Documentation:Trade Space Exploration with Ptolemy II

January 23, 2010

This document explains the the Trade Space Specification Tool's use of Ptolemy II as a simulation environment. For an explanation of the use of the Trade Space Specification tool please contact:

Yeou-Fang Wang
4800 Oak Grove Drive
Pasadena,CA 91109
yeou-fang.wang@jpl.nasa.gov

1 Overview

The objectives of the project are two-fold. The first is to use Ptolemy II for modeling of the various spacecraft architectures specified by the Trade Space Specification Tool. The model results will help determine performance, mass, costs, risk, and other trends based on architectures selected for the spacecraft. The second objective is to facilitate quick high-level actor development in the model environment in the initial stages of design.

To automatically determine the cost, risk, and mass of all the components within a craft architecture specification, one would need a model of each feasible architecture option in Ptolemy II. The Trade Space Specification Tool is able to output the options within an architecture but does not currently contain information about a model of the component/option itself. Since Ptolemy II uses actors for each component in a model, we realized we had two options to create an actor for each component. The first was to (1) create custom actors for each feasible option or (2) create one generic actor that can be customized for each architecture option.

Since the Trade Space Specification Tool focuses on the early design phase, we wanted there to be the use of a set of rules when there is no model and a need for rough order calculations for decisions. For this implementation, we used fuzzy logic. Our solution was to create a fuzzy logic actor within the Ptolemy II framework that uses Edward Sarzonov's Java Fuzzy Engine. Fuzzy Logic refers to reasoning that is approximate instead of precise. Common rules associated with fuzzy logic include: (1) if gas is expensive, then take the bus,

and (2) if gas is cheap, then drive your car. Though the rules are approximate in terms of expensive and cheap the defuzzification of the terms produces a precise result based on the price of gas. It is also relevant to note that the specification of expensive and cheap can change without the need to change the fuzzy rules.

Each instance of the fuzzy logic actor expects rules governing its behavior to be specified in an XML(XML/FCL Fuzzy Control Language XML) file, whose name is the generic name of the component. In the case of a Solar Power component, the fuzzy logic actor expects to find its rules defined in the power.xml file. An example of the expected file format is shown in Listing 1. The use of fuzzy logic allows quick, high-level actor development in the modeling environment.

To accomplish our second objective, we created the TSSTtoPTII model generator. The TSSTtoPTII model generator expects XML output from the TSST in the format shown in Listing 2. TSSTtoPTII model generator, written in Java, reads the file filename.xml specified by the user and produces a MoML XML file filenameModel.xml to be run by Ptolemy II.

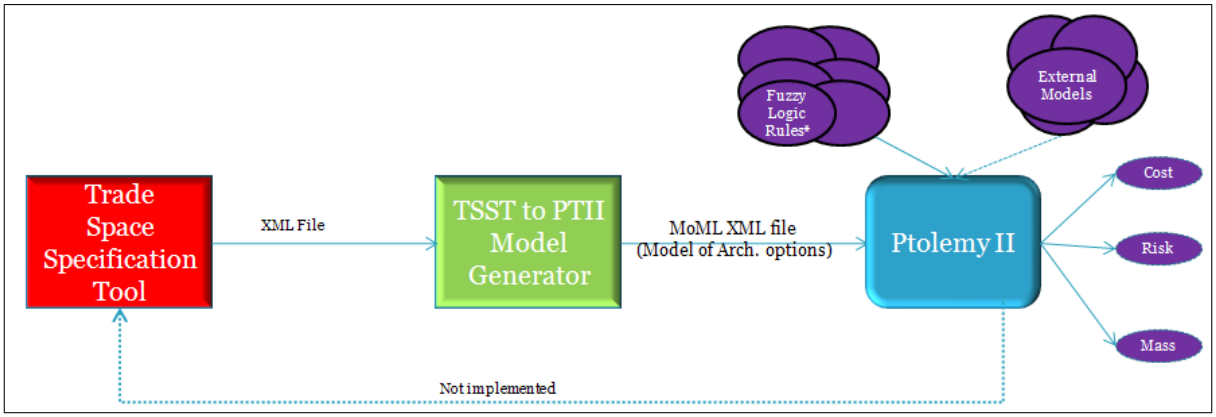

Figure 1: TSST to Ptolemy II Framework

We successfully created the initial TSSTtoPTII framework shown in Figure 1. Here the Trade Space Specification Tool produces an XML file that is sent to the TSSTtoPTII model generator. The TSSTtoPTII model generator produces a MoML XML file. Running Ptolemy II with the created MoML XML file determines the cost, risks, and mass of the model. Each fuzzy logic actor instantiated in the model has an associated XML file specifying its rules. These files contain fuzzy logic rules and relevant associated information for the model. Though shown in the framework diagram, it is important to note that we are not currently linking to external models for the simulation of components in this stage of the TSSTtoPTII framework.

