Integrating Existing Mainframe Applications with New Technologies

Includes CD-ROM

Designing & Programming

CICS Applications

John Horswill & Members of the CICS Development Team at IBM Hursley

O'REILLY®

O'REILLY®

Designing & Programming CICS Applications

Designing & Programming CICS Applications introduces new users of IBM's mainframe (OS/390) to CICS features. Experienced users will learn how to integrate existing mainframe systems with newer technologies, including the Web, CORBA, Java, CICS clients, and Visual Basic, as well as how to link MQSeries and CICS.

Each part of *Designing & Programming CICS Applications* addresses the design requirements for specific components and gives a step-by-step approach to developing a sample application. This book reviews the basic concepts of a business application and the way CICS meets these requirements. It then covers a wide range of application development technologies, including VisualAge for Java, WebSphere Studio, and Visual Basic. Users will learn not only how to design and write their programs, but also how to deploy their application.

Designing & Programming CICS Applications shows how to:

- Develop and modify existing COBOL applications
- Become familiar with the CICS Java environment and write a simple Java wrapper for a COBOL application
- Develop a web frontend using servlets, JSP, and JavaBeans
- Link the web frontend to an existing COBOL application using CORBA
- Write a Visual Basic application to develop a customer GUI
- Link an existing COBOL application using a CICS Client ECI call
- Develop a Java application using Swing as an MQSeries Client
- Use the MQSeries-CICS bridge to access an existing COBOL application
- Debug CICS applications

However you are connecting with your customers, partners, and suppliers over the Web—by intranet, extranet, or Internet—*Designing & Programming CICS Applications* can give you the power to harness the strength of your existing applications quickly and easily.

oreilly.com

US $44.95 **CAN $65.95**

ISBN: 978-1-565-92676-9

9 781565 926769 54495

IBM

RepKover™

Designing and Programming CICS Applications

Designing and Programming CICS Applications

John Horswill and Members of the CICS Development Team at IBM Hursley

O'REILLY®
Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo

Designing and Programming CICS Applications
by John Horswill and Members of the CICS Development Team at IBM Hursley

Published by O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472.

Editor: Sue Miller

Production Editor: Maureen Dempsey

Cover Designer: Hanna Dyer

Printing History:

July 2000: First Edition.

Library of Congress Cataloging-in-Publication Data

Horswill, John.
Designing and programming CICS applications/John Horswill and members of the CICS Development Team and IBM Hursley. p. cm.
ISBN 1-56592-676-5
1. Application software—development 2. CICS (Computer system) I. IBM United Kingdom. CICS Development Team. II. Title.
QA76.76.D47 H69 2000
005.4'3—dc21 00-056535

ISBN: 1-56592-676-5 [10/00]
[M]

Table of Contents

Preface

This book explains how to write applications for CICS—the world's favorite transaction processing system. Customer Information Control System (CICS) systems have been running for more than 30 years and CICS has changed dramatically during that time, from being a basic transaction processing monitor to being an advanced distributed application server.

Throughout its evolution, CICS has preserved and enhanced its support for developing and running a very large application workload. This book teaches you the traditional CICS skills and techniques that have delivered results for over 30 years that are still just as relevant today for building high-speed transaction processing applications. It also teaches the modern CICS skills and techniques that exploit today's advanced technologies—Java™, Web, MQSeries, workstation tools—technologies that modern businesses need to exploit in order to stay competitive.

The Book's Audience

This book is for new and experienced CICS application developers; whether you're an undergraduate, a new employee, or an experienced CICS developer who wishes to update your skills, this book is aimed at you. Chapter 1, *Introduction*, should be particularly useful to business managers who need to know how they can use CICS to add to, and improve, their existing business systems. Chapter 2, *Designing Business Applications*, should be read by system architects, designers and programmers. It explains how to design the architecture for a modern transactional application, with a particular emphasis on the use of CICS. Chapter 3, *Introducing the Sample Application*, discusses the components of a business application, and how you should approach the development of these components.

From Chapter 4, *Designing the Business Logic*, onwards, we assume you're an application programmer needing to develop CICS applications. These chapters teach specific CICS skills such as developing programs in COBOL (the business logic), CICS Java, and Visual Basic, or integrating MQSeries with CICS. They contain guidance about designing, coding, and running the components of a typical CICS sample application.

We point you to various books in the CICS library that fill in any gaps because, in a book this size, we won't be able to tell you *all* about CICS. We discuss, and base our examples on, a subset of the full CICS facilities. This makes things easier for you because it means we won't have to keep referring you to other books in the CICS library while you're learning. These other books are listed in the bibliography, and are shown in the library diagram for your particular release of CICS. The subset of CICS commands we've chosen gives you a sound framework for your first application program and offers a logical starting point for more advanced work.

The main purpose of this book is to provide a friendly, straightforward, and modern approach to the writing of CICS application programs. It follows the development of a sample application, and at the end of each part you should be able to generate the relevant code and run the application.

Organization of the Book

The book describes a COBOL application that creates, reads, updates, and deletes records from a database with and without a locking mechanism. In addition, the COBOL program includes modules that browse, capture errors, and use the CICS Basic Mapping System (BMS) for data input and output to a traditional green screen. There are five additional parts that describe how you can use CICS to access your core COBOL application:

- Through a CICS Java application
- Through a web-based application using a CORBA
- By using the CICS Basic Mapping System for data input and output
- By using Visual Basic to design and implement a CICS Client application
- By integrating MQSeries with your CICS application

Each part describes the design of the component and how to write the code to implement the design.

Having written your application, you are guided through a step-by-step process to deploy your application into a CICS system. There is also advice on how to deal with the issues arising from large-scale deployment. Finally, there is a chapter describing some of the debugging facilities available in CICS.

About the CD-ROM

The CD-ROM accompanying this book contains the source code of the sample application that is discussed in this book. This can save you a lot of time by not having to enter the code. Some of the code has been compiled for use with CICS Transaction Server Version 1.3. If you want to use it with other CICS releases, you will have to re-compile the source code. Appendix A describes how to transfer the code from the CD-ROM to your OS/390 system, to install the files and programs, and configure your CICS region so that you can run the application. The remaining components of the application access and use the COBOL programs you install on your mainframe.

In addition to the sample code, the CD-ROM contains the entire CICS Transaction Server Version 1.3 library in Portable Document Format (PDF) format. Together with this is a copy of the Adobe Acrobat reader. Other software includes the Java Development Kit (JDK) Version 1.1.8.

Refer to Appendix B for detailed descriptions of the contents of the CD-ROM. The README files contains important information about running the sample application.

