

AS/400 Application Modernization Strategies

September 1999

D.H.

ANDREWS

GROUP

AS/400 Application Modernization Strategies

**Sharon Hoffman
Ronald Fess
Paul Remtema
James Walts**

Published by:
**D.H. Andrews Group
700 West Johnson Avenue
Cheshire, CT 06410
(203) 271-1300**

September 1999

About D.H. Andrews Group

Since 1984, D.H. Andrews Group has been helping organizations make more effective use of computer technology. Much of the consulting practice focuses on replacing older applications with newer and more effective ones based on a mix of technologies including PCs, AS/400 servers, Unix computers, and mainframes. D.H. Andrews Group has pioneered the development of management techniques to speed up the implementation of improved applications. In 1998, D.H. Andrews Group opened a new division, Rochester Technology Center (RTC), in Rochester, Minnesota.

D.H. Andrews Group helps organizations plan their use of information technology and then assists in the implementation of new systems. We are also widely recognized for our clearly written computer industry reports on current technologies including IBM's AS/400, S/390, RS/6000, and Netfinity servers, Java, client/server, LANs, networks, and Internet computing.

D.H. Andrews Group Reports

D.H. Andrews Group has been a leading independent publisher of computer industry reports since 1987. Over 1,000,000 copies of our reports have been published in a variety of languages including French, German, Italian, Spanish, Danish, Dutch, Japanese, Korean, and Chinese. Recent reports include *Making Java Smoother And Easier: IBM's VisualAge For Java*, *Java And AS/400 – Perfect Together*, *ERP And IBM's RS/6000: A Solid Combination For Medium-Sized Enterprises*, *Contingency Planning For The Year 2000*, *Testing For The Year 2000 Deadline*, and *Global Government Computing: IBM's New S/390 Enterprise Server*. A complete S/390 and ERP series covering eight vertical markets was published in the summer of 1999.

About the Authors

Sharon Hoffman is a technical writer and educator specializing in the AS/400. She began programming IBM midrange systems in 1981 and is currently a senior technical editor for *News/400*. Sharon can be contacted by e-mail at shoffman@techreflections.com.

Ron Fess, a Consulting Software Engineer with Rochester Technology Center, was one of the early designers of OS/400. He has continued to work on the OS/400 design as well as AS/400 openness and technical strategy. At RTC, Ron specializes in the area of application modernization and the design and implementation of client's software solutions.

Paul Remtema, a Consulting Software Engineer, has recently joined Rochester Technology Center where his focus is on application modernization. His areas of expertise include Java language interoperability and Java performance. Paul was one of the original ILE architects.

Jim Walts is a Consulting Software Engineer with Rochester Technology Center. Jim was one of the key designers of the AS/400, leading the integration of emerging technologies in networking and system management environments. He led IBM's Network Computing Java initiatives by heading up the integration of Sun's JVM and IBM's Java JIT compiler into the IBM NetStation.

A special thanks to *Janet Krueger* of Rochester Technology Center for her contributions.

AS/400 Application Modernization Strategies is published by: D.H. Andrews Group
700 West Johnson Avenue
Cheshire, CT 06410
Phone: (203) 271-1300
E-mail: dha@dhagroup.com
Internet: <http://www.dhagroup.com>

Copyright September 1999 by D.H. Andrews Group, Inc.

Table of Contents

Introduction	3
Executive Overview	5
Structural Modifications	6
Training and Tools	6
Improved Usability	7
Evolving Application Models	8
A Look Ahead	8
Justifying Application Modernization	9
Beyond the Beginning	9
Application Modernization Overview	10
Application Modernization Decision Points	11
Justifying Application Modernization	12
Getting Started	14
Application Structure	14
Details Are Important	16
Tools of the Trade	17
Take Advantage of Mass Production	18
A View from 30,000 Feet	19
Defining Requirements	20
Evaluating the Status Quo	20
Eye on the Prize	21
Acquiring Skills and Development Tools	23
<i>The Road to e-business by David Slater, IBM Software Group</i>	24
User Interface Tools	26
Business Logic Tools	27
Database Maintenance	28
Building Application Development Skills	30
The Road Ahead	32
Implementation Issues	32
Application Servers	32
Messaging Software Concepts	33
Deployment Options	34
For More Information	36
Glossary	37

Not to be excerpted, copied, or republished without the express written consent of D.H. Andrews Group.

All trademarks are the property of their respective owners.

The information in this report is based on sources we believe to be reliable, but its accuracy, completeness, and interpretation are not guaranteed. This report should not be relied on as a sole source of information or opinion on the computing industry. D.H. Andrews Group does not assume responsibility for typographical errors, and the opinions expressed in this report are subject to change without advance notice.

Introduction

AS/400 shops are under intense pressure to deliver new and updated applications that incorporate features such as graphical user interfaces, data mining, and e-commerce. Application modernization is the process that enables these new functions and paves the way for integration of new technologies including Domino, Java, event-driven programming, and others that we can't even imagine today. Application modernization can include anything that makes applications more efficient for users, more adaptable to new technologies, or easier to maintain. In our discussion of application modernization, we include the process of enhancing programmer skills and tools to achieve these goals.

The financial justification for application modernization is often provided by the potential for increased sales, reduced costs, improved usability, or better customer service. For example, e-commerce can provide a company access to customers who might never consider purchasing its products through traditional channels; business intelligence solutions can minimize inventory levels and their associated costs; GUIs can reduce employee training time; and workflow applications can eliminate paperwork and improve response time for customer requests.

Application modernization can also deliver important IT benefits. New application development methods and program structures improve programmer productivity and application flexibility, making it easier to adapt to the next wave of user demands and emerging technologies—whatever they may be.

Because application modernization has different meanings for different businesses, the first step in any application modernization project is to determine the goals. Keep in mind that these goals may evolve over the life of a project as new technology becomes available and the staff learns new techniques and gains new insight into the application modernization process. There is a wealth of information that can help crystallize goals and define strategies for reaching those goals.

It's difficult to define a common strategy for AS/400 application modernization because the goals and the initial starting point of an application modernization project can vary so widely. Even for installations with similar goals and starting points, there are many decision points where the approach to application modernization may diverge.

In this report, we focus on why application modernization is important and what can be accomplished. We consider a wide spectrum of potential benefits ranging from increased sales to more accurate customer demographics. We address both short-term cosmetic goals, such as GUI front ends, and long-term application redesign goals, such as platform independence. We look at how these results can be achieved and the products and techniques that facilitate application modernization. We use specific products to illustrate the types of solutions that are available, but we have not attempted an exhaustive review of these tools, both because of space constraints and because the marketplace is evolving so

Introduction

rapidly. (For more information about products that can facilitate application modernization, see “Acquiring Skills and Development Tools.”) This report focuses on the challenges inherent in application modernization, the decisions you’ll need to make, and the factors that affect those decisions. Armed with this information, you can develop a customized roadmap for your own series of ongoing application modernization projects.

Throughout this report, we have used IBM and independent software vendor (ISV) products to illustrate the types of tools that are available to accomplish application modernization tasks. The use of a particular tool as an example does not imply an endorsement, that the specified tool is the only option available, or that the product performs as described. Readers are responsible for determining whether products are suitable for use in their installations and for testing all products in their environments. Readers are encouraged to investigate the entire spectrum of application modernization tools (see “For More Information” at the end of this report.) ♦

Executive Overview

Application modernization, as defined in this report, is a response to changes in the AS/400 marketplace. The spread of Internet technology and familiarity with PCs create a demand for new user interfaces and entirely new classes of applications such as e-business, workflow, and data warehouse solutions. These new methods of doing business are facilitated by the advent of tools such as Java and Domino for the AS/400.

Application modernization is an umbrella term that encompasses a whole spectrum of tools, techniques, and philosophies that help IT shops respond to these changes. Because application modernization is an ongoing process, it makes sense to define a series of discrete application modernization projects with goals for each one. Each application modernization project builds on the skills and successes of prior application modernization projects.

Application modernization is not limited to grandiose projects, to any specific technology, or to revolutionary changes to the business model. You can design an application modernization project with a specific deliverable—such as a web-enabled catalog ordering system. Or, you can focus on a less tangible goal—applications that adapt more easily to new technologies, for instance.

The simplest way to deliver application modernization may be by installing a new packaged solution. This works particularly well if a software package is already in place and it has not been modified heavily. It's very likely that the vendor has an

upgraded version of the product that can address user requests for new interfaces and functions.

Another low-impact application modernization project might use a screen scraper to convert green-screen applications to GUIs using an automated process. The rest of the application remains unchanged. Yet another small-scale application modernization project gives customers visiting a web site the option to request a catalog. The customers fill out a simple form, and the data is processed by the existing catalog request program in batch mode.

Application modernization is difficult to define because it's different for every IT installation, but there are some common threads that apply to most application modernization projects. The overall concept of application modernization can be viewed as four intertwined transition strategies. A particular application modernization project may include one, two, three, or all four of these elements.

1. *Structural modifications.* Changes to the application architecture address problems with the underlying structure of an application that inhibit the use of new technologies. In addition, structural changes pave the way for a possible transformation to object-oriented design in the future.
2. *Training and tools.* Enhancements to application development skills and new tools are critical components in application modernization because they improve programmer productivity and enable other types of changes.

Executive Overview

3. *Improved usability.* Shifting user interface expectations, such as demands for graphical user interfaces (GUIs) and web-enabled functions, are driving forces for many application modernization projects.
4. *Evolving application models.* Changing business environments may require completely new applications (e.g., data mining) or major revisions to existing applications (e.g., the transformation of a green-screen order entry system into an e-commerce application).

Our discussion of these application modernization strategies is targeted for RPG shops interested in delivering new application functions, such as GUIs, data mining, workflow, and e-commerce, and integrating new technologies, such as Java and Domino (perhaps as a mechanism for delivering the new application functions). However, much of the discussion in this report is applicable regardless of your development environment. In the next sections, we'll expand on the transition strategies outlined above.

Structural Modifications

An important application modernization goal is the flexibility to adapt as new technologies become available. To achieve this flexibility, applications need to be designed so that each logical function is a separate, standalone element, and the interfaces between the elements are clearly defined. This type of structure is inherent in good object-oriented design, but it can be achieved with any application development methodology. This report makes a distinction between two techniques that facilitate this separation of an application into its functional parts: application partitioning and modularization.

