LTI

Let's Solve

Whitepaper

Migrating Legacy EGL Platform to Multi-tier Web-based Architecture

Author : Jayamsakthi Shanmugam and Ravi Bhardwaj

A Larsen & Toubro Group Company

Contents

1. Overview 3
2. Introduction 4
3. Current Status 4
4. Proposed Solution Procedure 5
5. Current Implementation and evaluation of approaches 8
6. Conclusion 10
7. References 10
8. About the Author 11
9. About L&T Infotech 11

Overview

Enterprise Generation Language (aka EGL), an IBM marquee product, was one of the legacy modernization options pursued by many organizations, particularly in the North American and European regions, over a decade back. This was very different from many other options in the market at that time. EGL automatically generates Code in multiple languages, such as Java and COBOL. It was an attractive option at that time to many organizations, who were struggling with modernizing their legacy infrastructure; improving their customer experience, and reducing the infrastructure and resource spend, as EGL provides built-in functionality to use database objects and related functions without much changes. This was justified as being an intermediate solution, but unfortunately organizations never moved away from this.

EGL, as a technology within its original technology stack (mainly user interface design), is no longer relevant. Organizations which have implemented these intermediate solutions are now in a situation without the luxury of time in hand, and with the additional pressure of becoming irrelevant in the industry itself.

This whitepaper explores the options available to these organizations to expedite the migration of these intermediate legacy implementations like - EGL to a platform - which is more conversant with future innovations.

Currently, Java Server Faces Widget Library (JWL) within EGL provides an approach to migrate EGL codebase to POJO classes, which can be deployed on any Java container. However, this does not provide state-of-the-art features available with newer technologies like Ext JS, or even plain Spring MVC.

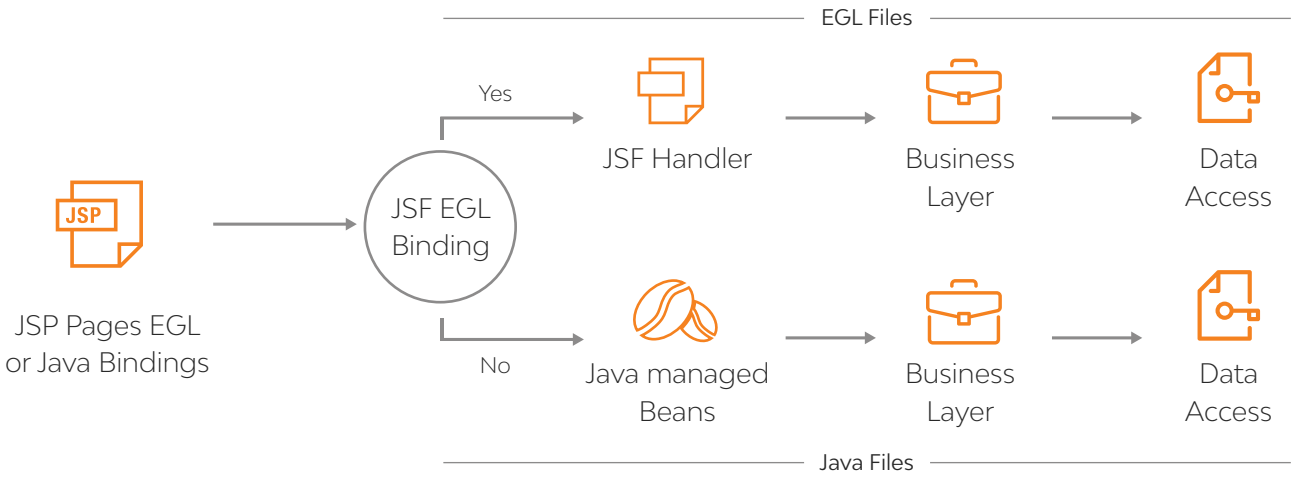
The whitepaper details a wrapper-based approach to solve the UI modernization issue. However, UI modernization is not the only goal. We need to have the ability to work with independent micro services across multiple technologies. It should also work with the existing EGL codebase until this gets migrated to independent business services in a phased manner or in sprints.

Introduction

Rational Business Developer IDE provides integration of EGL and JSF technology, producing an event-driven model in which a page-specific handler manages each request. 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 are written in EGL.

EGL Web Project allows creating a page only with JSF1.2 SunRI Spec (which is quite outdated now). For each JSP, there exists a unique EGL page handler, which includes variables and functions to process action performed on page. Business logic and database access logic are segregated as other layers. Thus, any action can go to the Handler-Business layer-Data access layer.

