Context Model FMI Integration

Standalone simulations that employ context model libraries are commonly written in languages that require specific interface protocols. The rationale and motivation for enabling a reliable co-simulation environment via contractual interfaces is described elsewhere [1][2]. As a potential solution proffered, the interoperability among simulations is proscribed by the MODELISAR project [3] (in support of the AUTOSAR vehicle design initiative) [4]. This appendix describes the process of prototyping context model interfaces to be exercised in a larger vehicle simulation environment.

1. Summary

There was a desire to make the context models available for use within the Modelica or Dymola environment. One of the ways this could be accomplished is to use the Functional Mockup Interface (FMI) that is a model import feature of Modelica/Dymola and can be purchased as a 3rd party toolkit in MATLAB. In order to demonstrate how to import models using FMI we created several Functional Mockup Units (FMUs) for several of the context models as prototypes on how the context models could be used to create components that could be imported into Modelica or Dymola.

1.1 Technical Challenges

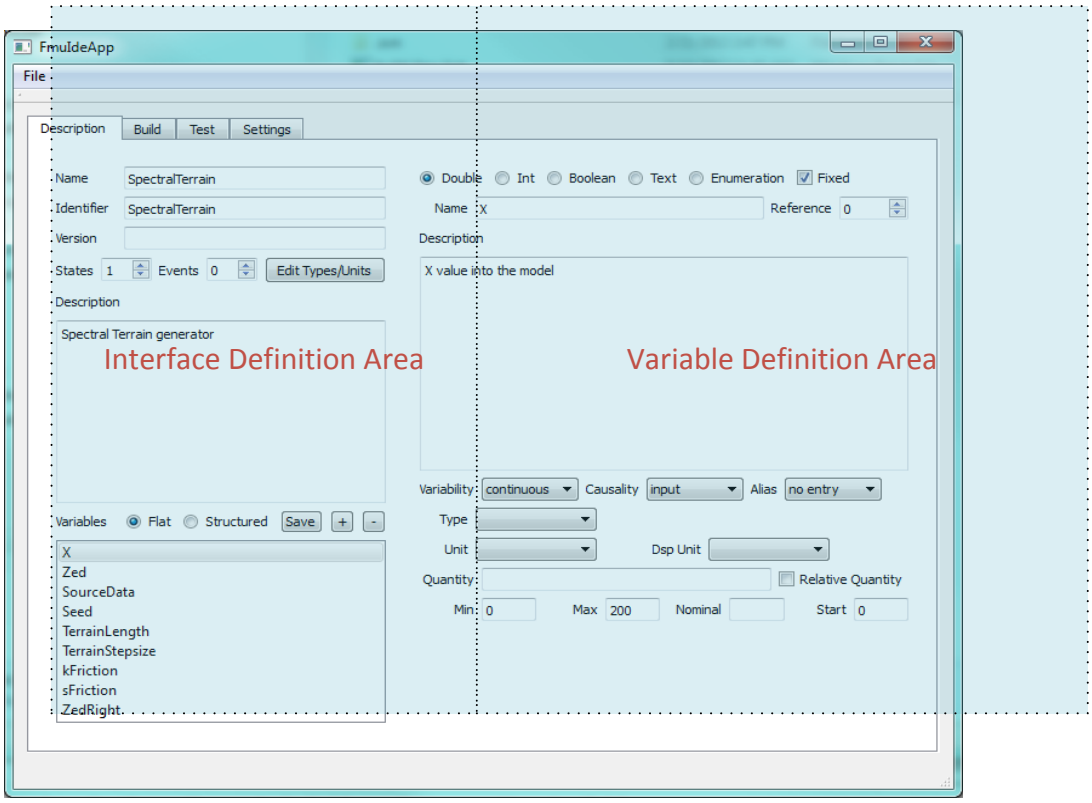
We looked into what tools were available for creating FMUs. What we found was a lack of open source tools for creating FMUs and the ones we found were limited in their capabilities and ease of use. Although the FMI specification was found to be sufficient for the designed purpose of integration with multiple tools we found that OpenModelica, Dymola and MATLABs implementation to be inconsistent. We found numerous bugs and non-standard implementations during our development indicative of an immature specification that isn't widely used.