Application partitioning describes the process of segregating an application into its three major parts: data maintenance, business logic, and user interface. Application partitioning makes it possible

to distribute applications across multiple platforms, to access a variety of databases, and to support new user interfaces and new devices without disrupting the business logic. Ideally, each partition should be developed independently.

Modularization is most apparent at the level of a programming task, and it addresses the fundamental structure of the application. Implementing application partitioning implies a certain level of modularization, but modularization has a more specific meaning. In a modular design, each function is independent, and the interfaces between functions are clearly defined. The building blocks used to accomplish this structure are less important than the structure itself. Each function (module) may be a separate program, or each function may be a subroutine. When several functions are combined into a single program, the modularization remains intact. AS/400 service programs that perform standard functions, such as pricing a product or calculating sales tax, are a good illustration of modularization that is nearly independent of application partitioning.

Training and Tools

New skills and application development methods may be byproducts of an application modernization project, or they can be a primary goal of the project. The most obvious example of skills as goals is a software consulting firm that specializes in custom programming. Obviously, if clients are looking for Java skills and the existing staff has RPG III skills, a skills enhancement project is a good investment. However, in-house IT departments should not dismiss skills enhancement as a goal of application modernization.

Investing in new development tools and skills can pay off in improved employee morale and higher productivity. In addition, prospective employees may see an opportunity to work with modern application development tools as an incentive to join the

staff. Finally, it may be necessary to learn new skills and design methodologies before tackling other application modernization projects. Otherwise, it is difficult to determine what options are available, what resources will be required to implement the project, and how to interface with existing applications. (For example, a programmer who has never used Domino probably can't assess whether it's a good solution for a particular project.) Your application modernization projects will probably be more successful if you build new skills gradually.

Even those shops that are not interested in adding new technologies to their application development portfolios should consider the efficiencies available with contemporary development and programming tools. There are substantial benefits to using a graphical source-code editor such as CODE/400, regardless of what kind of applications you develop. Similarly, even if you aren't interested in using new RPG techniques, you should consider using the current RPG compiler to ensure that you will have access to IBM support if it's needed.

Beginning with V3R1, IBM has focused its compiler development resources on RPG IV (ILE RPG). The RPG IV compiler includes new language functions such as the EVAL opcode that make writing traditional RPG programs more efficient. In addition, ILE constructs such as service programs and procedures make it much easier to develop modular code and therefore make RPG applications easier to maintain.

Some tools have the added benefit of insulating application development from changing technology. If you adopted an application generator several years ago, it probably created a client/server application using GUIs and RPG III business logic. Today, the same application generator may output a browser-based application using Java applets and RPG IV business logic. In the future, the same instructions can be used to generate new designs that include servlets, Enterprise JavaBeans (EJBs), and other

elements that have yet to be invented. An application generator makes it possible to adapt to this shifting application environment with minimal training and without disrupting the existing application design methodology. In order to reap these benefits, you must choose your application generator carefully—a vendor that is investing in application modernization projects of its own is more likely to implement new technologies as they become available. You must also be prepared to implement new releases of the application generator as they become available.

Defining an application modernization project solely in terms of deliverables ignores the programming productivity benefits that are potentially available. Whether improved productivity is seen as a fringe benefit or an important project goal, it's critical to include skills assessment, training, and funds to purchase application development tools in the application modernization equation.

Improved Usability

Changing expectations for user interfaces, particularly the demand for GUIs, are the primary impetus for application modernization in many IT environments. While we don't agree with those who think there's no place for green screens in a modern AS/400 environment, there *is* intense pressure to modernize user interfaces.

To address the requirement for modern user interfaces you must consider all aspects of the user interface—not just the appearance of the screen. A screen-scraping solution that retains the 5250 data stream may satisfy immediate demands. However, unless the application is rewritten to provide an event-driven flow (as opposed to the traditional procedural flow of an RPG program), users will be frustrated because the application will not behave like other Windows-based applications. Thus, new user interfaces, new application designs, and new business models are closely related. In this larger view,

Executive Overview

usability options run the gamut from GUI front ends to completely new applications and new ways of doing business, such as e-commerce.

Evolving Application Models

Contemporary technology, low-cost high-performance PCs, and changing business practices create the possibility and the need for entire classes of applications that didn't exist in the past. E-commerce is an obvious example of this tendency, but it is also evident in the requirement for data warehouse applications, the trend towards server consolidation, and the popularity of Domino for AS/400. It's worthwhile to spend some effort understanding the opportunities that are materializing as a result of these new technologies and ways of thinking.

In 1996, the idea of a web-enabled AS/400 catalog application would have been ludicrous. The application development tools weren't available, and most people didn't have access to the Internet; so, customers couldn't utilize a web-enabled catalog even if IT could deliver the application. In 1999, a web-enabled catalog application not only makes sense for many AS/400 shops, it may be a crucial competitive advantage. However, a web-enabled catalog could easily take 6 to 12 months to implement depending on the current applications, the skills available within the IT department, and other development priorities. During that time, new technologies and application development tools will certainly be introduced for the AS/400. It makes sense to design a project with measurable goals but to allow enough flexibility so that the specific goals can be adjusted in response to technological innovations and evolving skills and tools.

Evolving application models enabled by technologies such as distributed computing, Java, and workflow have enormous potential, but designing an application modernization project that involves a new business model requires ingenuity and a certain amount

of caution. Because there are no tried-and-true designs for these applications, it can be difficult to determine specifications, performance characteristics, and the right technical solutions. As a result, application modernization is best implemented as an evolutionary process that allows room to adapt to changing goals and technologies over the life of a project.

A Look Ahead

This report focuses on the immediate application modernization problems facing AS/400 shops whose current environment is primarily RPG III and green screen. Readers who have begun application modernization by adopting RPG IV, creating some client/server solutions, or utilizing techniques such as triggers to isolate database maintenance from business logic will also find plenty of useful material. Our primary goal is to illustrate the decision-making process that is integral to development of a customized application modernization roadmap. To illustrate this process we'll introduce a sampling of the many tools that are available to facilitate application modernization. (The chapter entitled "Acquiring Skills and Development Tools" includes more information about application modernization tools.)

The initial phase of application modernization discussed in this report delivers important benefits and positions you to implement new business models including (but not limited to) e-business, business intelligence, workflow, groupware (a.k.a. collaborative computing), distributed processing, and server consolidation. In "The Road Ahead," we'll explore the next phase of application modernization, which deals with technologies that can be used to implement these models. For example, Java is a popular option for implementing e-business, but choosing Java is just the beginning. Java can be deployed on the client, on the server, or both. Decisions have to be made concerning the infrastructure (e.g., firewalls and web servers) and the architecture of the applica-

tions (e.g., servlets, Enterprise JavaBeans, etc.). This next tier of decisions is beyond the scope of this document, but “The Road Ahead” provides a preview of the types of questions you’ll face.

Justifying Application Modernization

Application modernization can be an expensive, time-consuming, frustrating exercise. Why make this investment? Potentially, application modernization can provide both business benefits and technical benefits. Many shops invest in application modernization in order to facilitate new business models such as e-commerce or international sales. On the technical side of the equation, application modernization promises improved maintainability, platform independence, and the ability to adapt to changing business needs.

Benefits run the gamut from better employee retention (perhaps because of opportunities to learn new technology) to improved sales as a result of e-commerce. Obviously, it’s easier to justify application modernization if you can draw a direct correlation between profits and an investment in new application development techniques, but more often the benefits are measured in more intangible terms: improved customer satisfaction and more flexible application design. In the next chapter, we’ll take a detailed look at the risks and benefits associated with application modernization and how you can justify investments in these projects.

Beyond the Beginning

This executive overview is designed to illustrate the spectrum of ideas and functions that are included under the broad concept of application modernization. We’ve touched lightly on the four major strategies that amalgamate to form a comprehensive application modernization strategy. We’ve also given you a preview of the decision-making process that is

integral to the design of an application modernization project and some of the benefits you may expect for your efforts. In the main body of this report, we explore all of these issues in much greater detail. Our discussion is illustrated with examples of products that can facilitate the AS/400 application modernization process.

Application modernization is not without pitfalls, but we believe it is critical to the survival of many AS/400 shops. Using the guidelines we provide in this report, you can design a customized application modernization roadmap for an initial group of application modernization projects that will satisfy user and management demands for new interfaces and functions. ♦

Application Modernization Overview

Application modernization is a broad term, and application modernization projects vary greatly. For example, a project that provides a GUI front end for selected programs using screen-scraping technology can be classified as application modernization. At the other end of the spectrum is the application modernization project that includes entirely new application architectures developed in Java, user interfaces that incorporate Domino, GUIs and web-based options, business rules implemented in the database that use triggers and stored procedures, and data mining functions that use neural networks. In order to deal with such diverse requirements in a consistent manner, some basic definitions are required. Because many of the terms used in this report have multiple meanings, a glossary specifying the definitions used is provided on page 37. We also define the most important terms in the relevant sections. Following is a list of terms that are used extensively in this chapter.

Application modernization is the ongoing process of adapting applications to new technology and new business processes. It encompasses changes to user interfaces, application structure, programming methods, and development tools.

Application partitioning is an important component of many application modernization projects. It addresses the issue of segregating each application into three logical components: user interface, business logic, and database maintenance.

Modularization is related to, but different from, application partitioning. Modularization deals with

the internal structure of a program. In a highly modularized design, each functional component is a separate logical component. A program that isolates important functions as subroutines can be highly modular even though, for performance reasons, it isn't divided into separate physical modules or objects.

Adaptability is a key element of application modernization that builds on the concepts we've just defined: application partitioning and modularization. A successful application modernization project yields applications and skills that can adapt to changing requirements and technology as they evolve. Adaptability isn't a specific technique; it's a way of thinking about application development challenges so that the resulting application is more flexible. Throughout the report, we'll introduce tools and design concepts that help improve adaptability.

User interface redesign, especially the move towards graphical user interfaces (GUIs) and browser-based interfaces, is an important element of many application modernization projects. Most application modernization projects include some elements of interface redesign. In this context, the user interface encompasses more than the look of a screen (i.e., green screen vs. GUI). It includes the flow of user interactions (i.e., procedural vs. event driven) and even the distribution of an application across platforms (i.e., thin client vs. fat client).

