# PANDA-Æmilia: an Eclipse-based tool used to automatically detect performance antipatterns in Æmilia Architecture Description Language

Martina De Sanctis[1], Catia Trubiani[2], Vittorio Cortellessa[3], Antinisca Di Marco[3], and Mirko Flamminj[3]

[1] Fondazione Bruno Kessler, Trento, Italy,
msanctis@fbk.eu,
[2] Gran Sasso Science Institute, L'Aquila, Italy,
catia.trubiani@gssi.infn.it,
[3] University of L'Aquila, L'Aquila, Italy,
{vittorio.cortellessa,antinisca.dimarco,mirko.flamminj}@univaq.it

## 1 Introduction

The problem of interpreting the results of performance analysis is quite critical in the software performance domain: mean values, variances, and probability distributions are hard to interpret for providing feedback to software architects. Support to the interpretation of such results that helps to fill the gap between numbers and architectural alternatives is still lacking.

PANDA (Performance Antipatterns aNd feeDback on software Architectures [1]) is a framework for addressing the results interpretation and the feedback generation problems by means of performance antipatterns, that are recurring solutions to common mistakes (i.e. bad practices) in the software development. Such antipatterns can play a key role in the software performance domain, since they can be used in the search of performance problems as well as in the formulation of their solutions in terms of architectural alternatives.

This tool aims at enabling the usage of software performance antipatterns in the Æmilia Architecture Description Language (ADL).

## 2 Installation on Windows Platform

**Prerequisites:**

- Install a JRE, version 1.6 or newer;
- Download the exact OS version for Eclipse at `https://eclipse.org/downloads/packages/eclipse-modeling-tools/indigosr2`
- Download the TwoTowers tool from: `http://www.sti.uniurb.it/bernardo/twotowers/`;
- Download Panda plugins (panda.zip) from: `https://github.com/CatiaTrubiani/panda-aemilia`.

### Installation Guide

1. Unpack the downloaded eclipse. You will get the directory */eclipse*;
2. Download the pandaExternalPluginsFeatures.zip file at `https://github.com/CatiaTrubiani/panda-aemilia`;
3. Unpack pandaExternalPluginsFeatures.zip. You will get the directory */pandaExternalPluginsFeatures*;
4. Copy the contents of */pandaExternalPluginsFeatures/features* into the folder */eclipse/features*;
5. Copy the contents of */pandaExternalPluginsFeatures/plugins* into the folder */eclipse/plugins*;
6. Unpack TwoTowers.zip. You will get the directory */TwoTowers*;
7. Unpack panda.zip. You will get the directory */panda*;
8. Copy the plugins in the folder */panda/plugins* into the */eclipse/plugins* folder;
9. Run eclipse;
10. Specify the whole path of the TTKernel.exe of TwoTowers (located in the directory */TwoTowers/bin*) through the `Window - Preferences - TwoEagles Preferences` menu voice (see Figure 1);

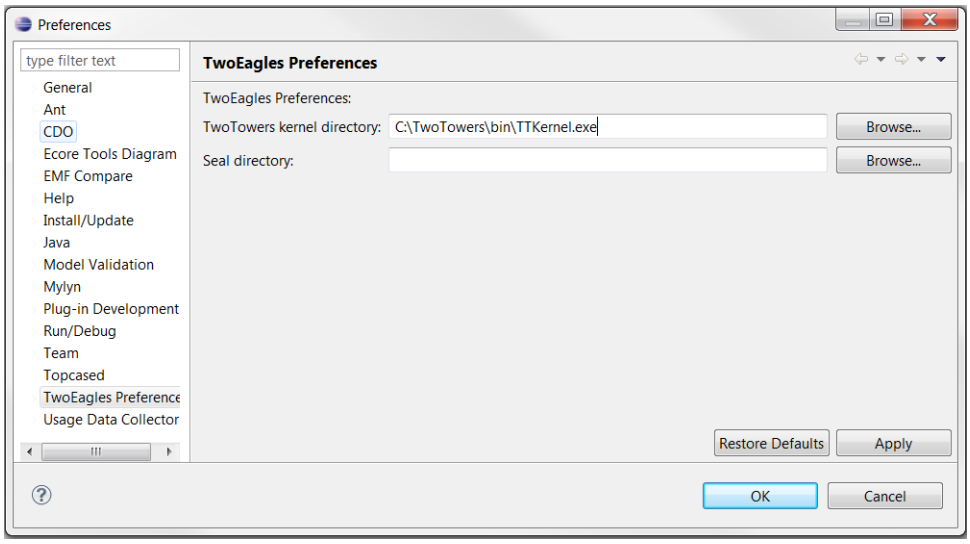



**Fig. 1.** TwoEagles Preferences Tab.

11. Open the TwoEagles Perspective by selecting the menu item `Windows - Open Pespective - Other... - TwoEagles Perspective`;
12. Select `File - New - Project...` menu item;
13. Select `General - Project` e press the Next button (see Figure 2);
14. Assign a name to the project and click the Finish button;

