Converting CA 2E (Synon) Code to Java on IBM i

A look at one of the leading code modernization solutions

Abstract

This white paper briefly describes the advantages of modernizing CA 2E (Synon) on IBM i. It includes IBM's evaluation of the maintainability of Java code converted from RPG by Fresche's X-2E Modernize solution. The X-2E Modernize solution converts and generates native, object-oriented Java code in an MVC Design pattern with RESTful interaction. CA 2E is ideally suited for conversion to Java as it contains a highly useful design model. The white paper also includes an overview of Fresche's conversion process and an architecture diagram for the converted code.

Modernizing CA 2E code

One of the best ways to take advantage of modern computing opportunities is to modernize code. Since its introduction, CA 2E (Synon) has been one of the most successful 4GL tools on IBM i. The CA 2E development environment combines a rich and precise model of designs and specifications with powerful code generation capabilities. Even with this powerful set of features, many companies now face real pressure to modernize their business applications beyond the scope and capability of what CA 2E can offer. The challenge is to move forward without discarding decades of investment in design, evolution and fine-tuning stored in the CA 2E model. Converting CA 2E applications to Java is one method of modernizing valuable code to take advantage of new opportunities. IBM Systems Lab Services recently evaluated the quality and maintainability of Java code generated from CA 2E by Fresche's X-2E Modernize solution, which combines tools and services to significantly automate the process of converting code to Java. X-2E Modernize automatically refactors and generates an MVC web application using object-oriented methods in Java. The solution takes advantage of CA 2E application architecture to automatically convert applications. Since CA 2E forces users to organize code into screen designs, action diagrams, business logic and database components, X-2E Modernize uses this internal 2E model to extract design information in a structure that maps cleanly into a modern MVC/OO/RESTful architecture.

IBM evaluation of Java code converted from CA 2E by Fresche

IBM Systems Lab Services was tasked with evaluating the maintainability of the Fresche Java code created by the X-2E Modernize solution. This evaluation took place from May 11 to June 6, 2016. While the main focus of this evaluation was the maintainability of the back-end Java services, the client technologies were also reviewed to get a perspective of the entire application. The IBM i customer perspective was kept in mind throughout the evaluation, as this is the target customer for X-2E Modernize.

Evaluation area

Methodology and summary of findings

The evaluation of the Java code for maintainability centered around four main criteria, taken from https://quandarypeak.com/2015/02/measuring-software-maintainability.

- Is the code understandable?
- Is the code modifiable?
- Does the code meet requirements?
- Is there a test harness?

The following tables summarize what is required to satisfy each criterion and what was found in the provided code.

Summary of Findings

Was the converted code understandable?

Criteria	Findings in converted code
• Is the code easily readable? • Is it self-descriptive? • Is it documented where needed? • Do the classes serve one and only one purpose? • Are the sizes of the methods manageable? • Do the frameworks used encumber the understandability of the code?	• The overall quality of the generated code is good. Both the REST services and the JPA implementation are very similar to how a Java developer would have coded them. • The classes served one purpose. • Method size was manageable. • Developers will need to be familiar with Spring and angular.js • Documentation did not appear to generate/install properly in all cases

Was the converted code modifiable?

Criteria	Findings in converted code
• Is the code easy to follow? • Does the code avoid duplication? • Are the packages generally cohesive? • Are the inheritance hierarchies not overly deep? • Does the code favor composition over inheritance where applicable?	• A skilled Java programmer with knowledge of the Spring components used will be able to support, maintain, re-factor and enhance the generated application. • The repeatable pattern created by the tool should make problems easier to fix, and allow for the application to be enhanced rather quickly by duplicating the pattern. • The packages built have a consistent pattern that is easily reproducible and understandable. • Rational® Analyzer metrics suggest the application in general is loosely coupled, so this should make re-factoring the code easier. The size of the methods is also a key contributor to maintainability. • Logging was mostly used for warnings; level should be adjusted. • Knowledge of Spring REST Controllers, Spring Authentication Mechanism and Spring JPA are essential; • Knowledge of angular.js framework is also essential.

Does the converted code meet requirements?

Criteria	Findings in converted code
• Does the code meet requirements?	• The general overall architecture provided by Fresche follows the specifications in the Fresche Java Reference Architecture document. Specifically, the generated application maintains the look and feel of the existing application, but it uses modern technologies. This has been well executed by the transformation tool. • The Fresche Java Reference Architecture document itself could be improved with more technical detail. • The Java architecture generated by Fresche's transformation tool is a viable and modern architecture.