Thin clients can be contrasted with *fat clients*. The description refers to the structure of the client component of a multitiered client/server application. The less application logic on the client platform

(typically a PC), the thinner the client. Generally, an application can be classified as a thin client if the only logic performed on the client is input-field editing.

Application Modernization Decision Points

Now that we have some basic definitions, let's take a high-level look at the process of application modernization. Although each AS/400 shop will choose its own route to application modernization, there are a few core issues that apply to nearly all AS/400 application modernization projects. For each of the decision points we've outlined, you will have to determine whether or not it applies to your application modernization plans and what actions are needed. In subsequent chapters, we'll explore many of these ideas in more detail.

Continued investment in RPG. Scare tactics notwithstanding, it's clear that for most AS/400 shops, RPG will continue to be a staple application development tool. If RPG is part of the development toolkit, it makes sense to move to RPG IV (ILE RPG) and take advantage of new opcodes, built-in functions, and design options. Because RPG IV is an ILE language, the language structure facilitates application partitioning and modularization. Any AS/400 RPG compiler licensed program product (LPP) at V3R1 or higher includes RPG IV. Although it takes some training before programmers experienced with RPG III can take full advantage of RPG IV, the initial transition presents a minimal learning curve.

Application partitioning. For long-term application modernization benefits, application partitioning is a critical step. Initially it may be difficult to justify the investment in application partitioning, and some shops will find it expedient to address pressing user needs first and return to application partitioning at a later time. This is an unavoidable situation, but it shouldn't be allowed to continue indefinitely. An investment in application partitioning can be

compared to an investment in machinery for your manufacturing line. Sometimes investments in machinery are forced by a single catastrophic breakdown or by demand for a new product that can't be manufactured with the old machinery. More often, there's a gradual awareness that the machinery isn't as efficient as it should be, maintenance is time consuming, and breakdowns are frequent. Similarly, an application that requires a major structural rewrite will continue to do its job, but it will be costly and difficult to repair, and it may not be possible to deliver new functions unless you make a major investment in application partitioning. Although the benefits of application partitioning may not be immediately obvious, in the long run it greatly reduces application complexity and makes applications much easier to modify and extend.

Web-enabled interfaces. There is enormous hype surrounding e-business and especially e-commerce; however, it's important to look beyond these obvious examples to understand the value of web-enabled applications to your customers and your company. Consider an intranet application that provides employees with information about benefits and company events. By using browser technology for the front end (the simplest definition of a web-enabled interface), the overhead involved in distributing and maintaining this human resources application is greatly reduced. Even shops that have no interest in "doing business on the Web" should explore the benefits of web-enabled applications.

Java. Java, like any other programming language makes it possible to achieve specific application development goals. Java's strengths are its object-oriented capabilities and its cross-platform portability. This means that Java helps enable certain types of application modernization projects. For example, server Java can provide a connection between a new front end built using Java or Domino and existing applications written in RPG.

Application Modernization Overview

Java components can be leveraged in many of the new business models, such as web-based customer service, that are emerging in the marketplace.

Among other things, Java gives you the option to create thin-client designs utilizing technology such as network computers. Although network computers have not achieved much success thus far, they offer potential. Network computers capitalize on Java's strengths and emphasize the necessity of building thin-client code.

Java also makes it possible to distribute an application across multiple servers in multiple locations. If you choose to include Java in your application modernization projects, you'll need to allocate significant resources to tools and training because, although Java offers great promise, it also presents a steep learning curve. Purchasing Java components can help you become productive with Java more quickly.

Justifying Application Modernization

There are many reasons why an AS/400 shop might embark on an application modernization project. It might be in direct response to a management directive to enable e-commerce or business intelligence functions. It might be a result of user pressure for GUIs. It might be an IT-initiated plan to implement Java. Most likely, however, a combination of these forces will converge, making application modernization practical.

One analogy that may help you convey the importance of application modernization is to compare investments in application modernization with investments in research and development (R&D). As most people realize, companies that don't invest in R&D may show short-term profits—even spectacular profits—but they are quickly left behind as others develop new products, new manufacturing techniques, and new marketing schemes. Thus, prudent companies always put R&D investments

high in their priority list, and when competition is fiercest, they increase rather than decrease R&D investments. The same principles should apply to your application modernization investments.

As with any IT project, the expense of application modernization must be justified in terms of both tangible and intangible benefits that are expected. The following list delineates some broad categories of benefits.

- *Reduce maintenance overhead.* Application partitioning, modularization, and the associated application development techniques have the potential to make the inevitable maintenance tasks more efficient because there is minimal (and clearly defined) interaction between program elements. New programming languages and programming tools such as application generators can also help reduce maintenance overhead.
- *Adapt to new technologies.* An application modernization project may be designed to implement a specific technology such as Java, but if the project is designed to allow flexibility, it will also be easier to incorporate other technologies as they become practical. For example, separating the user-interface logic from the business logic and the data maintenance portions of an application makes it easier to change the user interface as new technologies evolve.
- *Deliver new classes of applications.* This category of benefits includes applications that are difficult or impossible to create using traditional development methods. Examples include groupware and workflow applications developed using Domino, business intelligence applications that take advantage of new database capabilities and data mining tools, and web-enabled applications such as e-commerce that depend on Java.
- *Achieve platform independence.* Traditional AS/400 applications written with DDS and RPG

are inherently platform specific. Even those few other platforms that support RPG don't use it as a primary development language and don't have the extensions (e.g., support for interactive applications) that are available on the AS/400. New languages and tools, most notably Java and Domino, let AS/400 developers create applications that can run on a variety of platforms.

- *Improve ability to hire and retain skilled staff.* Anybody who has tried to hire AS/400 talent recently can attest to the acute shortage of trained personnel. The majority of colleges and universities do not teach RPG, but they *do* teach Java. Although Java is the most obvious example of this phenomenon, it also applies to other tools. If you choose Domino as one of your development tools, you can tap the pool of Lotus business partners for expertise. Of course, people with specific technical skills such as Java or Domino may not have any AS/400 skills, and they may have minimal knowledge of business practices in your industry. Switching from RPG to new development languages doesn't ensure an endless pool of trained talent, but it broadens the options somewhat. Another, perhaps even more important, element in this scenario is that it's much easier to attract and retain talent for a shop that's perceived as leading edge and willing to invest in training for personnel.
- *Enable distributed application deployment.* If the constraints of monolithic application design and platform-specific implementations are eliminated, applications can be distributed to servers based on user requirements and technical support logistics rather than being constrained by the application structure. Typically, distributed applications take advantage of a messaging infrastructure such as IBM's MQSeries to build platform-independent partitions. This structure may let you deploy portions of the application to an outside server (e.g., an ISP) even if you cannot control the server platform.

The benefits listed above apply in a broad sense to many application modernization projects. Specific business benefits will depend on the particulars of the project. A web-enabled order entry system might reduce data entry costs, increase sales by expanding the geographic reach of marketing efforts, and improve customer service by providing 24-hour ordering options. You may even find that you can leverage your new application design outside your own applications. For example, you might write a Domino application that sends an automated credit check via e-mail to a credit bureau and use MQSeries to asynchronously control completion of the transaction based on the credit report. ♦

Getting Started

Throughout an application modernization project, there's intense pressure to produce results. Users are eager to see the new interfaces, and programmers are eager to show their prowess with new tools. However, in order to achieve long-term success, an application modernization project may need to focus first on the application structure. Much of the existing code in AS/400 shops has its roots in S/36 or S/38 applications. The design and performance considerations that applied in those environments are not the most effective way to design applications in the current AS/400 environment because software advances and faster hardware eliminate many of the performance constraints that molded application design in the past.

If your AS/400 shop fits this profile, you will have to deal with two major issues as a prelude to specific application modernization deliverables: redesigning the application structure and equipping your developers with an appropriate set of skills and tools. It's possible to bypass these issues temporarily, for instance, by using screen scrapers to improve user interfaces or by installing a new application package. However, eventually, you'll have to address these two fundamental issues to lay the groundwork for all future application modernization.

Application Structure

Often it's necessary to create a new, more clearly defined structure so that an application can accommodate other kinds of changes. In some cases, it is efficient to develop this new structure concurrently

with changes that add new features and functions to applications. In many cases, however, a new application structure is the first goal of application modernization because modifying the architecture delivers intrinsic benefits such as better maintainability and the ability to use packaged components. Large, unwieldy programs are segmented into their logical components using application partitioning and modularization techniques. Although it's tempting to select an initial application modernization project that offers specific end-user benefits, in general it's better to complete application partitioning as a separate project before embarking on other application modernization projects.

The basic concept of application partitioning is deceptively simple; divide every application into three elementary elements: user interface, business logic, and database maintenance. Each element should have clearly defined interfaces to the other elements, and ideally, none of the partitions should require any knowledge of the other partitions. In a partitioned application, changing the structure of the database has no impact on the user interface and business logic portions of the application. Although this ideal isolation of each partition may be impractical, application partitioning *does* require a radically different method of designing applications compared to traditional RPG design methods, which assume a single job structure and shared information between programs.

In many cases, you'll find it helpful to further subdivide business logic into two components: public business logic and private business logic. Public

business logic includes support for answering a customer service inquiry, entering an order, displaying order status, and perhaps even checking inventory availability. Private business logic handles tasks such as calculating prices and commission payments. The reason for segregating business logic into two parts is to facilitate e-business. In this environment, there are usually some parts of the application that require 24X7 support and are subject to varying levels of use depending on the time of day, special promotions, etc. By segmenting the public business logic, you can potentially contract with an ISP to deploy the user interface and the public portion of the business logic. In this scenario, the ISP is responsible for network infrastructure, customer support, and ensuring sufficient server capacity to handle traffic at your web site. Capacity can fluctuate based on demand much more easily in this environment than if you must purchase additional equipment to handle temporary needs. The option to contract out a portion of your business logic is dependent on clearly defined application partitions and requires a platform-independent implementation for the public business logic.

Application partitioning owes its theoretical concepts to object-oriented programming. However, it's important to understand that, although application partitioning streamlines any future transition to object-oriented languages, such plans aren't the rationale for implementing application partitioning in the initial stages of application modernization. The reason application partitioning is so critical is that nobody can anticipate the user requirements, the technological solutions, or the changes to business processes that will emerge over the life of an application. By building a structure that is based on the idea that any element may be replaced, you'll minimize the overhead of adapting to future changes.

