versata Client API's, and the Model layer consists of Versata Connectors. The capabilities of Versata's Client API's and Connectors are explained later in this chapter. The capabilities of rule-enabled objects are explained below.

## 4.4.2 Rule-enabled Objects as WebSphere Components

Up to this point, you might have noticed that we use the terms "object", "entity", and EJB almost inter-changeably when referring to business objects. Here is a clarification on the types of WebSphere components used for rules-based business objects.

As we mentioned, Versata business objects are sub-classed from Java classes. One Java class will be created for each object and will contain its complete rule-defined behavior. These objects are under the management of the VLContext EJB (the business logic service EJB in the Versata Logic Server.)

When the set of business objects is deployed from the Versata Studio, the entire set of Java classes will be deployed (as a JAR) file, and will be added to the CLASSPATH of the VLContext bean. During rule-execution, the Versata Logic Server uses fast, local calls to these class objects.

In addition to being implemented as Java classes, business objects can also be deployed with EJB "faces". Like a common J2EE blueprint from JavaSoft, Versata business object classes and their EJB faces implement an Entity Bean --- Dependent Object pattern. Developers can automatically deploy these EJBs for any object whose methods may need to be referenced from an external component (such as non-Versata EJB.)

**Note:** Object data can be accessed by external components through the client libraries, even if they are not deployed as EJBs.

The choice to create an EJB for an object is made on an object "property sheet" in the Versata Studio. For these objects, Versata constructs both the Home methods to obtain the objects (read from disk, instantiate and return remote handle) and update them.

It is interesting to note that there is no rule or persistence performance penalty for simply deploying objects as EJB's, since the Logic Server will always execute the local implementation of the method, regardless of how it is invoked. There is however, they typical look-up and EJB-instantiation overhead if objects are accessed by their remote interfaces. For this reason, many Versata developers prefer to use the client libraries described below.

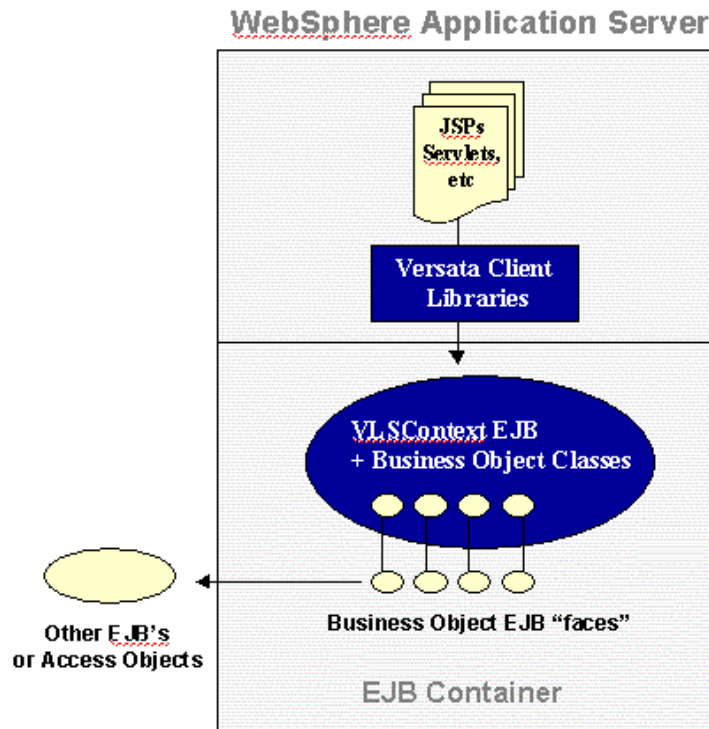


Figure 4-7 Business Object Deployed as Local Classes with EJB faces

### 4.4.3 ResultSet access and Just-in-time Object instantiation

As we diagramed in the MVC discussion, Versata provides another method to access the business objects under its control --- the Versata Client Libraries.

The Versata Client Libraries provide fast, ResultSet-based access to "rows" and "sets" of data objects. As with local rules processing, this avoids the overhead of remote object references. In addition, the Logic Server minimizes the use of shared server resources (memory) providing for "Just-in-time" instantiation of the objects.

