# Seeing Errors: Model Driven Simulation Trace Visualization

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.** Powerful theoretical frameworks exist for model validation and verification, yet their use in concrete projects is limited. This is partially due to the fact that the results of model verification and simulation are difficult to exploit. This paper reports on a model driven approach that supports the user during the error diagnosis phases, by allowing customizable simulation trace visualization. Our thesis is that we can use models to significantly improve the information visualization during the diagnosis phase. This thesis is supported by Metaviz - a model-driven framework for simulation trace visualization. Metaviz uses the IFx-OMEGA model validation platform and a state-of-the-art information visualization reference model together with a well-defined development process guiding the user into building custom visualizations,essentially by defining model transformations. This approach has the potential to improve the practical usage of modeling techniques and to increase the usability and attractiveness of model validation tools.

**Keywords:** Software visualization, trace exploration, embedded systems, model based validation, model dynamic analysis.

## 1 Introduction

Important efforts were deployed by research and industry in order to develop powerful verification and validation techniques for the design models used in the early phases of development of real-time embedded systems (RTES) [7,1]. In spite of the fact that a lot of interesting results were obtained, formal verification and validation are used on a very few concrete projects. This is partially due to the fact that the results of the formal verification are difficult to exploit Our thesis is that we can use models to significantly improve the information visualization during the diagnosis phase. To support this thesis, we have built *Metaviz* - a model-driven framework for simulation trace visualization. Metaviz aims to *support the user during the error diagnosis phases*, by allowing flexible simulation trace visualization. It is built on top of IFx-OMEGA [3,4] a simulation and verification toolbox for UML and SysML RTES models.

The goal of the simulation step in the validation process of a System Under Diagnosis (SUD) model is either the *interactive detection of design errors*, or

R.B. France et al. (Eds.): MODELS 2012, LNCS 7590, pp. 480–496, 2012.

the *understanding of the nature of errors detected by automatic verification.* Therefore, the purpose of performing interactive simulation is *diagnosis*, which is essentially a *cognitive* task: the user has to understand the overall behaviour of the system using a scenario exploring interface, and to discover errors in the scenario.

For the type of complex RTES design models that are targeted by IFx-OMEGA, it turns out that the diagnosis generally involves examining *multiple non-contiguous steps of a scenario*, and multiple *entities in the system (blocks, ports, message queues, etc.)*. While supporting some simple forms of view customization, the traditional simulation interface cannot define visualizations computed from different steps in the simulation scenario, and thus it is hard for a user to infer the cause of an inter-process communication error from the simulation of an error scenario.

It is a commonly accepted fact that the human working memory is limited to a few items [42], while dealing with a simulation trace usually implies watching values and relationships of tens or hundreds of elements at multiple steps in the trace. Therefore, what we need is a way to boost the user perception and cognition so that it can gather the right information for finding an error pattern. This kind of problem is addressed by many works in the field of information visualization [46]. The synergy between research in information visualization and software visualization is a promising research area [23] that we exploit in our approach, by defining new visualization facilities. As Larkin and Simsons [29], we believe that using a well designed visualization framework can make the exploration of the simulation traces more effective.

This paper illustrates a new application domain of modelling techniques: the visualisation customization. We apply it to simulation trace visualisation in order to assist the error detection during model validation and verification. The work was triggered by feed-backs we received from industrial partners on the use of traditional model validation and verification frameworks.

The rest of this paper is organized as follows: section 2 overviews the simulation and diagnosis features used in validation tools and the diagnosis process using them. Section 3 presents an extension to this process that includes the creation of customized visualisations. Section 4 presents Metaviz – the model-driven implementation of our approach, that is evaluated in Section 5 using the SysML model of the Solar Generation System (SGS) of the Automated Transfer Vehicle (ATV), designed by Astrium Space Transportation. Section 6 overviews related work.

## 2 Model Simulation and Diagnosis: Process, Toolset, Limits

In this section we present an overview of the simulation and diagnosis features currently used in our tools, which are representative of what is available in other model simulation and validation tools. We also outline some of the limits of currently used approaches, which motivate our work.

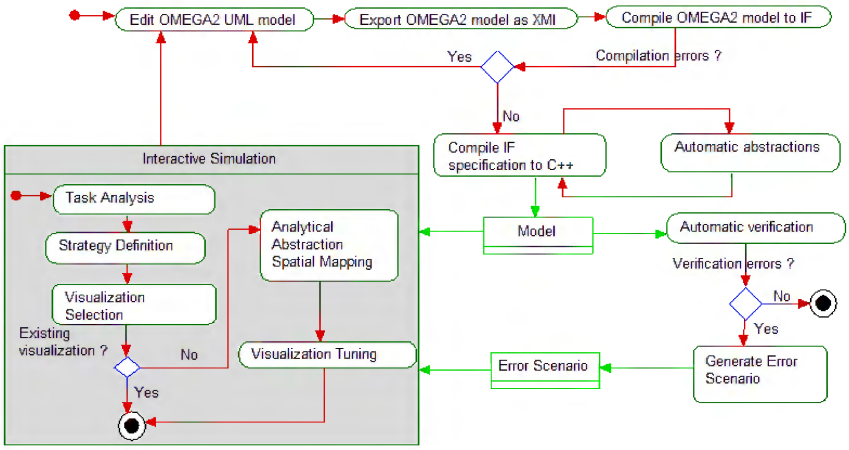


**Fig. 1.** The IFx validation process. The activity *Interactive Simulation* is refined in the new validation process to cover visualisation.

