ACCELERATOR DESCRIPTION EXCHANGE FORMAT

Nikolay Malitsky and Richard Talman
Newman Laboratory, Cornell University, Ithaca, NY 14853

Abstract

The Accelerator Description eXchange Format (ADXF) is in response to the Iselin-Keil-Talman letter[1], which calls for a new accelerator description standard aimed to provide connectivity between a variety of beam-dynamics programs and heterogeneous data sources. ADXF represents a flat and complete description of the current accelerator state. It has been designed as the additional independent fully-instantiated layer to existing design data structures, in particular the Standard Input Format (SIF). Though the ADXF preserves all SIF element types its open model provides a mechanism for introducing elements with arbitrary attributes. The proposed specification is based on the Extensible Markup Language (XML), an industrial standard for processing Web documents and application-neutral data.

1 PREFACE

Basing vertical domain applications on top of framework's reusable components and universal infrastructure is a reliable and effective approach in software development. However, a variety of different views and implementations of accelerator models prevent the direct integration of diverse accelerator programs and comparison of their output results. There is an actual need for a common format to exchange accelerator data between heterogeneous tools and data stores. In 1985 D.Carey and C.Iselin defined a Standard Input Format (SIF[2]) which significantly facilitated the combined usage and developement of several accelerator programs. The SIF implementation has been based on the MAD parser and has provided a description of the design model. Changes in computer and accelerator technologies determine new requirements (and solutions) for exchange mechanisms. In January 1998, C.Iselin, E.Keil, and R.Talman issued a letter[1], which called for a new Accelerator Description Standard(ADS) aimed to provide connectivity between a variety of beam dynamics programs and heterogeneous sources. The proposed Accelerator Description eXchange Format is a response to this letter. ADXF represent a flat, complete and independent description of the current accelerator state. The concepts described in this proposal are based on the Standard Input Format (SIF[2])), Standard Machine Format (SMF[3]), Standard eXchange Format (SXF[4])) and experience with actual accelerator applications, such as LHC[5], RHIC[6], CESR[7], FNAL Main Injector, and others.

2 RESPONSE TO ADS REQUIREMENTS

This section presents a list of the Iselin-Keil-Talman ADS requirements and their implementation in the ADXF specification.

2.1 Backward compatibility with the Standard Input Format

"It should generalize (but not replace) SIF in ways that experience has dictated appropriate."
"ADS should mimic SIF where possible, retaining basic accelerator objects and their attributes"

The Standard Input Format focuses on the *design* accelerator description and defines two of its components: *element* and *beam line*. ADXF offers an additional independent layer that deals with the *operational* fully-instantiated accelerator representation. The relationship between SIF and ADXF structures is provided by the references embedded in the ADXF objects. The ADXF format is based on the integration of the MAD *sequence*[2] and SMF *element buckets*[3]. The specification preserves all SIF element types and provides the mechanism for introducing new ones. SIF element attributes have been normalized and grouped in "orthogonal" buckets. The total list of ADXF element types and attributes are presented in Table 1 and Table 2.

"Other feature that have been suggested include: ... nested line preservation from underlying SIF design."

ADXF represents the accelerator by the flattened tree of accelerator nodes, elements and sequences. Sequences can be nested to an arbitrary depth and may have references to the corresponding design beam lines.

"Minimal completeness. All elements that can influence single particle motion (in their idealized operation) and only those elements are contained."

The proposed submission contains only SIF elements (see Table 1). However, its open model allows description of arbitrary element types (e.g. muon collider ionization cooling). This ADXF approach is determined by the actual accelerator problems, such as CESR superimposed solenoid and quadrupole elements, LHC and CESR parasitic beam-beam effects, and others.

2.2 Extensions of the Standard Input Format.

"Flexibility. Examples are: freedom (but not encouragement) to introduce additional elements or additional attributes to existing elements in a standard (for other purposes ignorable) way."

Extensibility has been the main criterion of the ADXF design and its implementation approach. The proposed specification allows accelerator description to be extended in two ways: introduction of new element types (e.g. a RHIC helical dipole or a CESR wiggler) and inclusion of new element buckets common for all element types.

"Full-instantiation. Every ring element has its own parameters and may have its own name (laboratory-wide, for example)."

It is provided.

" ... support for shared lines (such as two rings) "

Each SXF accelerator node, sequence or element, has a unique identifier within the particular accelerator name space, but it may be shared unambiguously by different accelerators (e.g. in interaction regions, injection and extraction systems, and others)

"... provision for definition of "families" of elements"

These families have been initially introduced in the Framework of Unified Accelerator Libraries as communication objects, adjusters and detectors, transported between the accelerator model and correction algorithms. Their specification will be defined after a working prototype of this system has been developed and alternative approaches analysed.

