Significantly Increasing the Usability of Model Analysis Tools through Visual Feedback

El Arbi Aboussoror, Ileana Ober, and Iulian Ober

IRIT, Université de Toulouse, 118 Route de Narbonne
F -31062 Toulouse, France
{El-Arbi.Aboussoror,Ileana.Ober,Iulian.Ober}@irit.fr

Abstract. A plethora of theoretical results are available which make possible the use of dynamic analysis and model-checking for software and system models expressed in high-level modeling languages like UML, SDL or AADL. Their usage is hindered by the complexity of information processing demanded from the modeler in order to apply them and to effectively exploit their results. Our thesis is that by improving the visual presentation of the analysis results, their exploitation can be highly improved. To support this thesis, we define a trace analysis approach based on the extraction of high-level semantics events from the low-level output of a simulation or model-checking tool. This extraction offers the basis for new types of scenario visualizations, improving scenario understanding and exploration. This approach was implemented in our UML/SysML analyzer and was validated in a controlled experiment that shows a significant increase in the usability of our tool, both in terms of task performance speed and in terms of user satisfaction.

1 Introduction

A plethora of theoretical results are available which make possible the use of dynamic analysis and model-checking for software and system models expressed in high-level modeling languages UML [1], SDL [2] or AADL [3]. These results have the virtue of allowing reasoning at the level of models instead of code, making possible early verification and validation, while taking advantage of the abstract nature of modeling languages. Unfortunately, the use of these advanced techniques is not as widespread as their capabilities could justify. One reason for this is that such techniques are often inaccessible to modelers with a basic software engineer training, as it demands advanced knowledge of these techniques and a high investment in learning the specificities of tools in order to apply them and to effectively exploit their results. Our thesis is that by improving the visual presentation of the analysis results, their exploitation by regular users can be highly improved. To support this thesis, we defined a trace analysis approach and we integrated it in Metaviz, a model-driven framework for simulation trace visualization introduced earlier in [4].

The typical functioning of a model-based formal analysis tool, consists in mapping the semantics of the high-level modeling language into a simpler language, more well-suited for formal analysis. Most of the time this language has

F. Khendek et al. (Eds.): SDL 2013, LNCS 7916, pp. 107–123, 2013.

a sound formal semantics that allows reasoning on the model and on its properties, using analysis tools. It is the case of vUML [5] that uses Promela [6], of IFx-OMEGA [7], that uses IF [8], etc. This change of context is in general not supported by a bi-directional mapping to and from the formal language, since the goal is to represent a complex modeling language with a limited set of primitives available in the formal language. Thus, the structure of a model may be very different between the two levels, and analysis results such as execution traces obtained from the lower-level model may be hard to interpret by a user whose knowledge is limited to the upper-level language and model. The trace analysis method is based on the extraction of high-level semantics events from basic events of the underlying semantics. It offers the basis for new types of scenario visualizations, improving the model execution understanding and exploration.

In order to validate these assertions we set up a controlled experiment that shows a significant increase in our model checking tool usability, both in terms of task performance speed and in terms of user satisfaction.

The rest of this paper is organized as follows: we start by an overview of existing methods for executable UML/SysML analysis, and by discussing why exploiting analysis results is challenging. Then we present the typical analysis tool integration pattern and finally describe our approach for extending the verification platform with trace analysis support. The evaluation of the new tool is detailed and discussed in Section 3.

1.1 Translational Semantics Approaches to Model Analysis

To be useful (not only descriptive) models need to have a well defined semantics. Among other advantages, formal definition of semantics provides the possibility to do analysis earlier in the design process. For example non-functional properties such as performance, schedulability or safety can be analyzed. For this purpose, mature formalisms and associated analysis techniques and tools already exist (Petri nets, queuing networks, Markov chains, etc.).

For Model Driven Engineering, the approach usually followed is to:

1. annotate the original models (e.g. UML [9]),
2. generate input artifacts for a formal analysis tool, and finally
3. perform analysis activities on the generated artifacts.

Target analysis formalisms can be process algebra, timed automata, queuing networks etc. Some examples of tools working in this way are IFx-OMEGA[10], vUML [5] or OPTIMUM [11].

Even if these model analysis techniques, called translational semantics approaches, are widely adopted in the industry they still suffer from limitations. Translational semantics approaches to model analysis bring a new complexity to the end user. The analysis results (e.g. model checking counterexamples) are not easily understood by the domain user. This is due to the gap between the two semantic levels: the original one and the target analysis formalism. Those results necessitate expertise in the low level analysis semantics. To enable a usable approach to model analysis a translation of the low level results to a more user friendly abstraction should take place in the process.

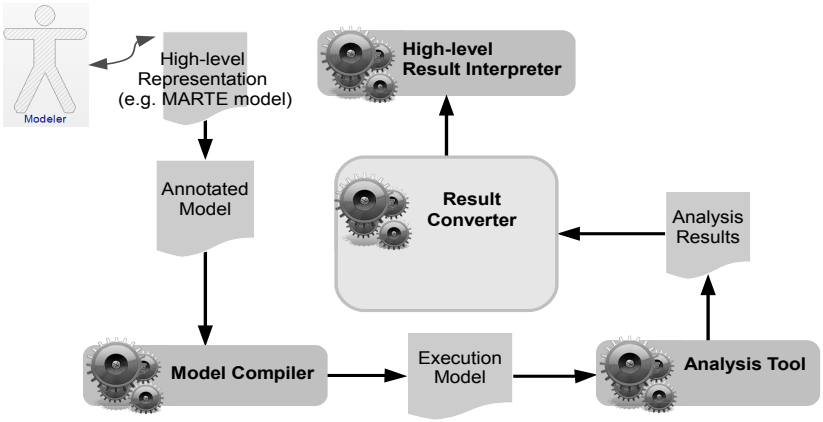

Fig. 1. Generic approach for analysis results exploitation. The annotated model is translated to a formal model for analysis. Results are then showed to the end user in a visual interpreter.

