DAVE-ML: The Proposed Implementation of the Simulation Model Exchange Standard

August 2005

Bruce Jackson
NASA Langley Research Center
Hampton, Virginia

Outline

- Motivation
- Why XML?
- DAVE-ML workshop
  - Features
  - Limitations
  - Examples
  - Demos
  - Resources
  - Grammar

Motivation

- Growth of teaming within aero industry
- Exponential increase in desktop compute power
- Increasing reliance on sim-based acquistion
- Emergence of web-based everything

- Medieval simulation technology hasn't kept pace

Why XML?

- Open standard (W3C)
- Transportable
- Human- and machine-readable UNICODE
- Programming language agnostic
- Same source used for detail design & pilot training
- Good fit for databases and models
- Archivable history of data with sources cited
- Transformable (->HTML, etc.)
- Lots of generic XML tools, editors, parsers
- XSLT

DAVE-ML features

- Non-linear function descriptions
  - Tabular (traditional aero models)
    - Unlimited dimensionality
    - Orthogonal or unstructured
  - Polynomial or exponential expressions
- Model build-up equations
- Units of measure
- Standard variable names
- Uncertainty bounds
- Provenance (history/source of data)
- Checkcase/verification data included

Current DAVE-ML limitations (opportunities!)

- Static models (dynamics not yet supported)
- Submodels not yet defined
- No native DAVE-ML editor
  - Implies hand-editing with generic text or XML editor

Existing DAVE-ML examples

Fighter subsonic aero model

- 51 variables, 18 tables, 744 points
- Switches & absolute value nonlinear elements
- 17 verification checkcases included
- 154 KB file with 2,712 lines

Concept development lifting body aero model

- Supersonic and subsonic regimes
- 361 variables, 168 tables, 6,240 points
- 24 verification checkcases included
- 1.2 MB file with 22,299 lines

Existing (known) DAVE-ML resources

- DAVE-ML Reference manual (website)
- JANUS (C++ library)
  - Australian DSTO/Ball Aerospace
- NASA Ames FTP tool (import/export Perl scripts)
- NAVAIR's CASTLE sim shell (latest release)
- XSLT conversion script
  DAVE-ML => XHTML
- DAVEtools (Java packages):
  DAVE-ML => Simulink

XSLT conversion script

- eXtensible Stylesheet Language Transformation
- XSLT scripts, themselves XML documents, specify how to convert XML-based files into other formats.
- 'DAVE_html.xsl' details a conversion from DAVE-ML XML file into an XHTML document
- Requires 'xerces' or similar XSLT tool
- Available for download =>
  http://daveml.nasa.gov

DAVEtools 0.53

- Two Java 1.4 packages
  - Base package, "DAVE"
    - Builds internal objects from DAVE-ML description
    - Performs auto-verification of model
    - Allows simple exercising of model
  - Simulink® export tool, "DAVE2SL" (built on DAVE)
    - Converts DAVE-ML models into Simulink blocks
- Java 1.4 source code available
  - Request public domain license via http://daveml.nasa.gov

Janus API Library

- Geoff Brian slides

Quick demonstrations

- Generate HTML
- Generate Simulink® subsystem
- Exercise command line DAVEtool

DAVE-ML grammar

- Top-level syntax
- Specified by DAVEfunc.dtd Document Type Def
- Examples taken from reference manual for 1.7
- Reference manual available for download from http://daveml.nasa.gov - see "DTDs"

Top-level syntax

- Top level-element is <DAVEfunc>

```
DAVEfunc :
        fileHeader :
        variableDef+ :
        breakpointDef* :
        griddedTableDef* :
        ungriddedTableDef* :
        function* :
        checkData? :
```

Key: *element : attribute [optional attribute]*
subelement
'+' means 1 or more; '' means 0 or more; '?' means 0 or 1*

File header element

- Provides documentation trail for model

```
fileHeader : [name]
        author : name, org
        fileCreationDate : date
        fileVersion? :
        description? :
        reference* :
        modificationRecord* :
        provenance* :
```

Key: *element : attribute [optional attribute]*
subelement
'+' means 1 or more; '' means 0 or more; '?' means 0 or 1*

