Modern software development for business-oriented developers
White paper
March 2008

Rational. software

IBM

Accelerate delivery of business solutions with IBM Rational Business Developer software.

Michelle A. Cordes, System z ecosystem, Rational software, IBM Software Group

Stefano Sergi, product line manager, Rational software, IBM Software Group

Contents

Unlocking the full potential of your development staff

New integrated information solutions that enable innovative interactions with partners, customers and employees are at the top of CIOs' agendas in 2008.[1] Accomplishing this goal requires optimal use of existing IT know-how. IT managers need to tap into the full range of skills available to speed improvements to current enterprise applications and accelerate the move to service-oriented architecture (SOA).

In many organizations, the developers who have the strongest business knowledge—and a stake in the successful delivery of new services and applications—are typically skilled in traditional systems but lack knowledge of and experience with emerging technologies. Their understanding of business requirements and their experience in how to implement new capabilities based on mainframe programs make these developers extremely valuable. But it's not always practical or cost-effective to retrain them in Java™; Java Platform, Enterprise Edition (Java EE); and related Web technologies (nor, by extension, is it always feasible to retrain them in the technology of environments like SOA). So many key developers are often relegated to maintaining legacy systems. In addition, IT managers are also looking for ways to make it easier for those developers with new skills to quickly learn and reuse existing application assets to create services and deploy new workloads on existing infrastructures.

Essentially, the challenge is overcoming what is often referred to as "skill silos," where teams with very different development skills, tools, methodologies and processes have to find ways to integrate one another's work, often resulting in less-than-optimal (and certainly not rapid) outcomes. And skill silos further reduce responsiveness and flexibility because they do not allow dynamic resource allocation for cross-platform projects.

Highlights

Developed by IBM, the innovative Enterprise Generation Language equips developers of almost any background with a simplified, high-level development approach for quickly delivering cross-platform, transactional data–centric services and applications.

IBM has evolved the capabilities of the IBM Rational® Software Delivery Platform to address this need. The platform now provides an integrated set of tools, methodologies and best practices to enable your current knowledgeable staff to use the latest technologies with minimal costs and effort. It also allows new generations of developers to create services that can be deployed to traditional mainframe platforms without requiring that those developers learn the technical nuances of the environment. That means they're able to unify the development team and provide the organization with faster response to new opportunities.

At the heart of the Rational Software Delivery Platform is a set of SOA design and construction capabilities delivered on top of the Eclipse workbench. Recognizing that programming in the Java language requires a long and continuous learning process for programmers who may not have experience with object orientation, and realizing that learning the minutiae of complex mainframe applications can be daunting, IBM has developed an innovative, modern and simplified development approach: the Enterprise Generation Language (EGL).

EGL enables developers of almost any background to quickly create applications and services for deployment to the Java platform as well as to traditional mainframe transactional run times—without requiring that those developers learn the technical intricacies of the targeted platforms.

This white paper provides background into the EGL conceptual foundation, which is based on abstraction—the defining characteristic of business-oriented development. It then defines EGL at a high level; describes the benefits of the language; and introduces IBM Rational Business Developer software, the IBM product that delivers EGL. It also discusses the details that pertain to building applications and services. And in conclusion, the paper walks you through an EGL application scenario that offers insight into the architecture behind EGL-based implementations. After reading this paper, you should have a good understanding of what EGL is, who would use it and what value it can bring to your company and IT organization.

Highlights

Abstraction—the defining characteristic of business-oriented development—is based on four principles: language neutrality, platform neutrality, automated code generation and debugging.

The conceptual foundation for business-oriented development

IBM centers the vision for its application development tools on the themes of developer productivity and robust platform support. Almost always, the vision has been focused on providing an environment that enables developers to efficiently apply their business knowledge to creating applications that can operate across various execution platforms. As far back as 1981, when IBM introduced its first rapid application development environment, its core mission has been:

To provide an integrated tools environment for the rapid development of scalable, robust, mission-critical applications using traditional enterprise application programming skills to create solutions capable of running under a variety of environments and topologies.

Abstraction: the defining characteristic

Over the years, IBM products have evolved to improve IBM's support of this mission. Each improvement and advancement has allowed developers to work on concerns further away from implementation details and closer to business problems under consideration. This is the principle of abstraction at work. Abstraction is the defining characteristic of business-oriented development, a term IBM uses to describe the process of building business software without low-level technical coding.

Working at increasingly higher levels of abstraction helps achieve higher levels of developer productivity, but abstraction also allows developers to write code that can run on different target run-time platforms. Therefore, to continue delivering on its mission, IBM follows four guiding principles: language neutrality, platform neutrality, automated code generation and debugging.