1.2 General Schema for Analysis Tool Integration

Analyzing models is assessing some properties of them. For example UML offers an extension mechanism, namely profiles that enable building a language to express non-functional properties. This approach offers a clear advantages like the reuse of the tools and techniques available for UML. Two well known examples are SPT and MARTE profiles. UML Profiles are integrated with analysis tools using the basic architecture described in Fig. 1. In this architecture model transformations are used to fill the gap between UML modeling technical space and the analysis technical space. This semantic gap is at the heart of the diagnostic problem described in the previous section1.1.

2 Extending a Verification Platform with Trace Analysis Support

IFx-OMEGA[1] platform integrates a compiler, a simulator and model-checker for a rich subset of UML and SysML [7]. The toolset relies on the automatic translation of models into a lower-level language (named IF) based on asynchronous communicating extended timed automata, and on the use of the extensive toolset available for this language [8]. The validation acts on a UML or SysML model, which is first translated to an IF model, and then compiled[2] to an executable program that will be used for automatic verification and interactive simulation.

[1] `http://www.irit.fr/ifx`

[2] In some cases, the model can be first simplified using automatic abstraction techniques.

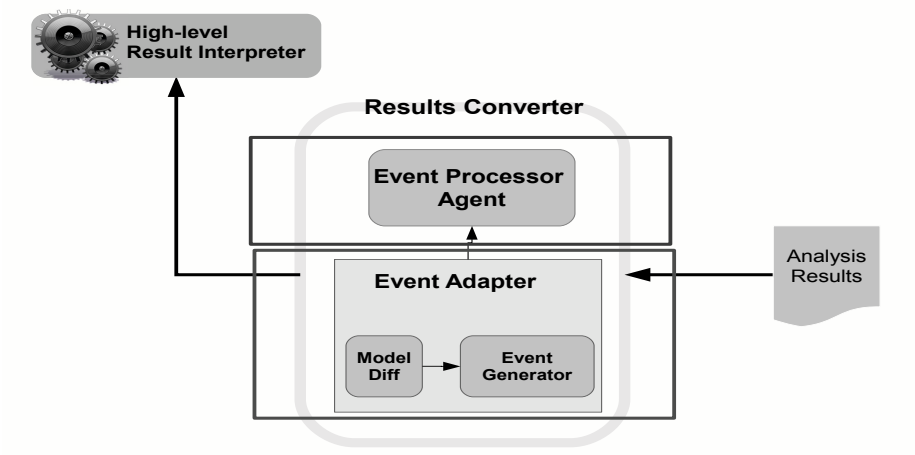

Fig. 2. Generic architecture for extracting high-level events from analysis results using a *Result Converter*. It is composed of an *Event Adapter* for generating relevant events from an operational semantics and an *Event Processor Agent* that translates those events to user level events.

The IFx validation approach has been applied to several industry-grade models such as Ariane-5 [12], MARS [13] and SGS [10], and has proven to be very effective in discovering design issues. As an application to this approach we propose to extend the IFx-OMEGA platform. The extension transforms low-level semantics analysis result. Figure 2 shows the architecture of this extension. The automatic verification and the simulation activities generates execution scenarios as analysis results. Those scenarios are expressed using the low-level semantics used by the IF model checker. The extension we propose translates the relevant information from the scenarios into a high-level like formalism syntax. To extend the IFx-OMEGA platform with result analysis facility we follow the generic architecture proposed in [14] and depicted in Fig. 1.

2.1 Extracting Execution Events Using Model Differences

The generic architecture has to be refined to enable a flexible and extensible implementation. We choose to take an event based approach to report the analysis results. The design rationale behind this choice is the decoupling introduced by this approach and its adequacy to our context. Indeed an event reports changes in monitored states [15] and since we are in the context of translating UML models to an operational semantics we have this notion of state changes within the execution model. Those state changes are captured using a model difference mechanism [16] and the difference model is then transformed into a set of events

by an *event generator*. Those events report what has occurred in the analysis results (e.g. state changes, message passing, ...). An example is shown in Fig. 3. This figure shows an original trace on the left (an XML file loaded in an tree-based view) alongside the extracted event. The event showed on the right correspond to the computed difference between two configurations of the system in the trace. In the example the difference between the system configurations (from step 15 to 16) is a state change of the *Model_EVC instance* from *Waiting* status to *Started*.

