Modern software development for business-oriented developers
White paper
April 2007

Rational. software

IBM

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

Contents

Introduction

The explosion of computer networks and the global reach of the Internet are transforming the way businesses operate and use information, and the era of traditional application islands is rapidly coming to an end. New integrated information solutions enabling innovative interactions with partners, customers and employees are at the top of many CIOs' agendas. But to avoid the pitfalls of application silos that hinder responsiveness and flexibility, the enlightened CIO is embracing service orientation as the architectural underpinning of all new business software.

This new generation of IT systems requires software development capabilities that support new middleware and emerging application architectures. At the same time, however, development teams have grown accustomed to levels of productivity and simplicity that had accompanied the emergence of client/server computing. The IBM® Rational® organization has evolved the capabilities of the IBM Rational Software Delivery Platform to address this need with an integrated set of tools, methodologies and best practices that enable effective governance of the software development process and accelerate the transition to service-oriented architecture (SOA).

At the heart of the software delivery platform is a set of SOA design and construction capabilities delivered on top of the Eclipse workbench, primarily targeting the Java™ platform. Recognizing that programming in the Java language requires a long and continuous learning process for programmers who have no experience with object orientation, IBM has developed an innovative, modern and simplified development approach, the Enterprise Generation Language (EGL).

Highlights

EGL enables developers of any background to quickly create applications and services for deployment to the Java platform as well as traditional mainframe transactional run times, without requiring that developers learn the technical intricacies of these platforms. This enablement is critical because in many organizations the developers with the strongest business knowledge—and with a stake in the successful delivery of new systems—are typically skilled in traditional technologies but lack the Java, Java Platform, Enterprise Edition (Java EE) and SOA technical expertise required for most new development platforms. Thus, these key resources are often relegated to maintaining legacy systems.

This paper first provides some background into EGL's conceptual foundation and introduces the notion of business-oriented development. It then describes EGL at a high level, along with the motivations for employing such a language and the IBM product that delivers the EGL integrated development environment (IDE): IBM Rational Business Developer Extension software. The paper goes on to present more details of EGL and Rational Business Developer Extension as they pertain to building applications and services. Finally, it provides some insight into the architecture behind EGL-based implementations. After reading this article, you should have a good understanding of what EGL is, who would use it and what value it brings to companies and their IT organizations.

Conceptual foundation

Before delving into the details of IBM Rational Business Developer Extension and EGL, let's review the conceptual foundation upon which EGL was built. IBM centers the vision for its application development tools on the themes of developer productivity and robust platform support. The vision has always been

Highlights

to provide 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.

Over the years, IBM products have evolved to continuously improve IBM's support of this mission. Many enhancements have been added to the language, the development paradigm and the tools to improve development productivity while embracing new technologies and supporting emerging run-time platforms. Each improvement and advancement has allowed developers to work on concerns further away from implementation details and closer to the problem under consideration. This is the principle of abstraction at work, and it is the defining characteristic of business-oriented development: the endeavor of building business software without the need for low-level technical coding.

Working at increasingly higher levels of abstraction helps achieve higher levels of productivity, but abstraction is also necessary to allow developers to write code that can run on different target run-time platforms. The following list outlines the basic guiding principles IBM follows to continue to deliver on our mission:

- ***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 the know-how of the development team. But what if we had a common language that was designed to generate applications into various other, more conventional languages? Such neutrality provides the developer with a common means of expressing application logic, which developers can later transform into the implementation language best suited for the selected target platform (e.g., COBOL, Java).*

Highlights

- ***Platform neutrality.*** *As with language neutrality, platform neutrality lets you support the run-time platform best suited for the application. To effectively provide platform neutrality, you must 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, you can generate the actual deployed application from this neutral development environment. As technology changes, the tooling vendor can provide drivers that transform the neutral application to the new target technology.*
- ***Code generation.*** *Code generation is the bridge between the business-oriented application written in a neutral language and a concrete implementation written in a conventional language. The generation technology also worries about how the concrete application gets deployed to a particular target run-time platform. Tooling vendors can provide generation drivers that automatically and transparently perform these transformations. These generators provide a high percentage of code that is associated with the application's structural "plumbing." Developers focus on business rules, which typically comprise a smaller percentage of the entire application code set. By separating business logic from infrastructural code, you 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 a business-oriented language like EGL to work, the developer 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.*

Highlights

EGL and Model Driven Architecture

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.

