Getting Started with OMNI

1 Installing OMNI

OMNI is provided as a compiled java application. A compressed tar file containing the package and supporting files has been tested on Ubuntu 16.4 and is available from:

colinpaterson.bitbucket.io/OMNI/.

In addition you will need to download the following third party software:

Software	Description	URL
HyperStar	a phase type fitting tool	www.mi.fu-berlin.de
PRISM	A probabilistic model checker	prismmodelchecker.org

Having downloaded the software make a note of the installation location of PRISM and HyperStar and then extract the OMNI archive to a local directory.

2 Creating an OMNI model file

To prepare a model for refinement a PRISM CTMC file must be annotated to define the use of PHD holding times and their relationship to states within the model.

If we consider a state transition statement in a PRISM CTMC file to be of the form:

```
[] s=1 -> r_1:(s'=2);
```

where this means that we move from state 1 to state 2 after a time described by an exponential with rate defined by the variable r_1.

Let us assume that state 1 mean that we are examining the rail arrivals board and state 2 means we are executing a search query. We therefore assign short names to the states to aid in readability as "arr" and "src" respectively.

We may then rewrite the lines as:

```
<#arr>s=<?arr> -> (s'=<?bin>);
```

here the opening tag <#arr> indicates that when leaving this state the holding times is as defined in the trace data for the "arr" component. Since OMNI will add states to the model we also use state names rather than explicit numbering. Therefore <?arr> may be read as "The state associated with the arrivals component".

A second more complex mapping from PRISM to OMNI is provided for clarity.

Consider a PRISM line:

```
[] s=2 -> r_2 * p1:(s'=3) + r_2 * (1-p1):(s'=4);
```

This may be read as follows remain in state 2 for a time determined by the exponential rate r_2 and then transition to state 3 with probability p1 otherwise move to state 4. Again we define short names to each state,

s=2 src - searching
s=3 loc - determine current location
s=4 dep - retrieve departure timetable

The OMNI line then becomes:

```
<#src> s=<?src> -> p1:(s'=<?loc>) + (1-p1):(s'=<?dep>);
```

Again the <#src> tag at the beginning of the line indicates that the transition timing is controlled by the trace data associated with the "src" component.

Where a state in the model is not associated with a component, for example and absorbing state, we use a slightly different syntax to describe the state. So assuming the absorbing state has the label "complete" we write:

```
<#dep>s=<?dep> -> (s'=<!complete>);
```

3 Creating component trace files

Each component in an OMNI model is described by a set of observed timing data. This may be available through direct monitoring of component performance or through the analysis of log files.

The timing data is stored in a set of text files with one file per component. Each line of the text file then contains a single integer which is the time associated with a single evocation of of the component.

A PRISM CTMC model assumes a time unit which is specified by the model designer and the unit assumed in the timing data file is then 1000^{th} of the model time units. For our web services case studies time time unit is seconds and hence timing data is provided in milliseconds. For the IT Support System hours is assumed and timing data is provided as hour/1000.

4 Defining a refinement task

When OMNI is executed it expects to find a file called "config.xml" in the same directory as the OMNI.jar file. This file contains all the information OMNI needs to execute a refinement task.

4.1 Dependancies

Having installed HyperStar on your computer we need to tell OMNI where the program is located. In addition a utility program (HyperStarToPrism.jar) is provided with OMNI which converts the output from HyperStar into PRISM format HyperErlang Models.

In the config.xml file the Tools section is used to defined where on your machine these files are located. For example:

```
<Tools>
     <Tool ID="HSLocation">/Users/.../HyperStar.jar</Tool>
     <Tool ID="H2PLocation">HyperStarToPrism.jar</Tool>
</Tools>
```

4.2 File handling

Having created an annotated PRISM model as an input file for the refinement process we specify it's location using the Model tag as:

```
<Model>inputModel.prism</Model>
```

The models which are produced by the OMNI process are defined in the OutFile tag where the character sequence "{P#}" is replaced with an ordinal value representing the property evaluated.

```
<OutFile>WSModel_{P#}.prism</OutFile>
```

The Template tag is for internal use by OMNI and simply gives the name to use for the intermediate file generated for model building. There is no need for the user to amend this tag.

4.3 Delay modelling

If the delay associated with a component is very small then OMNI can be told to omit the delay modelling stage of the refinement process. This can be done in one of two ways. By setting the UseDelay tag you can enable or disable delay modelling for all components:

```
<UseDelay>True</UseDelay>
```

Alternatively a value can be specified such that if the delay associated with a component is less than value specified then delay modelling is omitted for that component only.

```
<DelayCutoff>0.01</DelayCutoff>
```

4.4 Refinement Parameters

The parameters used to control the refinement process are defined in the Criteria block of the config.xml file and are given as:

```
<Criteria>
     <bCountLimit>30</bCountLimit>
     <bLengthLimit>200</bLengthLimit>
     <kStart>2</kStart>
     <kInc>2</kInc>
     <epsilon>0.1</epsilon>
     <staticStepCount>3</staticStepCount>
</Criteria>
```

where:

bCountLimit:	the maximum number of branches for a PHD fitting.
bLengthLimit:	the maximum length of a branch for a PHD fitting.
kStart:	number of branches and branch length for first fitting.
kInc:	how much to increase the size by at each step
epsilon:	the CDF-Difference termination value
staticStepCount:	for how many steps must the improvement be within epsilon.

4.5 Property definition

The Properties section contains one entry per property to be evaluated:

```
<Properties>
```

```
<Property>P=? [true U&lt;=T "close"]</Property>
<Property>P=? [(!"reopen" &amp; !"suspended")
   U&lt;=T "close"]</Property>
</Properties>
```

It should be noted that it is necessary to encode the CSL property string to make it XML safe and hence & becomes & whilst < becomes <

4.6 Component definition

The Components section includes one block per Component to be refined. For each component a Module long name and short name are defined. These are used as identifiers in the input model and the refined PRISM model. In addition the TimingFile node provides the location of the component trace file used to train the PHD model. Hence a complete component block is given as:

```
<Component>
<ModuleName>Traffic</ModuleName>
<ModuleShortName>tra</ModuleShortName>
<TimingFile>tracesWS/traData.csv</TimingFile>
</Component>
```

5 Executing OMNI

Since OMNI makes use of PRISM to classify states within the model we need to define the location of the PRISM library. Having extracted the OMNI archive locate the launch.sh script file.

```
#!/bin/bash
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/local/cap508/Prism/lib/
echo $LD_LIBRARY_PATH
java -cp . -jar OMNI2.jar
```

For Linux you must then change to location of the LD_LIBRARY_PATH variable to match the location where the PRISM library is located.

Then execute the script with the command:

```
./launch.sh
```