CHAPTER 5

EGL Scope

Before we explore additional details about EGL, we describe its scope.

Uses of EGL

EGL lets developers create business software without requiring them to have a detailed knowledge of runtime technologies or to be familiar with object-oriented programming. Developers can focus on business issues, and your company can retain those people and their business savvy as technology changes.

The Rational products that support EGL are based on *Eclipse*, which is the integrated development environment (IDE) described at *http://www.eclipse.org*. Developers familiar with Eclipse will be familiar with the most basic product features, including the *Workbench*, which is the interactive component.

The Workbench includes several *wizards*, each of which is a sequence of dialog boxes that elicit developer input. The input is then used to automate an aspect of the development process. Some wizards are similar to those in Eclipse and set up folders to contain the developer's source code. Other wizards go beyond the ones available in Eclipse, creating skeletal source code or Web pages.

The EGL developer writes and changes EGL source code in a text editor that provides interactive assistance. Also, the developer can use a source-code debugger to verify the code's runtime behavior and to test the effect of different data values. The debugging session switches seamlessly from technology to technology as the runtime situation changes. You might debug a program, for example, along with a service that is invoked by the program and is intended for use on a remote machine.

After the developer codes and debugs the EGL source, the next step is specific to EGL: with a keystroke, the developer submits the source as input to a process called *generation*. The primary output is Java or COBOL source code, which is the basis of an executable, or JavaScript. The JavaScript is subsequently included in a Web page, which is written in Hypertext Markup Language (HTML).

The use of EGL is illustrated in Figure 5.1.

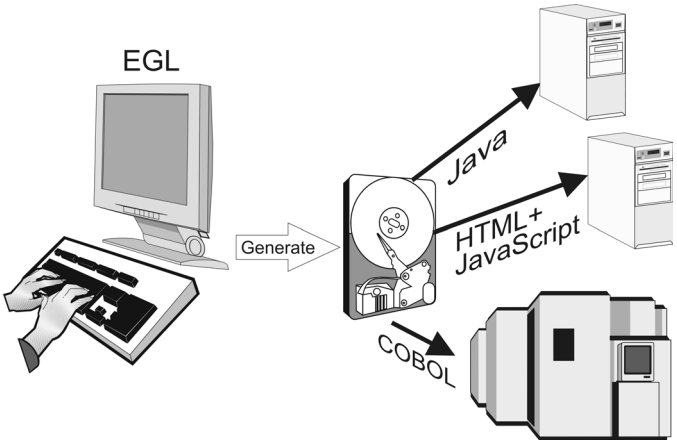

Figure 5.1: Generation and Deployment

The abstractions of EGL allow for simple coding and update. The output of the generation process is only an intermediate form. If you wish to make changes to the logic, you change the EGL source code and regenerate the output.

The products that support EGL can place Java or COBOL output on a deployment platform separate from the machine where development occurs. In the case of COBOL output, the creation of an executable occurs on the

deployment platform. EGL also produces supplementary files to help the deployment staff install the executable.

Another aspect of the overall development process is that each product that supports EGL provides two ways to generate output. A developer working in the Workbench interacts with the capabilities described earlier, including editors, debuggers, and wizards. In contrast, a developer or administrator working with the EGL software development kit (SDK) uses a command-line interface that includes none of the interactive features characteristic of the Workbench.

An organization might use the SDK in an automated build process. Developers prepare an application for testing, for example, and then store the EGL source files in a software-library system. A batch job periodically runs: first, to extract the files from the library system; second, to trigger generation; and third, to prepare and deploy the generated output. In response to errors found by the testers, the developers update and store the source code for another round of testing, which can occur only after the batch job runs again.

Each product runs on any of several Microsoft Windows platforms (Windows 2000®, Windows 2003®, Windows XP®, and Windows Vista®) or on Linux. Each product includes WebSphere® Unit Test Environment, which is a component for serving Web pages to browsers and for running applications under Java Enterprise Edition (JEE). You can use the WebSphere environment to test Web and JEE applications as you develop them. Moreover, you can test Rich UI applications directly in the Rich UI editor.