Current Operational Workflow



Current Status

The latest technical features couldn't be leveraged to support dynamic business requirements and, IBM will not be upgrading JSF support above JSF1.2 (SUN RI Spec), since it's already deprecated in RBD 9.1 version, and stopped from 9.5 (Latest Version) onwards.

As per IBM, there are two options provided – one is to reuse EGL libraries and the other one is to rewrite the entire application.

POC (Proof of Concept) is to verify and implement either one of the below approaches.

- Reuse EGL Libraries
  - Write JAVA wrapper class for all the EGL Handlers (Only for the attributes/variables tied to JSP)
  - Convert EGL JSF handlers to EGL REST Service & consume the REST service to render the screen
- Switch to EGL Rich UI

Proposed Solution Procedure

Below mentioned approaches are proposed after thorough research:

4.1 Approach 1: EGL Rich UI:

- 2 in 2005
- 40 in 2012
- 100 end 2016
- 140 end 2017

4.2 Approach 2: EGL Java Wrappers:

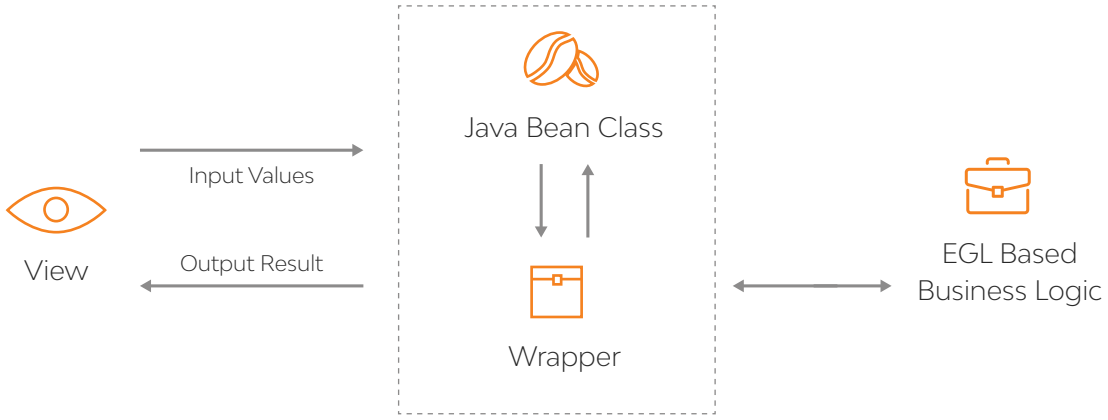
- View Layer: Approximately 25% change to replace EGL to java bindings. New web pages can be created in Java/JSP/Servlets.
- Control Layer: 10% change to convert JSF handler tag to EGL lib
- EGL Program: New Layer
- Business and Data Access layer no change

Rewriting involves enormous rework and testing efforts, and screens implemented with RUI can't be compiled and used together with existing application. Hence, the second approach is further explored and the Proof of Concept is built.

4.2.1 MVC Architectural changes for Approach 2:

The operational work flow can be broadly divided into three parts namely the –

- View
- Interactive unit
- Business Logic



Proposed Operational Workflow

Once the user enters the desired values and clicks on the desired action to be performed, the control passes to the associated bean class – with the values and the action requested to be performed.

The bean class also contains the instantiation of the wrapper class within itself. Thus, according to the action requested, the wrapper object would invoke a method of the parent EGL function associated to the action requested. While invoking the parent EGL method, we would be passing the input values with result set as parameters to the method. The wrapper is liken an interactive unit, which acts as a medium to invoke an EGL based from a java method.

The control passes to the EGL business logic and the appropriate business logic is executed, and the desired result is returned to the wrapper, which in turn, would be set as the result variable bound to the JSP, thus displaying the desired output to the end user.

Proposed Architecture:

Any UI interface enhancement requires to be modified as stated below. However, the new modified code will still work as it is with the existing application that is not changed, as the solution procedure guarantees the existing applications functioning without any impact.

View: Upgrade from JSF1.2 to JSF2.X using default JSF implementation or external user library. The view may be either a JSP or an XHTML.

