CoWolf - A Generic Framework for Multi-View Co-Evolution and Evaluation of Models - Tool Paper

Sinem Getir[1], Lars Grunske[1],
Christian Karl Bernasko[2], Verena Käfer[2], Tim Sanwald[2]

[1]Reliable Software Systems, University of Stuttgart, Germany
{sinem.getir, lars.grunske}@informatik.uni-stuttgart.de
[2]University of Stuttgart, Germany
{st106732, swt74174, swt80243}@stud.uni-stuttgart.de

Abstract. Agile and iterative development with changing requirements lead to continuously changing models. In particular, the researchers are faced with the problem of consistently co-evolving different views of a model-based system. Whenever one model undergoes changes, corresponding models should co-evolve with respect to this change. On the other hand, domain engineers are faced with the huge challenge to find proper co-evolution rules which can be finally used to assist developers in the co-evolution process. In this paper, we introduce the CoWolf framework that enables co-evolution actions between related models and provides a tooling environment. Furthermore, we demonstrate the results of a case study on the developed tool.

Keywords: Model evolution, multi-view modeling, model co-evolution, model synchronization, model differencing, quality of service models

1 Introduction

Models are a great aid to reduce the complexity of a software system so that analysis tools and humans can conceive it. Commonly, great parts of the program code are generated from domain specific models and analysis on performance, reliability and safety are completely done on separate models. Consequently, it is desirable to split the information into different models that are all specialized for a specific task allowing a well-founded theory on analysis methods and a rich tool infrastructure. This leads to the problem of co-evolving these different models, to keep them consistent when one of the models evolves over time. A solution to these problems is seen in incremental model transformation and synchronization, a process that identifies changes done to a source model and which translates only these changes to the target models.

The CoWolf tool presented in this paper delivers a framework for model development, transformation and analysis, implementing the idea of incremental model transformation. In CoWolf incremental transformation isolates changes that were done to a model to selectively propagate only these changes to the other models. In detail the contribution of the CoWolf tool comprises mainly two aspects:

- **The co-evolution of an associated model on the basis of evolutions.** As described, models may have to be updated if other models changed. Often, those updates can be described canonically. CoWolf features the definition of rules that define the relation between model types. Using these rules, co-evolutions can be done (semi)-automatically for all associated models.
- **Deliver utilities for model development and analysis.** For consistent development of the models, CoWolf provides a common environment with graphical and textual editors. Furthermore, it implements interfaces to external tools to analyse models.

2 CoWolf Framework

CoWolf is an extensible framework for model evolution and co-evolution management and it has mainly two goals. First goal is conducting the co-evolution process when one of the corresponding models undergoes changes. We denote these corresponding models as *couples*. Currently CoWolf supports seven different models: state charts, component diagrams and sequence diagrams as architectural models; and discrete time markov chains (DTMC), continuous time markov chains (CTMC), fault trees, layered queuing networks (LQN) as QoS models. We use Henshin graph transformations [1] to accomplish the co-evolution process. Implemented transformations and their directions between couples from architectural and QoS models are presented in Figure 1 (e.g. DTMC-CTMC or CTMC-fault tree are denoted as couples). As we observe, while there exist bidirectional transformations between state charts and DTMCs, there exist unidirectional transformations from component diagram to fault trees.

Second goal is delivering utilities for the model driven development and direct support for model analysis. During the continuous development of the models, CoWolf provides a common and user friendly environment with textual (Eclipse Xtext) and graphical editors (Eclipse Sirius) for different model types. Furthermore, it establishes interfaces to external tools to analyse models. In Figure 1, we display the integrated model solvers to the corresponding QoS models. The tooling environment is enriched with the textual editors to represent the verification properties. Following that the developer can send the model to the analyser with one button.

CoWolf is an Eclipse plug-in and designed to be highly extensible for new models. We demonstrate the architecture of the tool and used technologies in Figure 2. In the following, we expand on the working principle of the CoWolf framework and illustrate with the *Stop Watch* example.

2.1 Co-evolution with CoWolf

In the following, we explain the transformation and model difference process in the background of CoWolf, and demonstrate the evolution of example afterwards.

