

Automated document level XML-enabling of IBM COBOL systems

The challenge

The arrival of and the de facto standardization on XML for information exchange has created a modernization challenge for enterprise level application managers. While their existing COBOL systems' business logic implementations continue to serve their purpose and users well, more and more of these systems have to be upgraded so they can accept and produce XML documents. This raises many interesting questions. What is the best way to XML-enable these applications while ensuring that the existing business logic will not be compromised? Can this be achieved without having to re-engineer the whole application? Other issues and challenges include: time-to-market, cost of upgrade - including development and testing, risk management, long term maintenance costs, upgrading of skills and the availability of development resources. Choosing the right modernization approach to enable XML handling within your existing system can have significant short and long term benefits.

An application-centric architected approach

The "right" architected solution for XML-enabling existing applications would ideally protect and leverage the integrity of the existing business logic. Such a solution would result in reduced need for testing and reduced risk to existing business processes - two important considerations when modernizing applications. Using specialized sub-programs that translate XML to/from existing traditional data structures prior to or after execution of the existing business logic achieves these objectives while maintaining application level control on input and output data handling. In this way, existing business logic remains undisturbed eliminating the introduction of new logic errors during the modernization effort. Regression testing time and cost will be significantly reduced and time-to-market improved.

Document level XML-interface

Application centric XML-enabling approach isolates XML translation into specialized and well defined areas of the application. In the case of receiving XML documents, the implementation of this solution is accomplished via an XML handler subprogram (*XML reader*) that parses the content of the XML document into the existing traditional data structures before the business logic is invoked. Once the XML data is translated and made available in traditional data structures, existing business logic can process the data just as before.

The converse is true when an application has to produce XML documents. Having run data through existing business logic, an XML handler (*XML writer*) can be invoked. It will use the contents of the traditional data structures to create well-formed XML documents. In both cases the XML consuming or producing application is concerned with passing or receiving the whole XML document to the XML handler subprogram. This document level interface enables the isolation and separation of XML handling logic into the subprogram and allows the existing business logic to remain intact.

This isolation of XML translation logic also allows applications to utilize I/O in the most suitable way for the application. Because the main business logic program and the XML handler subprogram pass the XML in a buffer area, the application has complete control and flexibility of utilizing any input or output method it has access to. Whether the XML document is sent via MQ, CICS SOAP, or sequential file, this XML enabling approach works well and minimizes impact on the application.

Automated and repeatable XML-enabling solution

In essence, program code that translates data to/from XML provides utility functionality and has very little relation to the business logic those implements and automates business processes. While XML handler programs can be quite complex, their predictable and utilitarian nature allows for a high level of automation in creating them. How then do we avoid using highly skilled technical resources with expert vertical business knowledge from wasting their time implementing utility functionality? The answer to this very question is sought by many application systems managers.

The solution is an automated software development process that enables the design and generation of XML handlers based on specifications. The specifications must include the mapping of XML schema or DTD information to traditional data structures, and the capture of runtime processing and error handling options. Once the XML handler specifications are developed, they provide an abstract representation of the XML handlers that will parse or generate XML documents at runtime. This abstract representation serves as a well-defined specification for the XML handlers and thus automatic code generation can be employed to produce the desired XML handler program code.

XML Thunder, a bi-directional XML-COBOL development tool

Canam Software's implementation of this concept is *XML Thunder*, consisting of an easy to use visual designer and code generator. The workstation toolset enables the fast and easy visual specification of the mapping between XML and COBOL or C data structures, including repeating structures. Additional features, like optionality, white space handling, edit pattern definition, extensibility and mutual exclusivity can also be set during visual design.

Once the design of the handler is complete, the appropriate handler type, XML “reader” or “writer”, can be selected and the application source code that implements the specifications set during design will be automatically generated. The XML document and rules may be complex, but the solution is simple, automated, repeatable, and fast. This allows for a high degree of productivity by developers who are not experts at using XML.

