Proceedings of the 8th International Conference on Applied Informatics
Eger, Hungary, January 27–30, 2010. Vol. 2. pp. 417–427.

Let's Go the Verification Engineering*

Anna Medve[a], György Orbán[a], László Kozma[b]

[a]Department of Electrical Engineering and Information Systems, University of Pannonia
email: medve@almos.vein.hu

[b]Department of Software Technology, Eötvös Lóránd University
email: kozma@ludens.elte.hu

Abstract

This paper presents our experience on tooling and on processing the verification tasks, in one part with semi-automated model verifier integrated in modelling tools, and in another part with individual model checker tools. The goal of experiment is the method for verification tasks during Model-Driven Development software processes and to achieve the correct by construction principle.

The tools used for validation is the Telelogic Tau G2 tools, and for formal verification and validation is the VERIMAG IFx toolbox associated by the IF (Interchange Format) specification and description language. The integrated semi-automated validation tasks in a general-purpose commercial Tau G2 tools has been done by model-based automatic generation of the Model Verifier GUÍ and model-based semi-automated generating test cases. The formal verification tasks with VERIMAG IFx toolset it consist of engineering the verification environment, and processing several type of formal verifications. The built IF verification environment integrates some specific-purpose tools for modelling and static analysis, like ArgoUML, VERIMAG IFx toolset, and for model checking like SPIN and CADP. The main result of our experiment is the position to set out the method for engineering verification tasks and artefacts, and to provide intuitive methods for such a model to be visualised and possible incorrect behaviors to be earlier detected with state space reduction and deadlock checking.

Keywords: verification engineering, model-based testing, verification tools, MDA process, Telelogic Tau G2, UML2.0, VERIMAG IFx, SPIN, CADP

*This research work was supported by TÁMOP-4.2.1.B-09-1-KMR-2010-003.

1. Introduction

Nowadays, specification languages have an important role beyond model checking to perform verification at design time [9]. Validation early in the development cycle can improve the software quality and decrease the software development costs. The specification-oriented style of formal methods supporting correct by construction invokes the importance of tools. The question is how can we use specification languages and tools to express multiple dimensions of design and to prove that is correct? Most commercially tools promote productivity improvements by a process-driven automation with simulation and validation from formal domain specifications. Another's take many generalization efforts producing also code from formal specification.

The paper aims to address a method for verification tasks during model-based development processes. It presents our experience obtained by engineering formal verification tasks: first in a process with semi-automatic ways integrated in a general-purpose commercial tool and second in a built verification environment from individual tools, partly open-source. We present the verification workflow with these tools and we analyze their reuse and model-transformation capabilities.

The remainder of the paper is organized as follows Section 2 presents the VendingMachine case study for the model-based semi-automated verification process integrated in the general-purpose commercial Tau G2 [2]. In Section 3 are presented the VendingMachine case study in model-based verification tasks with VERIMAG IFx toolset[7,8] for engineering the verification environment and for processing the verification by combining tools, like ArgoUML[3], VERIMAG IF Static Analyser[6], SPIN[4] and CADP[1] [5], respectively.

2. Integration of verification within a scenario-based and state-based modelling process with Telelogic TAU G2

The Section contains our experience in verification processes obtained in semi-automated way within the Telelogic Tau G2 general-purpose commercial tools that it supports model design and verification in one framework. It is used for very large industrial and scientific purposes. For example in the year 2004 the 44% of the production of software in the world were made by Telelogic Tau tools. The specification language of Tau G2 is the UML2.0. It supports Model-Driven Architecture (MDA) process with model-based testing automated by code generation for Model Verifier artifacts and test simulations. [2].

The Tau G2 Verificator engine supports a model-based semi-automated verification process to model the test, create test cases and perform the test. The

[1] CADP - Construction and Analysis of Distributed Processes

verification engineer has to define the signals and signal parameters (sender, receiver, channel, value) for test cases generated from the design model.

We present the process of verification integrated in Tau tool [2]. The verification is supported by the GUI of Telelogic Tau G2. For this we must follow to create a new project "UML for Model Verification" and to create the domain model based on the requirements. This forms the Design Model of the system with UML diagrams in terms of an embedded system like signals, interfaces, and components.

