XML CONSTRUCTS FOR DEVELOPING DYNAMIC APPLICATIONS OR TOWARDS A UNIVERSAL REPRESENTATION OF PARTICLE ACCELERATORS IN XML

J. Chrin, R. Krempaska, H. Lutz, G. Prekas, Paul Scherrer Institut, Villigen, Switzerland
T. Pelaia, Oak Ridge National Laboratory, Tennessee, USA

Abstract

A recognized practice in the development of high-level beam dynamics applications is to separate data parameters destined for the configuration of the application from the programming language domain. The contemporary approach is to generate input files that provide the configuration parameters in a structured data format specified by the Extensible Markup Language (XML), enhancing flexibility and simplifying code maintenance. Furthermore, a careful consideration to the form of XML syntactic constructs i.e. structured elements, attributes, etc., that map well to the various accelerator components, provides a basis for portability of configuration classes and high-level applications. This has been exemplified by the XAL application software package which initiated an XML description of the Standard Machine Format (SMF) accelerator object model. We have since adopted and optimized XML-SMF to provide an XML representation of both the Swiss Light Source (SLS) and the SwissFEL 250 MeV Injector Test Facility. We demonstrate how a common set of XML constructs allows us to deploy the same, example orbit display application at both facilities. Our experience leads us to advocate a *Universal Machine Format (UMF)* that encompasses an all-inclusive XML vocabulary for the management of particle accelerator information pertaining to beam dynamics applications.

INTRODUCTION

Configuration files in beam dynamics applications typically provide a number of properties that the application may depend on. By keeping these data parameters outside the domain of the programming language, applications may be adapted to changes in the accelerator system by simple modification to the configuration file. Such a dynamic approach is particularly applicable in the development phase of an accelerator, where operation is interrupted at scheduled stages to add new accelerator components, and, more generically, in the configuration of any reusable stand-alone software component.

The customary approach, for many years, was to provide line-oriented flat configuration files that did not convey any structured relationship. Their interpretation, however, necessitated advanced knowledge of the data format, and required custom configuration data parsers to be written, often in various programming languages. The advent of the Extensible Markup Language (XML), however, offered a new mechanism for describing accelerator data in a hierarchial structure within a configuration file. Further supported by tools, such as XQuery and XPath for querying and traversing XML documents, and coupled with various categories of Application Programming Interfaces (APIs), e.g. stream-oriented, tree-traversal, and data binding, it is now the de facto standard for data exchange and portability. The contemporary approach is thus to provide configuration files that encompass data in a structured format specified by a markup language developed for XML. This was first embraced by the XAL high-level software package [1, 2] which defined the first XML markup language for machine applications, following a structure introduced by the Standard Machine Format (SMF) accelerator object model [3, 4]. The usefulness of XML has been likewise demonstrated in the field of accelerator design and simulations, where lattice files are formatted according to a dedicated vocabulary [5].

We have since adopted XML-SMF and optimized it for use at both the SLS and SwissFEL 250 MeV Injector Test Facility. We re-examine the use of flat configuration files at the SLS and draw on this experience to assess and anticipate requirements for applications at the SwissFEL and its injector test facility. An example beam orbit display application is presented to demonstrate how adherence to a considered set of XML constructs further enhances portability of applications.

XML CONFIGURATION FILES

The generic XML-SMF representation follows an `accelerator`, `sequence`, `node`, `properties/channelsuite` hierarchical schema, as illustrated in Fig. 1 (left). A `sequence` is a contiguous section of related beamline elements, while a `node` is a physical device, such as a magnet, diagnostic device, etc., located along the beamline at some given distance from a reference value. Most of the elements are annotated with attributes (name/value pairs) that provide metadata associated with the element. Attributes for the `node` element, for example, include its type, identifier, position and length. The `channelsuite` element is a container for `channel` child elements, whose attributes encode details of associated control system channels. The XML instance of the SwissFEL Test Injector is shown in Fig. 1 (right).

The accelerator file is generated automatically from different databases. The master "Device Reference" database,

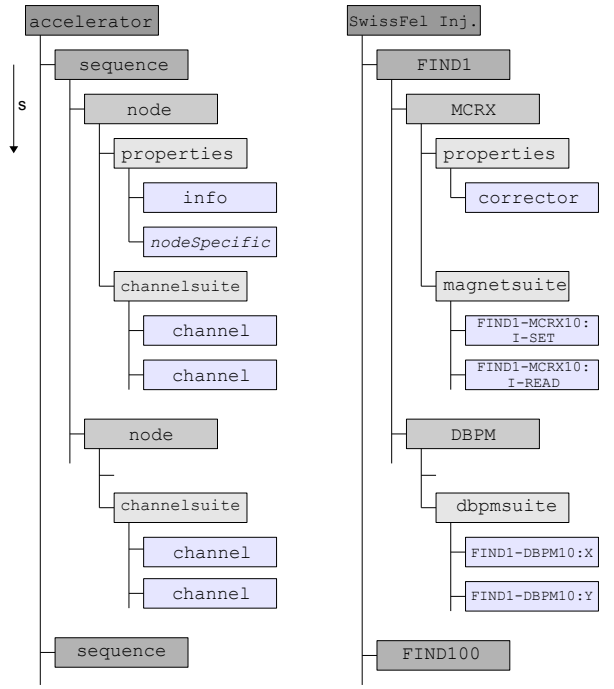


Figure 1: A hierarchical representation of a particle accelerator with associated XML elements (left), accompanied by a partial instance for the SwissFEL Test Injector (right).

