

TemLoPAC: Temporal and Logical Pattern Analyzer and Code-generator

INSTALLATION GUIDE AND USERS' MANUAL

Technical Report for NSTIP Funded Project 13-INF761-10

Track - Software Engineering and Innovative Systems

Version 3.1 - September, 2016

Aamir M. Khan

PhD Embedded Systems

Consultant MODEVES Project

NSTIP, Saudi Arabia.

Acknowledgments

This project is partially funded by NSTIP (National Science Technology and Innovative Plan), Saudi Arabia under the Track “Software Engineering and Innovated Systems” bearing the project code “13-INF761-10”. We would like to thank the government of Saudi Arabia especially the Ministry of Science, Technology and Scientific Research for providing us an opportunity to work on the cutting-edge technologies and latest research trends in this domain.

We would like to thank Prof. Dr. Frederic Mallet from University of Nice, Sophia-Antipolis, France for his relentless support for the project. Without his ideas and suggestions, it wouldn't have been possible. We would also like to thank the reviewers of this NSTIP funding and the reviewers from various reputed conferences (like TASE, SIES) and SoSym journal for their valuable comments and suggestions. All their critical remarks helped us evolve the work to its present form.

Contents

1	Setting Up Environment	1
1.1	Background	1
1.2	Setting the UML Environment	2
1.3	TemLoPAC Plugin Installation	6
2	Modeling Patterns and Code Generation	10
2.1	Modeling Graphical Patterns in UML	10
2.2	Transforming into MARTE Models	14
2.3	Generating Observer Code	16
	References	18

Chapter 1

Setting Up Environment

1.1 Background

The implementation approach used for the proposed framework is shown in Figure 1.1. The system modeling can be done in any UML tool but we suggest to use Eclipse-based tools (such as Papyrus UML) as it well incorporates our model transformation plugin. Modeling patterns using state machine diagrams requires Observation profile.

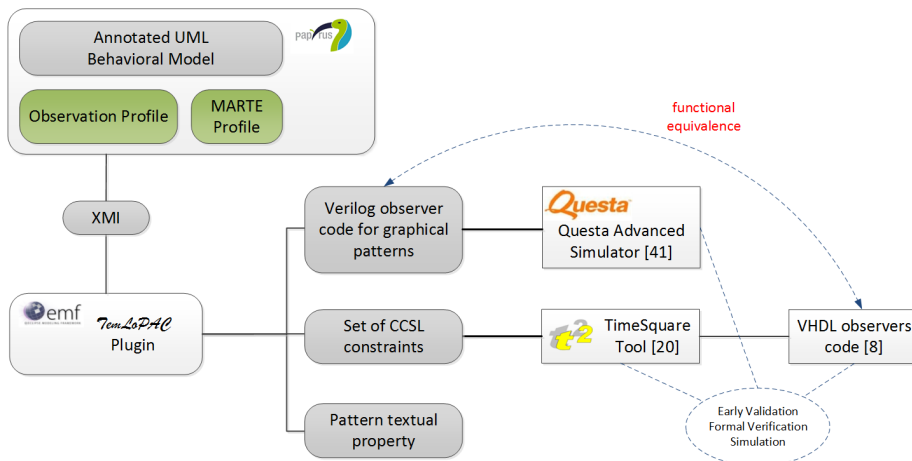


Figure 1.1: Framework Work-flow and Model transformation

We have developed a model transformation plugin named TemLoPAC (Temporal and Logical Pattern Analyzer and Code Generator). It is implemented in Java based on the Eclipse Modeling Framework (EMF) [1, 2]. The model transformation plugin is available online from the project website [3]. The plugin generates three types of code output files: pattern textual property, its observer as a Verilog code implementation, and the equivalent CCSL constraints (as shown in Figure 1.1). The graphical pattern textual code is based on the state relation operators presented in the earlier sections. These operators are the direct textual description of the graphical properties. The generated Verilog code is the observer for these relational operators derived directly from the state operator automata described earlier. Lastly, the transformation plugin also generates the equivalent code in CCSL in the form of CCSL constraints.

Representation of state-based behavior as CCSL constraints provides numerous advantages. CCSL is effectively being used in collaborative industrial projects [4, 5, 6] for verification and validation purposes. The generated CCSL constraints from our TemLoPAC plugin can be used in TimeSquare tool [7] for simulation to find a possible bug in the specification. As the CCSL constraints are available in the design long before the actual implementation, they provide an insight for the key performance metrics. Hence TimeSquare provides an opportunity for early validation of the system behavior and simulation of various possible test cases. Other than simulation, TimeSquare also provides the facility to generate VHDL code [8]. This generated code can be used as observers in physical design implementations. Moreover, researchers have also shown that CCSL specifications can be used to partially generate SystemC code [9].