IFx-OMEGA[1] is a simulation and verification (*model-checking*) toolbox for UML and SysML RTES models [34]. The toolset relies on the automatic translation of models into a lower-level language (named IFx) based on asynchronous communicating extended timed automata, and on the use of the extensive toolset available for this language [12]. Figure 1 shows the validation workflow. The activity *Interactive Simulation* is empty in a classical setting, as it corresponds to our current contribution that will be detailed in Section 3.

The validation acts on a UML or SysML model, which is first translated to an IFx model, and then compiled[2] to an executable program that will be used for automatic verification and interactive simulation.

The interactive simulator offers the possibility to store simulation scenarios (as XML) and replay them later. Additional simulation scenarios are generated for each error detected by automatic verification. The simulation interface allows the user to fire any of the enabled transitions in each step of the trace and to analyse the current state of the System Under Diagnostic (SUD) (variable values, message queues, state machine configurations, etc.). The interface offers a set of customizable tree-based views of the model state and the trace steps (transitions) and a set of user controls for interacting with these views. The state views can reflect the structure of the system and its components either at the IFx level (timed automata) or in terms of the UML/SysML concepts (objects/blocks, ports, etc.).

The IFx validation approach has been applied to several industry-grade models such as Ariane-5 [36], MARS [37] and SGS [19], and has proven to be very effective in discovering design issues. The issues are generally related to the

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

[2] In some cases, the model can be first simplified using automatic abstraction techniques as shown in Figure 1

distributed, concurrent and timed nature of the systems, and very often relate to undesired message processing patterns such as an unexpected message order. Nevertheless, neither IFx, nor any other simulation tool known to us, propose any kind of advanced visualization of the error scenario being played. Effective information exploration always relies on some form of *overview* of the analysed data [32], yet no tool supports this for simulation scenarios.

Another limitation of currently used model simulators is that the visualizations are ad hoc, i.e. not based on a visualization reference model [38]. Consequently, any visualization customization (data, visual structures or view customization [16]) is a challenging task. Any extension of the tool, to allow new kinds of visualization needs significant coding.

Our aim is to define an approach based on a flexible reference model that will guarantee a clear separation between the *simulation trace domain* and the *visualization concerns*. This model should enable a clear building and customization process for simulation trace visualizations. The new diagnosis platform architecture should offer extensible facilities for simulation trace visualizations. In the following section we present our approach to build a new visualization facility for simulation trace visualizations. The new tool facility should be integrated in the workflow; for this purpose, we have refined the current validation process, depicted in Figure 1, by refining the process step *Interactive Simulation*.

## 3 Diagnosis Process Assisted by Visualizations

The IFx-OMEGA tool set is used for validating UML design models, using the validation process illustrated in Figure 1. One of the main steps of this process is the *Interactive Simulation,* where the user extracts *error scenarios* while interacting with the *Interactive Simulation Interface*. To assist the user in this task, we have refined this step with a *diagnosis process* built around visualization concepts. The goal of this diagnosis process is to detect simulation errors and give insight into their reasons. For this purpose we provide the user with enhanced visualization of the simulation traces. In this work, we do not focus on an error taxonomy, but rather on an effective framework for building a visualization tool to support a trace exploration and analysis techniques.

The description of the different visualization stages given by Ware in [43] is the starting point for refining the interactive stimulation step. Ware describes 4 steps in designing visualizations: (i) The first step is the collection and storage of the data followed by (ii) a pre-processing step that transform the raw data into understandable data. (iii) This derived data is then displayed to (iv) enable the user to perform a perceptual tasks on it. To be effective, this high-level process needs to be refined, such it has been done by Ed Chi [18]. We believe that an efficient visualization tool should be designed based on a good understanding of the end user task. Moreover the visualization design should be primarily focused on the user task to be supported [32,11,45]. In this spirit, the visualisation design we use is composed of the following steps as shown in Figure 1 in the refinement of the *Interactive Simulation* activity.

1. User **Task Analysis**: In this context, the user task is to diagnose a certain type of errors in the IF specifications, such as message processing errors. We base our user task definitions on existing task taxonomies such as [39] and [44]. In this step one should analyse why the user task cannot be satisfied using the current means. In our setting, as we have discussed in Section 2, the visualizations cannot be customised.
2. **Strategy Definition**: The goal of this activity is to improve the user performance using external cognition. We can take into account the human perception system and study how the error scenarios that we want to diagnose should be presented to the user (see Section 5).
3. **Visualization Selection** that would amplify user cognition [8] and would support user performance improvement strategy which was defined. In this step we choose a suitable technique to support visualization, taxonomies such as [17,8] can be used. The execution of the selected technique leads to an *Analytical Abstraction*. If no existing technique is found satisfactory, new visualization techniques can be defined.
4. Define the **Analytical Abstraction Spatial Mapping** and find which variables of the Analytical Abstraction to map into spatial position in the visual structure. In fact, space is perceptually dominant [30], thus we have to identify first which data variables should be mapped to a given spatial position.
5. **Visualization Tuning** step covers all the tuning such as mapping of the variables, not considered in the previous step, to other visual coding (marks, connections, temporal encoding, etc.). It also covers *user controls* to enable interaction with the produced visualization and *attention-reactive features* to better manage user attention [45] (e.g. color highlighting). Although very important in the user data manipulation process this step is not covered by the current work.