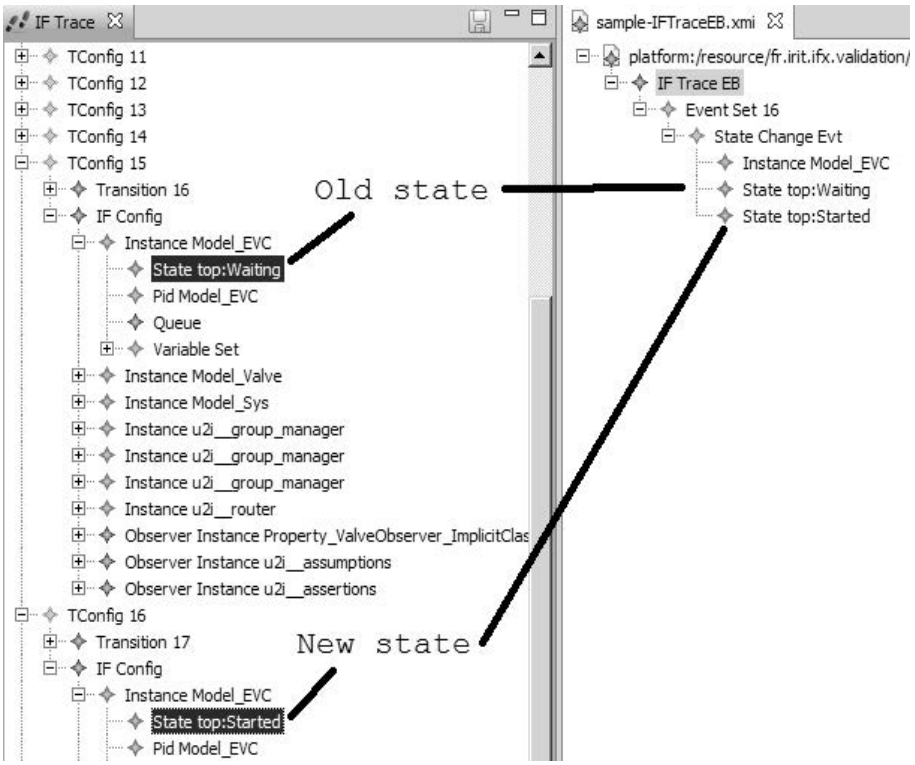

Fig. 3. Extracting execution events using model differences. State changes are captured using a model difference mechanism, the difference model is transformed into a set of events by an *event generator*.

2.2 Abstracting Execution Events to High-Level Semantics Events

Once the relevant execution events are extracted we have a clear view on what has changed in the execution semantics. But still, we needed a translation step to make the analysis results expressed in the high-level semantics. For this step we use an *Event Processor Agent* [15] implemented as a model transformation from low-level analysis results to a high-level set of events. Those events can be easily understood by the user and embed relevant concepts from the high-level design

model. High-level events can constitute a base for reporting the analysis results directly into the user models or for animating a part of the original specification. The two transformation rules in Listings 1.1 and 1.2 show an excerpt of the ATL transformation that abstracts execution events to high level events. The first rule transforms each *IF process state change event* into an *OMEGA state enter event*. The second rule extracts *OMEGA send events* from *IF enqueue events*.

```
StateChangeEvt2StateEnterEvent extends TraceEvent2OmegaEvent {
    from
        sEvt: IFTraceEB!StateChangeEvt
    to
        tEvt: OmegaTraceEB!StateEnterEvent (
            className <- sEvt.instance.type,
            name <- sEvt.instance.pid.name,
            stateName <- sEvt.newState.name,
            oldStateName <- sEvt.oldState.name
        )
}
```

Listing 1.1. ATL rule for extracting *OMEGA object state entering events*

```
rule EnqueueEvt2SendEvent extends TraceEvent2OmegaEvent {
    from
        sEvt: IFTraceEB!EnqueueEvt
    using {
         mes: IFTraceEB!Message = sEvt.messages -> first();
    }
    to
        tEvt: OmegaTraceEB!SendEvent (
            signal <- mes.signalType,
            by <- thisModule.pid2Object(mes."from")
            ...
}
```

Listing 1.2. ATL rule for extracting *OMEGA message sending events*

2.3 Interpreting High-Level Semantics Events

After abstracting execution events we get a set of high-level semantics events. High-level events gather state changes that can be easily understood by the user. The last step in the proposed process is the visual interpretation of the analysis results that is now transformed into high-level events. End user visualizations has to be carefully designed to help the user grasping the event information without visual effort. More on this point can be found in our previous work [4]. Figure 4 shows graphical rendering of *OMEGA state enter event* extracted from *IF process state change event*.