This two-step code generation (from UML to CCSL to VHDL) is equivalent to our EMF-based code generation directly from the model.

1.2 Setting the UML Environment

To create the temporal properties UML with desired extensions is required. Here are the instructions to setup the computer for the TemLoPAC plugin.

(A) Papyrus UML

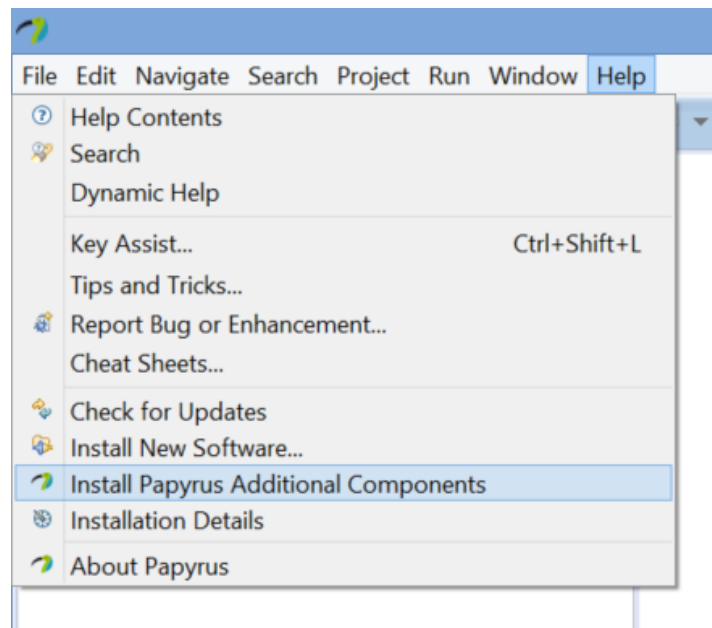
Papyrus is a popular freeware software to model in UML. It can be down-

loaded from the official website. Install the latest version from the link:

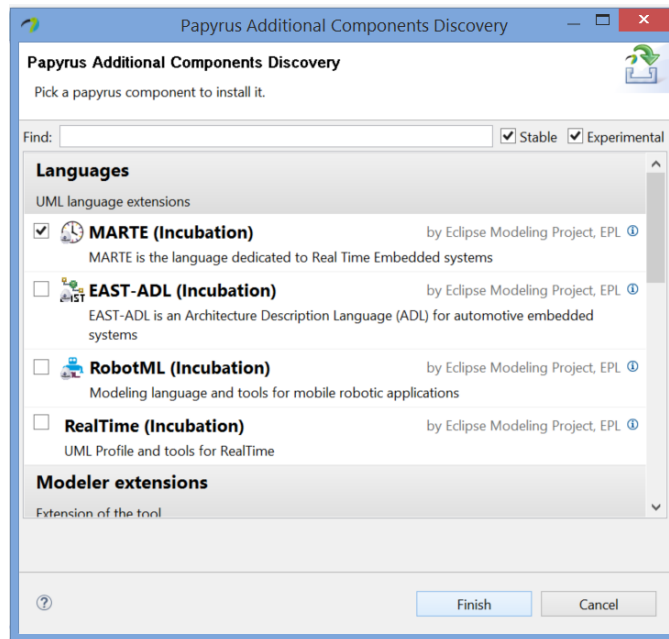
<https://eclipse.org/papyrus/download.html>

(B) MARTE Profile

Once the Papyrus UML is installed, open the software environment and click on Help – > Install Papyrus Additional Components, as shown in the Figure next.



Next, from the list of Papyrus Additional Components Discovery, select MARTE (Incubation) and click finish to install the MARTE profile.

