10th ICALEPCS Int. Conf. on Accelerator & Large Expt. Physics Control Systems. Geneva, 10 - 14 Oct 2005, PO2.094-5 (2005)

USE OF XML TECHNOLOGIES FOR DATA-DRIVEN ACCELERATOR CONTROLS

Michel Arruat, Stephen Jackson, Jean-Luc Nougaret, Maciej Peryt

Accelerators and Beams Department,

CERN, Geneva Switzerland

ABSTRACT

During the course of the next 2 years, many software systems will need to be developed for the control and instrumentation of the LHC injector chain. These systems will be based on a standardised formal model and represented by several XML documents. Developers will require tools to manage these XML documents and applications that allow them to define the structure and behaviour of each of their systems. In addition, tools will be required for testing and diagnostics of the software prior to commissioning in the machine. This paper describes how XML technologies have been extensively used to produce generic tools capable of fulfilling developers' needs while ensuring that all systems adhere to the underlying model. In addition, we will describe how XML and C++ templates have been used to develop high performance, data-driven real-time code that is capable of fulfilling the real-time software needs of the LHC.

INTRODUCTION

During the past decade, there have been several attempts to create a software framework capable of capturing the recurring software elements found in the real-time software systems used extensively in CERN's accelerator complex. Before the convergence between CERN's two accelerator divisions (PS and SL) in 2002, two totally independent frameworks had appeared, both attempting to formalise how a developer declares these real-time systems' data (internal variables and properties accessible over the controls middleware) as well as how the systems should react to the machine timing. In the late eighties, the PS division developed the Generic Module framework, which provided the basic building blocks for developers to declare their properties, define the values of initialisation data used for real-time task variables, as well as a set of well defined libraries and workflows developers should follow to synchronise their software to the accelerator. The PS system was heavily focused around a central Oracle database, so not surprisingly the graphical user interfaces were developed using Oracle Forms. In the early nineties, the SL division developed BISCoTO [1] which aimed to provide a similar framework, but focused more on delivering a product which automated as many parts of the final system as possible, giving the developer a fully operational real-time system instantly. A BISCoTO developer needed only to provide the definitions of internal data-structures, along with parameters to drive the built-in scheduler to have a system running in a matter of minutes. Unlike the PS system, BISCoTO was based on flat text files and the graphical user interfaces were in-house applications written in Swing (Java).

FRAMEWORK HARMONISATION

In 2002, a reorganisation within CERN meant that the PS and SL divisions were merged to form the new AB (Accelerators and Beams) department. In this new department, it was clear that the two existing frameworks would need to be harmonised, and a new framework should be created ready for the LHC era. The FEComSA [2] (Front-End Computer Software Architecture) project aimed to do exactly this, and in 2003 delivered a fully functioning framework, capable of operating in all of CERN's accelerators. One of the FEComSA project's main objectives was to prove that an AB-wide framework was feasible without focussing on the intricate details of integrating with existing infrastructure. It was therefore decided, not to tackle the issue of integrating the FEComSA framework into the complex PS Oracle database, so like BISCoTO, flat text files where used to store system designs and configuration data. Many of the Java based graphical user interfaces from BISCoTO were *adapted* to be compatible with the new framework, with an increasing focus on the use of XML and XSL [3] technologies during the automatic code-generation phase.

The FESA (Front End Software Architecture) project was launched in 2004 to complete the harmonisation, and was charged with producing a framework capable of operating seamlessly in all CERN's accelerators. The recent overhaul of the PS Oracle database meant that it was a perfect time to integrate the development framework directly into the new DBMS.