The verification tasks are supported by Model Verifier artifact. For this we have to build a new artifact that gives a new GUI named Model Verifier, see at Figure 1. The building tasks are for configuring the Model Verifier for some behavioral model elements and the target elements. The Model Verifier basically works with state machine and sequence diagrams. Add-in for activity diagram is supported by Telelogic Tau G2 license supplements. The target elements are the simulation kind (real time, standard, with environment), the target kind (win32, win32 gcc, Solaris gcc, Linux gcc). For this a supported C compiler is necessary (Visual Studio, Gcc).

The Model Verifier GUI starts with sequence diagram and the engineer has to configure the scenario-based verification (see at Figure 1, bottom right). Figure 1 illustrates the image of Model Verifier GUI for the test model and the testing process based on sequence charts.

On the left of Model Verifier windows the engineer can see the model elements, which are from the design model. On the right there is the scenario sequence view. The whole system behavior can be followed on the dynamically built sequence chart during the test case run. On the bottom there are tree windows for scenario choice, textual view of trace and signal matrix windows. The Model Verifier supports the saving of the test case model (test sequence and traces), to replay a scenario later for some motivations (i.e. to perform after the fault corrections, to reuse in MDA process).

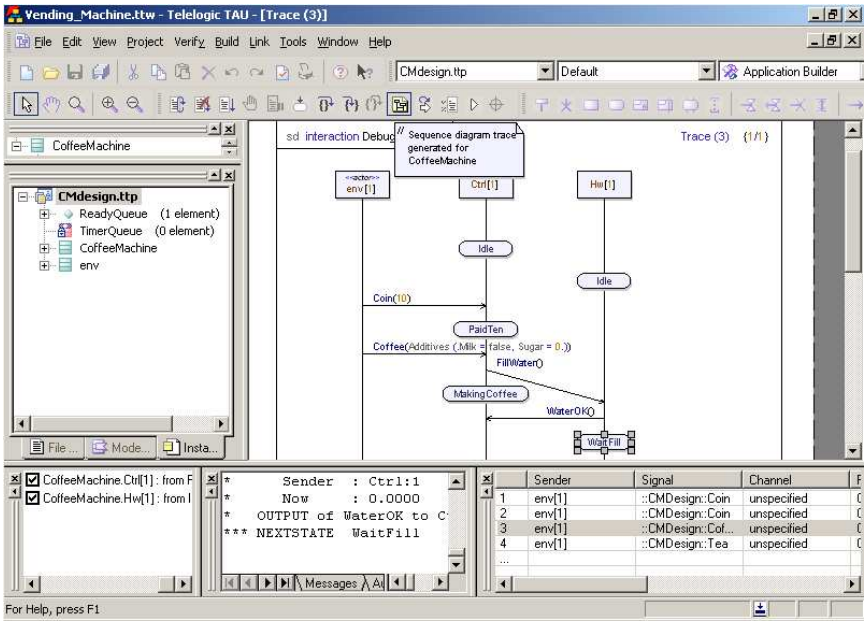

Figure 1: The scenario-based verification in the Model Verifier GUI built in Telelogic Tau G2

If the verification engineer wants to follow, how the system works, it needs to doing the state-based verification process (see Figure 2). If the verification engineer wants to follow, how the environment interacts with the system, (ie. how the system has working based on the user requirements), it needs to do the scenario-based verification process (see Figure 1).

The testing has been done with the simulations of signals. The verification engineer is who proceeds to send the signals to the system. The behavior (states, transitions) can be seen on the trace dynamically (scenario-based or state-based see in Figure 1 or 2). If a fault occurs, the textual view shows what is the failure (i.e. No instance scheduled for a transition, Signal caused an immediate null transition). The textual view (Figure 1, bottom middle) is used to trace the system, the engineer can see the states, the triggers, the signals, the sender of the signals and what will be the next state (Figure 2, bottom middle, too).