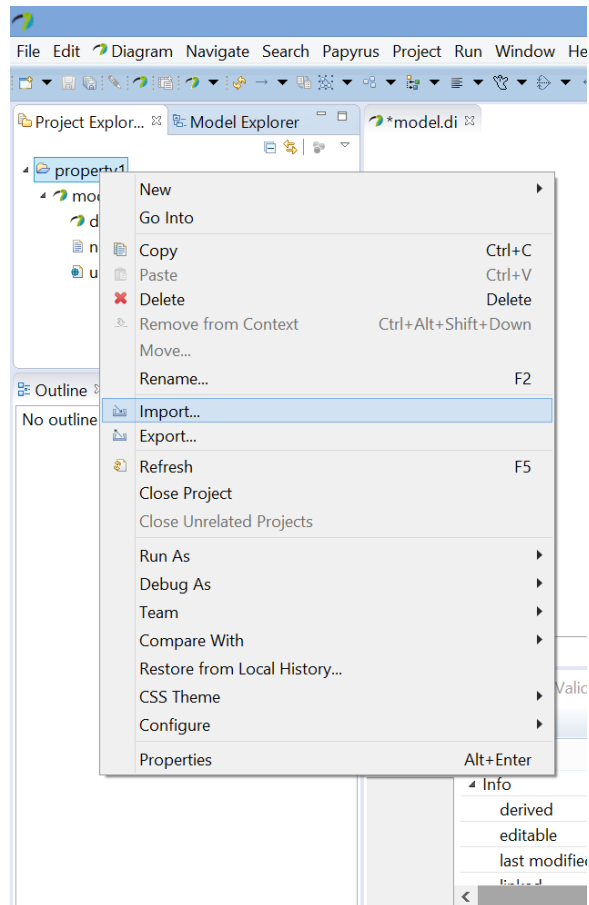


(C) Observation Profile

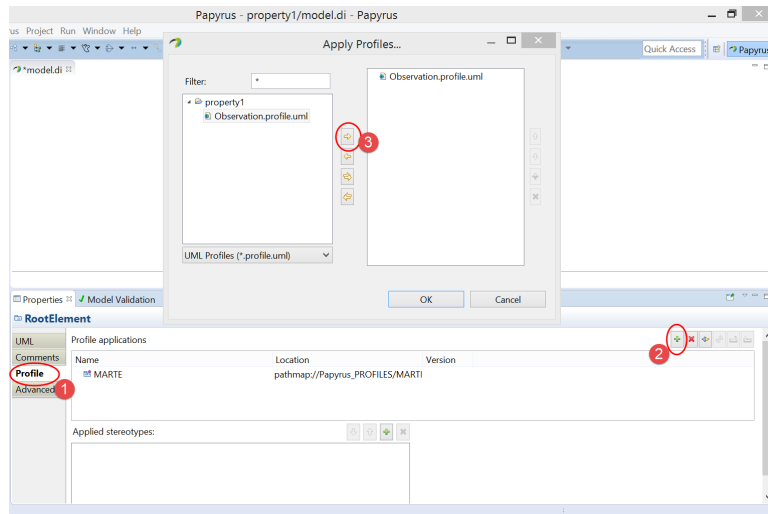
This framework supports to model properties in UML with the help of MARTE and Observation profiles. The Observation profile can be downloaded from the MODEVES project website:

<http://modeves.com/temlopac/Observation.profile.uml>

To install the profile, create a standard Papyrus project from the menu File –> New. In the Project Explorer, right-click on the create project and select import, as shown in the figure.



For the option select an import source, choose the 'File System' from the General folder. Navigate and select the file 'Observation.profile.uml' from the system and click finish. Now the profile is part of the project. Apply it to the model by selecting the RootElement from the 'Model Explorer'. Then select Profile in the properties window and click on Add Profile (plus sign) on the right. Then select the desired profile and click add. All these steps are shown in the figure with the numbers 1, 2 and 3.



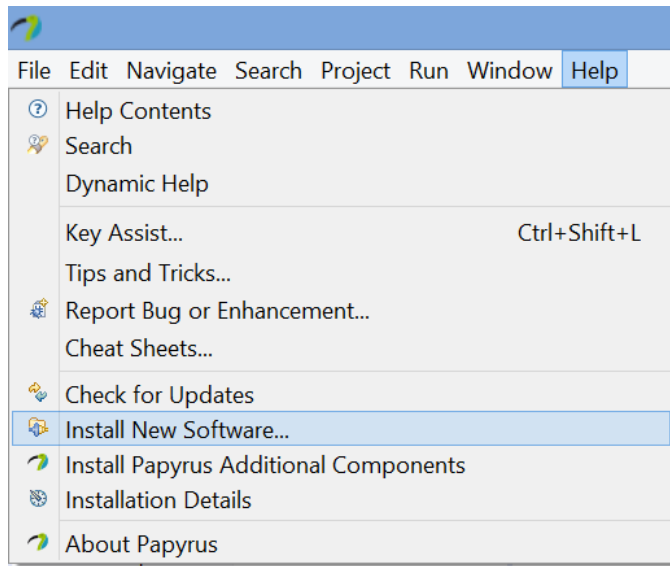
Next use these profiles to create temporal properties as discussed in the text. Some sample temporal properties are given on the site:

<http://modeves.com/temlopac/SampleProperties.zip>

1.3 TemLoPAC Plugin Installation

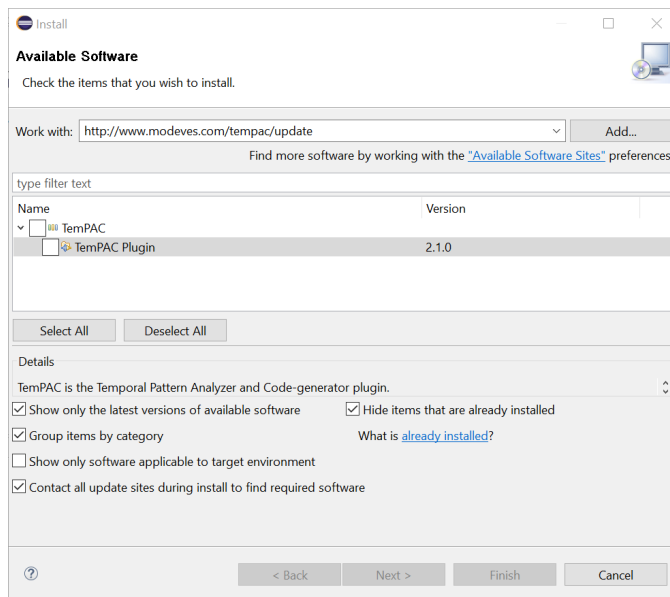
The TemLoPAC plugin is created in Java EMF and hence is well-integrated in the Eclipse environment. It will work well with any Eclipse-based UML editor. There are two way to install the plugin: using update site and through direct download.

The easiest way to install is through the update site. Run the Papyrus software and click the option “Install New Software” from the Help menu, as shown in the Figure next.



On the dialog box next, click on the Add option. The Add Repository dialog window will open. Enter any suitable name for the site (like TemLOPAC Update Site) and in the location field enter the update site address, also shown in the Figure.

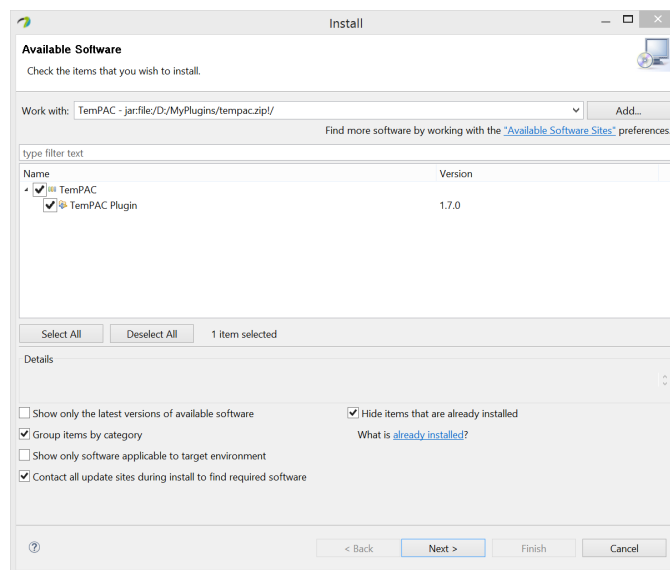
<http://www.modeves.com/temlopac/update>



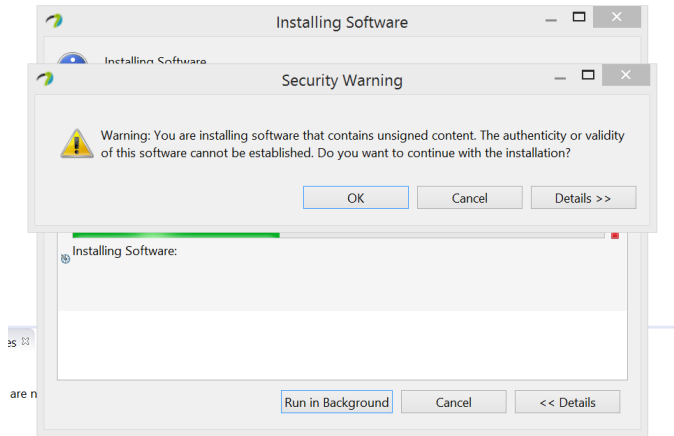
The plugin can also be downloaded as a compressed .zip file from the project website directly and installed manually.

<http://modeves.com/temlopac/temlopac.zip>

Once the plugin is downloaded, save it at some common place on the computer (like desktop). Back on the 'Add Repository' window, click on the Archive button and select the TemLoPAC zip file downloaded from the project site earlier. Click OK. Afterwards, in the available software window select the TemLoPAC plugin and click next.



On the next screen Papyrus will inform about the installation details as shown in the figure. Again just click next. Lastly, the license agreement is shown, just click agree and OK to install the plugin in the Papyrus environment. During the installation, eclipse will warn that the software being installed contains unsigned content, as shown in the Figure. Just ignore the warning and click OK.