Testing

Criteria	Findings in converted code
• Is there a test harness? • Are there runtime logging capabilities?	• There was no test harness as part of the original application, so no harness was brought forward in the conversion. • IBM recommends that logging levels in the transformed Java code be adjusted. Currently, logging appears to be used only for warnings.

Tools used to assist in the evaluation

1. PMD tool to check for Java Code problems
2. Rational Software Analyzer Edition 7.1. This provided some Java Metrics on cohesion, coupling, and code complexity.

Evaluator skill set

1. Evaluator has worked on several web applications through the years, which used the following technologies: Struts, JSF, JPA, JDBC, HTML, CSS.
2. Used Spring for JMS implementation on one project. Used the applicationContext.xml file for configuration.
3. The two previous web application projects used JQuery, JQuery-Mobile, Dojo, JavaScript, HTML, CSS, Java REST services and JDBC.

Fresche evaluation sheet from IBM Systems Lab Services

Quality of the generated code

How close is this code to resembling something that might have been hand rewritten by a person?

The overall quality of the generated code is good. Both the Rest services and the JPA implementation are very similar to how we would have coded them. The PMD logs sometimes contain the same warnings, which should be examined. The few we see where additional examination may be of some benefit are:

- "Local variable could be declared final"
- "Parameter is not assigned and could be declared final"
- "Uses StringBuffer instead of +="
- "Potential Violation of the Law of Dementer" – This may indicate tight coupling which tends to be less maintainable.
- "High amount of different objects as members denotes a high coupling" (eight instances)

What do you like about the generated code?

The packages built have a consistent pattern that is easily reproducible and understandable. The classes served one purpose. JPA entities and JPA Repository objects were packaged together, and Spring Controllers are packaged in hierarchical package names. This makes it easy to associate the REST services with the JPA services.

Typically, when we used JPA, the JPA objects were built in a JPA project and wizards were used to build the Java code. The separation of the JPA objects in separate packages serves this purpose.

Since this is being built by the transformation tool and Spring JPA annotations are being used, this package structure makes more sense. This is one of the reasons for the low lack of cohesion numbers generated by the Rational Software Analyzer. The only relatively high numbers appear to be caused by the JPA entity objects, which is a result of using JPA.

Another indicator of easily maintained code is method size. For this project, the method size was manageable.

Maintainability/Style of the generated code

Using modern, standard tools, how easy would it be for a skilled Java developer, with some source application knowledge, to take this application on and support, maintain, refactor and enhance it?

A skilled Java programmer with knowledge of the Spring components used should be able to support, maintain, refactor and enhance the generated application. Knowledge of Spring Rest Controllers, Spring Authentication Mechanism and Spring JPA are essential. Understanding the Spring Java configuration and Spring configuration annotations (instead of the using the application-context.xml configuration file) is also required.

Even a skilled Java programmer without Spring skills should have no problem maintaining this code after some preliminary ramp up time to understand the Spring interfaces that are prevalent throughout the code. What about this code contributes to its ease of maintainability? There is a repeatable pattern and consistent output that is created by the tool. This should make problems easier to fix, which should allow for the application to be enhanced rather quickly by duplicating the pattern.

The Rational Analyzer metrics suggest the application in general is loosely coupled, so this should make refactoring the code easier. The size of the methods is also a key contributor to maintainability.

Architecture of the transformed application

Keeping in mind that we're coming from an IBM i platform, how viable and modern do you consider the new architecture to be?

The general overall architecture follows the specifications in the Fresche Java Reference Architecture document. Specifically, the generated application maintains the look and feel of the existing application, but it uses modern technologies. This has been well executed by the transformation tool. In Fresche's case, customers want to keep the same look and feel as the green screen application, so the Fresche modernization strategy is entirely appropriate.

On our modernization projects and in our workshops, we find that our customers modernize because they want more real estate on each screen or have a responsive UI that can work on a mobile device. It is understood that the transformation tool does not have the capability to accomplish this task. Changing the UI would be handled as a follow-on service by an integration team. The architecture does make these changes easier to achieve.

The Java architecture generated by Fresche's transformation tool is a viable and modern architecture. A common architecture we have seen/used recently is one that employs single-page frameworks as client technology, web services as middleware and JDBC for the persistence model access. We have had several customers use angular.js for their client-side UI technology. As for web services, it appears the industry is moving toward REST services, and JPA is a viable and well-used persistence model access mechanism.

Quality of the generated interface

Would you use such a solution and/or recommend it to others to use?