## 4 MetaViz: Supporting the Simulation Process

To support the extended process introduced in the previous section, we have developed a model-based software architecture framework: Metaviz.

Metaviz assists the user from choosing a visualization technique to completing the visual mapping of the analytical abstractions. We did not find any effective formalisation of the first two steps, which are by nature highly informal. Therefore, corresponding to these steps, we assist the user solely with guidelines.

In order to build the Metaviz framework we have mapped the Data State Model [25] components to MDE concepts[4]. We use Metaviz *metamodels* and model to model transformations to represent *Data Stages* and *Transformation Operators* respectively.

**The Data State Reference Model (DSRM).** Various data-oriented visualization taxonomies exist. Some of them categorise the visualization techniques based on the visualised domain data. Others, such as the one proposed by Maletic et al. [32], are task-oriented. While these categorisations give a wide and clear

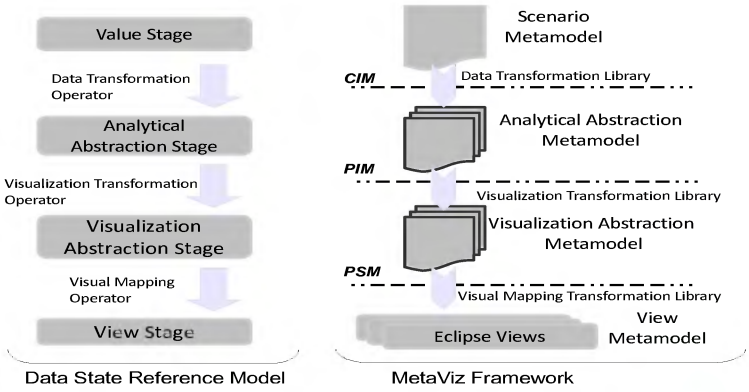


**Fig. 2.** The Metaviz architecture derived from the DSRM [17]

vision of the visualization design space, they are not refined enough to promote software decoupling and reuse in the implementation process. Another taxonomy, proposed by H. Chi [17], focuses not only on the *data types* manipulated through the visualization pipeline but also on the *visual processing operators*. The Data State Reference Model proposed here is based on (i) a set of *Data Stages* that gather the data structures and (ii) visual *Transformation Operators*.

Using a data state model, the user can build the visualization pipeline in a flexible manner, as it is possible to clearly see the intermediate results of the different transformation operators and to easily plan the future stages and transformations. Moreover, in an MDA [33] spirit this separation of concern enables the reuse of stages and operators.

Metaviz components are designed following the data state model stages and operators. Figure 2 shows the Metaviz architecture and the mapping of its components to the data state model. In this mapping, we have decoupled domain and visualization assets, thus encouraging data structure and transformation reuse.

The major strength of the Metaviz architecture is the *separation of the different visualization pipeline concerns*, that is made possible by the use of the data state model. To achieve the separation of concerns, the use of a model driven approach was a perfect choice.

The *Data Stages* were implemented by Ecore metamodels [4] and the *Data Operator Transformations* were implemented using the ATL [27,2] declarative programming style to make the visualization pipeline mappings explicit in the transformation rules.

The use of Ecore and its XMI serialization facility offers an opening point towards the use of different modeling tools. The Metaviz framework is composed of several metamodels and transformations corresponding respectively to the DSRM *stages* and *operators*. The model transformations are implementing the within and non-within *stage transformation Operators*.

**Value Metamodel** gathers the simulation trace raw data that we want to explore by the visualisation. For this we have defined a *scenario metamodel* to

*inject* [28] the XML-based simulation traces generated from the IFx automatic validation process or stored manually by the user during the interactive simulation. This metamodel gathers concepts such as fired transitions, processes, messages, etc. This metamodel can be reused or easily replaced by other descriptions [9,22] without breaking the visualization pipeline.

**Value Stage transformations** manipulate the scenario model elements and do not derive new data types. This transformation is implemented using OCL queries on the stage model elements. To execute those operators we have used the ATL *superimposition technique* [40] and a predicate-based query approach for filtering the relevant data.

**Data Transformation** are a set of ATL Transformations that transform the *Scenario* models into analytical abstractions. Any analytical technique that gives an insight into the trace data is categorized as a data transformation. For further reuse, these transformation library can be organized based on a categorisation of the existing exploration techniques, such as for instance the categories proposed by Andrienko et al. [11] : *see the whole, simplify and abstract, Look for recognisable* etc.