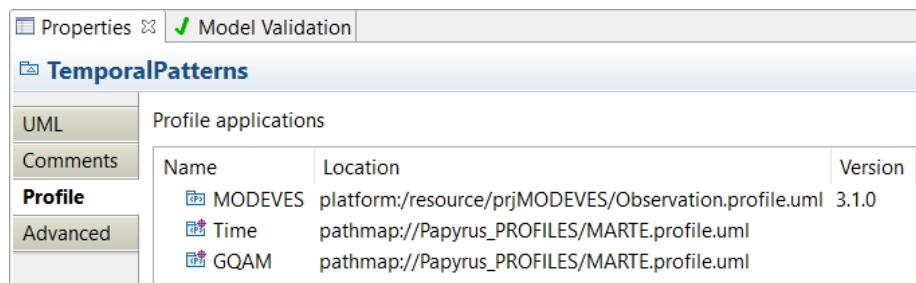


Chapter 2

Modeling Patterns and Code Generation

2.1 Modeling Graphical Patterns in UML

Before modeling the patterns in UML, make sure MARTE and Observation profiles are imported into the Papyrus tool environment as explained in the last chapter. Once we start modeling patterns, the top level model or the package of the patterns need to have the profile application of *Time* and *Generic Quantitative Analysis Modeling (GQAM)* sub-profiles installed from the MARTE. Moreover, our custom *Observation* profile should also be installed to the package/model, as shown in the figure next.

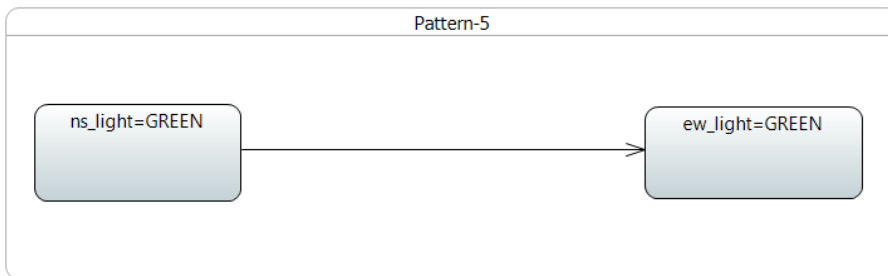


The screenshot shows the 'Properties' window in Papyrus, with the 'Model Validation' status bar at the top indicating a successful check. The main content area is titled 'TemporalPatterns' and displays a table of 'Profile applications'.

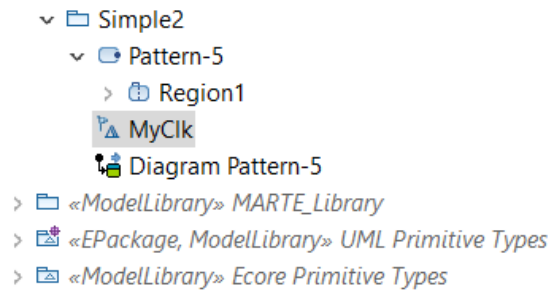
Name	Location	Version
MODEVES	platform:/resource/prjMODEVES/Observation.profile.uml	3.1.0
Time	pathmap://Papyrus_PROFILES/MARTE.profile.uml	
GQAM	pathmap://Papyrus_PROFILES/MARTE.profile.uml	

Modeling system **state-state behavior** is done using state machine diagrams. The idea behind these patterns is to have a behavior package in the (structural) model that has set of state machine diagrams. Each state machine represents one such pattern which is easy to implement and verifies one system constraint. Each state machine consists of a simple structure of two states and a transition between them. State names need to be meaningful (but not required) such as `ns_light=RED` etc. This simple modeling will give us a basic structure shown next. Here as an example, we model the exclusion constraint:

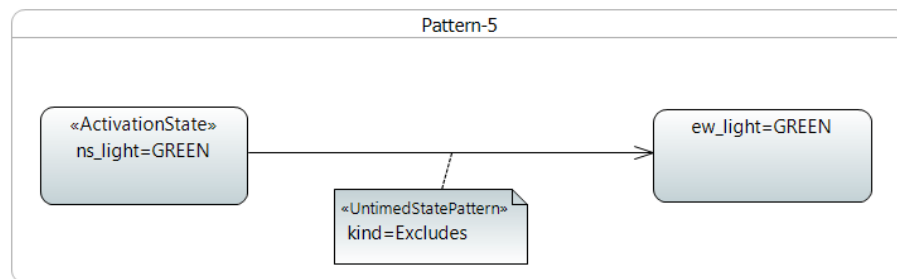
`ns_light=GREEN` **excludes** `ew_light=GREEN`