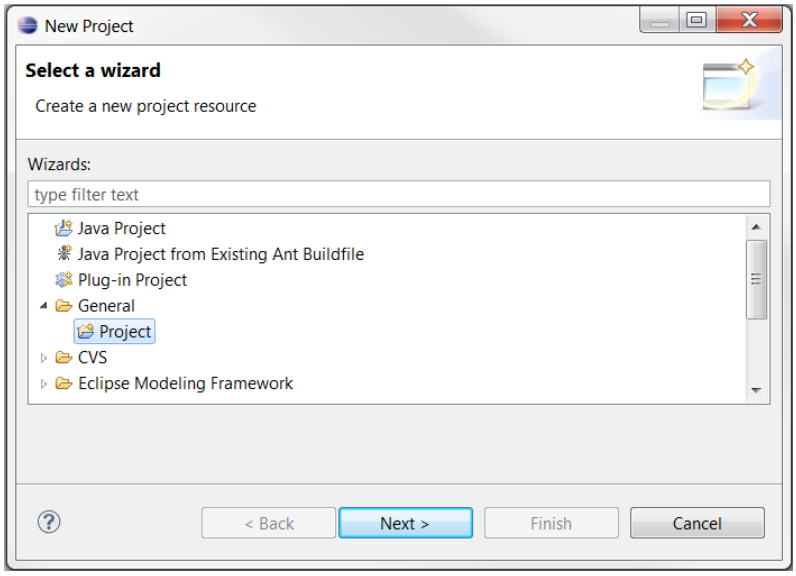



**Fig. 2.** New Project Tab.

15. Create and open in the editor a **.aem* file by right clicking on the new project and by choosing the `New - File` menu item. A case study is provided to test the tool [4]. For this step, you can copy the file *BoA.aem* into your project.
16. Click the Finish button;
17. To generate a Æmilia model, select `TwoEagles - Architectural Assistant - Aemilia model generator` menu item (see Figure 3). This step will generate a **.mmaemilia* file, that can be opened within the editor;
18. Create and open, as for the **.aem* file, a **.rew* file. You can copy and open the file *BoA.rew* into your project;
19. Select `TwoEagles - Performance Evaluator - Stationary reward-based measure calculator - Map Meaures To Indices` menu item;
20. Select a **.mmaemilia* model from the file system and press the Next button (see Figure 4);
21. In the successive windows, you should select at most one index for each measure defined in the **.rew* file (see Figure 5);
22. After all mappings have been defined, you will see a summary window (see Figure 6) and, by clicking the Finish button, a **.rewmapping* file will be generated. It is necessary to refresh the project to visualize the file.
23. Select the **.rew* file and open it in the editor; from here, select `TwoEagles - Performance Evaluator - Stationary reward-based measure calculator - Ga-`

[4] Download and unpack the BoA-caseStudy.zip file from `https://github.com/CatiaTrubiani/panda-aemilia`.