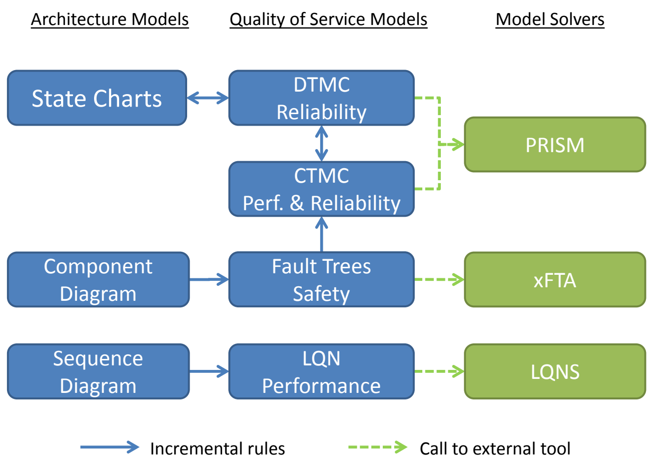


Fig. 1. Couples, transformations and model solvers supported by CoWolf

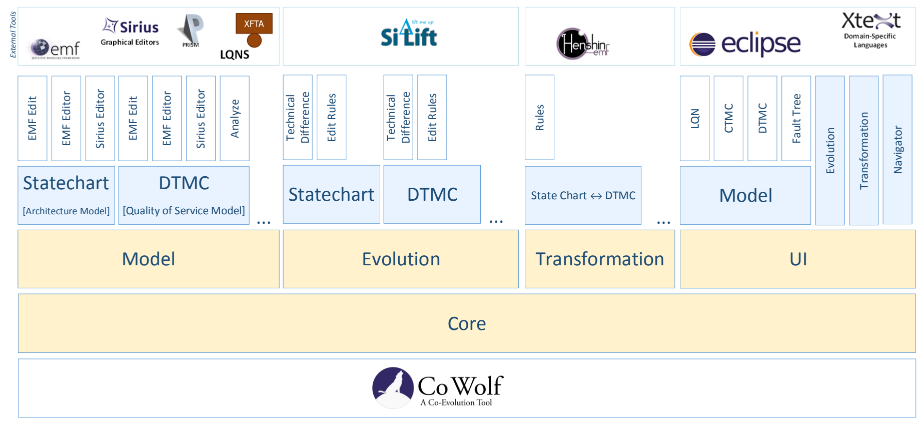


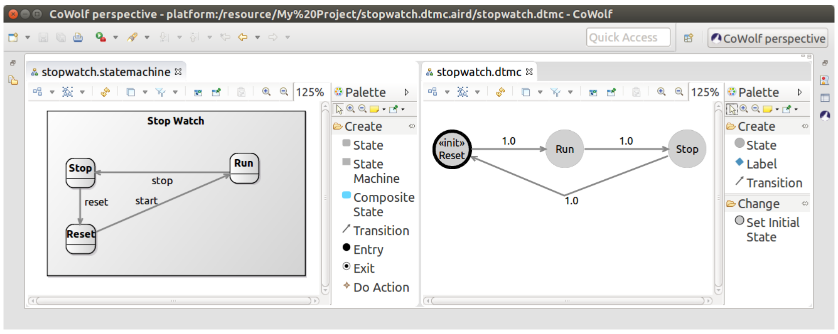
Fig. 2. CoWolf architecture

Transformation process After identifying coupled models, we described Henshin rules between the couples and the rules for the single models. While the Henshin rules can be both manually created and auto-generated in the SiLift [6] environment for a single model, the co-evolution rules should be created only manually since it requires mapping between the coupled model elements. Every co-evolution transformation is performed from a source model to a target model, which exposits the co-evolution direction. Defining the co-evolution transformations between the related models is not an easy task and requires domain knowledge. On the other hand, the effort describing the transformations differ from *couple* to *couple*. For example, transforming a state chart to a DTMC can be performed with one to one (assuming that we omit composite states) mapping considering the structure of the models. As another example, transforming CTMC and DTMC requires the normalization of the parameters only. However, transformations are not straight forward between fault trees and component diagrams [4]. As a result, CoWolf does not claim fully automatic and complete transformations, but aims the utilization of the co-evolution process with user interaction.

