ADML: Aircraft Design Markup Language for Multidisciplinary Aircraft Design and Analysis

Shubhangi Deshpande[a], Layne T. Watson[a,b,c], Nathan J. Love[c], Robert A. Canfield[c], and Raymond M. Kolonay[d]

[a]Department of Computer Science, Virginia Polytechnic Institute & State University, Blacksburg, VA
[b]Department of Mathematics, Virginia Polytechnic Institute & State University, Blacksburg, VA
[c]Department of Aerospace & Ocean Engineering, Virginia Polytechnic Institute & State University, Blacksburg, VA
[d]AFRL/RQVC, 2210 8th Street, Bldg 146, Room 218, WPAFB, OH 45433

Abstract *The process of conceptual aircraft design has advanced tremendously in the past few decades due to rapidly developing computer technology. Today's modern aerospace systems exhibit strong, interdisciplinary coupling and require a multidisciplinary, collaborative approach. Efficient transfer, sharing, and manipulation of aircraft design and analysis data in such a collaborative environment demands a formal structured representation of data. XML, a W3C recommendation, is one such standard concomitant with a number of powerful capabilities that alleviate interoperability issues in a collaborative environment. A compact, generic, and comprehensive XML schema for an aircraft design markup language (ADML) is proposed here to represent aircraft conceptual design and analysis data. The purpose of this unified data format is to provide a common language for data communication, and to improve efficiency and productivity within a multidisciplinary, collaborative aricraft design environment. An important feature of the proposed schema is the very expressive and efficient low level schemata (raw data, mathematical objects, and basic geometry). As a proof of concept the schema is used to encode an entire Convair B58. As the complexity of models and number of disciplines increases, the reduction in effort to exchange data models and analysis results in ADML also increases.*

Keywords: XML schema; markup language; interoperability; multidisciplinary design; unified data format; conceptual aircraft design.

1. Introduction

Aircraft design by nature is a multidisciplinary process where several different disciplines (see Figure 1) such as geometry, structures, aerodynamics, controls, propulsion, flight mechanics, and so on contribute to achieving an optimal design adhering to all the design constraints for all the disciplines involved. The first step in the design process, the conceptual design, is characterized by a large number of design alternatives and trade-off studies, and a continuous, evolutionary change to the aircraft concepts under consideration [22]. Conceptual design is primarily a search process that requires an extensive exploration of the design space in order to gain insight into the relations between the design variables and the aircraft performance. It formulates a set of design variable quantities, which, according to appropriate modeling principles and design constraints, defines a vehicle that fulfills a set of minimum requirements determined by the vehicle mission. Traditional conceptual design was conducted as isolated disciplines with low fidelity interdisciplinary coupling and mostly linear interactions between disciplines. However, due to rapidly developing computer technology and algorithmic improvements, conceptual design methods have advanced tremendously in the past few decades. Today's modern aerospace systems exhibit strong interdisciplinary coupling and require a multidisciplinary, collaborative approach [24]. Aircraft design has become a collaborative endeavor that involves many individuals from diverse groups around the world working together in an extended enterprise environment to achieve a common goal. Advances in computer capacity and speed, along with increasing demands on the efficiency of the aircraft design process, have intensified the use of simulation based design and analysis tools to explore design alternatives both at the component and system levels. Analysis methods that were once considered feasible only for advanced and detailed design are now available and even practical at the conceptual design stage. Rapid analysis methods also allow multidisciplinary design optimization methods to be implemented in conceptual design. This changing philosophy of conducting the conceptual aircraft design and analysis poses additional challenges beyond those encountered in a low fidelity design and analysis

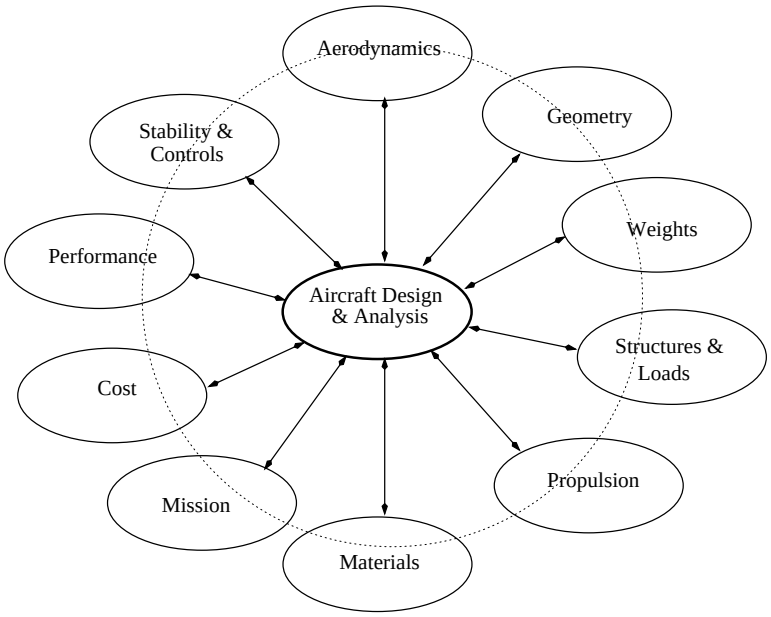

Figure 1. Disciplines involved in an aircraft design process.

of aircraft. Although the use of sophisticated design and analysis tools has become prevalent in the aerospace community, the field of interdisciplinary communication still remains in a primitive state. Aircraft design systems are not yet equipped with the state-of-the-art in data representation and communication that are prevalent in several other domains. An objective of this paper is to propose a unified data approach for bridging the gap in data communication methods and making the aircraft design process more agile.

Aircraft design and analysis involves manipulation of a large amount of interdisciplinary data including, but not limited to, inputs and outputs. Efficient transfer, sharing, and manipulation of aircraft design and analysis data across different platforms, applications, and users demands a formal structured representation of data in a well organized data sharing and validation environment. In general, due to the lack of a uniform representation, the same information is duplicated several times, each time in a different format specific to the underlying implementation. In order to exchange this information between different disciplines/applications/users, a translator needs to be designed that converts one format to/from another at every facility and for each format. Thus, the lack of a standard, uniform representation results in redundancy in codes and duplication of information and efforts incurring a lot of maintenance overhead. Another common problem with this kind of data exchange is data inconsistency. All these factors inhibit sharing and exchanging of interdisciplinary data and greatly hinder the conceptual design process making it less efficient. To alleviate this burden, a unified system is sought that provides certain capabilities for modeling the massive amount of multidisciplinary data, such as portability, maintainability, reusability, platform independence, integrity, (syntactic) correctness, and system recovery. With a platform and language independent data exchange standard like XML (extensible markup language), information can flow seamlessly in a heterogeneous environment with diverse computing platforms, programming languages, and hardware systems. This paper proposes some first steps for the conceptual aircraft design and analysis community to move in this same direction.

2. Background

2.1. Motivation

The former multidisciplinary optimization branch (MDOB) at the NASA Langley research center (LaRC) [16] identified frameworks for multidisciplinary analysis and optimization research, and promoted a multidisciplinary, collaborative approach and sharing of information among disciplines for aircraft design and analysis systems. The advanced engineering environments (AEE) study committee sponsored by NASA has investigated a number of technical, managerial, cultural, and

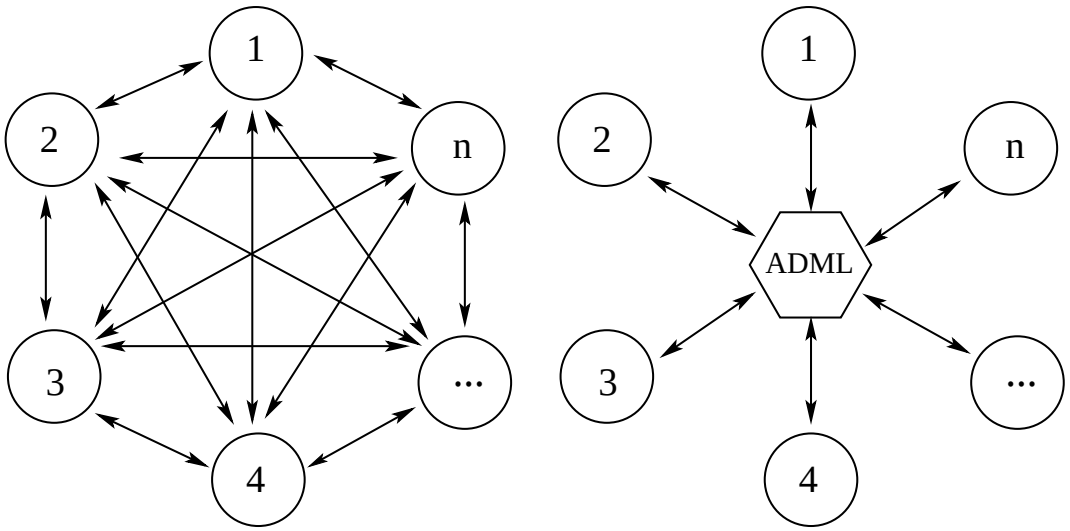

Figure 2. n^2 interfaces (left) without a standard data format vs. n interfaces using ADML (right).

educational barriers that need to be overcome in order to realize a multidisciplinary, collaborative approach [24]. Several design requirements related to information management and integration of tools, systems, and data need to be addressed first in order to realize a unified system. Based on the knowledge gained from the frameworks proposed by the MDOB ([14] and [25]), this section outlines several desirable features related to data management pertinent to multidisciplinary aircraft design, analysis, and optimization.

