Compilation of MPL Models in Lisp

Revised: 3/25/04

Model-Based Embedded and Robotic Systems Group, MIT

Introduction

This document describes 1) the modeling files required to compile an MPL model in Lisp, 2) the compilation steps and 3) how to verify the output model after compilation.

Required Modeling Files

MPL is the Model Programming Language originally used to create models for Livingstone (a model-based diagnosis system from NASA Ames). An MPL model describes various nominal and faulty discrete modes of each component in the system, the behavior of components in each mode, and the probabilistic transitions that can occur among the different modes. Refer to the "Modeling with Livingstone" document for an introduction to the MPL modeling language.

Here is a list of MPL modeling files required for compilation:

- **Component model files** (componentName1.lisp, componentName2.lisp, …):
  These model the behavior of independent components (automata) of a system. A component model defines observation and command variable domains and describes the different nominal and faulty modes along with their corresponding modal constraints and transitions guards. These modal constraints and transition guards are expressed in prepositional logic.
- **Module file** (moduleName1.lisp):
  A module file represents the system being modeled. It instantiates all the components in the model, defines the connections between components (i.e. dependant variables), declares observable and command variables, and provides necessary module creation functions.
- **Peripheral files** (*.lisp):
  These files contain functions which define relationships between multiple variables of the different components of the system. These functions may be used in the module files for instance.
- **System file** (moduleName1.system):
  A system file sets up system environment variables and paths necessary for compilation. A system file declares a module and the components of the module. All component and module names used in a system file must match with those of the corresponding lisp files.

Compilation Steps

Compilation of the modeling files described above will generate a single MOF file, which can be input to the Titan model-based executive. To compile:

- Open Lisp in the /Livingstone-MOF/mba directory or set the appropriate path to this directory.
- Load the system file by typing
  :ld system
- Switch to the directory where all the required MPL files described above are located. The following commands may be useful:
  :pw (displays the current working directory)
  :cd *path* (changes the working directory path to ***path***)
  :help (displays a menu of available commands)
- Compile a module by typing
  (mk: compile-system "moduleName")
  For example, if your system file is called simpleCircuit.system, you should type
  (mk: compile-system "simpleCircuit")
- Load the Livingstone package required by the compilation process by typing
  :pa tp
- Compile a module by calling an appropriate module creation function defined in the module file moduleName.lisp. For example, if the module file simpleCircuit.lisp contains a module creation function called create-circuit-with-mrp(), then create the module by typing
  (create-circuit-with-mrp)
- Save the MPL model to an MOF file with user-defined name by typing
  (mpl-save "MOF-Name")
  For example, if you want to save the simpleCircuit model to an MOF file called circuit.MOF, you should type
  (mpl-save "circuit")
- If the compilation is successful, you should obtain a MOF file in your working directory. To exit Lisp, type
  :exit

Model Verification

The MOF file is a compact representation of the entire MPL model encoded in propositional logic. As a result, it is possible to read and understand simple relationships but would be difficult to hand generate from scratch. After compilation, it is important to verify that the generated MOF file is correct. There are a few known problems associated with the compilation process which may require modifying the MPL files and recompiling or hand editing the MOF file. Some of these problems originate from modeling errors what were not caught by the compiler and others are problems with the compiler itself.

Here are a couple typical errors that you should be careful of:

- Entire transitions are left out of the MOF file
- Titan does not have the capability of handling disjunctive transitions guards. An example of this is as follows:

```
(TRANSITION ROVER.MODE
 FROM-VALUE IDLE
 TO-VALUE MOVING
 GUARD (OR (ROVER.COMMAND = MOVE_SLOW) (ROVER.COMMAND = MOVE_FAST))
 PROBABILITY 1)
```

  This is a current modeling limitation of Titan that will be addressed in the near future. Although Titan will not terminate, it will display: "`Warning: disjunctive clause being converted into Constraint!`" and could possibly lead to unexpected diagnoses.

Remember that the CCA model of the system must be well defined before you can begin implementing it in MPL and compiling it into a MOF file. The compilation process will only further aggravate problems that arise due to poor designs. If you cannot walk through a simple scenario of how Titan will diagnose and reconfigure the system, don't expect Titan to solve the problem for you.