Conceptually, application partitioning is simple enough, but it's not always easy to see divisions between the components. In a traditional AS/400 application development environment, the user

interface is somewhat isolated from the other portions of the application through the use of DDS. You can make significant changes to screen designs without impacting the program logic at all—the program is simply recompiled to incorporate the new version of the display file. However, in this RPG and DDS development environment, the flow between screens is part of the program—not the DDS—and it's completely controlled by the programmer (procedural program flow).

Suppose that you want to change this application to use a GUI and an event-driven program flow. Your first approach to such a requirement might be to modify the screen design. With a screen-scraping tool such as Seagull's J Walk, you can quickly create a basic GUI that uses the existing 5250 data stream and leaves the application intact. This is not a bad short-term approach because it alleviates a lot of pressure from users. However in the long term, a GUI alone will not address this requirement because users expect a Windows look *and* feel, and the event-driven flow is central to the "feel" of a Windows application.

Many of the screen-scraping tools such as Jacada can optionally generate HTML or Java applets. As with the Windows GUI, you'll still need to modify the application to get a true browser flow. Browser-based interfaces use both a different graphical design and a different event flow than interfaces modeled on Windows. It's easier to modify a green-screen interface to create a browser-based interface than to create a Windows interface because green-screen designs and browser designs are actually quite similar at the architectural level. Both send and receive data in large chunks (block I/O). In addition, user expectations for browser interfaces are less complex than for a Windows interface. For example, there's no requirement to enable drag-and-drop between windows in a browser.

You may find that you don't have a very good design when you employ screen-scraping technology; it may

Getting Started

not meet user expectations and it may be difficult to maintain. This is not the fault of the tool but rather of limitations of the 5250 data stream and differences in the characteristics of a good green-screen design versus the characteristics of a good Windows GUI or web-enabled design. Understanding such differences is one of the skills that developers must acquire to become effective in these new application development environments. Screen scrapers can be an adequate solution for many interfaces, leaving you free to focus on those portions of the application that suffer from performance problems or are most critical to users.

Because of drawbacks to the initial screen-scraping solution, your next step may be to develop a completely new front end for your application using a tool such as IBM's VisualAge RPG, Microsoft's Visual Basic, or Symantec's Visual Café. The new front end may include event-driven flow for the user interface, which may in turn precipitate a need for new program flow within the business logic. Consequently, as you begin implementing new user interfaces, you must also begin breaking apart the structure of the existing RPG program. Ideally, you want to take advantage of this necessity to partition the application, opening up options for future modifications.

Many AS/400 shops are hesitant to adopt new user interfaces because of concerns about performance. While it's true that many client/server projects foundered over performance, an event-driven design shouldn't mean that the entire application must run on the client. The goal is to isolate the user interface flow from the business logic and database maintenance partitions so that each portion of the application can be written using the optimal tools and tuned for performance without impacting the remainder of the application.

Many client/server performance problems are essentially fat-client problems (too much of the application resides on the client); others are caused by excessive communications traffic between the

client and the server. Current business models offer design alternatives for client/server applications that address these performance bottlenecks. Alternatives include Java applets running on network computers (e.g., IBM's Network Station) and the option to retain business logic on the server (e.g., servlets). Such solutions are easier to implement because Java allows the logical components of an application to be shifted between the physical server tiers.

The concepts involved in application partitioning are difficult to define, difficult to teach, and difficult to learn. You should be prepared to invest in some false starts to acquire the experience necessary to design applications that use application partitioning effectively. Even once you've gained some experience, it may take more time to create a partitioned application, but the tradeoff is that modifications can be made more easily. We believe that the results justify investments in application partitioning.

Details Are Important

The need for flexibility applies to program structures at a more granular level than application partitioning as well. If an application is built out of many small, self-contained modules, it will accommodate changes more easily. Consider the following:

- Sales tax calculations are based on a complex formula that takes into account state taxes, county taxes, and city taxes. If customers live in another state, they are not subject to sales tax. However, if you maintain a retail outlet in the state where the customer resides, the customer is subject to the sales tax of his home state and, as a retailer, you must report (and pay) that sales tax.
- Periodically, the sales tax rates and calculations change. This is a minor inconvenience if you have isolated sales tax calculations in a separate module, but if you have multiple programs that calculate sales tax, it can be a major headache.

The sales tax module is only one small element of the business logic for an application such as order entry, but by isolating the code needed to calculate sales tax, you drastically reduce the maintenance overhead associated with changes to the sales tax law.

Isolating the sales tax logic has two benefits. If this logic is used in more than one application, by isolating the code in a callable module, you'll only have to change it in one place. Even if the sales tax calculation is only used in a single application (order entry in this example), isolating the code reduces the amount of testing required when a change is made to the sales tax rules. You can obtain some of these benefits by writing well-structured subroutines, but new structures such as service programs, procedures, and objects enforce the modular structure and reduce the maintenance overhead of inevitable changes.

In practice, modularization violates every design technique that AS/400 programmers have learned to optimize performance. As a result, modularization requires that developers rethink application design models and acquire some new skills. Although the benefits of modularization may not be immediately apparent in terms of new functions and it may even feel counterproductive initially, you will continue to reap the benefits of modularization every time you modify an application. Modular design makes each part of the application more accessible, reduces the impact of changes to any portion of the application, and therefore improves quality control, minimizes testing, and makes it easier to modify an application to incorporate a new technology or distribute the components across platforms. Given that at least 80 percent of all programming is maintenance rather than new development, investments in modularization and application partitioning should be easy to justify.

Tools of the Trade

In conjunction with investments in application partitioning and modularization, you'll need to invest in staff skills and application development tools. As we've already mentioned, new application and business models require new design skills. Often they also require new programming languages and development methods. It makes sense to invest in the tools and training that make your staff effective with these new methods immediately so that your application modernization projects yield results as quickly as possible. Conversely, like all aspects of application modernization, the investment in skills and tools is a continual process. As your application modernization projects evolve and new methodologies and products appear on the horizon, you'll want to add new elements to your application development toolkit.

The skills you'll need range from object-oriented programming to Internet architecture. Developers will also need to learn new ways to do things that they are very experienced with in a traditional AS/400 development environment. As we've already mentioned, the things that make a good green-screen user interface aren't necessarily appropriate for a GUI. This need for new ways of optimizing old tasks also applies to things like database design, application customization, and communications infrastructure. The basic concepts remain intact, but the methods for accomplishing various tasks change dramatically with new business models and technologies.

You may encounter some resistance to the idea that new skills are an important part of your application modernization project. Both management and the development team may feel that experienced programmers should be able to pick up any new techniques on their own. In fact at one level, experience can work against developers who are learning new skills because the design skills required for application modernization are significantly different than those used to create traditional AS/400 applications.

Getting Started

We don't mean to imply that prior application development experience isn't valuable when learning new application development methods. There's no substitute for your staff's knowledge about your business and the understanding they have about programming logic and database design. All of these assets greatly improve their ability to learn new programming skills. However, in order to use the new tools and methods effectively, developers must let go of some tools and methods that have served them well in the past, and this is always a difficult thing to do.

The vast majority of AS/400 shops are still developing RPG applications using SEU. It's easy for a programmer to see the benefits of a graphical editor such as CODE/400 or a complete integrated development environment (IDE) such as VisualAge for Java. However, it may not be so easy to see the benefits of object-oriented design, and invariably there's a shortage of time. Often new tools are rejected or abandoned because developers cannot allocate enough time to learn to use them effectively. Consequently, it's critical to plan for short-term productivity reductions and to provide training and mentoring so that the new tools and techniques become productive quickly. Otherwise, everyone will become frustrated with the new methods and abandon them before they become really useful.

In addition to productivity enhancements, some tools provide an added benefit—they insulate developers from the moving target of application development trends. Visual builders, such as Inprise's JBuilder and IBM's VisualAge for Java, and application generators, such as LANSA and Magic/400, do a good job of separating the developer from the code. Using a variety of methods, these tools let you create an application by defining a set of business rules that is used to generate an executable application. In the past, the code generated might have been RPG IV or ILE COBOL, today it may be Java, and in the future...Who knows?

Along with tool investments, make sure you designate an adequate budget to upgrade programmers' PCs, the network infrastructure, and when you're ready to roll out your new applications, end-user workstations. It won't do much good to create a web-based quote system for outside salespeople if they don't have laptops or Internet connections.

Take Advantage of Mass Production

In addition to the products that you might traditionally think of as application development tools (source code editors, debuggers, application generators, report writers, etc.), you can leverage packaged software, such as an accounting application or an ERP solution, to streamline your application modernization projects.

If you're using a custom software solution and have had your current applications for some time, it makes sense to revisit the packaged software options that are available for your industry. You may find that a packaged solution is now available to meet your needs.

Even if you can't purchase a complete package, you may find that a new application structure makes it possible to use some software components. A *component* performs a specific function using a published interface. The idea is that anybody who needs that particular function can purchase the component and plug it into custom applications. Purchasing a sales tax component from an ISV and interfacing it with order entry could solve the sales tax problem we discussed earlier in this chapter.

In object-oriented parlance, groups of components can be purchased as *frameworks*. IBM's SanFrancisco project includes frameworks, written in Java, to handle standard system-management functions such as security, and business functions such as customer entry or general ledger posting.

Object technology (e.g., Java) makes the use of components particularly easy, but you'll find that a partitioned, modular application can accommodate many kinds of components. For example, our sales tax component could easily be written as an RPG IV procedure.

A View from 30,000 Feet

Because application modernization is an ongoing process that must adapt to a changing environment, you cannot specify an ultimate destination. By the time you reach the first milestone, the landscape ahead may be radically different than it is today. Although the broad concept of application modernization can't be clearly defined in terms of milestones to be reached or tasks to be accomplished, many AS/400 application modernization projects share some basic goals and intermediate checkpoints. There may be some variance in the order in which these landmarks are reached, but generally speaking, any AS/400 shop starting from a base of green-screen RPG code will have to pass through similar checkpoints. In the following chapters, we'll look at some of the checkpoints you're likely to encounter and at the tools and techniques that are applicable to each one. ♦

Defining Requirements

Application modernization is an ongoing process driven by changing business demands; therefore, defining application specifications for application modernization projects is an ongoing process as well. Just as an R&D effort never stops, a policy of application modernization will inevitably result in an ongoing, constantly evolving set of application modernization projects. An R&D team is constantly moving some ideas into production, rejecting others as not being commercially viable, and starting projects to explore new ideas. Likewise, an application modernization team is constantly finding new projects. What was “modern” yesterday (or even this morning) needs updating this afternoon.

