From BPEL to SAL And Back: a Tool Demo on Back-Annotation with VIATRA2

Ábel Hegedüs, István Ráth and Dániel Varró
Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary
Email: {hegedusa,rath,varro}@mit.bme.hu

***Abstract*—Model-driven analysis aims at detecting design flaws early in high-level design models by automatically deriving mathematical models. These analysis models are subsequently investigated by formal verification and validation (V&V) tools, which may retrieve traces violating a certain requirement. Back-annotation aims at mapping back the results of V&V tools to the design model in order to highlight the real source of the fault, to ease making necessary amendments.**

In this tool demonstration we present an end-to-end V&V tool for BPEL business processes that includes complex back-annotation support for representing V&V results as process execution traces in the design environment.

***Keywords*-back-annotation; traceability modeling; dynamic traceability**

I. Overview

Model transformations are increasingly involved in various fields of software engineering, from business process modeling through formal verification to code generation. The models acting as source and target for the transformations often represent different domains, thus identification of correspondence between them is non-trivial. Although in the field of critical systems and services the precise recording of *traceability information* is a strict requirement, in most industrial environments only ad-hoc solutions are used for handling this information.

Throughout the lifecycle of a system or product traceability information is generated and used for various tasks. The correspondence records are most often created at the time when the target model is produced using the source model during the execution of the transformation. This information can be later used for validation, verification, change management, maintenance or back-annotation. It is important to note that the traceability information itself may be accessed with model transformations thus *model-based traceability solutions* are advantageous.

Model-driven analysis (illustrated in Figure 1) aims at revealing conceptual flaws early in the design process. In the typical approach, high-level design models (UML, BPEL [1], SysML, etc.) are automatically transformed into mathematical models (e.g. Petri nets, transition systems, process algebras) to carry out analysis by formal methods.

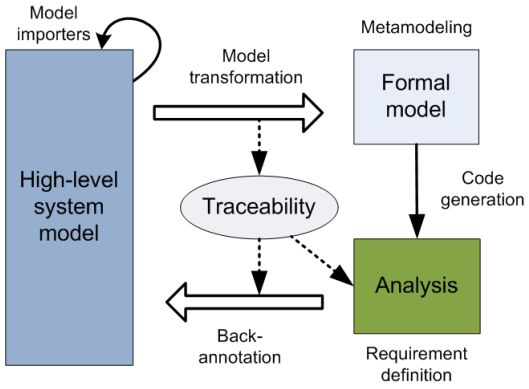


Figure 1. Methodological overview

The results of the analysis are then attempted to be back-annotated to the original source model to highlight flaws directly in the design models.

In case of dynamic modeling languages (e.g. statecharts, workflows), the back-end formal analysis tools frequently carry out simulation or model checking to ensure the functional correctness of the design using analysis models like Petri nets, process algebras or labeled transition systems. As a result, back-end analysis tools retrieve an execution trace (run) of the system as a designated or counter example.

A. Back-annotation

Counter-example traces can be very complex, resulting in well over 100 elementary steps in industrial scenarios. As a result, systems designers need to bridge a significant conceptual gap when they try to interpret what a counter-example means in the *original* design model. Back-annotation aims at automatically mapping back the results of V&V tools to the original design model in order to highlight the real source of the flaw.

Due to the semantic differences between high-level design models and lower-level formal analysis models, we argue in this paper that the *general back-annotation problem aiming to map a trace of a target model to a trace in the source model* can be very complicated. This is due

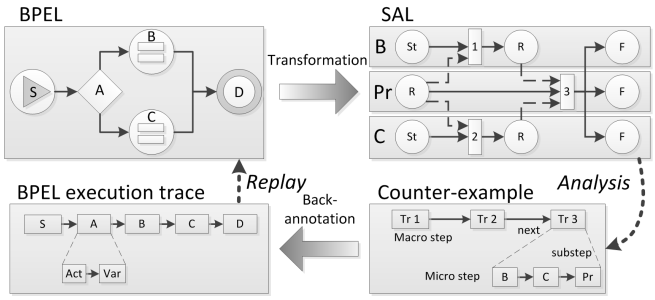


Figure 2. BPEL Verification overview

to the fact that most traditional source-to-target (design-to-analysis) model transformations carry out an abstraction, thus they are not reversible. However, in order to completely hide the underlying formal model from systems engineers, model-driven analysis need to provide automated support the back-annotation of target (analysis) models to the source (design) model.