CAPITALISING ON PRIOR EXPERIENCE FOR FESA

The expectations of the FESA framework were high. Not only did it have to provide the functionality of its predecessors, but it also had to have the flexibility to adapt to the needs and constraints set by the new DBMS system and new controls infrastructure. The choice of technologies for both the framework's code-generation as well as the graphical user interfaces used by the developers had to be made carefully so as to allow these parts of the framework to mutate with ever changing requirements. In addition, the scope of the FESA project had broadened to include other types of systems such as PLC based systems with special configuration requirements and no real-time constraints as such. To make matters worse, many of these new requirements had no real *specifications* meaning that the framework would have to pass through many iterations of rapid prototyping before it *got it right*. Clearly, the work required to change the code-generation components and the graphical user interfaces to a continuously changing set of constraints and requirements would have to be kept to a minimum. It was therefore decided to assess the positive and negative aspects of the previous

frameworks before making an informed decision on how to provide the next generation of framework tools.

Technologies for code generation

	Basic facilities for loops and conditionals	Automatically handles all framework's data-types	Easy to read and produce	Industry standard	Comments
In-house macro based templates	×	×	✓	×	Only really maintainable by the developer who wrote it!
XSL templates	✓	×	×	✓	Have to handle data-types manually
C++ templates	×	✓	✓	✓	Only provide support for handling differing data types
XSL driven C++ templates	✓	✓	×	✓	Very difficult to read and produce but very powerful in the end!

Figure 1: Code generation requirements of the FESA framework versus technologies available

The use of a bespoke macro based solution for code-generation was successful in both the BISCoTO and Generic Module frameworks although it suffered 2 major drawbacks. As with any bespoke solution, the main disadvantage of a non-industry standard solution is that any new requirements must be implemented in-house. In the case of the earlier frameworks, simple substitution macros (substituting a tag with values in the developer's design) were easy to implement, but more complex constructs such as loops, conditional statements or methods are much more tricky to implement. The 2nd and more obvious disadvantage of an in-house solution is that the knowledge of how it all works is held by 1 or 2 people, meaning that nobody else in the department has the ability to troubleshoot when those people aren't available. This problem is confounded as the in-house solution becomes more complex to incorporate more complex macros.

XML and XSL technologies are relatively new, but are already proving to be the way forward in industry. In order to use them in the FEComSA project, it was necessary to encode all design information in an XML document, and then pass these documents through several XSL templates to produce the data-driven code for the framework. The rich XSLT language allowed the development of much more complex framework code templates, incorporating loops, conditional statements, re-usable *methods*, etc. What was lacking however was the ability to handle generic framework code for each of the framework's primitive types (int, short, float, etc). The FEComSA templates therefore became cluttered with repeated code, making extensive use of choose/when statements (similar to a switch statement in C) to handle the different data types.

The positive experience gained from FEComSA, meant that XML and XSL templates were considered perfect for the FESA framework. To overcome the problem of repeated code for different primitive types in the framework, it was decided to combine the power of XSL templates with the generic code facilities of C++ templates. What resulted was a set of intelligent, data-driven, object oriented, industry standard templates. These templates have been continuously modified during the evolution of the FESA framework and have proved perfect for handling the ever-changing framework code.

Graphical User Interfaces for framework users

	Compatible with new AB's Oracle based controls database?	Interaction with a running server	Handles XML and XSLT?	Industry standard	Comments
Oracle Forms	✓	×	×	✓	Problems incorporating code generation and testing over middleware
Java Swing with flat files	×	✓	✓	×	Not compatible with the AB Oracle-based control system
Java Swing with JDBC	✓	✓	✓	✓	Encapsulating SQL statements into Java code becomes very un-maintainable
Java Swing with XML over JDBC	✓	✓	✓	✓	Up to the database to *shred* XML to extract data into underlying tables

Figure 2: GUI requirements of the FESA framework versus technologies available

As with code-generation, the FESA framework had several options to choose from when deciding how to tackle the development GUIs. Given the 2 main compatibility constraints, XML and Oracle, it was clear that the use of Oracle Forms (as used in the Generic Module framework) or a Java application based on flat text files (as used in BISCoTO and FEComSA) would not be suitable. This left the option to either chose Java applications with embedded SQL or Java applications which dealt only with XML (passing the XML document directly to Oracle for *shredding*). In effect, the choice was between fat client applications (incorporating the necessary logic to send the differing parts of an equipment's design to different database tables) or thin client applications (knowing nothing about the underlying data structures, but acting as a basic XML editor). Given that the framework's internal data structures and corresponding database tables were likely to change a lot over time, it was decided to choose the second option and try to base all framework GUIs on a basic XML editor with no knowledge of the underlying framework or databases.