Just as any business investment requires an assessment of the potential financial risks, your application modernization policy should be based on an assessment of potential risks. If you adopt leading-edge technology, it may give you an edge over your competitors (you may be able to offer e-business options before others in your industry), but you also run all the inherent risks: changing standards, performance problems, the need to rewrite applications as newer technologies emerge, and high startup costs.

For any particular project or group of projects, it's important to define specific, measurable goals. In this chapter, we'll look at some goals that are common to many application modernization projects and provide insights that will help you define your own projects.

Evaluating the Status Quo

In most cases you'll find that there's a lot of value in your existing application portfolio, and you'll want to preserve those applications that are working well along with significant portions of many other applications. It's highly unlikely that you'll jettison your DB2/400 database in your initial application modernization projects. Likewise, you should be able to retain much of the business logic in your existing RPG programs. The trick to this process is to subdivide application modernization into a series of manageable tasks that can be tackled independently.

Before you can map out a path for application modernization, you have to take a serious look at where you are today. This appraisal process applies to applications and to the skills and tools used to develop code, maintain existing programs, and modernize applications. In the next chapter, we'll examine skills and development tools, but first let's focus on the applications themselves.

Your first task is to catalog existing applications and make a series of judgements about their viability. You may even find that there are some applications that have no documentation and whose purpose nobody clearly understands.

To compare the relative viability of your applications, you can create a spreadsheet with a row for each major application and columns for measures of viability. Potential column headings include: Outstanding user requests, Ease of maintenance,

Percentage of dead code (i.e., functions that are no longer used or have been commented out of the application), Number of known bugs, Performance, End-user satisfaction, Indicator usage, and Levels of modularization and application partitioning.

You'll also want to include measurements of how important (i.e., mission critical) the application is, how volatile it is (i.e., how often it's changed), and how well it's understood by your current staff. You can use this spreadsheet to evaluate which applications are in most urgent need of modernization and as an ongoing status report on application modernization. This type of evaluation lets you judge, not just which wheel is squeaking the loudest, but also which one is actually in danger of breaking.

You should be as brutally honest in this process as possible. Older applications often ignored modularization and partitioning in order to deliver acceptable performance. However, you may also have newer client/server applications that were blithely designed with fat clients and little regard for performance. Either type of application is an excellent candidate for modernization. There may also be requests for completely new applications due to emerging technologies and the changes they cause in the business climate. Such projects are also part of the overall scope of application modernization.

Once you have a clear picture of your existing application base and current requests, you can begin prioritizing application modernization projects. It's advantageous to preserve your investment in applications whenever possible. In most cases, you won't be discarding all your existing applications; instead, you want to salvage anything that's useful and build a new structure without disrupting existing applications. Like a road-widening project, your application modernization may cause some disruptions and slowdowns, but generally, it isn't practical to shut down traffic completely. You may have to time modifications to avoid peak business seasons,

and you can't afford to disrupt critical business processes such as payroll or order entry. This may mean that a single application must be modernized in stages or that you'll need to sequence modernization projects one application at a time. Either approach is fine; what's important here is to avoid a massive project without any measurable success. It's also important to communicate your plans and progress to end users and management so that they become active participants in the application modernization process.

Eye on the Prize

While the overall goals for application modernization may be a bit nebulous, the goals for any particular project should be as specific and measurable as possible. It's important to recognize that even the most technical goal must have its roots in business benefits in order to justify your application modernization efforts. However, when it comes to measuring progress and success, it's helpful to make the goals as concrete as possible and this often leads to isolating specific technical accomplishments as measurements.

Application modernization goals may include technical measures such as the percentage of code written in RPG IV or Java, the speed of deployment to a new platform, and the elimination of left-hand indicators in RPG programs. All these goals can be measured objectively, a valuable quality in the process of evaluating your application modernization progress. You'll also have goals that are less quantifiable such as achieving a more modular programming style, assimilating new technology more quickly, and leveraging distributed application technologies.

Looking at goals from another perspective, it's possible to measure things, which currently don't exist, that an application modernization project will deliver. Application modernization goals that fit this criteria

Defining Requirements

include deploying applications to an intranet or the Internet, implementing e-commerce, and creating a data warehouse from a production database.

All of these goals should be explicitly correlated with the business benefits that drive the process. If you expect to improve data entry efficiency, you should measure the current number of orders processed per hour so that you can compare the old system with the new system. Whenever possible, you should also measure less tangible factors such as customer satisfaction. Perhaps you track the percentage of returned merchandise and capture the reasons for each return. If the percentage of orders returned for “wrong item” goes down, you’re on legitimate ground if you claim that the new system improves customer satisfaction.

The process of defining requirements for an application modernization project is no more complex than defining requirements for any other application development project. However, application modernization projects are more likely to change during the course of development than most other programming projects. Therefore, you’ll want to include regular project reassessments as part of the development process. You can use your application viability spreadsheet as a starting point and plan regular assessments on a monthly or quarterly basis. ♦

Acquiring Skills and Development Tools

Of all the investments you'll make in application modernization, the investment in training and tools for your development team is the most critical. Although you may solve specific application problems by purchasing a new software package or hiring a consultant, if you want to change your IT environment, it's ultimately up to your programming team. Investments in the best possible development team and the things that make them more productive are investments in the future of your organization.

Although it's true that some people may take new skills and use them as leverage for a higher salary elsewhere, in general, people recognize the value of an employer who's willing to invest in their education and training. Moreover, a good programmer—like any other craftsman—takes pride in his work and feels that it is not properly valued if he isn't given the tools, training, and work environment to do his job as efficiently as possible. Failure to invest in your development team will probably cost you more talent than you'll ever lose to the lure of higher pay for new skills.

The decisions concerning what types of skills and tools are needed are critical to the success of your application modernization projects. Most of the material you've read so far provides general guidelines about application modernization and several key elements such as application partitioning. In this chapter, we take a more detailed look at some of the tools and skills that facilitate application modernization. This information is inherently incomplete; the AS/400 tools marketplace is evolving so rapidly that even in the course of researching this

report we found cases where a company had acquired an entirely new product line and shifted its focus substantially between our initial investigation and the write-up. To address this problem we have used tools as illustrations rather than trying to provide you with a comprehensive list of all the tools in a particular category.

Once you understand the value of a particular type of tool, we suggest you use the web sites referenced in "For More Information" to research the current options and select the best tool for your particular needs. We're aware that many people are dissatisfied with this approach. In the "good old days," IBM told you precisely what tools to use, who to purchase them from, and how to get appropriate training. This model, for all its arbitrariness, was amazingly effective for many years in the IBM midrange environment. However, it's almost impossible to provide that level of guidance in today's AS/400 environment for several reasons.

First, there are many more types of applications (e.g., e-business and business intelligence), and the tools that are best for one type of application may not be appropriate for other applications.

Depending on your mix of applications, you may be able to select a few tools that will be effective across your application set—a sort of adjustable wrench approach to the problem. There are few jobs for which an adjustable wrench is the perfect tool, but it does many different jobs effectively. If you only have a few tools in your toolkit, it's important that they be very versatile. Conversely, there are jobs

 Acquiring Skills and Development Tools

that cannot be accomplished without precisely the right tool—a torque wrench for example. If you don't have the tool, you can't do the job.

Similar rules apply to programming tools and the associated skills. If you want to develop a workflow project such as insurance claim processing or employee reviews, there are only a few AS/400 tools available. The two options that we're aware of are Domino for AS/400 and jFlow/400 from Workflow

Automation Corporation. On the other hand, if you're creating an e-commerce application with an AS/400 database, you have many choices. IBM offers Domino, AS/400 Toolbox for Java, Net.Commerce, HTML and Net.Data with RPG back-end processing, and VisualAge RPG. Other vendors also provide many tools for AS/400 e-commerce application development. The sidebar, "The Road to e-business," gives you a glimpse of the solutions that the IBM Software Group recommends.

The Road to e-business
 by David Slater, Market Manager,
 AS/400 Application Development Products; IBM Software Group

Editor's note: On the subject of e-business, IBM's message is remarkably consistent: e-business is the future of computing. However, even in this arena, different business units have different strategies for how the goal should be achieved. To give you one of these perspectives, we've invited David Slater to share his views concerning AS/400 application development tools and how they facilitate the transition to e-business.

Application modernization is extremely important to IBM in the context of its e-business strategy. IBM believes that e-business is the future and that Java is the language of e-business. E-business incorporates the advantages of client/server applications while addressing many of the problems associated with deployment of these applications. In an e-business environment, client function can be downloaded as an applet (a client application that executes in a browser) or customized HTML generated by a servlet or server code. Many of the systems management problems associated with client/server are minimized because client code is distributed only when it's needed. In addition, e-business uses Internet-based communications, greatly reducing communications costs, especially for a geographically dispersed set of clients.

The AS/400's machine-independent architecture and its performance-optimized JVM are key capabilities in making the AS/400 an industry leading e-business server. The D.H. Andrews Group report, *Java and AS/400 – Perfect Together*, documents the superior capabilities of the AS/400 as an e-business server.

To realize its potential as an e-business server, the AS/400 needs more web-enabled applications. There are only three sources for these web-enabled applications:

1. Newly written e-business applications.
2. E-business applications ported from competitive platforms.
3. AS/400 applications that have been web-enabled.

IBM has developed its AS/400 VisualAge tools—VisualAge for Java and VisualAge RPG—to help AS/400 customers and ISVs contribute to the pool of e-business applications.

Smoothing the Path

VisualAge for Java is IBM's Integrated Development Environment (IDE) for editing, compiling, and debugging Java programs. This tool is ideal for porting Java applications to AS/400, for creating new Java applications for AS/400, and for creating Java GUIs that interact with AS/400 applications. VisualAge for Java yields all of the benefits associated with object-oriented (OO) programming as well as the "write once, run anywhere" capabilities of Java. Java satisfies the requirements for cross-platform and distributed computing, and of course, it is the current language of choice for e-business and the Web.

