Enterprise Generation Language (EGL): an overview

Abstract

Enterprise Generation Language (EGL) is a simplified high level programming language that lets you write full-function applications quickly. It frees you to focus on the business problem rather than on complex software technologies. The details of middleware programming and Java/J2EE are hidden from you, so you can deliver enterprise data to browsers even if you have minimal experience with Web technologies.

Within the WebSphere Studio and Rational design and construction product families, EGL is part of a platform that integrates several rapid development technologies, such as JSF, within the Eclipse framework. Together, these technologies combine to produce a highly productive development environment. Developers write their business logic in EGL, the EGL is used to generate Java or COBOL, and the runtime artifacts can be deployed to whatever runtime platform you have targeted for your application.

This article first defines and describes EGL at a very high level, and the motivations for such language and development environment. It then goes on to present more details of EGL as it pertains to building applications. Finally, it provides some insight into the architecture behind an EGL based application.

After reading this article, you should have a good idea of what EGL is, who would use it and why someone would use it. If at the end of the article you are interested in knowing more about EGL, a roadmap is also included.

What is EGL

At the most basic level, EGL is a procedural language that can be use by enterprise level or business oriented developers to implement applications quickly. The word "Generation" in the name implies that the logic that is written in EGL will be transformed or generated into runtime artifacts that can be deployed to a number of different target platforms.

In a broader and more comprehensive definition, EGL is not only a language but a highly productive development environment. Integrated into several IBM/Rational products, EGL offers the productivity not only with the language abstraction and simplicity but also with the integration with other key technologies such as Java Server Faces and Eclipse.

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

- **Simplifying the specification:** EGL provides an easy to learn programming paradigm that is abstracted to a level that is independent from the underlying technology. Developers are shielded from the complexities of a variety of supported runtime environments. This results in a reduced training costs and a great improvement in productivity.

- **Code generation:** High productivity comes from the ability to generate the technology neutral specifications (EGL) or logic into optimized code for the target runtime platform. This results in less code that is written by the business oriented developer and in turn a reduced number of bugs in the application.
- **EGL Based Debugging:** Source level debugging is provided in the technology neutral specification (EGL) without having to generate the target platform code. This provides complete, end-to-end isolation from the complexity of the underlying technology platform.

Benefits of EGL

Many companies are under business pressure to quickly roll out new systems with the overall strategy to adopt the emerging J2EE and Web Services standards because of the obvious benefits of these technologies.

However, the pool of available developers, although extremely valuable because of their expertise in the business domain and their comfort in understanding the business requirements and how to implement them, cannot be simply re-trained in Java and J2EE. The costs of such training have been estimated to be in the order of well over $20,000 per developer and the time required to reach a level of proficiency in new technologies may not be compatible with the business pressures. Considering the degree of complexity in the new technologies and the relative inexperience of the developer pool, the results may not be ideal

As a development environment, EGL can address many of today's development challenges. Integrated into the IBM/Rational development tool offerings, EGL can be the enabling technology that breaks through these challenges. It can allow you to leverage your current business-domain knowledgeable staff to use the latest technologies with minimal costs and effort. The result will allow your company to be more flexible and responsive to new business opportunities.

Who should consider using EGL

Simply put – developers who will benefit most from the EGL technology are developers who need to solve business problems, not technology problems. If your developers fall into any of the following categories, they would probably be an excellent candidate for adopting EGL:

- Developers who need higher productivity.
- Developers who need to deploy to diverse platforms.
- "Business Oriented" Developers;
    - **4GL Developers (Oracle Forms, etc…):** A community of IBM Business Partners can help you transform your legacy 4GL applications to the IBM/Rational development platform with EGL.

- **Visual Basic Developers:** EGL offers similar but more powerful development efficiencies for the less technically skilled developers as does VB.

- **Database Developers:** EGL takes the pain out of having to learn the database manipulation language and code the Create, Read, Update, Delete (CRUD) functionality by simply doing it for you.