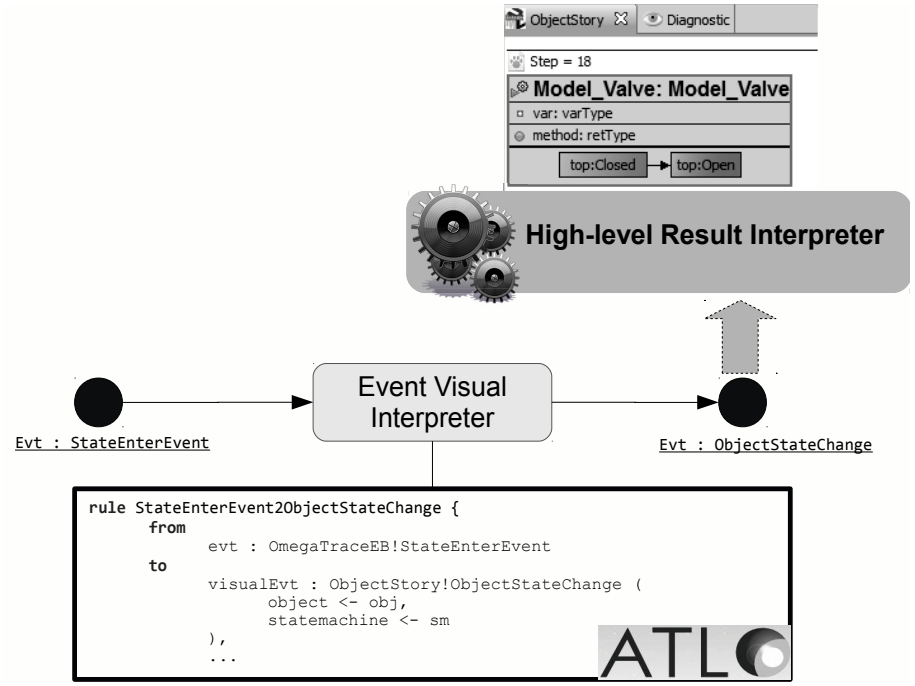

Fig. 4. Graphical rendering of high-level events. The *Event Visual Interpreter* executes a model transformation to generate the input model for the visual renderer.

3 Evaluation

In order to assess the advantages and limitations of the new visualization, we need to evaluate its usage. Several techniques can be used for such an evaluation. Some of these are purely subjective, while others use an objective and quantitative approach [17]. Moreover, evaluation techniques can be categorized according to the stakeholders: techniques such as *cognitive walk-through* or *heuristic evaluation* involve human factors experts, while *observational* or *experimental* techniques rely on user participation.

The premier goal in our study is to make our model analysis tool more accessible to a wider audience. Today, the overwhelming majority of modelers are not familiar with analysis techniques such as model checking. In order to assess whether we have achieved our premier goal, we need to evaluate the approach on a sample of users compliant with this profile. Therefore, due to its objective and quantitative orientation, we decided to use a *controlled experiment* that involves user participation. However, since this technique does not provide a detailed user impression and satisfaction overview, we complement our validation with a *subjective approach*, by means of a questionnaire technique based on the System Usability Scale [18].

3.1 Defining Formal Methods Usability

Before defining the hypothesis and designing an experiment, we need to understand the notion of *usability* in the context of formal analysis techniques. The ISO standard definition [19] for usability, defines it as:

> "the extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use."

To customize the usability definition in our context, we need to set its three characteristics - effectiveness, efficiency and satisfaction - in the context of formal methods:

- *Effectiveness* is the ability of users to understand, find and correct errors reported by the analysis process.
- *Efficiency* is the time and cognitive effort needed to perform each of the three above-mentioned tasks.
- *Satisfaction* is the subjective impression the users get after using the integrated tool suite that supports the three task.

3.2 Hypotheses Formulation

As mentioned above, our goal is to increase analysis tools usability. To achieve this goal, we propose to enhance tools with visualization techniques. Thus we can formulate the following hypothesis:

H. *Effectively supporting the user in understanding model analysis* ***results****, will increase the analysis tool* ***usability****.*

The notion of *analysis tool usability* was defined in the previous section. Based on this, our hypothesis **H** can be further refined:

H1. Participants using the new tool extension will spent less time understanding the trace.
H2. Participants using the new tool extension will have a better understanding of the trace.
H3. Participants using the new tool extension will spent less time locating the error in the trace.
H4. Participants using the new tool extension will locate the error in the trace with more precision.
H5. Participants using the new tool extension will spent less time understanding the error cause(s).
H6. Participants using the new tool extension will have better understanding of the error cause(s).

These refined hypothesis will allow more insights into the experiment results.

3.3 Experiment Design

In this section we will go further with the experiment design, by defining several of its characteristics. The terminology used in this section is the one defined in [20].

Participants. Participants were chosen from Master and PhD students from the University of Toulouse. The experiment was conducted with 10 subjects distributed in two groups, the experimental group (group A in next sections) and the control group (group B). All the participants were already familiar with UML, two of them have already used a formal analysis tool, but none of them has used ours. In order to assess their adequacy with the user population, participants were asked to fill a questionnaire. The questionnaire has 7 parts and 12 questions such as education level, experience with modeling, and verification techniques abilities. In the first group technical experience median is 4, mean is 4,4 and standard deviation is 1.67. In the second group mean is 4.6 with a median 4, the standard deviation is 1.52.

Experimental Unit. Participants were asked to visualize the execution trace of a small OMEGA UML model representing an Electronic Valve Controller. A timed constraint was set on the model. The OMEGA model is then used as input for the IF Model Checker. Since the model was intentionally violating the constraint, the model checker generated a counterexample. The model was built to get a representative counterexample of what a modeler would get from using the IFx-OMEGA toolbox. Participants will explore this counterexample and perform some well defined tasks.

Experimental Variable. It corresponds to the software product used by the participant to explore the analysis results. We consider the Metaviz [4] tool being the experimental unit. It offers support for IF traces exploration alongside with elaborate visualizations. Figure 5 presents a screen shot of the trace analysis support. The right panel contains the error trace browser, which is similar to what is available in other analysis tools. The middle panel contains the trace visualization feature that we added.

