Roberto Parrotto – Politecnico di Milano, Italy
Johan Åkesson – Lund University & Modelon AB, Sweden
Francesco Casella – Politecnico di Milano, Italy

An XML representation of DAE systems obtained from continuous-time Modelica models

EOOLT 2010 – Oslo, 3 Oct 2010

Motivations of the work

- **Modelica** gaining popularity for system-level modelling of etherogeneous physical systems
- Current Modelica **tools** mainly focused on **simulation**
- Many other possible usages of the model
  - (Dynamic) optimization
  - Parameter identification
  - Transformations of the DAEs into specific forms for control analysis and design (e.g. LFT, linearized transfer function)
  - Model order reduction
  - Derivation of inverse kinematics and inverse dynamics controllers
  - ...
- Tools already exist to perform these activities (input data: continuous-time DAEs)

Goals of the work

- Definition of a formalism for the interfacing between **Modelica front ends** and **Equation-Based back-ends**
- Representation of continuous time DAEs models at the lower possible level
  - Scalar DAEs
  - No hierarchical aggregation/inheritance
  - No complex data structures
  - **Suppor t of functions** (widely used in Modelica)
- Easy generation from the internal AST representation of the flattened model
- Easy transformation into the input of anyback-end tool

⇨ **XML Schema / XSLT**

Why not ”Flat Modelica”?

- Modelica is meant for high-level, efficient and convenient modelling of structured systems
- Semantics far too rich for representation of plain DAEs
  - Hard to define a ”flat enough” unique subset of the language for this purpose
- Translation of flat Modelica into the input of back-end tools requires a Modelica compiler
  - Highly specialised software
  - In most cases (commercial tools) not possible to write your own extensions to the compiler
- XML parsers and XSLT tools widely available and free

⇨ Much easier to write your own back-end Interface starting from an XML representation

DAE System: set of variables

$$f(\mathrm{der}(x), x, u, w, t, p, q) = 0$$

- $\boldsymbol{x}$ vector of time-varying state variables
- $\boldsymbol{u}$ vector of time-varying input variables
- $\boldsymbol{w}$ vector of time-varying algebraic variables
- $\boldsymbol{p}$ bound time-invariant parameters
- $\boldsymbol{q}$ other unknown time-invariant parameters
- $\boldsymbol{t}$ continuous time variable

DAE System: set of equations

Dynamic equations

$$F_i(x, \mathrm{der}(x), u, w, t, p, q) = 0$$

- Every function F_i denotes a valid scalar expression
- **Residual form** *<exp1> - <exp2> = 0*
- These equations determine the values of $\boldsymbol{w}$ and $\mathrm{der}(\boldsymbol{x})$, given $\boldsymbol{x}$, $\boldsymbol{u}$, $\boldsymbol{p}$, $\boldsymbol{q}$ and $\boldsymbol{t}$
- Commonly used for simulation, once initialization has been performed

DAE System: set of equations

Parameter-Binding Equations

$$p_i = G_i(p)$$

- **Acyclic** system of equations (strictly diagonal BLT)

Initial Equations

$$H_i(x, \mathrm{der}(x), u, w, p, q)=0$$

- Combined with the Dynamic equations and Parameter Binding equations,
  determine the values of $\boldsymbol{x}$ and $\boldsymbol{q}$ at the initial time $\boldsymbol{t_0}$

Important remark

- Different subsets or the equations for different problems
- Simulation
  - Complete set of equations numerically solved at initialization
  - Dynamic equations numerically solved at each time step, with fixed p and q
- Transformation into LFT form
  - Parameter binding equations solved symbolically for the uncertain parameters
  - Results symbolically substituted into dynamic equations
  - Initial equations irrelevant
- Optimization
  - Some parameters might be subject to dynamic optimization, so their numerical values are not fixed a priori during the optimization run
  - Also initial conditions might be subject to optimization

Representation of Modelica functions

- Equations and variables are brought into scalar form
  - Systems are typically heterogeneous, so maintaining arrays and complex data types is not that useful
  - Eventually all scalars grouped into one big ”system vector”

But...

- Modelica function algorithms involve complex data structures (not easily scalarized)

- Equations involves scalars only
- Original data structures are kept in the function definition
- At the interface, constructors populated with scalar variables are used
- Easy translation into any back-end!