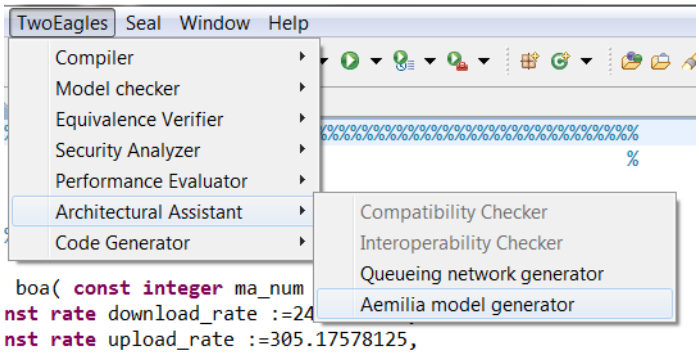



**Fig. 3.** Æmilia model generator.

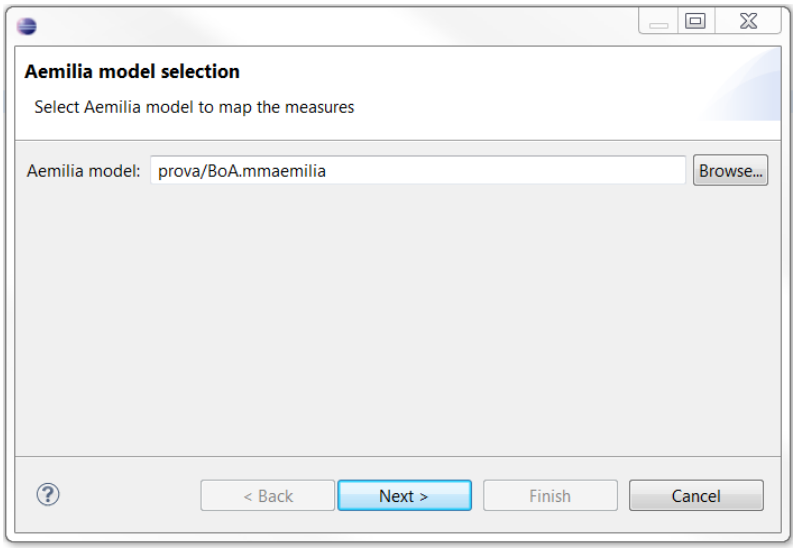



**Fig. 4.** Æmilia model selection.

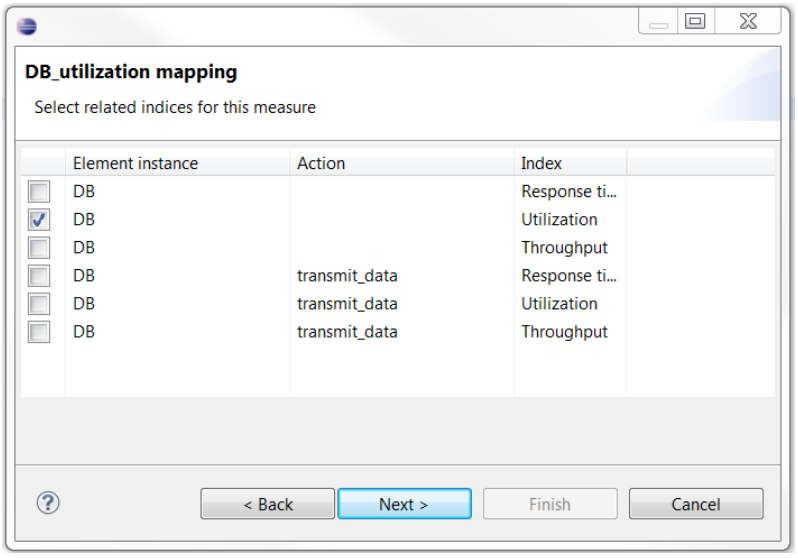



**Fig. 5.** Index selection.

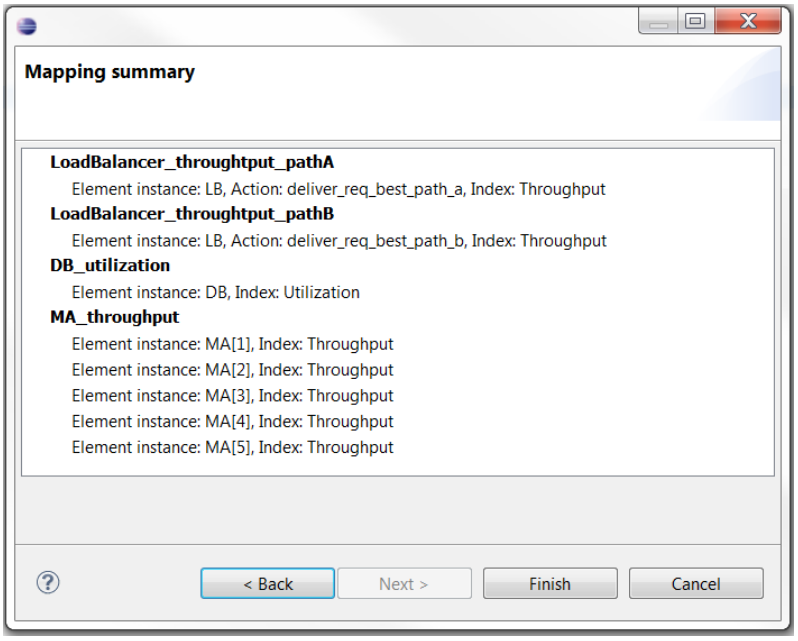



**Fig. 6.** Mapping Summary.

`ussian elimination` menu item and attend that the performance evaluator of TwoTowers generates the **.val* file;