- Use of standards: Use of standard data formats facilitates maintainability of codes and eliminates duplication of information and effort.
- Data sharing: Intra- and interdisciplinary data sharing in a multidisciplinary, collaborative environment is a crucial feature for solving interoperability issues and automating design processes.
- Extensibility: Advances in aircraft conceptual design processes entail a flexible and extensible data format for supporting different design variants as well as new configurations.
- Platform independence: A platform, language, and vendor neutral format is sought for seamless data communication across different platforms, applications, and users in a multidisciplinary environment.
- Object oriented programming: The data format should support object oriented design principles that facilitate aircraft design processes with several useful capabilities such as data binding and integration, object encapsulation, extensibility through inheritance, flexibility through polymorphism, and so on.

The implementation of a data standard adhering to all these requirements is a major challenge. A platform and language independent format to represent aircraft design and analysis data is a desirable way to meet all these requirements and support a multidisciplinary system distributed across a network of heterogeneous computing environments. This is the motivation behind proposing an XML based data format as a first step towards meeting that challenge.

A multidisciplinary collaborative design environment enables engineers to cooperate by means of structured and mostly autonomous exchange of information. This exchange is mostly conducted through input/output interfaces between design and analysis modules. Hence, the number of interfaces is a critical factor for an efficient exchange of data and a central information model is a key feature. As indicated in Figure 2 (left) the number of interfaces required without a standard data format grows quadratically ($O(n^2)$) with the number n of disciplines, application codes, or users. However, using a unified data format such as XML, the number of interfaces grows linearly in n resulting in ($O(n)$) interfaces.

2.2. Existing data formats, standards for representing product data

Several data modeling languages and technologies have emerged over the past two decades or so for representation and exchange of product manufacturing information. IGES ([13], [18]) is a language neutral data format that allows exchange of product data among computer aided design (CAD)

systems. A vendor neutral system CAPRIS is described in [10] for accessing a variety of CAD systems though a unified and simple programming interface. CAPRIS maintains a boundary representation (BRep) data structure common to all participating CAD systems. CAPRIS uses SOAP (simple object access protocol) for exchanging structured information and relies on XML for messaging. However, the geometry schema for CAPRIS is not publicly available. A successor of IGES, STEP (standard for exchange of product model data), is a family of standards defining a robust and time-tested methodology for describing product data throughout the lifecycle of a product. STEP is a comprehensive ISO standard (ISO 10303) ([20], [21]) that describes a mechanism to represent and exchange product data and has been widely used in the aerospace, automobile, electrical, electronic, and other industries [5]. As discussed in [19], STEP has a proven record of success in modeling aircraft geometry. The part STEP AP209 (application protocol: composite and metallic structural analysis and related design — ISO 10303-209:2001) of STEP has been developed to address data exchange in a design/analysis/manufacturing process. The second edition of AP209 has recently been renamed as "multidisciplinary analysis and design", and is in the final stage of development as of June 2013 [1]. STEP uses a data modeling language called EXPRESS ([20], [27]) to describe and exchange product data between CAD, CAM (computer aided manufacturing), CAE (computer aided engineering), and other CA* systems. EXPRESS combines ideas from the entity-attribute-relationship family of modeling languages with object modeling concepts. However, unlike XML, EXPRESS is not easily extensible and is not supported by many widely used software tools. Although EXPRESS provides rich facilities for data modeling at the semantic level, unfamiliarity of today's application programmers with the traditional STEP based data modeling techniques impedes its widespread usage. Furthermore, XML has become a de facto standard for representing and exchanging digital data for several domains, including domains that are within the scope of STEP. Since STEP can semantically model the high fidelity information required by many XML applications, the STEP data modeling standard and XML are complementary technologies. It is a logical next step to merge the traditional STEP technology within XML. With the integration of the two, the best of both worlds can be achieved.

2.3. Rationale for using XML

XML, a W3C (World-Wide Web Consortium) recommendation [29], is a standard concomitant with a number of powerful capabilities (extensibility, flexibility, reusability, maintainability, and so on) and a generic, robust syntax for developing specialized markup languages. Unlike HTML (hypertext markup language), XML by itself specifies neither preconceived semantics nor a predefined tag set; it instead provides a means for defining content and semantics of XML documents. One of the major requirements in a multidisciplinary collaborative environment is the data sharing ability to overcome disciplinary isolation. The platform, language, and vendor independent format of XML makes it well-suited to the task of satisfying multidisciplinary aircraft data requirements.

XML is a profile of an existing ISO standard, ISO 8879, known as SGML (standard generalized markup language) [12], and is an acceptable candidate within other ISO standards without further standardization ([16]). The simple ASCII text format of XML allows aircraft applications running on heterogeneous systems with diverse platforms to readily communicate with each other. Aircraft design application written in any programming language can process the same XML document without any modification, thus eliminating redundancy and offering reusability. In addition, the inherent hierarchical nature of XML provides a way to define structural relationships that exist in the data and facilitates application of object oriented principles to conceptual aircraft design data. Name, attributes, and content of an XML element are closely related to class name, properties, and composition associations in an object oriented aircraft design. Thus, with the use of an XML based markup language, it is possible to faithfully model aircraft design and analysis data as well as structural and functional relationships among different data elements.

A variety of XML parsers for almost all high level programming (and scripting) languages are abundantly available for automatic generation and parsing of XML content. XML itself is a metalanguage—a language that is used to define an unlimited number of special purpose markup languages. XML data semantics (grammar) can be specified using either a document type definition (DTD) [29] or an XML schema [28]. An added benefit of using DTD or XML schema is that they provide support for data validation. A data file encoded in XML is considered valid if it complies with the corresponding DTD or XML schema. Without using a schema for an XML document, a

separate validation tool needs to be implemented. An XML schema provides additional significant advantages over a DTD, such as more advanced data types and a very elaborate content model. The aircraft design markup language proposed here is based on XML schema.

2.4. XML based markup languages pertinent to multidisciplinary aircraft design

There are several XML based languages developed for various application domains. There are compelling examples of success from various disciplines, e.g., a systems biology markup language (SBML) [11] developed for systems biology models and data; MathML [26], an XML based language developed for mathematical notations; Office Open XML, a Microsoft file format (commercial application) for storage of electronic data, and many more.

Although the aerospace industry is no exception for developing XML based standards for exchanging aircraft data and models, there are only a handful of successful examples. The JSBSim flight dynamics model software library [4] is a batch simulation application aimed at modeling flight dynamics and control for aircraft. JSBSim is an XML based model description specification where input files are supplied in XML format. These XML files contain descriptions of aerospace vehicles, engines, scripts, etc. DAVE-ML is an XML based markup language for a draft AIAA flight dynamic model exchange standard [2], inspired by JSBSim, for the interchange of flight dynamics modeling data between facilities. Both JSBSim and DAVE-ML are intended to provide a platform and language neutral format for exchanging flight dynamics modeling, verification, and documentation data where the major XML elements are mathematical objects. However, JSBSim provides its own XML tags for representing mathematical constructs (e.g., product, sum, quotient, etc.), whereas DAVE-ML uses the verbose MathML format for representing mathematical constructs.

An XML based markup language, MatML [3], developed in coordination with the National Institute of Standards and Technology (NIST) targets multiple industries for facilitating the exchange of a wide variety of material properties. The latest version of MatML (MatML3.0 and beyond) ported the language specification from DTD to XML Schema, and has many refinements over previous versions.

The finite element modeling markup language (femML) [7] was proposed to address the data interpretation and application interoperability in the finite element modeling domain. The project was initiated by members of the composite materials and structures group at the Naval Research Laboratory and the International Science and Technology Outreach Society. femML uses MatML as a namespace in its specification.

All these XML based markup languages discussed heretofore target a single or a subset of disciplines involved in a multidisciplinary environment. A recent development effort at the German aerospace center DLR involves a new data exchange format CPACS (common parametric aircraft configuration schema) for representing all the necessary data required for conceptual and preliminary aircraft design and analysis. After evaluating how well the proposed ADML effort fits within the context of CPACS, it was found that the goals of the collaborative exercise at DLR are closely aligned with ADML objectives; however, the fundamental difference is that the ADML effort started bottom-up with powerful constructs for functions and abstract mathematical objects, and with unconventional aircraft configurations in mind, whereas the current version of CPACS started top-down from entire aircraft to single data objects (point lists), and can only currently handle traditional aircraft designs.

3. Aircraft Design Markup Language (ADML)

The ADML project started with an intention to address the data communication needs of the recently founded (2009) Collaborative Center for Multidisciplinary Sciences (CCMS) for the development of future aerospace vehicles, involving Virginia Tech, Wright State University, and Air Force Research Laboratory at Wright Patterson Air Force Base (WPAFB), Ohio. The collaborative center specifically investigates multidisciplinary analysis and design of several futuristic aircraft such as the joined-wing SensorCraft, flapping micro air vehicles, and efficient supersonic air vehicles. A flexible, extensible, and comprehensive XML based format ADML is proposed to handle these futuristic aircraft designs.

ADML is based on XML technologies making it human readable and computer processable. It is designed to accommodate data for numerous disciplines involved in the conceptual design phase

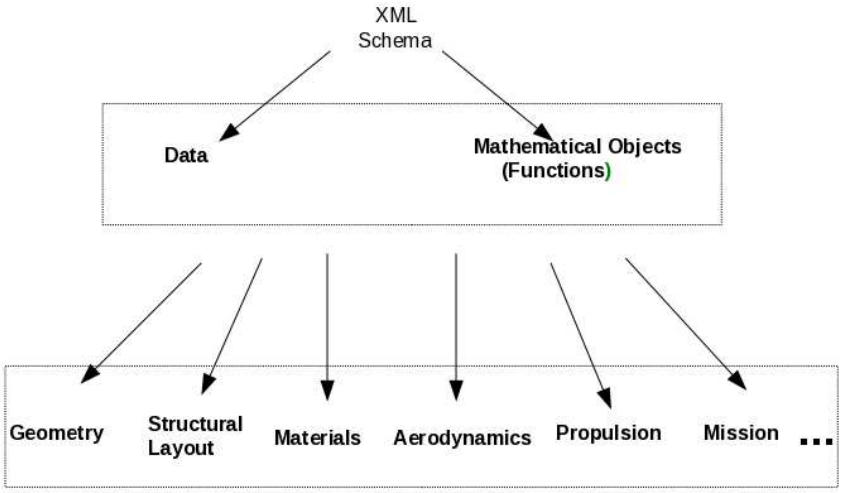

Figure 3. ADML development approach.

and can be extended to high fidelity analysis. ADML includes capabilities for a model to be self-validating and self-documenting, with the provenance of a model's components included within the model and transferred with it (see Section 3.3.2(B) for a detailed description).