for a given accelerator, contains a complete inventory of the hardware components, reflecting the topology of the accelerator. It does not, as such, map directly to the XML-SMF structure, nor does it contain information on associated control system parameters. Information on the latter resides in a separate database. The adoption of the "device", "properties" paradigm for naming the EPICS-based control system channels, however, enables hardware nodes, i.e. "devices", listed in the "Device Reference" database, to be associated with their respective suite of control channels. A snippet of the generated SwissFEL Test Injector XML accelerator file is listed in Table 1.

The XML accelerator file is intended as the principal configuration file for machine applications. While the necessary information required to produce an inclusive XML file for applications is yet to be fully realized, already the incorporation of data for a modest set of nodes, specifically magnets, radio-frequency cavities, beam position monitors, screens and cameras, is sufficient to provide for a broad spectrum of applications. To simplify integration of data from the XML file into our mainly MATLAB and Qt application environments, helper classes that utilize the event-driven SAX API have been provided. Ultimately our aim is to expand their functionality and house them in a separate parser library.

A further usage of the XML accelerator file is as a data source for the extraction of related data sets, i.e. nodes of the same type, to form device collections that may be referenced in a control systems interface as a single logical software entity. In the example to follow, interaction with

Table 1: An XML View of the SwissFEL Test Injector

```
<xdxf system="SwissFEL" ver="R30">
  <sequence id="FIND1">
    <node type="MCRX" id="FIND1-MCRX10"
          pos="0.166" len= "0.005">
      <properties> ... </properties>
      <channelsuite name="magnetsuite">
        <channel handle="I-SET"
                 signal="FIND1-MCRX10:I-SET"
                 settable="true" />
        <channel handle="I-READ"
                 signal="FIND1-MCRX10:I-READ"
                 settable="false" />
      </channelsuite>
    </node>
    <node type="DBPM" id="FIND1-DBPM10"
          pos="0.437" len= "0.073">
      <properties> ... </properties>
      <channelsuite name="dbpmsuite">
        <channel handle="X"
                 signal="FIND1-DBPM10:X"
                 settable="false" />
        <channel handle="Y"
                 signal="FIND1-DBPM10:Y"
                 settable="false" />
      </channelsuite>
    </node>
  </sequence>
</xdf>
```

the EPICS-based controls system is accomplished through CAFE [6, 7], an in-house C++ library that provides a multifaceted interface to the standard channel access client library. The API includes methods that can act on a group of channels, aggregated from pre-defined device collections and/or from individual channels; such interfaces allow several requests to be delivered efficiently with a single method invocation. Since CAFE provides its own XML schema for defining collections, a dedicated parser, constructed from the QtXMLPatterns library, traverses the accelerator XML file to pluck out information from nodes of a given type, and transforms the selected data into the CAFE collection XML schema. The parser is executed from the command line with output piped into an XML file that defines the collections and their members:

xmlfilter $Node_1$ $Node_2$ $Node_n$ > CAFECollections.xml .

The extraction of data from the accelerator file to automatically produce CAFE collection XML files alleviates the need for their manual instrumentation and renders a separate schema validation mechanism less important. The pre-defined collections are loaded from the XML file on initializing CAFE and may be addressed by interfaces through their identifier.