**Analytical Abstraction Metamodels** are defined or chosen among the existing techniques to extract meaningful information from the traces regarding relevant user task. Some widely used abstractions are communication graphs, inter-process communication patterns, event statistics. This metamodeling layer makes the capitalisation and reuse of the data analysis techniques possible. The visualization approaches often merge this layer with the visualization abstraction in the implementation phases. However we have explicitly separated this layer to enable the reuse of different analysis techniques. This layer gathers a set of ready to use trace analysis techniques. Up to now we have implemented *an inter-process communication graph* and *a trace summarizing technique.*

**Visualization Transformation Operators** produce visualizable content, mostly tree-based or graph-based structures, from analytical abstractions.

**Visualization Abstraction Metamodels** are preparing data for a set of visualization tools. It is the last step before the end-user visualization interface. A typed-node link graph is one of the mostly used abstraction in this step. A hierarchy of nodes or a more elaborated abstraction could also be used in this stage. Bull gives an overview of some widely used abstractions in [14]. The advantage of using those visual abstractions is that a set of visualization tools can share the same set of visualization abstractions.

**Visual Mapping Transformation** is the last transformation step to produce the visualization end product. It is usually implemented using geometric tools and layout techniques.

**View Metamodels** are tool specific. they gather data that is optimised for a specific visualization tool. In section 5 we will enrich this metamodeling layer with metamodels for Graphviz [6] and Zest [3]. The concept of View used in Chi's taxonomy should be refined using different models according to a tool specific criteria to enable effective implementation and reuse. Consequently we have two

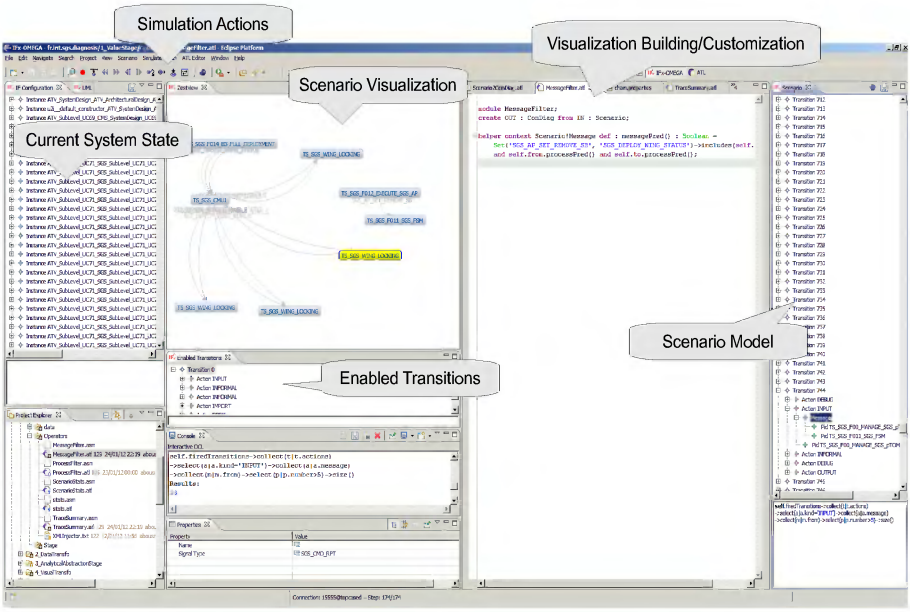


**Fig. 3.** IFx-OMEGA User Interface enhanced with Metaviz features

types of metamodels in this layer: a tool-independent and a tool-specific model. A well defined visualization process and supporting framework are step forward to enable effective diagnostic of real-time and embedded system models. But they are not enough to make the users adopting the validation tool. Metaviz has to be seamlessly integrated to the IFx-OMEGA platform and the different user roles have to be clearly defined. For this purpose we have implemented the new simulation interface on top of the Eclipse platform. The figure 3 gives an overview of this new interface [3].

## 5 Evaluation

To illustrate our visualization framework we *build a new graph-based* visualization and we *customize* an existing one using a *trace summarizing technique*. For this, we execute the extended Diagnosis Process 3 on an industrial case study.

The Solar Generation Subsystem (SGS) [19] is a software part of the ATV (Automated Transfer Vehicle) program, a spacecraft developed by Astrium Space Transportation for ESA (European Space Agency). The ATV aims at supplying the International Space Station (ISS). The purpose of SGS is to provide functional chains to realise the solar arrays deployment and their rotation.