Unfortunately, existing back-annotation approaches are either dedicated to the source and target languages, or they make strong assumptions on the back-annotation problem.

B. BPEL to SAL and back

In the demonstrated tool, BPEL business processes can be verified against requirements and the results are presented as an execution run of the business process. Since the BPEL standard does not define precise formal semantics for business process, formal languages (such as Petri nets or labeled transition systems) are used to formalize their behavior. In our tool, BPEL processes are transformed (using the VIATRA2 framework [2]) to the labeled transition system language of the Symbolic Analysis Laboratory (SAL) [3] model checking framework (illustrated in Figure 2). Then the analysis is carried out using SAL and results in a counter-example which contains a sequence of transitions which violate a requirement. The counter-example is back-annotated to a BPEL execution and can be replayed in the high-level domain.

We developed a model-based approach for persisting, handling and back-annotating execution traces (e.g. counter-examples, simulation traces). The foundation of the approach is a *generic metamodel for execution traces* which can describe the traces of various discrete event-based languages. On top of this generic metamodel, we defined *language-independent methods for navigating* through traces, and *replay* their effects on the model itself (i.e. the SAL transition system). Furthermore, the *back-annotation of traces* is possible by mapping the steps of the analysis trace to steps of the original trace [4].

Generic trace metamodel: While execution traces are traditionally interpreted as a sequence of elementary operations, in our approach, we use hierarchical trace models (see Figure 2) consisting of *micro steps* (atomic operations, m) and *macro steps* (complex operations, M), which is compliant with recent approaches [5] to define semantics for big-step DMLs like statecharts.

In the example illustrated in Figure 2, a macro step of the counter-example is firing a transition (e.g. 1) in the SAL system, while the micro steps are variable assignments (e.g. variable B changes from St to R). In the BPEL execution trace, a macro step is some activity event (e.g. B starts, A finishes), while micro steps are either activity state changes (Act) or variable manipulations (Var).

Trace replay and Visualization: The demonstrated tool is able to replay the persisted execution traces of both analysis and design models as long as they conform to the generic metamodel. Replaying can be navigated through a user interface component of the Eclipse framework where arbitrary BPEL processes and associated traces are handled. The tool also includes an intuitive graphical representation of execution trace replaying with a modified Eclipse BPEL Designer [6]. The activities and variables of the BPEL process are colored based on their runtime state.

C. Target audience amd benefits of using the tool

We believe that our tool demonstration can be beneficial to the following groups:

- Researchers and industry members with an interest in formal analysis and verification, as our tool includes both a V&V method and an intuitive approach to represent analysis results in the design language context.
- Users of the BPEL language and generally process modeling practitioners, who can benefit from a presentation on how processes can be verified design-time using hidden formal methods.
- Model transformation enthusiasts, who can gain an insight on the use of the VIATRA2 framework in a complex scenario including back-annotation and modeling of execution traces.

The benefits of using the presented tool is the ability to exploit the features of hidden formal methods for verifying business processes with precise formal models while being able to examine the results in the original design perspective.

II. TOOL DEMO CONTENTS

The tool demonstration is separated into three main parts. First a short preliminary part introduces the main languages and techniques used through the demonstration. Next a longer part deals with the presentation of the tool itself and its uses in a similar order to model-driven engineering methodology (see Figure 1). Finally, the underlying model transformation techniques are detailed using the VIATRA2 framework.

A. Preliminaries

1) **BPEL Overview**
   First, we give a brief introduction on the BPEL process description language and show the usage of the Eclipse BPEL Designer developer tool with creating a sample business process that we will use as an example in the rest of the tutorial.
2) **VIATRA2 basics**
   Here, we introduce the VIATRA2 model transformation framework including its model space, metamodeling capabilities, and transformation engine. We use the created BPEL process as an example to show how external models are imported into the model space as static models and how VIATRA2 is used for code generation to create external files.
3) **Model checking with SAL - a quick overview**
   Finally, we present the SAL model checking framework and its transition system description language. We show how linear temporal logic theorems are used to define requirements on the transition systems and are verified by the SAL model checker. Furthermore, we focus on the counter-examples that are the result of model checking and how they represent execution traces of the SAL model.

