1/27/2016

EnterpriseIQ for the Web

Development Strategy

David Izada R, Gabriel Forner
IQMS

Contents

Introduction

To execute the migration from the current desktop version of EnterpriseIQ to a web application based on UniGUI in the shortest possible timeframe – while providing enough assurances to our customers and shareholders – we should define and follow a precise project plan.

The final plan must be the result of our due diligence. Instead of trying to start a massive migration as soon as possible, we need to take into account every factor possibly affecting the expected outcome and devise a plan incorporating that information.

EnterpriseIQ (current desktop version)

Our current desktop application is composed of several modules integrated into one main application (IQWIN32) and a few independent applications. Several factors can affect the migration:

1. It uses several technologies for accessing the database (BDE, dbExpress, and FireDAC).
2. There is a very heavy RAD-style programming in use which focuses on visual components. This includes a substantial amount of business logic being placed in the front end. There is no dedicated business layer or separation of responsibilities. Although the ORACLE database itself takes care of some business rules, there is a large quantity of business rules spread among data modules, forms, and supporting units.
3. The code base is almost compatible with Delphi 7 which means the architecture and code style is circa 2002. The significance of this is that we are not benefiting from modern programming practices and techniques, including such features as Generics, Anonymous Methods, attributes, the extended RTTI, nor are we utilizing tools like DUnitX or Spring4D.
4. Our shared components, both non-visual and visual (IQVCL), are supporting all types of data access technologies, making them complicated, with cross references, and no clear separation of responsibility. This in turn consumes more resources.
5. The application depends on several 3rd-party components, many of which are outdated and no longer maintained.
6. The biggest issue is a highly complex interdependence throughout all of the various modules in the system: units using other units (not just the common units), and classes inheriting from other classes (including forms).

Ultimate Goal (it could be partially achieved)

The best possible outcome for "EnterpriseIQ for the Web" could be the following:

1. Migrate to a single data access technology, FireDAC.
2. Get rid of as many 3rd-party components as possible. (all the visual components, while keeping modern versions of non-visual components like those for interfacing OPC Servers)
3. Move to Delphi 10 Seattle and Windows 10.
4. Rearchitect and refactor shared components and units in order to detangle some of the cross dependencies.
5. Begin the process of creating a proper business layer separation by moving most, if not all, of the business rules out of the UI and into data modules. This can be further improved by creating data model units, and possibly later the integration of a full ORM.

6. The final object will be for the main EnterpriseIQ application to be replaced by EnterpriseIQ Web. Other supporting applications will still be desktop applications (like applications monitoring the floor plant hardware for updating the database).

Suggested steps to achieve our goal

We already executed several steps in our path to achieve the desired migration.

First Steps

1. Compare several technologies for taking EnterpriseIQ to the Web.
    a. Java & JavaScript
    b. Delphi plus IntraWeb
    c. Delphi plus IntraWeb and CGDevTools
    d. Delphi plus UniGUI
2. Evaluate the technologies by creating proof-of-concept applications
    a. Java & JavaScript took a very long time and resources
    b. IntraWeb, with or without CGDevTools was a failure
    c. UniGUI proved to be a solution
3. Evaluate how to migrate to FireDAC
    a. The original tool provided by Embarcadero was modified for automating the migration from BDE.
    b. dbExpress was excluded because it is still supported by Embarcadero, but it should also be migrated to simplify our data access handling through all the application.
    c. As every module should be migrated to FireDAC, no further analysis was done about how to execute the task. The lack of documentation regarding the interdependencies between modules are currently causing delays to this project.
4. Execute the migration to FireDAC
    a. Every developer received a list of modules to migrate (in no specific order, just alphabetically).
    b. The estimated completion time was too short because several developers were engaged in other high priority tasks and proper upfront planning was not performed.
    c. Testing our changes is still an ongoing task. Each module should take at least 3 iterations before closing its issue.

Moving to new platform and development environment

As part of our migration process, but trying to avoid any interruption in our work, we should start working with Delphi 10 Seattle on Windows 10 (Professional or Enterprise).

Setting up our development PCs takes a couple of days, and we cannot afford to risk the stability of our current development environment (Delphi XE5 on Windows 7). Due to this situation, I describe below the more quick way to migrate our development platforms. Below this section I included change recommendations to our current system in order to minimize risk for maintaining development platforms.

Based on our current system the quickest and most convenient way to perform this upgrade would be as follows:

1. Get new licenses for upgrading from Delphi XE5 to Delphi 10 Seattle

2. Get new laptops with licenses for Windows 10
3. Update our Installation Guide for the new environment
    a. We were already looking for Delphi 10 support from our 3rd-party component providers
    b. We will need to update anything related to other basic tools like Oracle 12c (probably already done by IT)
    c. Someone could start doing it before everyone else receive the new laptops, trying to make this step even easier
4. After receiving the new laptops, licenses, and updated Installation Guide, set up the new laptop while still using our current laptop, and check that everything works as expected
5. Finally, the old laptop should be returned and everyone will be working with the new Delphi and Windows 10. If something fails, it should be solved before returning the old equipment.