Highlights

Through abstraction, developers are able to design application logic that is language neutral and platform neutral; automated code generation enables deployment to the implementation language best suited to the target platform.

Language neutrality

When it comes to creating business applications, the choice of development language is typically tied to the deployment run time as well as to the knowledge and skills of the development team. But what if there were a common language designed to generate applications for various other, more conventional, languages? Such neutrality could offer a means for expressing application logic, which developers could later transform into the implementation language best suited for the selected target platform (such as COBOL or Java).

Platform neutrality

As with language neutrality, platform neutrality allows developers to support the run-time platform best suited to the application. To effectively provide platform neutrality, users must be able to support virtually any platform in the marketplace—from the largest mainframe to the smallest workstation or desktop PC. Abstraction provides a mechanism for developers to design and implement their applications with a language that is not tied to a specific technology. In doing so, they can generate the deployed applications from this neutral development environment. And as technology changes, tooling vendors, such as IBM, can provide drivers that transform the neutral application to the new target technology.

Code generation

Code generation is the bridge between a business-oriented application that's written in a neutral language and a concrete implementation that's written in a conventional language. Code generation technology addresses how the concrete application gets deployed to a particular target run-time platform. Tooling vendors can provide generation drivers that automatically and transparently perform

Highlights

By separating business logic from infrastructural code, EGL helps developers focus on solving business issues rather than on underlying software complexities.

these transformations. Such generators deliver a high percentage of code that is associated with the application's structural "plumbing." Developers are able to focus on business rules, which typically comprise a smaller percentage of the entire application code set. By separating business logic from infrastructural code, developers can later cast the entire application into a new implementation technology by simply using a new set of code-generation drivers. The result is a new realm of development productivity.

Debugging

For an abstract, business-oriented language to work effectively, the developer who is writing code at the abstract level must also be able to debug at that level. The tools environment should have a testing facility that includes a source-level debugger that permits stepping through the abstract program code using real data before the application is deployed into the target environment.

What is EGL?

At the most basic level, EGL is a procedural programming language that developers can use to implement applications quickly. It fulfills the three criteria for an abstract programming language:

- *It must be familiar to business-oriented developers.*
- *It must automatically manage lower-level programming details.*
- *It must be transparently deployed to a set of potentially available execution platforms.*

The current EGL is the result of the evolution of IBM research and development in the area of rapid application development technology over the past 25 years. EGL combines some of the most powerful tenets of legacy fourth-generation language (4GL) technologies. It augments these 4GL capabilities with modern modular programming constructs; integrates them with new technologies, such as JavaServer Faces (JSF) and Web services; and extends their reach with new code-generation drivers for the latest run-time platforms. EGL continues to provide developers with a nearly unparalleled abstraction layer that can enhance productivity by providing a conduit to multiple run-time platforms.

Highlights

The word "generation" in the EGL proper name implies two things:

- *Business logic written in EGL will be transformed into lower-level code.*
- *Run-time artifacts will be created to help natively execute the generated application on a desired target platform.*

Because EGL programs can be written, tested and debugged at the EGL-source level, developers can defer actual code generation until they know the programs work.

EGL programs are written, tested and debugged at the EGL-source level, not on the generated-code level. This means that developers can defer actual code generation until they have satisfactorily tested the EGL application or service functionality. This aspect differentiates EGL from traditional code generators. The EGL developer never changes the generated code—virtually all changes are made at the EGL level.

EGL and MDA

Readers familiar with the Object Management Group (OMG) and its Model Driven Architecture (MDA) initiative will notice parallels to EGL. MDA is a form of model-driven development based on the Unified Modeling Language (UML) and other OMG standards. MDA calls for modeling the software lifecycle at distinct levels of abstraction, coupled with transformations that map and manage the relationships among those models. In addition to defining the concept of a platform-independent model (PIM) to which EGL can match nicely (albeit as a textual "model"), MDA defines a platform-specific model (PSM), which corresponds to EGL-generated code (such as Java/Java EE or COBOL). Essentially, MDA model transformations are analogous to the EGL code generation.

IBM Rational Business Developer software supports model-driven development by providing an automated bridge between UML models and the downstream implementation with EGL specifications.

These comparisons suggest that EGL can offer traditionally skilled developers an opportunity to practice what the MDA initiative is all about: separation of concerns, modularized reuse across the lifecycle, managed complexity and the ultimate in productivity. The Rational Business Developer tool takes this notion even further by providing an automated bridge between UML models and the downstream implementation with EGL specifications. The benefits are compelling.