We've also made the code sample available on the O'Reilly web site:

http://www.oreilly.com/catalog/cics

Conventions in This Book

Throughout this book, we've used the following conventions:

Bold
: Indicates the code you need to edit within code examples.

UPPERCASE ITALIC
: Indicates CICS-supplied transactions, the Application Programming Interface (API) commands, and their command options.

Italics
: Indicates CICS command utilities, filenames, menu options, variable names, display text, examples and in-text references to syntax models. For example, if a procedure asks you to type *filename,* you must type the actual name of the file. Italics also indicates menu options as well as the first occurrence of a new term.

Indicates a tip, suggestion, or general note, For example, we'll tell you about some shortcuts or if an operation requires certain privileges.

Indicates a warning or caution. For example, we'll tell you if you need to check your site's procedures before carrying out a particular action.

How to Contact Us

We have tested and verified the information in this book to the best of our ability, but you may find that features have changed (or even that we have made mistakes!). Please let us know about any errors you find, as well as your suggestions for future editions, by writing to:

O'Reilly & Associates, Inc.
101 Morris Street
Sebastopol, CA 95472
(800) 998-9938 (in the U.S. or Canada)
(707) 829-0515 (international/local)
(707) 829-0104 (fax)

You can also send us messages electronically. To be put on the mailing list or request a catalog, send email to:

info@oreilly.com

To ask technical questions or comment on the book, send email to:

bookquestions@oreilly.com

We have a web site for the book, where we'll list examples, errata and any plans for future editions. You can access this page at:

http://www.oreilly.com/catalog/cics

For more information about this book and others, see the O'Reilly web site:

http://www.oreilly.com

Acknowledgments

This book is the product of the combined efforts of many individuals. A lot of the initial work was done by Ian McCallion and Bernard Swords. Phil Appleby developed the organization and structure that we have in the book today. Our thanks go to all three.

Part I was written largely by Ian McCallion but many people had a hand in its organization.

Andy Krasun and Peter Missen reviewed the book extensively. Our thanks to them for pointing out the inconsistencies and adding valuable details to the text. Andy, in particular, was able to add a lot of valuable information based on his extensive experience working with customers over many years.

The COBOL code on which Part II and Part V are based was developed by Jerry Ozaniec. Becca Dunleavey, Joanne Hodges and others revised and improved the application.

Part III and Part IV were written by Rob Breeds, who developed the application. He also spent a lot of time very patiently explaining things to Phil Appleby and myself. There must have been times when he despaired.

Part VI and Part VII were written by Mike Moynihan and Steve Young. Mike developed the Visual Basic component and persevered with the application when lesser mortals might have given in. Steve wrote the Java code for the MQSeries part of the book and helped us put the CD-ROM together.

Part VIII, written by Janet Righton, whose experience of debugging CICS programs is second to none!

Joyce Cousins spent a great deal of time ensuring that we had a mainframe application that worked. She also spent time with our graduates ensuring their tests worked.

Norman Bell has also been very helpful in ironing out the wrinkles in the application, and he helped us gain a much clearer understanding of the way that CICS works in an OS/390 environment.

Bob Yelavich always responded with copious comments, and gave us much valuable insight from his wealth of experience. We appreciate his commitment and support.

Finally, I have to thank the people from O'Reilly, including Frank Willison, who supported the original idea of producing this book, and Robert Denn, who followed through with the contract. Our thanks to our editor, Sue Miller, who kept us on the straight and narrow when we all wondered if this project would ever see

the light of day. Our thanks to Steven Abrams for his patience in guiding me through the tools and formats and juggling the files and managing the external review. Our thanks go to Rob Romano for his work on the illustrations and to Maureen Dempsey for her role as production editor.

There are many others who have spent a lot of time reviewing and providing invaluable comments on this book, and I hope that I haven't omitted anyone.

I

Introduction to CICS

You may not be aware of it, but hardly a day goes by when something that you do has not involved a CICS application somewhere in the world—whether it is a trip to the supermarket, taking money from your bank account, having a package delivered to your house, managing your company's accounts, stock control or personnel records—CICS is involved. CICS is also involved in many manufacturing plants, providing feedback about the production processes and stock levels, and it may even be linked to suppliers so that stocks can be replenished when necessary. In short, CICS is likely to have played a part in much of the underlying software (often called middleware) that underpins all types of industry applications.

Part I looks at how CICS can help in the world of business applications. It contains the following chapters:

- Chapter 1, *Introduction*, describes the essentials of a business application, and the benefits of using CICS with business applications.
- Chapter 2, *Designing Business Applications*, looks at the key design elements in a business application, and the CICS facilities that support the application designer.
- Chapter 3, *Introducing the Sample Application*, describes the planning of a CICS application that uses existing COBOL business logic and a variety of presentation logic including Dynamic HyperText Markup Language (DHTML), CICS Java (JCICS), a Visual Basic front end using a CICS client, a Java frontend integrating MQSeries with CICS, as well as a traditional 3270 frontend.

In this chapter:

- *The Essentials of a Business Application*
- *Business Applications as Creators of Value*
- *Using CICS for Business Applications*

1

Introduction

Computer systems are used for many different purposes in business today. These range from keeping personal to-do lists to developing business-critical applications in banks. Applications are often categorized by their purpose. For example:

Personal productivity and groupware
: The use of Personal Computers (PCs) for word processing, electronic mail (email), and document sharing using a Local Area Network (LAN).

Design and development
: Computer-aided design and software development.

Manufacturing and production
: Monitoring and control applications in factories.

Business intelligence
: Data warehousing applications used to aid decision-making, arising from powerful, large-scale databases.

Business operations
: Business operations applications (sometimes called *line of business applications*) that "transact the business" of a company—in other words, they perform business transactions on behalf of the company. This is not limited to cash-for-goods transactions. It can include any buyer/supplier transaction that can be translated into a digital format, as well as internal business processes dealing with company resources. For example:

  — Credit card transactions

  — Cash transactions from a bank's Automatic Teller Machine (ATM) or supermarket cash dispenser

  — Stock market transactions for a stock exchange or brokerage

— Information transactions for collecting, collating, and distributing news—such as the results and medal tables for the Olympic Games

— Payroll transactions (essential for the smooth operation of any corporation)

— Logistics transactions, such as the scheduling of vehicles in a transportation company

— Voice application transactions ("Press 1 to enter your meter reading....") for a computer integrated telephony system

— Sales transactions for companies doing business through the Internet

Business applications are crucial to many large and medium-sized companies. For such companies, doing business without these applications would be unthinkable; a bank that lost its computerized account records would cease trading. Many, if not most, of the largest business applications around the world run on CICS.