Factors. Also called independent variables, are those we are going to manipulate in the participants environment to see how other variables (response variables) are affected. Our experiment has the goal to analyze how a better support for the users would affect their understanding of the model analysis results. Thus we run the experiment with a two-level factor: Metaviz with elaborate trace visualization support activated and deactivated. Consequently we have the following levels:

level 1: only basic Metaviz views are activated
level 2: advanced visualization is activated

Response Variables. They correspond to the experimental aspects impacted. To investigate their quantitative values we have designed a set of user tasks distributed in three categories. Each category corresponding to a response variable. The categories are related to the following cognitive tasks:

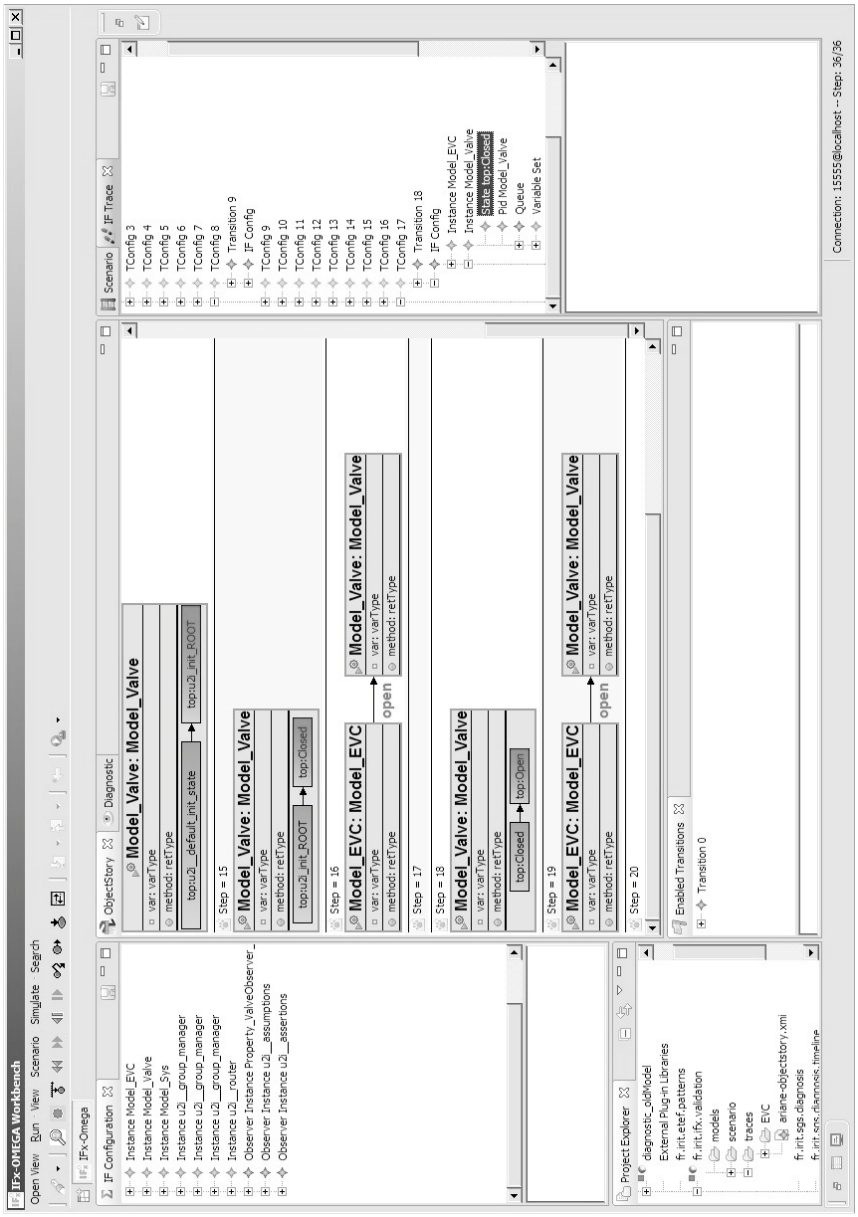

Fig. 5. Metaviz with the full support for trace analysis

– understanding the trace;
– locating the error;
– understanding the error's cause and fixing the model.

For each category we assess two characteristics: the speed and the quality.

Tasks. The set of tasks each participant has to perform was designed to cover the understanding of the trace, the error and its cause. Each participant was given a set of 7 tasks to perform using the trace analysis tool. Additionally, the time spent by each participant performing a task is monitored with an external stopwatch application. We do not take into account the time spent by the participant writing down the answers. The table 1 gives an overview of the tasks.

Table 1. Task per each cognitive category. Participants were asked to perform a set of tasks that spans three cognitive task types.

Cognitive Types	Tasks
Trace Understanding	1. What are the instances created during this execution and at what step
	2. What are the exchanged messages, at what step and between which instances
	3. At which step of the trace, the instance Model_Valve enters the state Closed
Error Locating	4. Find in which step the property is violated
Cause Understanding	5.Which diagram should be modified to satisfy the property ?
	6. Explain informally (in English) what modification you want to do
	7. Fix the error in the model (syntax is free).

3.4 Results

The goal of our experiment is to see whether extending an analysis tool with the event-based visualization mechanism described previously would improve analysis results exploitation. Figures 6 and 7 show the results for each task, in terms of time and success rate. Participants that use the Metaviz extension belong to the group A (experimental group) , while group B (control group) corresponds to participants using the basic version of the tool (i.e. without the Metaviz extension).