B. Applying hidden formal methods

1) **Verification Tool UI and features**
   First, we show our approach in creating a user interface for hidden formal methods. The created tool is embedded in Eclipse and incorporates all the features detailed above. BPEL processes can be transformed to SAL and common requirements can be defined without expertise in temporal logic. Requirements can be verified using the SAL model checker from the same tool and the results can be viewed as well.
2) **Customizing the modeling tool**
   The BPEL Designer tool provides only process definition support, it cannot show the dynamic state of a process instance. Here, we briefly outline how we modified the tool to be able to give graphical representation to the dynamic state of BPEL processes.
3) **Trace handling tool and features**
   Finally, we present the execution trace controller tool that uses the output of the SAL2BPEL transformation to update the dynamic state of the BPEL process instance in the customized BPEL Designer tool.

C. Underlying VIATRA2 technology highlights

1) **Metamodeling**
   First, we show how the operational semantics and the dynamic behavior of a language can be used to define metamodels in which trace information can be persisted during simulation execution or using a counter-example after model checking.
2) **Transformation development**
   In this part, we present the transformation language of VIATRA2 and its main features which are used to implement the trace replaying and back-annotation.
3) **Visualization**
   Finally, we show the visualization capabilities of the framework, that can be used for visualizing model and pattern graphs. Domain-specific graph layouts help in debugging the transformations during development, while they are also used for visualizing static and dynamic traceability models in order to aid verification and back-annotation [7].

III. TOOL AVAILABILITY AND MATURITY

The demonstrated tool is partially the result of the SENSORIA European project where it was also used as a demonstration tool [8] for the traceability visualization and back-annotation capabilities of the VIATRA2 framework. The tool itself is the result of a year of research and development, and is available as an Eclipse update site, since it is entirely developed as Eclipse components. Although the SAL model checker framework is not Eclipse-based and has to be installed and configured independently, it is a completely free tool available from the Symbolic Analysis Laboratory Website[1]. A simple webpage describing the tool and the installation procedure is available[2].

IV. SCREENSHOTS FROM THE TOOL

In this section we present the actual user interface of the BPEL Verification and Animation tool and the underlying VIATRA2 framework through screenshots.

Figure 3 shows the user interface of the BPEL Verification Tool, where the structural transformation (from BPEL to SAL) is executed and requirements can be defined, which are then verified against the business process. Generic requirements can be selected from a drop-down list thus relieving the user from using Linear Temporal Logic [9] expressions. Process-specific requirements can be captured using the LTL expression field, which also includes a basic syntax checker.

The tool provides an option for choosing either the symbolic or the bounded model checker of the SAL framework (which may have different performance on the same LTL expression). Finally, verification of a given requirement can be started with the *Check Property* button. Since model checking may take a long time for complex processes, the verification runs as a background task.

Figure 4 shows a part of a counter-example returned by the SAL model checker for a requirement that is violated by the business process. The LTL expression for the requirement (variable is never read while uninitialized) is highlighted in the upper part. The steps of the counter-example are presented in a textual format (retrieved straight

[1] http://sal.csl.sri.com/

[2] http://mit.bme.hu/~hegedusa/exectraces/

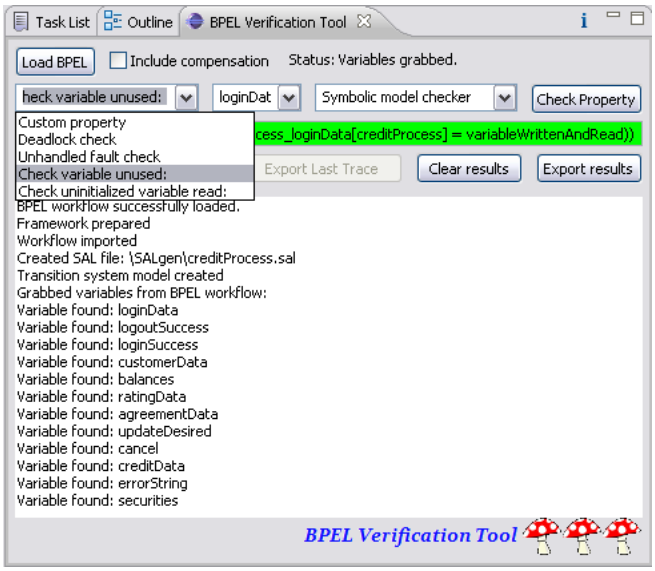


Figure 3. Aided business process requirement definition

from the SAL output) at this stage. Each step (e.g. *Step 90-91*) contains information about which transition fired (and where it is located in the transition system description) and the values of variables that changed as a result of the transition. The counter-example can be exported at any time into a file with a click of a button (*Export Last Trace*).