- **RPG Developers:** EGL offers a procedural language that is familiar to RPG developers. This will enable developers to move to a modern platform with minimal training costs while reaping the benefits of the latest technologies.

- **COBOL/PLI Developers:** By generating COBOL from EGL, your COBOL developers can move to a new platform that leverages the latest technologies. Moving to EGL within the IBM/Rational development tools will free developers that have been trapped in legacy platforms but who can contribute greatly to new projects with their business domain expertise.

- **Visual Age Generator Developers:** IBM provides time tested and easy to use and highly automated migration capabilities that can bring your valued legacy Visual Age based application to a modern development environment and to a modern set of runtime technologies.

- **Informix 4GL Developers:** As a new capability in the portfolio, IBM is now enabling Informix 4GL based applications to easily migrate to a modern extensible IBM/ Rational development tools.

Application Development with EGL

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

EGL Language

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

EGL has verbs like “get” that simplify the programming model by providing a consistent specification to a variety of target data sources. For example, a “get” statement can refer to records in a database or messages in a message queue. Developers are not required to learn and code technology dependent database manager or message oriented middleware programming.

Writing your applications in EGL can also protect your development investment. The abstracted language can be cast or generated into any other language. Currently, EGL can generate Java or COBOL. As technology changes and evolves, your investment is

protected by having the ability to re-generate into new target platform that have been improved or to entirely new platforms – without the need to modify your application.

EGL Libraries

EGL has a construct called a Library. An EGL Library is simply a file that includes EGL code. EGL libraries provide the application developer with the ability to easily decouple the business logic from other application code. EGL Libraries provide a variety of entry points – one per function. These functions can be called from other functions in other Libraries or from EGL code in EGL Programs or EGL Page Handlers.

EGL Programs

EGL Programs can also be used to package up business logic, but with a single entry point.

EGL Page Handler

In an EGL based application, every page will have a "shadow" 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 JSP that displays the page. The page handler itself includes variables and the following kinds of logic:

- An OnPageLoad function, which is invoked the first time that the JSP 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)
- Optionally, validation functions that are used to validate Web-page input fields
- Private functions that can be invoked only by other page-handler functions

It is considered a best practice that the page handler should have no logic. It implements the controller component of the MVC model. Although the page handler might include lightweight data validations such as range checks, it's best to invoke other programs or functions to perform complex business logic so that you follow MVC principles.

Database Connectivity with EGL

Accessing data from databases can sometimes be challenging to developers whose primary objective is to provide their users with the information that is optimal for them to make business decisions.

To be able to access data, a developer needs to

- connect to a database,
- know and use the database schema,
- be proficient in SQL in order to get the appropriate data,
- provide the primitive functions to perform the basic CRUD database tasks, and
- provide a test environment to efficiently test your application

EGL provides capabilities that make this task very easy for that business oriented developer.

Connectivity: Wizards will take these developers through a step by step process of defining connectivity.

Database Schema: If you are using an already existing database, EGL provides an easy to use import capability that will make the schema structure available to your application.

SQL Coding: EGL provides the generation of SQL statements based on your EGL code. You then have the option to use the SQL that was generated or for those power SQL users, you can alter the generated SQL to suit your needs.

Primitive functions: The EGL generation engine will automatically generate the typical CRUD – Create, Read, Update, and Deleted functions that are the workhorse functions for database driven applications.

Test capabilities: The IBM/Rational development tools have a test environment that eliminates the complexities that are associated with deploying and running your application in complex target platforms.

Application Architecture with EGL

In order to make your application complete, the elements described above need to come together in the context of an application architecture. When this integration is complete you have a concrete application that can be deployed.

The architecture that EGL generates is the J2EE architecture. Other important technology that needs to be understood due to the role they play in the application architecture includes Java Server Faces (JSF) and Model View Controller (MVC).