Time Spent in Each Task. Results for tasks *T1* and *T2* show that participants in the *group A* perform 5 times faster than the *group B* participants. For task *T3* they perform 8 times faster, while for the last set of tasks *T5* to *T7* they perform 2 times faster. The overall unbalanced improvement rate is

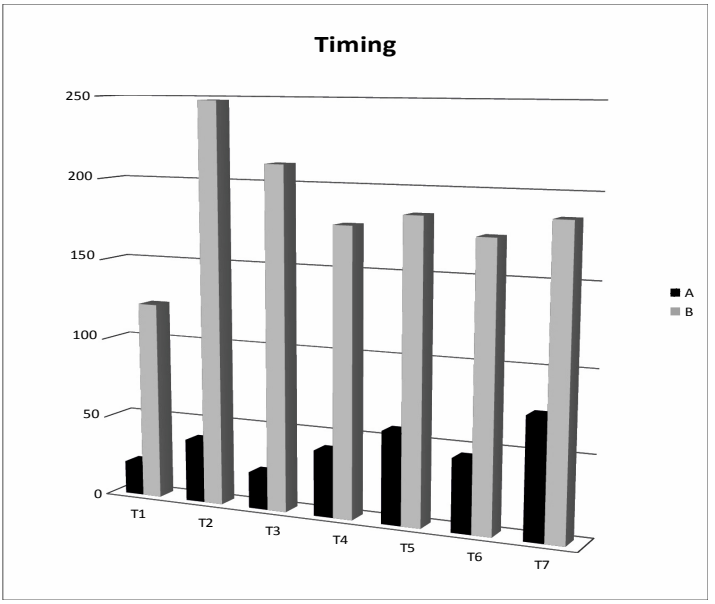

Fig. 6. Time (in secondes) to perform each task. Participants that use an elaborate visualization perform tasks 4 times faster.
A: experimental group, uses an elaborate visualization
B: control group, uses a basic visualization

therefore of 400%. This important increase in task performance speed is due to the cognitive nature of each task category. For instance, one can notice that the experimental group *group A* was 8 times faster in performing the task *T3*. This task is the most demanding in cognitive workload for the participants. Indeed, it asks participants to find the step in the trace where a certain instance enters a particular state. The participant has to recall the instance and state names while he goes step by step through the whole trace. This is where we can see the power of having a visualization that presents only what has changed in the trace. The Metaviz visualizations focus on the state change of the relevant instance and filter other irrelevant information. Figure 5 shows the basic visualization alongside the elaborate one. The visual vocabulary used is highly intuitive and thus enabled the control group to perform the task *T3* 8 times faster.

Quality. Concerning success rates, results revealed an increase of the participant's answers quality. Results for understanding the trace (task *T1* to *T3*) show that understanding of the instances interactions (i.e. message passing) is 25% better. For locating the error (task *T4*) the improvement is about 9%. For the last cognitive category, namely *understanding the cause* the improvement is of 40%. Figure 7 shows the success rate for each task.

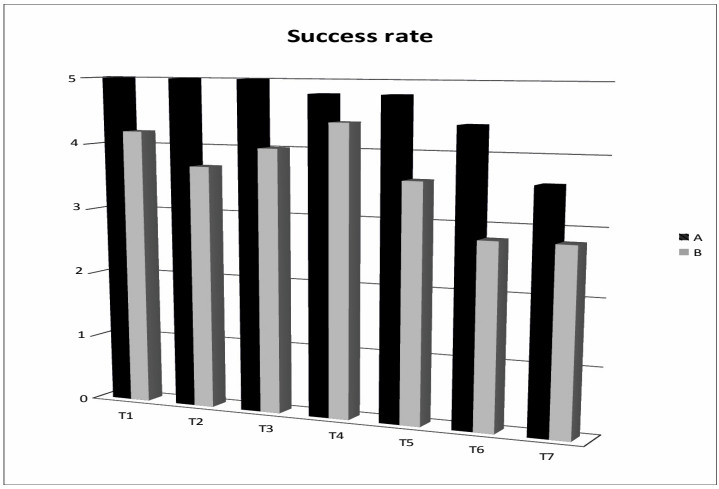

Fig. 7. Success rate for each task.Participants that use an elaborate visualization produce task outputs of a better quality.
A: experimental group, uses an elaborate visualization
B: control group, uses a basic visualization

To complement our analysis we have run a questionnaire based evaluation. This evaluation assessed the user satisfaction. For this purpose we relied on the System Usability Scale test[18]. We have chosen this analysis method since it is lightweight and reliable [21]. Results for user satisfaction are slightly higher for

Table 2. Statistical results by cognitive category. Results show means and Student's t hypothesis tests for each cognitive category of tasks.

	Trace Understanding (T1 to T3)		**Error Understanding (T4)**		**Cause Understanding (T5 to T7)**	
	Success	**Timing**	**Success**	**Timing**	**Success**	**Timing**
Group A	5	44.33	5	18	5	35
	5	27.33	5	21	4.33	55.67
	5	27	5	59	5	30.67
	5	8.33	5	51	3.33	62.67
	5	29.67	4	55	3.67	110
Group B	5	103	5	20	3	100
	3.17	325	5	540	0	600
	5	183.33	5	27	5	90.33
	5	120.67	5	32	5	23.33
	1.67	238	2	267	2.33	94.33
t	**0.16**	**0.004**	**0.54**	**0.22**	**0.26**	**0.28**

the experimental group (*group A*) with 68% versus 61% for the control group (*group B*). The results presented in table 2 show an increase of about 11% of the user satisfaction.