[3] A video demonstration is available at: `http://www.irit.fr/~El-Arbi.Aboussoror/metaviz.html`

### 5.1 Building a Visualization

The complexity of the SGS model leads to an important amount of data that needs to be analysed during the diagnostic phase. Part of the diagnostic is done by feeding back verification results at OMEGA level, as it was reported in previous work [35] still this is not enough. The kind of visualization field engineers use (e.g. message communication graph), cannot be found directly at OMEGA/UML, it has to be created by combining information from the model and from the error scenarios. A simulation scenario of the SGS model is an XML file of tens of thousands lines. The listing 1.1 give a small excerpt of this file.

```
name="u2i___default_constructor_TS_SGS_F01_EXECUTE_SGS_COMMANDS" no="0" /></by>
</IfEvent> <IfEvent kind="INFORMAL" value="--create sub-component
TS_SGS_F012_EXECUTE_SGS_AP--"> <by><pid
name="u2i___default_constructor_TS_SGS_F01_EXECUTE_SGS_COMMANDS" no="0" /></by>
</IfEvent> <IfEvent kind="IMPORT" value=""> <by><pid
name="u2i___default_constructor_TS_SGS_F01_EXECUTE_SGS_COMMANDS" no="0" /></by>
</IfEvent>
```

**Listing 1.1.** SGS scenario excerpt

Obviously, the user cannot answer questions about the participation of a process to a trace and the messages it exchanges with certain processes from this kind of trace. One of the information that is useful to trace during diagnostic concerns the concrete communication occurred during the execution. For this, we create a communication diagram visualization that explores the simulation scenarios of SGS.

The first step, corresponding to the *Overview* task in Shneiderman's taxonomy [39], is to *explore* the simulation traces stored by the IFx toolset user. With the classic interfaces the user can play the entire scenario step by step, but he can not see the whole scenario communication trace in a visual form. Since the human working memory restricts the amount of information one can reason about, an effective visualization should take into account this limitation and provide features that augment user cognition by external means [45]. Thus, encoding simulation trace needs to take into account the reader (OMEGA designer) and use UML-like constructs. The reader decoding tasks should not take too much effort, in order to let the user focus on understanding the system behaviour encoded in the visualization [26].

Visualizing a huge set of objects with inter-communication relationships can be performed by extracting a *communication graph* (a simplified UML communication diagram). This enables the user to grasp the process types and the exchanged messages. A communication graph is a set of *nodes* representing process types and a set of relating *edges* representing a message passing between two processes. In the SGS case study verification, the communication diagram was instrumental in reasoning about the system and in achieving the verification goals [19]. It was used to detect clusters of objects for which abstractions could be defined and used to solve the state space explosion problem. Following the diagnosis process, we have created a visualization that is precisely characterised in figure 4. This table shows the stages and operators alongside with their implementation components.

| Stages | Operators | Description | Implementation |
|---|---|---|---|
| ***Value*** | | A metamodel describing the simulation trace data | Scenario.ecore |
| | ***Value Operators*** | Model injection, Filtering IFx internal messages and constructors | MessageFilter.atl (superimposed to Scenario2ComDiag.atl) |
| | ***Data Transformation*** | Creates communication graph from the set of messages | Scenario2ComDiag.ecore |
| ***Analytical Abstraction*** | | Inter-process communication graph | ComDiag.ecore |
| | ***Visualization Transformation*** | Creates a typed-node link graph from the communication graph | ComDiag2Graph.atl |
| ***Visualization Abstraction*** | | A typed-node link graph | Graph.ecore |
| | ***Visual Mapping*** | Creates tool specific visualization model (e.g add layout directives) | Graph2Dot.atl, Graph2Zest.atl |
| ***View*** | | A metamodel specific for each targeted visualization tool | Zest.ecore, Dot.ecore |

**Fig. 4.** Characterization of the visualization technique

The visualization produced by executing this transformation chain is rendered on 2 different tools, namely Graphviz [6] and GEF/Zest [3]. An overview of the Zest view rendering is shown at the right of the figure 5


**Fig. 5.** Filtering a simulation trace

## 5.2 Customizing a Visualization

**Message Type Filtering.** After exploring an error scenario the user may need to understand why an error is occurring in a certain scenario. It is a *Focus* type of task. We need to construct a visualization that helps in exploring inter-process communication for a restricted set of message types. To build this new visualization we have customised the previous view by adding a *filtering* operator. This operator will filter in the scenario model the messages based on their types. Since a customization is a variation of the base behaviour we have used the

*superimposition* feature of ATL to perform this filtering operations. The ATL code (a predicate on the trace messages) is given in listing 1.2. One can notice that this customization needs only few lines of code. The end product of executing the visualization pipeline is rendered on a Zest view, see figure 5 on the left. In this figure we can see the result of executing the following within-stage transformation:

```
helper context Scenario!Message def : messagePred(): Boolean=
    Set{'SGS_AP_SET_REMOVE_SB', 'SGS_DEPLOY_WING_STATUS'}
        ->includes(self.signalType)
    ;
```

**Listing 1.2.** ATL helper for message filtering

**Trace Summarizing.** To move forward in abstracting the information obtained from the error trace and illustrate the ability of Metaviz to implement more elaborated trace visualizations, we use summarising techniques such as [24]. These techniques were originally defined for the static analysing of the the call graph, but we have adapted them to build inter-process communication dynamic traces. The new customization is implemented as an ATL superimposition module. The ATL helpers in listing 1.3 were written to enable this customization.

```
helper context Scenario!Message def: messagePred(): Boolean=
    self.from.processPred() and self.to.processPred();
helper context Scenario!Pid def: processPred(): Boolean=
    self.forwardingMetric() < thisModule.threshold;
helper context Scenario!Pid def: forwardingMetric(): Real=
    let N : Integer = thisModule.allProcessSize in
      (self.fanin()/N)*(N/(self.fanout()+1)).log()/N.log();
```

**Listing 1.3.** ATL helpers for trace summarizing

The *forwardingMetric* helper computes for each process $p$ in the trace, a metric based on the set of processes that send messages to $p$ (the *fanin* helper) and the set of processes that receive messages from $p$. For more details on the metric used in this helper the reader is referred to [24].