Supported Technologies

We describe the commercial technologies supported by EGL. The breadth of support has a long-term implication: the language and skills taught in this book are likely to remain useful even as your organization changes technologies.

Runtime Environment

EGL-generated code runs under any of the following environments.

- *Java Platform, Standard Edition (JSE)*. JSE is the simplest Java runtime. EGL-generated Java JSE code runs on the platforms AIX®,

HP UX, IBM i, Linux®, UNIX System Services (on System z™), and the supported Windows platforms.

HTTP servers such as the Apache HTTP server are JSE environments.

- *Java Platform, Enterprise Edition (JEE)*. JEE is a Java runtime that provides special handling for database access, Web applications, and other technologies. Developers can generate code that will run in one of the following three ways: first, as an *application client* (a Java program, but one that does not compete for all the resources needed to present data to a browser); second, in a *Web application* (logic that interacts with a browser); or third, as an *Enterprise JavaBean stateful session bean* (a modular unit of business logic). EGL-generated Java JEE code runs on the platforms AIX, HP UX, IBM i, Linux, Solaris®, UNIX System Services, and the supported Windows platforms.

  You can run JEE Web applications in Apache Tomcat, which you can download from *http://tomcat.apache.org*, and you can run any kind of Web or JEE application on WebSphere Application Server. EGL helps you to work with JEE security on either server.

- *IBM i*. On the midrange IBM Power Systems, in the context of the operating system IBM i, EGL-generated COBOL code includes interactive programs, batch programs, and services. Moreover, EGL lets you quickly create Web services that expose the logic available in any of the following kinds of programs: rpgle, cbl, cblle, sqlrpgle, sqlcbl, and sqlcblle.

- *z/OS*®. On the mainframe computer System z, in the context of the operating system z/OS, EGL-generated COBOL code runs in any of the following environments:

  - *z/OS batch* is the z/OS batch-processing environment.

  - *Customer Information Control System (CICS)* is a *transaction manager*, which is a runtime for handling large numbers of business transactions such as customer orders. Developers can generate interactive programs, batch programs, and SOAP services, all in COBOL.

  - *Information Management System (IMS™)* is another transaction manager. Developers can generate COBOL

programs that use any of the major IMS facilities on System z. Generated interactive programs can be IMS Message Processing programs (MPPs). Generated batch programs can be IMS Batch Message Processing programs (BMPs), DL/I Batch programs, or MPPs. EGL also supports the IMS FastPath facility.

- *z/VSE*™. Also on System z, in the context of the operating system z/VSE, EGL-generated COBOL code runs in either of the following environments:
  - *z/VSE batch* is the z/VSE batch-processing environment.
  - CICS is available; in this case, developers can generate interactive or batch programs.

EGL offers a special benefit when you are writing interactive code for CICS or IMS. In this case, you structure your code as if the user were having a conversation with a program that is always in memory, even though the runtime code (in the usual case) is repeatedly brought into memory and taken out of memory during the program's interaction with the user. The complexity of the conversation is handled in the logic generated by EGL and not in your EGL source code, which is relatively simple to write and understand.

Persistent Data Storage

The EGL developer uses intuitive I/O statements (such as **add** and **get**) to access data from a relational database, a hierarchical database, a message queue, or a file:

- *Relational databases*. The standard language for accessing relational databases is Structured Query Language (SQL). For simple applications, the developer can rely on the SQL statements used by default in EGL I/O statements. For complex applications, an EGL developer familiar with SQL can go beyond the defaults. Moreover, EGL is structured so that an SQL developer can write sophisticated database-access code for other developers to use.

  EGL supports access of DB2® Universal Database (DB2 UDB) from COBOL code and supports access of the following databases by way

of Java Database Connectivity (JDBC): DB2 UDB, Informix®, Microsoft SQL Server®, Oracle®, Cloudscape®, and Derby.

- *Hierarchical databases*. The standard language for accessing hierarchical databases is Data Language/I (DL/I). The developer can rely on default EGL I/O statements, can go beyond the defaults, and can write database-access code for other developers.

  EGL supports access of hierarchical databases on IMS, CICS, and z/OS batch.