GUIS FOR EACH PHASE OF THE DEVELOPMENT WORKFLOW

Developing a FESA class, consists of 4 main development phases:

- Designing the class structures (internal variables, real-time scheduling, external API, etc)
- Deploying the class on a front-end computer
- Instantiating 1 or more instances of a deployed class (defining configuration values for internal variables, real-time scheduling, etc)
- Testing over the accelerator middleware

To assist the developer through the framework's workflow, it was decided to provide a *shell* application that incorporated individual applications for each phase. As previously mentioned, the goal was to transport all framework definitions and data in XML, creating thin GUI's for each of the 4

development phases. To make this possible, a generic XML editor was created in Swing (Java), configurable by means of a series of W3C compliant Schema [4] documents. This XML editor component forms the backbone in the design, deployment and instantiation applications and all *specifics* for these applications are defined in their corresponding Schemas. This effectively means that very little maintenance of the applications is required. Instead, the schemas that drive the generic editor are edited to reflect changes in the framework and the underlying Java code remains untouched.

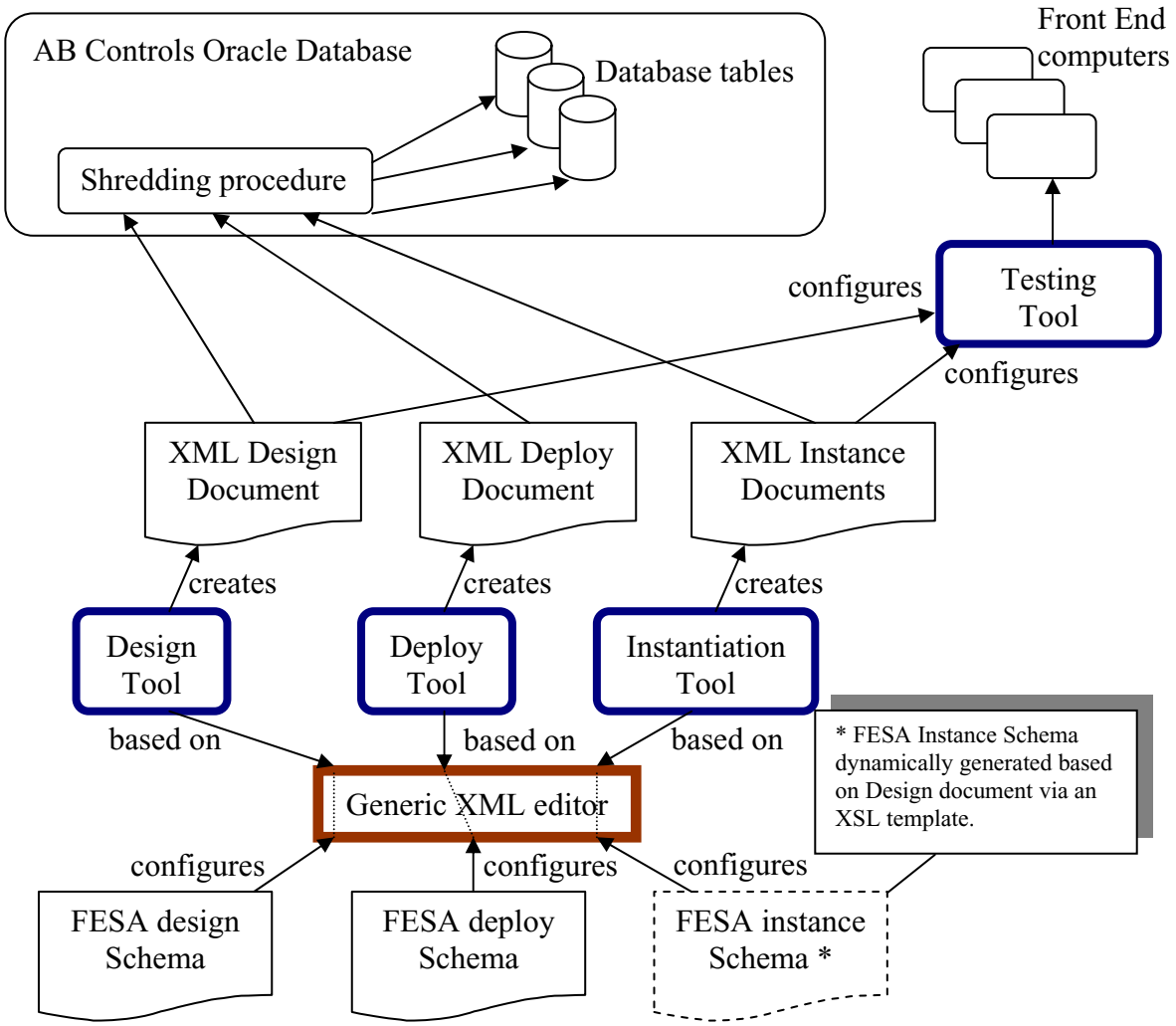

Figure 3: Relationship between GUIs, the generic XML editor, the database and the front-end computers

The final application, the testing tool, is the only one that doesn't create an XML document. It therefore doesn't incorporate the generic XML editor, but is driven by the design document (to find out the details of the properties in the instrument's API) and the instance documents (to discover the names of instances to test). The concept of a *thin client* is also employed in this application as it uses XSL templates to convert the design and instance documents into Java property files (the standard way to configure Java applications). Again, this means that when the structure of the framework's design and instance documents changes, it is often sufficient to only change the XSL templates to produce Java property files understood by the testing application. As with the first 3 applications, this means that the underlying Java code needs to be rarely touched for maintenance.

