1. State Chart 2 Finite State Machine Example

This example specifies a transformation that converts models belonging to the StateChart (A hierarchical concurrent state machine) paradigm to a semantically equivalent model in the FiniteStateMachine paradigm.

1.1. Directory Organization

- **StateChart**
  - HSM2FSM.mga - SC 2 FSM transformation
  - HSM2FSM.xme - transformation exported
  - **Meta**
    - **Icons** - Icons for the paradigms
    - FiniteStateMachine.mga - FSM metamodel
    - FiniteStateMachine.xme - FSM metamodel exported to XML
    - GS_FiniteStateMachine.xmp - FSM paradigm file
    - StateChart.mga - SC metamodel
    - StateChart.xme - SC metamodel exported to XML
    - GS_StateChart.xmp - SC paradigm file
  - **Models**
    - 2BitCounter.mga - Example model for SC paradigm
    - 2BitCounter.xme - Example model exported
    - 3BitCounter.mga - Example model for SC paradigm
    - 3BitCounter.xme - Example model exported
  - **Udm** - Will contain the Udm meta files
  - **Gen** - Will contain the CG tool generated files

1.2. How to run StateChart 2 Finite State Machine example?

Open HSM 2 FSM transformation model

- Directly open $/ HSM2FSM.mga, if it fails, open GME, choose File/Import XML, and choose $/ HSM2FSM.xme

HSM2FSM.mga contains the transformation rules, UDM compatible meta information paradigms and configuration information. Following is the folder structure which is shown in browser:

- HSM2FSM
  - GS_StateChart - Input Metamodel in UML class diagram format
  - GS_FiniteStateMachine - Output Metamodel in UML class diagram format
  - CrossLinks - UML class diagram for cross reference associations
  - zt_HSM2FSM - Folder containing the transformations
  - zz_Config - Folder containing configuration information

Run the HSM 2 FSM transformation model

- Invoke the GReAT Master Interpreter with icon (**This is a required step for the first time running**). Use the default file paths provided.
- The transformations can be invoked in various ways
  1. GR Engine – Performs the transformations in an interpretive manner
  2. GR Debugger – Provides a user interface and debugging features such as break points, single step, step into etc.
  3. Code generator – Converts the transformation into code that can be compiled and executed.
- To run GR Engine, it could be done either :
  - In the GReAT Master Interpreter dialog , check the box of "Run GR Engine"; **-or-**
  - Directly invoke the GR Engine interpreter with icon .
  - The default input file is $/Models/TwoBitCounter.mga
  - The output files will be $/Models/OutSC.mga and $/Models/OutFSM.mga
- To run the GR Debugger
  - Open a command prompt and go to the sample directory $/. Invoke GRD by calling GRD.exe , then load the config file $/config.mga **-or-**
  - Directly invoke the GR Debugger interpreter with icon .
- To run Code Generator, it could be done either :
  - In the same dialog of GReAT Master Interpreter, check the box of "Run Code Generator"; **-or-**
  - Directly invoke the Code Generator interpreter with icon ; **-or-**
  - Open a command prompt and go to the sample directory $/. Invoke CG by calling CG.exe , with config file $/config.mga
  - After the files have been generated open the generated Visual Studio project using VS71 and compile the project
  - You can run the generated code with default arguments by setting the working directory to be ..\ and Program argument to be –d (default)