VisualAge RPG is the recommended tool for web-enabling existing RPG applications. It is an integrated development environment for editing, compiling, and debugging RPG programs that execute on the client. In the latest release of VisualAge RPG, IBM has added the ability to generate Java applications and Java applets from VisualAge RPG source code simply by changing the build options for the application. Because you are generating Java source from VisualAge RPG source, you can compare the RPG and

(continued)

The second challenge in making recommendations about tools is that you need a set of tools that work together. For example, choosing a development tool for your e-commerce application isn't enough. You'll also need an infrastructure that includes a web server such as WebSphere and perhaps messaging software such as MQSeries. The wrong combination can be as frustrating as having a set of drill bits that don't fit your electric drill.

These first two challenges can be met to some extent by providing a matrix of tools that have varying characteristics. You'll find such listings referenced in "For More Information" at the end of this report. However, selecting a development tool is like choosing a piece of sporting equipment: the right solution depends on many factors and your personal preferences. For example, when you choose a bicycle it has to be the right size, but it also has to fit your budget and your riding style—road rider or mountain

Java source to help you learn the relationship between RPG and Java language constructs.

VisualAge RPG is easy to learn and easy to use. The VisualAge RPG compiler has been enhanced to include GUI capabilities, but the ILE RPG compiler and the VisualAge for RPG compiler have the same base code. It is easy to move code from host to client and back again. This makes it possible to reuse 80 to 90 percent of your host code, and the compatibility between host and client compilers makes the product easy to learn. With two days of training and two weeks of practice, an experienced RPG programmer will be effective using VisualAge RPG.

The most difficult concept to master when you are moving applications from the AS/400 to VisualAge RPG is the change from a procedural, or process model, of application development to an event-driven model. This change to an event-driven model yields several benefits. The resulting VisualAge RPG application has point-and-click navigational characteristics associated with Windows applications. Moving to an event-driven model also helps you make your code more modular and therefore easier to maintain.

Developing and distributing VisualAge RPG applications is very cost effective. There is no runtime license for VisualAge RPG; there is only a development license. VisualAge RPG and the generated Java applications will run on any system after V3R2, and the generated Java applets are compatible with V4R2 and higher.

A Head Start

If the future of application development is e-business and Java is the language of e-business, why would AS/400 developers choose VisualAge RPG to web-enable their applications? Why wouldn't they move directly to VisualAge for Java? There are several reasons.

- *Speed of deployment of e-business applications.* You can become proficient with VisualAge RPG in two weeks. You can learn Java syntax in 21 days (at least that's what the book claims), but Java is an OO language and you will not become proficient in using Java to create applications for at least 6 to 12 months.
- *Leverage.* Using VisualAge RPG will enable you to leverage and extend your investment in RPG applications and RPG-trained programmers. You will be able to participate in the world of e-business as you make your investments in developing Java skills.
- *Evolution.* Future releases of VisualAge RPG will focus on interoperability with Java and WebSphere. This will allow you to move to Java in an evolutionary fashion, which is much more productive and less disruptive than a sudden change in development tools and processes.

Conclusions

The demand for e-business applications is one of the prime drivers for AS/400 application modernization. AS/400 has the technical capabilities to be a premier e-business server, and there is an ever-increasing demand for web-enabled applications, but time is of the essence. The technical lead that AS/400 has established over its competition will not last long. To capitalize on its capabilities and realize its potential as an e-business server, the AS/400 requires a wealth of e-business applications.

If the demand for e-business applications is met, the AS/400 will assume its proper place as a key strategic e-business server and banish forever the notion that it is an antiquated, proprietary host for green-screen applications...and the market will reward the solution providers who satisfy this demand. ♦

Acquiring Skills and Development Tools

biker, cross-country or downhill. Similarly, you have to make qualitative judgements about application development tools that include factors such as the vendor's reputation in the marketplace, recommendations from colleagues, preferences among the programming staff, and price. The price includes not just the purchase price but ongoing costs such as training, maintenance, and runtime support. Realistically, with all the choices currently available, nobody can make decisions about application development tools for you. What we can do is show you the kinds of tools available and explain the types of environments for which they are most useful.

We've classified tools by function: user interface, business logic, and database logic. In the following sections, we discuss representative tools for each category. We caution you again that this is not a comprehensive list but rather a sampling designed to help you understand the available options

User Interface Tools

The user interface tools are probably among the most familiar to you, but they are also the part of the toolset that's changing most rapidly. The tools in this category include simple cosmetic solutions that put a web browser or Windows GUI face on a 5250 data stream. Although this category can encompass more complex solutions that generate application logic, generally when we talk about user interface tools, we focus on the screen design and the ability to generate HTML pages and Java applets.

J Walk (Seagull)

J Walk lets you develop web-enabled interfaces as well as capitalize on existing screen designs and processing logic. The Collector component of J Walk processes DDS and RPG source code to build a repository of information about existing screens. You use a point-and-click interface to select elements from this repository (known as the Picture Album) to create Java clients. The resulting thin clients can be

launched from a web browser, making them appropriate for e-business and network computer environments. There is also the option to integrate personal productivity applications (e.g., e-mail) and programs written in Visual Basic, Java, C++, etc. With announced extensions to developer and server functions, J Walk provides the ability to compose back-end business logic using Enterprise JavaBeans (EJB) and Extended Markup Language (XML) wrappers. This capability provides the logic to integrate web pages with RPG, COBOL, Domino, or Java.

Jacada

Jacada can be used as a simple screen scraper, but it also can be used to augment and redesign user interfaces. Jacada builds a client interface written in Visual Basic or Java. Either client can be installed on a Windows desktop as an application, and the Java client can also be accessed through a browser in the form of an applet. Jacada lets you combine multiple screens into one, control navigation, and add functionality to the generated interface. Custom modifications are stored separately and are reapplied to the resulting interface if the client interface is regenerated.

An important feature of Jacada is a knowledge base of the screen design rules that can be used to create and enforce screen standards. The knowledge base can be created manually based on analyzing the 5250 data streams as application panels are viewed, or it can be populated based on DDS and modified as needed. Additionally, Jacada now includes a Java-based server that runs on either the AS/400 or an intermediate system that can be used to create a batch interface to the application.

VisualAge for Java (IBM)

VisualAge for Java is a comprehensive development environment, but we've included it here as an example of a client-based, user-interface development tool. With VisualAge for Java, you drag and drop visual components on the screen and use a graphical interface to specify screen attributes, application flow

(e.g., the actions that take place when an item is selected), and interfaces to business logic and database maintenance components (also developed using VisualAge for Java). For more information, see the D.H. Andrews Group report, *Making Java Smoother and Easier: IBM's VisualAge For Java*.

VisualAge RPG (IBM)

VisualAge RPG is a tool that can be used to modernize user interfaces. Because VisualAge RPG is a programming language, it can be used to develop an entire application. However, you might first apply it to create a GUI event-driven front end for an existing AS/400 application. VisualAge RPG includes a conversion tool that creates a basic GUI from DDS. The resulting screens can then be edited to make them more compatible with GUI standards. In addition, the VisualAge RPG language supports extensions to RPG IV that make it possible to incorporate event-driven logic into applications. Finally, VisualAge RPG can be used to generate Java applets and applications, providing an RPG interface to Java.

Business Logic Tools

Many tools cross the borders between application partitions and provide the ability to modernize two or more parts of an application. In this section, we'll focus on tools that can be used to generate programs based on a set of business rules, a class of products generically known as *application generators*. Technically, any programming language falls into this category, but there are additional benefits when the tool gives you a level of abstraction from the code. One of the benefits of such abstraction is that the tool vendors, rather than your developers, are responsible for adapting to changing technology standards and integrating that support into the tool and possibly the runtime environment. Tools that generate Java today generated RPG or COBOL in the past, and in the future these tools may be able to use the same specifications to generate some other language.

You should be aware of two major drawbacks to the application generator approach. First, you are at the mercy of the vendor's response to technology trends. If you think it's time to jump on the Java bandwagon and the vendor does not, you may be stuck because of your investment in a particular tool. In your assessment of any tool, include the vendor's current plans and its track record for implementing new technology. Second, you should never expect to create all your applications using an application generator. Like any automated process, there's a level of fine-tuning that cannot be achieved with a generated application. Problems may surface with performance or with the ability to create a particular function. Instead of battling this trend, you are better off supplementing any application generator with other tools.

In this section we look at six different application generators: COOL:Plex, GeneXus, LANSA, Magic/400, mrc-Productivity Series, and VisualAge Generator. All of these tools claim to provide rapid application development (RAD) and to be repository based. However, such terms can mean different things to different vendors. The goal of these tools is to capture business rules, data specifications, and user-interface standards and to use them to generate applications. The results, learning curves, and degree of interoperability with RPG programs vary widely.

COOL:Plex (Sterling Software)

COOL:Plex (formerly Obsydian) is a model-based application generator that uses object-like constructs. The business models are used to generate RPG, COBOL, or Java for server code. On the client side, COOL:Plex generates C++ and Java and creates .exe and .DLL files. It is the most complex of the application generators we've included here, and it's also among the most robust. In addition, packaged applications written using COOL:Plex are available for a number of industries such as manufacturing, health care, and distribution. Being so complex, COOL:Plex has a significant learning curve; approximately three months of training are needed.

Acquiring Skills and Development Tools

GeneXus

GeneXus is a sophisticated tool that starts by collecting descriptions of all the objects managed by users such as invoices, products, clients, etc. It uses these descriptions to generate the database including a data warehousing solution, applications, and a set of entity-relationship diagrams that display objects graphically. GeneXus simplifies analysis of how a change to one user object affects the rest of the system and it can automate ongoing maintenance. Application designs are totally platform independent, insulating the developer from technology changes. GeneXus was one of the earliest tools to step up to generating 100% pure Java.

LANSa

LANSa provides data modeling, programmable templates, a 4GL, report generator functions, and the ability to interoperate with existing RPG applications. It is part of a family of products, which includes tools to generate RPG, C++, Java, HTML, and JavaScript. LANSa optionally generates RPG or Java and can integrate with IBM's San Francisco frameworks.

