Why Clarion2Java?

1 Introduction

This document outlines the case for clarion2java. It is written by the original project creator of Clarion2Java and it provides a personal narrative as to the motivations and chain of events that lead to the inception of this project.

2 Do not throw away code

There are a number of reasons why I started this project, which I will list below shortly. But there was one overriding principal driving this entire project – value your software assets.

I develop point of sale software for motorcycle retailers in Australia. For over 15 years the solution provided by C8 Software has been based on Clarion. From Clarion DOS, to Clarion 2.0, to Clarion 5.5. Each progression from one version to the next was driven primarily by growth in expectations from our customers, and along that growth path we continued to use Clarion technologies. Prior to Clarion, we used a business basic language, called AB-86. Every major technology shift required us to minimally rewrite major sections of the software, if not a complete rewrite.

For a number of reasons, our needs outgrew the capabilities of Clarion 5.5. I will list these reasons shortly, but what these are are not critical : because it is an inevitable aspect of the software engineering lifecycle which we have been through multiple times already.

We looked at upgrading to Clarion 6 and Clarion 7 – but were not entirely confident that this move would address all of our issues. We also looked at Clarion.NET. But our immediate problem with Clarion.NET was that it was not a faithful re-implementation of the Clarion Language Specification and that we would be forced to rewrite large tracts of our software.

This was not acceptable to us. It is a common anti-pattern in the software industry, one which we have gone through ourselves multiple times now : if your business needs outgrow your current technology stack then the way forward, even if you stick with the same stack vendor (i.e. Clarion 5.5 to Clarion.NET) is to substantially rewrite your code for a new stack.

But the software represents years of effort and investment and painstaking testing and bug fixing. Why throw all that hard one knowledge and working asset away?

Also the customers are used to how the software behaves. They are used to the fact that at a certain point of Point of Sale use case for example, you press F2, Enter discount, F10, Enter cash tender, F10, Down Arrow, Enter and they do it automatically, reflexively. To rewrite the code in a new system will involve either painstakingly ensuring that user experience matches perfectly, or alternatively risk upsetting your customers.

This is the basis of clarion2java. I did not want to solve problems I already solved. I wanted to give my customers a new system which closed all our open issues but looked and felt the same.

So instead of rewriting code, solving problems I already solved – I decided to extend the capabilities of the product – by adding, not replacing. I added a compiler and runtime system.

This doesn't logically mean that, had you programmed in cobol in 1970, you should still be programming in cobol come YEAR 2040. But providing a transitional migration path allows us to gradually migrate more and more from Clarion to Java provides us options to keep the software asset modern and relevant, but not undergo the risk and expense of a rewrite. Right now we are writing more and more code in java. We will continue to use clarion where it is useful and we will use java were useful.

2.1 Clarion and Java – best of both worlds

Clarion 5.5 is a great environment. I like working in it, and I would like to continue to work in it – because it is simple, rapid and intuitive.

Clarion2java gives me the best of both worlds.

I continue to do the bulk of my programming in Clarion – because it is a great language for expressing business logic. The app generator is excellent.

Yet there are times when I want something closer to general purpose programming, something that does graphics (charts), networking and other sophisticated concerns like cryptography efficiently and simply. I want to be able to tap into one of the worlds most popular programming languages, into a language which is supported by a vibrant culture of open source so that I can reuse powerful technologies and libraries, but with the added benefit of not having to pay licencing for the right to use those libraries.

The java itself is beginning to expand number of supported languages that can co-exist in the Java Virtual Machine (JVM). So there is scope for considering mixing even more languages, such as Scalar, JRuby, and JPython just to name a few.

2.2 Issues with Clarion 5.5

Firstly, all technology stacks have their strengths and limitations.

Clarion 5.5's limitations were beginning to impact us.

2.3 Application Stability

Number one was runtime stability both database and the program itself. The application would regularly crash and generate a GPF deep within the bowels of the system. Without access to source code for C55RUN.DLL for example, we had no idea what was causing it. We found a correlation between performance/age of the desktop computer and the issues we were experiencing. We also determined that disabling graphics acceleration seemed to alleviate the problem. But not fix it.

We considered going to Clarion 6 and 7 to get relief but we were not confident that this would fix the problem. Trawling through change logs on SV website we could not find a specific fix. Also there is a risk we would inherit a new set of problems.

2.3.1 Database ACID

For years, since that days of Clarion for DOS, we have been using Clarion .DAT files. .DAT files are just simple ISAM files with attached b-tree indexes. They do not journal/transaction well, they do not provide ACID. But our application was beginning to do more and more complicated operations which required ACID and more and more shops were running multiple terminals. Key files would regularly corrupt. And occasionally databases would lose data or lose referential integrity as a consequence of undetected Key file errors.

We considered TopSpeed system some years ago : TPS. But going to TPS would mean forgoing very valuable tools like cscn and cfil for which TopSpeed had little in way of an analogue. Especially cfil which is a very important tool for managing database configuration and upgrades.

For years we recognised the need to goto a system like PostgreSQL – which I have had alot of positive experience with in other projects.

2.3.2 Migrate to PostgreSQL - problems

We migrated to PostgreSQL over a period of 3 to 4 months. The experience was not positive. A number of issues:

- Performance. certain browse screens in particular were unacceptably slow. By inspecting the SQL the ODBC file driver submitted to postgres we could see a number of problems which we know we could fix if we could write the SQL ourselves. But that would involve not using Clarion browse wizard and rolling our own. Not desirable.
- Complexity. Installing a new terminal was too complicated. You had to install the

ODBC driver and configure it – including configuration of non default settings such as using Cursor Fetching (in order to try and claw back some performance). If a customer were to install a new terminal or have a disk failure on a terminal we simply could not count on local community IT shop/support from being able to manage the corrective procedures to get the terminal working again.

2.3.3 Change in Business - Networking and Analytics

Business is constantly changing. 10 years ago, no motorcycle shops were connected to the internet. Now every shop has a permanent ADSL connection, they want to be able to email customers, SMS customers, integrate into factory electronic stocking and ordering systems etc. Factories started hiring companies to do best practice audits which demands more sophisticated data access and analytics. None of the things we could cope well with with Clarion 5.5. Networking is difficult and clumsy, data analytics is difficult in Clarion 5.5.

2.3.4 clarion2java

With all of the above – java was a natural fit for us.