24. Open the **.val* file with the Values Editor and select `TwoEagles - Architectural Assistant - Update Aemilia model` item menu;
25. By selecting the **.rewmapping* file from the file system you will notice that related performance features of Æmilia model (the file with *.mmaemilia* extension) will be setted with the values of the **.val* file. To check this, use the Properties view by selecting `Windows - Show view - Other - General Properties` (see Figure 7);

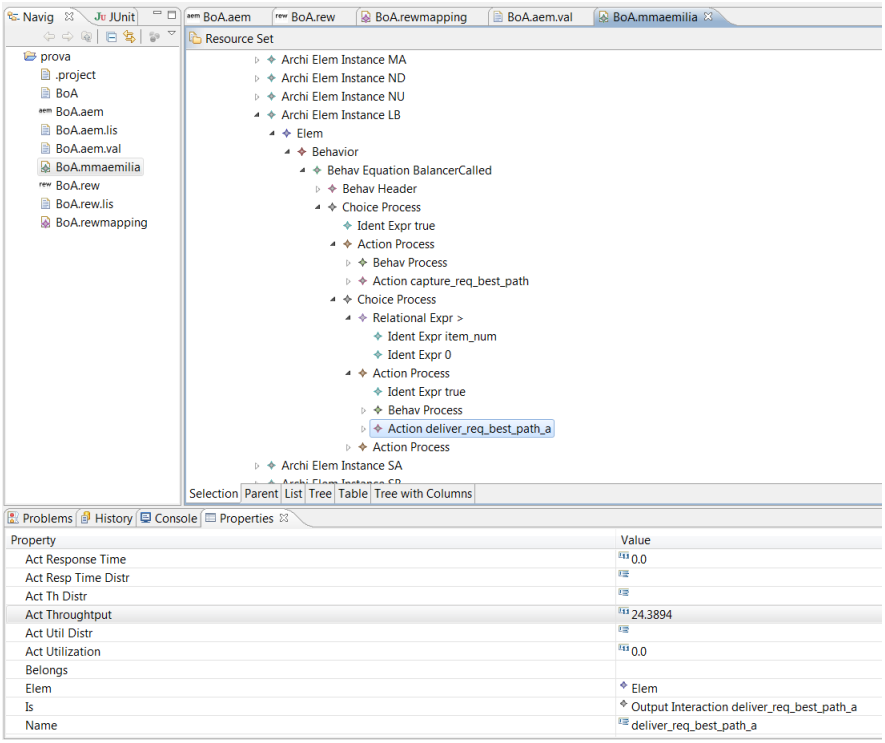



**Fig. 7.** Annotated Æmilia model.

## 3 Æmilia model validation

1. Before executing the validation, you need to define a **.mmaemilia* content type association for the OCL Checker by selecting `Window - Preferences` and then `General - Content Types`. Here, you first select OCL Checker in the top window and then you can add the **.mmaemilia* entry in the file associations window (see Figure 8);

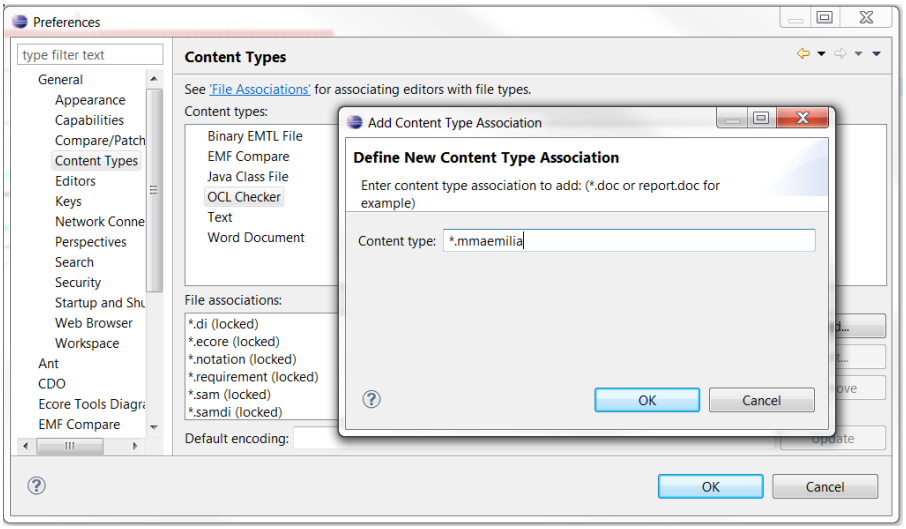



**Fig. 8.** OCL Checker Content Types definition.

2. Right-click on a **.mmaemilia* file and select OCL check menu item (see Figure 9);
3. Select the **.ocl* files to include in the validation (see Figure 10);
4. The validation results are shown in the Global results window (see Figure 11);