Highlights

There's less code to write and less need for extensive developer training in order to take advantage of the latest technologies and development styles and technologies.

Less code to write

Generating a large portion of the application code—particularly the infrstructural plumbing required as part of any target architecture, such as Java EE platform—shields developers from having to learn about or write special code for most of the application. The developer can instead focus on writing only the business rules.

Reduced training requirements

The time and cost of retraining developers can be a significant barrier, preventing many legacy developers from moving into object-oriented programming and other new technologies. Code generation helps reduce the cost and time needed to become proficient in designing and implementing applications.

Faster adoption of iterative and agile development

The ability to specify application behavior in less technical constructs, and to immediately animate and verify these specifications, encourages and promotes iterative and agile development. Real-world results show that the adoption of an iterative prototyping evolution approach and agile development style leads to the faster and more accurate delivery of business services and applications.

Easier transition to new technology

As technology evolves, the training costs and disruptions caused by applying new technologies to applications can be very high. The neutral language application, combined with a code-generation driver for the new technology, can help make this transition much easier. In this way, developers keep the application definition constant while leveraging improvements in implementation technology.

Improved application quality and performance

Code generation offers the benefit of leveraging pretested and proven application infrastructure frameworks, which constitute a great portion of the generated code. The custom code that developers need to write for a given application is typically limited to business rules and behavior. This can reduce bugs and improve quality and performance.

Highlights

EGL provides an easy-to-learn programming model with a syntax that is familiar to business-oriented developers, which can help reduce training costs and boost productivity.

The business value of EGL and Rational Business Developer software

EGL development is enabled through Rational Business Developer, which delivers an Eclipse-based comprehensive and highly productive integrated development environment (IDE) that can support key technologies and tools in the IBM Rational Software Delivery Platform. EGL provides a simplified approach to application development through a familiar programming model, transparent code generation, run-time platform robustness, and EGL-based debugging.

Familiar programming model

EGL offers an easy-to-learn programming paradigm embodied in a traditional procedural programming syntax that is familiar to business-oriented developers. The developer's view is abstracted to a level independent of the underlying implementation technology. It shields developers from the complexities of various supported run-time environments. This abstraction can result in significantly reduced training costs, a great improvement in programming productivity and a nearly seamless transition of traditionally skilled developers to modern computing technologies.

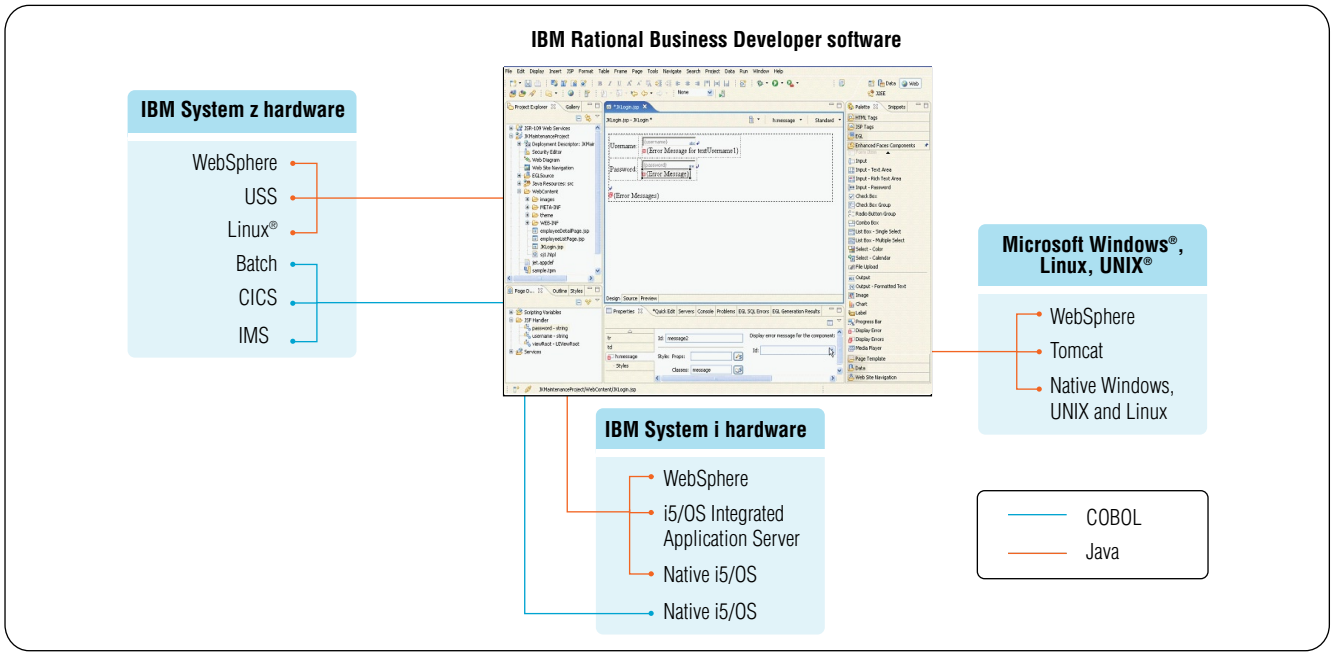

Figure 1: IBM Rational Business Developer software is designed to automatically generate COBOL or Java source code, depending on the deployment platform.