JSF and EGL

JSF is a set of Java classes and JSP tag libraries that provide a framework for developing Web applications. Its implementation in Rational Web Developer (RWD) and Rational Application Developer (RAD) allows developers to drag and drop JSF controls onto a page canvas instead of having to implement pages using hand coding techniques.

The integration of EGL and JSF produces and an event-driven model in which each request is handled by a page specific handler. The page handler can act on information submitted with the request, or forward it 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 means that you won't need Java™ skills to write your applications. All of the Java code will be generated from the EGL code.

The JSF with EGL duo makes for an extremely high productivity page development environment.

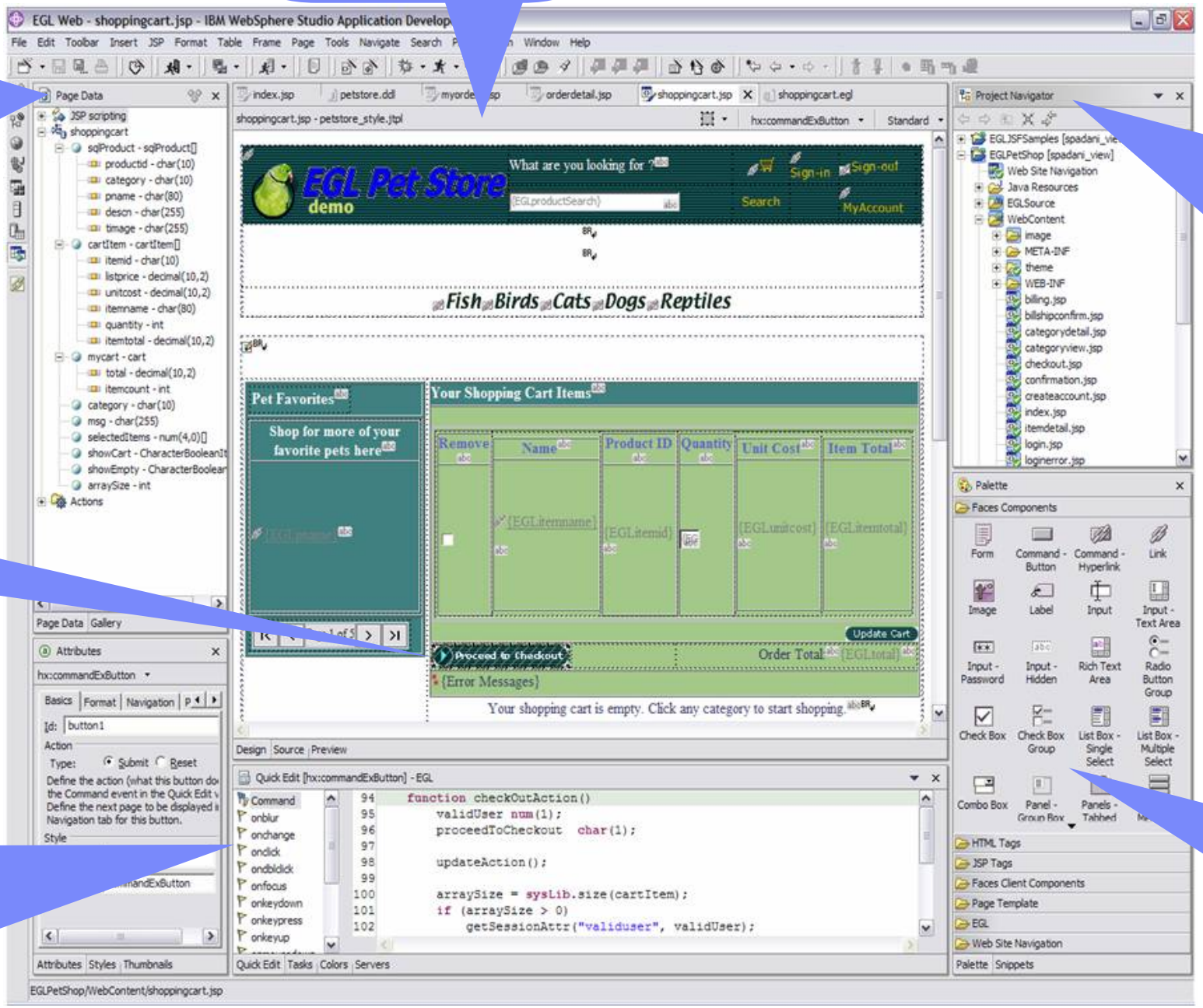
Below you find a screen shot of the RAD Page Editor.