The Essentials of a Business Application

Even though the computer is at its center, a business application is focused on people. It is a human system as much as a computer system. The purpose of a business application is to keep accurate, up-to-date, and secure operational business information and deliver it rapidly to the users of the application. There are a number of key features that any business application needs. They have to be fast, accurate, secure, and auditable. In addition, the information has to be up to date and available to multiple users across a company, its suppliers, customers, and business partners. A model of the relationship between computers and people is shown in Figure 1-1.

Division of Responsibilities

Accuracy (in the sense of adhering to the intent of the business) depends on the computer system being controlled appropriately; that is, having clear lines of responsibility and division of responsibilities. It is essential to have organizational responsibilities that the system itself monitors and enforces. To this end, business applications broadly separate system development, system operations, and system use with checks and balances. These may be as official as a system audit. There are, of course, many subdivisions of these roles; for example, system development may be subdivided into architecture, design, programming, and testing.

Division of responsibility ensures that different groups of people involved with an application are unable to take advantage of their situations. Consider a payroll application, it should be impossible for payroll clerks to update their own salaries without being monitored; programmers would have built into the program an

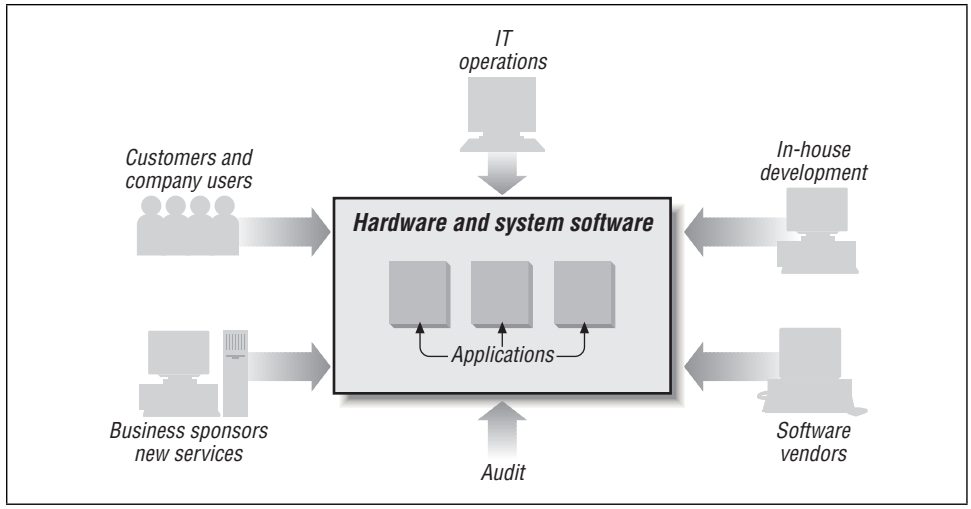

Figure 1-1. Business applications as people systems

audit log that is checked by the audit department. Similarly, the program should include a log of all software updates to ensure that system administrators are unable to make fraudulent changes to the program without trace.

Lifecycle Requirements

Business applications define the business rules that control the delivery and update of critical data; therefore, they require support throughout their lifecycle. The stages of a lifecycle include:

Design
: Design user interfaces that meet users' needs; design for growth and extension; design to build complex applications with many features and capabilities.

Develop
: Develop efficiently, using modern tools and techniques.

Test
: Test thoroughly and efficiently to find problems and track down causes of problems.

Update
: Update in such a way as not to disrupt the existing version when upgrading to a new version.

Technical Requirements

In addition to lifecycle requirements, business applications must also meet a set of general technical requirements:

Accessibility
: The application can be used from any appropriate place on the network.

Availability
: The application is available for use by authorized persons at all designated times; it does not need to be shut down for routine maintenance and can be upgraded without interruption.

Communication
: Rapid communication is possible between distributed parts of the application.

Manageability
: Systems administrators can monitor the application to detect problems, and can take corrective action *before* users complain.

Prioritized use of the hardware
: A management capability should be in place to determine how much the machines are used, so that the workload can be distributed evenly.

Rapid response
: The response time for end users is appropriate (which usually means short!).

Reliability
: The application is not expected to fail, but if it does, it provides diagnostic information to help identify the cause of the failure.

Recoverability
: The application restarts quickly after a failure, without loss of information or of data integrity.

Scaleability
: The application can support as many users as needed without slowing down excessively or requiring excessive resources.

Security
: The application includes the ability to control who can use it, and which actions the users can perform.

CICS was originally seen as a transaction processing system. Indeed behind the scenes this is a lot of what it is doing. But, like a lot of middleware, CICS comes to life by virtue of the many applications and operating systems that it supports. It not only provides an extensive Application Programming Interface (API), but it also controls the resources behind the applications; for example, security, databases, files, programs, transactions and so on, that the applications use. Hence, as

CICS has evolved, describing it as an application server gives a much truer picture of its role today. In Figure 1-1 we see a loose arrangement of applications, which largely function independently of each other. Figure 1-2, on the other hand, draws those applications together so that there can be, for example, shared resources distributed across a computer network. To support the division of responsibilities, lifecycle requirements, and general technical requirements, an application server is required to manage the business applications. This is where CICS fits in. IBM's product CICS is an application server.

If you have key applications that run 24 hours a day for 7 days a week and if your business requires that applications can be recovered completely after failure, you have good reason to move to CICS. If your business already uses CICS, extending your CICS system provides an integrated solution for your ever-increasing business requirements.

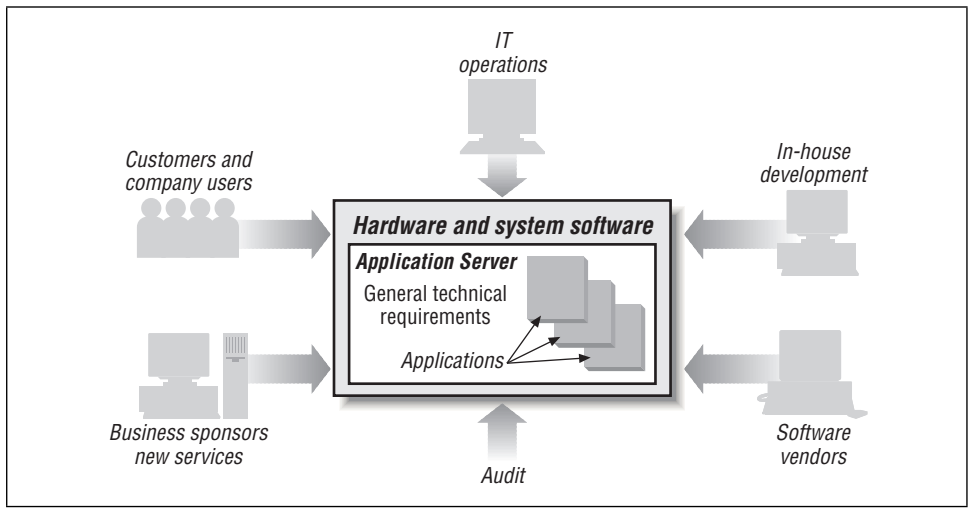

Figure 1-2. Application server supporting business applications

Business Applications as Creators of Value

Historically, companies adopted and became dependent upon business applications because of the reduced costs, improved accuracy, and timeliness of information achieved by transferring key operational data for their business onto computers. Today's business applications must enable rapid development of modern services and must be able to exploit new hardware and software technology for competitive advantage. The latest technology is, of course, the Internet.