Magic/400 (Magic Software)

Magic/400 uses a table-driven interface to create a repository of business rules. An application is defined by entering values in tables for data definition and application logic. Magic uses these values to generate the application that can coexist with RPG applications on the AS/400. (The Magic engine is also available for many other platforms.) Magic/400 benefits include flexible messaging, integration with middleware, and the ability to generate both client/server and web-based user interfaces. There are limited interfaces to other third-party tools and functions.

mrc-Productivity Series (Michaels, Ross & Cole)

mrc-Productivity Series, the most AS/400-centric of the tools we've reviewed here, uses an open template technology that looks similar to RPG. It can generate RPG IV server code and Java or Visual Basic client code, and in addition, it can import existing RPG and DDS. mrc-Productivity Series

interfaces with many third-party tools and application packages. In the future, support for Java server code will be added.

VisualAge Generator (IBM)

VisualAge Generator is a cross-platform application development environment that combines visual programming with templates to generate code. The templates and component parts may be supplied by IBM or third-party vendors, or they may be custom built. You can use VisualAge Generator to manipulate data, present data to end users, manage navigation among multiple windows, and provide online help. Data can be retrieved using SQL or direct DB2/400 record-level access. On the client side, VisualAge Generator supports 5250, GUI, HTML, and Java. AS/400 server support is currently limited to ILE COBOL. VisualAge Generator's strength is its cross-platform support, which includes AIX, HP/UX, VSE, VM, and MVS, in addition to OS/400. VisualAge Generator's major drawback is complexity: there are many different ways to accomplish each task, which may increase the amount of training that will be required before developers can use this product effectively.

Database Maintenance

In the database maintenance partition, our discussion focuses on DB2/400 functions rather than on specific tools. Beginning in V3R1, DB2/400 is enhanced with stored procedures, triggers, and constraints. These database functions make it easier to isolate database maintenance from the business logic and user interface partitions of an application.

Implementing Business Rules in DB2/400

A *stored procedure* is SQL terminology for a called program. On the AS/400, any program object can be cataloged as a stored procedure. Stored procedures give you the option to partition applications without completely rewriting the database maintenance portion. You can retain back-end processing with minor

modifications and replace the user interface with newer GUI and web-based solutions. The front-end tools typically use SQL syntax for database access so that you can call your back-end RPG programs using an SQL stored procedure call.

Triggers are programs that are automatically called (fired) in response to specific database events (i.e., add, update, or delete of a record in a physical file). Being programs, there are very few limitations on what a trigger can do. The value of triggers is that, because they are part of the database definition, they cannot be bypassed. If a trigger is assigned to an update event for a file, it will be fired every time a record in the file is updated whether via an RPG program, a utility, or a client/server application. Triggers are a simple way to begin partitioning applications.

Constraints are the most complex of the new database functions. Many constraint functions were already available within DDS, but in order to give them an industry-standard implementation, they have been implemented as constraints using SQL. Constraints can also be implemented using CL, but unfortunately, there's no DDS interface (part of the ongoing trend away from DDS support and towards SQL-based database functions on the AS/400). Whether or not we agree with IBM's strategy, the bottom line is that developers should consider learning SQL, both to take advantage of new database functions and to help them use other tools, such as query tools for data warehouses, more effectively.

Constraints don't do anything that cannot be done with application programs; however, they do insulate data integrity tests from the application code and thus improve application partitioning. By defining the rules at the database level, you can ensure that they will be enforced. However, you should be aware that implementing constraints is not a straightforward proposition. There is significant difference of opinion in the AS/400 development community about the best way to handle constraint violations.

Some people suggest comparing constraints with duplicate key rules. They point out that although DDS is used to specify unique keys, developers still write application code to prevent duplicate key errors from surfacing. On the other side of this debate are developers who prefer to discard the old database validations in application code, move business rules to the database using constraints, and add constraint error handling to applications as needed. They point out that the value of constraints is that they don't have to be implemented in every application program that can affect a file. Opponents argue that including constraint error handling in every application program is simply trading one type of error handling for another.

As of V4R4, DB2/400 supports four types of constraints: primary key constraints, unique constraints, referential constraints, and check constraints.

- *Primary key constraints* and *unique constraints* are industry-standard methods for defining the key to a file. Typically, you'd use a primary key constraint in the same situations that call for a uniquely keyed physical file. Unique constraints are used for alternate, but still unique keys (e.g., an employee master file that has both employee number and social security number keys).
- A *referential constraint* defines the relationship between a parent file and a dependent file. Using a referential constraint, you can define a business rule that requires a valid customer number in the order header file. The customer master is the parent file, and the order header is the dependent file in this scenario.
- *Check constraints* are field-level constraints that test for compliance with simple rules such as "start month equals 01-12," "ship via is not equal to blanks," or "maximum balance less than or equal to credit limit" (both fields must be in the same record). The V4R4 announcement includes plans to allow user-defined functions in DB2/400,

Acquiring Skills and Development Tools

thus greatly expanding the potential of check constraints. This support is expected to begin shipping in November 1999.

Database Access Standards in DB2/400

In addition to tools that help isolate database functions, OS/400 supports database access standards that are widely used on other platforms. By implementing standard database access methods in DB2/400, IBM opens the database to applications developed using any technology. From the developer's standpoint, implementing database access using standards such as ODBC and JDBC makes it possible to redeploy an application to a different platform or across multiple platforms. Although these database access methods are not as efficient as native AS/400 database access (e.g., from within an RPG program), they make it possible to provide an interface between applications running on or written for other platforms (e.g., client/server applications) and DB2/400. For example, a Java program that uses the JDBC standard can access DB2/400 simply by specifying the correct JDBC class library. If you take advantage of record-level DB2/400 access classes provided in the AS/400 Toolbox for Java, you may get better performance at the expense of a non-standard interface that must be modified to port the Java application to another platform.

Look for tools that implement industry standards in ways that give you the flexibility to store your data on whichever platform is appropriate. For AS/400 shops, this means that the tool should include or be able to use a driver that feeds data requests to DB2/400 and returns results. For example, Java-based tools should include the ability to use JDBC classes, and client/server tools should include an ODBC driver or the ability to use one of the available ODBC drivers such as the one provided with Client Access.

Database Essentials (Centerfield Technology)

Just as there aren't really tools to implement application partitioning, you won't find many products

that are devoted to database maintenance. One that's representative of the products you may find useful is Database (DBA) Essentials. This product helps you control the use and performance of SQL queries on the AS/400 by analyzing the SQL statements generated by packaged solutions as well as the SQL statements coded by your programmers. DBA Essentials addresses SQL problems from three angles: database administration including index and performance management, SQL usage analysis and control (i.e., limit user access during critical performance windows), and development tools including SQL statement optimization and SQL impact analysis. DBA Essentials is integrated with the AS/400 DB2 query optimizer and the AS/400 DB2 monitor commands. It works for any SQL DB2/400 environment including ODBC, JDBC, DRDA, and embedded or dynamic SQL programs.

Business Intelligence

If you're interested in business intelligence applications, you'll also want to investigate tools for replicating data and building data marts and data warehouses as well as tools that can analyze and present this data such as data mining solutions and query products. Although we haven't dealt extensively with business intelligence in this report, it offers many application modernization opportunities. To learn more about such tools, consult the web sites listed in "For More Information" on page 36.

Building Application Development Skills

Thus far we've discussed tools that can be used to facilitate application development. Often these tools require specific training; in the case of the more complex application generators, it can take several months to get started. In addition to the tool-specific training, a transition to new application development methods requires some underlying skills, possibly even entire job descriptions that don't currently exist in your AS/400 shop. Incorporating database rules into the database may require hiring or train-

ing a database administrator—a job title that’s rare in AS/400 shops today. It’s likely that you’ll also need people with skills in web design, graphic arts, and other disciplines that are foreign to traditional programming skills. Finally, system management becomes much more complex as you add multiple tiers, multiple platforms, and multiple development methodologies. To address these systems management requirements, you’ll need experts in network administration, team programming methods, and change management tools.

As mentioned previously, GUI design and event-driven programming logic are significantly different from green-screen design and procedural programming logic. In time, your developers will become adept at these tasks, but the more training you provide them, the easier the transition will be. With a minimal investment, you can get the training process started. By purchasing one of the popular PC development tools, you can provide your programmers opportunities to experiment with these two important skills. You’ll also find a good collection of literature about such products at any bookstore.

Yet another category of skills that requires investment is Java programming and the associated (and far more difficult) object-oriented (OO) design topics. Don’t expect these skills to be learned by osmosis. You’ll need to invest in formal training and mentoring to gain the Java and OO skills you need. Most of all, your developers will need time to assimilate the new skills. In this sense, training new Java programmers—even those with existing programming skills—is similar to training junior programming staff. Programmers will learn syntax and specific techniques relatively easily, but they’ll have to work through several small projects before they become proficient with Java. It takes time for new ways of looking at problems to infiltrate a developer’s thinking. This is one of the important reasons for a mentoring process: it gives your programmers access to expertise and models of how OO applications are designed effectively.

In the process of acquiring Java and OO design skills, you should consider the type of application development environment you anticipate; Java developers can be roughly divided into those who create components and those who assemble components. Although your environment will probably include some of each type of development, as more components become available, it becomes possible to build Java applications entirely from preexisting parts. This approach requires a minimal knowledge of Java syntax, but understanding OO design methodologies and your business requirements are both critical skills.

Some of the other skills that are needed include SQL, performance analysis and tuning, TCP/IP, and HTML. Finally, you’ll need to allocate resources for the continual evaluation of new development tools and methods.

We don’t dwell extensively on how to teach AS/400 developers new skills because you’ll find plenty of discussion elsewhere. However, it’s critical to incorporate training into your application modernization plans on an ongoing basis. ♦

The Road Ahead

The majority of this report focuses on the initial transition from traditional, green-screen, RPG applications to more modern user interfaces and new types of applications such as e-business, workflow, and business intelligence. In this chapter, we provide a glimpse of one of the next stages of the journey and the decisions you'll face if you begin transitioning to Java-based development.

While Java currently appears to be the clear path for the future of application development, the entire landscape can change very quickly. For this discussion, we assume that you have chosen Java as your server development tool, and we ignore the impact of user-interface choices on server Java implementations. Our goal is to give you a glimpse of the application modernization decisions that lie beyond the scope of this report. With these caveats in mind, let's look at some of the issues you'll face as you expand your application modernization projects to use Java more effectively.