#### ResultSets of objects

Versata client libraries access the Versata Logic Server much like JDBC libraries access a database. A client establishes a "session" with the Logic Server, "connects" to a collection of business objects and issues "query" commands on either a Data Object or a Query Object. Commands can be used to retrieve, update, insert and save object changes.

Unlike JDBC, however, the commands and queries supported by the Versata libraries are independent of any database or physical storage. Instead, the libraries communicate to the Logic Server, who manages details of database or legacy data access through the "Connector tier" (explained below.)

There are several performance features built into Versata ResultSets. First, serialized values, instead of objects or handles, are returned to the client. Like the J2EE "ValueObject" pattern, this allows the client to access multiple attributes with a single call, copy the attributes to a local object, and operate on it without remote reference.

Unlike ValueObjects, however, ResultSets can return groups of serialized objects, further reducing overhead. This is similar to the "ValueObjectList" pattern, where the Logic Server acts as a general purpose "ValueObjectAssembler" for all of the business objects in the system. In addition, Versata ResultSets are updatable, extending the benefit of ValueObjects to all data accessed by the client.

To support this functionality, the Versata client library provides an optimized execution framework. The framework retrieves ResultSets into scrollable buffers. The size of the buffer is tunable (10 rows or 50 row buffers, for instance.) Clients can loop through ResultSets using first, last, next and previous methods. In addition, clients can insert rows and modify values in the ResultSet. When ResultSet modifications are complete, the set can be "saved" to send it to the Logic Server for processing. (A tunable optimistic locking mechanism preserves data source concurrency.)

ResultSet access to EJB-tier business objects is designed to provide the efficiency of JDBC with the logic encapsulation and reuse of EJB's. An example of ResultSet access to the Trade EJB's is shown in Chapter 10.

## WebSphere Application Server

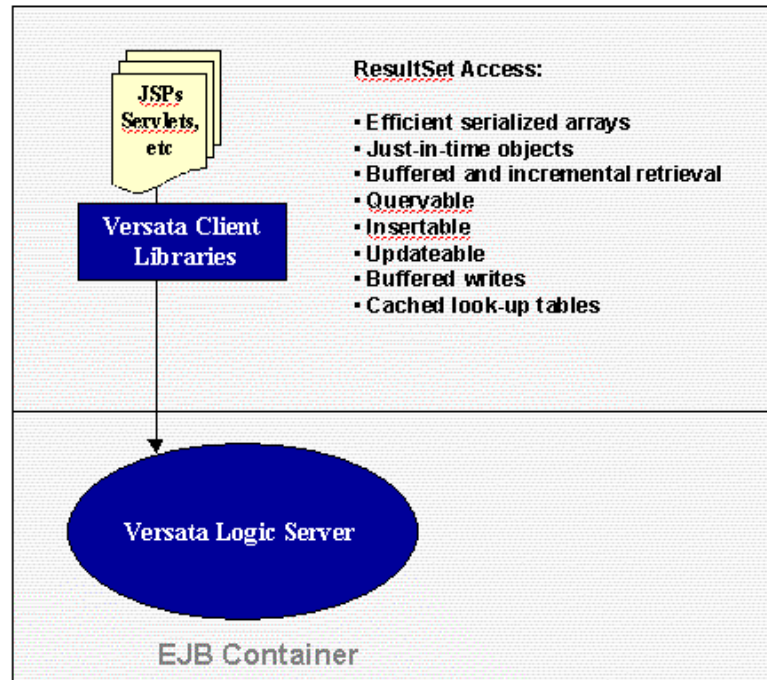


Figure 4-8 Value-based Access Through Versata Client Libraries

### Just-in-time Objects

To conserve system resources, the Versata Logic Server does not create server objects for ResultSets. Instead, ResultSets are retrieved as serialized arrays directly through the Versata Connector.

Business objects in the Logic Server are instantiated at the just-in-time moment that the client completes its changes and issues a "save" method. At that time the Logic Server instantiates objects for all data that will be affected by the change. This technique is similar to the Java Pattern for "lazy object instantiation", although the Logic Server framework for execution is more extensive.