Responsibility for developing and managing business applications has changed from being a separate business function to a central part of competitive strategy—from merely boosting operational efficiency to profoundly changing the nature of products, services, and business processes.

The ability to adapt and extend applications has become increasingly important when launching new products and services for maximum advantage. This applies when you're bringing your own ideas to market quickly, and bettering your competitor's offerings. Companies that are today maximizing the potential of Information Technology (IT) to create value are taking radical approaches to developing the systems necessary. The key features are:

- Use of cross-functional teams having responsibility to the business—especially between central and departmental IT groups—because command and control management needs to include all interested parties.
- Maximizing the amount of information held by IT networks.
- Maximizing connectivity to provide information where it is needed.
- Selective use of contracted skills—for example, in web design—rather than attempting to maintain in-house skills across the entire range of technologies.

At the heart of all successful implementations of this approach are the business applications that have been running the business for years—but expanded with more data, applications, processing power and connectivity, and augmented with technologies such as web servers and computer telephony integration. In Figure 1-2, we emphasized how CICS as an application server draws together the applications and resources of a computer system. Figure 1-3 shows application servers have to interact with other systems, both software, for example, web servers and firewalls, and hardware, for example, telephony. Interconnectivity between operating systems and hardware is critical. As a result, a modern business application looks something like that shown in Figure 1-3.

Using CICS for Business Applications

This book shows you how the CICS environment enables you to build a business application consisting of a varied set of components. By satisfying the essentials of a modern business application, CICS provides solutions for your business that improve efficiency, competitiveness, and productivity. Additionally, CICS can help your business implement an e-business strategy—competing in a global marketplace for worldwide customers who find you and trade with you electronically using the World Wide Web.

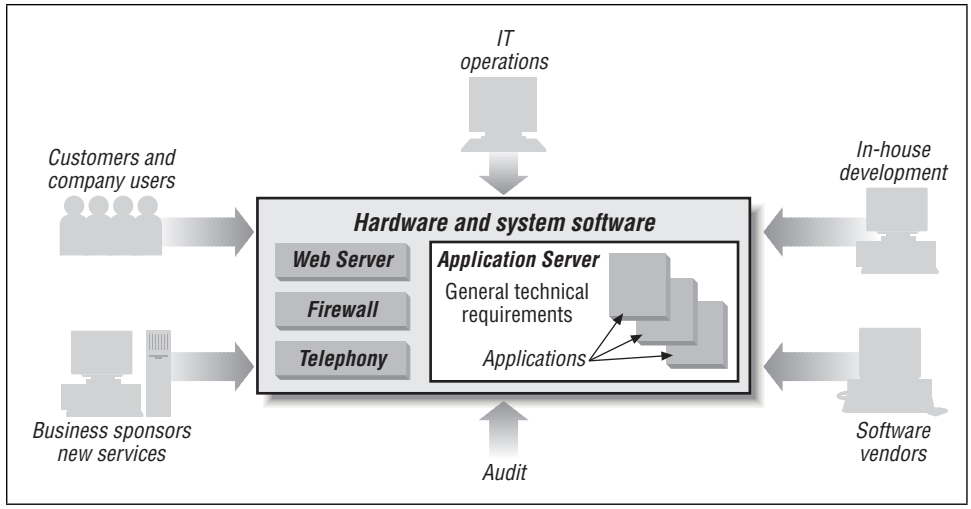

Figure 1-3. Structure of modern business applications

Examples of Business Applications That Use CICS

CICS is used in many different ways by many different businesses. Let's look at a few examples in which CICS is used to help businesses. These examples are based on real customers. For a much longer list of case studies, go to the CICS home page at *http://www.ibm.com/software/ts/cics/* and select case studies.

Financial services (banking, securities, investment services, and insurance)

Company A provides a wide range of services, from consumer banking to securities brokerage. Transaction processing is essential in providing these services. The company uses CICS to produce timely, accurate financial information, in the knowledge that if problems arise they can be resolved easily. Recovery of data is critical. There are a large number of vital CICS applications, written in COBOL many years ago, which Company A doesn't want to give up. But, at the same time, it wants to embrace the World Wide Web and spread some of its applications to workstations and Unix systems. The requirement is to retain the reliability and integrity of its mainframe-based systems while meeting the need from internal and external customers for more flexible, distributed processing.

With CICS, this is not a problem. The company's IT managers can use their existing COBOL programs, retaining all the existing CICS transactions. In addition, they decide to use their current files as their database and then use the CICS External Call Interface (ECI) as a gateway for non-CICS servers to gain access to their business data. This ensures that they extend their business to the distributed environment. Having done that, they are then able to implement a web-trading application very quickly. Six months later, they were

processing 60,000 transactions per day in this environment, on top of a peak load of 1,200 CICS transactions per second. The key to their success was to build an infrastructure that enabled them to extend their mainframe processing to the distributed environment.

Bank B, which provides full banking sevices with 800 branches online connected with distributed CICS servers, processes 30 million transactions a day using all the business attributes of CICS: reliability, recoverability, scalability, security, and so on.

Retail

The distributed processing model can be extended to the retail market. Take the case of Company C, which runs a chain of pharmacies across the country. With the help of CICS, a data sharing system is set up for the processing of customer prescriptions. The system allows customers to input their information (name, address, specific allergies, and so on) and allows for detailed online checking to ensure that the drugs being dispensed are right for the customer. The information required to give the complete picture can be built up on the pharmacist's display in real time using CICS and DB2.

Distribution

The distributed processing model can also be used to track packages, from the time that they leave manufacturing to their final delivery. In Company D, drivers using hand-held devices record the delivery of the packages, and this information is sent to the server and made available to the anxious customer. The flexibility and variety of modes to input information—together with the ability to instantly deliver that information to the place it is wanted—are the true benefits of this real-time system.