XML Thunder generated program code supports the IBM Enterprise COBOL “XML PARSE” statement or, as an alternate, it can generate its own parsing program logic in native COBOL.

Benefits of using XML Thunder for XML-enabling applications

The benefits of using the visual designer and automatic code generation approach can be summed up as follows:

- Increased productivity
- Lower development, training and maintenance costs
- Improved quality
- Abstraction
- Consistency

Let’s examine each of these benefits individually.

Increased productivity

Utilizing a visual designer and an automatic program code generator results in significantly improved productivity for both development and maintenance. Visual design has proven to be much faster, yielding significantly faster time-to-market results than manual development. Once the abstract design is developed and the implementing program code is generated, the job at hand collapses to merely calling the generated XML handler as a regular subprogram and checking and handling any error codes returned after the call. No XML logic programming, no XML event handling, no formatting of data! Just high speed use of a high ROI utility! The productivity gains of code generation lie in the agility of the code generators to rebuild the code base to suit the changing requirements (specifications) of the project. With one generation iteration you can reliably add or remove large segments of code as required.

Lower cost of development

Higher productivity means lower development costs. The massive amount of industry proven code provided by the inherent code generation pattern, coupled with the complete job done by the generator tool, shrinks development time and cost to an unprecedented fraction of what you would otherwise expect. Time is not spent on trying to figure out how to handle XML specific data transformation program logic, but instead can be used to address business logic issues. This results in a significantly lower cost for solving XML handling requirements.

Lower cost of maintenance

While the generated program code should be readable and maintainable by hand, as is the case with XML Thunder, the source of the XML handler should remain the abstract level XML handler definition. This definition was originally created using XML Thunder and should be maintained by utilizing the XML Thunder toolset during the system maintenance lifecycle in order to continue to derive the productivity and other benefits of this approach. In fact, one may make the argument that the overall payback may even be larger during maintenance than during development, as the maintenance phase of an application is typically much longer than the development phase. The same high ROI enjoyed during the development phase carries over into ongoing maintenance!

Quality improvement

Code generators use patterns to automatically build the specified code, in our case, the abstract XML to COBOL/C mapping. The better the design pattern, the better the generated program code. As the vendor continuously improves the internal design patterns, the quality of the overall generated code base is iteratively improved. Because code generator vendors make it their business to address detailed design issues for specific application solutions they generate program code for, they become experts in the subject areas of their tools. In addition, code generator vendors benefit from ongoing feedback from a wide and varied client audience, thus improving their product over time. This results in the implementation of industry best practices for given tasks.

Abstraction

Abstract representation of XML handler specifications allows language independent representation of XML handling requirements. This allows developers to address XML processing issues at a high level without having to worry about implementation programming details. It also enables novice, or even experienced developers new to XML, to be productive immediately without going through a steep and potentially time consuming learning curve of not only learning XML but also learning the limitations and specific peculiarities of implementing XML handling options in their programming language. (Note: this alternative learning curve would most likely translate into higher cost during development, a lower quality deliverable, and higher cost during maintenance and slower response to changing user or business requirements).

Consistency

Automatically generating program code from abstract specifications and patterns results in the consistent implementation of application requirements. For example, calls to utility subprograms and error handling are consistent across all application programs no matter who developed the business logic. This consistency enables a more disciplined architected systems development approach, more flexible deployment of development resources, and reduced maintenance costs.

Summary

XML is now in the mainstream and is here to stay. Our challenge, as an industry, is to provide our organizations and clients with the benefits of XML while continuing to leverage existing and proven application systems assets. These applications are the

workhorses of businesses and our business leaders and executives are typically interested in the introduction of new technology only to further their business goals. Risk management and risk reduction are also business key objectives. XML has been proven to add value to business processes. Organizations can reduce their risk and their costs by utilizing XML Thunder when XML-enabling their applications.

For further information please visit <http://www.xmlthunder.com> or write to info@canamsoftware.com.