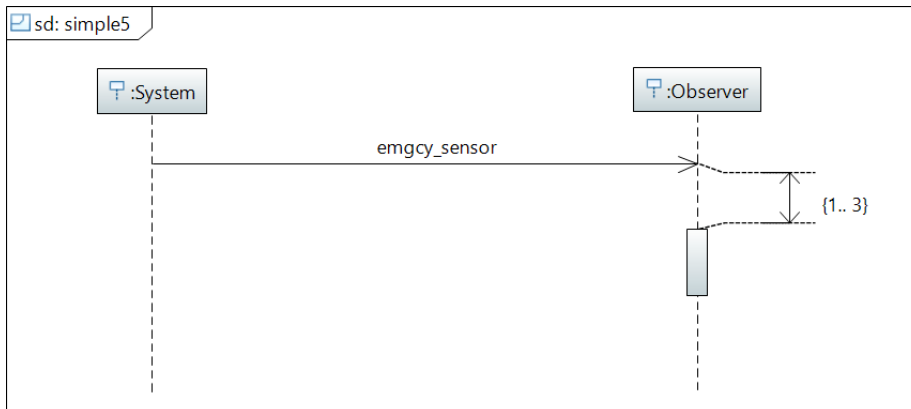
Next step is to apply the *Observation profile* to use. The *ActivationState* stereotype is used on the first state of the state machine (`ns_light=GREEN` in this case). In many state patterns (like in the given example of exclusion pattern) the ordering of the states doesn't matter at all. State transition is applied with the appropriate stereotype *TimedStatePattern* or *UntimedStatePattern*. In case of untimed state patterns, user specifies the *kind* of the pattern selecting from the list. In case of timed state patterns, user also needs to specify the duration limits minimum (min) and maximum (max). Moreover using the *on* tagged value of *TimedStatePattern*, the user also needs to specify the base event (clock) on which these intervals are counted. A *ChangeEvent* element can be created in the parent package of the state machine to represent as a clock signal, as shown in the figure next.



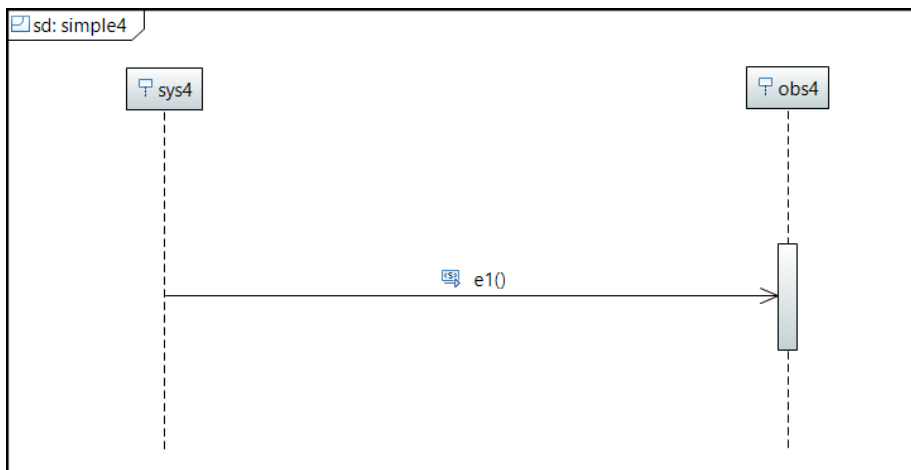
Note that this is the only major effort required by the design/verification engineer to model one single graphical pattern. Rest of the things are mostly done automatically. After the application of Observation profile, the pattern is shown in the figure next.



Modeling system **state-event behavior** is done using sequence diagrams. Two classes are defined at the root/package level to represent the *System* and *Observer*, later used as lifelines in the sequence diagram. Moreover, a signal is also defined to represent the asynchronous message passing. In this type of patterns, the event is always displayed as an asynchronous message call between the lifelines and the state is shown as UML artifact *Behavior Execution Specification*. Unlike the state machines, the duration in the timed patterns is not specified in the stereotypes but is shown graphically using *DurationConstraint* formalism. Note that the two ends of the duration constraint must be connected properly to the *message occurrence specification* and behavior's *execution occurrence specification*. The model for the property “emgcy_sensor” is shown in the figure next.

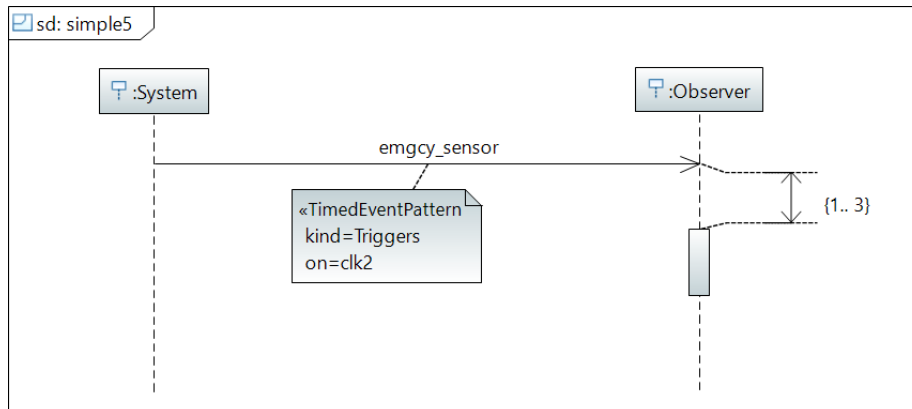


To model untimed patterns, we don't use the duration constraint and the asynchronous message directly connects the behavior execution specification representing the state. This connection must be done between the start and the end execution occurrence specification, as shown in the figure next.