News and information

At the Olympics, there is a huge amount of computer processing. Two of the main requirements are controlling the movements and providing adequate security for the competitors and officials, and dealing with the results from thousands of events.

During the games, more than 150,000 competitors and officials require access to 80 venues and facilities. Part of the process is a timely, accurate procedure for registering, authenticating, and badging the competitors and officials, and using those badges as a means to manage access to venues, thus ensuring the safety and security of the events. The badging process alone involves 5,000 complex transactions a day, together with background transactions involving 100–150 concurrent users, and emphasizes the need for a system that supports a high transaction throughput in a distributed environment.

The results system gathers, calculates, and tabulates information from the timing, scoring, and judging stations. The results are immediately sent out to the

venue scoreboards and interactive touch screens, where they are checked by judges. They are then transferred to the mainframe for distribution to other venues, to printers and to the massed ranks of sports journalists. The Commentator Information System (CIS) communicates directly with the mainframe and PCs that run the results systems. A touch screen allows commentators to pull up information about different sports and participants, allowing a commentator in one venue to comment on several events taking place elsewhere. Altogether the data collection system contains over one million data fields of researched, validated historical results, 20,000 biographies and over 30,000 paragraphs of text—a total of 60 gigabytes of information.

CICS works as an integral part of both systems, people management and processing of results.

2

Designing Business Applications

In this chapter:

Chapter 1, *Introduction*, looked at the essentials of a business application, and the advantages of using CICS to create and run business applications. It also described how many new types of applications, such as interactive web sites, involve an application server such as CICS. As well as having a long pedigree in supporting traditional business applications, CICS also has all of the characteristics needed to support the new types of applications. In this book we are going to develop a fictitious company called KanDoIT. They have been in business for a number of years and now want to expand their business and benefit from e-business opportunities either through the Web or by using message queuing technology or using clients to link to their CICS servers. Initially they have to set about gathering requirements from users and begin to develop an application that satisfies those needs. Much of the remainder of this book describes the design and the programming of the components of the KanDoIT company's application.

Before looking at the details of the application, this chapter gives you some more ideas about the facilities that you can exploit in CICS to make writing scalable transactions with integrity easier. It covers the following topics:

- The key design elements that you need to consider when developing general business requirements
- The CICS facilities that support the key design elements
- The process for developing the components of a business application

The Heart of a Business Application

There are three key aspects of a business application design that support the general business requirements outlined in Chapter 1:

- Components
- Transactions
- Error handling

These are described in more detail as follows.

Components

An important principle of business application design is to separate program code into components. Although this may sound obvious, this has not always been done and is such an important topic that we are going to spend some time on it.

Components are not the same things as objects, nor are they simply the *divide and conquer* aspect of implementing a large project. Components are about managing a complex IT environment, keeping it in step with the needs of the business, and about reuse—the ability to use large amounts of an existing application to build a new one. With good component design, you have the ability to enhance a business application rapidly in response to market needs or to exploit a newly-emerging technology with adherence to business rules assured and without loss of auditability.

There are three aspects that fundamentally differentiate the parts of a modern business application:

Different responsibilities within the overall application
: For example, in a bank application, one program might deal with personal accounts, and others with scheduled transactions or cash.

Different types of responsibility
: For example, one program may be dealing with presentation of data to users, another with interest rate calculations or credit to an account which results in the update of databases.

Different hardware and software platforms
: For example, you might have part of an application running in CICS Transaction Server, part running on another server platform, part running on a client workstation, and part running on a web server that presents static HTML and converts business data to dynamic HTML. Some are *server components*; others are *client components*.

  The advantage of server components is that they are (usually) installed in one place and shared by all users. Therefore they can be maintained easily. The

advantage of client components is that users get more predictable response times from the client code. However, because client code is installed in many places, it can be more difficult to maintain unless you employ methods that automatically update client code.

Where you place the function of a business application is determined by finding a balance between achieving good response times and maintainability.

There are also some practical differences; that is, the components may be:

- Run in different geographical locations
- Run on different types of hardware
- Written by different people (in different companies, different groups, or with different skills)
- Developed, tested, and deployed at different times
- Modified, retested, and redeployed at different times

These differences cannot be ignored. Instead, they must be taken into account when designing and developing an application. In other words, your application should be structured into components.

The key components of any business application are *business logic components* and *presentation logic components*. The business logic component is responsible for business calculations and updating databases. These components are the hardest to develop, the most critical to the operation of the business, and the most valuable to reuse. Get these right and everything falls into place. The presentation logic components and their component interfaces control the presentation of information to end users. The presentation logic components are represented in this book as the 3270 interface using COBOL, as a web interface using servlets and Java Server Pages (JSP), as a Graphical User Interface (GUI) using Visual Basic, and as a Java program using MQSeries to access data in our CICS application. In Figure 2-1 we show the 3270 frontend. Each of the three other frontends have a similar input screen and account display screen.

Business logic components

The individual functions of a business logic component should be designed to operate in a server environment. Business logic functions typically:

- Validate input data
- Search the database
- Cross-validate input data and database data
- Update data (including additions and deletions)
- Log activities

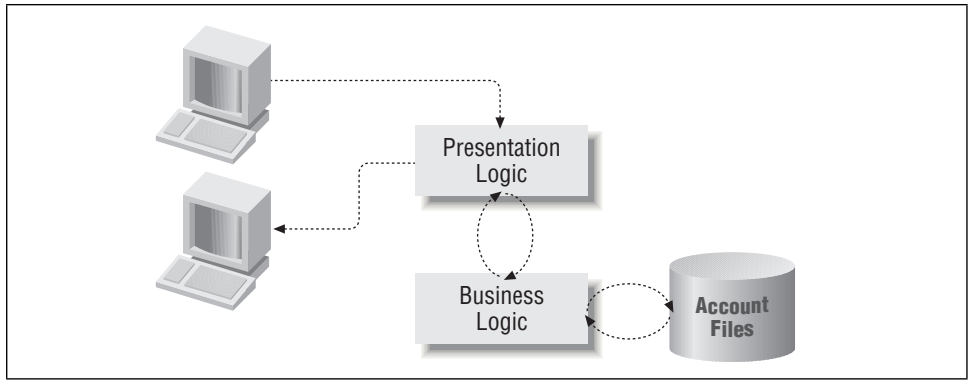

Figure 2-1. Initial outline of a 3270 application

Business logic components should do these things in a way that adheres to business rules and ensures their consistent implementation. For example, they should:

- Update a journal on a particular database containing sensitive material (such as a payroll) for an audit trail.
- Provide the one-and-only approved program that performs complex computations such as interest earned on a deposit account or discount granted on a customer order.
- Obey rules imposed by regulatory bodies.
- Ensure that consistency cannot be compromised. For example, critical updates that are inextricably linked should be done in the same program.