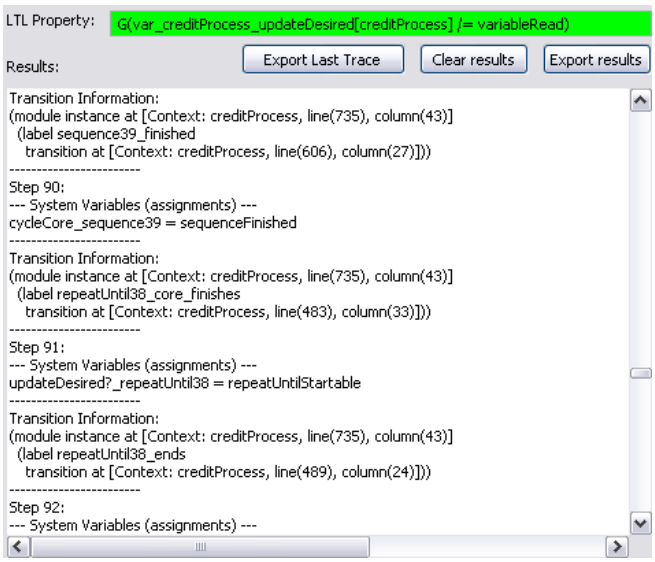


Figure 4. Verification results before back-annotation

Figure 5 shows the BPEL Animation Controller view, where exported counter-examples (traces) can be opened (*Load Trace*). After the textual file is processed, the VIATRA2 framework initializes the trace models and the back-annotation transformation. When the framework is ready, the navigation buttons can be used to animate the process execution. Apart from step-by-step navigation (*Step back/forward*), the tool also includes continuous animation mode (*Animate!/Stop*), quick return to the initial state (*Reset*) and animation speed-up (*Fast stepping*) for easier handling of long traces. Finally, the underlying model space can be saved for further use (*Save Modelspace*).

Figure 5. Animation controller

Figure 6 shows the customized BPEL Designer at a given state during the animation of an example BPEL process. The activities and variables of the process are colored depending on their dynamic state. Thus the counter-example of the model checker can be observed visually in the design perspective as an execution of the BPEL process. For the activities, light blue means *startable* state, light green *active*, dark green *finished*. For variables, yellow is *uninitialized* state, green is *correct* and red is *faulty*.

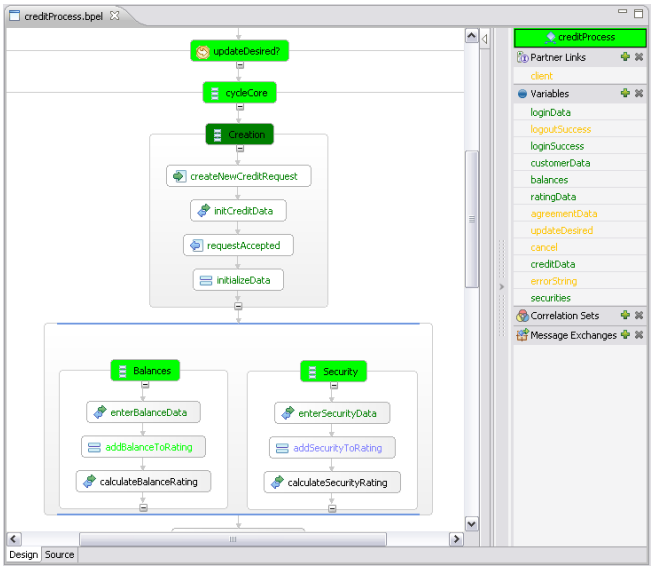


Figure 6. Animation of execution trace

Figure 7 shows the default model space editor of the VIATRA2 framework. Various models are stored in a containment hierarchy visualized in a tree view and model transformation programs can operate on the whole model space. Note that the models for the BPEL process and SAL systems are in different subtrees and the static, dynamic and trace models are separated as well. The metamodels which the various models conform to are also stored in the same model space.