This is a SYNON-specific solution. We typically do not do a lot of work with SYNON customers. That said, we would recommend this solution for customers who have an edict to move solely to Java and who want to keep the same look and feel as the green screen application. Even for those who don't want the same look and feel, we would recommend this solution as one of the options to shorten the development cycle by building the Rest and JPA services for the SYNON model.

With regard to the Fresche Java Reference Architecture_23102015- MP1.docx:

The application architecture document could be improved for IBM i customers. A discussion about the Java pattern (including package naming conventions) would be beneficial from an education perspective. We also suggest adding a bit more technical information about the Spring components used. In addition to the short explanation on Spring authentication, maybe include a how-to on conversion using LDAP or other mechanisms. It may be beneficial to also include information about the Spring Java configuration and how some of the annotations are used. Finally, it would be helpful to include some discussion about how the CRUD JPA services are enabled. Specifically, mention JpaRepository, and how the underlying implementation is SimpleJpaRepository. Note that this will be more for the IBM i customer who is new to Spring and not the Spring expert, but it may be best to target the least common knowledge base. An additional note: there is a statement that JPA is more efficient than creating the SQL manually. You may want to clarify this statement.

General comments/summary

In evaluating the Java code transformed from Synon by X-Modernize, the code in general is maintainable. The patterns that have been created should be easily duplicated by most Java programmers. Unless the developers are extremely knowledgeable with the frameworks used (mainly Spring and angular.js), there will be a bit of a learning curve to get comfortable with enhancing and maintaining the code.

Fresche's conversion process

This section briefly describes Fresche's automated approach for conversion of CA 2E to Java.

Target reference architecture

A common objective of modernization projects is the implementation of a modern architecture. The most widely accepted design principles utilize the concepts of multi-tiered components, developed as objects, in a model-view-controller (MVC) pattern. This architecture helps achieve higher-level goals of scalability, modularity and maintainability and supports the use of agile development.

Fresche's modernization approach transforms CA 2E to a highly scalable, industry best practice, Java Linux reference architecture using automation through the X-2E Modernize toolset. The screens are transformed from DSPF to AngularJS, the latest web UI technology for Java applications. The application security and access control framework is also reproduced.

The business logic is transformed into business logic services, which can easily be enabled as callable services, if required. Fresche's database transformation tools can be used to migrate the database to a new, more modern database target, and the X-2E Modernize Suite can generate the Java necessary to run the application against the newly transformed database.

The modernized application is transformed into an n-tier architecture, broken into three distinct layers: presentation, business and data access. This leverages a proper MVC architecture.

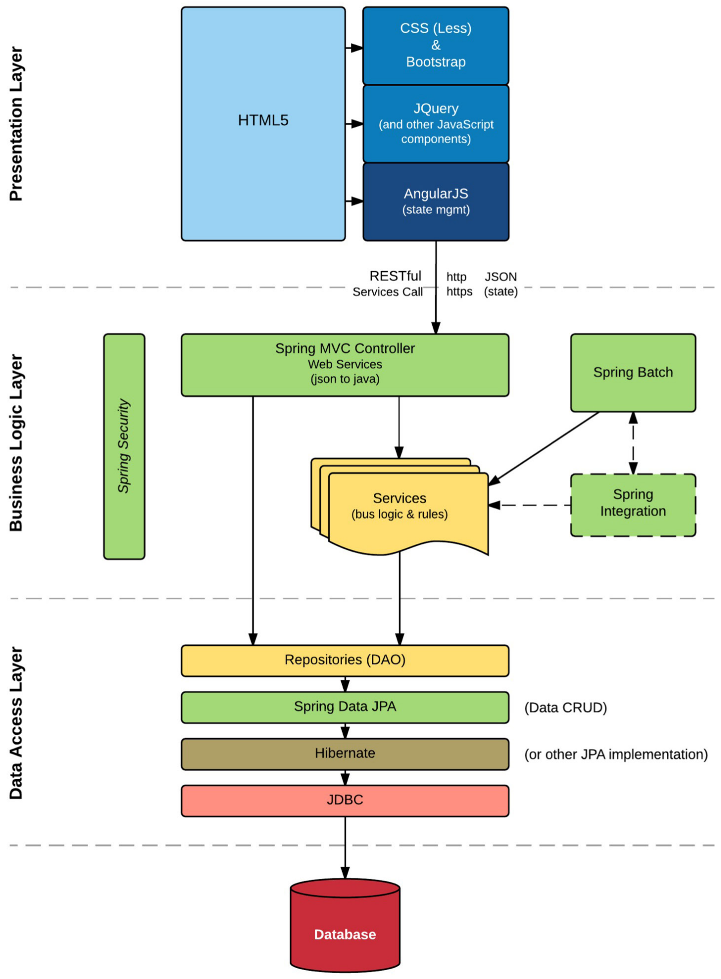
The following are the characteristics of the code once it is transformed into Java:

MVC design pattern – Model-View-Controller, as in a model layer of components that contains business logic, a view layer that contains the outward presentation and a controller layer that handles events in the other layers and directs process flow.