The criteria for grouping business logic functions into business logic components are pragmatic. Business logic components should:

- Encapsulate as much data as possible.
- Have an interface that can be tested independently.
- Be able to move to another server without affecting other components.
- Have a purpose and responsibility that can be discussed in a business context and should not have a purely technical definition such as *everything running on a server X*.

Presentation logic components

This is the code primarily concerned with the presentation of data to an end user and receipt of data from an end user. It should:

- Invoke the general-purpose presentation management code that controls the layout of data on a visual display screen or other output device.
- Validate input.

- Handle interactions in the correct sequence.
- Confirm completeness (that is, that all necessary information has been provided).
- Invoke one or more pieces of business logic, as needed.

Presentation logic may be designed to run on a client. For example, if the user sits at a personal computer, the presentation logic could be a Visual Basic application; if the user uses a web browser, the presentation logic could be a set of Java servlets running on a web server.

Component interfaces

Not all components have interfaces that can be invoked by other components. Those that have invocable variables must:

- Publish the interfaces for use when developing client components (which may not be programmed in the same language or run on the same type of machine).
- Be tested thoroughly against the interfaces.
- Protect the interfaces using security mechanisms.
- Make the interfaces available to any executing client, locally or remotely.

A note about traditional CICS applications

Many CICS applications written in the past combined presentation logic and business logic. Such applications are difficult to treat as components. As such, you would have to use the External Presentation Interface (EPI) to do many of the processes that are described in this book. For example, EPI allows a Java program to emulate a CICS 3270 terminal, and start CICS transactions on the server. The program sends and receives data as CICS terminal datastreams. Programming in EPI involves *screen-scraping*, where you must extract data from a screen-orientated buffer row by row and column by column. It is intended for use by CICS server programs that cannot be modified to be called by the External Call Interface (ECI) due to the tightly coupled presentation and business logic. See Chapter 12, *Designing the Visual Basic Component*, for more information about the ECI.

Transactions

In a business application, a *transaction* has the same meaning as it does in everyday English—a single event or item of business between two parties. Business application transactions should have the so-called ACID* (Atomicity, Consistency,

* The term "ACID properties of a transaction" was coined by Haerder and Reuter in 1983 and used by Jim Gray and Andreas Reuter. In CICS, ACID properties apply to a "unit of work" (see "CICS Transactions").

Isolation, and Durability) properties, which are described in more detail later on in this section. However, before explaining what ACID transactions are, consider the problems that can occur with transactions.

Imagine a component that operates on bank accounts. The component has three services that could be invoked by end users, one to add to an account, one to delete from an account, and one to move money between accounts.

The most difficult of these is to move money, because it must do everything necessary to ensure that all the accounts are updated and that the operation only proceeds when appropriate. Consider what would happen if things went wrong—if the server failed or if the database contained errors after some, but not all, of the updates had been done. One customer could have his money withdrawn, without the other customer receiving any money (providing an unintended bonus for the bank!). Or, even worse (for the bank at least!), both customers could receive money, resulting in an unintended loss for the bank.

Also, consider what could happen if two bank employees try to move money from the same account at the same time. If the component is designed to read the database to check whether there are sufficient funds and then update the database with the new balance, one of the updates could be lost.

To prevent problems such as these, we need to create ACID transactions:

"A" is for Atomicity
: To be atomic, a transaction must execute completely or not at all. This means that every file, database, or queue operation within the transaction must execute without error. If any part of the operation fails, then the entire unit of work is terminated, which means that changes to the data are undone. There is no trace of the attempt to execute the transaction. The requirement may exist to log the start of a transaction for audit purposes. If all the operations execute successfully, the transaction can be committed, which means that the changes to the data are made permanent or durable.

"C" is for Consistency
: This means that the transaction taken as a whole does not violate any of the integrity constraints associated with the state of the resources. Obviously, the program itself is the arbiter of consistency in a business sense. In our simple example, moving money should not alter the total amount of money in the accounts.

"I" is for Isolation
: Isolation means that even though transactions execute concurrently, they appear to be serialized. In other words, it appears to each transaction that any other transaction executed either before it or after it, but not simultaneously.

Remember that we are developing a multi-user system accessing a shared database. When some but not all of the updates needed for consistency have been done, the database is inconsistent. When all the updates have been done but the transaction has still not yet committed, the transaction could back out. In either of these cases, allowing other transactions to see or update the records could result in errors in the database.

Isolation means that other transactions can be prevented from seeing or updating the same records. In other words, the data that a transaction accesses cannot affect or be affected by any other part of the system until the transaction is completed. If moving money was not isolated, an inconsistent view of the database would be possible—or worse, concurrent updates could corrupt the database. Complete isolation is logically equivalent to forcing serialization of all transactions against the database, only allowing concurrent transactions that do not affect each other in any way at all.

"D" is for Durability

This means that after a transaction completes successfully (that is, commits), its changed state survives failures. This normally requires that all the data changes made during the course of a unit of work must be written to some type of physical storage when the transaction is successfully completed.

Trying to implement the ACID properties by unique application code in every business application component would be difficult. It would totally obscure the business logic, making the code difficult to maintain and audit. The problem is that some things are not business issues but technical issues—things the technology should address in order to ensure the code is as easy to write as possible—so as to allow the business logic programmer to focus on the business issues. There is a clear difference between business exceptions, (such as, "the account does not have enough money in it") and technical exceptions (such as, "the account database has only accepted one of my updates"). Our system software, our middleware, should hide the messy realities of the machinery and present us with services that don't need us to solve this problem.

Using an application server such as CICS, you can obtain the ACID properties by defining groups of updates that must all be done together. As the transaction proceeds, the updates are done on a provisional basis, and logged. If they all work, you can commit (syncpoint) the changes, that is, make them permanent. But if there are problems you can cancel the changes, that is, back them out. If the system fails—say there is a power outage before you have issued the commit—the system automatically backs out the changes you made. Your program's responsibility has ended, though the client may need to resubmit the request once power is restored. You can concentrate on ensuring that business exceptions are handled, and you can safely leave the technical exceptions to the system.

Limitations of ACID transactions

Does the above sound too good to be true? You're right, it is! Here are the potential problems.