3.5 Hypothesis Tests

The null hypothesis corresponding to each hypothesis formulated previously can be summarized by *there is no difference between participants that use the elaborate visualization and those who use the basic one in performing the tasks from the category CTx*. Were *CTx* is one of the three cognitive task categories listed in 1. As mentioned previously task performance is measured with the speed and quality of three cognitive task types (i.e. understanding a trace, locating an error, understanding its causes). As we describe in section 3.3 the standard deviation can be considered equal and we can test the null hypothesis using the Student's t test. We applied a two-tailed Student's t test to the six null hypothesis. The results are showed in table 2. The experiment was run with five participants in each group, that gives us 8 degrees of freedom and a value for $t_{0.99}$ of 3,355. All the values of t are under the value of $t_{0.99}$, we can then reject the null hypothesis. For the user satisfaction results showed in table 3 are also statistically sound (t value of 0,7) and shows that the null hypothesis corresponding to user satisfaction can also be rejected. Initial hypotheses are thus accepted, meaning that *effectively supporting the user in understanding analysis results increases analysis tool usability.*

Table 3. Usability tests using the System Usability Scale[18]. Results show an increase of 11% in usability for the experimental group (group A). Results are statistically sound according to the Student's t test.

	System Usability Scale
Group A	67.7
Group B	61.0
t	**0.71**

3.6 Threats to Validity

Threats related to the conclusion. We have used statistical test, namely Student's t test. Our samples has the same number of individuals and similar variances which ensure the soundness of the test conclusions. *Construct threats.* The role of the trace abstractions is to reduce the amount of data to a relevant set of information for a specific user task, this avoid the cognitive overload. The visualization has a different role. It brings to the user these relevant information in a domain oriented way, that is, in terms of concepts already known by the user. No additional effort is needed from the user to understand the notations and semantics of the visualization constructs (perception overload). The visualization alone is not enough, this is why the control group (that uses a basic visualization of the low-level traces) has the worst results. The huge amount of

data gathered to the user in a basic view (without abstraction) is cognitively demanding and time consuming. *Internal threats*: The statistics shown in the section 3.3 ensures that there's is no significant difference in the participants technical level. *External threats*: While used with a small model, the technique is more likely to accelerate user exploring and understanding for bigger models. We think that for experts, the results may not be as significant as for users with average technical background. But we should be careful of this generalization, the *expertise reversal effect* as coined by Sweller [22] may arise. Thus, further experiments should be conducted to assess the results for expert users.

4 Related Work

Early work in analyzing UML models like [8,9] set among others the foundations of model-driven analysis. Recent work on the MARTE profile [11] introduces the use of user feedback in an integrated MDE approach. Results of MARTE [23] models analysis is used to annotate back the original user models. The RT-Simex research project [24] tackles the problem of user feedback with an elaborate user interface. This is very similar to our approach but the work focuses on clock reconciliation between independent traces and no controlled experiment is conducted to assess the effectiveness of the user feedback. In the work of [25] an emphasis is put on the lack of user friendly interfaces in model checkers. They effectively address the problem of understanding analysis results but directly at the low level semantics. A broader view to the issue of designing SE notations is addressed by Moody [26]. He emphasizes the lack of foundational work on the syntax of software engineering visual notation. Combemale et al. adressed the problem of extracting high-level traces from low-level executions in [27]. They show how to extract high level trace from low level traces. They also assume an existing execution model for the high-level formalism and editors. Our work emphasize the importance and the effectiveness of the user feedback but go further by validating the added value in a controlled experiment.

5 Conclusion and Future Work

Today's integrated development environments offer many debugging facilities, that allow the developer to follow closely the code execution, to debug it and to understand it. While modeling languages aim at raising the level of abstraction in software development and could support much more powerful analysis techniques, the tools existing to support them are still difficult to use [28]. Using languages such as UML, SysML, SDL, to model the software, performing model-based analysis on these models and understanding the analysis results is still a challenging task. Improving the existing tools and mechanisms for exploring and understanding model based analysis results is greatly needed for a larger adoption of formal methods. This paper presents our contribution in this direction. Our approach provides a semi-automatic technique to implement a feedback mechanism in a SysML/UML translational semantics approach. Based

on an existing SysML/UML tool that offers the possibility to perform verification, we develop an advanced and flexible trace analysis support mechanism. As described in this paper, this mechanism allows the analyst to reason at model level on the model execution, in terms of easily understandable high-level events. The pertinence of our approach is assessed through an evaluation, based on well established evaluation mechanisms (Usability Scale System, ISO notion of usability, etc). In order to perform such an evaluation, we needed to adapt the notion of usability to the context of formal methods usability, and to adapt the evaluation process to our setting. The goal of this experiment was to see whether extending analysis tools with a well designed event-based visualization would improve analysis results exploitation and the results are meeting our expectations. Several directions could be taken for future work. Beyond improving the existing approach/toolset, we intend to add new visualization techniques, based on specific tasks to be performed during the analysis (variable change impact, queue size evolution, etc), to identify new kinds of user profile based visualizations that may assist the user, depending on its level of expertise, to perform new experiments with different user profiles, etc. We strongly believe that by facilitating the access to analysis tools by regular users, the interest in using modeling techniques could significantly increase and the much advertised advantages of these techniques could finally get accessible to a larger panel of users.