Highlights

Developers can produce higher-quality code more quickly and simply reuse the same EGL source code whenever there's a change in the target run-time platform.

Transparent code generation

Developers write their business logic in EGL source code while the tools included in Rational Business Developer do the rest. These tools transform business logic into Java or COBOL language, and optimize the infrastructure code for the target run-time platform. Because there's less code to write, developers can produce applications more quickly and with fewer bugs. As an example, when generating Java code to run in a Java EE application server that invokes a mainframe service, Rational Business Developer is designed to automatically generate the Java classes necessary to invoke the associated IBM CICS®/IBM IMS™, COBOL or RPG program elements.

Run-time platform robustness

Whenever a change to the target run-time platform occurs, only a new code-generation driver for the new platform is needed. The application source code remains constant, enabling developers to easily leverage improvements in implementation technology. For example, if a new Web services technology becomes available, developers can reuse the same EGL source code—they simply regenerate the application using the new driver.

EGL-based debugging

Source-level debugging is provided within Rational Business Developer at the EGL level; therefore, developers don't need to generate code and deploy the final executable to the production platform before debugging it. This capability provides developers with comprehensive isolation from the complexity of the run times and middleware, which can mean huge productivity gains and a significant reduction in time to market. Developers debug at the logical EGL level even if the application or service makes calls to other, non-EGL components. For example, if a developer has IBM Rational Business Developer and IBM Rational Developer for System z software, and an EGL service calls a COBOL-stored procedure in the IBM DB2® database to execute on the IBM z/OS® platform, the debugger can step through both the EGL and COBOL-stored procedure call.

Highlights

Most business-oriented developers can simplify their jobs with EGL.

Who benefits from using EGL and Rational Business Developer software?

Anyone who needs to focus more on solving business problems and less on underlying implementation technologies can benefit from using EGL and Rational Business Developer software. The most common types of business-oriented developers who can simplify their jobs with EGL include those working with IBM Informix® 4GL, RPG or COBOL IBM System i™, COBOL or PL/I IBM System z™, IBM VisualAge® Generator and IBM VisualGen® technologies, as well as legacy 4GL, Microsoft® Visual Basic and databases.

IBM Informix 4GL developers

Rational Business Developer comes with a special utility that largely automates the conversion of existing Informix 4GL-based applications to EGL. Developers can modernize and extend these applications using a state-of-the-art development environment, and they can deliver Web and service-oriented solutions.

RPG or COBOL IBM System i developers

EGL offers a simple way to extend existing RPG or COBOL applications to the Web and allows developers to create new services that can be deployed in System i environments, including the IBM i5/OS® integrated application server. Thanks to the familiar procedural nature of the language, there is minimal retraining. System i developers can also use EGL to deliver traditional text user interface (5250) applications, to access data queues and data areas, or to invoke integrated language environment (ILE) procedures from those applications.

COBOL or PL/I IBM System z developers

Similar to System i developers, System z developers are typically devoted to legacy system maintenance. With EGL and Rational Business Developer, System z developers can now become key contributors by using valuable business-domain expertise to deliver innovative modern solutions—without major retraining. These developers can also use EGL to deliver traditional batch and text user interface (3270) applications for the CICS and IBM IMS/TM and IBM IMS/DB environments.

Highlights

The simplicity and ease of learning EGL make it a particularly attractive replacement for obsolete mainframe and distributed tools.

IBM VisualAge Generator and IBM VisualGen developers

For VisualAge Generator and VisualGen developers, EGL represents the next generation and a logical migration path. The environment provides easy-to-use and highly automated migration capabilities that bring valued VisualAge Generator and VisualGen applications into a modern development environment—an environment in which the applications can leverage a modern set of run-time technologies.

Legacy 4GL developers (including Computer Associates Cool:Gen, Computer Associates Telon, Natural and Oracle Forms)

The broad platform coverage, the simplicity and the ease of learning EGL make it particularly attractive as a replacement for obsolete or orphaned 4GL mainframe or distributed tools. EGL offers legacy 4GL developers an enterprise modernization alternative through a community of IBM Business Partner tools and services that largely automate the transformation to EGL and to the Rational Software Delivery Platform.