Functions with structured inputs

```
record R
Real X;
Real Y[3];
end R;
```

```
function F
input R X;
output Real Y;
end F;
```

An equation with a function call to F is represented as:

```
F(R(x,{y[1],y[2],y[3]}))-3=0
```

Functions with structured output

```
function f
input Real X;
output Real Y[3];
end f;
```

`x + f(y) * f(z)=0` (* scalar product) is mapped into

```
({aux1,aux2,aux3}) = f(y);
({aux4,aux5,aux6}) = f(z);
X+aux1*aux4+aux2*aux5+aux3*aux6=0
```

Functions with multiple outputs

(out1,out2,...,outN) = f(in1,in2,...,inN)

```
record R1
Real X;
Real Y[2,2];
end R1;
```

```
function F1
input Real x;
output Real y;
output R1 r;
end F1;
```

A call to F1 is mapped into a special form of equation (not in residual form):

```
(var1,R1(var2,{{var3,var4},{var5,var6}}))=F1(x)
```

The FMI XML Schema

- The **FMI 1.0** schema as a starting point:
  - Advantage of starting from an **accepted standard**
  - Already contains a **definition of variables**
- Definition of variables extended with **qualified names** supporting array indices
- The schema has been **extended** with the representation of equations, functions and records
  - Functions cannot be fully scalarized
  - Arrays and Records serve as containers for scalar variables in function arguments

http://www.functional-mockup-interface.org/

XML Schema : modularity

- A **modular** approach based on namespaces:
  - Reuse
  - Extensibility
  - Easier maintenance

- **Modules**:
  - Expressions (exp)
  - Equations (equ)
  - Functions (fun)
  - Algorithms (fun)
  - Optimization (opt)

XML Schema : expressions

- Supported expressions:
  - Literal expressions
  - Unary operations (including built-in functions)
  - Binary operations (+,-,*,/,^,...)
  - Function Calls (referring to user-defined functions)
- Example: `3+der(x)`

```
<exp:Add>
   <exp:IntegerLiteral>3</exp:IntegerLiteral>
   <exp:Der>
      <exp:Identifier>x</exp:Identifier>
   </exp:Der>
<exp:Add>
```

XML Schema : equations

- Dynamic equations:
  - Residual form equations, e.g. `der(x) = -x`
  - Function call equations,e.g. `(v,w) = F(4)`
- Initial equations
- Binding equations, e.g. `p_3 = p_1+p_2`

```
<equ:Equation>
  <exp:Sub>
    <exp:Der>
      <exp:Identifier>
        <exp:QualifiedNamePart name=”x"/>
      </exp:Identifier>
    </exp:Der>
    <exp:Neg>
      <exp:Identifier>
        <exp:QualifiedNamePart name=”x"/>
      </exp:Identifier>
    </exp:Neg>
  </exp:Sub>
</equ:Equation>
```

XML Schema : functions

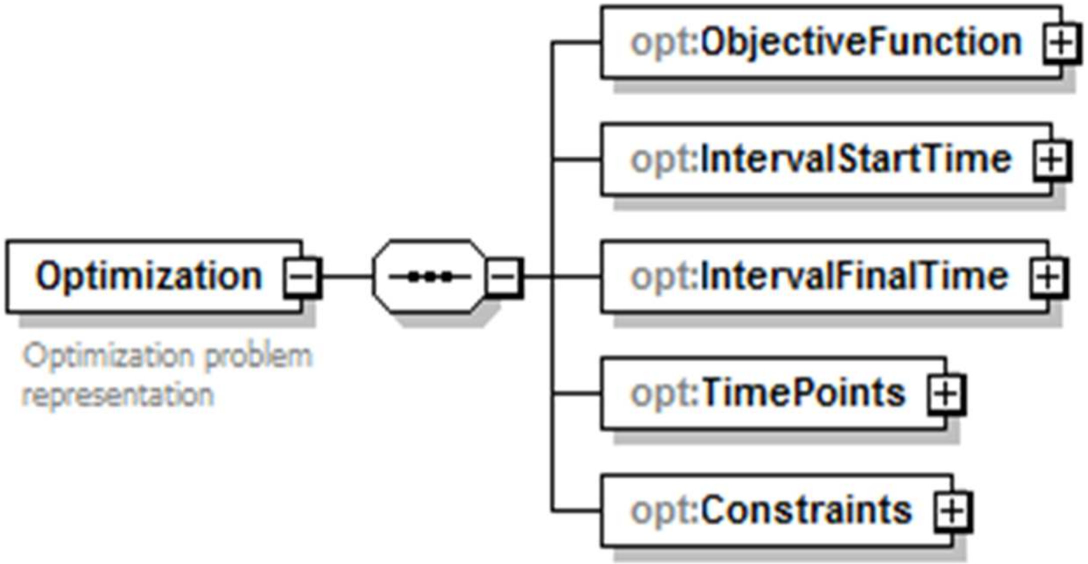
- Algorithms:
  - Represent the algorithm of user defined functions
  - Vectors and records are supported
- Function definition
- Function call in equations can have left hand side of type vector of scalars, record of scalars, scalars, null elements `(v,w) = F(4)`

```
<equ:FunctionCallEquation>
  <equ:OutputArgument>
    <exp:Identifier> <exp:QualifiedNamePart name="v"/> </exp:Identifier>
  </equ:OutputArgument>
  <equ:OutputArgument>
    <exp:Identifier> <exp:QualifiedNamePart name=”v"/> </exp:Identifier>
  </equ:OutputArgument>
  <exp:FunctionCall>
    <exp:Name> <exp:QualifiedNamePart name="F"/> </exp:Name>
    <exp:Arguments>
      <exp:IntegerLiteral>4</exp:IntegerLiteral>
    </exp:Arguments>
  </exp:FunctionCall>
</equ:FunctionCallEquation>
```