As explained in our discussion of rules, all of the objects changed by the client or changed by a chain of rules triggered by the client operation are brought into a transaction cache to reduce database i/o. Rules ripple through the cache, evaluating and updating data as needed. A single object may be updated several times in the cache. When all rule operations have completed, the objects are written back through the Versata Connector to the data source.

Like sophisticated DBMS, the Logic Server supports implicit single-update transactions or Begin/Commit verbs to bundle several changes into a larger transaction. Even in the case of a single changed row there may be multiple object updates initiated by rules. One of the most productive features of the Logic Server is that such updates are automatically bound into the transaction, thus eliminating transaction bugs.

## The Connector Layer

As mentioned above, Client ResultSets are retrieved directly through the Connector Layer and transactions were written through Connectors. Here is an explanation of the Logic Server persistence mechanism that uses this layer.

One of the design goals of the Logic Server is to decouple rule-constructed transactions from any concern for the data's physical persistence mechanism. Since rules operate on data from relational databases (which may be supported by EJB Container Managed Persistence) and non-relational data sources (which are not currently supported by CMP), persistence must be managed directly by the Logic Server.

Recall that the Logic Server framework is implemented as an EJB. Recall also that business objects, as Java classes, are managed by this EJB. The Logic Server manages the persistence of these Java classes through the VLSTContext bean. It does this by calling the Connector mapped to the business object in the Versata Console. (The Console is an administration interface for a running Logic Server.)

The connector implements object persistence for each class of objects, such as objects persisted to DB2, or to Oracle, or through MQSeries. Where possible, the connector uses the transaction and connection pooling services provided by the EJB container (using the Java Transaction Services API, JTS, from WebSphere, for instance). For data sources not supported by the EJB container, the connector implements its own, similar functionality.

Similarly, where WebSphere supports two-phase-commit protocol (2PC), the connector will also support 2PC. This is done using the JTA ability to assure that a single call is made for starting transactions (rather than using a separate call for each connection, which would introduce the opportunity for errors). This API

requires that the EJB code “register” each connection on behalf of the user thread. This enables the BeginTrans call to communicate with each connection. With the Versata Logic Server, this occurs as database connections from the WebSphere pool are obtained for a specified thread.

As with transaction management, 2PC can occur automatically where the WebSphere application server supports it.

**Note:** For WebSphere 3.5 this includes DB2 and Oracle.

The connector architecture of the Logic Server offers a high-degree of convenience and flexibility. As with container managed persistence, developers do not need to program transaction boundaries or database details. In addition, developers do not need to specify transaction attributes or database details in EJB deployment descriptors. The connector used by a data object can be modified during runtime, without re-deploying the object.

The Versata Logic server is shipped with a number of out-of-the box Connectors, including one for each of the major relational databases. In addition, Versata provides a standard interface definition and a set of transactional API's for use in developing customized Connectors.

## 4.5 Look to the Future - EJB 2.0 and JCA

We will conclude the discussion of persistence and connectors with a look at two upcoming Java standards --- the EJB 2.0 specification and the Java Connection Architecture (JCA). Both proposed standards overlap some of the functionality now provided by the Versata Logic Server.

One of the first things we notice is that the Versata architecture has many things in common with EJB 2.0 and JCA. This is probably not a coincidence, since both Versata and the new J2EE specifications were designed to solve the same problems surrounding distributed, heterogeneous applications. The similarities should make it straightforward for Versata to adapt the Logic Server to use new EJB and JCA functionality. In fact, Versata has announced support for both standards as they become available for WebSphere.

It must be specially noted that when migrating between EJB 1.1 and EJB 2.0, Versata customers may have a big advantage over most Java developers, because rules-based automation abstracts away the implementation of EJB relationships and persistence --- the two big areas addressed in the new specification. This may turn out to be one of the most significant advantages of developing with a high level framework such as Versata.

To leverage new standards in hand coded components, development teams often face fundamental re-writes of their applications. For instance, EJB 2.0 EJBs and EJB 1.1 EJBs may not be mixed in the same container. When using a higher level approach such as Versata's, however, an existing set of business rules can be used to automatically produce a completely new set of integrated components that comply with the changed specification. This transfers the migration burden from the developer to the vendor. It also removes the possibility of introducing new bugs into existing code.