Implementation Issues

One of the driving factors for application modernization is the desire to leverage the Internet (or more specifically, the Worldwide Web) to create new business opportunities. Here are some of the things you can do with today's Java technology:

- Develop Java applications that coexist and interact with RPG applications.
- Incorporate information from DB2/400 and AS/400 applications into web pages.

- Integrate Domino applications and data with AS/400 applications and data.
- Make applications accessible to a broader range of workstation devices including network computers and remote users via web browsers.

In order to implement these technologies, you must make decisions about how to deploy Java and build an appropriate infrastructure. Developers whose experience is limited to green-screen AS/400 development are often unfamiliar with the infrastructure requirements because, in that traditional AS/400 environment, the entire infrastructure is provided in OS/400. However, in any multiplatform application—client/server or web-based—tools are required to communicate between the application components.

Infrastructure tools are often referred to as middleware, plumbing, or glue. They provide communications between application partitions, platforms, and databases. These tools include application (or web) servers, such as WebLogic and WebSphere, and messaging software, such as MQSeries.

Application Servers

Application servers are a relatively new concept and the definition of an application server is not entirely consistent throughout the IT industry. In this report, we define an *application server* as a middleware product that simplifies the deployment of Java applications on a server by managing server resources such as database connections and threads. Application servers are typically deployed on the same

server as the public portion of the business logic. They also manage the communication between client applications and Java programs running on the server. Application servers usually contain an HTTP server. Many of them also include the option to plug in a separate HTTP server such as IBM's HTTP server or the Apache HTTP server.

The server Java code is written in the form of a servlet or a Java server page that is associated with an HTML page. For more complex transactions, the Java programs are written using Enterprise JavaBeans (EJBs).

Application servers distinguish themselves by providing services beyond simple web-serving and EJB function. For instance, an application server could provide messaging or event functions not found in the EJB architecture. Application servers may also offer performance load balancing and high-availability services. Most application servers also provide implementations for directory services (JNDI), database services (JDBC), transaction services (JTS), security services like client authentication and SSL encryption, and session tracking which manages session state between the client and the server. WebLogic and WebSphere are two application servers that run on the AS/400.

WebLogic (BEA)

The BEA WebLogic Server (formerly Tengah) is written in pure Java and was ported to the AS/400 in a joint development effort by WebLogic and the IBM Laboratory in Rochester, MN. The WebLogic Server implements Enterprise Java APIs including Enterprise JavaBeans (EJBs), servlets, JDBC, and JNDI. The WebLogic Server does not include development tools, but WebLogic Server can be used with any integrated development environment (IDE) that supports EJBs. BEA has announced an alliance with Symantec, the vendor of Visual Café. As a result, you will probably find that the interface between the IDE and the web server is smoother with the combination of Visual Café and WebLogic Server than if you

chose other IDEs. This may be an important criterion for AS/400 developers who are used to tightly integrated tools and who may find the level of integration more critical than specific leading-edge features of either tool.

WebSphere (IBM)

The WebSphere product line is a cross-platform family of products that includes PC-based development tools, performance management tools, and web servers. There are no AS/400-specific performance management tools. The most important features of WebSphere for AS/400 are runtime support for Java servlets and Java server pages. These two technologies provide the ability to create dynamic HTML pages with minimal client code and without the complexity of CGI processing. WebSphere can work with any IDE or with the PC-based WebSphere Studio development environment. There are three versions of WebSphere: Standard, Advanced, and Enterprise. The Standard version of WebSphere is a no-charge feature of OS/400. The Advanced version of WebSphere for the AS/400 should be available before the end of 1999 (whether it will be a chargeable product is unclear).

Messaging Software Concepts

Messaging software provides a standard interface for communications between disparate applications—in affect, a standard way to pass parameters. Every application that implements a certain messaging standard can communicate regardless of the platform it's implemented on or the language it's written in. The structure of the message is predefined and only the content varies depending on the specific task to be accomplished. The standard structure allows the message to be passed through the system until it reaches the destination application, which actually performs the requested task. None of the intermediaries needs to be able to interpret (or even look at) the message contents.

The Road Ahead

The programming style used for the interprogram communications facilitated by messaging software is quite different from the models that are most familiar to AS/400 developers. The closest analogy in AS/400 development to messaging software is the use of data queues for interprogram communications. All messaging software communications are asynchronous: one program sends a message and another program receives it, but the sending program does not wait for an acknowledgment (although an acknowledgment may be required to finally complete the transaction).

MQSeries (IBM)

The MQSeries family of products provides services that enable application programs to communicate with each other using messages that are managed via message queues. It provides assured, once-only delivery of messages. The program sending a message can continue processing without having to wait for a reply from the receiver. If the receiver is temporarily unavailable, the message can be forwarded at a later time. MQSeries also provides mechanisms for acknowledging the messages received.

The programs that comprise a MQSeries application can be running on different computers, on different operating systems, and at different locations. To enable portability, the applications are written using an interface that is common across the various platforms. A message has two parts: the message descriptor and the message data. The message descriptor is in a standard format and, in turn, defines the format of the message data, which is unique to an application.

MQSeries provides tools and utilities to help create the necessary MQSeries objects for applications. Although MQSeries is available for many different platforms, you'll need to verify that an appropriate level of support is available for each platform used in your business if you plan to use MQSeries as a universal integration layer for applications.

Deployment Options

Theoretically, you can develop a Java application on any platform and redistribute the component parts in any configuration to multiple platforms if they all support the appropriate level of Java Virtual Machine (JVM). In reality, how a Java application is deployed has a profound affect on performance and maintainability, and many of these decisions are inherent in the design of the application. Some of the issues you'll need to consider include security, performance, development methodologies, and integration with server applications and data.

Security questions include how to identify and validate users, what level of security is required (e.g., SSL) for a transaction, and whether the application can access AS/400 data. Performance is impacted by how the application is distributed and built. For example, servlets typically provide better performance than the same function implemented using applets because the need to transfer the applet code to the client each time the application is initiated has been eliminated. In the long run, because both applets and servlets offer relatively limited functions, you'll want to consider building your applications using EJBs. Integration issues include the ability to maintain transaction state, concerns about threaded applications (e.g., Java) interacting with non-threaded applications (e.g., RPG or COBOL), and database maintenance.

IBM's AS/400 Toolbox for Java addresses some of these issues by providing Java functions (classes) that access AS/400 functions and data. The Toolbox classes are written in 100 percent pure Java and utilize Java standards so that you can access them from practically any Java development environment, and they will run on any JVM. However, when you use the Toolbox, you lose platform independence because the target platform must be an AS/400.

These examples should give you an understanding of the types of questions you'll have to address if you want to deploy Java applications on the Internet.

You'll also have to choose a set of development tools and make sure your staff has appropriate training.

For more insight into these issues see the references in "For More Information" and the two previous D.H. Andrews Group reports in this series: *Java and AS/400 – Perfect Together* and *Making Java Smoother and Easier: IBM's VisualAge for Java*. ♦

For More Information

IBM Resources

IBM's AS/400 Tools Network

<http://www.as400.ibm.com/developer/tools/index.html>

IBM AS/400 Application Development Languages and Tools

<http://www.software.ibm.com/ad/platforms/as400.html>

Databases of AS/400 Product Information Maintained by Industry Publications

Midrange Computing

<http://www.midrangecomputing.com/yellowpages/>

Midrange Systems

<http://www.midrangesystems.com/Buyguide/Search/Srchguid.htm>

NEWS/400

<http://www.news400.com/sourcebook/sourcebook.htm>

Glossary

adaptability. The concept of designing applications to be flexible enough to incorporate different technologies in future modifications. A successful application modernization project yields applications and skills that can adapt to changing requirements and technology as they evolve.

application modernization. The ongoing process of adapting applications to new technology and new business processes. It encompasses changes to user interfaces, application structure, programming methods, and development tools.

application partitioning. An important component of many application modernization projects. It addresses the issue of segregating each application into logical components.

application server. A middleware layer that simplifies the deployment of Java applications on a server. Application servers manage communication between client applications and Java programs and administer server resources such as database connections and threads. Other publications and industry segments may have different definitions for this term.

business intelligence. IBM terminology for the entire spectrum of applications and technologies related to data warehousing and data mining. In the industry at large, data warehousing is more commonly used as the umbrella term.

client/server. Applications that reside on multiple systems, typically a PC client that provides a GUI front end communicating with a server (e.g., an AS/400) where the database is stored. Depending on the design of the application, the business logic may be executed on the client, the server, or it may be distributed across both platforms. Client/server also refers to technologies and methodologies used to implement client/server applications.

collaborative computing. Also known as groupware. These applications automate tasks that involve multiple people working together to produce a deliverable such as a document or an application.

Domino. The server component of Lotus Notes. Domino is a software tool that can be used to develop collaborative computing and workflow applications. It includes e-mail and calendar functions, but it can also be used to create complex applications that combine relational database information with other types of information such as word-processing documents, image files, and spreadsheets. It is ideal for developing web-based applications such as online catalogs.

e-business. Any type of business transacted electronically.

e-commerce. A subset of e-business that involves a financial transaction.

Glossary

Enterprise JavaBean (EJB). A Java component that provides a transactional interface for business logic on a server. It must adhere to a specific standard published by Sun Microsystems.

graphical user interface (GUI). The visual component of an application. A GUI can be contrasted with a text-based interface. It uses icons and a point-and-click device such as a mouse for navigation.

groupware. See collaborative computing.

Java. A programming language based on object-oriented concepts.

middleware. Software components used to connect components of an application across different platforms, databases, and communications protocols.

modularization. An application structure that builds each function independently from all other functions and clearly defines the interfaces between functions.

object oriented. A programming and design methodology that focuses on how components interact rather than the internal structure of each component. Object-oriented languages implement specific behaviors such as abstraction, inheritance, and polymorphism.

screen scraper. A development tool that creates a new look for an existing screen design without altering the underlying logic. In AS/400 environments a screen scraper reads from and writes to the 5250 data stream.

Structured Query Language (SQL). A standard for requesting data from and making changes to databases.

workflow. Any task that requires input from multiple sources in a structured sequence. Typical workflow applications include purchasing approval and insurance claim processing. Workflow applications are often implemented using Domino. ♦

D.H.

ANDREWS

GROUP

**700 West Johnson Avenue
Cheshire, CT 06410
(203) 271-1300
(203) 272-8744 Fax
<http://www.dhagroup.com>**