

# Request for Expression of Interest

27 March 2009

Computing Integrity is currently evaluating technology that can be used to transform ABL source code and is looking for companies who have legacy code needing modernization or other improvement so that we can better understand the transformation requirements they face. The transformations we are considering vary from simple operations that improve code quality to substantial transformations required in modernizing an application. While some transformations need some manual aspect, e.g., the need for human design skills, the bulk of the work will be highly automated, ensuring great speed and accuracy. It is anticipated that we will achieve as much as 5 to 1 improvements in the cost to implement any given transformation relative to current methods. Please let us know your transformation needs so that we can insure that our tools address all requirements. The transformations we are currently evaluating are listed below.

## **User Interface Layer Separation and Replacement**

We propose to identify user interface (UI) code, transform components with UI into separate UI and business logic (BL) components with a well-defined interface, and move the user interface code into a new separate layer consistent with OpenEdge Reference Architecture (OERA) principles. Both client-server and N-tier models are possible and the process will allow for replacing the technology in the user interface with browser clients with Ajax, .NET clients, Java clients, ABL GUI for .NET clients, and probably even ChUI clients if there is demand. It is anticipated that the new UI will fulfill the same contract as the existing UI, but is likely to require human input on the final design for optimum results.

## **Data Access Layer Separation**

We propose to analyze the existing application to determine how and where data are used, to generate a set of data access (DA) components which implement those uses consistent with OERA principles, and then to transform existing direct references to data into references to the new DA components. This could be combined with DataServer enablement (see below).

## **Implementing Common Infrastructure Components**

OERA design principles include the use of common infrastructure components to provide services needed by many different components. In existing applications there are typically either prototypes of such services or in-line code and the implementation is limited compared with design goals for a new application. We propose to create high quality infrastructure components and provide transforms to replace existing component references or in-line code with references to the new components. Sample Common Infrastructure Components include Authentication, Authorization, Logging, and Session Management<sup>1</sup>.

## **Object-Oriented Repackaging**

While no simple transformation will convert a procedural ABL application into an exemplar of best object-oriented (OO) design, we propose to provide a series of incremental transformations which will move an existing procedural application in the direction of OO encapsulation by identifying behavior clusters, replacing these with classes, and then refactoring the application to change direct behavior references to references to the classes. We expect this to be an incremental process with the opportunity for human interaction, including the human creating new classes and then refactoring them into existing code.

## **DataServer Enablement**

While DataServers allow the use of ABL code with non-Progress databases, there are some ABL features which cannot be used with a DataServer and some best practices that need to be followed to insure behavior consistent with that in Progress databases. We propose to automate the identification and modification of legacy ABL data access code to create a DataServer enabled application.

---

<sup>1</sup> See <http://www.oehive.org/OERAOSI> for other components.

## Shared Variable Refactoring

We propose to analyze the application to determine which existing shared variables were used to establish context and which were used for passing values from one program to another. For the context set we would create appropriate context manager superprocedures or singleton classes and transform existing shared variable references into references to such components. For the shared variables being used like parameters, we would transform references into explicit inter-program parameters to make the interface and coupling clear.

## Code Quality Check and Repair

ProLint<sup>2</sup> provides the ABL community with a powerful and flexible tool for insuring conformance to standards and detecting bad practice or questionable constructs. However, it is most commonly used to evaluate individual new or modified code units because human effort is required to modify any problems that are detected. We propose to implement an automated process for detecting all problems of a particular type throughout the code base and then, following any needed exception handling markup, to provide transformations which will correct the code to the desired form.

## ProRefactor Transforms

There are existing transforms available through ProRefactor<sup>3</sup> which we would propose to replicate in the new tool in order to provide an integrated service. In some cases we might extend the functionality currently available. These transforms include:

1. Force table and field names in code to match those in the schema;
2. Force full qualification of all field names;
3. De-abbreviate all table and field names;
4. Rename Schema in database and code; and
5. Convert string concatenation to SUBSTITUTE() for translatability.

## Code Formatting

A number of shops have utilized programs like beauty.p to impose common stylistic standards on their code base, but these functions are typically limited in flexibility and are best applied on a program by program basis. We propose to create a highly configurable tool which can be applied to the code base as a whole, thus both assuring conformance to the standard and allowing for standards to change over time.

## Not Magic

While some of the transformations discussed above might sound like magic, and thus raise doubt that they are possible, the technology is neither new nor particularly unusual, simply new in its application to ABL development issues. This technology has already been used for more than 10 years on other languages for transformation and code quality and the tools are already parsing ABL code and doing some transformations similar to those discussed here. To guide further work developing this technology for ABL, we are looking for companies that are interested in one of more of the listed transformations or who might have some other transformation need that we can explore. We need your input to know where to concentrate effort. Let us hear from you!

---

<sup>2</sup> See <http://www.oehive.org/prolint>

<sup>3</sup> See <http://www.oehive.org/prorefactor>



# COMPUTING INTEGRITY

---

INCORPORATED

60 Belvedere Avenue  
Point Richmond, CA  
94801-4023

Voice: 510-233-5400  
E-Mail: [thomas@cintegrity.com](mailto:thomas@cintegrity.com)  
Web: <http://www.cintegrity.com>

**PROGRESS**  
Application Partner