Page Designer: *Java Server Faces based GUI Page Designer for Web*

Page Data: *Drag and Drop EGL Data Model Records and Data Items to build dynamic web pages using Page Designer*

Command Event: *Trigger Server side EGL business logic from visual controls*

Business Logic: *Interactive logic development and debugging in EGL (For developers experienced in COBOL, RPG, PL/SQL, PowerBuilder, Informix, Visual Basic and other 4GL programming languages.*



Model View Controller

The model-view-controller framework (referred to as MVC or "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 of Application Server) contains the model.

Model

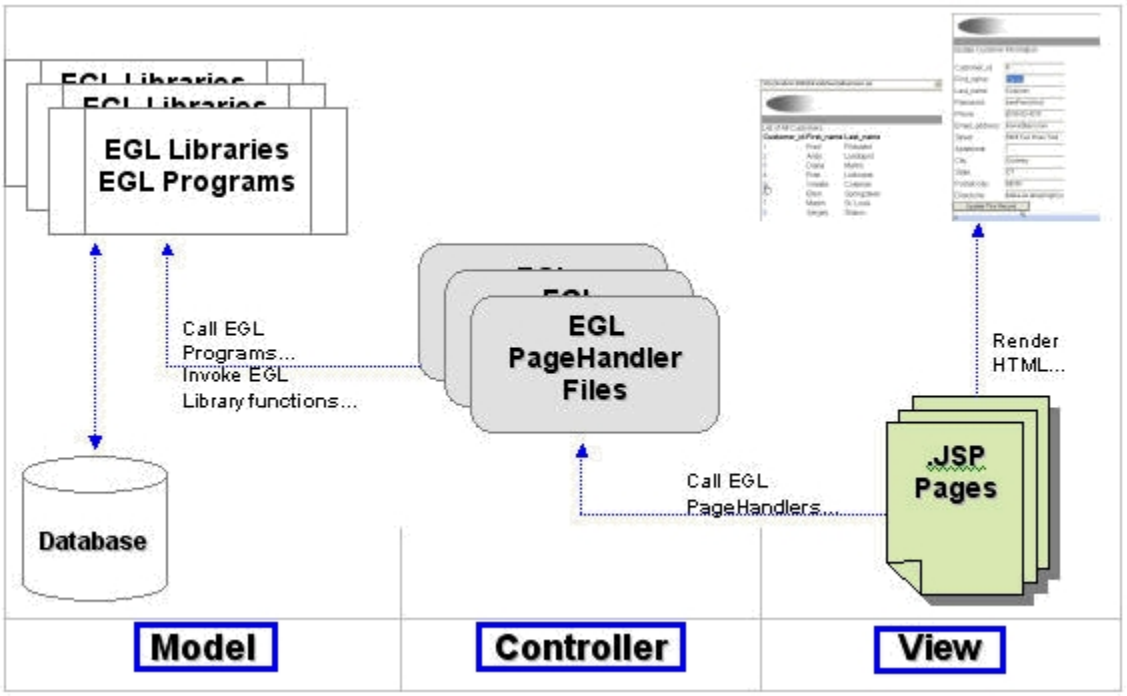
The business logic, which in most cases involves accessing data stores such as relational databases, can be found in the EGL Libraries and Programs.