2 Installation and Use

Trade Space Exploration with Ptolemy II explores the use of the Ptolemy II modeling and simulation framework to determine the cost, risk, mass, etc. of architecture options as well as to facilitate a human in the loop scheme in the modeling environment. To use

Ptolemy II with the Trade Space Specification Tool, Ptolemy II must be installed. Ptolemy II is installed in `C:\DocumentsandSettings\sforbes\ptII` on longbeach.jpl.nasa.gov. To install Ptolemy II on a different computer please visit `http://ptolemy.eecs.berkeley.edu/ptolemyII/` and follow the download and installation instructions. The Ptolemy II installation on longbeach was checked out from svn (version control) under an account with read,modify,write privileges. It is also possible to check out a read only copy of the Ptolemy II source code from svn. The repository location is `source.eecs.berkeley.edu/home/svn/chess/ptII/`. After downloading Ptolemy II, you launch the graphical user interface either from the command line with cygwin, or from Eclipse. Instructions on setting up eclipse for Ptolemy II can be found at `http://ptolemy.berkeley.edu/ptolemyII/ptII7.0/ptII7.0.1/doc/coding/eclipse.htm`

The TSSTtoPTII framework uses a Fuzzy Logic Actor defined in Ptolemy II. The java code for fuzzy logic actors can be found in the FuzzyLogic.java file in `ptII\ptolemy\actor\lib\logic\fuzzy`. The fuzzy logic actor in Ptolemy II uses Edward Sarzonov's Java Fuzzy Engine. Sarzonov's Java Fuzzy Engine is open source but it is not under a BSD license. As a result, it must be downloaded and placed in the `ptII\ptolemy\actor\lib\logic\fuzzy` directory. Please download the Fuzzy Engine Code(.tgz or .zip) from `http://people.clarkson.edu/~esazonov/FuzzyEngine.htm` and place it in your `ptII\ptolemy\actor\lib\logic\fuzzy` directory. Open the fuzzy folder,inside the Fuzzy Engine folder, and add the line *package ptolemy.actor.lib.logic.fuzzy.FuzzyEngine.fuzzy;* to all the java files in the directory.

For the TSSTtoPTII framework to function correctly you will need to do the following:

1. Generate XML output for architecture options from the TSST.

   XML output for Architecture options form the TSST is done by selecting the corresponding menu option inside the TSST. After generating the xml file copy it into the `ptII\ptolemy\actor\lib\logic\fuzzy` directory.

2. Create XML files that specify the fuzzy logic behavior of general components

   The XML schema for fuzzy logic rules is shown in Listing 1. Inside the function block entity, there is a list of input and output variables with a name, type, and range. Next the list of variables that need to be fuzzified along with their term names and the points for fuzzification. After the variables that need to be fuzzified listed are the variable(s) that need to be defuzzified along with it's terms. After the list of variables to be fuzzified and defuzzified comes the rule block. Each rule is given a number and the content of the rule is specified in the TEXT attribute. The current program that parses the rules file will process an arbitrary number of fuzzify variables but will only work correctly for one defuzzify variable.
   Place all fuzzy logic rule xml files in the `ptII\ptolemy\actor\lib\logic\fuzzy` directory.

3. Run the java program ModelCreator (located in `ptII\ptolemy\actor\lib\logic\fuzzy`). When prompted enter the name of the xml file with the output from TSST. This program will parse the XML file and create a Ptolemy II model for the architecture.

   To execute the ModelCreator program there is a command line and an eclipse option.
   To run from Eclipse. Start Eclipse.
   Eclipse- Run, Open Run Dialog,
   Create a new Java application with the name ModelCreator.
   On the main tab the project name is PTII, the main class is
   ptolemy.actor.lib.logic.fuzzy.ModelCreator. Select the run button.