Microsoft Visual Basic and Sybase PowerBuilder developers

EGL offers powerful development efficiencies for Visual Basic or PowerBuilder developers, particularly in the areas of enterprise scalability and multiplatform run-time support.

Database developers

EGL virtually eliminates the need to learn the database manipulation language and to code the create, read, update and delete (CRUD) functionality by simply doing it for you. Rational Business Developer also provides database developers with an open source reporting system that can help turn raw data into HTML or Adobe® PDF reports.

Highlights

IBM offers a broad selection of development tools that organizations can adopt as needed to help accelerate the delivery of high-quality applications and services that support business objectives.

Rational Business Developer software and the IBM Rational Software Delivery Platform

IBM offers a portfolio of software development solutions designed for those organizations looking to contain development costs and accelerate delivery of high-quality applications and services that support business objectives. For those companies that deploy to the IBM WebSphere® Application Server platform, IBM Rational Business Developer software can be used alone. If businesses want to automate and improve the testing of new EGL solutions, they can use IBM Rational Functional Tester software together with Rational Business Developer software. The addition of IBM Rational ClearCase® and IBM Rational ClearQuest® software can help organizations provide a well-managed EGL development environment for software changes and releases.

The IBM Rational Software Architect product offers a comprehensive design and construction tool that leverages model-driven development with UML. It enables developers to create well-architected applications and services, and it includes the capabilities of IBM Rational Application Developer software. Further, it offers UML modeling for users who want a model-driven approach to their EGL development. With Rational Business Developer, the architect or designer can produce EGL services directly from UML models, and then leverage the richness of the EGL development facilities to complete the application development for the platforms supported by EGL.

If organizations have System i or System z developers who work in EGL, as well as other programming languages, they can benefit from a unified workbench. IBM Rational Business Developer can be combined with IBM Rational Developer for System z software to provide this capability. And Rational Business Developer software is included in IBM Rational Developer for System i SOA Construction software. In these kinds of development environments, EGL is typically used for new development, especially Web application development, and to create business services for SOA solutions. It can also be used for extending and modernizing existing COBOL or other legacy programs.

Highlights

IBM Rational COBOL Runtime for System z is a product that provides the runtime libraries for programs that are developed with Rational Business Developer and generated into COBOL for deployment to supported System z environments.

Application development with EGL

It's important to understand the EGL elements that are essential for developing applications.

Abstracted language

EGL is a full-featured, procedural language that abstracts out the details of a target technology. It has verbs like `get`, which simplify the programming model by providing a consistent specification to various target data sources. For example, a `get` statement can refer to records in a relational database, to an indexed file or to messages in a message queue. Developers are not required to learn and code technology-dependent database managers or message-oriented middleware programming.

As technology changes, applications written in EGL can simply be regenerated into the new language or for a new target platform—helping to protect development investments.

Writing applications in EGL can also help protect development investments. Organizations can cast or generate the abstracted language into other languages. Currently, EGL generates Java or COBOL code. As technology changes and evolves, organizations can protect their investments by regenerating the application into a new language or for a new target platform or to entirely new platforms—without having to modify the application.

EGL libraries

An EGL library is simply a file that includes EGL code. EGL libraries allow application developers to easily decouple the business logic from other application code. And these libraries provide various entry points—one per function. Developers can call functions from other functions in other libraries, or from EGL code in EGL programs or EGL page handlers.

Highlights

The use of EGL libraries is optional, but it is the best way to reuse components. Developers can compare EGL libraries to COBOL copy books or Java packages. Many EGL libraries are provided directly in the products with built-in functions. This is similar to Java classes provided by Java toolkits or frameworks. EGL libraries have the potential to greatly simplify and accelerate application development.

EGL programs and functions

Developers can also use EGL programs to code the business logic, but with a single entry point. Similar to COBOL programs, EGL programs can be main programs, or they can be called in the same way they're called in COBOL. EGL code within programs can invoke EGL functions. Developers can then compare an EGL function to a paragraph in the COBOL procedure division or to a Java method. EGL programs are units of EGL source that can be generated into COBOL or Java code.

Specifically designed to support the Web and service-oriented architectures, EGL enables developers to define a construct called a service.

EGL services

EGL is specifically designed to support service-oriented architecture. Developers can define a construct called a *service*, which is a set of operations that can be invoked by a client (any application component that can reside on the same or on another platform) or application. Services can be deployed as either EGL services or as Web services. The former can be accessed from EGL code directly or by way of a TCP/IP connection, and the client in this case can be a program, handler, library or another service. The latter can be accessed over an HTTP connection from clients written in nearly any language.