An ACID transaction should be "short," because it locks shared resources.
Here, "short" is a relative term. In a large system running hundreds of transactions per second, short would be less than a second. It is up to the designer to set a target for the duration of a transaction, but in almost all cases it is undesirable that human think-time be allowed to control the duration of an ACID transaction. This is important because it means the programs should not wait for user input in the middle of a transaction, however appealing it is to ask for user confirmation before committing. "Maintaining File Integrity: Using Locking" in Chapter 4 shows some of the practical ways to work around this issue.

ACID transactions should not be "big."
Here, "big" is also relative. For example, don't implement a transaction that runs through the entire database correcting telephone numbers due to changes in the national telephone numbering plan. It should not be logically necessary to do this atomically, that is, do one record or row at a time and commit the result before passing it on to the next, because otherwise it would effectively lock out all other users until completed.

Be aware that under certain conditions a state known as "deadlock" can occur.
This is where two or more transactions are trying to access the same set of resources and each ends up waiting for the others to complete. For example, program A has updated a record with key X and then wishes to update a record with key Y. Program B wishes to do the reverse. Neither can continue because invalid data would result. The system detects this and arbitrarily cancels one of the transactions which, since there is no actual error, should simply retry and start from the beginning.

Complete isolation may have intolerable throughput implications.
There are cases with adding or deleting records where complete isolation may not be recommended, in situations where, for example, all transactions are serialized. So typically, systems provide the ability to relax the isolation property, allowing improved concurrence while still preventing multiple concurrent updates to the same record.

Error Handling

It is a well known fact that in computing 80% of programming code deals with handling errors and/or exception conditions, that is, with knowing what is supposed to happen, checking that it has, and doing something about it if it has not!

Suppose a device error occurs on a disk—the data will not be available until the disk is replaced and the data restored. But suppose the application that was trying to read the disk failed to check that the operation succeeded and, as a result,

wrote incorrect data to another file. Merely restoring the original data will no longer fix the problem, and finding out why the second file contained garbage could be very difficult.

Suppose a software bug in an infrequently used part of a program caused an undetected overwrite of stored data. Not only could subsequent uses of the program fail because some piece of memory is corrupted, but finding out why the problem occurred could be very difficult.

These examples show why handling errors is vitally important. Errors are a fact of life. In developing applications, it is best to remember these two maxims:

- "Anything that can happen will happen, usually at the worst possible time."
- "Absolutely *anything* can happen!"

Errors that disrupt business applications can stop a company doing business until the problem is fixed, so it is vital to detect errors early, clean up after the errors to allow operations to continue, and record diagnostic information so that problems can be fixed quickly.

The need for a methodical approach

In component-based applications, errors mostly occur in the interface between two components. It is therefore important to document and test the interfaces in a methodical way. Also, components are supposed to hide complexity from their users, so we want to notify errors in a simple way from the user's point of view.

How is that done? First, some definitions:

Fault
: Something going wrong, such as a LAN (Local Area Network) failure, software bug or unexpected data received.

Error
: The symptom of a fault, such as a parameter check failing.

Abend (Abnormal ending)
: The situation where program execution cannot continue normally, such as a storage violation.

Error notification
: The passing of information that an error has been detected from one component to another component. This can either be by return code or by executing an abort command.

Signalling component
: The component that notifies an error.

Receiving component
: The component to which notification of an error is given.

As for noticing an error, we need to decide if there is a fault causing it. Not all errors are the result of a fault; for example, a module issuing a database read command that returns a "record not found" error may regard the error as acceptable.

If there is a fault, you need to:

1. Record enough diagnostic information so that computer support staff can quickly determine what the fault is and correct it.
2. If possible, bypass the error so that no other components are affected by the problem.
3. If the error cannot be bypassed:
    a. Clean up as much as possible.
    b. Notify other components of the error.
    c. For presentation logic components, provide clear instructions to end users regarding who to inform about the error and what to do to continue working.

In practical terms, the above guidelines translate to the following:

During design:

- Always specify that component interfaces must have return codes.
- Consider what the end user should do when errors occur.

During development:

- Always check for return codes.
- Give meaningful return codes.
- Clean up before returning.
- Log errors.
- Include code for an abnormal end (abend) situation.
- Include error messages, explaining what needs to be done, in the end user interface.

You'll find more information about error handling in "Handling Errors" in Chapter 4, which looks at error handling in the COBOL business logic component of the sample application.

How CICS Can Help the Application Designer

A business application should incorporate the key design elements of components, transactions, and error handling. This section provides some more background to show how CICS supports the application designer to structure an application so that it meets the business requirements described in Chapter 1, and incorporates the key design elements. CICS provides the following facilities:

- An environment for executing presentation logic and business logic components
- Calls between components
- Efficient control of concurrently running application programs serving many online users
- Provision of ACID properties through management of the units of work (see the following section)
- Shared error handling
- System management

It's useful at this point to look at CICS transactions, and CICS programs and linking, in more detail. We'll also consider how CICS deals with critical activities such as error handling and security.

CICS Transactions

In a CICS application, a transaction is the processing initiated by a request, usually from an end user. A transaction starts, executes, and ends. A single business transaction (such as the enrollment of a new customer) may involve several CICS transactions.

As well as referring to a single event, *transaction* can also refer to the class of similar events. Thus, we speak of adding Mary Smith to the payroll file with a (single) *add* transaction, but we also speak of the *add* transaction, meaning the class of additions to that particular file.

A CICS transaction may contain one or more Units of Work (UOW) where a UOW begins with the first action to alter a protected resource and ends with either an explicit or implicit syncpoint. To summarize, a business transaction involves one or more CICS transactions that, in turn, can involve one or more units of work.

When processing transactions, CICS accumulates performance statistics and monitors the resources used. This provides the information that enables user departments in your organization to be charged accordingly. It also allows you to find out which parts of CICS are being heavily or lightly used. This helps your systems managers to tune the system to improve its performance.

CICS Programs and Linking

A program is the smallest replaceable unit of an application. Programs are compiled, linked or bound, and turned into a single executable file which is then deployed. They can be written in many languages and run in many different environments. A component of a business application typically consists of multiple programs.

CICS provides a variety of ways for programs using CICS to invoke and be invoked by other programs running inside or outside CICS, for example on web servers or end-user workstations. Program calls can be:

Synchronous
: Control is not returned to the calling program until the call is complete.

Asynchronous
: The calling program continues executing while the call is performed. The access method informs the application program after the operation is completed.

Synchronous calling between programs

There are two main ways in which one program can invoke another program inside CICS. These are discussed in more detail in "Commands for Passing Program Control" in Chapter 5. For the time being suffice it to say that the CICS API consists of a number of commands that define resources, make calls to other programs and so on. In this book you will meet a wide range of these commands particularly in Part II and Part V. Two of these commands create links from one program to another. Figure 2-2 shows the linking process and relationship of one program to the next. The commands are:

EXEC CICS LINK
: Allows one program to transfer control to another in a synchronous manner and continue execution after the called program has returned. This also occurs by means of native programming language facilities such as a COBOL CALL statement."The LINK Command" in Chapter 5 gives more detail.