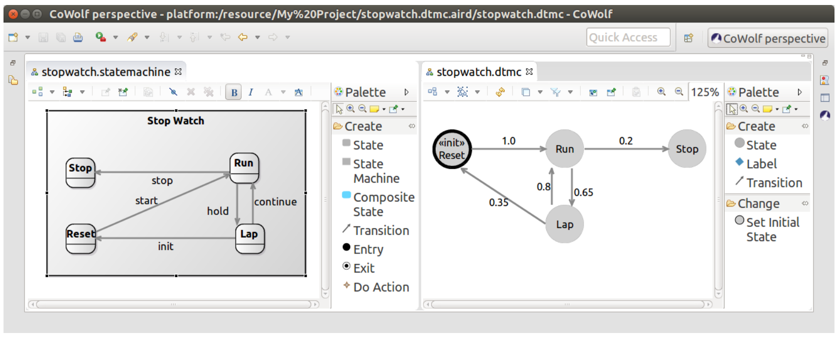
When the user wants to apply the co-evolution between couples, the changes between the current version and the last version are calculated for the source model. If it is the first co-evolution between the models, the differences between an empty model and the current model are calculated. We use SiLift [6] for the model difference calculation. After the calculation, we perform the corresponding changes to the target model to accomplish the co-evolution. Note that there has to be a full set of rules for every possible change (predefined) in the source model to do a co-evolution. SiLift produces the difference output in the representation of Henshin rules, which makes the co-evolution process applicable in our framework. After the changes were detected, the rules can be applied and the target model can be co-evolved. There is a high amount of work in the background process. We refer the interested readers to the website `http://cowolf.github.io/` for the details and the source code.

Running example We demonstrate a running example called *Stop Watch* in Figure 2.1 in CoWolf's graphical editor, which enables a drag and drop facility from the menu. The source model is a state chart and initially has three states with three transition. When the user wants to apply a co-evolution, it is possible to have several target models for one source model. For instance in the menu, the user can select DTMC, CTMC as couple models of a state chart. In Figure 3(a), assuming that the target model is selected as DTMC, we display the DTMC model after the first co-evolution action (complete transformations are applied). After the first co-evolution, the two models are now connected and if a change happens in the initial model, an out-of-date-warning is shown for the target model. At some point of time, the state chart evolves as shown in Figure 3(b). A new state *Lap* and its transitions are added to the watch system, and one transition is deleted. The changes are calculated by SiLift, whose output is also visible in CoWolf environment by the user when requested. Based on the corresponding changes, the DTMC co-evolves with incremental transformations. As shown in 3(b), the applied transformations generate exactly the same structure as in the state chart. On the other side, the model is incomplete because of the parameters, therefore the user interaction is needed for valid models.

Extending CoWolf CoWolf is an extensible MDD framework for new types of models with its flexible architecture (Figure 2). For this, the developer needs to provide four artifacts: 1) Metamodels of the coupled models, 2) Henshin rules for the single models to detect changes between two instances (manually created or auto-generated in SiLift environment). 3) Henshin rules for the co-evolutions and 4) a GUI. We refer the readers to `https://github.com/CoWolf/CoWolf` for the details of the architecture.