**Note:** We remark that the panda.zip file used to install the tool contains, between other files, the following projects:

- the ***detection tool*** project, which contains the **.ocl* files with the rules for both the antipatterns detection and the metamodel checking;
- the ***metamodel*** project, which contains all the metamodel packages making the AEmilia metamodel;
- the ***text2ModelTransf*** project, which contains the text to model transformation tool from Æmilia textual specifications to Æmilia models conforming to the Æmilia metamodel.

These three projects are available for the download also as *source projects* at `https://github.com/CatiaTrubiani/panda-aemilia`.

## References

1. Catia Trubiani. *Automated generation of architectural feedback from software performance analysis results*. PhD thesis, University of L'Aquila, L'Aquila, Italy, 2011. Online: http://cs.gssi.infn.it/catia.trubiani/phDthesis/PhDThesis-CatiaTrubiani.pdf.

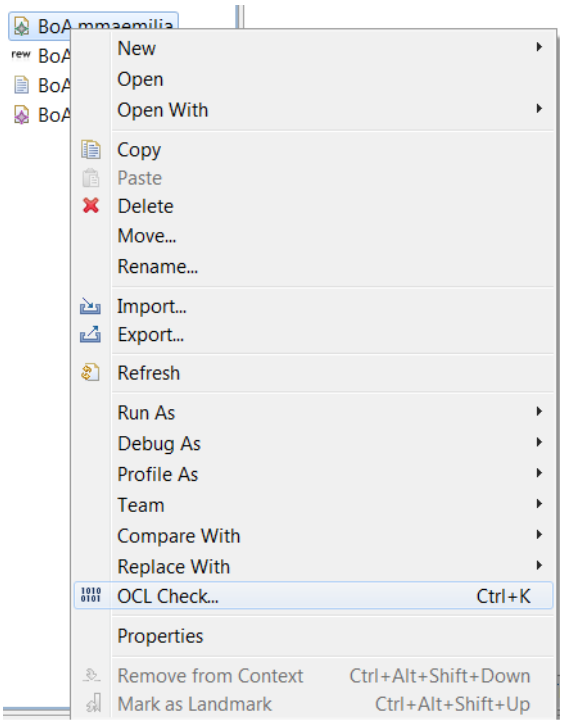



**Fig. 9.** OCL Check.

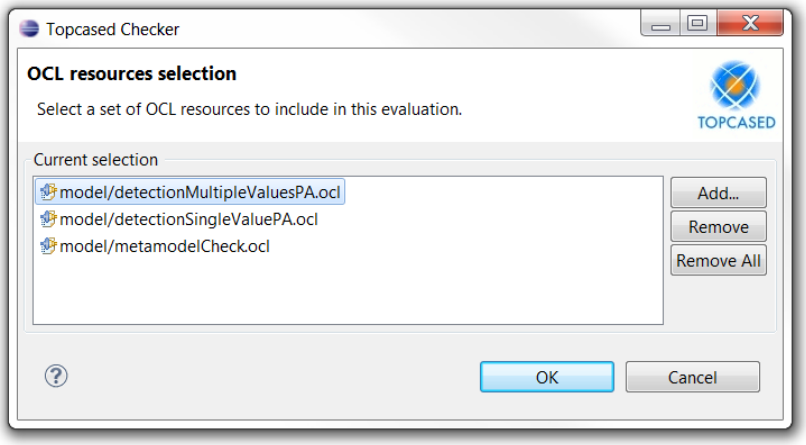
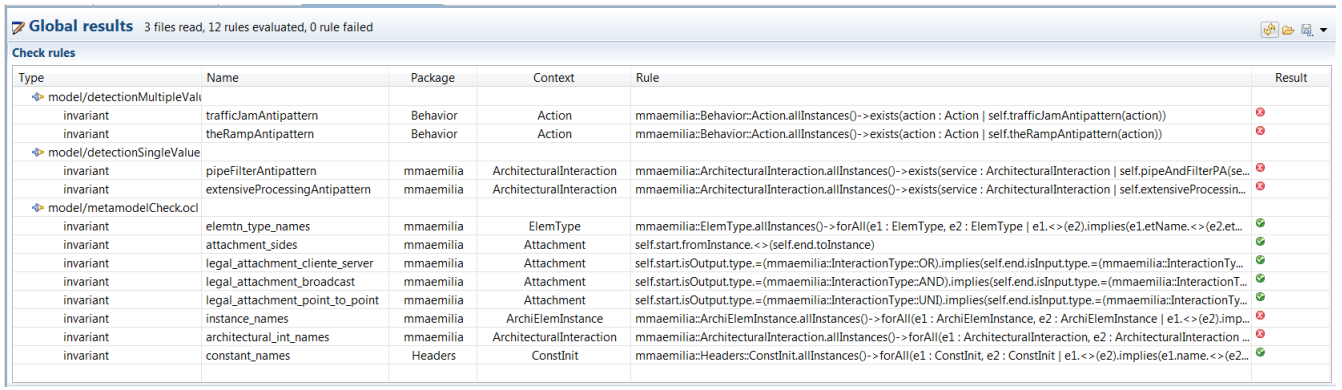