1.2 General Methodology

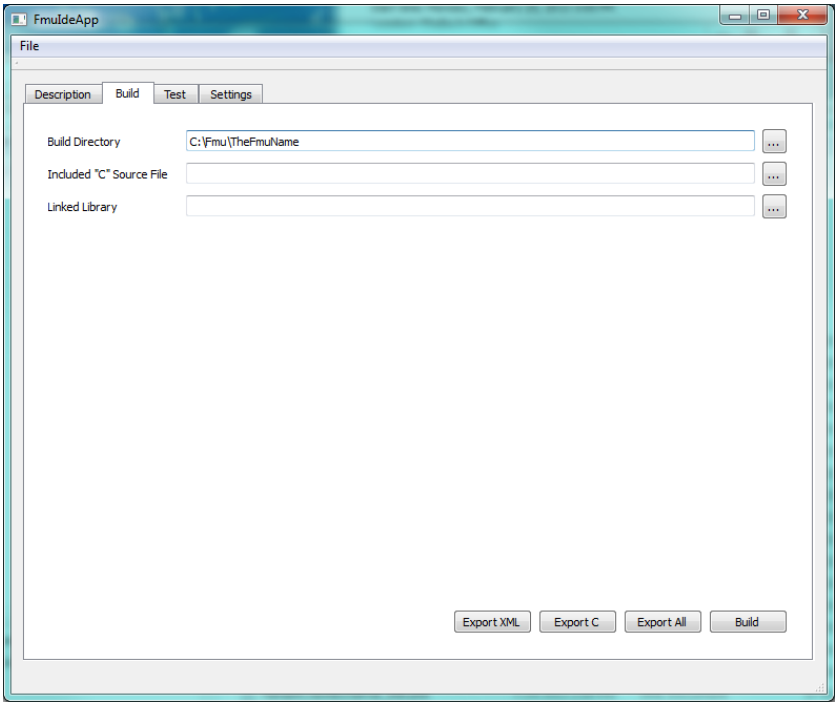
After a search of the available tools we settled on the FMU SDK as an open source toolkit that would allow us to package our C++, Visual Studio based models into FMUs that could be imported using the FMI import features of the META development tools. Creating the needed files to create an FMU using the FMU SDK was found to be a manual process that is prone to error. In order to reduce the chances of error and to ensure that the resulting interface was compliant with the FMI specification we decided that a FMU development tool was needed. This tool could be used to define the FMU using an easy to use GUI interface that would only aid the user in developing the interface and then export the files needed for implementing the interface using the FMU SDK.

1.3 Technical Results

We created a tool called the FMU IDE (Integrated Development Environment) to aid in FMU creation. The tool (see below) has an interface definition tab that has an interface definition area that takes up the left half of the tab and on the right half is the variable definition area that shows the information for the currently selected variable.



When the user is satisfied that the interface is defined they can use the build tab to export the XML, the C code or both. The tool will export to the directory define in the Build Directory text box. The user can use the "…" button to browse to the desired directory. As an option the user can define a C language source file to be included in the exported file via a "#include" statement. This included source file could have additional links to source files or functions that are used to define the behavior of the interface.



Prior to building the FMU the developer must unzip the fmusdk.zip file that is in the …\Models\Tools directory into that directory.

In order to build the FMU the developer must do the following steps:

1. In the directory directly below the directory containing the FMU's C and XML file make a copy of the build_all.bat file that is in …\Models\Tools\exec
2. Modify the file for the specific FMU
3. Run the modified build_all.bat from the command line or by double clicking from the Windows Explorer.
4. It will create an FMU in the directory ..\fmu\me for model exports FMUs or ..\fmu\cs for co-simulation FMUs.

1.4 Important Findings and Conclusions

As stated earlier the acceptance of the FMI specification is in its infancy and in need of a greater breadth of tools that use FMUs for importing models. To that end we created a tool that could be available as an open source addition to the FMU SDK. This tool would enable standardized models to be available via FMUs. Although this tool and the resulting context model FMUs were developed using Visual Studio 2010 on Windows 7 it should be fairly easy to port to other environments because the tool uses QT and doesn't use Windows only extensions to standard C++.

We created six FMUs and examples that demonstrate their use. The six are:

1. TerrainService.fmu: This model is similar to the SpectralTerrain.fmu and the DiscreteObstacle.fmu but this uses the terrain server. An example that uses this FMU

is the TerrainCrawler2server.mo that is located in the directory …\Models\Land\terrains\grades_slopes\interface\Modelica.