Here we look at EJB 2.0 and JCA to understand their overlap and migration with Versata.

### 4.5.1 EJB 2.0 --- Container Managed Relationships (CMR)

The biggest difference between the EJB 1.1 and 2.0 specification is the significant change to Container Managed Persistence, especially support for Container Managed Relationships (CMR) and faster EJB access (through local interfaces).

Container Managed Relationships allow an EJB container to maintain associations between container-managed entity beans. The relationships are defined in the XML-descriptors of the EJBs and are implemented within the bean with coordinating get and set methods for each logical field. The get method on the "many" side of a one-to-many relationship is implemented with a Java collection and iterated over when traversing the relationship.

The EJB container maintains basic referential integrity. For instance, in the case of an Account with many Holdings, the container can automatically delete the Holdings for a deleted account, and can ensure that a related Account exists before inserting a Holding.

There are some differences between the new CMR and Versata's current relationship rules. For instance, Versata relationships are automatically bi-directional. In addition, there are several more enforcement options: child objects can be automatically changed when parent objects change (their primary and foreign keys will be automatically updated), parent deletions can be prevented (rather than just cascaded), and so on.

The biggest value that Versata may offer when it migrates to the new CMR scheme is in automatically coordinating the data model, the EJB implementation and the deployment descriptors. CMR requires a high-degree of synchronization between the logic implemented in an EJB (names and parameters of abstract methods, for instance), and that placed in deployment descriptors (which must be



ties to EJB logical fields). These must be further coordinated with the exact implementation of the get methods on each side of the relationship which differ depending on whether a single object (one-to-one relationship) or a collection of objects (one-to-many relationship) is being defined.

This degree of synchronization between data modeling, EJB development and sophisticated deployment makes a strong case for automating the production of these artifacts as a result of higher level rules --- the approach taken by Versata.

## 4.5.2 EJB 2.0 --- Local Interfaces

Another significant change in EJB 2.0 is the introduction of local interfaces. Local interfaces allow EJBs within the same JVM to communicate with simple Java calls and pass data by reference, rather than by value. This is designed to address a major performance shortcoming in EJB 1.1 and is almost identical to the approach now used by Versata (although the APIs differ and will be adjusted by Versata).

In EJB 2.0, developers have the choice of either developing local EJB interfaces (which inherit from a new `EJBLocalHome`) or providing remote interfaces (which continue to inherit from `EJBHome`). A developer who wants to optimize local access while also allowing for remote access, will need to create and coordinate two sets of interfaces. Versata simplifies this process by creating local and remote interfaces automatically. In addition, it automatically synchronizes changes in an object's remote interface with changes to the local object.

## 4.5.3 Java Connector Architecture (JCA)

The other area of overlap between the current Versata Logic Server and the up-coming Java specifications is the Java Connector Architecture (JCA). JCA provides a Common Client Interface (CCI) that provides access from J2EE clients, such as enterprise beans, JavaServer Pages, and servlets, to an underlying enterprise information systems.

When implemented through application servers such as WebSphere, JCA will take over part of the role now performed by the Versata Data Access layer. Specifically, it will allow Versata applications to utilize adapters provided by 3rd parties, instead of calling hand-coded Connectors written to Versata's Data Connector API.

Versata is eager to expand connectivity using the JCA and says that future versions of the Logic Server will fully exploit the standard. We do note, however, that the current JCA specification (version 1.0) lacks support for metadata, XML, and asynchronous communication. (The JCA 2.0 draft specification is working to address these.) Until then, Versata support for these features will still be useful.

#### 4.5.4 Other J2EE standards used by the Logic Server

Throughout this chapter we have alluded to a number of other places where J2EE standards apply to the Versata Logic Server. For instance, the WebSphere servlet redirector "finds" a Logic Server (Java Naming and Directory Service - JNDI). Also, the remote servlet "communicates" with the Logic Server (RMI over IOP).

This is a complete picture of the interaction of the Logic Server with WebSphere and other components.