**Fig. 10.** OCL files selection.



**Global results** 3 files read, 12 rules evaluated, 0 rule failed

**Check rules**

| Type | Name | Package | Context | Rule | Result |
|---|---|---|---|---|---|
| model/detectionMultipleVal | | | | | |
| invariant | trafficJamAntipattern | Behavior | Action | mmaemilia::Behavior::Action.allInstances()->exists(action : Action \| self.trafficJamAntipattern(action)) | ✗ |
| invariant | theRampAntipattern | Behavior | Action | mmaemilia::Behavior::Action.allInstances()->exists(action : Action \| self.theRampAntipattern(action)) | ✗ |
| model/detectionSingleValue | | | | | |
| invariant | pipeFilterAntipattern | mmaemilia | ArchitecturalInteraction | mmaemilia::ArchitecturalInteraction.allInstances()->exists(service : ArchitecturalInteraction \| self.pipeAndFilterPA(se... | ✗ |
| invariant | extensiveProcessingAntipattern | mmaemilia | ArchitecturalInteraction | mmaemilia::ArchitecturalInteraction.allInstances()->exists(service : ArchitecturalInteraction \| self.extensiveProcessin... | ✗ |
| model/metamodelCheck.ocl | | | | | |
| invariant | elemtn_type_names | mmaemilia | ElemType | mmaemilia::ElemType.allInstances()->forAll(e1 : ElemType, e2 : ElemType \| e1.<>(e2).implies(e1.etName.<>(e2.et... | ✓ |
| invariant | attachment_sides | mmaemilia | Attachment | self.start.fromInstance.<>(self.end.toInstance) | ✓ |
| invariant | legal_attachment_cliente_server | mmaemilia | Attachment | self.start.isOutput.type.=(mmaemilia::InteractionType::OR).implies(self.end.isInput.type.=(mmaemilia::InteractionTy... | ✓ |
| invariant | legal_attachment_broadcast | mmaemilia | Attachment | self.start.isOutput.type.=(mmaemilia::InteractionType::AND).implies(self.end.isInput.type.=(mmaemilia::InteractionT... | ✓ |
| invariant | legal_attachment_point_to_point | mmaemilia | Attachment | self.start.isOutput.type.=(mmaemilia::InteractionType::UNI).implies(self.end.isInput.type.=(mmaemilia::InteractionTy... | ✓ |
| invariant | instance_names | mmaemilia | ArchiElemInstance | mmaemilia::ArchiElemInstance.allInstances()->forAll(e1 : ArchiElemInstance, e2 : ArchiElemInstance \| e1.<>(e2).imp... | ✗ |
| invariant | architectural_int_names | mmaemilia | ArchitecturalInteraction | mmaemilia::ArchitecturalInteraction.allInstances()->forAll(e1 : ArchitecturalInteraction, e2 : ArchitecturalInteraction ... | ✗ |
| invariant | constant_names | Headers | ConstInit | mmaemilia::Headers::ConstInit.allInstances()->forAll(e1 : ConstInit, e2 : ConstInit \| e1.<>(e2).implies(e1.name.<>(e2... | ✓ |

**Fig. 11.** Validation Global Result.