(a) The initial state chart and the co-evolved DTMC



(b) The state chart and DTMC model after the second co-evolution

Fig. 3. A co-evolution from state chart to DTMC in CoWolf

2.2 Integrated Model Solvers

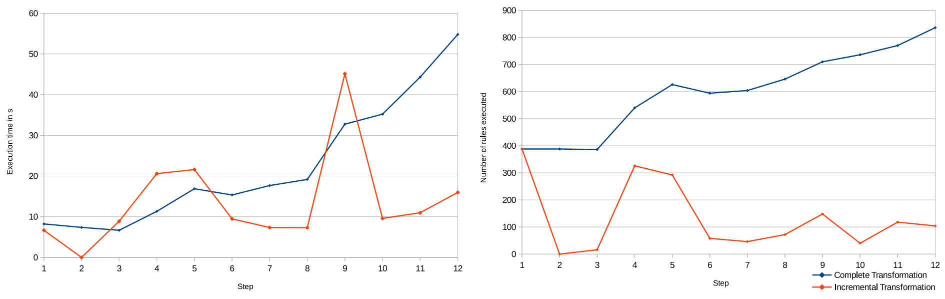
Besides the incremental transformation of models, CoWolf is also capable of measuring quality aspects of models. For this, we implemented a user friendly interface to the external solvers for the corresponding QoS model. As presented in Figure 1, CoWolf supports the models solvers for DTMC, CTMC, LQN and fault trees via PRISM [7], LQNSolver [2] and Xfta [9] respectively. The solvers produce analysis results (e.g. measurement and prediction) for systems' availability, performance and safety attributes. With this feature of CoWolf, developers do not have to fully understand the modelling language of the external model solvers (e.g Prism grammar or Open-PSA script in Xfta) and can run the analysis from CoWolf directly. The developer then only needs to set the properties and trigger the analysis button on the selected model. However, the analysis steps differ from model to model. For example, a fault tree analysis is always performed on the top event in the model. On the other hand, a CTMC model requires property description in PCTL. CoWolf produces a solution for this and enables the user to write the properties in an editor whose design was inspired by ProProST tool [5]. Hence, our Xtext editor for the properties is capable of generating the full PCTL.

In the background, whenever the analysis is triggered, CoWolf transforms the model to the language supported by the solver and runs the evaluation with the selected tool. Afterwards CoWolf receives and parses the results from the tools and presents them in Eclipse as an extra view. When requested, exporting the models in the language of the external tools is also possible. As a result, this feature enables users to benefit the full functionality of the solvers.

3 Evaluation

We evaluate the CoWolf framework on a standard automation case study called Pick&Place Unit (PPU) [8]. The Pick&Place unit has four main components: storage, crane, stamp and sorter, which stores, conveys, processes and sorts the work pieces on the platform respectively. The system has 14 predefined evolution steps and all the steps have different affects on various types of models. We perform co-evolution actions between state charts and corresponding discrete time markov chains (DTMC) and compare the incremental transformations, which are executed with co-evolution process, and complete transformations as demonstrated in Figure 4.

In Figure 4(a) we present the comparison in terms of the execution time. In general co-evolution actions are faster than the execution of complete transformation. However at steps such as 4,5 and 9, the co-evolution process takes much longer than the complete transformation. The reason for this is the calculation of the difference between the models in addition to the execution of the incremental transformations. We observe in the evolution steps that the changes between 3-5 and 8-9 are much bigger compared to the other steps. As aforementioned, we use an external tool (SiLift) to calculate the diffs between the models. Therefore, we provide the second evaluation by only evaluating the number of rules executed with incremental transformations in Figure 4(b) to support this argument. The number of rules for incremental transformation is apparently much less than the number of complete rule executions as expected.



(a) Execution time (b) Executed number of rules

Fig. 4. Performance comparison between incremental and complete transformations for each step

4 Conclusion

Domain engineers are faced with big challenges to manage co-evolution in multi-view model based systems. In this paper, we have introduced an extensible framework for co-evolution and model analysis to assist the developers. CoWolf is an open source project for the community and extensible for any kind of model. Since it is generic, plug-in based and includes SiLift, we would like to integrate a co-evolution analysis [3] to improve the co-evolution actions between the models as a future work.

Acknowledgments. This work is supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future - Managed Software Evolution. The authors would like to thank Christian Karl Bernasko, Manuel Borja, Verena Käfer, David Krauss, Michael Müller, Philipp Niethammer, Tim Sanwald, Jonas Scheurich, David Steinhart, Rene Trefft, Johannes Wolf and Michael Zimmermann for their great work in the CoWolf development.