Example of `fileHeader` element

Opening of `fileHeader` element

```
<!--                         =================                          -->
<!-- ============        =====   FILE HEADER   ======================== -->
<!--                         =================
```

Author information

```
<fileHeader>
    <author name="Bruce Jackson" org="NASA Langley Research Center"
    email="nospam@nasa.gov">
      <address>MS 132 NASA, Hampton, VA 23681</address>
    </author>
    <fileCreationDate date="2003-03-18"/>

    <fileVersion>$Revision: 1.24 $</fileVersion>
    <description>
      Version 2.0 aero model for HL-20 lifting body, as described in
      TM-107580. This aero model was used for HL-20 approach and
      landing studies at NASA Langley Research Center during 1989-1995
      and for a follow-on study at NASA Johnson Space Center in 1994
      and NASA Ames Research Center in 2001. This DAVE-ML version
      created 2003 by Bruce Jackson to demonstrate DAVE-ML.
    </description>

    <reference refID="REF01"
      author="Jackson, E. Bruce, Cruz, Christopher I. &amp; and Ragsdale, W. A."
      title="Real-Time Simulation Model of the HL-20 Lifting Body"
      accession="NASA TM-107580"
      date="1992-07-01"
    />
```

Model description subelement

We define `refID`' s here; used elsewhere

`fileHeader` *example, cont'd*


```
  <reference refID="REF02"
    author="Cleveland, William F
    title="Possible Typo in HL
    accession="email"
    date="2003-08-19"
  />

  <modificationRecord modID="A" refID="REF02">
    <author name="Bruce Jackson" org="NASA Langley Research Center"
                email="nospam@nasa.gov">
        <address>MS 132 NASA, Hampton, VA 23681</address>
    </author>
    <description>
        Revision 1.24: Fixed typo in CLRUD0 function description which
        gave dependent signal name as "CLRUD1." Bill Cleveland of NASA
        Ames caught this in his xml2ftp script. Also made use of 1.5b2
        fileHeader fields and changed date formats to comply with
        convention.
    </description>
  </modificationRecord>

</fileHeader>
```

Modification record defines `modID` for easy reference later, where modification has been applied

Note citation of `refID`

Ending fileHeader tag

Variable Definition element (`variableDef`)

- Used to define each variable used in the model
  - Includes inputs, outputs, constants, parameters and local (temporary) variables

```
variableDef : name, varID, units,
        [axisSystem, sign, alias, symbol, initialValue]
    description? : (description character data)
    calculation? : math (defined in MathML2.0 DTD)
    isOutput? :
    isStdAIAA? :
    uncertainty? : effect (additive | multiplicative | percentage | absolute)
        (normalPDF : numSigmas | uniformPDF : symmetric )
```

Key: element : attribute [optional attribute]
subelement
'+' means 1 or more; '' means 0 or more; '?' means 0 or 1*

Example of `variableDef` *for two input elements*

Standard Name flag

Name for documentation

Symbol for documentation

Internal name

Sign convention

Units

Note UNICODE symbol

```
<!-- ================
<!--  Input variables
<!-- ================ -->

<variableDef name="MachNumber" varID="XMACH" units="" symbol="M">
  <description>
        Mach number (dimensionless)
  </description>
  <isStdAIAA/>
</variableDef>

<variableDef name="dbfll" varID="DBFLL" units="deg" sign="ted" symbol="∂bfll">
  <description>
        Lower left body flap deflection, deg, +TED (so deflections are
        always zero or positive).
  </description>
</variableDef>
```

Example of local variable definition

Simple definition; used later

```
<!-- PRELIMINARY BUILDUP EQUATIONS -->

<!--  LOWER LEFT BODY FLAP CONTRIBUTIONS -->

<!--    table output signal   -->
  <variableDef name="Cldbfll_0" varID="CRBFLL0" units="">
    <description>
          Output of CRBFLL0 function; rolling moment contribution of
          lower left body flap deflection due to alpha^0 (constant
          term).
    </description>
  </variableDef>
```