2. TempFMU.fmu: This model returns the diurnal temp. It uses the time from the simulation environment to determine the temperature.
3. SpectralTerrain.fmu: This FMU returns two Z values for right and left (ZedRight and Zed) values (m) for every x value input into the model. The initialization values a seed for the random number generator, the desired terrain length (m), the step size for the terrain (m), needed are an enumerated value that is defined in the file Terrain PSD metadata.xlsx. The file resides in the directory …\Models\Land\terrains\grades_slopes\data\WaveNumberSpectra. It also returns the kinetic and static friction for the course. Examples that use this FMU are TerrainCrawler.mo and FrictionTerrainCrawler.mo that reside in the directory …\Models\Land\terrains\grades_slopes\interface\Modelica.
4. SeaState.fmu: This FMU provides a pressure value (in Pa), the height of the water surface, and the u,v,w of the water velocity values for x,y,z (in m/s). The input is the heading of the vessel (in radians), the speed of the vessel (in m/s) the ocean depth (in meters) and the u,v,w of the water velocity value. An example that uses the SeState.fmu is buoy.mo in the directory …\Models\Aquatic\features\sea_state\interface\Modelica.
5. DiscreteObstacle.fmu: Provide two Z values (in meters), for each X input value (in meters), for left and right tracks for a defined obstacle course. The main initialization input variable is an enumerated value for the obstacle that is documented in …\ModelsLand\obstacles\data\Obstacle metadata.xlsx. The state of the model is determined by varying X and the model returns a Zr(right) and Zl(left). An example that uses this FMU is the ObstacleTerrainCrawler.mo in the directory …\Models\Land\obstacles\interface\Modelica.
6. ClimateServe.fmu – Provides the current temperature. The input is an enumerated value for the climate design type:
   0) Hot dry: QSTAG 360 = A1 (north Africa, Middle East, Pakistan, India, southwest US, northern Mexico)
   1) Hot humid: QSTAG 360 = B3 (Persian Gulf, Red Sea)
   2) Constant high humidity: QSTAG 360 = B1
   3) Variable high humidity: QSTAG 360 = B2
   4) Basic hot, QSTAG 360 = A2 (US, Mexico, Africa, Asia, Australia, South America, southern Spain, SW Asia)
   5) Basic cold, QSTAG 360 = C1 (high latitude coasts, eg, southern Alaska)
   6) Cold, QSTAG 360 = C2 (Canada, Alaska, Greenland, northern Scandinavia, northern Asia, Tibet, Alps, Himalayas, Andes)
   7) Severe cold, QSTAG 360 = C3 (interior Alaska, Canadian Yukon, Greenland icecap, north Asia)

   Also the other input is the enumerated value for the condition type as follows:

   0) Operational: conditions in open to which materiel may be subjected to during operations or standby
   1) Storage transit: Conditions materiel may be subjected to in storage or transit

   The examples for this FMU are the ColdStart.mo and the WeatherClient.mo

1.5 Implications for Further Research

Expanding the FMU IDE to include build and test capabilities would make the FMU creation easier. The current hand building of the build batch file could easily be done under the covers of the FMU IDE. There is also plenty of room to mature the implementation of the the FMI specification in OpenModelica or teaming with Dassault Systemes to mature the implementation in Dymola. Also an open source implementation for MATLAB and other tools is an area to needs work.

1.6 References

[1] H. Heinecke, W. Damm, B. Josko, A. Metzner, H. Kopetz, A. Sangiovanni-Vincentelli, and M. Di Natale, "Software components for reliable automotive systems," presented at the Proceedings of the conference on Design, automation and test in Europe, 2008, pp. 549–554.

[2] S. Bankes, D. Challou, T. Haynes, H. Holloway, P. Pukite, J. Tierno, and C. Wentland, "META Adaptive, Reflective, Robust Workflow (ARRoW)," BAE Systems, Final Report TR-2742, 2011.

[3] T. Blochwitz, M. Otter, J. Akesson, M. Arnold, C. Clauß, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, and D. Neumerkel, "Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models," presented at the 9th International Modelica Conference, Munich, 2012.

[4] "MODELISAR - Wikipedia, the free encyclopedia." [Online]. Available: http://en.wikipedia.org/wiki/MODELISAR. [Accessed: 26-Feb-2013].