EGL page handler

When coding EGL applications to be deployed for the Web, the preferred method uses JavaServer Faces (JSF) technology. This framework uses JavaServer Pages (JSP) screens. In an EGL-based application, virtually every page is associated with

Highlights

an EGL page handler. The EGL page handler controls a user's run-time interaction with a Web page. Specifically, the page handler provides data and services to the page-displaying JSP screen. The page handler itself includes variables and specialized logic such as:

- *An* `OnPageLoad function`*, which is invoked the first time the JSP screen renders the Web page.*
- *A set of event handlers, each of which is invoked in response to a user action (specifically, clicking a button or opening a link).*

The page handler implements the controller component of the model-view controller (MVC) pattern (see the MVC description later in this paper). Therefore, it is recommended that developers not include business logic directly in the page handler. Although the handler might include lightweight data validations, such as range checks, developers should invoke other programs or functions to perform complex business logic in order to follow MVC guidelines.

EGL rich Web support

IBM has extended EGL to support the development of rich Web applications that conform to the Representational State Transfer (REST) architecture. EGL rich user interface handlers are designed to allow the definition of rich Web interfaces and the generation of JavaScript to conform to the Asynchronous JavaScript and XML (Ajax) framework. Consistent with the principles of abstraction, this is accomplished without requiring that the EGL developer learn or understand the intricacies of JavaScript, XML or Ajax. This innovative development approach is available as an IBM alphaWorks® project at www.alphaworks.ibm.com.[2]

Relational database connectivity with EGL

EGL takes the complexity out of accessing data for developers who primarily want to provide relevant information to business users in the context of an application.

Accessing data from databases can sometimes be challenging for developers who primarily want to provide users with the information to make the best business decisions. To access data, a developer needs to connect to a database; know and use the database schema; be proficient in structured query language (SQL)

Highlights

or other data manipulation language in order to get the appropriate data; develop the primitive functions to perform the basic CRUD database tasks; and provide a test environment to efficiently test the application.

- ***Connectivity.*** *With EGL, wizards take developers through a step-by-step process to define connectivity. This helps users locate the databases in remote locations, such as System z hardware.*
- ***Database schema.*** *When dealing with existing databases, Rational Business Developer provides a near-seamless import facility that makes the schema structures available to the EGL application.*
- ***Database coding.*** *Rational Business Developer generates SQL or other data manipulation statements based on EGL code. Developers can then use the generated code or alter it to suit their needs.*
- ***Primitive functions.*** *The Rational Business Developer product comes with generation facilities that are designed to automatically generate the typical CRUD functions for database-driven applications either as EGL libraries or as EGL services.*
- ***Test capabilities.*** *Rational Business Developer includes a test environment that helps eliminate the complexities associated with deploying and running applications in complex target platforms.*

EGL enables developers to access hierarchical database systems using I/O keywords, from which Rational Business Developer generates DL/I statements in the output COBOL code.

Hierarchical database access with EGL

EGL can provide powerful notations to access business data stored in the IMS/DB hierarchical database systems. Developers can interact with these databases through EGL input/output (I/O) keywords such as `add`, `get` and `replace`, which generate DL/I statements in the output COBOL code. The default DL/I statements that Rational Business Developer generates from EGL keywords can optionally be customized. EGL can also provide the ability to code explicit DL/I statements in one of two ways: by customizing an EGL I/O keyword statement through the use of a `#dli` syntax, or by using a set of library functions supporting DL/I calls.

Highlights

File access with EGL

Developers can also use EGL programs and libraries to access data storage other than databases, such as serial files, indexed and relative record files (virtual storage access method [VSAM]) and other system files. As a result, developers have significant flexibility in the types of data sources that they can use within an EGL application system.

Application architecture with EGL

To complete Web-based applications, organizations need to bring together the elements discussed above in the context of an application architecture. When this integration is complete, they'll have a concrete, deployable application. The architecture that EGL generates is a Java EE architecture conforming to the JSF technology–based MVC pattern.

JSF and EGL

A set of Java classes and JSP tag libraries, JSF provides a framework for developing Web applications. Rational Business Developer lets developers drag and drop JSF controls onto a page canvas instead of having to implement pages using hand-coding techniques.

IBM Rational Business Developer integrates EGL and JSF technology for an event-driven model that can greatly simplify the construction of Web applications.