XML Schema : optimization problem

Extension of the DAE schema

- Objective function
- Optimization intervals
- Constraints



XML Code Generation in JModelica

- Modelica models are first **flattened**
- XML schema structure mapped to the **abstract syntax tree** of the compiler
- **Aspect oriented** implementation of the code generation, using JastAdd

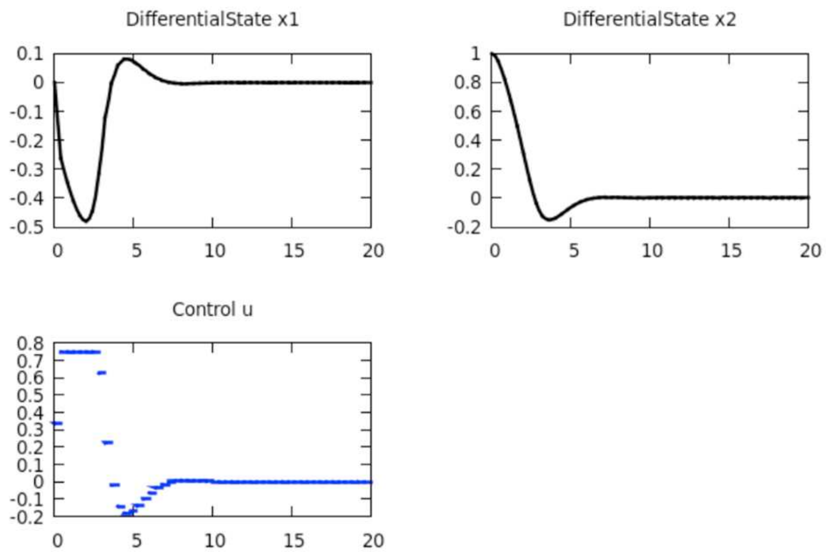
```
public void FArtmBinExp.prettyPrint_XML(Printer p, PrintStream str, String indent, Object o){
    String namespace = "exp";
    String tag = this.xmlTag();
    FExp left = getLeft();
    FExp right= getRight();

    str.println(indent + "<" + namespace + ":" + tag + ">");
    left.prettyPrint_XML(str,p.indent(indent));
    right.prettyPrint_XML(str,p.indent(indent));
    str.println(indent + "</"+ namespace + ":" + tag + ">");
}
```

Test case: ACADO

- *ACADO*: optimization tool developed by *KU Leuven*
- **Export** of model from JModelica.org platform
- **Transform** the XML document into ACADO's native input format
- **Import** the model in ACADO
- **Solve** optimization problem in ACADO

```
optimization VDP_Opt (objective=cost(finalTime),
startTime = 0, finalTime = 20)
  Real x1(start=0,fixed=true);
  Real x2(start=1,fixed=true);
  input Real u;
  Real cost(start=0,fixed=true);
equation
  der(x1) = (1 - x2^2) * x1 - x2 + u;
  der(x2) = x1;
  der(cost) = x1^2 + x2^2 + u^2;
constraint
  u<=0.75;
end VDP_Opt;
```



Conclusions and future work

- With this work a representation for (continuous-time) DAE is proposed
- It is shown how to map the schema to the Modelica language and, concretely, to the JModelica.org compiler
- It is shown how to extend the schema according to special purpose needs, such as optimization problems
- Future work
    - Extension to **hybrid models** $\rightarrow$ complete coverage of Modelica models
    - Standardization within the Modelica Association (as an extension of FMI?)
    - Extension to allow separate compilation (as an extension of FMI?)
    - Continued work on integration with ACADO

Modelon