A specialized grammar of ADML, the ADML schema, provides a format for the exchange of the aircraft design and analysis data, therefore each discipline is required to design import/export tools that comply with the schema one time only. In this data-centric setup the number of interfaces is minimal and effective communication can be established, resulting in substantially reduced cost and time required to exchange aircraft data. Use cases (presented in Section 3.4) have indicated significant reduction in effort to exchange simple models when utilizing this format. Even greater benefits could be attained for large complicated models or more disciplines.

Although an XML based markup language is well-suited for addressing interoperability issues involved in a multidisciplinary, collaborative environment, the actual development is not as easy as it first appears. Developing a generic, comprehensive, and compact XML schema for each and every discipline involved in the aircraft conceptual design phase is a very challenging task. Every discipline has its own set of modeling requirements and constraints that adds up to the overall complexity of the final design. Accommodating new aircraft configurations for futuristic air vehicles is even more challenging, and demands a comprehensive and extensible data format. The inherent hierarchical nature and extensibility of an XML schema plays a significant role in structuring various components of conceptual aircraft design. Figure 3 presents a simplified version of the bottom up development approach of the ADML schema. The specification for the ADML schema would need to include the capability to define aircraft data specific to each and every discipline and component or subsystem of the aircraft involved in the conceptual design phase. An overview of the ADML schema modules (data, functions, basic geometry, and high level aircraft design constructs) and the existing and the future modeling capabilities of the proposed XML schema follow.

3.1. Low level schemata: common components

3.1.1. Data schema

At a very high level, everything is data. However, the rationale for dividing the XML schema in different sections (data, functions, geometry, etc.) is to exploit the functional and logical distinction among different aircraft model objects and to maintain their inherent hierarchy. The data schema is at the lowest level of the hierarchy in a top down view, representing the simplest form of data. Elements of the data schema are used as the building blocks for all other higher level elements.

Most of the elements in ADML require a *name*, a *description*, and a *unit* associated with them. Therefore, a complex type XML element *nd* is defined to encapsulate these elements. All the elements in the data schema, as well as in other ADML modules, that require any or all of these descriptive identifiers (name, description, and/or unit), can be derived from the *nd* element as a base.

The major element of the data schema is the *variable* element. Variables are used to define inputs and/or outputs to/from a design or an analysis. A variable element has a human readable *name*, a *description*, a *unit*, a *value*, a *min*, a *max*, some *flags*, and other *scalar* parameters associated with it, and a machine readable variable identifier, *vid*. A variable defined in an ADML document can be referenced at a later point in a mathematical expression. The value of a variable element consists of a *scalar* (an atomic value) or a *tensor* (a multidimensional array). A tensor is an ADML element defined recursively to represent an array of arbitrary dimensions. A higher rank tensor is defined in terms of a lower rank tensor. A vector (*vtype* element) is a rank 1 tensor and a matrix is a rank 2 tensor. In general, a k-dimensional array can be defined as a rank k tensor, e.g., a $2 \times 3 \times 4$ tensor is defined as a sequence of two 3×4 matrices that are defined in terms of three 4-dimensional vectors each. A typical use of a tensor element could be to define relational data (function tables). An example of a $2 \times 3 \times 4$ tensor follows, omitting the schemata defining the tags.

```
<variable>
 <name>tvar1</name>
 <description>2x3x4 tensor example</description>
 <value>
  <t>
   <t>
    <v>1 2 3 4</v>
    <v>1 2 3 4</v>
    <v>1 2 3 4</v>
   </t>
   <t>
    <v>5 6 7 8</v>
    <v>5 6 7 8</v>
    <v>5 6 7 8</v>
   </t>
  </t>
 </value>
</variable>
```

3.1.2. Representing mathematics

A significant part of aircraft design and analysis data comprises mathematical objects such as functions, expressions, arbitrary dimensional lists, and operators. Therefore, communicating mathematical objects among different entities (applications, users, and/or platforms) plays a crucial role in exchanging data in a multidisciplinary, collaborative conceptual aircraft design and analysis environment. Careful thought has been given to a format for representing mathematical constructs while developing the proposed ADML schema. Three possible candidates are the XML based markup language MathML and two widely used computational software tools, Mathematica and Matlab. The most significant advantage of using a MathML format to represent mathematics is that MathML itself is an XML based markup language and can be parsed and validated easily using available XML parsers; however, MathML is an extremely verbose and unreadable format. Editing mathematical expressions in MathML requires a special editor because the markup is very complex. This makes it impractical to edit by hand. Furthermore, the conceptual aircraft design and analysis community is more interested in communicating content rather than representing mathematical objects. Moreover, Matlab and Mathematica are among the most popular tools used to evaluate mathematical expressions in the aerospace community. Therefore, this paper proposes the use of Mathematica or Matlab syntax over the verbose MathML format for representing mathematics. However, if an application needs to parse the mathematical data being exchanged at the other end, then parsing subroutines need to be written specific to the underlying implementation. The supported format is more useful when the mathematical objects being exchanged are meant to be passed to either Matlab or Mathematica tools for evaluation. For sharing mathematical data (mathematical lists or

arrays) that are meant to be parsed, an application should make use of the more relevant and easy to parse XML element, tensor, defined in the data schema.

The major schema elements for representing mathematical objects include *operator*, *relation*, *mlist*, and *expression*. These elements can be represented in either Mathematica or Matlab format using the *format* attribute associated with them.

An *operator* is a generalization of the familiar notion of a function. Typically, an operator is used to represent the operations performed on functions to produce other functions. An example of an operator on functions, composition with a Bessel function, represented in the Mathematica format follows, omitting the schemata defining the tags.

```
<operator format="Mathematica">
 <name> f </name>
 <description>
  Composition with Bessel function of the first kind, order 0
 </description>
 <domain> AnalyticFunctions </domain>
 <range> AnalyticFunctions </range>
 <arguments> z </arguments>
 <definition>
  f[z_][x_] := BesselJ[0,z[x]]
 </definition>
</operator>
```

Another type of element is the *relation* element. A relation might be defined by an expression that involves logical or relational operations. A relation can also be viewed as a subset of the Cartesian product of k sets. Thus, the first $k-1$ values in a k-tuple correspond to the arguments or inputs to the relation, and the kth value corresponds to the output. The corresponding relation table can be defined using the *mlist* element. Although an mlist element somewhat resembles a tensor element from the data schema, its intended usage is quite different. A tensor element is primarily used to transfer a multidimensional array across different systems (platforms or users) and not for manipulating the array. However, the intended use of an mlist element is to define and manipulate an arbitrary list structure (where every list element can have a different cardinality). A tensor, being a recursive XML element, facilitates an easy parsing process at the other end, whereas an mlist element has the advantage of a compact representation using either Matlab or Mathematica format. The rationale for having two different elements (expression and relation) to represent mathematical relations is that a relation is a special type of an expression involving only relational operations. The intended use of an expression element is to represent intermediate computations or evaluations in an analysis or a design process.

The schema definition for an expression element and an example of an expression that estimates the drag divergence Mach number (M_{dd}) as a function of an airfoil technology factor (K), the thickness-to-chord ratio (t/c), the lift coefficient (c_l), and the sweep angle (L) follow, again omitting some of the schemata.

```
<xs:element name="expression">
 <xs:complexType>
  <xs:complexContent>
   <xs:extension base="nd">
    <xs:sequence>
     <xs:element ref="variables" />
     <xs:element ref="definition" />
     <xs:element name="patternStr" type="xs:string" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute ref="format"/>
    <xs:attribute name="eid" type="xs:ID"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:element>
```

```
<expression format=Mathematica>
 <name>M_dd</name>
 <description>
  estimation of drag divergence Mach number
 </description>
```

```
 <variables>K, L, t, c, c_l</variables>
 <definition>
  M_dd=K/Cos[L]-(t/c)/Cos^2[L]-c_l/10*Cos^3[L]
 </definition>
</expression>
```

A simple example of an *mlist* (generalization of tensor), omitting some of the schemata, is

```
<xs:complexType name="mlist">
 <xs:complexContent>
  <xs:extension base="nd">
   <xs:sequence>
    <xs:element ref="variables"/>
    <xs:element ref="definition"/>
   </xs:sequence>
   <xs:attribute name="lid" type="xs:ID"/>
   <xs:attribute ref="format"/>
   <xs:attribute ref="structure"/>
   <xs:attribute name="dimension" type="xs:string"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

```
<mlist format="Mathematica" structure="general">
 <name>L</name>
 <description>an arbitrary list structure</description>
 <definition>
  { { 0,0.1 },{ { 18,-0.1 },{ 19,-0.09 } },
  { { 20,-0.08 },{ 22,-0.05 },{ 23,-0.05 } },
  { { 25,-0.07 },{ 27,-0.15 },{ 90,-0.6 } } }
 </definition>
</mlist>
```

3.1.3. Basic Geometry

The ADML schema for aircraft geometry starts with low level, common geometry elements such as *point*, *pointList*, *line*, *plane*, *nurbs*, and *frame*, and builds up more complex components of an aircraft such as airfoils, wings, fuselages, etc. A *point* is defined as a list of real values (coordinates); a *line* is defined using two *point*s; and a *plane* is defined using a *point* and a *normal*.

Another fundamental geometry element is *nurbs*. The geometry schema presented in this paper supports NURBS (nonuniform rational B-spline) based geometry model to represent curves and surfaces. Each *nurbs* element is defined in terms of a set of control points (the *controlPoints* element), a knot vector (the *knotVector* element), a weight vector (the *weightVector* element), and the NURBS order (the *order* element). The XML schema definition for a NURBS element and an example of a NURBS curve of order three with five control points associated with five weights and eight knots follow.