2.3 Consistency across different accelerator phase models.

"It should serve from the early phases of conceptual design, through the engineering design and analysis, to the operation of the accelerator."

The present ADXF version includes only the flat *operational* view of the accelerator; however it provides a mechanism for its integration with the existing site-specific design hierarchical models. This approach stems from the desire to make this format neutral to different conceptual models and adaptable to arbitrary data stores. ADXF can be considered as an additional independent layer to existing design data structures. The interface between the ADXF flat accelerator description and local design models is site-specific and therefore is not a part of this specification.

2.4 Separation of the accelerator physics algorithms and approaches.

"Containing only element and lattice description and no beam dynamics, it is to be usable without prejudice by any physical method."

It is provided.

2.5 Separation of technology issues.

"Multiple-realization, in forms optimized for efficient computation (independent of particular computer language), ease of modification, network transmission, database management, and human editing."

The ADXF conceptual model is described in the modern object-oriented Unified Modeling Language (UML) accepted as an industrial standard by the Object Management Group and other organizations. Expressing ADXF semantics in UML will guarantee multiple-realizations.

2.6 Compliance with computer standards.

"It should respect modern computer science standards, especially concerning database management and accessibility over networks."

The ADXF format is based on the Extensible Markup Language (XML[8]), an industrial standard for processing Web documents and application-neutral data. Its connectivity with disparate data stores (such as ODBMS, RDBMS, file systems, and others) can be implemented using the effective mapping techniques[9] or XML supporting tools.

3 ADXF OBJECT MODEL

An accelerator description is a complex system that combines elements with heterogeneous attributes in nested hierarchical structures. The complexity of this organization prompts a variety of different views and implementations of accelerator models. The ADXF is intended to provide a uniform format of accelerator description and interoperability across diverse accelerator applications. This goal determines the major design principles of the ADXF object model. It must be *flat*, *concrete*, and *extendible*.

In most accelerator programs, an accelerator element shares attributes with different elements. These attributes can be inherited from its design prototype and may be stored in global repositories. The choice of an optimal organization is a trade-off between many factors, such as application requirements, software environment, and others. To facilitate mapping diverse accelerator representations into the ADXF format, the ADXF element is described by the *flat* collection of *all* (inherited, global, and local) element attributes. The connectivity with application-specific hierarchical structures is provided by optional references embedded in the accelerator elements. It makes the ADXF

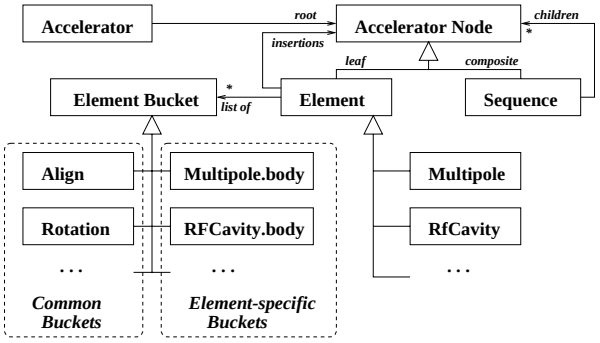

Figure 1: The ADXF object model.

format neutral to different conceptual models and adaptable to arbitrary data stores.

Despite a variety of different element types, accelerator elements have to be organized and processed polymorphically in uniform collections. There are two major approaches for modeling reusable interfaces: inheritance and prototyping. Inheritance is the most natural way of allowing users to change code in object-oriented programming. An architecture based on inheritance has well-defined organization and data types. However, inheritance results in *statically* binding related components. The prototype is a generic class that is able to *dynamically* produce objects with different structure and behavior according to their embedded metadata. This approach is based on the powerful *Reflection* pattern[10] and used in many operation systems and horizontal frameworks, for example, ActiveX Automation, CORBA Dynamic Invocation Interface (DII), and others. For the ADXF object model, an intermediate approach has been chosen. All accelerator element types have a uniform two-level structure: a *dynamic* open collection of *statically* defined buckets with orthogonal fine-grained sets of element attributes. Some buckets belong to particular element types, others can be included in all elements. This approach facilitates selection and classification of well-defined *concrete* data types. On the other hand, it provides a consistent *extendible* mechanism for describing and integrating new accelerator elements and effects. Fig. 1 shows the ADXF object model. Semantics, structure, and the XML declaration of its elements are described in the following sections.