View

The code responsible for the presentation layer consists of JSPs and Java beans that store data for use by the JSPs. The creation of pages is simplified greatly by the use of JSF controls that are available within the RWD or RAD Page Editor.

Controller

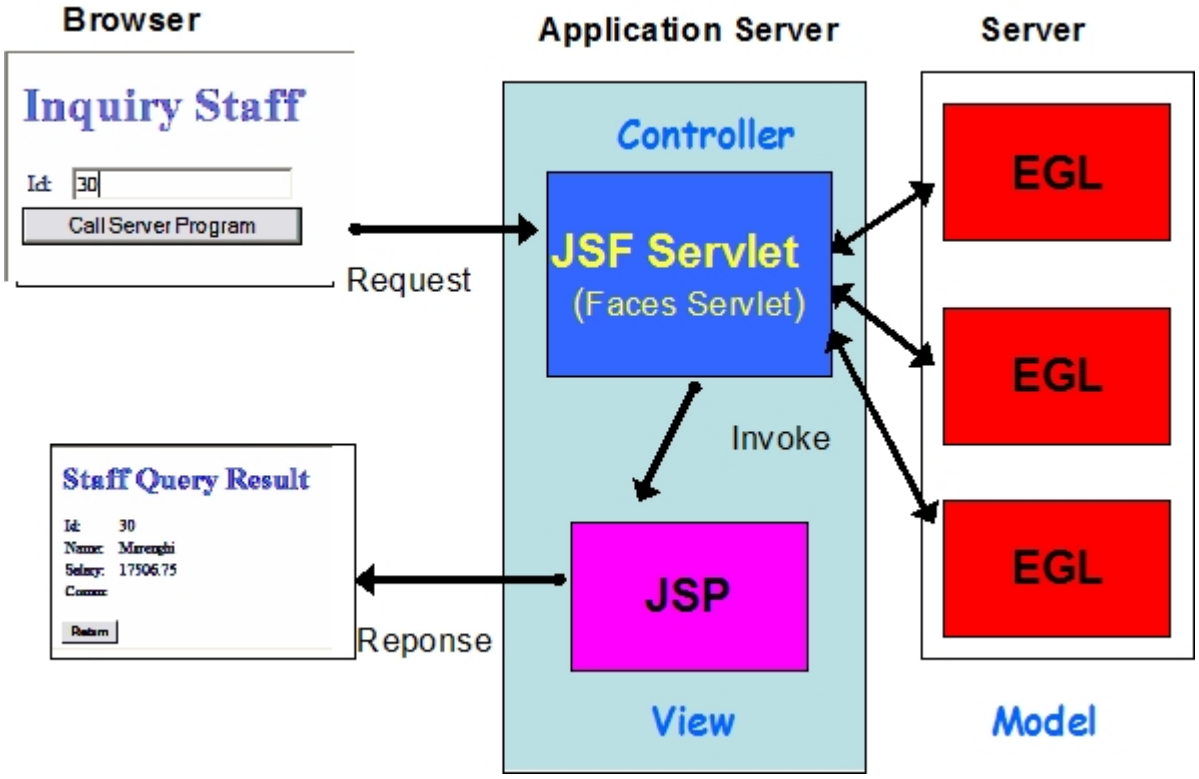
The code that determines the overall flow of events is contained within the EGL Page Handlers.



The beauty of the EGL development environment is that the business oriented application developer is not confronted and does not need to understand how to implement MVC. The generation engine does it for them.

Walking through an EGL application scenario

The following steps will walk you through the flow of information and tasks associated with a simple application scenario – all in the context of an EGL application.