```
<xs:simpleType name="nurbsType">
 <xs:restriction base="xs:string">
  <xs:enumeration value="curve"/>
  <xs:enumeration value="surface"/>
  <xs:enumeration value="BezierCurve"/>
  <xs:enumeration value="BezierSurface"/>
 </xs:restriction>
</xs:simpleType>
```

```
<xs:complexType name="nurbs">
 <xs:complexContent>
  <xs:extension base="nd">
   <xs:sequence>
    <xs:element name="controlPoints" type="mlist"/>
    <xs:element name="knotVector" type="mlist"/>
    <xs:element name="weightVector" type="mlist"/>
    <xs:element name="order" type="xs:integer"/>
   </xs:sequence>
   <xs:attribute name="ntype" type="nurbsType" />
   <xs:attribute name="ncp" type="xs:integer" />
```

```
  <xs:attribute name="nurbsID" type="xs:ID"/>
 </xs:extension>
</xs:complexContent>
</xs:complexType>

<nurbs ntype="curve" nurbsID="NC1">
 <name>ncurve1</name>
 <description> NURBS curve of order 3 </description>
 <controlPoints format="Mathematica" structure="array">
  <definition>
   { { 0,0,0.5 }, { 0,-0.5,0.5 }, { 0,-0.5,0 }, { 0,-0.5,-0.5 }, { 0,0,-0.5 } }
  </definition>
 </controlPoints>
 <knotVector format="Mathematica">
  <definition> { 0,0,0,0.5,0.5,1,1,1 } </definition>
 </knotVector>
 <weightVector format="Mathematica">
  <definition> { 1,0.707107,1,0.707107,1 } </definition>
 </weightVector>
 <order>3</order>
</nurb>
```

3.2. High level aircraft design constructs

3.2.1. Modeling aircraft geometry

Rapid development of computer technology over the past decade has changed the conduct of conceptual aircraft design. Aircraft analysis methods that were considered feasible only for advanced and detailed designs are now available and even practical at an early stage of the aircraft design process. To fully exploit the available computing resources and analysis methods, the geometric model of aircraft must be generated rapidly and easily so as not to inhibit the conceptual aircraft design process. However, aircraft geometry is one of the most complex constructs among various conceptual aircraft design components, and likewise the representation of the geometry model is complex.

In the development of any new aircraft, the outer mold line (OML) is key to designers in almost every discipline. Core differences in the utilization of the aircraft geometry often lead to the development of multiple aircraft representations, that cater to different design disciplines, only later to be merged into one final aircraft design. Not only is this process inefficient, it is also difficult to implement in an MDO framework that executes autonomously. Multidisciplinary analysis and design requires a single parametric geometry representation for a configuration that is shared amongst the various disciplines involved. The software behind most commonly used CAD systems is extensive and tailored to serve its community of mechanical engineers. In contrast, computational design optimization is an extension of conceptual design merged with high fidelity computational models; the geometric requirements are significantly specialized compared to general industrial CAD systems. Taking this requirement into consideration, a geometry model supported in ADML is the practical implementation VT-CST ([17]) of class shape transformation (CST, [15]) developed at Virginia Tech. VT-CST is capable of rendering tailless supersonic configurations with embedded engines as well as conventional and joined-wing configurations. In addition, ADML can handle other types of parametric geometries such as boundary representation (BRep) constructs using NURBS curves.

3.2.2. Representing an entire aircraft

As mentioned previously, ADML development follows a bottom-up approach where all the basic common components are defined first, and other more complex high level constructs are built using the low level schemata as and when required. The root elements for the low level schemata are *data*, *functions*, and *geometry*, and that for the high level aircraft design and analysis is *aircraft*. Each *aircraft* element consists of one or more instances of *model* and *analyses* elements. An aircraft design is described using a *model* element, which consists of *wings*, *fuselages*, *landingGears*, and *propulsion* as subelements along with some catalog elements such as *materials*, *performance*, *mission*, and *global*. Figure 4 shows the hierarchy for the first few levels of the ADML schema. Owing to the complexity

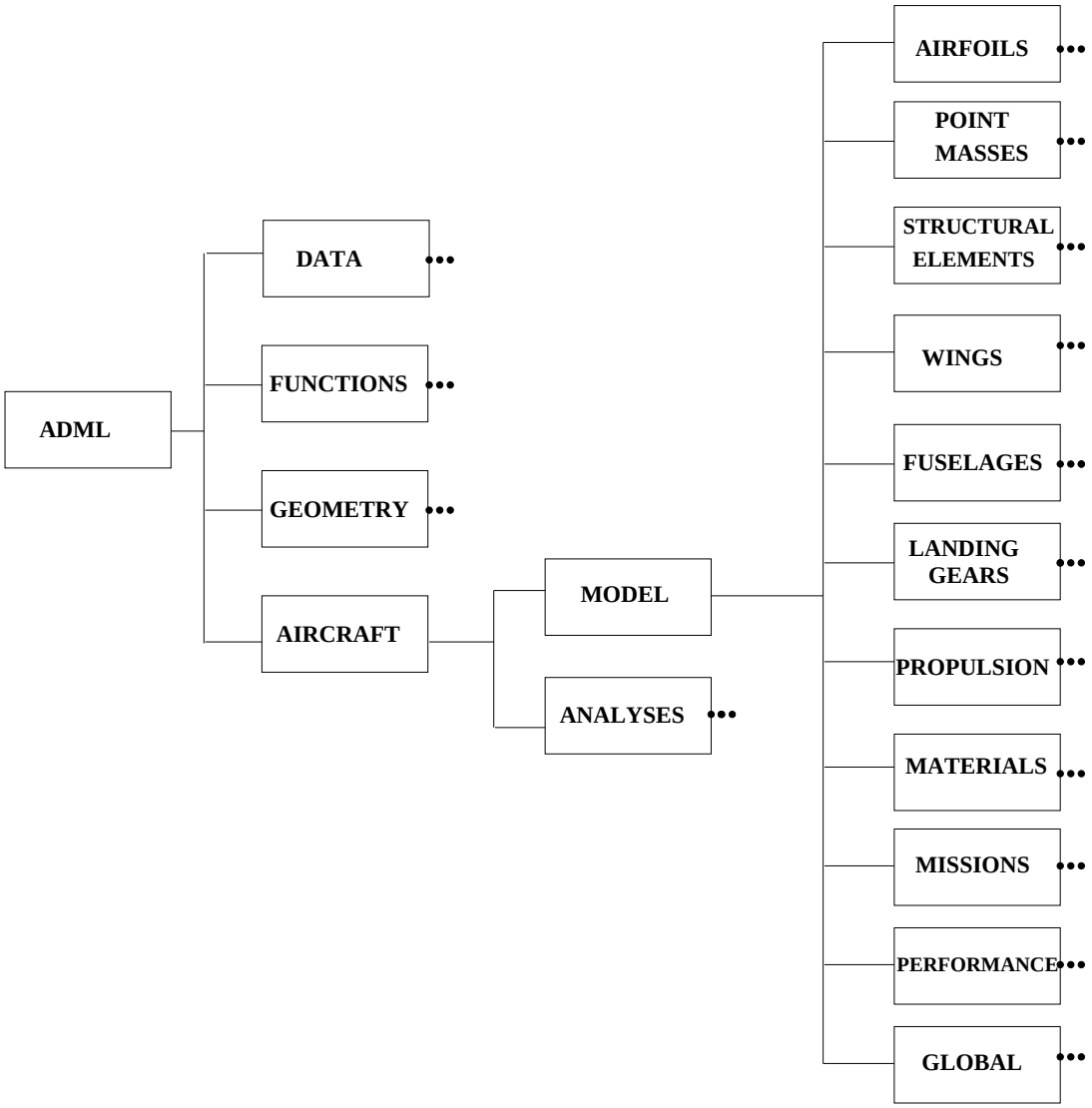

Figure 4. ADML taxonomy.

of the ADML schema and the large number of XML elements needed to represent aircraft design components, it is not feasible to list and discuss each and every element in this paper. Instead a brief discussion of the high level elements follows with the full ADML schema in Appendix A.

< airfoils >

The *airfoils* element consists of a sequence of *airfoil* elements. Each *airfoil* element is defined, using the choice data structure available in the XML Schema, as a choice among a parametric definition, a NURBS based cross section, a VT-CST based geometry definition, or a string reference to an external definition (for example a NACA airfoil definition). An airfoil defined in such a way can be referenced in a *wing* definition using the associated *uID* attribute.

< pointMasses >

A primary goal in airplane conceptual design is to determine an estimate of the mass and the moment of inertia tensor of the airplane. There are many internal components of an airplane that are difficult and/or unnecessary to precisely define until a later stage in the design cycle. However, the mass and the moment of inertia tensor of these objects must still be accounted for in the conceptual airplane design as they have a nonnegligible mass thus directly impacting the structural design and performance of the airplane. Therefore, instead of creating a separate definition for each

individual type of object, only a simple description of the location of the component, its mass, and its moment of inertia tensor are required. Usually the mass and the moment of inertia tensor of such components are estimated using empirically based methods and a simple Cartesian location is used to place the object in or on the airplane. In the ADML schema, a *pointMasses* element has been created to account for any component of the airplane which could be described in this way. Each *pointMasses* element is defined as a sequence of *pointMass* elements. A *pointMass* is defined using four elements: *mass*, *inertiaTensor*, *location*, and *provenance*.

< structuralElements >

The primary members that make up an aircraft structure are beams and plates. However, in structural analysis using the finite element method, beams can be and are often modeled as plates depending on their shape. In modern aircraft design, one may wish to evaluate a structure made up of multiple different materials including both metals and laminated fiber composites. For example, one design may utilize laminated fiber composite spars whereas another uses traditional aluminum spars. In order to handle this variation in material properties (and thus the number of design variables necessary), two descriptions of the properties of a plate were developed, one for isotropic materials (metals) and one for laminated composites (fiber composites and sandwich panels). This definition is generic enough that it can be used to define the properties of all of the primary structural members of a wing: the spars, the ribs, and the skin panels.

The *structuralElements* element consists of a sequence of two elements, *structElementIsotropic* and *structElementComposite*, corresponding to plates made up of isotropic and composite elements, respectively. A section of a wing can have vectors of references (using the associated *uID* attribute) to these elements. This in effect divorces the material properties and the thicknesses of the structural members from the structural layout allowing for a designer to easily switch material properties and thicknesses by simply changing the reference numbers to the properties.