3.1 Accelerator

The *accelerator* represents a current accelerator state. It is defined as the root in a hierarchical tree of accelerator nodes, elements and sequences of elements:

```
<!ELEMENT accelerator
    ( sequence | %adxf.element; )* >
```

The accelerator state has several attributes:

id	an accelerator name. The *id* identifies the accelerator instance and determines the naming context for all its accelerator objects.
version	a version of the current accelerator configuration.
timestamp	a date and a time of the current accelerator configuration.
design	a reference to the design prototype.

In the XML DTD format, these attributes are declared as follows:

```
<!ATTLIST accelerator
    id        CDATA #REQUIRED
    version   CDATA #REQUIRED
    timestamp CDATA #REQUIRED
    design    CDATA #IMPLIED>
```

Definition of accelerator parameters in the ADXF file is illustrated by the next example:

```
<accelerator
  id = "lhc"
  design = "lhc"
  version = "5.0" timestamp =
  "Mon, 18 Aug 1998 09:27:55">
...
</accelerator>
```

3.2 Accelerator Node

There are two types of accelerator nodes: sequence and element. They have three common attributes:

id	a name of the accelerator node.
design	a reference to the design prototype. Because the ADXF accelerator description is complete and independent from external sources this link does not affect node parameters and provides only the relationship with site-specific accelerator design models.
at	a longitudinal position of the node entry to the start of the parent sequence. If *at* is missing, the node is assumed to be butted up against the previous accelerator node.

According to the *Composite* design pattern[11], sequence and element have to be derived from the common class. Since the present XML specification[8] does not support any inheritance mechanism, these attributes have been combined in a sharable XML entity:

```
<!ENTITY % adxf.node.attributes
      "id      CDATA #REQUIRED
       design  CDATA #IMPLIED
       at      CDATA #IMPLIED">
```

3.3 Sequence

Sequence is a composite node in an accelerator hierarchical tree. Then its structure repeats the accelerator model, a list of accelerator elements and subsequences:

```
<!ELEMENT sequence
    ( sequence | %adxf.element;)*>
<!ATTLIST sequence
    %adxf.node.attributes; >
```

In contradistinction to the accelerator semantics, each sequence has a parent, and it does not define a separate naming context for its children.

3.4 Accelerator Element

Element represents a minimal identified accelerator entity. There are many different element types (such as quadrupole, solenoid, *etc.*). In the object-oriented approach (see Fig. 1), all element types have to be implemented by individual classes derived from the common ancestor Element. It allows one to describe the sequence of heterogeneous elements via a homogeneous collection of Element instances. However, the present XML specification does not support any inheritance mechanism and requires the explicit inclusion of all accelerator types in the accelerator declaration. To satisfy the XML constraints and to provide the ADXF extensions, all element types have been divided into *core* and *extra* categories:

```
<!ENTITY % adxf.element
  "(%adxf.coreElement; |
    %adxf.extraElement;)">
```

The *adxf.coreElement* category contains a list of element types defined in the standardized ADXF DTD file. At this time, it comprises SIF elements[2]:

```
<!ENTITY % adxf.coreElement
   "(marker |
     drift  |
     sbend  |
     rbend  |
     ... )">
```

Extra elements can be added later in project-specific ADXF extensions. For example, the Wiggler declaration can be included in the local XML document as:

```
<!DOCTYPE accelerator [
  <!ENTITY % adxf.extraElement
             "wiggler" >
  <!ENTITY % adxf.core SYSTEM
             "adxf.dtd">
  %adxf.core;
  ...
]>
```

According to the SMF analysis patterns, all element data are naturally represented via orthogonal buckets of element attributes (such as multipole, aperture, rotation and others). All buckets have a common ancestor and can be stored in a single homogeneous container. As it has mentioned before, the present XML specification does not support any inheritance mechanism. Then the element attributes have been divided into three parts: *frame*, $''element\_type''.body$, and other buckets sharable by all element types. The *frame* represents the element geometry. The $''element\_type''.body$ bucket can be composed from a mixture of existing and new attributes relevant to the particular element type and recognizable by the corresponding integrators (trackers). The proposed model is able to describe most element types. However, there is a category of accelerator applications using composite elements with superimposed characteristics, such as the intrinsic beam-beam effects, IR quadrupoles installed in the detector solenoid region, and others. To address these tasks, each element type supports addition of an array of element insertions:

```
<!ELEMENT insertions %adxf.element;* >
...
<!ATTLIST quadrupole
   %adxf.node.attributes;>
<!ELEMENT quadrupole (
   frame?,
   quadrupole.body?,
   %adxf.buckets;,
   insertions?) >
```