Using an appropriate *threshold* value, we can filter the set of processes that are performing only message forwarding. Once again, building this trace summarizing visualization, only needs a few lines of ATL code. The trace summary is rendered on a *Graphviz* view in figure 6. The main processes that collaborate in the SGS model are shown in a manner that highlights the structure, which is obviously more useful than the raw XML file 1.1. The user would confront these results to the original model and see how those processes are communicating to execute the system main functionalities.

**Evaluating the Process.** The use of verification and validation techniques remains marginal in the industry. Therefore, we have not been able to evaluate Metaviz by deploying it to a large set of users and questioning them on its

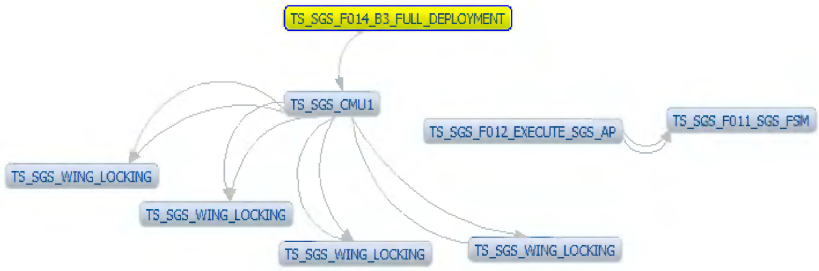


**Fig. 6.** Trace Summarizing

effectiveness and user-friendliness. The feed-backs we had from the industrial partners we worked with in originally validating SGS were very positive and encouraging, still we feel that we need a more objective evaluation basis.

To evaluate the Metaviz creation and customization approach we have compared it to an ad-hoc visualization implementation, as used in the validation of SGS, and that proved very useful for handling complexity. Several taxonomies [32,44] can successfully be used to evaluate the two approaches. Bull [14] offers a more complete and yet practical evaluation method. It consists of functional requirements and a set of design recommendations. Before performing the comparison it is important to notice the importance of defining the target audience [32] for the visualizations. We are tageting the IFx-OMEGA platform users. They are expected to be familiar with MDE techniques especially UML modelling with the OMEGA profile.

The figure 7 refers to the use of Metaviz versus the use of a command line ad-hoc implementation of the communication graph visualization. The comparison uses the evaluation approach above-mentioned.

One of the big drawbacks of the ad-hoc implementation is the use of UNIX command line utilities (e.g. sed) that target users are rarely familiar with. That makes the visualization building process difficult to understand or to customize. Adding automatically user controls to the generated visualization is not feasible, unless the viewer (Graphviz) is changed, but then the code is not more working.

The code necessary for this ad hoc visualization is given in listing 1.4.

```
cat o.aut | grep −v '"' | grep −v u2i___default | grep −v u2i___init | sed 's/^.*<<//g'
| sed 's/!.*}{/{/g' | sed 's/>>.*//g' | sort −u | sed 's/}//g' | awk '
BEGIN { print "digraph LTS {"; print " node [shape = circle];"; }
END { print "}"; }
/des.*/ { next; } // { split($1,a1,"_"); split($2,a2,"_");
print "\"" a1[length(a1)] "\"" " −> " "\"" a2[length(a2)] "\"" ";" ; }
' | dot −Tepdf > net.pdf
```

**Listing 1.4.** Ad-hoc visualization builder

In contrast, Metaviz uses ATL, a largely used model transformation engine, and targets a model driven viewer [3]. It is also seamlessly integrated to the new

IFx-OMEGA Eclipse-based interface. The specification of the visualization pipeline is explicit thanks to the declarative style used to write the transformations.

| | implementation | Ad hoc | MetaViz |
|---|---|---|---|
| **Functional Requirements** | **Data Customization** | Yes | Yes |
| | **Presentation Customization** | Yes | Yes |
| | **Control/Behaviour Customization** | No | feasible |
| **Design recommandations** | **Efficient view specification** | No | Yes, using model transformation declarative style |
| | **Integrated tool support** | No | Yes |
| | **Familiar language** | No | Yes, QVT based |
| | **Provide view model adapters** | No | Yes, using Zest framework |
| | **Use existing viewers** | Yes (Graphviz) | Yes (Zest, Graphviz, ...) |
| | **Explicit view specification** | No | Yes, using model transformation as mapping rules |

**Fig. 7.** Ad hoc and Metaviz implementation comparison

## 6 Related Work

In this section we overview related approaches on execution trace visualisation. A lot of effort is invested in program comprehension through the dynamic analysis of *execution trace visualization*. A tremendous number of approaches focus on the visualization techniques, making the tool implementation an ad-hoc exercise. Few rely on a well defined reference model. [20,25,41,21,31].