Rational Business Developer provides integration of EGL and JSF technology, producing an event-driven model in which a page-specific handler manages each request. The page handler can act on information submitted with the request, or it can forward the information to another handler for processing. This event-driven model greatly simplifies the building of Web applications. Control logic in the page handler is written in EGL. Business logic in the libraries and programs is also written in EGL. There is no need to master Java skills in order to write the application user interface or business logic. Rational Business Developer is designed to generate the necessary Java code. The JSF with EGL capability makes for a potentially high-productivity page development environment.

Highlights

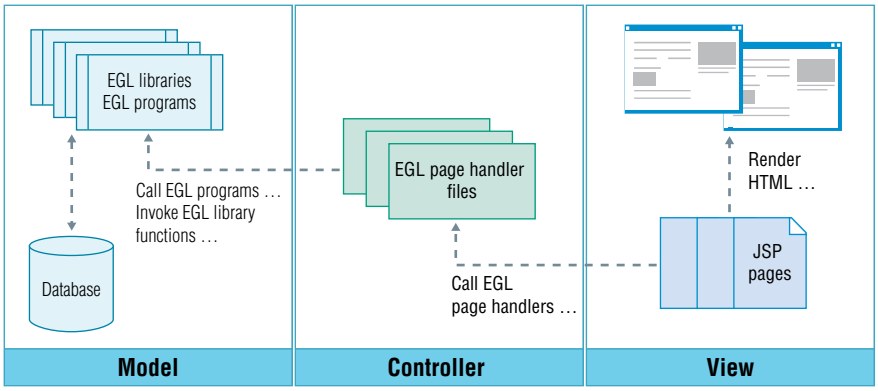

Figure 2: Relationships between elements of MVC-compliant EGL application

The MVC, or model 2, framework, is widely considered a best practice for building Web applications.

Using the MVC framework to develop Web applications

The MVC framework (also referred to as "model 2") has many benefits and is often considered a best practice for developing Web applications.

At run time, the application server contains both the view and the controller components of an MVC Web application, while a third tier (which can be inside or outside the application server) contains the model.

As shown in figure 2 in the model component of the MVC framework, developers can find the business logic for the Web application—which in most cases involves accessing data stores such as relational databases—in the EGL libraries and programs.

In the MVC framework view component, the code responsible for the presentation layer consists of JSP and Java Beans technology that stores data for the JSPs to use. Page creation is greatly simplified by using JSF controls available within the Rational Business Developer page editor.

Highlights

The EGL development environment frees business-oriented developers from having to implement the MVC pattern—the Rational Business Developer generation engine does it for them.

In the MVC framework controller component, the EGL page handlers contain the code that determines the overall flow of events.

The beauty of the EGL development environment is that business-oriented application developers are not confronted with implementing—and do not need to understand how to implement—the MVC pattern. The Rational Business Developer generation engine does that for them.

Walking through an EGL application scenario

Figure 3 illustrates a simple sales employee inquiry application scenario using EGL. The following steps show the flow of information and the tasks associated with simple EGL application development:

1. *The user enters an employee number and clicks a button.*
2. *Clicking the button creates a request that the JSF servlet handles.*
3. *In turn, the controller servlet invokes the employee inquiry server program.*
4. *The employee number is then passed to the server program.*
5. *The server program validates the employee number by reading an IBM DB2 database using the employee number as the key to find. The server program can be a function within an EGL library or an EGL program.*
6. *If the server program finds the employee number, it collects information (name, salary and commission) and returns the data.*
7. *The server sends the returned data to the result JSP page, which displays it for the user. If the server program does not find the employee number, it creates an error message and returns the sales employee number.*
8. *The server sends the error message to the error JSP page, which displays it.*

Highlights

A simple example of a sales employee inquiry application shows how EGL and the Rational Business Developer page editor conceal the underlying complexity of Web technologies, simplifying the process for developers.

To accomplish this simple application using Rational Business Developer, developers create the three pages using the drag-and-drop page editor shown in figure 3. The controller logic that calls the appropriate server function is written as an EGL page handler, and the server function that performs the validation is written as an EGL library function. Rational Business Developer can then generate Java code and pull these pieces together into a Java EE technology–based application that can be deployed and run.