A more complex variable definition


```
<!--    lower left body flap lift buildup -->
  <variableDef name="CLdbfll" varID="CLBFLL" units="">
    <description>
          Lift contribution of lower left body flap deflection
          CLdbfll = CLdbfll_0 + alpha*(CLdbfll_1 + alpha*(CLdbfll_2 + alpha*CLdbfll_3))
    </description>
    <calculation>
      <math>
        <apply>
          <plus/>
          <ci>CLBFLL0</ci>
          <apply>
            <times/>
            <ci>ALP</ci>
            <apply>
              <plus/>
              <ci>CLBFLL1</ci>
              <apply>
                <times/>
                <ci>ALP</ci>
                <apply>
                  <plus/>
                  <ci>CLBFLL2</ci>
                  <apply>
                    <times/>
                    <ci>ALP</ci>
                    <ci>CLBFLL3</ci>
                  </apply> <!--              a*c3   -->
                </apply> <!--         (c2 + a*c3)  -->
              </apply> <!--         a*(c2 + a*c3)  -->
            </apply> <!--    (c1 + a*(c2 + a*c3)) -->
          </apply> <!--    a*(c1 + a*(c2 + a*c3)) -->
        </apply> <!-- c0 + a*(c1 + a*(c2 + a*c3)) -->
      </math>
    </calculation>
  </variableDef>
```

This MathML2 encodes:

$$C_{L\partial BFLL} = C_{L\partial BFLL0} + \alpha C_{L\partial BFLL1} + \alpha^2 C_{L\partial BFLL2} + \alpha^3 C_{L\partial BFLL3}$$

An output variable definition


```
      <!-- ================== -->
      <!--  Output variables  -->
      <!-- ================== -->

<variableDef name="CL" varID="CL" units="" sign="up" symbol="CL">
  <description>
    Coefficient of lift
    CL = CL0 + CLBFUL + CLBFUR + CLBFLL + CLBFLR +
                      CLWFL + CLWFR + CLRUD + CLGE + CLLG
  </description>
  <calculation>
    <math>
      <apply>
        <plus/>
        <ci>CL0</ci>
        <ci>CLBFUL</ci>
        <ci>CLBFUR</ci>
        <ci>CLBFLL</ci>
        <ci>CLBFLR</ci>
        <ci>CLWFL</ci>
        <ci>CLWFR</ci>
        <ci>CLRUD</ci>
        <ci>CLGE</ci>
        <ci>CLLG</ci>
      </apply>
    </math>
  </calculation>
  <isOutput/>
</variableDef>
```

Normal `variableDef` preamble

Sum of inputs

Output flag

Breakpoint set definition element (`breakpointDef`)

- Used to define a common set of independent variable values associated with one or more tables of function data

```
breakpointDef : bpID, [name, units]
        description? : (description character data)
        bpVals : (list of comma-separated set of breakpoint values)
```

Key: *element : attribute [optional attribute]*
subelement
'+' means 1 or more; '' means 0 or more; '?' means 0 or 1*

Two examples of `breakpointDef` *elements*

Breakpoint values

Breakpoint ID

```
<!--                    ====================
    ===   BREAKPOINT SETS   ======
                        ====================

<breakpointDef na    Mach" bpID="XMACH1_PTS" units="">
  <description>
    Mach number brea  oints for all aero data tables
  </description>
  <bpVals>
     0.3, 0.6, 0.8, 0.9, 0.95, 1.1, 1.2, 1.6, 2.0, 2.5,
  </bpVals>
</breakpointDef>
```

Breakpoint ID

```
<breakpointDef name="Lower body flap" bpID="DBFL_PTS" units="deg">
  <description>Lower body flap deflections breakpoints for tables</description>
  <bpVals>0., 15., 30., 45., 60.</bpVals>
</breakpointDef>
```

Breakpoint values

Gridded function table definition

- Defines nonlinear function dependent values

```
griddedTableDef : [gtID, name, units]
    description? : (description character data)
    provenance? :
        author : name, org, [xns, email]
            address? : (address character data)
        functionCreationDate
        documentRef* : docID
        modificationRef* : modID
    breakpointRefs :
        bpRef+ : bpID
    uncertainty? : effect (additive | multiplicative | percentage | absolute)
        (normalPDF : numSigmas | uniformPDF : symmetric )
    dataTable : (list of comma-separated set of table values)
```