References

1. Arendt, T., Biermann, E., Jurack, S., Krause, C., Taentzer, G.: Henshin: advanced concepts and tools for in-place emf model transformations. In: Intl. Conf. on Model Driven Engineering Languages and Systems, pp. 121–135 (2010)
2. Franks, G., Maly, P., Woodside, M., Petriu, D.C., Hubbard, A.: Layered queueing network solver and simulator user manual. Dept. of Systems and Computer Engineering, Carleton University (2005)
3. Getir, S., Rindt, M., Kehrer, T.: A generic framework for analyzing model co-evolution. In: Model Evolution, International Conference on Model Driven Engineering Languages and Systems (2014)
4. Getir, S., Van Hoorn, A., Grunske, L., Tichy, M.: Co-evolution of software architecture and fault tree models: An explorative case study on a pick and place factory automation system. In: Intl. Workshop on Non-functional Properties in Modeling: Analysis, Languages, Processes. pp. 32–40 (2013)
5. Grunske, L.: Specification patterns for probabilistic quality properties. In: Proc. of ICSE, 2008. pp. 31–40 (2008)
6. Kehrer, T., Kelter, U., Taentzer, G.: A rule-based approach to the semantic lifting of model differences in the context of model versioning. In: Intl. Conf. on Automated Software Engineering. pp. 163–172 (2011)
7. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) Proc. of International Conference on Computer Aided Verification. LNCS, vol. 6806, pp. 585–591. Springer (2011)
8. Legat, C., Folmer, J., Vogel-Heuser, B.: Evolution in industrial plant automation: A case study. In: Proc. of IECON 2013. IEEE (2013)
9. Rauzy, A.: Anatomy of an efficient fault tree assessment engine. In: Virolainen, R. (ed.) Proceedings of PSAM'11/ESREL'12 (June 2012)

A Demonstration of CoWolf

CoWolf is an eclipse plug-in and has some pre-installation requirements such as Henshin, SiLift and Sirius in Eclipse EMF environment. The following instructions will describe how to set up a CoWolf project, create a model, co-evolve models incrementally and analyse a model with an external model solver. Please visit `www.youtube.com/channel/UCCLr7fwKSqJTkeekTa3YO9Q` to see the tutorial video.

A.1 How to Create a Project

To create a *"CoWolf project"*, we go to File⇒New⇒Other. In the wizard window, we select CoWolf Project, specify a project name and complete the project creation. To benefit from the functionality of CoWolf, maintaining the CoWolf perspective is important, because the project in the project explorer is only visible in this perspective.

A.2 How to Create a Model

We select the predefined *"models"* folder from the project folder to create a model. The context menu with a right click will be opened. Afterwards, selecting the New⇒ Other sub-entry of the context menu opens the wizard window. We find the sub folder *"Models"* of the CoWolf entry. Our goal in this tutorial is to create a state chart representation of a *Stop Watch* shown in Figure 5. For this example, we select the state chart model, specify a name and click finish to create the model. The project should now contain two files, one with the extension *"aird"* and the other with the extension *"statechart"*. The file with the extension *statechart* represents the model as a tree view, while the extension *"aird"* is a graphical representation of the model. Double click on the *"*.aird"* file opens the graphical representation that will bootstrap the graphical view. You should now see a little triangle which indicates that this entry has now a folder structure. We navigate to *"Representations per category"* and select the innermost file in this nested folder.

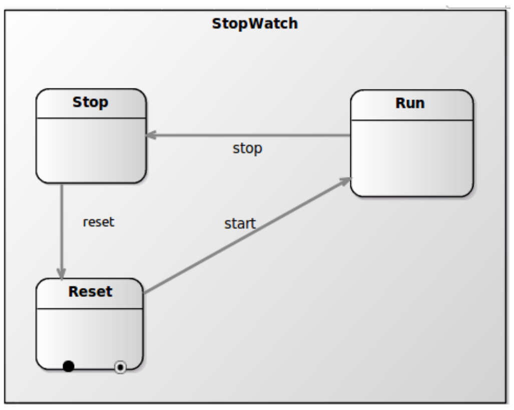


Fig. 5. State chart stop watch example.

A.3 How to Co-Evolve a Model

Our next goal is to co-evolve the state chart model with a DTMC model. After creating the state chart model, we want to obtain a DTMC representation of *Stop Watch*. A complete transformation from state chart to a DTMC will be performed, since this is the first co-evolution step. For the first co-evolution, an empty DTMC model is required. After the empty DTMC model was created, we select the state chart model the sub-entry *"Co-Evolve"* of the *"CoWolf"* entry. The project wizard will then list the models that can be used for the co-evolution. We choose the DTMC model and click finish to start the co-evolution process between these two models. After the co-evolution process is completed, we need to specify model parameters such as transition probabilities manually for the DTMC model. Otherwise, the model will not be valid as demonstrated in Figure 6 for the generated DTMC.

