Functional Mock-up Interface

For Model Exchange

Presenter: Mohsen Torabzadeh-Tari

Slides: Azam Zia

Functional Mock-up Interface

- Purpose:

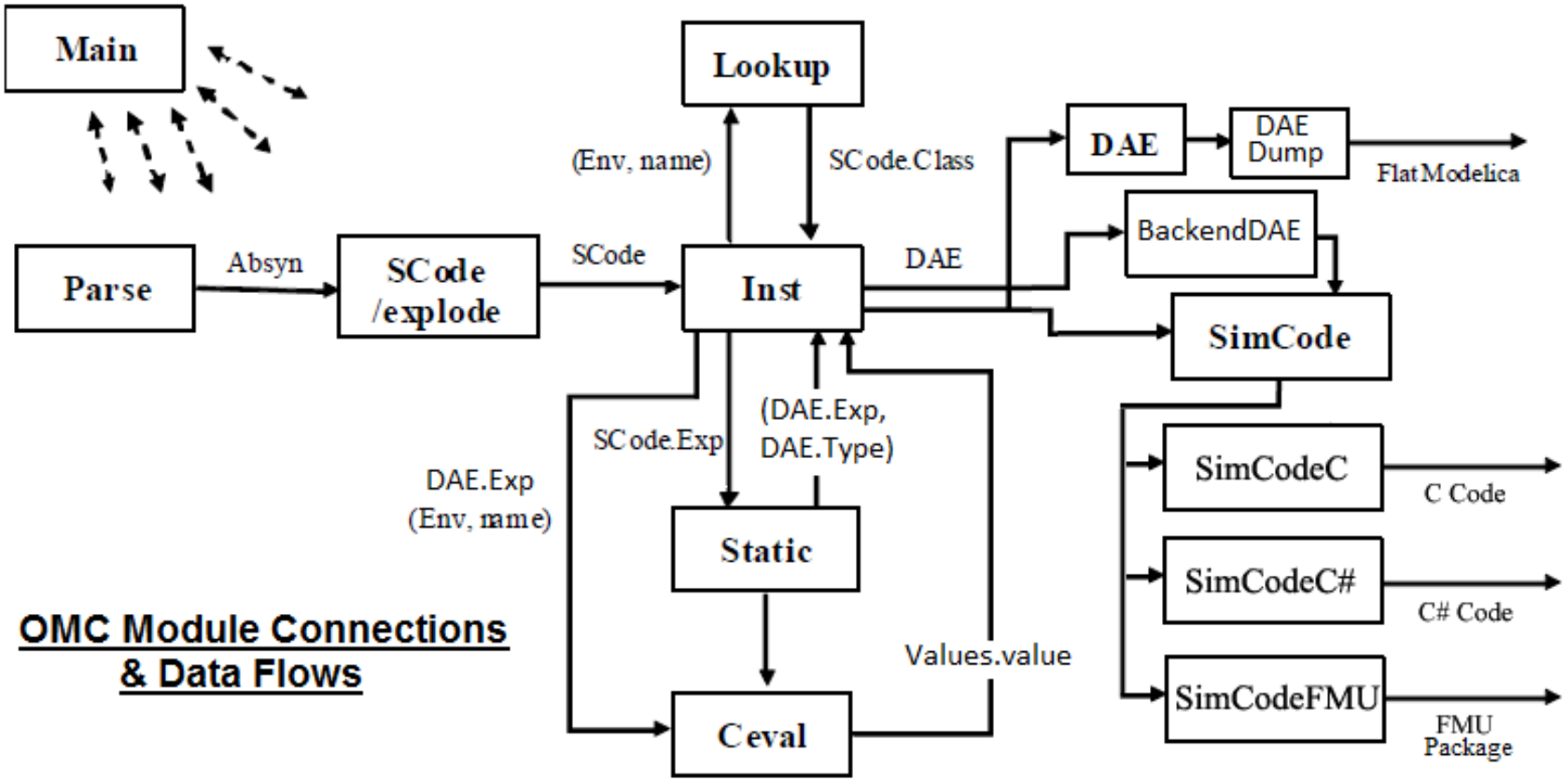
  To describe models and provide an interface to evaluate them as needed in different simulation environments.
- Defines an interface for an executable called FMU (Functional Mock-up Unit).
- FMI functions are called by a simulator to create instances of the FMU, called models.
- An FMU may either be self-integrating (co-simulation) or require the simulator to perform numerical integration.