To better understand how EGL works, we will walk through the workings of a very simple application. In this application:

1. The user types an ID and clicks a button.
2. Clicking the button creates a request that is handled by the JSF Servlet. The controller Servlet, in turn invokes the appropriate Server program. The server program is passed the ID.
3. The server program validates the ID by reading a 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 ID is found, the server program collects information (Name, Salary and Commission) and returns it.
5. The returned data is sent to the Result JSP page and displayed.
6. If the ID is not found, the server program creates an error message and returns it.
7. The error message is sent to the Error JSP page and displayed.

To accomplish this simple application, the developer will create the 3 pages using the drag and drop Page Editor shown in the diagram earlier in this article. 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 also written in EGL as an EGL Library function.

The EGL development environment will generate Java and pull all of these pieces together into a J2EE based application that can be deployed and run.

Conclusions

You can see with this simple walkthrough example that the complexity behind the new and evolving Web technologies is hidden from the developer by an easy to use Page Editor and by an easy to use program language – EGL.

The purpose of this overview was to get you thinking about the possibilities of using EGL to speed up the adoption of emerging web technologies, improve productivity, leverage legacy developers and increase your likelihood of success in building applications.

FAQ's

The following are questions that are often asked in context to the use of EGL. The answers provided, along with the descriptions provided above, will help you determine if EGL is for you.

Why Learn a New Language?

I have heard this question many times. In fact, this question was asked frequently by C++ and Smalltalk programmers when the new Java language was born. So:

Why another language? One answer might be: because a procedural language programmer has more affinity with EGL than Java.
Why not continue with COBOL? Maybe because COBOL does not have elements that help it integrate with the Web world.
And why not use Java only? This one is simple: because EGL productivity is higher than Java; some people say that EGL can increase developer productivity nearly 10 times compared to Java coding. In one IBM internal study, 507.5 hours was necessary to code the famous pet store sample application using Java without code generators, compared to only 55 hours using EGL. Here, we also return to the old Assembler vs. COBOL and COBOL vs. 4GL battles.

Another big advantage that I see when using 4GL is that we code at a specification level that is higher than Java or COBOL, which is the reason for greater productivity. Also, we all know the changes that happened in 3 or 4 years of J2EE; anyone who coded version

1.0 (and later 2.0) Enterprise Java Bean (EJB) components will understand this issue very well. When we code at a higher specification level, the Java code generated is the one that applies to the current Java version.

How will Java programmers react to EGL?

The argument of which language is the best is always a polemic discussion and often subjective. I have seen a wide range of reactions to different changes over the years. Not long ago, during C++ vs. Smalltalk, Java was a big help since it has syntax similar to C++ and architecture similar to Smalltalk architecture (like the Java Virtual Machine). However, since EGL is a new language, some developers are bound to have strong reactions, particularly the experts. A new discussion on the scale of COBOL vs. 4GL is born again.

Usually, 4GL languages tend to attract more enthusiasts at the management level, since productivity and ease of learning are seen as big advantages. Developers typically believe that their own code is more efficient than the code produced by code generators, which could be very true if the developers used object models and frameworks, but this is not typically the case.

In fact, developers should be more concerned with the business logic rather than the technology. If a programmer creates an application with good performance, good quality and that allows future maintenance, he has reached his objective.

Can I trust another 4GL language?

Usually one 4GL language is associated with a tool that has its own editors and code generators -- which is not a characteristic of traditional languages like COBOL and Java. I know many software companies that have created 4GL languages and associated tools that are now no more; most of the 4GL languages born at the '80's are not around any longer, but some are still alive and evolve still today, with Web support, Java code generation, and so on. IBM started its leap into 4GL with CSP in 1981, has continued to stand fully behind this programming paradigm over the years, and is committed to remain behind it for the foreseeable future. EGL is an evolution and the embodiment of that commitment, as demonstrated by the available migration paths from CSP, VisualAge Generator and VisualAge Generator to EGL.

One question here is fundamental: Which language is the 4GL generating? If the answer is Java or COBOL, then even if the tools that are used along with the 4GL die out, you still will be able to maintain the generated code.