MDA defines the notion of a platform-independent model (PIM) to which EGL matches nicely (albeit as a textual "model"). MDA also defines a platform-specific model (PSM), which corresponds to EGL-generated code (e.g., Java/Java EE or COBOL). The notion of MDA's model transformations is analogous to EGL's code generation.

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. Rational Business Developer Extension tools take this notion even further by providing an automated bridge between UML models and the downstream implementation with EGL specifications as described later in this paper.

These guiding principles result in real, tangible benefits that can help increase the likelihood of a project's success. The benefits include:

- ***Less code to write.*** *Generating a large portion of the application code—particularly the infrastructural plumbing required as part of any target architecture, such as Java EE—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.*** *Due to the time and cost required to train legacy developers, training proves to be a barrier for legacy developers moving into object-oriented programming and other new technologies. Code generation helps to reduce the cost and time needed to become proficient in designing and implementing applications.*

Highlights

- ***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 the adoption of an iterative prototyping evolution approach and agile development style that lead to faster and more accurate delivery of business services and applications.*
- ***Proxy to new technology.*** *As technology evolves (a change that we know is a constant), the training cost, as well as the disruption, 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 to make this transition much easier. In this way, you are keeping the application definition constant while leveraging improvements in implementation technology.*
- ***Improved 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, with resulting higher quality and performance. 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.*

What is EGL?

An abstract programming language, as described in the previous section, must meet the following goals:

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

Highlights

Today's 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; extends them 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 an unparalleled abstraction layer that enhances productivity by providing a conduit to multiple run-time platforms.

At the most basic level, EGL is a procedural programming language that enterprise-level or business-oriented developers can use to implement applications quickly. The word "generation" in the 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.*

EGL programs are written, tested and debugged at the EGL source level, not on the generated code level. This means that you can defer actual code generation until you have satisfactorily tested the EGL application/service functionality. This aspect differentiates EGL from many other types of code generators. The EGL developer never changes the generated code—all changes are made at the EGL level.

EGL development is enabled through the Rational Business Developer Extension, which delivers an Eclipse-based comprehensive and highly productive development environment as an extension of any IBM Rational Software Delivery Platform programming workbench, such as the IBM Rational Application Developer for WebSphere® Software product, further enhancing the intrinsic productivity of the EGL programming abstraction model with world-class modern IDE capabilities and integration with other key technologies and tools such as JSF visual construction and Web service visual editing.

Highlights

The business value of EGL and Rational Business Developer Extension software

EGL provides a simplified approach to application development based on these principles:

- ***Familiar programming model.*** *EGL provides 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 results in significantly reduced training costs, great improvement in programming productivity and nearly seamless transition of traditionally skilled developers to modern computing technologies.*
- ***Transparent code generation.*** *Developers write their business logic in EGL source code while the tools included in Rational Business Developer Extension do the rest. These tools transform business logic into Java or COBOL language, and optimize the infrastructure code for the target run-time platform. This results in less user-written code, which means faster turnaround time and fewer bugs in the deployed application. As an example, when generating Java code to run in a Java EE application server that invokes a mainframe service, Rational Business Developer Extension automatically generates the Java classes necessary to invoke the associated IBM CICS®/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. This allows the application source code to remain constant while improvements in implementation technology are leveraged. For example, if a new Web services technology becomes available, you can reuse the same EGL source code—you only need to regenerate the application using the new driver.*

Highlights

- ***End-to-end EGL-based debugging.*** *Source-level debugging is provided within Rational Business Developer Extension at the EGL level; therefore, you don't need to generate code and deploy the final executable to the production platform before debugging it! This provides developers complete, end-to-end isolation from the complexity of the run times and middleware with huge productivity gains and reduction of 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 new EGL service needs to call a COBOL IBM DB2® database-stored procedure that will execute on the IBM z/OS® platform, the EGL debugger works even when stepping into such components.*

Many companies are under pressure to quickly roll out new systems based on existing mainframe programs and emerging Java EE and Web services standards; this is due to the obvious benefits of these technologies. While many available developers lack the technical skills needed in these areas, they are extremely valuable because of their expertise in the business domain, their understanding of business requirements and their general experience in how to implement such systems. However, simply retraining this workforce in Java, Java EE and related Web technologies is not practical or cost-effective. The result of this situation is what is often referred to as the "skill silos" effect, 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. These skill silos are also an impediment to responsiveness and flexibility because they do not allow dynamic resource allocation across cross-platform projects.