Object orientation – Organization of code into objects (classes), with those classes containing functions. The functions are either callable from other classes or protected so they can only be called from within their own class.

RESTful interaction – Server components have no inherent knowledge of session state (stateless). Session information that needs to be preserved between workflow activities (screens) is preserved and represented from client-side memory or through session management functions within the application server software.

The following diagram depicts the Java reference architecture into which the application will be transformed.



From loosely structured to modern architecture

To get a high-quality result, legacy program modernization cannot be accomplished by simply converting from one language syntax to another. The modernization of the application is done using tools to automate the conversion of existing data and program objects into a fully relational database and Java objects. The conversion effort includes the tuning of the tool to maximize its automation effectiveness in converting the programming patterns and practices that were used by the original authors of the application. This tool automation optimization phase is an iterative process where the code is repeatedly run through the converter and the output is verified for pattern-based optimization opportunities.

Conversion steps

Briefly, there are four main steps in the conversion process:

1. Understand and extract the legacy artifacts using X-2E Analysis

X-2E Analysis, a commercially available analysis tool for the IBM i, is a key component of Fresche's modernization process. X-2E Analysis provides analysts, developers, architects and operations teams with detailed analysis and interactive diagrammatic constructs that enable rich understanding of existing applications. X-2E Analysis extracts the details of the existing CA 2E model, providing an excellent base for efficient and effective design recovery, system documentation and analysis, and forward-engineering. This extraction phase is the first major step in the modernization process. The legacy application artifacts are housed in a repository that is later queried by X-2E Modernize to reconstruct the new application into the target architecture. This step can save as much as 30 percent from a traditional rewrite project, not to mention the benefits on quality that an automated process will bring.

2. Transform and migrate the database

Fresche's database modernization solution is highly configurable and handles the unique problems presented by IBM i DDS and DDL databases. The solution provides data integrity validation information that provides an audit of the data migration, proving all data was moved. The solution is built on the X-2E Analysis product platform and leverages the data model produced by X-2E Analysis in its generation capabilities. The solution supports multiple target databases and is aware of unsupported features that impact the migration and can validate reserved words and recommend alternatives. The solution can launch and manage multiple parallel (peer-to-peer) database migrations simultaneously.

3. Transform into the Knowledge Discovery Metamodel (KDM)

The goal of the Knowledge Discovery Metamodel is to ensure interoperability between tools for maintenance, evolution, assessment and modernization. KDM is defined as a metamodel that can also be viewed as an ontology for describing the key aspects of knowledge related to the various facets of an enterprise application.
While other tools claim to generate modern, maintainable code, this cannot be achieved using a simple line-by-line parser to code converter. The KDM is used in order to house and represent the future modernized application in a format that facilitates its generation into a properly structured MVC architecture, thus abstracting it away from its original legacy state.

4. Generate the code using X-2E Modernize

When the KDM representation of the system has reached the desired state, the target application source code is generated from it. The cycle from KDM to code generation is repeated during the early stages of the modernization project. This is called the "tool tuning" stage and can take several months to achieve optimal benefits. The resulting generated source code (for example, Java) will then likely need to be manually tweaked and perfected by the Fresche modernization specialists.

It should be noted that details of the target platform (Java, .NET, PHP and so forth) have been abstracted away so that the component that reads from the KDM can be reused for other targets. Also, the source of this architecture is the structured, normalized X-2E Modernize repository. The significance of this is that the generator tool can be customized as necessary by Fresche to accommodate different standards, frameworks or functional needs. There is a cost associated with this customization, but it does provide for a flexible end result that much more closely meets a customer's optimum target reference architecture, without the need to change any of the extraction tools or data. Any functional application enhancements needed by the client would be planned and typically prioritized so as to minimize risk, disruption and overall effort and cost.

For more information

To learn more about IBM i and the supported IBM server platforms, please contact your IBM marketing representative or IBM Business Partner (BP) or visit the following websites:

ibm.com/systems/power/
or
ibm.com/power/i/

To learn more about Fresche and X-2E Modernize:
freschesolutions.com

QLW12351USEN-00