- *WebSphere MQ message queues*. WebSphere MQ calls allow program-to-program communication that involves a set of queues managed by WebSphere MQ rather than by either program. The application that sends data is not dependent on the immediate availability of the application that receives the data, yet message delivery is assured.

  When accessing a message queue, the EGL developer usually relies on default EGL I/O statements. Specialized expertise is not as necessary as in the case of database access.

  EGL supports access of WebSphere MQ message queues on all platforms.

- *Files*. EGL supports access of *sequential files,* whose constituent records are accessed in record order. Access of those files is available for any target platform. EGL also supports access of two other types of files, for target platforms that allow the choice. Those other types are *indexed files*, whose records are each accessed by the value of a key in the record, and *relative files*, whose records are each accessed by an integer that represents the record's position in the file.

  For some platforms, you can associate a sequential, indexed, or relative file with any of several file technologies. You write your EGL code, then choose a file technology at generation time. The generated source code includes the I/O statements that are specific to the technology chosen.

Here are the technologies:

- *Virtual Storage Access Method (VSAM)*. EGL supports VSAM files, each of which is organized as a sequential, indexed, or relative file. EGL-generated code that runs on any of several platforms can access either local VSAM files or (in the case of IBM i) an equivalent type of file.

  The platforms for local access are AIX; IBM i; CICS; z/OS batch; and IMS (but only for EGL-generated BMPs on IMS). In addition, EGL-generated code that runs on a supported Windows platform can access VSAM files that reside on a remote System z.

- *CICS-specific technologies*. EGL supports access of the following kinds of data stores on CICS: spool files, which primarily hold program output for subsequent printing; temporary storage queues, which hold data for subsequent processing in the same or another program; and transient data queues, which submit data to another program.

- *IMS-specific technologies*. EGL supports using I/O statements to access *IMS message queues*, whether to submit data to another program or, in some cases, to retrieve data into a program.

  EGL also supports *Generalized Sequential Access Method (GSAM)* files, which are sequential files accessed by way of DL/I calls. Those calls allow processing to resume, after a failure, from the middle of a file rather than from the start.

  GSAM files are available to BMPs and on z/OS batch.

EGL supports access of two file types that are specific to IBM i: *physical files*, each of which contains data, and *logical files*, each of which provides a subset of the data in a physical file.

User Interface

With EGL, developers can create applications that interact with users in one of several ways, depending on the target system where the code runs. EGL

supports a Web-based interface; a traditional character-based interface (for CICS COBOL, IMS COBOL, IBM i COBOL, and Java applications); and a more interactive, largely character-based interface (for Java applications migrated from Informix 4GL).

Web-Based Interface

EGL supports Web-based interactions in three ways: first, by providing Rich UI, which is a new technology for writing Rich Internet Applications that will be deployed on JEE-compliant application servers or on JSE Web servers; second, by providing support for JavaServer Faces (JSF), which is a technology that runs on JEE; and third, by offering a migration path for the VisualAge Generator Web transaction, which also runs on JEE but is older and less flexible than the alternatives.

Each of the three mechanisms allows for elementary processing. The user can receive a Web page, type input into a form, and click a button to provide data for subsequent processing by application logic. Also, each mechanism allows a division of labor. A graphics designer who has minimal knowledge of software can create a Web page by dragging controls from a palette, dropping them on a drawing surface, and customizing them in a variety of ways.

Rich UI. In Rich UI, the application logic is EGL-generated JavaScript that runs in a browser. The developer writes the code in EGL.

For advanced purposes, the developer can write custom JavaScript or use JavaScript libraries instead of relying on the default behavior provided by EGL. For example, the developer can use Rich UI to access the following software:

- The Dojo Toolkit (http://dojotoolkit.org/)
- Microsoft® Silverlight (http://silverlight.net/)

We describe Rich UI in Chapter 6.

JavaServer Faces (JSF). Many Web applications are not based primarily on client-side processing; instead, they are server-centric. Logic on a server guides the construction of a stream of HTML and transmits that stream to the browser. The user periodically submits data back to the server, which processes the input as appropriate and responds with another HTML stream.

An important technology for developing server-centric Web applications is *JavaServer Faces (JSF)*. We describe JSF in Chapter 14.