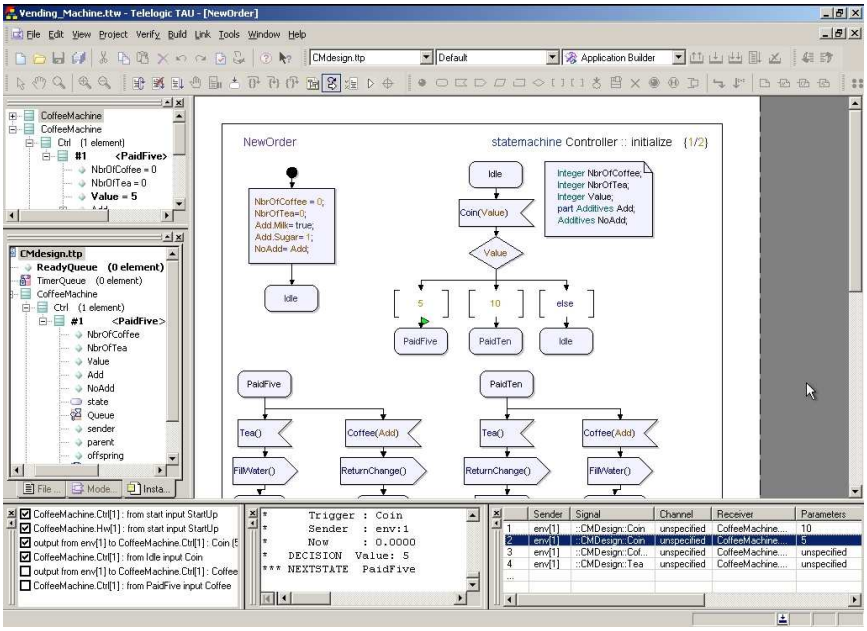

Figure 2: The state-based verification in the Model Verifier GUI built in Telelogic Tau G2

The case study had taken for the well-known vending machine, modeled as embedded system with controller and hardware part, which gets the signals from the environment (the machine User). User choice is to purchase a coffee with or without additives. The machine initially is in Idle state, the user puts coins that trigger the system's actions (between environment, controller, and hardware)

Figure 2 illustrates the image of Model Verifier GUI generated from Vending-Machine model to form the test model and the testing process based on state machines for the verification of deadlock properties in a state-based verification. The state-based trace (see at middle in Figure 2) shows from a system view the states, signals, variables, conditions and actions of the controller state-machine verification, animated during the state-based verification process.(see Figure 2). From the system's environment (the user) the controller gets a signal when the user puts a coin, and go to the PaidFive state.

The verification is semi-automated, the verification engineer is who starts the test cases to do, i.e. have sent to the system another signal (Figure 2 bottom left), i.e. the customer wants to buy a coffee. The response from the system, (in Figure 2 bottom middle) is that the VendingMachine gives back the money (it sends back a signal to the environment) because it was not enough.

The whole behavior of VendingMachine behavior can be followed within the scenario-based validation from the dynamically built sequence chart during the test case run (see at Figure 1).

The refinement of design model after the verification is processed in design view.

For example to use more parameters in a signal, like coffee with milk and more sugar, we have to extend the model in the design view. Its verification do at the same manner, with building the Model Verifier also the verification by refined steps it is the iteration of semi-automated test case specification and verification.

The usability of Model Verifier GUI and the tools-based MDA process in Telelogic Tau G2 general-purpose commercial tools has very useful for non-experts and experts too. The verification artefacts are reusable parts of the model-driven development process handling it within reusable system modules.

3. Verification environments and tasks from integration of specifically built individual formal verification tools from VERIMAG IF (Interchange Format) toolset

In a formal verification with individual tools, the main tasks of a verification engineer are to build the verification environment, to create the specification and configuration files, and to analyze verification traces [1]. To build thus a verification environment it consists of to integrate some individual tools by deploying from three categories of tools:

1. Front-end tools for model transformation interfaces with higher-level languages and with other validation tools

2. Static analyser tools for the state space reduction before model checking.

3. Behavioural tools for simulation, verification of properties, automatic test generation.

Figure 3 shows the architecture of verification environment that we integrate from tools as Verimag IFx toolbox, SPIN, CADP, GraphViz, ArgoUML. This architecture is compiled using Debian Linux and should be easy to port to other platforms as well.

The **VERIMAG IFx (Interchange Format) toolbox** is developed at VERIMAG for modelling and validating distributed systems [6, 7]. The descriptions in IF language is used as a format for inter-connecting model-based tools by front-end tools of IFx toolbox.

