The New OpenModelica Instance-Based API

Per Östlund    Adeel Asghar

Santa Anna Institute

OpenModelica Annual Workshop February 6, 2023
The Problem

Many old issues that have been hard to solve, for example:

#2081 Conditional connectors not handled by OMEdit
#2891 Hierarchical editing of models
#6111 Support for "visible" attribute missing in OMEdit
#7826 Modifier of redeclared classes should be available through parameter dialog
The Common Issue

- OMEdit uses the OpenModelica Compiler (OMC) to get information about models:

  ![Diagram showing OMC and OMEdit connected by Scripting API]

- The scripting API is mostly based on the abstract syntax tree.
- Hard to deal with dynamic model changes from e.g. modifiers.
Old API Example

model A
  parameter Boolean isVisible;

  annotation (Icon(
      graphics = {Rectangle(
          visible = isVisible,
          ...})});
end A;

model B
  extends A(isVisible = true);
end B;

> getIconAnnotation(B)
{ }

> getInheritedClasses(B)
{ A }

> getIconAnnotation(A)
{ -,-,-,-,-,-,{Rectangle(isVisible, ...)} }
Issues With The Old API

• The old API doesn’t instantiate models, because the old frontend was too slow.
• OMEdit needs a lot of API calls to get the information it needs.
• The scripting API mostly returns poorly documented lists of values.
The Solution

• The new frontend is fast enough to make instantiation feasible.
• One call to get all the information OMEdit needs from the instantiated model.
• Return JSON instead of a list of values.
New API Example

model A
    parameter Boolean isVisible;
    annotation(Icon(
        graphics = {Rectangle(
            visible = isVisible,
            ...)}));
end A;

model B
    extends A(isVisible = true);
end B;

> getModelInstance(B)
{
    "name": "B",
    "restriction": "model",
    "extends": [
        {
            "baseClass": {
                "name": "A",
                "annotation": {
                    "Icon": {
                        "graphics": [
                            {
                                "$kind": "record",
                                "name": "Rectangle",
                                "elements": [
                                    {
                                        "$kind": "cref",
                                        "parts": [
                                            { "name": "isVisible" }
                                        ]
                                    }]
                            },
                            ...
                        ],
                        "components": [
                            {
                                "name": "isVisible",
                                "type": "Boolean",
                                "value": { "binding": true }
                            }
                        ]
                    }
                }
            }
        }
    ]
}
Challenges

• Avoid unnecessary frontend work that’s not needed by the instance API.
• Keep information that the frontend normally throws away.
• Get information from erroneous models.