Highlights

Through EGL, Rational Business Developer Extension can address many of these challenges. It allows you to leverage your current business-domain knowledgeable staff to use the latest technologies with minimal costs and effort. It allows new generations of developers to create services that can be deployed to traditional mainframe platforms without having to learn the technical nuances of the environment, enabling the creation of a unified development team by breaking through the skill silos. The result may allow your company to be more flexible and responsive to new business opportunities.

Who benefits from using EGL and Rational Business Developer Extension software?
Simply put, anyone who needs to focus more on solving business problems and less on underlying implementation technologies. To be more specific, the list below features the most common types of business-oriented developers who can benefit from using EGL:

- ***IBM Informix® 4GL developers.*** *Rational Business Developer Extension comes with a special utility that largely automates the conversion of existing Informix 4GL-based applications to EGL, enabling you to maintain and extend these applications using a state-of-the-art development environment, and begin delivering 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 you to create new services that can be deployed in your System i servers with minimal retraining, thanks to the familiar procedural nature of the language.*
- ***COBOL or PL/I IBM System z™ developers.*** *Similar to System i developers, System z developers who are typically devoted to legacy system maintenance can now become key contributors, thanks to their valuable business-domain expertise, to delivering innovative modern solutions with minimal retraining.*

Highlights

- ***IBM VisualAge® and IBM VisualGen® developers.*** *EGL represents the next generation and logical migration path for these developers. The environment provides easy-to-use and highly automated migration capabilities that bring your valued VisualAge 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 (e.g., Computer Associates Cool: Gen, Computer Associates Telon, Natural, 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 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 developers.*** *EGL offers similar but more powerful development efficiencies than the Visual Basic platform, particularly in the areas of enterprise scalability and multiplatform run-time support.*
- ***Database developers.*** *EGL simplifies having to learn the database manipulation language and having to code the create, read, update and delete (CRUD) functionality by simply doing it for you.*

Rational Business Developer Extension software and the IBM Rational Software Delivery Platform

Rational Business Developer Extension integrates with and extends all the IBM products that deliver the design and constructions facilities of the IBM Rational Software Delivery Platform. Once installed on top of any of the following products, Rational Business Developer Extension delivers to the users of these programming environments the full range of EGL core capabilities.

Highlights

The notes below help position which hosting product is the most appropriate for your use of the Rational Business Developer Extension.

- ***The IBM Rational Application Developer for WebSphere Software product.*** *A comprehensive IDE for rapidly designing, developing, analyzing, testing, profiling and deploying applications using Java, Java EE, Web, Web services, SOA and portal technologies, Rational Application Developer for WebSphere Software combines with Rational Business Developer Extension to allow you to freely and seamlessly switch between Java and EGL development, with full interoperability between the two languages and tight integration between the typical Java technology-based tools, such as JSF technology-based visual composition of Web pages with EGL data and logic, the ability to exploit native WebSphere tools for testing and deployment support, or the ability to develop portal solutions with EGL services and application back ends.*
- ***The IBM WebSphere Developer Studio Client for System i product.*** *An IDE for developing Java, Web, Web services and client/server applications specifically to run on the System i server, WebSphere Developer Studio Client for System i combined with Rational Business Developer Extension enriches the environment with rapid development EGL tools that either seamlessly integrate with existing System i legacy programs and data, or generate new services that deploy natively to the System i server.*
- ***The IBM WebSphere Developer for System z product.*** *This is an IDE optimized for developing COBOL or PL/I native applications running on System z platforms. You can optionally create Java EE and Web applications integrated with legacy transactional environments (CICS and IBM IMS™) and their associated languages (COBOL and PL/I). The product includes wizards that help you write Web services against existing legacy COBOL CICS programs using either Java connectors or SOAP for CICS technology. Once installed over WebSphere Developer for System z, Rational Business Developer Extension enriches the environment with rapid development EGL tools that either seamlessly integrate with existing System z legacy programs and data, or generate new services that deploy natively to the System z server.*

Highlights

- ***The IBM Rational Software Architect product.*** *A compressive design and construction tool that leverages model-driven development with UML, Rational Software Architect enables you to create well-architected applications and services. This product also includes all the capabilities of Rational Application Developer and offers UML modeling for users who want a model-driven approach to their EGL development. Once Rational Business Developer Extension is installed over Rational Software Architect, 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 any of the platforms supported by EGL.*
- ***The IBM Rational COBOL Runtime for zSeries® product****. This product provides the run-time libraries for programs that were developed with Rational Business Developer Extension and generated into COBOL for deployment to any of the supported System z environments.*

Application development with EGL

The following sections describe EGL elements that are important to developing applications.