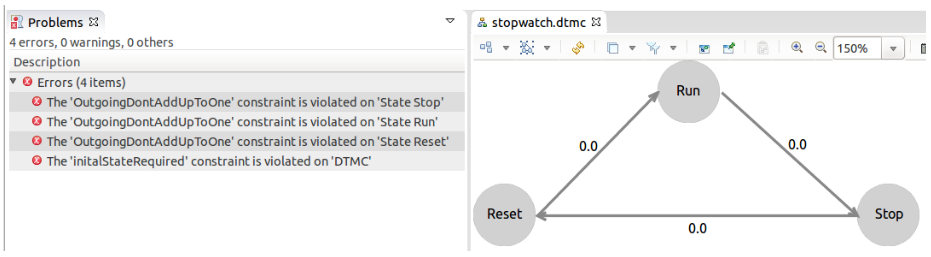
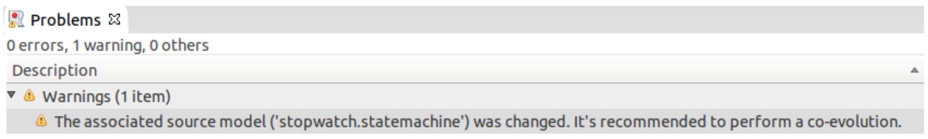

Problems
4 errors, 0 warnings, 0 others
Description
Errors (4 items)
The 'OutgoingDontAddUpToOne' constraint is violated on 'State Stop'
The 'OutgoingDontAddUpToOne' constraint is violated on 'State Run'
The 'OutgoingDontAddUpToOne' constraint is violated on 'State Reset'
The 'initalStateRequired' constraint is violated on 'DTMC'

Fig. 6. Validation result and the generated DTMC

As a second co-evolution step, we want to perform an incremental co-evolution when the state chart model is modified e.g. by adding a state and transitions. After saving the changes, CoWolf indicates in the problems tab of the Eclipse environment that the DTMC model is out of date. To update it, we can perform an incremental co-evolution step by co-evolving the state chart model again with the DTMC model. The co-evolution process will now only add the missing changes to the DTMC model and not perform a complete transformation of the whole state chart model. We display the out of date warning when the state chart is modified and generated models after the second co-evolution step in Figure 7(a) and in Figure 7(b) respectively.

A.4 Analysing a Model

In this tutorial, we use the probabilistic model checker Prism, to analyse the DTMC model. To be able to access the Prism analyser with the CoWolf interface, we need to specify the path where the Prism .exe file is, in the settings configuration. For this, we go to Window⇒ Performance⇒ CoWolf⇒ Models⇒ DTMC and specify the path to the Prism root directory. To perform an analysis on a DTMC model, we use the context menu and select the sub entry *"Analyse"* of the CoWolf entry. Then the wizard will allow us to choose between analyse and verify. Analysis/Verification via Prism can be performed in two ways. 1) reachability analysis that requires state selection in the wizard. 2) reliability/performance analysis with more complex properties in PCTL(Probabilistic Computational Tree Logic) that requires property specification

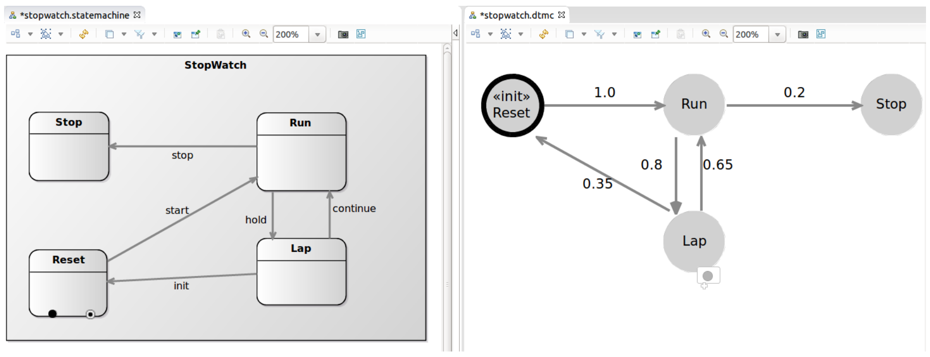


Problems ☒

0 errors, 1 warning, 0 others

Description
▼ ⚠ Warnings (1 item)
⚠ The associated source model ('stopwatch.statemachine') was changed. It's recommended to perform a co-evolution.