In the Eclipse implementation of De Pauw's tool, TPTP [5], only some components are model-based (e.g. the metamodel of the trace) but the tool relies on ad-hoc architecture which make the reuse or customization of the visualization difficult. Chi has proposed the DSRM and has implemented his tool around this model but the implementation was tailored to the spreadsheet visualizations and was not intended to be extended easily by new *stages*. Walker et al. [41] use models for *stages* and a mapping language for describing explicitly some *operators*, but the mapping language does not cover the whole visualization process.Thus, most of the contribution implementations make the comprehension and the reuse of the tool challenging.

Only three approaches offer a flexible Model Driven approach for customizing the visualizations [14,31,13]. Bull in [14] has taken similar approach to build his tool around a Visualization Reference Model (VRM) [15]. This Model Driven Visualization approach is suitable for visualization that do not involve complex

analytical abstractions. To implement complex transformations like execution trace summarising techniques [24], this approach needs to be refined to take into account a categorization of the operators, since the VRM is a high level data-oriented model and does not cover the whole visualization pipeline. Another similar approach is the Portolan Framework [31], but like MDV, it uses a generative approach for rendering the views, consequently a latency is introduced in the visualization prototyping loop. In contrast Metaviz promotes an interpretive approach to render the models at runtime and produce the visualizations. Finally, Buckl's approach [13] is tidily coupled to the Enterprise Architecture field and does not enable reusing the framework in visualizing simulation traces.

## 7 Conclusion

Previous work on verification and validation [36,37,19], performed in the context of the real-time systems specification and validation tool set IFx-OMEGA [34], as well as interactions with practitioners, convinced us about the need for meaningful, flexible and effective visualisations. In this paper we present Metaviz: our approach to support the user during the validation phase in performing model diagnosis.

Based on the Data State Reference Model [17], we extend *the diagnosis process for IFx-OMEGA*. The new diagnosis process includes the definition of *a set of simulation trace visualizations* that effectively help the user during the interactive simulation process .

Metaviz relies heavily on a model-driven implementation of the Data State Reference Model, in terms of a chain of re-usable model transformations and meta-models libraries, leading to customized visualisations. By defining new visualizations (e.g. message graph), the engineers would use these newly created visualizations instead of having to dig into large models (system level, class level, state machine level) and error scenarios etc...

We illustrate the effectiveness of Metaviz by applying it the validation of the industrial case study SGS [19]. It is impossible to perform a large scale evaluation of an approach that makes model based validation and verification more accessible, since the validation and verification are themselves marginally used by the industry. Therefore, our evaluation is done on a case study that was subject to verification and validation at Astrium, and we show how our visualisations helps the verification.

This work opens the way to several future research directions such as using Metaviz to implement new visualizations and an automatic mechanism for inter-process communication error pattern recognition using techniques such as [10]. Moreover, we intend to enhance the tool support, for instance by managing the visualization modelling artefacts (e.g. dedicated explorer for the *stages* and the *operators*) or by adding user controls to the visualizations. On another direction, we intend to enrich our work by coupling it with the goal-oriented verification engine existing in the framework.

## References


1. Atego Web Site, http://www.atego.com/
2. ATL Transformation Language, http://www.eclipse.org/atl
3. Eclipse GEF Zest Framework, http://www.eclipse.org/gef/zest
4. Eclipse Modeling Framework, http://www.eclipse.org/modeling/emf
5. Eclipse Test & Performance Tools Platform, http://www.eclipse.org/tptp/
6. Graphviz, http://www.graphviz.org/
7. IFx-OMEGA, http://www.irit.fr/ifx
8. Information Visualization. In: The Human Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications, p. 509. Lawrence Erlbaum Associates (2008)
9. Alawneh, L., Hamou-Lhadj, A.: MTF: A scalable exchange format for traces of high performance computing systems. In: ICPC, pp. 181–184 (2011)
10. Alawneh, L., Hamou-Lhadj, A.: Pattern recognition techniques applied to the abstraction of traces of inter-process communication. In: CSMR, pp. 211–220 (2011)
11. Andrienko, N., Andrienko, G.: Exploratory analysis of spatial and temporal data: a systematic approach. Springer (2006)
12. 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)
13. Buckl, S., Ernst, A.M., Lankes, J., Schweda, C.M., Wittenburg, A.: Generating visualizations of enterprise architectures using model transformations. In: EMISA, pp. 33–46 (2007)
14. Ian Bull, R.: Model Driven Visualization: Towards A Model Driven Engineering Approach For Information Visualization. PhD thesis, University of Victoria, BC, Canada (2008)
15. Card, S.K., Mackinlay, J.: The structure of the information visualization design space. In: Proceedings of the 1997 IEEE Symposium on Information Visualization (InfoVis 1997). IEEE Computer Society, Washington, DC (1997)
16. Card, S.K., Mackinlay, J.D., Shneiderman, B. (eds.): Readings in information visualization: using vision to think. Morgan Kaufmann Publishers Inc., San Francisco (1999)
17. Chi, E.H.: A taxonomy of visualization techniques using the data state reference model. In: Proceedings of the IEEE Symposium on Information Vizualization 2000, INFOVIS 2000, IEEE Computer Society, Washington, DC (2000)
18. Chi, E.H.H., Riedl, J.T.: An operator interaction framework for visualization systems. In: Proceedings of IEEE Symposium on Information Visualization, pp. 63–70 (October 1998)
19. Conquet, E., Dormoy, F.-X., Dragomir, I., Graf, S., Lesens, D., Nienaltowski, P., Ober, I.: Formal Model Driven Engineering for Space Onboard Software (regular paper). In: International Conference on Embedded Real Time Software and Systems (ERTS2). Society of Automobile Engineers (SAE), Janvier (2012)
20. De Pauw, W., Helm, R., Kimelman, D., Vlissides, J.: Visualizing the behavior of object-oriented systems. In: Proceedings of the Eighth Annual Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA 1993, pp. 326–337. ACM, New York (1993)
21. Favre, J.-M.: Gsee: a generic software exploration environment. In: Proceedings of the 9th International Workshop on Program Comprehension, IWPC 2001, pp. 233–244 (2001)