EGL language

EGL is a full-featured, procedural language that abstracts out the details of a target technology. EGL 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, 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.

Highlights

Writing your applications in EGL can also protect your development investment. You can cast or generate the abstracted language into any other language. Currently, EGL can generate Java or COBOL code. As technology changes and evolves, you protect your investment by having the ability to regenerate into a new, improved target platform or to entirely new platforms—without the need to modify your 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 they provide various entry points—one per function. You can call these functions from other functions in other libraries, or from EGL code in EGL programs or EGL page handlers.

The use of EGL libraries is optional, but it is the best way to reuse components. You 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. These 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. EGL programs are similar to COBOL programs in that an EGL program can be a main program, or it can be called in the same way it is called in COBOL. EGL code within programs can invoke EGL functions. You can 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.

Highlights

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 in the same or 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 any other language.

EGL page handler

When coding EGL applications to be deployed for the Web, the preferred method utilizes JSF technology. This framework has one or more JavaServer Pages (JSP) screens. In an EGL-based application, every page is associated with 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 specific user action (specifically, by the user clicking a button or link).*

The page handler implements the controller component of the Model View Controller (MVC) pattern. (The MVC will be discussed later in this paper.) Therefore, it is recommended not to include any business logic directly in it. Although the handler might include lightweight data validations, such as range checks, you should invoke other programs or functions to perform complex business logic in order to follow MVC principles.

Highlights

Database connectivity with EGL

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 SQL data manipulation language in order to get the appropriate data.*
- *Develop the primitive functions to perform the basic CRUD database tasks.*
- *Provide a test environment to efficiently test your application.*

EGL provides capabilities that make this task extremely easy for the business-oriented developer:

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

Highlights

File access with EGL

You can also use EGL programs and libraries to access data storage other than relational databases, such as serial files, indexed and relative record files (Virtual Storage Access Method [VSAM]), IBM MQSeries® queues and other system files. This provides a large amount of flexibility in the types of data sources that you can use within an EGL application system.

Application architecture with EGL

To complete end-to-end Web-based applications, you need to bring together all the elements discussed above in the context of an application architecture. When you complete this integration, you'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

JSF is a set of Java classes and JSP tag libraries that provides a framework for developing Web applications. Its implementation in Rational Application Developer lets you drag and drop JSF controls onto a page canvas instead of having to implement pages using hand-coding techniques.

Rational Business Developer Extension plugs into Rational Application Developer and 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. This totally eliminates the need to master Java skills to write your application's user interface or business logic. Rational Business Developer Extension will generate all the necessary Java code. The JSF with EGL duo makes for an extremely high-productivity page development environment.

Highlights

MVC framework

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

- ***Model.*** *You can find the business logic, which in most cases involves accessing data stores, such as relational databases, in the EGL libraries and programs.*
- ***View.*** *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 Application Developer page editor.*
- ***Controller.*** *The EGL page handlers contain the code that determines the overall flow of events.*

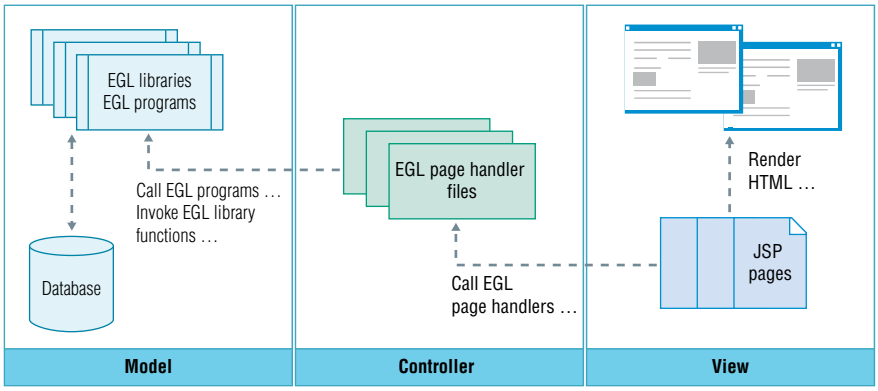

Figure 1: Relationships between elements of an MVC-compliant EGL application

Highlights

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 Extension generation engine does that for them.