Web transactions. As mentioned earlier, EGL also provides a third, less flexible way of serving business data to Web browsers. The developer in this case writes a program called a *Web transaction*, which is a pre-set flow of logic that transmits Web pages and receives data back. A JEE-compliant application server is required. The primary purpose of Web transactions is to migrate code from IBM VisualAge Generator.

Text UI

EGL-generated programs can process business logic and periodically display a *text form*—a set of character-based fields that are presented at a standalone terminal or in a workstation window. After displaying the form, the program waits for user input. A particular keystroke (ENTER or a specified function key) causes the program to receive the user's input and continue processing.

This interface technology is called *Text UI.* It provides support for interactive COBOL applications running on CICS, IBM i, or IMS. Text UI is also available for interactive Java applications; specifically, for JSE applications and JEE application clients. Use of this technology in Java is primarily to migrate code from IBM VisualAge Generator.

Console UI

EGL offers a user-interface mechanism called *Console UI*, primarily for code migrated from Informix 4GL. In this case, the users interact with buttons, drop-down lists, and the like, in a workstation window. Console UI is available for JSE applications and JEE application clients.

Support for Service-Oriented Architecture

EGL strongly supports the business use of service-oriented architecture (SOA) and includes a construct that helps the developer to create a service, which may be deployed in one of three ways:

- As a *Web (SOAP) service*, which exchanges data in a text-based format called SOAP. We refer to this kind of service as a SOAP service. The Workbench also provides ways to create and use a Web

Services Description Language (WSDL) file, which tells how to access a SOAP service.

An EGL-generated SOAP service can be deployed on any of three platforms: WebSphere Application Server, Apache Tomcat, or CICS. This kind of service can be accessed by many kinds of logic, including EGL-generated Java code, Rich UI applications, and EGL-generated COBOL code running on CICS or IBM i.

- As a *Web (REST) service*, which provides a simple way of exchanging data in a variety of text-based formats. We refer to this kind of service as an EGL REST service.

  An EGL REST service can be deployed on any JEE-compliant application server and, at this writing, is accessible only to a Rich UI application.

- As an *EGL service*, which uses a proprietary format for data exchange. The main benefits of using an EGL service are first, it gives faster response than is possible with either kind of Web service, and second, it reduces your company's need to maintain WSDL and related files.

  An EGL service can be deployed on and made available to almost any EGL target platform. One exception is IMS. A second exception relates to an EGL Rich UI application. The Rich UI application can access SOAP or REST services from any source, but cannot access the EGL services that exchange data in a proprietary format.

Last, you can expose the logic in an IBM i called program or service program as if that logic were provided from a SOAP service or EGL REST service.

Network Communication

EGL lets your company avoid some of the effort needed to integrate logic that runs on different platforms. Specifically, your organization has less need to write *interface code*, which is software whose purpose at run time is illustrated in Figure 5.2.

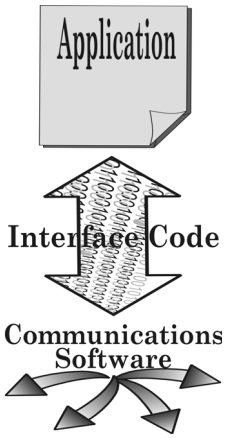

Figure 5.2: Interface Code

Interface code transfers application data to and from communications software, which in turn transmits the data from one platform to another. Your organization avoids the burden of writing interface code when EGL-generated Java logic calls a remote program deployed on IBM i, CICS, or IMS. Supported communications software for IBM i is IBM Toolbox for Java; for CICS, CICS Transaction Gateway; and for IMS, IMS Connector for Java.

Report Production

The EGL developer code can create output reports using either of several tools: Business Integration and Reporting Tools (BIRT), EGL text reporting, and print forms.

Business Intelligence and Reporting Tools (BIRT))

Business Intelligence and Reporting Tools (BIRT) is a reporting technology that delivers formatted business data to printers and screens. The technology produces sophisticated output in PDF or HTML format, including graphics, tables, charts, and graphs. The developer can sort and otherwise manipulate data from databases, variables, or Web services. For additional background details, see *http://www.eclipse.org/birt*.