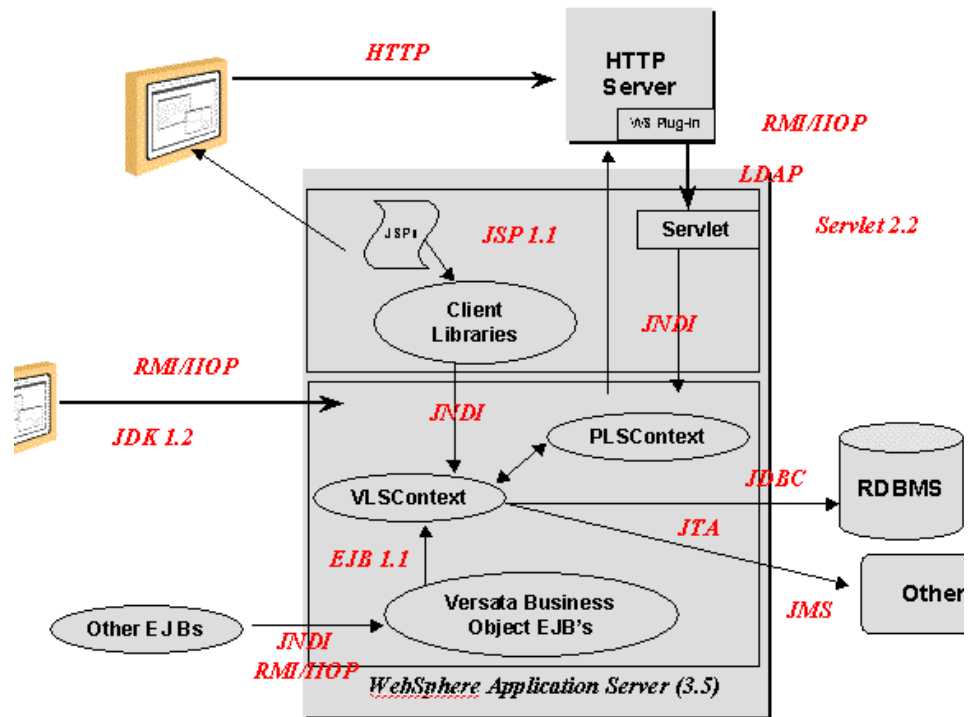


Figure 4-9 Standards used for WebSphere version 3.5

## 4.5.5 Recap of the Versata Logic Services

This chapter concludes with a review of the services provided by the Versata Logic Server, specifically those provided by the Transaction Rules Engine, separate from any specific client tier. They are generally higher level services than those provided by the J2EE application server and can be thought of in the following way:

J2EE services free the developer from routine infrastructure programming to concentrate on business logic, while Versata services free the developer from routine business logic programming to concentrate on just those issues, like connectivity to legacy systems, that are unique to his environment.

Most of these services have been mentioned in the last two chapters. Here we will highlight them before moving on to develop the Trade application.

### Four service objectives

Versata services can be grouped around four objectives:

1. Executing high-level, declarative assertions about a domain of business entities, where those assertions (rules) were defined without regard to sequence or dependencies
2. Providing a framework that allows the developer to customize and extend constructed logic and preserves his extensions through repeated development iterations
3. Enabling fast, convenient access by client applications for all business functions (ad-hoc query as well as object update.)
4. Optimizing the performance and persistence of inter-object logic, regardless of the data source

The Versata Libraries, used by the VLSSContext EJB at runtime, provide most of these services. Detailing the Versata Libraries is beyond the scope of this Redbook, however, we can examine some of the classes for examples of their capabilities.

### For the Logic Server EJB itself:

- ▶ Establish and destroy sessions to the Logic Server, maintain session context

- ▶ Limit number of connections per logic server (before redirecting to another server)
- ▶ Maintain and report statistics on connections
- ▶ Establish and maintain connection pools to data sources
- ▶ Trace and report rules execution
- ▶ Grant fine-grained authorization to business objects, including distinct authorizations to update (vs insert), and to read specific attributes.
- ▶ Manage the transaction cache