Key: element : attribute [optional attribute]
subelement
'+' means 1 or more; '' means 0 or more; '?' means 0 or 1*

Example of `griddedTableDef`

Header information

```
<griddedTableDef name="CLBFL0" gtID="CLBFL0_table">
  <description>
    Lower body flap contribution to lift coefficient, polynomial constant term
  </description>
  <provenance>
    <author name="Bruce Jackson" org="NASA Langley Research Center"/>
    <functionCreationDate date="2003-01-31"/>
    <documentRef docID="REF01"/>
  </provenance>
  <breakpointRefs>
    <bpRef bpID="DBFL_PTS"/>
    <bpRef bpID="XMACH1_PTS"/>
  </breakpointRefs>
  <dataTable> <!-- last breakpoint changes most rapidly -->
    <!--  CLBFL0    POINTS    -->
    <!-- DBFL =        0.0    -->
 0.00000E+00 , 0.00000E+00 , 0.00000E+00 , 0.00000E+00 , 0.00000E+00 ,
 0.00000E+00 , 0.00000E+00 , 0.00000E+00 , 0.00000E+00 , 0.00000E+00 ,
 0.00000E+00 , 0.00000E+00 , 0.00000E+00 ,
    <!-- DBFL =       15.0    -->
-0.86429E-02 ,-0.10256E-01 ,-0.11189E-01 ,-0.12121E-01 ,-0.13520E-01 ,
-0.86299E-02 ,-0.53679E-02 , 0.76757E-02 , 0.11300E-01 , 0.62992E-02 ,
 0.51902E-02 , 0.38813E-02 , 0.37366E-02 , etc.
    </dataTable>
</griddedTableDef>
```

Breakpoint set references

Data set values

An aside about table ordering…

- The appearance of tables in DAVE-ML files can be confusing, in that they appear to be 2-dimensional
- In actuality, the values are a vector of the 'unraveled' n-dimensional tables with last index changing fastest
- Example: given $C_L = C_L(\alpha, M, \delta f)$, value sequence is

  `<datatable>`

  $C_L(\alpha_1, M_1, \delta f_1)$, $C_L(\alpha_1, M_1, \delta f_2)$, … $C_L(\alpha_1, M_1, \delta f_n)$,
  $C_L(\alpha_1, M_2, \delta f_1)$, $C_L(\alpha_1, M_2, \delta f_2)$, … $C_L(\alpha_1, M_2, \delta f_n)$,
  $C_L(\alpha_1, M_m, \delta f_1)$, $C_L(\alpha_1, M_m, \delta f_2)$, … $C_L(\alpha_1, M_m, \delta f_n)$,
  $C_L(\alpha_2, M_1, \delta f_1)$, …

  `</datatable>`

- Line breaks can occur anywhere - they are unimportant

Aside - table ordering (cont'd)

- Previous example could also be encoded

<datatable>

$C_L(\alpha_1, M_1, \delta f_1)$, $C_L(\alpha_1, M_1, \delta f_2)$, …

$C_L(\alpha_1, M_1, \delta f_n)$, $C_L(\alpha_1, M_2, \delta f_1)$,

$C_L(\alpha_1, M_2, \delta f_2)$, … $C_L(\alpha_1, M_2, \delta f_n)$,
$C_L(\alpha_1, M_m, \delta f_1)$,

$C_L(\alpha_1, M_m, \delta f_2)$, … $C_L(\alpha_1, M_m, \delta f_n)$,
$C_L(\alpha_2, M_1, \delta f_1)$, …

</datatable>

Aside - table ordering (concluded)

- Order of `<bpRefs>` in `<breakpointRefs>` element is important!
- In previous example, order would have to be

```
<breakpointRefs>
      <bpRef bpID="ALPHA_PTS"/>
      <bpRef bpID="MACH_PTS"/>
      <bpRef bpID="DELTA_FLAP_PTS"/>
</breakpointRefs>
```

…so interpreter will know which breakpoint set to associate with which dimension of this function table.

- This is one of the few places in DAVE-ML where order is important.