An EGL program invokes the BIRT report engine to create the report. The engine can then invoke EGL functions used as *event handlers*, which are logical units that respond to a particular kind of runtime event. For example, the engine might invoke one event handler at the start of the report, one event handler at the start of a predefined report group (such as the sales data for a single type of product), and another event handler at the end of the report. For another example, an event handler might change the color of report text in

response to a value received into the report from the EGL program or from a database or file.

BIRT reports are available for logic that runs under JSE or JEE.

EGL Text Reporting

EGL text reporting creates reports that deliver the output of sophisticated business processes when you need neither graphical content nor an HTML- or PDF-formatted deliverable. The benefit is speed at both development and run time.

Several details described for BIRT reporting also hold true for text reporting. A text report can include values submitted by the EGL program that drives the report-creation process; EGL functions can act as event handlers; and EGL text reporting is available for logic that runs under JSE or JEE.

Print forms

An EGL *print form* is a set of character-based fields that a program writes periodically to a printer, either directly or by way of a file. Print forms are available for COBOL programs, JSE applications, and JEE application clients.

Integration with Existing Code

Your company can integrate EGL-generated code with existing software. EGL-generated COBOL code can interact with *native* (non-generated) programs on the same platform, whether the platform is CICS, IBM i, IMS, or z/OS batch. Similarly, EGL-generated Java code can call local, native programs written in C, C++, or Java; and can call remote CICS, IBM i, or IMS programs.

EGL provides two additional ways to integrate EGL-generated and native Java code. First, EGL lets you access a native Java interface or class from within your EGL code. You're able to use EGL syntax to work with the Java-based logic.

Second, you can cause a native Java class to call an EGL-generated program. This kind of integration involves a *Java wrapper*, a set of Java classes that will

be deployed with the native class. The Java wrapper acts as an intermediary between the externally created code and the EGL-generated program.

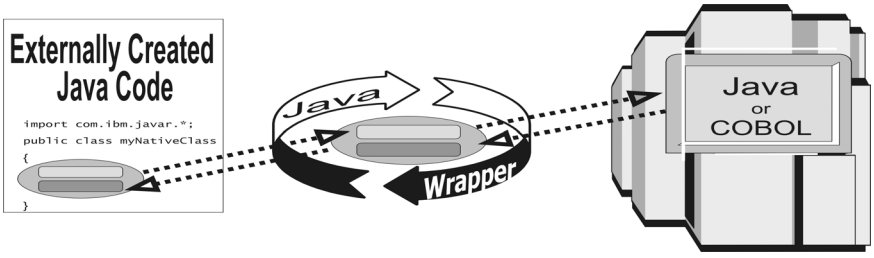

Figure 5.3: Use of a Java Wrapper at Run Time

As suggested in Figure 5.3, the EGL technology hides the details of data conversion. The native code invokes the Java wrapper, submitting data for transfer to the program. The wrapper then calls the program, which may be on a remote platform. The wrapper accepts the data returned from the program and relays the data back to the native code.

The Java wrapper is specific to the EGL-generated program being called, and the wrapper and program can be generated at the same time (Figure 5.4).

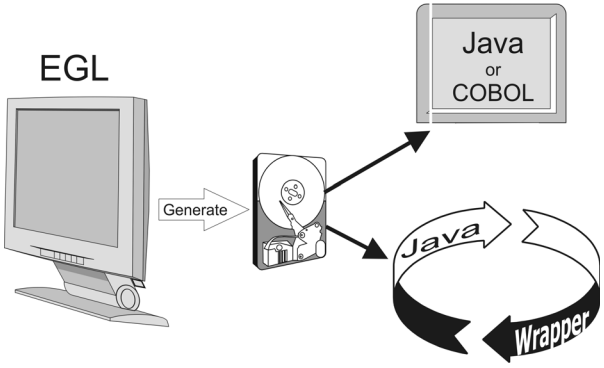

Figure 5.4: Generation of a Program and a Related Java Wrapper

Integrating Multiple Products that Support EGL

You can install an IBM Rational product that supports EGL and also install compatible products on the same machine. You have the option of working in a single, integrated development environment for all products.