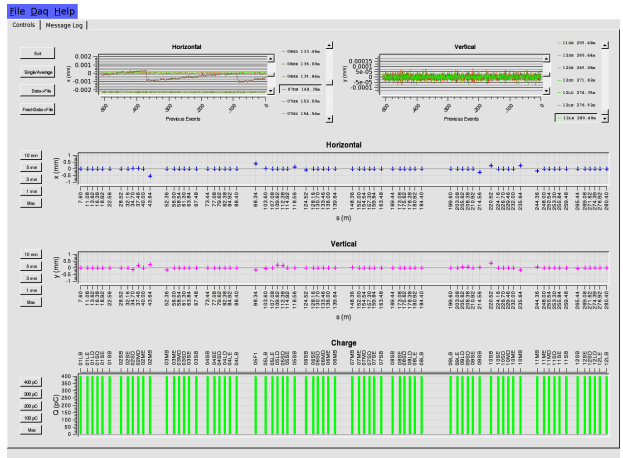
A DYNAMIC MACHINE APPLICATION

An orbit display application is presented to demonstrate how use of the XML constructs allows us to provide the same functionality on different accelerators by implementing identical code. In this example, the application separates the actual data processing from its display. An Event Processing Agent (EPA) is assigned the task of aggregating, verifying, and analyzing the low-level Digital Beam Position Monitors (DBPM) data, and distributes summarized results to interested clients through a publish and subscribe mechanism defined by the Data Distribution Service (DDS). A record of recent event history is also captured in shared memory managed by DDS. Averages and deviations of beam positions and trajectories over thousands of events may be calculated, and results written to files for off-line diagnostic analysis. Further details of the event driven procedure appear elsewhere [6].

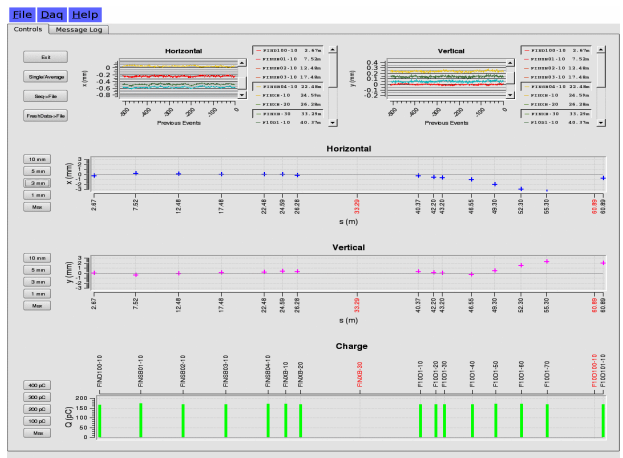
The deployment of these EPAs on different accelerators is made possible by adopting a policy to separate configuration parameters, such as static accelerator data and control system signals, from the rest of the application code. Data related to the node type, in this case the number, the names, the positions of the DBPMs, are obtained from the master accelerator XML file. Control system parameters are read from a CAFE configuration XML file which defines a group of DBPM channels by reference to their device collection identifier and attributes, enabling a callback mechanism to be easily established through a tailored interface. The application is thus fully configurable through XML and can be deployed on both the SLS and SwissFEL accelerators by hosting the relevant software packages and corresponding configuration files, as demonstrated in Fig. 2.

A UNIVERSAL MACHINE FORMAT

The adoption of XML-SMF has allowed us to write applications that can easily adapt to changes in accelerator topology, an important consideration during the development phase of a facility such as the SwissFEL, and be equally applied to different accelerators at PSI. Our experience further leads us to recognize the benefits of an all-inclusive XML markup language to advance portability of certain standard beam dynamics applications. (For instance, additional `channelsuite` child elements could cater for local variations in definitions of a channel's state; node types may be mapped to customized classes.) This may be realized if communities of interest conscribe to using the same XML constructs. Such an XML vocabulary can be shared via a repository containing the appropriate elements of metadata, facilitating disclosure and usage. OpenXAL [8], which extends XML-SMF to other facilities, is a natural home base for discussion, and finalizing on a *Universal Machine Format (UMF)*. Adapting standard machine applications to a particular accelerator may ultimately be reduced to selecting the required software packages and stacking configuration files with the adopted UMF constructs.



(a) SLS Storage Ring



(b) SwissFEL Test Injector Facility (Phase 2)

Figure 2: The orbit displays of (a) and (b) stem from two implementations of identical code, configured dynamically through their corresponding XML input files.

REFERENCES

[1] J. Galambos *et al.*, "XAL Application Programming Framework", ICALEPCS 2003, Gyeongju, Korea, pp. 332-336.

[2] A. Shishlo *et al.*, "The XAL Infrastructure for High Level Control Room Applications", ICAP 2009, San Francisco, California, USA, pp. 131-136.

[3] N. Malitsky *et al.*, "A Proposed Flat Yet Hierarchical Accelerator Lattice Object Model", Part. Accel. 55:313-327, 1996.

[4] N. Malitsky, T. Satogata, R. Talman, "Configurable UAL-Based Modeling Engine for Online Accelerator Studies", PAC 2003, Portland, Oregon, USA, pp. 3482-3484.

[5] D. Sagan *et al.*, "The Accelerator Markup Language and the Universal Accelerator Parser", EPAC 2006, Edinburgh, UK, pp. 2278-2280.

[6] J. Chrin, G. Prekas, "A Taste of CAFE", ICALEPCS 2009, Kobe, Japan, pp. 821-823.

[7] J. Chrin, M.C. Sloan, "CAFE, A Modern C++ Interface to the EPICS Channel Access Library", to be presented at ICALEPCS 2011, Grenoble, France, 10-14 Oct. 2011.

[8] OpenXAL, `http://xaldev.sourceforge.net`.