Function definition element

- This element ties breakpoint sets to tables
- Several different syntaxes; below is most common

```
function : name
    description? : (description character data)
    provenance? :
    independentVarRef+ : varID, [min, max, extrapolate]
    dependentVarRef : varID
    functionDefn : [name]
        griddedTableRef : gtID
```

Key: element : attribute [optional attribute]
subelement
'+' means 1 or more; '' means 0 or more; '?' means 0 or 1*

Example of `function` *element*


```
    <!-- ============================== -->
    <!-- Lower left body flap functions -->
    <!-- ============================== -->

<function name="CLBFLL0">
  <description>
    Lower left body flap lookup function for lift, polynomial constant term.
  </description>
  <independentVarRef varID="DBFLL"  min="0.0" max="60." extrapolate="neither"/>
  <independentVarRef varID="XMACH" min="0.3" max="4.0" extrapolate="neither"/>
  <dependentVarRef varID="CLBFLL0"/>
  <functionDefn name="CLBFL0_fn">
    <griddedTableRef gtID="CLBFL0_table"/>
  </functionDefn>
</function>
```

We define a function `CLBFLO_fn` to be

`CLBFLO_fn = CLBFLO_fn(DBFLL, XMACH)`

with limits on input values and no extrapolation

Verification data (`checkData`) element

Contains input/output vector pairs (time slices)

```
checkData :
    staticShot* : name [refID]
        checkInputs :
            signal* :
                signalName :
                [signalUnits :]
                signalValue :
        checkOutputs :
            signal* :
                signalName :
                [signalUnits :]
                signalValue :
                tol :
```

Key: element : attribute [optional attribute]
subelement
'+' means 1 or more; '' means 0 or more; '?' means 0 or 1*

Example checkData dump

Checkcase identifier

Checkcase input vector

name

units

value

```
<checkData>
  <staticShot name="Nominal" refID="NOTE1">
    <checkInputs>
      <signal>
        <signalName>True_Airspeed_f_p_s</signalName>
        <signalUnits>ft/sec</signalUnits>
        <signalValue> 300.000</signalValue>
      </signal>
      <signal>
        <signalName>Angle_of_Attack_deg</signalName>
        <signalUnits>deg</signalUnits>
        <signalValue>   5.000</signalValue>
      </signal>
      <signal>
        <signalName>s_Body_Pitch_Rate_rad_p_s</signalName>
        <signalUnits>rad/sec</signalUnits>
        <signalValue>   0.000</signalValue>
      </signal>
      <signal>
        <signalName>delta elevator</signalName>
        <signalUnits>deg</signalUnits>
        <signalValue>   0.000</signalValue>
      </signal>
        .
        .     (other input values)
        .
    </checkInputs>
      .
      .
      .
```

Example `checkData` *element (cont'd)*

Checkcase output vector

```
      .
      .
      .
    <checkOutputs>
      <signal>
        <signalName>CX</signalName>
        <signalValue>-0.00400000000000</signalValue>
        <tol>0.000001</tol>
      </signal>
      <signal>
        <signalName>CZ</signalName>
        <signalValue>-0.41600000000000</signalValue>
        <tol>0.000001</tol>
      </signal>
      <signal>
        <signalName>CLM</signalName>
        <signalValue>-0.04660000000000</signalValue>
        <tol>0.000001</tol>
      </signal>
        .
        .     (other output values & tolerances)
        .
    </checkOutputs>
  </staticShot>
      .
      .     (other input/output vector pairs)
      .
</checkData>
```

`checkData` *elements can have intermediate values to assist debugging*


```
<checkData>
  <staticShot name="Skewed inputs">
    <checkInputs>
      .
      .    (checkcase input values)
      .
    </checkInputs>
    <internalValues>
      <signal>
        <signalID>vt</signalID>
        <signalValue>300.0</signalValue>
      </signal>
      <signal>
        <signalID>alpha</signalID>
        <signalValue>16.2</signalValue>
      </signal>
      <signal>
        <signalID>q</signalID>
        <signalValue>-0.76</signalValue>
      </signal>
          .
          .    (additional internal values; normally one for every internal variable)
          .
    </internalValues>
    <checkOutputs>
      .
      .    (checkcase output values)
      .
    </checkOutputs>
  </staticShot>
</checkData>
```