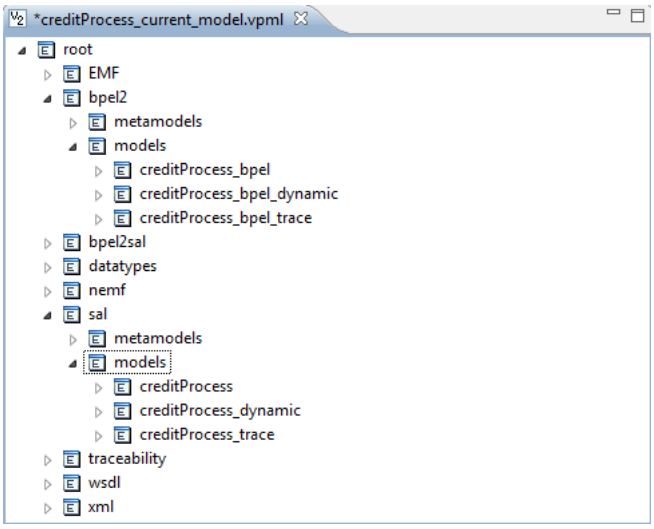


Figure 7. Modelspace view (BPEL and SAL models)

Figure 8 shows the static traceability model presented using a domain-specific layout through the model space visualization component of the VIATRA2 frameworkvisualization framework. A relevant subset of BPEL model elements are grouped on the left, the records of the static traceability model are placed in the middle, while corresponding SAL model elements are displayed on the right.

Figure 8. Visualized static traceability model (BPEL and SAL models)

Figure 9 shows the dynamic traceability model using the same visualization component, though with a different layout. The steps and substeps of the BPEL trace model are grouped on the left, while the steps of the SAL trace model, which are used in the back-annotation transformation are displayed on the right.

ACKNOWLEDGMENT

This work was partially supported by the EU project SecureChange (ICT-FET-231101) and CERTIMOT (ERC_HU_09).

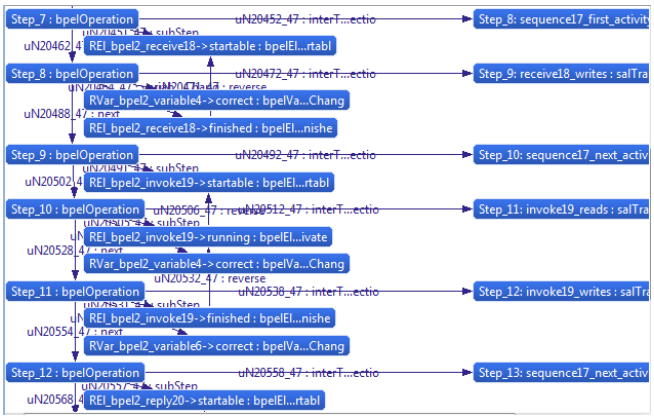


Figure 9. Visualized dynamic traceability model (BPEL and SAL traces)

REFERENCES

[1] OASIS, "Web Services Business Process Execution Language Version 2.0 (OASIS Standard)," 2007, "http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html".

[2] Fault Tolerant System Research Group, BME, "VIATRA2 Model Transformation Framework, An Eclipse GMT Subproject," http://www.eclipse.org/gmt/VIATRA2/.

[3] S. Bensalem, V. Ganesh, Y. Lakhnech, C. M. noz, S. Owre, H. Rueß, J. Rushby, V. Rusu, H. Saïdi, N. Shankar, E. Singerman, and A. Tiwari, "An overview of SAL," in *LFM 2000: Fifth NASA Langley Formal Methods Workshop*, C. M. Holloway, Ed., Hampton, VA, jun 2000, pp. 187–196.

[4] Á. Hegedüs, I. Ráth, and D. Varró, "Back-annotation of Simulation Traces with Change-Driven Model Transformations," in *Proceedings of the Eigth International Conference on Software Engineering and Formal Methods*, 2010, accepted. http://home.mit.bme.hu/~hegedusa/assets/publ/sefm10-back-ann.pdf.

[5] S. Esmaeilsabzali and N. A. Day, "Prescriptive semantics for big-step modelling languages," in *Fundamental Approaches to Software Engineering, 13th International Conference, FASE 2010, Proceedings*, ser. LNCS, D. S. Rosenblum and G. Taentzer, Eds., vol. 6013. Springer, 2010, pp. 158–172.

[6] "Eclipse BPEL Designer, An Eclipse Project," http://www.eclipse.org/bpel/.