Here one important thing to mention is a bug in Papyrus UML. The *Observer* lifeline contains the track of sequence of occurrences using the property *Covered By*. When the model is initially created, this property contains all the names in proper ordering like *e1*, *AStart*, *AFinish* etc. But due to some bug in Papyrus, once the software is restarted, this information is partially deleted or messed-up.

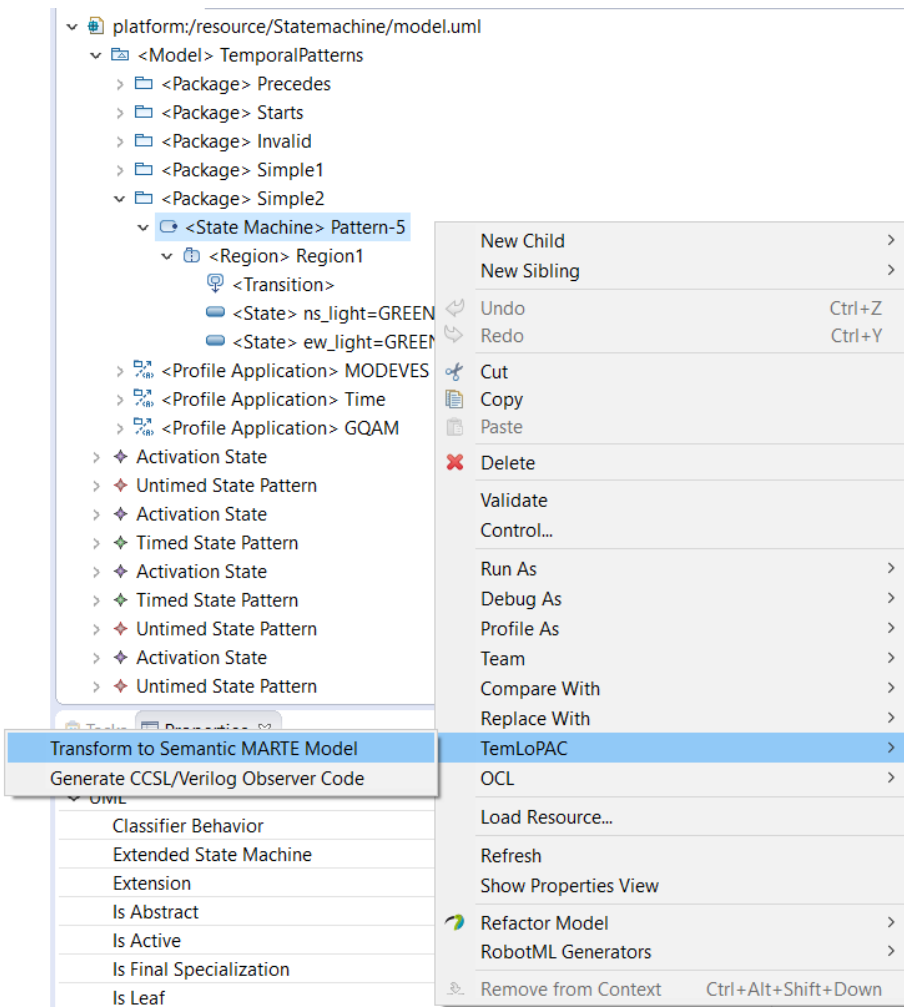
So while creating models, this thing has to be considered carefully especially for untimed models as our TemLoPAC software will generate error if this ordering is not correct. Once the modeling is complete, the Observation profile is applied. The *TimedEventPattern* is applied to the asynchronous message call. The kind of the message and the base clock event are also required. This is the amount of effort required by the user, as shown in the figure next.