Checkcase intermediate values

`uncertainty` *Element*

(every project should, and probably does, have one)

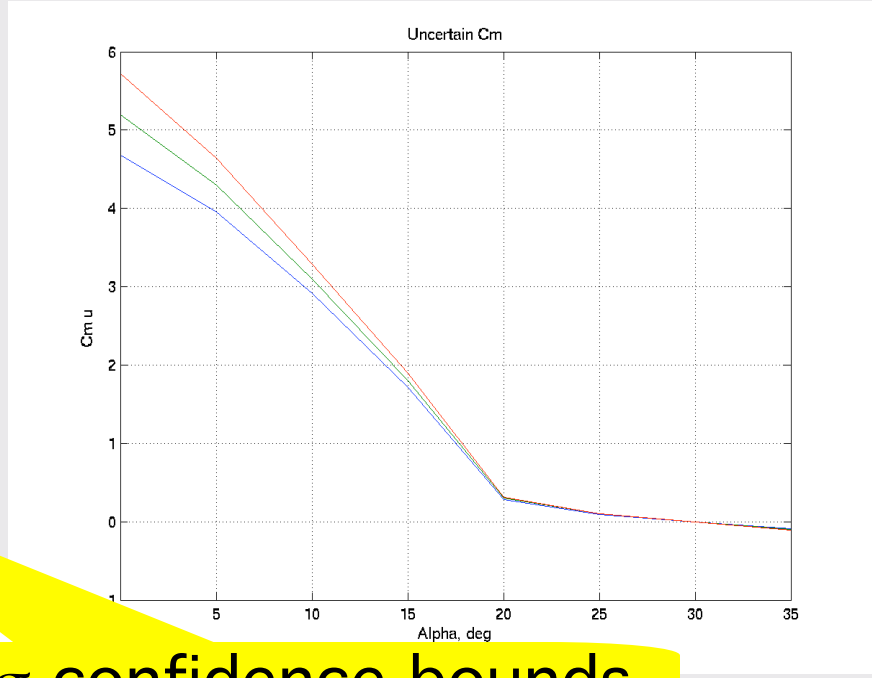
```
uncertainty : effect=['additive'|'multiplicative'|'percentage'|'absolute']
   EITHER
    normalPDF : numSigmas=['1','2','3',…]
         bounds : (value corresponding to 1, 2, 3,… σ)
   OR
    uniformPDF : symmetric=['yes'|'no']
         bounds : (symmetric or lower bound value)
         [bounds : (upper bound value)]
```

Key: *element : attribute [optional attribute]*
subelement
'+' means 1 or more; '' means 0 or more; '?' means 0 or 1*

Application of `uncertainty` *element*

Example: add 5-12% uncertainty bound around C_m

```
<function name="Uncertain Cm">
  <independentVarRef varID="Alpha_deg"/>
  <dependentVarRef varID="Cm_u"/>
  <functionDefn>
    <griddedTableDef>
      <breakpointRefs>
        <bpRef bpID="ALP"/>
      </breakpointRefs>
      <uncertainty effect="percentage">
        <normalPDF numSigmas="3">
          <bounds>
            <dataTable>
  0.10, 0.08, 0.06, 0.05, 0.05,
  0.06, 0.07, 0.12
            </dataTable></bounds>
        </normalPDF>
      </uncertainty>
      <dataTable>
  5.2, 4.3, 3.1, 1.8, 0.3, 0.1, 0.0,
      </dataTable>
    </griddedTableDe
  </functionDefn>
</function>
```



3 σ confidence bounds

(assuming ALP = [0:5:35])

Nominal function values

Summary

- DAVE-ML sufficient for some production models
  - static models only, such as aero, mass properties
- Limited tools available, but slowly growing
  - Chicken or egg problem right now
  - Hand-coding not recommended - write export script
- Syntax will evolve, in backwards-compatible way
- Still to tackle: dynamics & subsystems
- Help wanted! Join the mailing list - see website

http://daveml.nasa.gov