[7] Á. Hegedüs, Z. Ujhelyi, I. Ráth, and Á. Horváth, "Visualization of Traceability Models with Domain-specific Layouting," in *Proceedings of the Fourth International Workshop on Graph-Based Tools*, 2010, accepted.

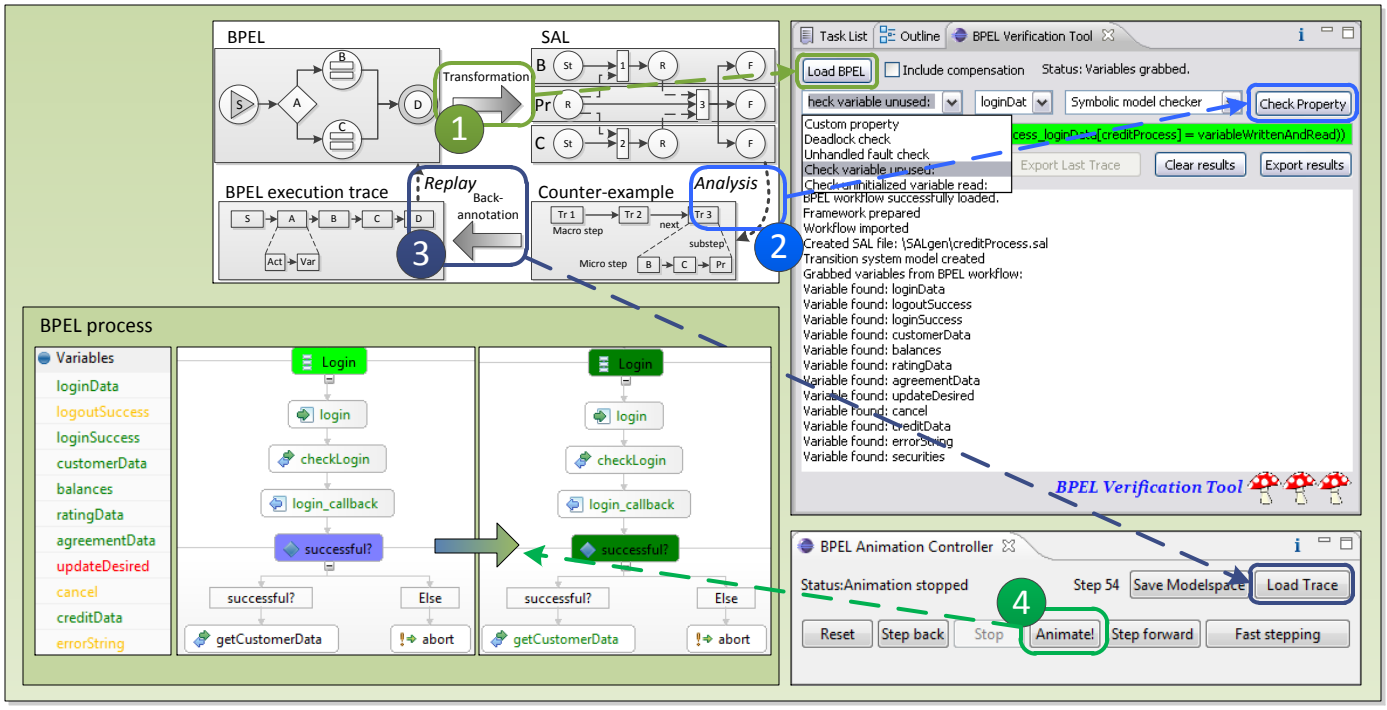
[8] L. Gönczy, Á. Hegedüs, and D. Varró, "Methodologies for Model-Driven Development and Deployment: an Overview," in *Rigorous Software Engineering for Service-Oriented Systems: Results of the SENSORIA project on Software Engineering for Service-Oriented Computing*, M. Wirsing, Ed. Springer-Verlag, 2010, to appear.

[9] E. A. Emerson, *Temporal and Modal Logic*. Elsevier, 1990, vol. B, Formal Models and Semantics, pp. 995–1072.