22. Garces, K., Deantoni, J., Mallet, F.: A model-based approach for reconciliation of polychronous execution traces. In: 37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), pp. 259–266 (2011)
23. Gotel, O., Marchese, F.T., Morris, S.J.: The potential for synergy between information visualization and software engineering visualization. In: Proceedings of the 2008 12th International Conference Information Visualisation, pp. 547–552 (2008)
24. Hamou-Lhadj, A., Lethbridge, T.: Summarizing the content of large traces to facilitate the understanding of the behaviour of a software system. In: 14th IEEE International Conference on Program Comprehension, ICPC 2006, pp. 181–190 (2006)
25. Chi, E.H.H.: A Framework for Information Visualization Spreadsheets. PhD thesis, The University of Minnesota, USA (1999)
26. Iliinsky, N., Steele, J. (eds.): Designing Data Visualizations. O'Reilly Media, Inc. (2011)
27. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Sci. Comput. Program. 72(1-2), 31–39 (2008)
28. Jouault, F., Bézivin, J., Kurtev, I.: TCS: a dsl for the specification of textual concrete syntaxes in model engineering. In: Proceedings of the 5th International Conference on Generative Programming and Component Engineering, GPCE 2006, pp. 249–254. ACM, New York (2006)
29. Larkin, J., Simon, H.: Why a diagram is (sometimes) worth ten thousand words. Cognitive Science (1987)
30. MacEachren, A.M.: How Maps Work - Representation, Visualization, and Design, ch. 8, p. 368. Guilford Press (2004)
31. Mahe, V., Perez, S.M., Doux, G., Brunelière, H., Cabot, J.: PORTOLAN: a Model-Driven Cartography Framework. Rapport de recherche RR-7542, INRIA (February 2011)
32. Maletic, J.I., Marcus, A., Collard, M.L.: A task oriented view of software visualization. In: Proc. 1st Int. Workshop on Visualizing Software for Understanding and Analysis (Vissoft), pp. 32–40. IEEE (2002)
33. Miller, J., Mukerji, J.: MDA guide version 1.0.1. omg/2003-06-01. Technical report, OMG (2003)
34. 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), pp. 373–378 (March 2010)
35. Ober, I., Ober, I., Dragomir, I., Aboussoror, E.A.: UML/SysML semantic tunings. Innovations in Systems and Software Engineering 7(4), 257–264 (2011)
36. 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)
37. Ober, I., Graf, S., Yushtein, Y., Ober, I.: Timing analysis and validation with UML: the case of the embedded mars bus manager. Journal on Innovations in Systems and Software Engineering 4(3), 301–308 (2008)
38. Robertson, P., De Ferrari, L.: Systematic Approaches to Visualization: Is a Reference Model Needed? In: Scientific Visualization: Advances and Challenges. Academic (1994)
39. Shneiderman, B.: The eyes have it: a task by data type taxonomy for information visualizations. In: Proceedings of IEEE Symposium on Visual Languages, pp. 336–343 (September 1996)

40. Wagelaar, D., Van Der Straeten, R., Deridder, D.: Module superimposition: a composition technique for rule-based model transformation languages. Software and Systems Modeling 9, 285–309 (2010)
41. Walker, R.J., Murphy, G.C., Freeman-Benson, B., Wright, D., Swanson, D., Isaak, J.: Visualizing dynamic software system information through high-level models. SIGPLAN Not. 33, 271–283 (1998)
42. Ware, C.: Information visualization: perception for design. Morgan Kaufmann Publishers Inc., San Francisco (2000)
43. Ware, C.: Information visualization: perception for design, vol. 1. Morgan Kaufmann Publishers Inc., San Francisco (2000)
44. Wiss, U., Carr, D., Jonsson, H.: Evaluating three-dimensional information visualization designs: a case study of three designs. In: Proceedings IEEE Conference on Information Visualization, pp. 137–144 (July 1998)
45. Wood, S., Cox, R., Cheng, P.: Attention design: Eight issues to consider. Computers in Human Behavior 22, 588–602 (2006)
46. Zhang, J.: The nature of external representations in problem solving. Cognitive Science 21(2), 179–217 (1997)