< wings >

The *wings* element consists of a sequence of *wing* elements that define instances of wings and/or tails of an aircraft. Each *wing* element is defined as a sequence of *planform*, *structure*, and *controlEffectors* elements, and a set of attributes. The airfoil geometry can be in VT-CST format or NURBS based. In ADML, a wing *structure* is defined as a set of *section*s; each *section* is defined in terms of ribs and spars (using number of, thickness, materials, etc.). This simplification is assumed to be sufficient for a wing definition in the conceptual design phase. Each *section* can have a different number (thickness and material) of spars and ribs, that way adding flexibility to accommodate a myriad of wing configurations.

< fuselages >

A *fuselages* element consists of one or more *fuselage* elements that follows VT-CST's parametric geometry definition from [17] drawn from a cross section class function (in the Y-Z plane) defined along a distribution class function (in the X-Z plane), both of which are scaled with the length and width of the desired fuselage.

< landingGears >

A *landingGears* element is comprised of a sequence of one or more *landingGear* elements chosen from three different configurations, tricycle, quadricycle, and multibogey, each defined using a set of design parameters (mass, lowered and raised coordinate combinations in X, Y, Z, etc.). A tricycle or a multibogey configuration has one nose gear centered on the aircraft body whereas a quadricycle configuration has two nose gears.

< propulsion >

Each *propulsion* element consists of four subelements, *engines*, *cowls*, *ramps*, and *EEWSs* (EEWS stands for engine exhaust washed structures), and follows VT-CST's parametric definition. As mentioned earlier, VT-CST was developed to design tailless supersonic aircrafts with embedded engines. An *engines* element is defined as a sequence of one or more *engine* elements that are defined

as a set of design parameters. Likewise each of these *cowls*, *ramps*, and *EEWSs* elements is defined as a sequence of one or more *cowl*, *ramp*, and *EEWS* elements, respectively.

< materials >

The data defined for the *materials* element is classified as a reusable dataset, and a reference to material IDs is provided in other elements to encode their material properties. Two types of materials—isotropic and orthotropic—are defined in a *materials* element.

< missions >

Here, a list of *missions* can be specified. The *missions* are built up from *mission* segments, which allow for simple conceptual design definitions. An aircraft uses a reference to one or more of these *missions* as its design missions. A mission segment is a specific maneuver that the airplane is designed to perform. For example, a mission segment for the aircraft to cruise would contain the desired cruising altitude, cruising Mach number, and a specified distance or flight time. The missions are typically used in a flight performance analysis and optimization. The *missions* element is again classified as a catalog element, and is defined outside the aircraft *model* element.

< performance >

The performance element is utilized to encapsulate some high level information regarding the behavior or limitations of the aircraft design. Often these parameters will be the result of an analysis such as flight performance, structural analysis, or aerodynamics. However, these parameters may also be used as constraints on the design in an analysis depending upon the users desire. An example of some data that would be stored under the performance element are the maximum cruise Mach number, maximum altitude (service ceiling), maximum range for certain fuel and payload levels, or dive speeds at certain altitudes.

All these high level constructs constitute the third level in the ADML taxonomy as shown in Figure 4.

3.3. Features of ADML

3.3.1. Modular development

Modular schema development facilitates logical decomposition of XML elements into subsets where each individual subset focuses on specific functional capabilities thereby enabling reusability. Each small subset or module that results from this exercise can work as a building block for other more complex modules thereby enabling extensibility. The inherent modular or hierarchical structure of multidisciplinary aircraft design elicits modular schema development. The top level modules in the XML taxonomy, data and mathematical objects, serve as the foundation for developing more complex aricraft design constructs that appear at a lower level in the inheritance hierarchy. Every discipline involved in an aircraft design phase can be viewed as a separate module in the XML schema development process and can be used either as a single, isolated entity or as a part of a hierarchical structure built by combining several disciplines together.

3.3.2. Object oriented approach

A W3C XML schema, with a hierarchical type system, closely resembles an object oriented programming paradigm. Amongst the significant features of an XML schema are extensions (and restrictions), element references, and an object like behavior of an element (that carries attributes and other elements). The modular schema development of the proposed schema, as discussed in the previous subsection, facilitates reusability and extensibility. All the high level elements (corresponding to high level constructs in an aircraft design) in ADML follow an object oriented programming approach. The aircraft design applications such as VT-CST (written in C++) that use object oriented technologies can greatly benefit from this by converting the ADML schema to the classes of the high level language, and then accessing the schema elements as objects of those classes.

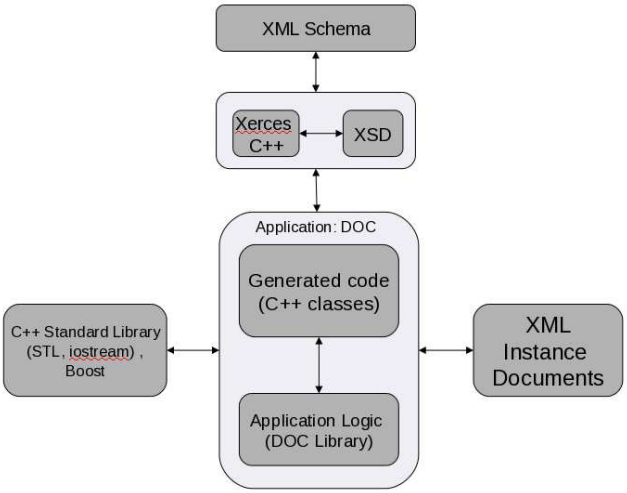

Figure 5. Integration of geometry schema with DOC project.

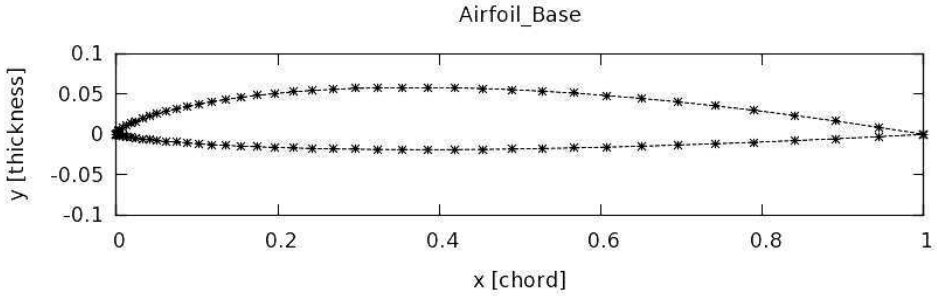

Figure 6. Airfoil geometry.

3.3.3. Provenance capability

A provenance capability is provided for all the high level constructs to describe the origin or history of the associated data, and is defined as an XML string describing author, date, etc.

4. Use Cases

4.1. Airfoil shape optimization

A C++ project, design optimization in C++ (DOC) [6], that computes design sensitivities for conceptual aircraft design applications is used as a pilot project to demonstrate an application of the proposed ADML schema. An open source, cross platform W3C schema to C++ data binding compiler, Codesynthesis XSD, is used to convert the XML schema to C++ classes. Once the C++ classes are generated from the XML schema, the data stored in XML instance documents can be accessed through the C++ objects (member variables and functions) rather than dealing with the intricacies of reading and writing XML. The software architecture of the application in Figure 5 depicts the geometry integration and related tools/packages. The XSD software uses Xerces-C++ as the underlying XML parser. Xerces-C++ is a validating XML parser written in a portable subset of C++ and is available under the Apache Software License.

The geometry schema presented in this paper supports a NURBS (nonuniform rational B-spline) based geometry model to represent curves and surfaces, as for the airfoil shown in Figure 6. Below is the XML schema definition corresponding to the C++ code for a NURBS structure. Each *nurbs* element is defined in terms of a set of control points (the *controlPoints* element), a knot vector (the *knotVector* element), a weight vector (the *weightVector* element), and the NURBS order (the *order* element).

A sample code listing for the ADML schema definition for a NURBS based airfoil object follows.