SUMMARY

The FESA framework was charged with consolidating all the requirements of its predecessors as well provide a transparent *cross-accelerator* framework ready for the LHC era. This has meant that the framework has passed through many prototyping phases, implying that many aspects including the fundamental structure of the framework have changed many times. To support such a volatile environment, the technologies chosen for code-generation and development GUIs had to be made with care. Schema controlled XML documents combined with XSL and C++ templates have provided a basis to produce truly generic framework code-generation components and GUIs that are maintainable by anybody familiar with the Schema/XSL notation. In terms of maintenance, this means that it is rarely necessary to change underlying Java code in the end-user applications, and the code-generation parts of the framework are maintainable by a larger subset of developers.

Transferring all database logic to database stored procedures has lead to ultra-thin GUIs. This allows the database to mutate and change its structure at-will, so long as it is still able to accept the well-defined XML documents from the framework applications. Insulating the GUIs from the database creates a clean separation, whilst the intermediate XML Schema (which is understood by both the database and the generic XML editor) forms the *contract* between the two framework components.

The FESA framework will continue to grow and change as more requirements for the LHC injection chain become apparent. With the outlined structure in place for the framework tools, these changes should be relatively painless for the framework developers, and more importantly, almost transparent to the users of the framework.

REFERENCES

[1] ICALEPCS 2003 - Common Template And Organisation for CERN Beam Instrumentation Front End Software Upgrade (BISCoTO)
http://icalepcs2003.postech.ac.kr/db/proc_papers/MP563/MP563.pdf
[2] ICALEPCS 2003 - CERN Front-End Software Architecture for Accelerator Controls (FEComSA)
http://icalepcs2003.postech.ac.kr/Proceedings/PAPERS/WE612.PDF
[3] XSL specification
http://www.w3.org/Style/XSL/
[4] W3C Schema specification
http://www.w3.org/XML/Schema
[5] ICALEPCS 2005 – Equipment Software Modeling for Accelerator Controls