Oracle® Forms
Modernization Through Automated Migration
A Technical Overview
# Table of Contents

Document Overview .......................................................... 3  
Oracle Forms Modernization ............................................. 3  
Benefits of Using an Automated Conversion Tool .................... 3  
OpenText Composer CipherSoft (Features and Key Benefits) .......... 4  
The Converted Application from User Perspective .................... 4  
The Converted Application from Developer Perspective ............. 7  
Overview of the Generated Architecture ............................... 9  
About OpenText ............................................................... 11
Overview

This document contains details on the OpenText™ Composer CipherSoft solution for Oracle Forms migration to Java. The intent is to provide a technical overview of the solution and the architecture.

Oracle Forms Modernization

Organizations have invested substantial resources building very successful applications in Oracle Forms, but are looking to modernize their applications with today's technologies to keep them relevant. Several options include: web enabling the application through an upgrade, rewriting the application, and automated migration. Each approach has pros and cons depending on your requirements. OpenText Composer CipherSoft provides solutions for modernizing Oracle Forms around all of these options. We have extensive experience in upgrading and converting Oracle Forms, databases, and reports for clients around the world.

Benefits of Using an Automated Conversion Tool

There are two options for migrating Oracle Forms applications to Java: One is to rewrite the application, designing new screens, screen flows and work processes from the ground up based on the use cases. The other approach is to migrate the Oracle Forms screens to Java, keeping the same look and feel, and allowing the users to continue working on the migrated environment as they did in Oracle Forms and then (after a testing period) change the look and feel if necessary.

The first approach—rewriting the application—can be risky, expensive and lengthy. This is particularly true for large and complex applications. All processes need to be re-examined—it is the equivalent of starting a new application development project. There is the risk of leaving out processes, data migration failures, discontinuation of user processes, and difficulty performing a parallel run to compare the results of both systems.

The one-to-one migration is proven to be significantly faster, more cost effective, highly achievable, practical, and less risky. Why?

Stay competitive

• Standardize to more open and independent platforms for building and maintaining business applications.

Significantly reduce costs

• Automating the process of conversion significantly cuts the costs and time required to manually convert applications.
• Saves 90 percent of the time and 80 percent of the costs with an automated conversion.

Lower risk

• The automated process greatly lowers conversion risks by removing human error. Converted applications are fully maintainable and flexible, making it simple to modify converted applications when necessary.
• Converted applications keep effectively the same UI and functionality as the original Forms application so that users don’t need retraining. This reduces disruption and impact on business operations.
OpenText Composer CipherSoft

Automated conversion of Oracle forms & PL/SQL to Java and Java EE

OpenText Composer CipherSoft is an automated tool that converts Oracle Forms and PL/SQL applications to open and standards based technologies, such as Java, XML and browser-based solutions. OpenText Composer CipherSoft will also convert the PL/SQL within the database to standard Java classes or leave it in the database unchanged, and generate the appropriate code to invoke these program units from the converted Form, library and menu modules. The Forms themselves are converted to XML-based displays that run within the Java based interface. The product is highly automated, requiring little to no manual coding.

One of the main benefits of CipherSoft is it is available as a software license as well as a service.

Features of OpenText Composer CipherSoft

• Automatically converts Oracle Forms menus, triggers, stored procedures and libraries (including all PL/SQL code within libraries)
• All PL/SQL code can be exposed and presented as complete Java classes
• Produces native Java that is 100 percent maintainable and completely portable
• Maintains semantic content and coding standards
• Does not create wrapper classes

Key benefits of using OpenText Composer CipherSoft

• Shorter learning curve for Oracle Forms developers to maintain and extend the Java code, as conversion maintains the structure and hierarchy of Oracle Forms in Java
• Three GUI options: Browser with a Java plug-in, Oracle ADF, Java desktop client
• Provides Java and XML code that is portable, maintainable, and reusable
• Enables the use of Java-specific development tools available today and in the future (i.e., designers, modelers, testing tools, source code archives, change management software, application servers, JSP and JSF integration tools, Java compilers, etc.)
• Preserves component names (Forms, menus, libraries); visual item names (canvas, items, buttons); and Triggers

Converted Application from a User Perspective

To allow for business continuity, the generated Java application needs to provide the users with the same experience as Oracle Forms so they feel comfortable in the new environment and perform their tasks in a similar fashion to which they are accustomed. They also need a way to compare the converted environment with the old one (User Acceptance - Parallel Testing).

OpenText Composer CipherSoft converts the user interface to an identical look and feel. This minimizes any retraining of end users. However, if a new look and feel is a key objective of the initial migration project, customer defined style sheets can be used to modernize the interface as part of the conversion project.