```
<xs:element name="airfoilType">
 <xs:complexType>
  <xs:sequence>
   <xs:choice>
    <xs:sequence>
     <xs:element name="configParam" type="variable"/>
     <xs:element name="analysisResult" type="variable" minOccurs="0"/>
    </xs:sequence>
    <xs:sequence>
     <xs:choice>
      <xs:element name="coordinateList" type="mlist"/>
      <xs:element name="pointList" type="plist"/>
     </xs:choice>
    </xs:sequence>
    <xs:sequence>
     <xs:element name="curveTop" type="nurbs"/>
      <xs:element name="curveBot" type="nurbs"/>
    </xs:sequence>
    <xs:element name="shape" type="xs:string"/>
   </xs:choice>
   <xs:element name="provenance" type="provenanceType"/>
  </xs:sequence>
  <xs:attribute name="uID" type="xs:integer"/>
 </xs:complexType>
</xs:element>
```

The geometry model for the airfoil shown in Figure 6, defined by two NURBS curves (top and bottom) with four control points associated with four weights each, follows.

```
<airfoilType>
 <curveTop ncp="4">
  <controlPoints format="Mathematica">
   <definition> { { 0,0,0 }, { 0,0.020,0 }, { 0.25,0.12,0 }, { 1,0,0 } } </definition>
  </controlPoints>
  <knotVector format="Mathematica">
   <definition> { 0,0,0,0,1,1,1,1 } </definition>
  </knotVector>
  <weightVector format="Mathematica">
   <definition> { 1,1,1,1} </definition>
  </weightVector>
  <order>4</order>
 </curveTop>
 <curveBot ncp="4">
  <controlPoints format="Mathematica">
   <definition> { { 0,0,0 }, { 0,-0.005,0 }, { 0.25,-0.04,0 }, { 1,0,0 } } </definition>
  </controlPoints>
  <knotVector format="Mathematica">
   <definition> { 0,0,0,0,1,1,1,1 } </definition>
  </knotVector>
  <weightVector format="Mathematica">
   <definition> { 1,1,1,1 } </definition>>
  </weightVector>
  <order>4</order>
 </curveBot>
</airfoilBase>
```

4.2. Encoding Convair B58

The Convair B58 Hustler is used as a proof of concept for encoding an entire aircraft in ADML. The reason for using the B58 as the testbed is that most of the data for the B58 is public domain. Also the ADML schema development is in direct response to CCMS needs and the B58 aircraft has been used as a benchmark for design projects in CCMS. The Convair B58 has a delta wing and a vertical tail with four General Electric J79 engines in pods under the wing. The ADML encoding for the B58 is about 700 lines (about 22KB), and uses 111 elements from the total number of 412 ADML elements. Table 3 lists the number of occurrences for those 111 elements that are used for

Table 1
B58 Element Statistics

Element Name	Count
ADML, aircraft, model, airfoils, airfoil, shape, wings, fuselages, fuselage, length, width, hTop, hBot, X0, kLoc, kMag, kWidth, propulsion, engine, weight, global, machNumber, altitude, desiredMeshSize, cowl, N1, N2, topWidth, botWidth, height, length, x0, y0, Nx, Ny, LEAmplifier, thicknessAmplifier, TEAmplifier, ramp, length, width, topHeight, botHeight, x0, y0, kLoc, kMag, materials	1
wing, planform, structure, sections, numRibs, numSpars, semiB, cRoot, cTip, controlEffectors, controlEffector, IBEta, OBEta, chord, Nx, Ny, N1, N2, TEBreak, LEBreak, lambdaLE, lambdaTE, Bu, Bl, shear1, shear2, twist0, twist1, twist2, theta, beta, flapin, flapout, flapchord, inflapdef, outflapdef, orthoTropicMaterial, E1, E2, NU12, G12, RHO, Xt, Xc, Yt, Yc, S, LMType, LThicklb, LThickub	2
provenance	3
t	4
section, crossSection, sparElementIDs, ribElementID, trueRibIDs, ghostRibIDs, airfoilID	18
name	22
v	34
sval	77
description	108

encoding the B58. Owing to the complexity of the ADML schema and the large number of ADML elements required to represent the entire B58 aircraft, it is not feasible to list and discuss each and every element in this paper. Instead, a detailed description of just one element, the *wing* element for the B58, follows (the full ADML encoding of the B58 is in the supporting files for this paper).

The wing geometry for the B58 is in VT-CST format. Each *wing* element consists of four subelements, *planform*, *structure*, *controlEffectors*, and *compositeWingBoxes*. The planform is defined by the root chord (*cRoot*), the tip chord (*cTip*), the half span of the wing (*semiB*), standard vectors of leading edge and trailing edge sweep angles (*LESweep* and *TESweep*), and vectors of the nondimensional leading edge and trailing edge break locations (*LEBreak* and *TEBreak*). In addition to these parameters, a *planform* also consists of definitions for the lower and upper surface amplifiers (discussed in great detail in [17]) and a set of corresponding design variables (e.g., N_x and N_y defining the order of the Bernstein polynomials in x and y, respectively, etc.). By storing the leading and trailing edge sweep angles and break locations in a vector, planforms ranging from very simple delta wings to complex wings with multiple breaks and sweeps can be defined. All of the vectors are defined using the *vtype* element from the data schema, and all of the tensors are defined using the *tensor* element from the data schema.

The airfoil cross section for the B58 aircraft, NACA 0003.46-64.069 for the root and NACA 0004.08-63 for the tip chord, is defined as a reference to the *airfoilType* defined outside the *wing* element. The wing *structure* is defined as a set of wing *sections*. Each *section* consists of a unique identifier *uID*, a *name*, a *description*, a reference to a *crossSections*, and definitions for materials and thicknesses for ribs, spars, and skins of a wing.

The *controlEffectors* are defined using VT-CST geometry for control surfaces, a set of parameters for describing a hinge, and the *provenance* element. The *compositeWingBoxes* element defines one or more *compositeWingBox*es, each defined using a set of parameters describing the orientations and the core and layer thicknesses for the ribs, spars, and skins.

A snippet (a subset of the parameters in *planform*, one of the nine *sections* of the main wing of the B58, and the *controlEffectors*) of the ADML wing encoding for the B58 aircraft follows, omitting the schemata defining the tags.