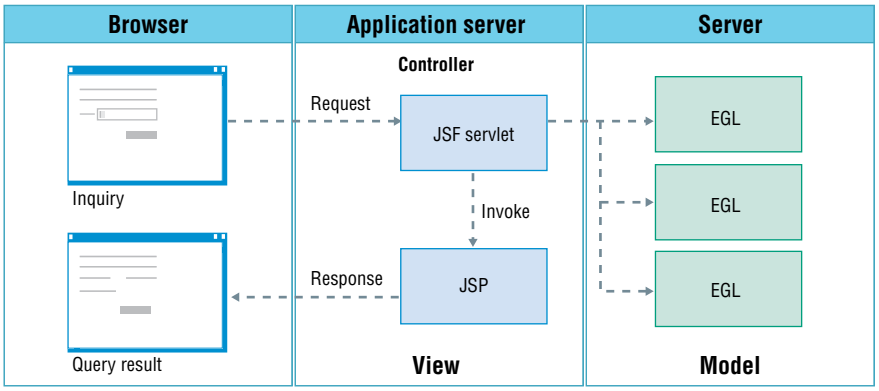

Figure 3: This graphical depiction of the Rational Business Developer page editor shows the detailed components of the sales employee inquiry application example, as well as the flow of information.

From this simple walk-through example, it is easy to see that the complexity behind the new and evolving Web technologies is hidden from the developer by an easy-to-use page editor and EGL.

Highlights

Simplifying migrations and enterprise modernization

It is not the intent of this white paper to address the various driving forces behind enterprise IT migrations, the modernization of legacy applications or the various planning steps involved. However, what is of importance is the role that EGL and Rational Business Developer can play in simplifying such initiatives.

The new generation of IT systems requires software development capabilities that support new middleware and emerging application architectures. Flexibility is the name of the game. But because of the variety and complexity of activities and artifacts necessary to implement these solutions, it is essential that development organizations establish a systematic and comprehensive approach to developing software with proven methodologies, processes and tools. At the same time, legacy application developers are asking IBM to preserve the levels of productivity and simplicity that they have become accustomed to, and which support business-critical applications.

Through Rational Business Developer, EGL can simplify and streamlines IT migrations and application upgrades, helping to make them more efficient and cost-effective.

Through Rational Business Developer, EGL becomes a core component of a broad, comprehensive application development, governance and lifecycle management solution from IBM that spans modeling and application development through testing and software configuration management tools. Because EGL offers a familiar development model for business-oriented developers, there is no need to restaff. Developers who are already employed at an organiztion can use EGL to call business-critical application components. Or, if and when appropriate, organizations may choose to convert existing applications to EGL.

Through code conversion of legacy 4GL such as VisualAge Generator; VisualGen; Informix 4GL; Natural; RPG; Computer Associates Cool:Gen, Telon, Ideal and Synon, organizations can leverage existing investments and move rapidly onto a modern software development platform. This is made possible through automated conversion utilities designed to migrate the specific legacy environment to EGL—efficiently and cost-effectively.

Highlights

Because it is easy for both legacy and new developers to learn, EGL can break down skill silos and unify development teams, helping to optimize resource utilization.

Once organizations convert their code, they have the opportunity to step back and analyze whether a redesign is required, particularly from a user interface perspective. If a redesign is necessary, EGL can assist organizations in bringing their applications to the Web.

Unifying siloed development teams

EGL helps break down skill silos among development teams, enabling organizations to make optimal use of programming resources. Legacy developers often have difficulty working with Web and SOA technology, and new developers are not able to deal with earlier-generation technology, so managing cross-technology teams can be challenging and inefficient. But virtually all developers can easily learn EGL, creating a unified pool of specialists who can work on comprehensive projects in a single technology. EGL can also help motivate uninspired developers, retain existing talent and attract new talent, because it offers a modern, powerful programming environment that targets legacy environments.

Is EGL right for your organization?

EGL is not intended as a substitute for the Java or COBOL language. While large percentages—and even 100 percent—of an application can be delivered using only EGL, there will be cases where additional code may be required or even appropriate for the requirements at hand. But EGL is ideal for business-oriented development teams that:

- *Value ease of learning.*
- *Need high levels of productivity.*
- *Must deliver modern applications and services but cannot afford the cost, time or risk of transforming each developer into an expert in mainframe or Java development.*

For more information

To learn more about IBM Rational Business Developer software, contact your IBM representative or IBM Business Partner, or visit:

ibm.com/software/awdtools/developer/business

To learn more about EGL and IBM Rational Business Developer software and how they can help your organization speed the adoption of emerging technologies, improve productivity, leverage legacy developers and increase the likelihood of your success in building modern applications, visit:

ibm.com/developerworks/rational/products/rbde

And to experiment with IBM Rational Business Developer software, visit:

ibm.com/developerworks/downloads/emsandboxes/systemz.html

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
03-08

---

1 *CIO*, "The State of the CIO 2008: The CIO's Time to Shine," December 10, 2007, http://www.cio.com/article/163700.

2 All statements regarding IBM's plans, directions and intent are subject to change or withdrawal without notice.

RAW11012-USEN-01