4. Run the Ptolemy II model to determine the outputs.

   To run the model there is a command line option and an eclipse option.
   To run from the command line simply type - vergil filenameModel.xml
   To run from Eclipse: Start Exlipse.
   Eclipse- Run, Open Run Dialog,
   Create a new Java application with the name Vergil.
   On the main tab the project name is PTII, the main class is ptolemy.vergil.VergilApplication.
   On the arguments tab enter the absolute path of the model file such as
   `ptII\ptolemy\actor\lib\logic\fuzzy\ArchiModel.xml` in the Program
   arguments box. Select the run button. This should start Vergil and open
   the Ptolemy II model created for the architecture.
   Feel free to move the actors around. When you are ready simply select the
   run button(one button to the left of pause, two buttons to the
   left of stop in the Vergil environment)

If any questions concerning the TSSTtoPTII framework arise, please feel free to contact:
Shanna-Shaye Forbes[1]
Dept. of Electrical Engineering and Computer Sciences
545 Cory Hall, University of California, Berkeley
Berkeley, CA 94720
sssf@eecs.berkeley.edu

[1]This work was conducted with support from a Jenkins Predoctoral Fellowship Mini Grant Award.

Listing 1: Fuzzy Logic Rule File Format:XML/FCL Fuzzy Control Language XML

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<FUNCTION_BLOCK>
  <VAR_INPUT NAME="Water" TYPE="REAL" RANGE="0 100" />
  <VAR_OUTPUT NAME="Power" TYPE="REAL" RANGE="0 75" />
  <FUZZIFY NAME="Water">
    <TERM NAME="Cold" POINTS="0 0 20 40" />
    <TERM NAME="Tepid" POINTS="30 50 50 70" />
    <TERM NAME="Hot" POINTS="50 80 100 100" />
  </FUZZIFY>
  <DEFUZZIFY METHOD="CoG" ACCU="MAX" NAME="Power">
    <TERM NAME="Low" POINTS="0 25 25 50" />
    <TERM NAME="High" POINTS="25 50 50 75" />
  </DEFUZZIFY>
  <RULEBLOCK AND="MIN" OR="MAX">
    <RULE NUMBER="1" TEXT="IF (Water IS Cold) OR (Water IS Tepid) THEN Power IS High" />
    <RULE NUMBER="2" TEXT="IF (Water IS Hot) THEN Power IS Low" />
  </RULEBLOCK>
</FUNCTION_BLOCK>
```

Listing 2: Expected Output format from the Trade Space Specification Tool

```
<linked-list>
  <gov.nasa.jpl.trades.ui.menu.ExportArchitecture_-ArchitectureExport>
    <architectureName>Office prefered</architectureName>
    <options class="linked-list">
      <gov.nasa.jpl.trades.ui.menu.ExportArchitecture_-ArchitectureExport_-OptionExport>
        <optionName>Solar</optionName>
        <associatedDimensions>Power</associatedDimensions>
        <metadata>
          <description>
            <entryBy>Unknown</entryBy>
            <lifeStart>2009-06-26 21:40:20.590 PDT</lifeStart>
            <lifeEnd>2019-07-04 21:40:20.590 PDT</lifeEnd>
          </description>
        </metadata>
        <this_-1 reference="../../.."/>
      </gov.nasa.jpl.trades.ui.menu.ExportArchitecture_-ArchitectureExport_-OptionExport>
      <gov.nasa.jpl.trades.ui.menu.ExportArchitecture_-ArchitectureExport_-OptionExport>
        <optionName>High gain</optionName>
        <associatedDimensions>Antenna</associatedDimensions>
        <metadata>
          <description>
            <entryBy>Unknown</entryBy>
            <lifeStart>2009-06-26 21:40:35.949 PDT</lifeStart>
            <lifeEnd>2019-07-04 21:40:35.949 PDT</lifeEnd>
          </description>
        </metadata>
        <this_-1 reference="../../.."/>
      </gov.nasa.jpl.trades.ui.menu.ExportArchitecture_-ArchitectureExport_-OptionExport>
      <gov.nasa.jpl.trades.ui.menu.ExportArchitecture_-ArchitectureExport_-OptionExport>
        <optionName>Electrical</optionName>
        <associatedDimensions>Propulsion</associatedDimensions>
        <metadata>
          <description>
            <entryBy>Unknown</entryBy>
            <lifeStart>2009-06-26 21:40:42.824 PDT</lifeStart>
            <lifeEnd>2019-07-04 21:40:42.824 PDT</lifeEnd>
          </description>
        </metadata>
        <this_-1 reference="../../.."/>
      </gov.nasa.jpl.trades.ui.menu.ExportArchitecture_-ArchitectureExport_-OptionExport>
    </options>
    <outer-class/>
  </gov.nasa.jpl.trades.ui.menu.ExportArchitecture_-ArchitectureExport>
</linked-list>
```