```
<wings>
 <wing uID="1" type1="main" type2="horizontal">
  <planform>
   <semiB>
    <description>(semi span, double)</description>
    <sval>28.4</sval>
   </semiB>
   <cRoot>
    <description>(root chord, double)</description>
    <sval>54.3</sval>
   </cRoot>
   <cTip>
    <description>(tip chord, double)</description>
    <sval>0.01</sval>
   </cTip>
   ...
   <TEBreak>
    <description>(span TE break locations, vector double)</description>
    <sval>n</sval>
   </TEBreak>
   ...
   <Bu>
    <description>(upper surface amp, vector of vector of double)</description>
    <t>
     <v> 1.0 0.0599672208644 0.0504355463712 0.0577872061211 0.0454111489566 1.0 1.0 </v>
     <v> 1.0 0.0680553345373 0.0462947699499 0.0353887592720 0.0562369691330 1.0 1.0 </v>
     <v> 1.0 0.0700420166919 0.00664473683005 0.0507840408258 0.0554809619909 1.0 1.0 </v>
     <v> 1.0 0.0745163222007 0.0245454256447 0.0628921947699 0.0792447631647 1.0 1.0 </v>
     <v> 1.0 0.075914436711 0.0193230458869 0.0591386907617 0.0455063767249 1.0 1.0 </v>
     <v> 1.0 1.0 1.0 1.0 1.0 1.0 1.0 </v>
    </t>
   </Bu>
   ...
  </planform>
  <structure>
   <numRibs>9</numRibs>
   <numSpars>9</numSpars>
   <sections>
    <section uID="1">
     <name>sec1_main_wing</name>
     <description>section 1 of main wing</description>
     <crossSection uID="1">
      <airfoilID>1</airfoilID>
     </crossSection>
     <sparElementIDs>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
     1 1 1 1 1 1 1 1 1 1 1 1 1</sparElementIDs>
     <ribElementID>2</ribElementID>
     <trueRibIDs>1 4</trueRibIDs>
     <ghostRibIDs>2 3</trueRibIDs>
    </section>
    ...
   </sections>
  </structure>
  <controlEffectors>
   <controlEffector>
    <IBEta>0.1656</IBEta>
    <OBEta>0.6933</OBEta>
    <chord>6.971</chord>
   </controlEffector>
  </controlEffectors>
 </wing>
<wings>
```

4.3. Comparison with CPACS

A careful review of CPACS XML schema suggests that the CPACS schema development follows a top-down approach with the most detail at a high level of aircraft design constructs (see Figure 7).

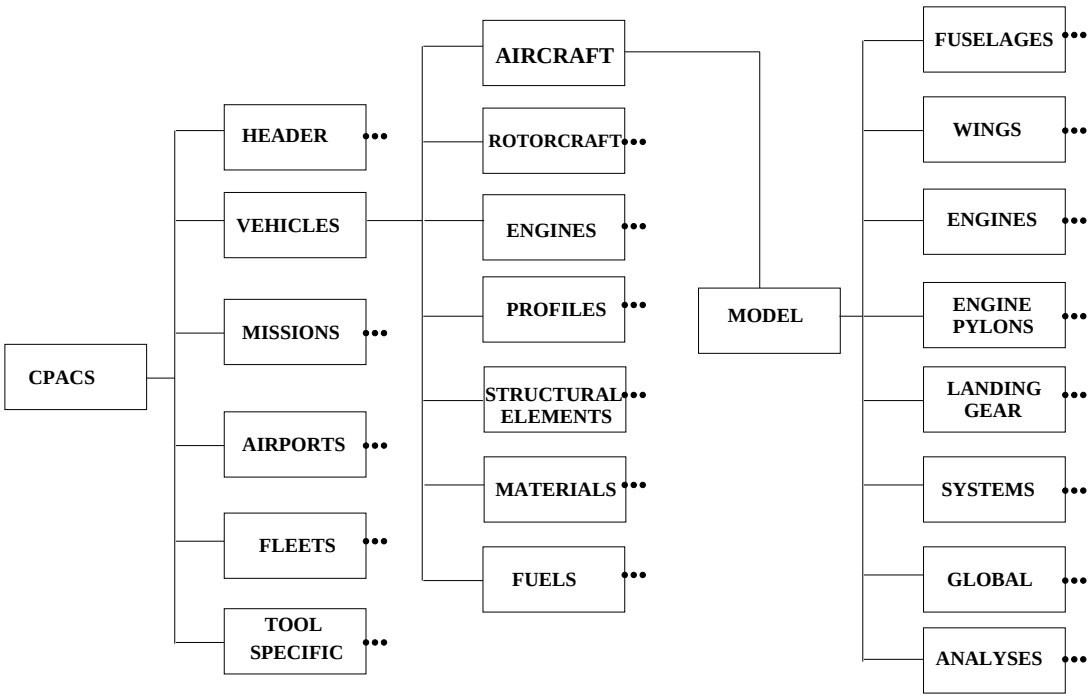

Figure 7. CPACS taxonomy.

On the other hand, ADML follows a bottom-up approach whereby emphasis is given to low level components (raw data, mathematical functions, and basic geometry) that makes it very efficient and expressive. CPACS does not yet support certain detailed geometry, e.g., parametric or NURBS based airfoil design, and the CPACS development team is planning to address those in the next version. In order to support the CST ([15]) based parametric geometry in CPACS, the generated geometry associated with the set of design variables is converted to CPACS format by a special initializer routine. ADML, on the other hand, can *directly* represent the CST input (in VT-CST format) in XML format.

ADML has a very sophisticated way of encoding these constructs, using its low level elements (e.g., ADML has an element called "nurbs", and can directly encode arbitrary parametric functions), and the current version of ADML enables a complete representation of an aircraft. The idea behind having detailed low level schemata for ADML is that once a strong foundation is in place with all common, generic, reusable elements at a low level, one can easily build upon those all other high level aircraft design components with the flexibility to accommodate several different configurations. Careful thought has been given to a format for representing mathematical objects while developing the proposed ADML schema. A major difference between CPACS and ADML, as mentioned earlier, is the representation of low level data elements. CPACS does not have any provision in the current version for representing a mathematical function in analytical form. All the *profile* elements (e.g., fuselage and wing cross sections) in CPACS are defined as *pointList*s. Earlier versions of CPACS defined each *pointList* as a sequence of three XML elements defining three coordinate axes. This verbose definition of a list data type incurs significant overhead in terms of the storage of the XML documents. The *pointList* definition has been modified in the current version of CPACS so that a list of points along a coordinate axis is represented as a vector. Another peculiarity of CPACS is that both the vector and the array data types are defined as XML strings (an array is a flattened list), and there is no data structure in CPACS to handle multidimensional arrays. Reshaping a matrix from a string significantly increases the cost for parsing the CPACS data.

CPACS has been adopted as a data standard for exchanging aircraft design and analysis data in several DLR projects and integrated environments, and a number of tools (e.g., TXL—a geometry engine) have been developed for automating the multidisciplinary aircraft design process. ADML is still in an early stage of development, and has to evolve further to accommodate a wide variety of aircraft configurations (CPACS has about two thousand five hundred elements whereas ADML has about four hundred elements) The immediate goal is the adoption of ADML within the CCMS,

and ultimately within a large segment of the aircraft design community. Another possibility is to merge CPACS and ADML to achieve the benefits of both. This could be achieved by using the existing import facility available in the XML Schema definition. The XML Schema *import* element facilitates adding multiple schemata with different target namespaces to an existing schema. That way CPACS schema could be imported into the ADML schema, and all the elements in CPACS could be accessed through ADML without any difficulty.

5. Conclusion and Future Work

An XML schema based generic, comprehensive, and compact aircraft design markup language (ADML) is proposed to represent aircraft design models (geometry, structures, propulsion, etc.) and analysis data (raw data and mathematical objects). ADML addresses data exchange and interoperability issues involved in a multidisciplinary, collaborative, conceptual aircraft design environment by providing a common language for data communication. The XML schema discussed in this paper follows a modular schema development and takes a bottom up approach by starting the schema development from the simplest form of data and building on that more complex constructs in a conceptual aircraft design process. Thus, the XML elements from the data and function schema serve as the building blocks for other more complex elements. An airfoil geometry example presented in Section 4 illustrates the modeling capabilities of the proposed geometry schema, and the aircraft model represented using the Convair B58 shows the scope of the proposed schema. ADML supports both the VT-CST geometry as well as NURBS based BRep constructs. The ADML schema supports several disciplines (geometry, structures, configuration layout, propulsion, mission, performance, payload, and materials) involved in a multidisciplinary, collaborative conceptual aircraft design and analysis process where all disciplines can natively understand the ADML standard and can communicate with each other through a common language and platform neutral data format. The schema described in this paper is organized for the design and analysis of fixed wing aircraft, but it is readily extensible to flapping wing MAVs (micro air vehicles) and morphing vehicles, whose shapes change in time. ADML is still in an early stage of development, and has to evolve further to accommodate a wide variety of aircraft configurations, though ADML is complete enough to represent an entire B58 used as a conceptual design benchmark by CCMS and others.

References

[1] Aerospace and Defence Industries Associations Europe, "Standard for the Exchange of Product model data (STEP - ISO 10303) Application Protocol 209: Multidisciplinary analysis and design," 2013

[2] American Institute of Aeronautics and Astronautics: Flight dynamics model exchange standard (draft), "BSR/AIAA S-119-201X," AIAA, 2010

[3] Begley E. F. and Sturrock C. P., "MatML: XML for Material Property Data," in *ASM Internationals Advanced Materials and Processes*, available online at http:// xml.coverpages.org / begley-ampmatml.pdf, 2000

[4] Berndt J. S., "JSBSim, an open source platform independent flight dynamics model in C++," JSBSim Reference Manual v1.0., available online at http://jsbsim.sourceforge.net/ JSBSimReferenceManual.pdf, 2011

[5] Bhandarkar M. P. and Nagi R., "STEP-based feature extraction from STEP geometry for Agile Manufacturing," in *Computers in Industry*, vol. 41, pp. 3-24, 2000

[6] Blair M., "Air Vehicle Environment in C++: A Computational Design Environment for Conceptual Innovations," in *Journal of Aerospace Computing, Information, and Communication*, Vol. 7, 85-117, 2010

[7] Composite Materials and Structures Group, "FemML for Data Exchange between FEA Codes," in *ANSYS Users group conference, Univ. of Maryland, College Park*, available online at http: // femml.sourceforge.net /, Oct. 2001

[8] Deshpande S. G., Watson L. T., Canfield R.. A., Blair M., and Beran P. S. , "XML Schema for Aircraft Conceptual Model Representation," in *International Conference on Information and Knowledge Engineering*, Las Vegas, Nevada, 2011

[9] Gopalsamy S. and Yu T., "A Geometry Engine for CAD/GRID Integration," in *AIAA 2003-800, 41st Aerospace Sciences Meeting and Exhibit*, Reno, Nevada, Jan. 2003

[10] Haimes R. and Dannenhoffer J. F., "Control of Boundary Representation Topology in Multidisciplinary Analysis and Design," in *AIAA 2010-1504, 48th AIAA Aerospace Sciences Meeting*, Orlando, Florida, Jan. 2010

[11] Hucka M. et al., "The systems biology markup language (SBML): A medium for representation and exchange of biochemical network models," Bioinformatics
Vol. 19 (4), pp. 524–531, 2003

[12] Information Processing – Text and Office Systems, *Standard Generalized Markup Language (SGML)*, ISO 8879:1986