Walking through an EGL application scenario

Figure 2 illustrates an application scenario using EGL. The following steps will show the flow of information and the tasks associated with simple EGL application development.

In this application:

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

Highlights

To accomplish this simple application using Rational Business Developer Extension, you will create the three pages using the drag-and-drop page editor shown in figure 2. 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 Extension will generate Java code and pull all these pieces together into a Java EE technology-based application that you can deploy and run.

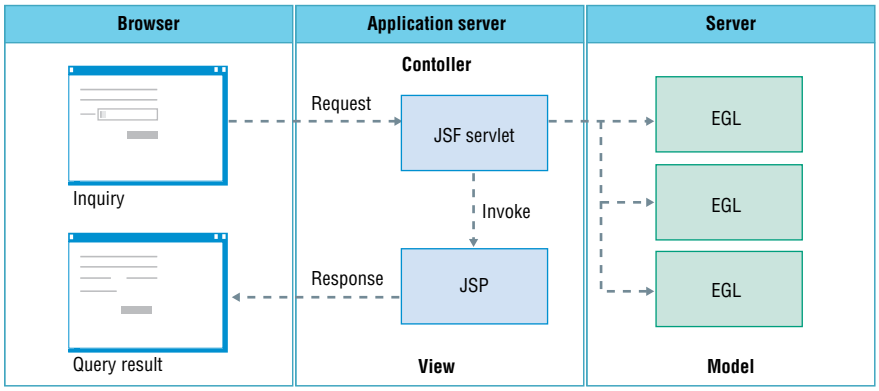

Figure 2: Typical EGL application scenario

You can see with this simple walk-through example that the complexity behind the new and evolving Web technologies is hidden from the developer by an easy-to-use page editor and an easy-to-use programming language called EGL.

Migrations and enterprise modernization

This white paper does not intend to address the various driving forces behind enterprise IT modernization of legacy applications or the various planning steps involved. However, what is of importance to note in the context of this paper is the role that EGL and Rational Business Developer Extension can play in such modernizations.

Highlights

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. However, 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 Extension, EGL becomes a core component of a broad, comprehensive application development, governance and lifecycle management solution from IBM Rational that spans modeling and application development through testing and software configuration management tools. This highly productive language provides a familiar development model that business-oriented developers are accustomed to. There is no need to restaff; instead, you can rewire and leverage your existing staff with EGL. And knowing how crucial your investment in existing legacy systems is, EGL can simply call your business-critical applications, leveraging what is best about your existing legacy environment. Or, if and when appropriate, you may choose to convert your existing applications to EGL.

Through code conversion of legacy 4GL such as VisualAge; VisualGen; Informix 4GL; Natural; RPG; Computer Associates Cool:Gen, Telon, Ideal and Synon; and more, IT organizations can leverage their 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 in a cost-effective, efficient manner.

Highlights

Once you convert your code, you have the opportunity to step back and analyze whether a redesign is required, particularly from a user interface perspective. If a redesign is required, EGL can assist you in bringing your applications to the Web or with leveraging a rich client platform (RCP).

To summarize, EGL can play a critical role in enterprise application modernization, helping to future proof IT application organizations from the ever-changing world of middleware, databases, languages, platforms and computing environments.

Conclusion

Our purpose in this paper was to introduce EGL and the Rational Business Developer Extension product: what they are, why IBM has invested in their development and how so many traditional software developers benefit from them. Java, Java EE, SOA and all the modern Web technologies are powerful, yet their complexities can make them challenging to learn. IBM is making advancements in all its product lines to make development of such modern solutions easier. IBM also understands the importance of leveraging existing legacy code, as needed, for business-critical applications. EGL is one important technology enabling us to address these critical application development issues.

We hope that this paper has piqued your interest in EGL and how it can help you speed up the adoption of emerging technologies, improve productivity, leverage legacy developers and increase the likelihood of your success in building modern applications.

For more information

To learn more about how IBM Rational Business Developer Extension software and EGL can benefit you or your organization, contact your local IBM sales team or visit:

ibm.com/developerworks/rational/products/rbde

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

Produced in the United States of America
05-07

RAWxxxxx-USEN-00