1. Signal Flow 2 Flat Signal Flow Example

This example converts models of the SignalFlow paradigm (a hierarchical data flow paradigm) to the equivalent models in the FlatSF paradigm (a flat data flow representation with buffers on the edges)

1.1. Directory Organization

- 📁 **SignalFlow2FlatSF**
    - SignalFlow2FlatSF.mga - SignalFlow 2 FlatSF transformation
    - SignalFlow2FlatSF.xme - Transformation exported XML
    - SignalFlow2FlatSF_test_globalObject.mga - Global namespace example
    - SignalFlow2FlatSF_test_globalObject.xme - Exported XML of above
    - SignalFlow2FlatSF_test_sortingFunc.mga - Sorting function example
    - SignalFlow2FlatSF_test_sortingFunc.xme - Exported XML of above
    - SignalFlow_test_distinguished.mga - Distinguished cross product example
    - SignalFlow_test_distinguished.xme - Exported XML of above
    - 📁 **Meta**
        - 📁 **SignalFlow_Meta**
            - 📁 **Icons** - Icons for the SignalFlow paradigms
            - SignalFlow.mga - SignalFlow metamodel
            - SignalFlow.xme - SignalFlow.mga exported to XML
            - SignalFlow.xmp - SignalFlow paradigm file
            - SignalFlow-uml.mga - UML class diagram of SignalFlow
            - SignalFlow-uml.xme - SignalFlow-uml.mga exported to XML
        - 📁 **FlatSF_Meta**
            - 📁 **Icons** - Icons for the FlatSF paradigms
            - FlatSF.mga - FlatSF metamodel
            - FlatSF.xme - FlatSF.mga exported to XML
            - FlatSF.xmp - FlatSF paradigm file
            - FlatSF-uml.mga - UML class diagram of FlatSF
            - FlatSF-uml.xme - FlatSF-uml.mga exported to XML
    - 📁 **Models**
        - SignalFlow_1.mga - Model of SignalFlow paradigm
        - SignalFlow_2.mga - Model of SignalFlow paradigm
        - SignalFlow_3.mga - Model of SignalFlow paradigm
        - SignalFlow_4.mga - Model of SignalFlow paradigm
        - SignalFlow_5.mga - Model of SignalFlow paradigm
        - SignalFlow_6.mga - Model of SignalFlow paradigm
        - SignalFlow_7.mga - Model of SignalFlow paradigm
        - SignalFlow_1.xme - SignalFlow_1.mga exported XML
        - SignalFlow_2.xme - SignalFlow_2.mga exported XML
        - SignalFlow_3.xme - SignalFlow_2.mga exported XML
        - SignalFlow_4.xme - SignalFlow_2.mga exported XML

    - SignalFlow_5.xme - SignalFlow_2.mga exported XML
    - SignalFlow_6.xme - SignalFlow_2.mga exported XML
    - SignalFlow_7.xme - SignalFlow_2.mga exported XML
  - **Gen**
    - Gen.dsp - project file to compile generated code
    - Gen.dsw - workspace to compile generated code
  - **Udm** - Will contain the Udm meta files

1.2. How to run SignalFlow 2 FlatSF example?

Step 1: Register SignalFlow and FlatSF paradigm

- Open GME, choose File/RegisterParadigms, click on "Add From File", and choose $/meta/SignalFlow_meta/SignalFlow.xmp;
- Same registration process for $/meta/FlatSF_meta/FlatSF.xmp

Step 2: Open SignalFlow 2 FlatSF transformation model

- Directly open $/ SignalFlow2FlatSF.mga, if it fails, open GME, choose File/Import XML, and choose $/ SignalFlow2FlatSF.xme

SignalFlow2FlatSF.mga contains the transformation rules, UDM compatible meta information paradigms and configuration information. Following is the folder structure which is shown in browser:

- SignalFlow2FlatSF
  - CrossLinks - UML class diagram for cross reference associations
  - FlatSF - FlatSF Metamodel in UML class diagram format
  - SignalFlow - SignalFlow Metamodel in UML class diagram format
  - zt_SF2FSF - Folder containing the transformations
  - zz_Config - Folder containing configuration information

Step 3: Run the SignalFlow 2 FlatSF transformation model

- Invoke the GReAT Master Interpreter with icon (**This is a required step for the first time running**), Use the default file paths and names 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 same dialog box of GReAT Master Interpreter, check the box of "Run GR Engine";
  - Directly invoke the GR Engine interpreter with icon .
  - The default input file is $/Models/SignalFlow_1.mga
  - The output files will be $/Models/outSF1.mga

- To run the GR Debugger
  - Open a command prompt and go to the sample directory $/.
  - Invoke GRD by calling GRD.exe
  - Load the config file $/config.mga
- To run Code Generator, it could be done either :
  - In the same dialog box of GReAT Master Interpreter, check the box of “Run Code Generator”;
  - Directly invoke the Code Generator interpreter with icon ;
  - After the files have been generated open $/gen/gen.dsw 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.

1.3. Global Object, Sorting Function and Distinguished Merging examples

You can follow the same steps listed above to run these examples. The UMT files for these examples are as below:

1. SignalFlow2FlatSF_test_globalObject.xme – This is the example demonstrating the usage of Global Objects in GReAT.
2. SignalFlow2FlatSF_test_sortingFunc.xme – This example demonstrates sorting in GReAT
3. SignalFlow_test_distinguished.xme – This example demonstrates distinguished merging in GReAT.

Please refer to the GReAT user manual for more information on these features.