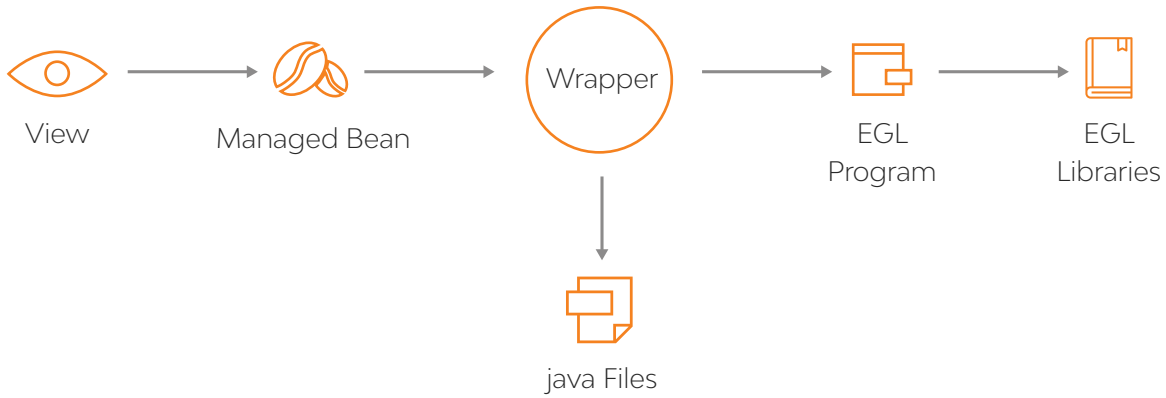
The existing EGL bindings were replaced by the Java beans.

Java Wrapper: A new layer called as EGL program is developed which would generate Java wrapper by changing the EGL build descriptor. The Java wrapper is an EGL generated Java file, which would allow interacting from Java to EGL. The Java wrapper will contain all the related variables bound in JSP. It would be instantiated from managed bean of the corresponding JSP page.

The user-initiated request would be passed to bean, which would be then passed to EGL program (using Java wrapper), and from there would call our existing EGL libraries.

JSF Handler: The existing JSF handler type would be changed to a Basic Library type.

Other EGL related files would remain the same.



Proposed Multi-tier Navigation

4.3 Approach 3: EGL REST Service

- View Layer: This approach proposes to create an EGL REST service and to consume the service, existing JSF pages and managed beans are to be changed by 15%. New web pages can be created in JSF/Java/JSP/Servlets or using the frameworks like Angular.
- Control Layer – JSF handlers to be converted to an EGL service, and 10% of the code changes are expected or a utility can be developed to do the conversion.
- There is no change in the business and data access layers.

Proposed architecture for Approach 3:

Any UI interface enhancement to the existing application using EGL JSFHandler requires to be modified as stated below. However, the new modified code will still work as it is with the existing application that is not changed, as the solution procedure guarantees the existing applications functioning without any impact.

View: JSF1.2 view and corresponding managed bean. The view can either be a JSP or an XHTML.

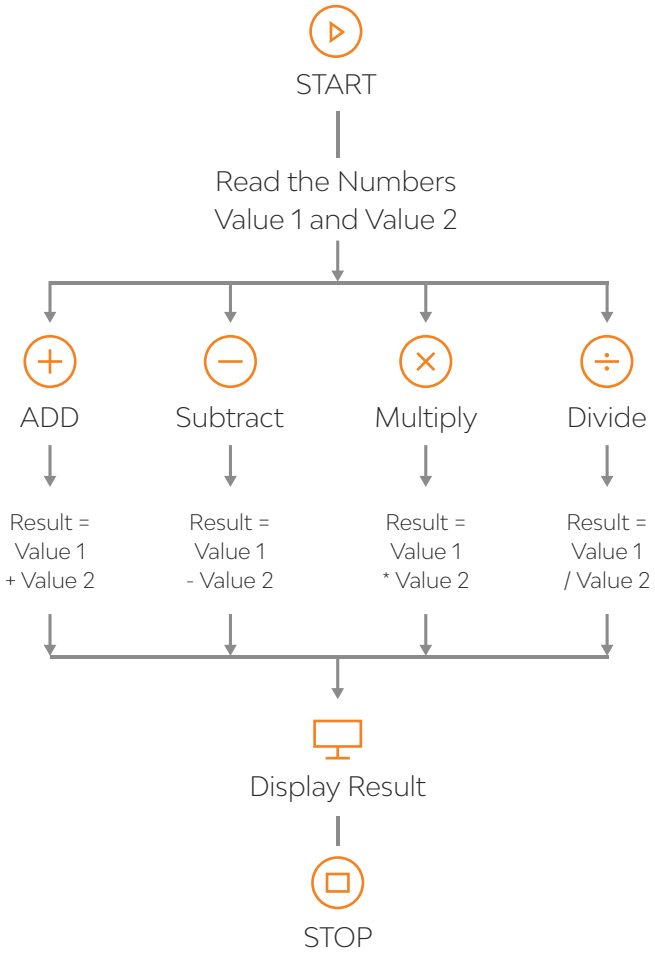
The managed Bean, leverages the reusable libraries to invoke the REST service. The user-initiated request would be passed to bean, which uses the REST client API to invoke the EGL REST service, which in turn invokes the corresponding EGL Business Logic.