The following are recommendations for the long term. Ideally we would change how we maintain development platforms as follows:

1. Centralize the installation location of all 3rd party tools we are using. The 3rd party tools should all be located within our version control system. For example ../p4/delphi/iqwin_2015/3rdparty/. By doing this we would then be able to version the 3rd party tools being used with a specific version of Enterprise IQ. Currently we are not tracking which version of the 3rd party tools were used with which build of Enterprise IQ.
2. Create a tool that allows us to build and install all 3rd party tools. This tool would be needed so that each time we change what release of Enterprise IQ we are working with, we can rebuild the 3rd party tools that correctly correspond with that product release.
3. Create a proper development computer image with necessary tools fully installed. This includes the OS, development environment, and all the development tools.
4. If we had the previous 3 steps completed the process of setting a new development machine would be to image the computer, check out the latest version from the source control, and run the custom 3rd party tool.

Next Steps

Learning from our current experience while migrating to FireDAC, we suggest the following steps:

1. Create a Stop Gap solution that provides immediate access to a web based Enterprise IQ.
    a. A successful stop gap will buy us time to properly execute a UniGui migration
    b. Following Farshad suggestion, we studied and tested Thinfinity VirtualUI and found that it could provide a quick-and-dirty single-line-of-code solution to customers not willing to wait for our migration.
    c. Even if this application is just some kind of improved Remote Desktop, its performance should be enough for some customers if the servers are correctly sized and we offer them load balancing and failover.
    d. It could be also good business… Buy this now, buy / upgrade to something better later.
2. Create a tool named Source Code Analyzer (SCA) with the following features:
    a. Use a Delphi Parser as its core engine. This will provide the means to extract all necessary information from our source code.
    b. Analyze our source code and create tree structures that demonstrate:
        i. Modules

        ii. Units in our modules
        iii. Classes in our units
        iv. Properties for classes
        v. Units using other units
        vi. Classes inheriting from other classes
    c. Provide reports about the dependencies
        i. Between units
        ii. Between classes, including class inheritance and visual form inheritance
    d. Apply Topological Sort to all of the dependency structures. This sort shall return migration order.
        i. This topological sort of the code base will allow us to quickly identify any critical paths for migration by identifing the following scenarious
            1. This sort will prioritize the list and show any modules that can be migrated without dependences on any other modules.
            2. Using the information already extracted from the source, it is now possible to select modules for migration that are dependant on the modules already migrated. We can also now migrate some modules in parallel which have their prerequisite dependencies migrated.
        ii. If possible, this result should be presented as a directed network graph, but it is also possible to export its information to Microsoft Project to create a Gantt chart which could be used later for tracking the migration process.
        iii. Detect cycles in the directed network graph. They must be resolved before creating the Gantt chart.
    e. Finally, this tool should provide a safe, automatic migration, of a TForm to a TUniForm.
        i. Before implementing this feature we will need to perform research and create a prototype as described later in this document.
3. Use the first results from SCA for planning the migration
    a. Find out all the components used in our TForm(s) and descendants
    b. Choose the right translation for each component
        i. A standard TUni<control>
        ii. Create a migrated version of all our IQVCL components (IQWebSearch from IQSearch)
        iii. Create a new IQWebVCL component inheriting from a standard TUni<control>
        iv. Create a new IQWebVCL component built as a TUniFrame (and converted to a UniGUI compatible control) with an interface identical to some components we use (like a TwwDBGrid)
    c. Implement the translation for each supported component
        i. We should keep all the original properties whenever possible
        ii. By using our own TUniForm child, we could automate some tasks like ShowModal by providing a common implementation using anonymous methods (and avoiding manual changes to the code)
4. After having the results of the previous analysis we should start implementing the components needed for the automatic migration

   a. Each component should be created first without any functionality, but with all the required published properties
   b. The migration tool will be able to work even without having all the finished components. This means we can test the migration on an incremental basis.
   c. The core components should be implemented in house, after defining each one of them according to the UniGUI components and the minimum requirements for the initial version of the User Interface for the Web.
5. Once the main components are ready, it is time for creating a prototype application to be used as best-practices, our pattern-to-follow for the new development. Everyone should be involved in shaping this prototype as it will be running in the "real" IQWeb application, using our core components, and showing the selected style.
6. After a successful prototype has been produced we can use the information provided by the SCA in order to assign migration tasks to all the developers. These migration tasks can consist of individual modules, since we should have any critical paths identified.
7. We can now use a project tracking tool. A general project tool, such as Microsoft Project, will allow us to follow the plan execution, detect risks to the critical path, reallocate resources, and collect statistics. A more specific project tool, that has focus on sprints and burn down charts, will not only provide us the metrics we gain from a Project Management tool, but we also gain specific developer metrics that provides a more transparent view of the project life cycle.

Gross guesstimate

Task	Completion Date
FireDAC migration	Mid-February (testing could increase this time)
Source Code Analyzer	Ongoing, first version without migration Mid-February
Component selection, techniques, first place-holder components, a couple of "pattern" components (visual and non-visual)	End of February
Core components development	End of March
Prototype application to use as a pattern-to-follow	End of March
Module migrations	Starting on April, using automated migration, expected duration depending on collected statistics about performance