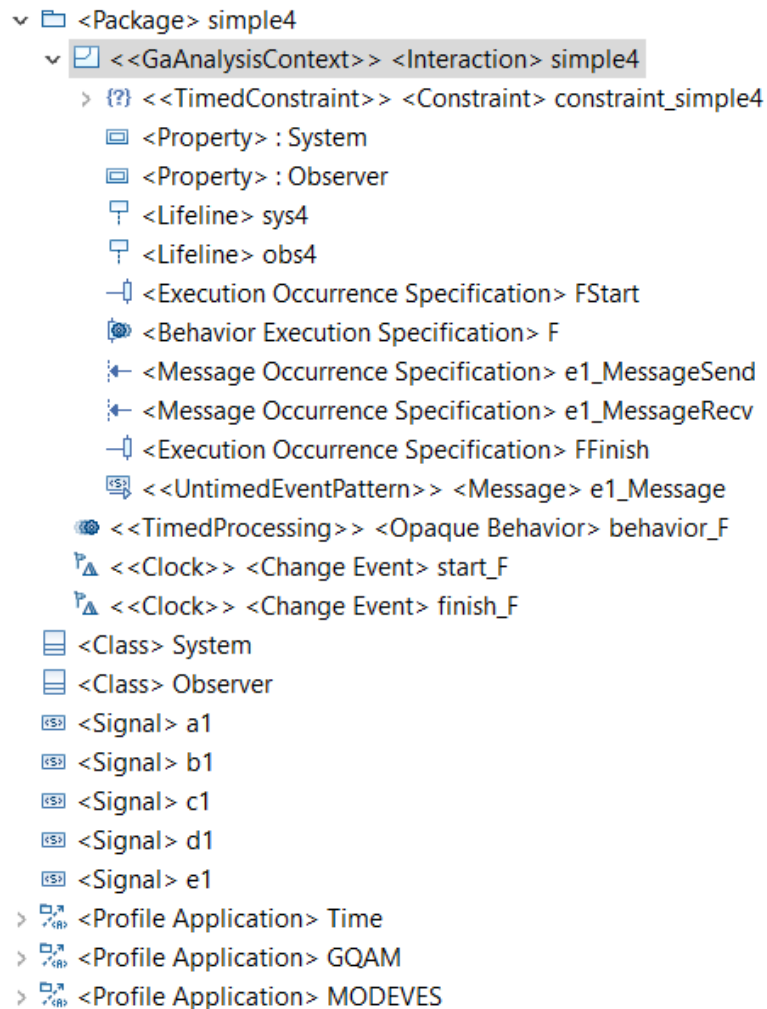
2.2 Transforming into MARTE Models

Once graphically modeling the property is done, open the UML model from the *Project Explorer* using the *UML Model Editor*. This will give us the pure UML model in the tree structure. From the tree structure, we have to right-click the <State Machine> or the <Interaction> element. As shown in the figure next, from the right-click menu select,

TemLoPAC – > Transform to Semantic MARTE Model,



This will add all the required features of MARTE into the UML model including stereotypes like *AnalysisContext*, *TimedProcessing*, *TimedConstraint*, *Clock* and so on. One beautiful thing about the TemLoPAC software is that it tries only to fill the missing information and if some stereotype or information is already present, it will just skip it. The semantic MARTE model of a pattern will look something like in the figure next.



2.3 Generating Observer Code

Once properties are rightly transformed to semantic MARTE models or directly created correctly, they can be use to generate CCSL and SystemVerilog code. TemLoPAC plugin only works on the State machines and Interaction diagrams. From the right-click menu in the Eclipse environment select,

TemLoPAC – > Generate CCSL/Verilog Observer Code.

If there is any error in property, there will be some hint given in the message

regarding the type of the error. Else if the property is valid, the message “*CCSL/Verilog Observer Code Successfully Generated!*” will be displayed. Three new files are created in the installation directory of the Papyrus: CCSLPlus.txt, CCSL.txt and SystemVerilog.sv.

References

- [1] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0*, 2nd ed. Addison-Wesley Professional, 2009.
- [2] The Eclipse Foundation. Eclipse Modeling Framework (EMF). [Online]. Available: <http://www.eclipse.org/modeling/emf/>
- [3] A. M. Khan, “TemLoPAC: Temporal and Logical Pattern Analyzer and Code-generator EMF Plugin.” [Online]. Available: <http://www.modeves.com/temlopac.html>
- [4] J. Suryadevara, G. Sapienza, C. Seceleanu, T. Seceleanu, S.-E. Ellevseth, and P. Pettersson, *Wind Turbine System: An Industrial Case Study in Formal Modeling and Verification*. Cham: Springer International Publishing, 2014, pp. 229–245. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-05416-2_15
- [5] M. A. Peraldi-Frati and J. DeAntoni, “Scheduling Multi-Clock Real-time Systems: From Requirements to Implementation,” in *2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, March 2011, pp. 50–57.
- [6] J. Suryadevara, C. Seceleanu, F. Mallet, and P. Pettersson, *Verifying MARTE/CCSL Mode Behaviors Using UPPAAL*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–15. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-40561-7_1
- [7] J. Deantoni. TimeSquare: Logical Time Matters. [Online]. Available: <http://timesquare.inria.fr/>

- [8] C. André, F. Mallet, and J. DeAntoni, “VHDL Observers for Clock Constraint Checking,” in *Industrial Embedded Systems (SIES), 2010 International Symposium on*, July 2010, pp. 98–107.
- [9] J. Peters, R. Wille, and R. Drechsler, “Generating SystemC Implementations for Clock Constraints Specified in UML/MARTE CCSL,” in *Engineering of Complex Computer Systems (ICECCS), 2014 19th International Conference on*, Aug 2014, pp. 116–125.