Other EGL related files would remain the same.

Proposed Multi-tier Navigation

Current Implementation and evaluation of approaches

For a start, let us consider this example of implementing a simple calculator program. This program would allow user to perform addition, subtraction, multiplication and division between two numbers, and would provide the desired output.



Flow Diagram of a Basic Calculator Program

The view part of the application would consist of a JSP page, which would have two input elements Value 1 and Value 2, namely for user entry. There would be action buttons through which the user would select the desired arithmetical operation to be performed. The result obtained by the arithmetical operation would be displayed as an output text with variable name 'Result'. .

To migrate above application. We will consider the three approaches:

Approach 1:
Rewriting involves enormous rework and testing efforts, and screens implemented with RUI can't be compiled and used together with existing application. Hence, the second & third approaches is further explored and the Proof of Concept is built.

Approach 2:
The arithmetic logic is implemented using an EGL program. For every EGL-based project, a build descriptor file (*.eglbld) is associated. Within this build descriptor, we would have to provide a linkage, which would be generating the corresponding wrapper class associated to the program.

We can generate the Java wrapper classes by using a build descriptor that has these characteristics:

- The enableJavaWrapperGen build descriptor option is set to yes or only.
- The linkage build descriptor option references a linkage options part that includes a callLink element to guide the call from wrapper to program.
- The callLink element, remoteComType element is set either to DIRECT or DISTINCT. In either case, the native Java code and the EGL-generated program run in the same Java Virtual Machine. The call is remote because EGL middleware is involved, as necessary to handle data conversion between the Java wrapper and the EGL-generated program.
- The main program can complete a transfer to program, but not a transfer to transaction.

This wrapper class once generated should be instantiated within the bean class associated with the View page. In this bean class, invoke the wrapper class methods to make the actual call to the EGL-based functions.

We should ensure all the functions which can be invoked from presentation layer, are dealt with case statement. This is to add simplicity within the code to ensure invoking only the necessary operation to be performed.

Approach 3:
The arithmetic logic is implemented using an EGL program. This EGL program is exposed as REST service. In the Managed Bean, invoke the REST service methods to make the actual call to the EGL-based functions. The Managed Bean corresponding to JSF page uses the REST client APIs to invoke the corresponding functions exposed in the EGL REST Service.

For the EGL-based project, a deployment descriptor file (*.egldd) is associated. Within this deployment descriptor, we need to configure Service deployment to generate REST services.

Conclusion

The prototype developed effectively achieved the objective

- To use wrapper classes to interact with Java Bean and EGL class libraries. This approach will allow loose coupling of presentation layer available within various technologies that can communicate with POJO classes. Thus, introducing the flexibility of using any UI, that supports Java.
- To use EGL REST service to interact with the EGL business logic. This approach used loose coupling of presentation layer and enables use of any technology that can consume REST service to render the web page.

Further to this, authors have to conduct a feasibility test for confirming whether EGL plugin can be used with existing Java-specific IDEs (e.g. Eclipse).

At the same time, it is extremely important for the business to continue operational, and rapidly rollout features into production, ensuring that the new capabilities work well with the existing implementation. When business requirement changes often and quickly, technology is expected to deliver with the same pace and to improve the speed-to-market.

References

IBM Knowledge center,
"How to generate Java Wrappers"

last accessed August 01, 2016

IBM Knowledge center,
"Java Wrapper Classes"

last accessed August 01, 2016

IBM Knowledge center,
"Class Loaders"

last accessed August 06, 2016

Web Tools Platform,
"Creating a J2EE Application (EAR) Module"

last accessed August 08, 2016

IBM Knowledge center,
"EGL REST-RPC message structure"

last accessed June 14, 2018

About the Author

Jayamsakthi Shanmugam
Head of FSTI- Product Delivery - India

She is a Business savvy, process oriented and positive minded go-getter with a strategic approach to problem solving and with highest appreciation for values and dedication. She obtained her Ph.D. from Birla Institute of technology and Science, Pilani. Her research interest includes software architecture, web security and frame work assessments.

Ravi Bhardwaj
Project lead

He has a Bachelor of Engineering (Electronics & Communication) from V.T.U. His area of interest includes Graphical User Interface design, learning new technologies.