In comparison to the original Oracle Forms application (Fig.1 and Fig.3 on the next two pages), you can see how the new application in a browser (Fig.2 and Fig.4) is very similar to the original with a few minor differences in fonts and color. Otherwise they are nearly identical. All of the functionality is maintained, including the list of values, hot keys, event handling mechanism, toolbar, default menu, validation logic and so on. All of this is done automatically through the converter.
FIGURE 1

The Original Oracle Forms Application – Customer Form

FIGURE 2

The Converted Java JEEE Application – Customer Form
FIGURE 3
The Original Oracle Forms Application – Order Form

FIGURE 4
The Converted Java EE Application – Order Form
The Converted Application from a Developer Perspective

The next set of screen shots (Fig. 5 through Fig. 8) compares the design components between the Forms application and the converted Java application from a developer perspective. Note the similar folder structure and hierarchy. This makes it much easier to navigate the converted application for maintaining and extending the code. It also ensures the proper sequence of trigger execution.

**FIGURE 5**

The Original Oracle Forms Application (Order Form) in Forms Builder

**FIGURE 6**

The Converted Java EE Application (Order Form) in JDeveloper
You will notice there are additional lines in the generated Java code, which provides new functionality compared to the original application. With a new trace flag introduced, the sequence of Oracle triggers converted to Java event handler methods can be monitored.
Overview of the Generated Architecture

- Because of the complexity of the Java EE API, it was important for OpenText Composer CipherSoft to create a simple architecture that could be easily understood and maintained, and allows for easy handling of the business rules without dealing with the technical aspects of the application.

- Oracle Forms knowledge is rewarded but not necessary: Keeping the terminology, object and method names, class hierarchy helps tremendously in reducing the learning curve for Oracle Forms programmers and makes the transition to Java as smooth as possible. The automated migration requires significantly less Java, Java EE expertise and will produce a consistent and easy to read application that is easier to navigate and maintain.

- Model View Controller design pattern: The application generated by the Conversion Tool divides into three logical categories of objects (MVC Design pattern). Some objects are responsible for the presentation aspects of the application. Some are responsible for the business rules and data, and some objects handle user requests and control the business object to serve those requests (See Fig. 9 and 10).

  - **Presentation** – Web based technology provides a rich client experience to mimic the Oracle Forms behavior and to retain the original look and feel of the application (business continuity and user acceptance).

  - **Controller Servlet** – A component that captures the events sent by the View and routes them to the proper class in the model.

  - **Model** – Represents the application data along with the methods that operate on the data:
    
    - Business logic specific Java code (business objects/model) for the PL/SQL defined in the form, menu and library modules;
    
    - XML files for the presentation, persistence and controller layers; and

    - General, reusable, common Java classes that are used by the business logic specific Java code (abstraction, inheritance, object oriented design).

- **UI Independence achieved**: Since the Java EE architecture and the Model-View-Controller (MVC) design pattern played a key role in the product design and implementation phases, the presentation layer (View) can be changed or replaced relatively easily without changing the application layer. The generated Java code (Model, Business Logic and Persistence Layer Components) for the plug-in version and the HTML version (View) is 100 percent identical. This can only be done with clear separation of the tiers.

- **Java EE architecture**: The generated application is a 100 percent Java/Java EE application. In the generated architecture, the key concepts and fundamental principles of object-oriented programming, such as classes, inheritance, interfaces, objects, packages, abstract classes, polymorphism, method overloading, Java data types, Java naming convention, well-known Java EE design patterns, and so on can be easily identified.

- **High level of abstraction achieved**: A good system architecture leads to reusable components. Components and functionality used repeatedly in the design have been added to a reusable Java class library. This high level of abstraction hides the implementation details, performs most of the plumbing tasks for the programmer, and enables the programmer to focus on the business logic. These components handle the trigger execution logic (event handlers), global and system variables, synchronization between the fields on the screen and fields in the database (data control and binding services), and provide an easy to use API. The API is documented and available to customers.

- **Declarative Development**: One of the prime objectives of the architecture was to enable application development with as little manual coding as possible. Presentation layer, persistence layer and workflow components can be configured through XML files without recompiling and redeploying the application.
FIGURE 9
Application Architecture

FIGURE 10
The resulting code when converted into Java in Eclipse
About OpenText

OpenText enables the digital world, creating a better way for organizations to work with information, on premises or in the cloud. For more information about OpenText (NASDAQ: OTEX, TSX: OTC) visit opentext.com.

Connect with us:

- OpenText CEO Mark Barrenechea’s blog
- Twitter | LinkedIn | Facebook