BPEL to SAL and Back: An End-to End Business Process Verification Tool

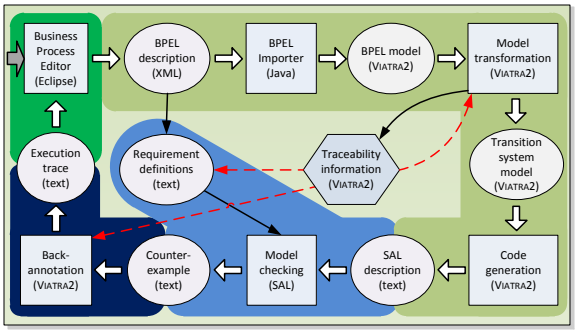
Ábel Hegedüs, István Ráth, and Dániel Varró (Budapest University of Technology and Economics)
{hegedusa,rath,varro}@mit.bme.hu

Business processes orchestrated from web-services using the Business Process Execution Language standard (BPEL) are widely used in business-critical and high-availability systems. In order to detect design flaws early, various approaches exist for *transforming BPEL processes* into mathematical models on which *analysis* (verification & validation) can be carried out, e.g. with the Symbolic Analysis Laboratory (SAL) framework. The results of the analysis (often a sequence of steps, i.e. counter-example or *trace*) are *back-annotated* into the original BPEL process, where they can be *replayed* for the developer. By providing automated solutions for these steps, *business process verification is possible using hidden formal methods*.



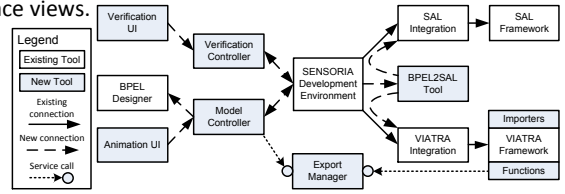
Methodological overview

The initial input of the tool is a *BPEL process description* (stored as an XML file), often created using a graphical editor, which is imported into the model space of the VIATRA2 model transformation framework using a Java-based *importer*. The resulting *BPEL model* is *transformed* into a *transition system model*, then the *SAL description* is created from this model using *code generation*. The *model checking* capabilities of the SAL framework is used to verify the BPEL process against *requirements* (defined in Linear Temporal Logic). The results (*counter-example*) are imported into VIATRA2 and transformed into BPEL trace (*back-annotation*). Traceability information created during the transformation is stored as a model and is used repeatedly throughout the method.



Integrated toolchain

The complete verification and animation tool is built using the Eclipse framework by integrating existing techniques and creating new components and links between the parts of the toolchain. The components on the left side of the figure provide the user interface and are connected with two controllers. The controllers handle the low-level transformation and verification tools in order to prepare data for the user interface views.



Tool features

- Transform BPEL business processes into a precise and formal SAL transition system.
- Guided definition of generic and process-specific requirements.
- Automated model checking of defined require-ments using the SAL framework.
- Model-based persistence and handling of SAL and BPEL simulation traces.
- Automated back-annotation of SAL simulation traces into BPEL execution traces.
- User-guided, graphical animation of BPEL execution traces based on SAL traces.
- Graph visualization of static and dynamic traceability models using domain-specific layout algorithms.

Publications & Links

- L. Gönczy, Á. Hegedüs, and D. Varró, "**Methodologies for Model-Driven Development and Deployment: an Overview**," in *Rigorous Software Engineering for Service-Oriented Systems: Results of the SENSORIA project on Software Engineering for Service-Oriented Computing*, M. Wirsing, Ed. Springer-Verlag, 2010
- Á. Hegedüs, I. Ráth, and D. Varró, "**Back-annotation of Simulation Traces with Change-Driven Model Transformations**," in *Proceedings of the 8th International Conference on Software Engineering and Formal Methods*, 2010
- Á. Hegedüs, Z. Ujhelyi, I. Ráth, and Á. Horváth, „**Visualization of Traceability Models with Domain-specific Layouting**", in *Proceedings of the 4th International Workshop on Graph Based Tools*, 2010

- Tool available at: http://mit.bme.hu/~hegedusa/exectraces/
- VIATRA2: An Eclipse GMT Subproject: http://www.eclipse.org/gmt/VIATRA2/
- Symbolic Analysis Laboratory: http://sal.csl.sri.com
- Eclipse BPEL Project: http://www.eclipse.org/bpel/

Budapest University of Technology and Economics
Fault Tolerant Systems Research Group
H-1117 Budapest, XI. Magyar Tudósok krt. 2
Phone: (+36 1) 463 3579

MŰEGYETEM 1782

SEFM'2010
13-16th September,Pisa