Acknowledgments. We thank Anke Brock and Antonio Serpa for their support, the volunteer participants to the experiment, and the reviewers for their valuable suggestions.

References

1. Object Management Group: Unified Modeling Language, `http://www.uml.org/`
2. International Telecommunication Union: ITU-T Recommendation Z.100 (12/11) – Specification and Description Language – Overview of SDL-2010 (2011), `http://www.itu.int/rec/T-REC-Z.100-201112-I/en`
3. SAE International: SAE Architecture Analysis & Design Language (AADL) AS5506 Rev.B (2012), `http://standards.sae.org/as5506b/`
4. Aboussoror, E.A., Ober, I., Ober, I.: Seeing Errors: Model Driven Simulation Trace Visualization. In: France, R.B., Kazmeier, J., Breu, R., Atkinson, C. (eds.) MODELS 2012. LNCS, vol. 7590, pp. 480–496. Springer, Heidelberg (2012)
5. Lilius, J., Paltor, I.P.: vUML: A tool for verifying UML models. In: 14th IEEE International Conference on Automated Software Engineering (ASE 1999), pp. 255–258. IEEE Computer Society (1999)
6. Holzmann, G.J.: Design and Validation of Computer Protocols. Prentice Hall (1991)
7. Ober, I., Dragomir, I.: OMEGA2 – A new version of the profile and the tools. In: 15th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2010), pp. 373–378. IEEE Computer Society (2010)
8. Bozga, M., Graf, S., Ober, I., Ober, I., Sifakis, J.: The IF toolset. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 237–267. Springer, Heidelberg (2004)

9. Espinoza, H., Dubois, H., Gérard, S., Medina, J.L., Petriu, D.C., Woodside, C.M.: Annotating UML Models with Non-Functional Properties for Quantitative Analysis. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 79–90. Springer, Heidelberg (2006)
10. Conquet, E., et al.: Formal Model Driven Engineering for Space Onboard Software. In: Embedded Real Time Software and Systems (ERTS2) (2012), http://www.erts2012.org/Site/0P2RUC89/7B-4.pdf
11. Mraidha, C., Tucci-Piergiovanni, S., Gerard, S.: Optimum – a MARTE-based Methodology for Schedulability Analysis at Early Design Stages. SIGSOFT Software Engineering Notes 36(1), 1–8 (2011)
12. Ober, I., Graf, S., Lesens, D.: Modeling and Validation of a Software Architecture for the Ariane-5 Launcher. In: Gorrieri, R., Wehrheim, H. (eds.) FMOODS 2006. LNCS, vol. 4037, pp. 48–62. Springer, Heidelberg (2006)
13. Ober, I., Graf, S., Yushtein, Y., Ober, I.: Timing Analysis and Validation with UML – the case of the embedded Mars bus manager. Innovations in Systems and Software Engineering 4(3), 301–308 (2008)
14. Official reference MARTE Tutorial, http://www.omg.org/omgmarte/Tutorial.html
15. Etzion, O., Niblett, P., Luckham, D.: Event Processing in Action. Manning (2010)
16. Lin, Y., Gray, J., Jouault, F.: DSMDiff – a differentiation tool for domain-specific models. European Journal of Information Systems 16(4), 349–361 (2007), http://dx.doi.org/10.1057/palgrave.ejis.3000685
17. Dix, A.: Human-Computer Interaction. Pearson/Prentice-Hall (2004)
18. Brooke, J.: SUS – A quick and dirty usability scale (1996), http://hell.meiert.org/core/pdf/sus.pdf
19. International Standards Organisation: ISO 9241-11:1998 Ergonomic requirements for office work with visual display terminals (vdts) part 11 – Guidance on usability, http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=16883
20. Juzgado, N.-J., Moreno, A.-M.: Basics of Software Engineering Experimentation. Springer (2001)
21. Stanton, N., et al.: Human Factors Methods – A Practical Guide for Engineering and Design. Ashgate (2005)
22. Sweller, J.: Evolution of human cognitive architecture. Psychology of Learning and Motivation 43, 215–266 (2003)
23. The UML Profile for MARTE, http://www.omgmarte.org/
24. DeAntoni, J., et al.: RT-SIMEX – Retro-analysis of execution traces. In: Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2010), pp.377–378. ACM (2010)
25. Yokoyama, S., Sato, H., Kurihara, M.: User-friendly GUI in Software Model Checking. In: Proceedings of the 2009 IEEE International Conference on Systems, Man and Cybernetics (SMC 2009), pp. 468–473. IEEE Press (2009)
26. Moody, D.L.: The "Physics" of Notations: Toward a scientific basis for constructing visual notations in software engineering. IEEE Transactions on Software Engineering 35(6), 756–779 (2009)
27. Combemale, B., Gonnord, L., Rusu, V.: A Generic Tool for Tracing Executions Back to a DSML's Operational Semantics. In: France, R.B., Kuester, J.M., Bordbar, B., Paige, R.F. (eds.) ECMFA 2011. LNCS, vol. 6698, pp. 35–51. Springer, Heidelberg (2011)
28. Hutchinson, J., et al.: Empirical Assessment of MDE in Industry. In: Proceedings of the 33rd International Conference on Software Engineering (ICSE 2011), pp. 471–480. ACM (2011)