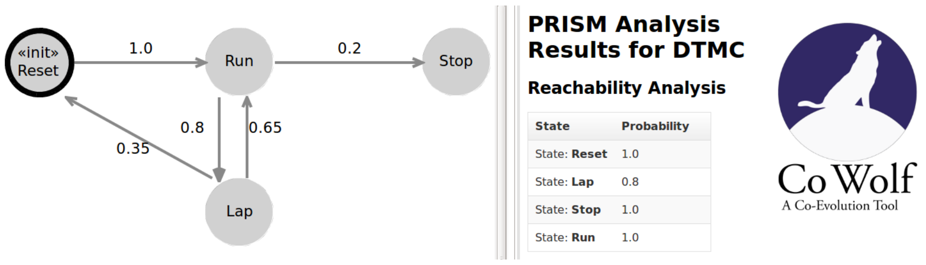
(a) Out-of-date warning after the evolution step for the state chart



(b) Coupled models after one evolution step

Fig. 7. Incremental co-evolution between a state chart and DTMC model

together with state/label selection (See Figure 9). Afterwards we can click finish to compute the analysis on the valid models and obtain the results in a specific CoWolf view. In this example, we compute a reachability analysis on the co-evolved DTMC model and demonstrate the analysis result in Cowolf view in Figure 8.

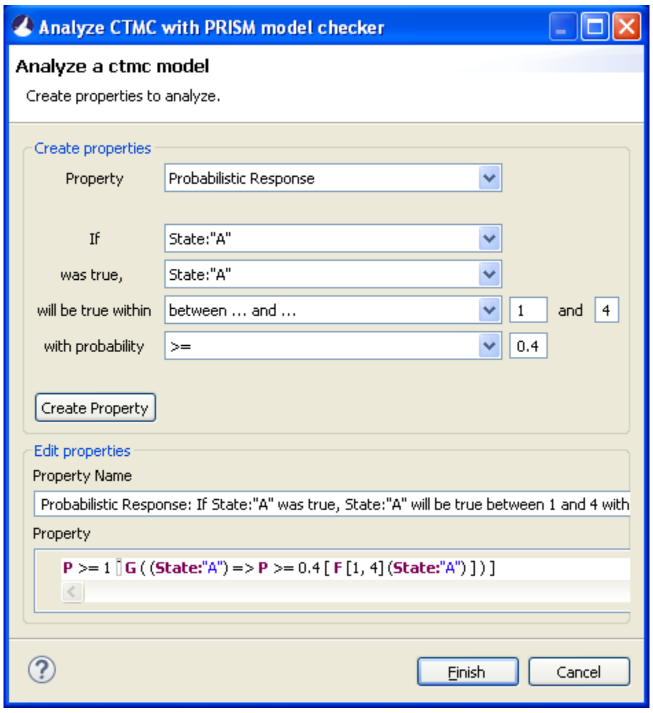


PRISM Analysis Results for DTMC

Reachability Analysis

State	Probability
State: **Reset**	1.0
State: **Lap**	0.8
State: **Stop**	1.0
State: **Run**	1.0

Fig. 8. DTMC analysis results in CoWolf view



Analyze a ctmc model

Create properties to analyze.

Property	Probabilistic Response
If	State:"A"
was true,	State:"A"
will be true within	between … and … 1 and 4
with probability	>= 0.4

Property Name

Probabilistic Response: If State:"A" was true, State:"A" will be true between 1 and 4 with

Property

```
P >= 1 [ G ( (State:"A") => P >= 0.4 [ F [1, 4] (State:"A") ] ) ]
```

Fig. 9. PCTL property specification with CoWolf textual editor