EXEC CICS XCTL
: Allows for one program to call another in a synchronous manner but unlike *LINK* will not receive control back when the called program returns. This has

no equivalent high-level language facility. It is particularly useful in error processing when an unexpected error is detected that implies that the program which trapped the error cannot continue. "The XCTL Command" in Chapter 5 gives more details.

In both cases a COMMAREA (for "communication area") is used to pass parameters and returned values. For more information about COMMAREAs see "Saving Data: Using a Scratchpad Facility" in Chapter 5.

The *LINK* and *XCTL* commands introduce the idea that different programs involved in the processing for a transaction can be executing at different logical levels. A program invoked directly by CICS is considered to be at the highest logical level of the task (level 1). If it then uses the *LINK* command to link to another program then the linked-to program is considered to be at a lower logical level (level 2). However, if the program detects an unexpected condition it can use *XCTL* to call an error handler which will be considered by CICS to be executing at the same logical level as the program issuing the *XCTL* request. Figure 2-2 illustrates this principle.

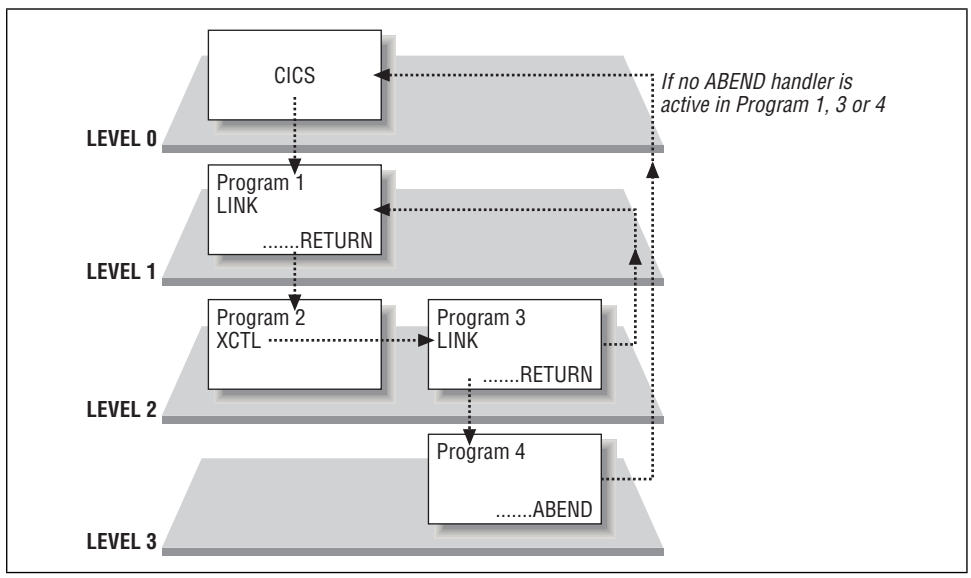

Figure 2-2. Transferring control between programs (after an Abend)

Asynchronous calling between programs

One CICS program can invoke another asynchronously passing data in the *FROM* area defined in the *EXEC CICS START* command. The program for which *START* has been issued is executed as logical level 1, independent of the level of program that issued the *START*. The *FROM* area can be accessed by executing a *RETRIEVE* command.

Alternatively you can use MQSeries. This is discussed in "Working with MQSeries," later in this chapter.

Calling CICS programs from non-CICS programs

CICS provides a number of ways in which non-CICS programs executing on a variety of hardware and software platforms can call CICS programs. In fact, there are more ways than we are able to describe in this book, but here are a few examples:

A software package known as the CICS Client provides the ECI and EPI facilities:

ECI (External Call Interface)
: Allows the calling program to call a CICS program as though it had been linked to (using the LINK command) by another CICS program. The ECI uses a COMMAREA.

EPI (External Presentation Interface)
: Allows the calling program to call a CICS program as though it had been invoked by a user at a 3270-type device.

The CICS Client is designed to run on end-user workstations and meet the needs of a single user running programs that invoke CICS programs. Our sample application makes use of the CICS Client ECI function call.

For programmers using the ECI or EPI, there are application tools that build some of the program calls automatically. One such tool is VisualAge Interspace. The sample application demonstrates the use of Interspace.

For server applications, a software package known as the CICS Transaction Gateway also supports the ECI and EPI. The CICS Transaction Gateway would be used, for example, by programs running under the control of a web server.

When the calling program is written in Java, there is a pure Java version of the ECI and EPI facilities that access the CICS Transaction Gateway using TCP/IP.

Finally, a CICS program written in Java can be invoked using the Internet Inter-ORB Protocol (IIOP); the sample program uses IIOP.

Defining resources

CICS has a set of resources that are grouped into categories, for example, File Control (FC), Temporary Storage (TS), and so on. These resources can be defined in batch using the DFHCSDUP utility program or online using the Resource Definition Online (RDO) transactions. A historical note: several years ago, many resources were assembled (sic) together into tables; so, for example, File Control resources were aggregated into a File Control Table (FCT).

Working with MQSeries

For asynchronous processing, you can achieve the greatest flexibility by using IBM's broad range of MQSeries products. Neither the calling program nor the called program needs to know anything at all about where or when the other program executes because MQSeries provides a common set of facilities for sending and receiving messages, takes responsibility for routing messages to the required location, and holds the message until the receiving program wants it. The sample application uses MQSeries to invoke a CICS program.

Error Handling Facilities

Error handling facilities are critical in any application. CICS includes several error handling facilities, including:

- ABEND
- Return codes in commands
- DUMP
- TRACE
- Sending messages to system consoles where operators can respond to the problems

These are described in more detail in "Handling Errors" in Chapter 5.

Security

Security is a complex subject that requires careful planning before it is implemented; as such, it is not dealt with thoroughly here. As we have already mentioned, many people, with many different roles, interact with business applications and the security, auditability, and accuracy of the application depends on these roles being kept separate and identifiable. Therefore it is natural that security of business applications should be role-based.

Once each user's role has been understood, security can be implemented on the following basis:

Authentication
: Where the user's identities are verified, typically using a user ID and password approach

Authorization
: Where an attempted action by a user is checked to see whether it is permitted

CICS works in co-operation with OS/390 security managers (for example, Resource Access Control Facility (RACF), Access Control Facility (CA-ACF2), or TopSecret) to implement this role-based security, with CICS calling the security manager as