The following example illustrates the definition of quadrupole data in the ADXF-compliant document:

```
<quadrupole id = "quad2">
  ...
  <insertions>
    <multipole id = "entry" at = "0.0">
    </multipole>
    <multipole id = "exit"  at = "0.1">
    </multipole>
  </insertions>
</quadrupole>
```

3.5 Bucket of Element Attributes

The element bucket encapsulates the minimal set of attributes relevant to the single effect or element feature (such the magnetic field, aperture, *etc.*). In the ADXF specification, these attributes are explicitly defined in the XML attribute-list constructs. For example, the *frame* bucket is declared as:

```
<!ELEMENT frame EMPTY>
<!ATTLIST frame
    l         CDATA #IMPLIED
    hangle    CDATA #IMPLIED
    vangle    CDATA #IMPLIED
    n         CDATA #IMPLIED>
```

where l, hangle, vangle, and n are the *frame* attributes.

There are two categories of element buckets. Members of the first category, *body* buckets, belong to a particular element type; other buckets can be shared by all elements. Table 1 and Table 2 present the ADXF proposed classification of type-specific and common element attributes, spectively.

The ADXF specification provides a consistent mechanism for integrating new element types. The same appoach has been employed for writing new common buckets. These buckets have been split also into *core* and extra collections:

```
<!ENTITY % adxf.bucket
   "(%adxf.coreBucket; |
     %adxf.extraBucket;)">
```

The *adxft.coreBucket* collection contains a list of standardized element buckets defined in the ADXF DTD file:

```
<!ENTITY % adxf.coreBucket
   "align | rotation | ... ">
```

The *extra* buckets can be added later in project-specific ADXF extensions.

4 REFERENCES

[1] F.C.Iselin, E.Keil, R.Talman. 21 January, 1998

[2] D.C.Carey and F.C.Iselin: *Standard Input Language for Particle Beam and Accelerator Computer Programs*, Snowmass, Colorado, 1984.

[3] N.Malitsky and R.Talman: *Unified Accelerator Libraries*, AIP 391, Williamsburg, 1996

[4] F. Pilat *et.al.*: *Standard eXchange Format (SXF) for Accelerator Description*, RHIC/AP/155/, 1998.

[5] N.Malitsky and R.Talman: *Study of LHC Aperture Dependence on Tune Separation Using Thin Lenses, Phase Trombones, and "Unified Accelerator Libraries"*, LHC Project Note, 1998.

[6] F.Pilat, S.Tepikian, C.G.Trahern, N.Malitsky: *A model of RHIC using the Unified Accelerator Libraries*, RHIC/AP/146, 1998.

[7] N.Malitsky and T.Pelaia: *Integration of Unified Accelerator Libraries with CESR*, CBN 98-9, March 1998.

[8] *Extensible Markup Language (XML), 1.0 WC3 Recommendation*, 10 February 1998, REC-xml-19980210.

[9] M.Blaha and W.Premerlani: *Object-Oriented Modeling and Design for Database Applications*, Prentice Hall, 1998.

[10] F.Buschmann, R.Meunier, H.Rohnert, P.Sommerlad, M.Stal: *A System of Patterns*, John Wiley & Sons, 1996.

[11] E.Gamma, R.Helm, R.Johnson, and J.Vlissides: *Design Patterns: Elements of Reusable Software Architecture*. Addison-Wisley, 1995.

Element Type	Body Attributes	Comments
marker	-	
drift	-	
sbend	knl	array of normal multipole components
rbend	ktl	array of skew multipole components
	e1	rotation angle for the entrance pole face
	e2	rotation angle for the exit pole face
quadrupole	knl	array of normal multipole components
sextupole	ktl	array of skew multipole components
octupole		
multipole		
kicker		
hkicker		
vkicker		
solenoid	ksl	solenoid integrated strength
rf cavity	volt	array of RF voltage harmonics
	lag	array of phase lags
	harmon	array of harmonic numbers
elseparator	volt	voltage
hmonitor	-	
vmonitor		
monitor		
ecollimator	-	
rcollimator		

Table 1: Element body attributes

Bucket Type	Attributes	Comments
frame	l	magnetic length along the design orbit
	hangle	horizontal bend angle
	vangle	vertical bend angle
	n	splitting number
align	x	offset in the x-direction
	y	offset in the y-direction
	z	offset in the z-direction
rotation	phi	rotation around the x-axis
	theta	rotation around the y-axis
	psi	rotation around the s-axis
aperture	shape	aperture shape
	x	horizontal half-aperture
	y	vertical half-aperture

Table 2: Common buckets and their attributes.