[13] Initial Graphics Exchange Specifications, "A Century of Excellence in Measurements, Standards, and Technology—A Chronicle of Selected NBS/NIST Publications, 1901 - 2000, David L. Lide, Editor," NIST Special Publication 958, Jan. 2001

[14] Krishnan R., "Evaluation of Frameworks for HSCT Design and Optimization ," NASA/CR-1998-208731, Oct. 1998

[15] Kulfan, B. M., "Universal Parametric Geometry Representation Method," in *Journal of Aircraft, Vol. 45, No. 1, pp. 142158*, Jan-Feb 2008

[16] Lin R. and Afjeh A., "An extensible, interchangeable, and sharable database model for improving multidisciplinary aircraft design," in *AIAA 2002-5613*, Atlanta, GA, 2002

[17] Morris C. C., Allison D. L., Schetz J. A., and Kapania R. K., "Parametric geometry model for multidisciplinary design optimization of tailless supersonic aircraft," in *in proc. of AIAA Modeling and Simulation Technologies Conference*, Minneapolis, Minnesota, Aug. 2012

[18] Nagel R. N., Braithwaite W. W., and Kennicott P. R., "Initial Graphics Exchange Specification IGES, Version 1.0," Washington DC: National Bureau of Standards, NBSIR 80-1978, 1980

[19] Peak R., Lubell J., Srinivasan V., and Waterbury S. C., "STEP, XML, and UML: Complementary Technologies," in *ASME, Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Salt Lake City, USA,2004

[20] Pratt M., "Extension of ISO 10303: The Step Standard, for the Exchange of Procedural Shape Models," in *Proc. Int'l Conf Shape Modeling and Applications (SMI)* , 2004

[21] Rappoport A., "An Architecture for Universal CAD Data Exchange," in *Proceedings, Solid Modeling 03*, Seattle, WA, SCM Press, Jun 2003

[22] Raymer D. P., *Aircraft Design: A Conceptual Approach*, AIAA Education Series, New York, NY, 2006

[23] Rizzi A., Oppelstrup J., Zhang M., and Tomac M., "Coupling Parametric Aircraft Lofting to CFD and CSM Grid Generation for Conceptual Design," in *AIAA 2011-160, 49th AIAA Aerospace Sciences Meeting*, Orlando, Florida, Jan 2011

[24] Roth G. L., Livingston J. W., Blair M., and Kolonay R., "CREATE-AV DaVinci: Computationally based engineering for conceptual design," in *AIAA 2010-1232*, Orlando, FL, Jan 2010

[25] Salas, A. O., and Townsend, J. C., "Framework Requirements for MDO Application Development," in *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, AIAA Paper 98-4740, Sept. 1998

[26] Sandhu R., *The MathML Handbook*, Charles River Media, Edition 1, 2002

[27] Schenck D. and Wilson P., *Information Modeling the EXPRESS Way*, Oxford University Press,1994

[28] World Wide Web Consortium, *XML Schema Part 0: Primer*, online at http://www.w3.org/ TR / xmlschema-0, , May 2001

[29] World Wide Web Consortium, *Extensible Markup Language (XML) 1.0, 3rd ed.*, available online at http://www.w3.org/TR/2004/REC-xml-20040204/, Feb. 2004

Appendix A: ADML Schema

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="ADML">
 <xs:complexType><xs:sequence>
  <xs:element ref="data"/>
  <xs:element ref="functions"/>
  <xs:element ref="geometry"/>
  <xs:element ref="aircraft"/>
 </xs:sequence></xs:complexType>
</xs:element>
<xs:element name="data">
 <xs:complexType><xs:sequence>
  <xs:element ref="variable"/>
  <xs:element ref="flag"/>
  <xs:element ref="scalar"/>
  <xs:element ref="tensor"/>
 </xs:sequence></xs:complexType>
</xs:element>
<xs:complexType name="nd"><xs:sequence>
 <xs:element name="name" type="xs:string" minOccurs="0"/>
 <xs:element name="description" type="xs:string" minOccurs="0"/>
 <xs:element name="unit" type="xs:string" minOccurs="0"/>
</xs:sequence></xs:complexType>
<xs:simpleType name="scalarType">
 <xs:union memberTypes="xs:integer xs:double xs:string xs:boolean"/>
</xs:simpleType>
<xs:element name="min" type="scalarType"/>
<xs:element name="max" type="scalarType"/>
<xs:simpleType name="flagType">
 <xs:union memberTypes="xs:integer xs:boolean"/>
</xs:simpleType>
<xs:element name="flag">
 <xs:complexType><xs:complexContent>
  <xs:extension base="nd"><xs:sequence>
   <xs:element name="flagval" type="flagType"/>
  </xs:sequence></xs:extension>
 </xs:complexContent></xs:complexType>
</xs:element>
<xs:simpleType name="vtype">
 <xs:list itemType="scalarType"/>
</xs:simpleType>
<xs:complexType name="tensorType"><xs:choice>
 <xs:sequence minOccurs="2" maxOccurs="unbounded">
  <xs:element name="t" type="tensorType"/>
 </xs:sequence>
 <xs:sequence minOccurs="2" maxOccurs="unbounded">
  <xs:element name="v" type="vtype"/>
 </xs:sequence></xs:choice>
</xs:complexType>
<xs:element name="value" type="valueType"/>
<xs:complexType name="valueType">
 <xs:choice>
  <xs:element ref="scalar"/>
  <xs:element ref="tensor"/>
 </xs:choice></xs:complexType>
 <xs:element name="scalar">
 <xs:complexType><xs:complexContent>
  <xs:extension base="nd"><xs:sequence>
   <xs:element name="sval" type="scalarType"/>
  </xs:sequence></xs:extension>
 </xs:complexContent></xs:complexType>
</xs:element>
<xs:element name="tensor">
 <xs:complexType><xs:complexContent>
  <xs:extension base="nd">
   <xs:sequence>
    <xs:element name="t" type="tensorType"/>
   </xs:sequence>
  </xs:extension></xs:complexContent>
 </xs:complexType></xs:element>
 <xs:element name="variable">
 <xs:complexType><xs:complexContent>
  <xs:extension base="nd">
   <xs:sequence>
    <xs:element ref="value"/>
    <xs:element ref="min" minOccurs="0"/>
    <xs:element ref="max" minOccurs="0"/>
    <xs:element ref="flag" minOccurs="0"/>
    <xs:element ref="scalar" minOccurs="0"/>
   </xs:sequence>
   <xs:attribute name="vid" type="xs:ID"/>
  </xs:extension></xs:complexContent>
 </xs:complexType></xs:element>
 <xs:element name="functions">
```

```
 <xs:complexType><xs:sequence>
  <xs:element ref="operator"/>
  <xs:element ref="expression"/>
  <xs:element ref="relation"/>
 </xs:sequence></xs:complexType>
</xs:element>
<xs:element name="arguments" type="xs:string"/>
<xs:element name="variables" type="xs:string"/>
<xs:element name="listRef" type="xs:IDREF"/>
<xs:element name="expressionRef" type="xs:IDREF"/>
<xs:element name="definition" type="xs:string"/>
<xs:simpleType name="formatType">
 <xs:restriction base="xs:string">
  <xs:enumeration value="Matlab"/>
  <xs:enumeration value="Mathematica"/>
 </xs:restriction></xs:simpleType>
 <xs:attribute name="format" type="formatType"/>
 <xs:attribute name="structure" type="structureType"/>
 <xs:simpleType name="structuretype">
 <xs:restriction base="xs:string">
  <xs:enumeration value="array"/>
  <xs:enumeration value="general"/>
 </xs:restriction></xs:simpleType>
 <xs:element name="operator">
 <xs:complexType><xs:complexContent>
  <xs:extension base="nd"><xs:sequence>
   <xs:element name="domain" type="xs:string"/>
   <xs:element name="range" type="xs:string"/>
   <xs:element ref="arguments"/>
   <xs:element ref="definition"/>
  </xs:sequence>
  <xs:attribute ref="format"/>
  <xs:attribute name="opid" type="xs:ID"/>
 </xs:extension></xs:complexContent>
</xs:complexType></xs:element>
<xs:element name="expression">
<xs:complexType><xs:complexContent>
 <xs:extension base="nd"><xs:sequence>
  <xs:element ref="variables"/>
  <xs:element ref="definition"/>
  <xs:element name="patternStr" type="xs:string" minOccurs="0"/>
 </xs:sequence>
 <xs:attribute ref="format"/>
  <xs:attribute name="eid" type="xs:ID"/>
 </xs:extension></xs:complexContent>
</xs:complexType></xs:element>
<xs:complexType name="mlist">
<xs:complexContent>
 <xs:extension base="nd">
  <xs:sequence>
   <xs:element ref="variables"/>
   <xs:element ref="definition"/>
  </xs:sequence>
  <xs:attribute name="lid" type="xs:ID"/>
  <xs:attribute ref="format"/>
  <xs:attribute ref="structure"/>
  <xs:attribute name="dimension" type="xs:string"/>
 </xs:extension></xs:complexContent>
</xs:complexType>
<xs:element name="relation">
 <xs:complexType><xs:complexContent>
  <xs:extension base="nd">
   <xs:sequence><xs:choice>
    <xs:element ref="expressionRef"/>
    <xs:element ref="listRef"/>
   </xs:choice></xs:sequence>
   <xs:attribute ref="format"/>
  </xs:extension></xs:complexContent>
 </xs:complexType></xs:element>
 <xs:complexType name="geometry">
 <xs:complexContent>
  <xs:extension base="complexBaseType">
   <xs:sequence>
    <xs:element name="point" type="point"/>
    <xs:element name="plist" type="pList"/>
    <xs:element name="line" type="line"/>
    <xs:element name="plane" type="plane"/>
    <xs:element name="nurbs" type="nurbs"/>
   </xs:sequence></xs:extension>
 </xs:complexContent></xs:complexType>
 <xs:simpleType name="point">
 <xs:list itemType="xs:double"/>
</xs:simpleType>
<xs:element name="pList">
 <xs:complexType>
  <xs:sequence maxOccurs="unbounded">
```

```
  <xs:element name="p" type="point"/>
 </xs:sequence></xs:complexType>
</xs:element>
<xs:complexType name="line">
 <xs:sequence>
  <xs:element name="point1" type="point"/>
  <xs:element name="point2" type="point"/>
 </xs:sequence></xs:complexType>
 <xs:complexType name="plane">
 <xs:sequence>
  <xs:element name="P0" type="point"/>
  <xs:element name="normal" type="point"/>
 </xs:sequence></xs:complexType>
 <xs:element name="frame">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="origin" type="point"/>
   <xs:element name="angles" type="xs:string"/>
  </xs:sequence>
  <xs:attribute name="frameID" type="xs:ID"/>
  <xs:attribute name="parentID" type="xs:IDREF"/>
 </xs:complexType></xs:element>
 <xs:simpleType name="nurbsType">
 <xs:restriction base="xs:string">
  <xs:enumeration value="curve"/>
  <xs:enumeration value="surface"/>
 </xs:restriction>
</xs:simpleType>
<xs:complexType name="nurbs">
 <xs:complexContent>
  <xs:extension base="nd">
   <xs:sequence>
    <xs:element name="controlPoints" type="mlist"/>
    <xs:element name="knotVector" type="mlist"/>
    <xs:element name="weightVector" type="mlist"/>
    <xs:element name="order" type="xs:integer"/>
   </xs:sequence>
   <xs:attribute name="nurbsID" type="xs:ID"/>
   <xs:attribute name="ntype" type="nurbsType" minOccurs="0"/>
   <xs:attribute name="ncp" type="xs:integer" minOccurs="0"/>
  </xs:extension></xs:complexContent>
</xs:complexType>
</xs:schema>
```