Parts of FMU

A model is distributed in one zip-file that contains several files:

- XML file which contains the definition of all variables in the model and other model information.
- C file which has all needed model equations with a small set of easy to use C-functions. These C-functions can either be provided in source and/or binary form.
- Further data like a model icon, documentation files and maps needed by the model. All object libraries or DLLs that are utilized.

OMC System Design



Susan Template Language

- We have implemented FMU module in Susan Template language. A Susan compiler translates source code in the Susan language into the MetaModelica language.

```
template DefineVariables(SimVar simVar, Integer x)
  "Generates code for defining variables in c file for FMU
  target. "
::=
match simVar
  case SIMVAR(__) then
  let description = if comment then '// "<%comment%>"'
  <<
  #define <%crefStr(name)%>_ <%prefix%>;
  <%description%>
  >>
end DefineVariables;
```

MetaModelica Code

- Generated MetaModelica Code for susan function.

```
public function DefineVariables
  input Tpl.Text in_txt;
  input SimCode.SimVar in_a_simVar;
  input String in_a_prefix;
  output Tpl.Text out_txt;
algorithm
  out_txt :=
  matchcontinue(in_txt, in_a_simVar, in_a_prefix)
    local
      ...
    case(txt, SimCode.SIMVAR(comment = i_comment, name = i_name), a_prefix )
      equation
        txt = Tpl.writeTok(txt, Tpl.ST_STRING("#define "));
        ...
      then txt;
    case ( txt, _, _ )
      then txt;
  end matchcontinue;
end DefineVariables;
```

XML Data File

```
<?xml version="1.0" encoding="UTF-8"?>
<fmiModelDescription
  fmiVersion="1.0"
  modelName="BouncingBall"
  modelIdentifier="BouncingBall"
  guid="{36dcdcac-cb78-431e-8f91-dd0f993b24cf}"
  generationTool="OpenModelica Compiler 1.6.0"
  generationDateAndTime="2011-2-1T15:3:11Z"
  variableNamingConvention="structured"
  numberOfContinuousStates="2"
  numberOfEventIndicators="2">
  <ModelVariables>
  <ScalarVariable name="h“ valueReference="10“
   description="height of ball“ variability="continuous“
   causality="internal“ alias="noAlias">
    <Real start="1.0" fixed="true"  />
  </ScalarVariable>
   ...
  </ModelVariables>
</fmiModelDescription>
```

FMI Import Functions

- fmiModelTypes.h contains basic type definitions e.g:

```
typedef int  fmiInteger;
```

- These header files must be utilized when compiling a model. In order to have unique function names, every "real" function name is constructed by prepending the function name by "MODEL_ID" + "_" where "MODEL_ID" is the short name of the model.
- fmiModelFunctions.h defines all functions for model execution interface e.g

```
DllExport fmiStatus fmiGetDerivatives(fmiComponent
  c, fmiReal derivatives[], size_t nx);
DllExport fmiStatus fmiTerminate (fmiComponent c);
```

C Code

- Some code for import interface functions is implemented in C language:

```
fmiStatus fmiSetReal(fmiComponent c, ... ){
  int i;
  ModelInstance* mi = (ModelInstance *)c;
  if (validateState(mi, modelInstantiated))
    return fmiError;
  ...
  for (i=0; i<totalReals; i++) {
    if (varOutOfRange(mi, TOTAL_REALS))
      return fmiError;
    comp->r[vr[i]] = value[i];
  }
  return fmiOK;
}
```

Whats Done

- XML Data export.
- C FMI interface functions.
- C export for variable definitions.
- C export for functions like model initialization, real variable etc.
- FMU Makefile
- API for FMI translation
  translateModelFMU(ModelName)

Problems Faced

- Understanding existing code for OMC
- Reuse of existing functions
- Learning Template Language

Whats Left

- Getting event indicators/zero crossings. (10 days)
- Event update function.(5 Days)
- Getting time events.(10 Days)
- Testing exported FMUs.(5 Days / Until fixed)

Thanks