### **For Data Objects**

- ▶ Create, destroy data objects
- ▶ Process queries using a datasource-neutral syntax (i.e. masks whether the source is SQL or not). Queries can use any attribute or combination of attributes and may join objects
- ▶ Explicitly get and set any attribute of any object
- ▶ Find related objects
- ▶ Get and set attribute of any related object by relationship navigation (no find method required)
- ▶ Listen to and respond to client or server "save" events
- ▶ Maintain before and after data values for use in rules
- ▶ Listen and respond to rule events
- ▶ Apply defaults, performs calculations, adjust values, check constraint and guarantee referential integrity
- ▶ Listen and respond to user defined events
- ▶ Call external methods
- ▶ Formulate commit and rollback transactions
- ▶ Maintain object metadata (attribute data types, lengths, defaults, formatting, captions, validations, etc.)
- ▶ Provide object metadata to any function that calls for it.

### **For Query Objects**

- ▶ Map attributes to underlying data objects (may be from multiple sources)
- ▶ Derive virtual attributes
- ▶ Overload names, captions and other metadata

- ▶ Coordinate save operations to multiple data objects (manage parent and child data objects)

### **For Connectors**

- ▶ Marshall data from data source to serialized arrays used by the client
- ▶ Translate datasource-neutral commands to the syntax required by the data source
- ▶ Create the database or other connection needed to access the datasource

### **For Client Libraries**

- ▶ Provide ResultSets as arrays to client applications
- ▶ Manage buffers, providing first, next, previous and last behavior
- ▶ Incrementally retrieve rows as buffers are emptied
- ▶ Maintain Old and New values of attributes
- ▶ Provide cached data validation lists to client applications
- ▶ Allow index array access to rows and attributes in the ResultSet
- ▶ Support insert, delete of rows in the ResultSet
- ▶ Access business object metadata (captions, update permissions, formatting, etc.)
- ▶ Save changed ResultSets to the Logic Server





**5**

# **Beginning Rules-based Development**







**6**

# **Rules to Implement Core Trade Application Transactions**





7

# Automating the Presentation Layer with Versata





**8**

# **Deploying Business Objects and Client Applications**





9

# Enhancing Business Logic with Rules







**10**

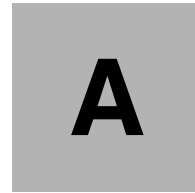
# **Accessing the Logic Server through non-Versata Clients**





# **Beyond Topology 1 and Conclusions**





## Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

### Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG24????>

Alternatively, you can go to the IBM Redbooks Web site at:

[ibm.com/redbooks](http://ibm.com/redbooks)

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG24????.

### Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
?????????.zip	????Zipped Code Samples????

?????????.zip	????Zipped HTML Documents????
?????????.zip	????Zipped Presentations????

## System requirements for downloading the Web material

The following system configuration is recommended:

<b>Hard disk space:</b>	????MB minimum????
<b>Operating System:</b>	????Windows/UNIX????
<b>Processor:</b>	???? or higher????
<b>Memory:</b>	????MB????

## How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

## Using the CD-ROM or diskette

The CD-ROM or diskette that accompanies this redbook contains the following files:

<i>File name</i>	<i>Description</i>
?????????.cpp	????Code Samples????
?????????.html	????HTML Documents????
?????????.prz	????Presentations????

## System requirements for using the CD-ROM or diskette

The following system configuration is recommended for optimal use of the CD-ROM or diskette.

<b>Hard disk space:</b>	????MB minimum????
<b>Operating System:</b>	????Windows/UNIX/S390????
<b>Processor:</b>	???? or higher????
<b>Memory:</b>	????MB????
<b>Other:</b>	????CD-ROM drive????

## How to use the CD-ROM or diskette

You can access the contents of the CD-ROM or diskette by pointing your Web browser at the file `index.html` in the CD-ROM or diskette root directory and following the links found there.

Alternatively, you can create a subdirectory (folder) on your workstation and copy the contents of the CD-ROM or diskette into this folder.





# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

## IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 88.

- ▶ *????full title???????, xxxx-xxxx*
- ▶ *????full title???????, xxxx-xxxx*
- ▶ *????full title???????, xxxx-xxxx*

## Other resources

These publications are also relevant as further information sources:

- ▶ *????full title???????, xxxx-xxxx*
- ▶ *????full title???????, xxxx-xxxx*
- ▶ *????full title???????, xxxx-xxxx*

## Referenced Web sites

These Web sites are also relevant as further information sources:

- ▶ Description1  
<http://?????????.???./???/>
- ▶ Description2  
<http://?????????.???./???/>
- ▶ Description3  
<http://?????????.???./???/>

## How to get IBM Redbooks

Search for additional Redbooks or Redpieces, view, download, or order hardcopy from the Redbooks Web site:

[ibm.com/redbooks](http://ibm.com/redbooks)

Also download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

Redpieces are Redbooks in progress; not all Redbooks become Redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

### IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

# Special notices

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere., The Power To Manage., Anything. Anywhere., TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others

# Abbreviations and acronyms

<b>abbreviation1</b>	Description1 (sort names - highlight rows>Table>Sort> Column1)
<b>abbreviation2</b>	Description2
<b>IBM</b>	International Business Machines Corporation
<b>ITSO</b>	International Technical Support Organization
<b>abbreviation3</b>	Description3
<b>abbreviation4</b>	Description4



# Index

## R

Redbooks Web site 88  
Contact us xiv





To determine the spine width of a book, you divide the paper PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50# smooth which has a PPI of 526. Divided 250 by 526 which equals a spine width of .4752". In this case, you would use the .5" spine. Now select the Spine width for the book and hide the others: Special<Conditional Text>Show/Hide>SpineSize(-->Hide-->Set

Draft Document for Review September 20, 2001 4:46 pm

**6510spine.fm 95**



**Redbooks**

# Application Development Using Business Logic Automation for

(1.5" spine)  
1.5" <-> 1.998"  
789 <-> 1051 pages



**Redbooks**

# Application Development Using Business

(1.0" spine)  
0.875" <-> 1.498"  
460 <-> 788 pages



**Redbooks**

# Application Development Using Business Logic Automation

(0.5" spine)  
0.475" <-> 0.875"  
250 <-> 459 pages



**Redbooks**

# Application Development Using Business Logic Automation for

(0.2" spine)  
0.17" <-> 0.473"  
90 <-> 249 pages

(0.1" spine)  
0.1" <-> 0.169"  
53 <-> 89 pages

To determine the spine width of a book, you divide the paper PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50# smooth which has a PPI of 526. Divided 250 by 526 which equals a spine width of .4752". In this case, you would use the .5" spine. Now select the Spine width for the book and hide the others: Special<Conditional Text>Show/Hide>SpineSize(-->Hide)>Set

Draft Document for Review September 20, 2001 4:46 pm

**6510spine.fm 96**



**Redbooks**

# Application Development Using Business

(2.5" spine)  
2.5"<->mm,n" "  
1315<-> mmn pages



**Redbooks**

# Application Development Using Business

(2.0" spine)  
2.0"<->2,498"  
1052<-> 1314 pages





Draft Document for Review September 20, 2001 4:46 pm

# Application Development Using Versata Business Logic Automation for WebSphere



**Learn options for automating business logic in the EJB-layer**

Patterns for e-business are a group of proven, reusable assets that can help speed the process of developing applications. This redbook demonstrates a method of developing and managing the business logic in the "self-service business pattern" (formerly known as the user-to-business pattern).

**Explore declarative logic design using rules**

The book describes the process of developing a stock trading application, based on the IBM "Trade" benchmark, using business logic rules to automate the construction and interaction of the transactional (EJB) components. It demonstrates substantially enhancing the business logic of the application through rule changes.

**Understand Versata Logic services in WebSphere**

Two methods of constructing the presentation layer of the application are examined. The first uses Versata presentation automation techniques. The second adopts the Model-View-Controller (MVC) framework of the existing IBM Trade application.

The redbook demonstrates how to use the JSP's, servlets and Java beans of the existing Trade application to interface to the EJB-based business logic and explains the role of the runtime Versata logic services installed into the WebSphere Application Server.

**INTERNATIONAL  
TECHNICAL  
SUPPORT  
ORGANIZATION**

**BUILDING TECHNICAL  
INFORMATION BASED ON  
PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:  
[ibm.com/redbooks](http://ibm.com/redbooks)**

SG24-6510-00

ISBN