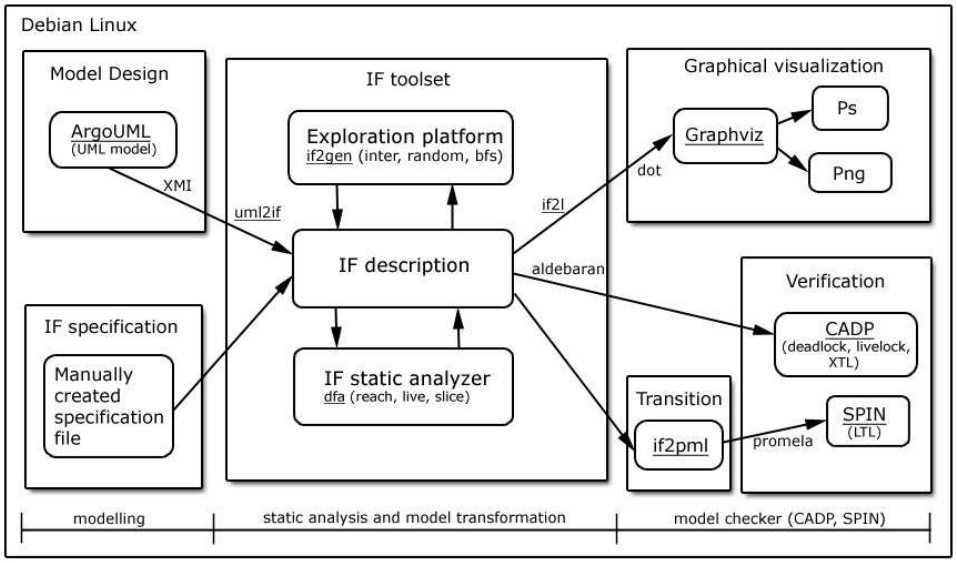

Figure 3: The verification environment built from Verimag IFx, SPIN, CADP, GraphViz, ArgoUML

CADP ("Construction and Analysis of Distributed Processes") is a popular toolbox for the design of communication protocols and distributed systems, contains several model-checkers for various temporal logic and mu-calculus, such as XTL (eXecutable Temporal Logics). CADP is developed by the VASY team at INRIA Rhone-Alpes and connected to various complementary tools. CADP is maintained, regularly improved, and used in many industrial projects [5]. **SPIN** is a general tool for automated verifying the correctness of distributed software models. [4]. It was written by Gerard J. Holzmann.

The **Graphviz** (short for Graph Visualization Software) is a package of open source tools initiated by AT&T Research Labs for drawing graphs specified in DOT language scripts. It also provides libraries for software applications to use the tools. It supports the PostScript (ps) and Portable Network Graphics (png) output formats. We integrated the Graphviz in IF environment because the Verimag IF models do not have a graphical representation.

The **ArgoUML** is an UML modeling tool from Open Source Development project [3]. It runs on any Java platform and it uses the XMI open file format (XML Model Interchange format).

3.1. The verification engineering's tasks phases in the built IF environment

Modelling phase in this case consist for obtaining the IF (Interchange Format) model of the system. An IF model defines the structure of a system with a set of

communicating processes, their behaviour with state machines, and the real time constraints with clock variables and guard conditions on them.

Standard-based modelling tools have an important role in XMI translation into IF model. In the case when the used modelling tools cannot support XMI translation, or the model is not given, the IF model of the system has to be create manually. For more details on the IF language and its semantics see to [8].

Static analysis and model transformation phase has two act. First act consist of starting from IF model as input to generate with if2gen the IF Simulator and to create IF descriptions of the configuration files and to perform static analysis (reach, live, slice) iterations. Iteration for static analysis has been done in command mode from IF Static Analyser, there is no GUI. In a static analysis action, the tool drives the back-end IF Simulator, and translates the validation results back to the level of the original model, refactoring them. Such a model can be visualised (if2l, dot) with GraphViz to observe and understand how the system works. The simulation means have been walking in the state space, random or interactive, where the verificator can decide which should be the next state, the actual state and the variables together their actual values. The possible incorrect behaviors can be detected with the IF toolset and the refactored model has resulted in some iterations. The second act of this phase is to translate the refactored model with Verimag IF toolset translators into input for model checkers individual tool, like if2pml and if2adb into CADP or SPIN for searching deadlock and livelock.

Model checking phase consist for reporting and checking of the translated file from static analysis phase. The verification tools operate as a simulator, following one possible execution path through the system and presenting the resulting execution trace to the verificator.

3.2. The VendingMachine case study in IF environment

The Verimag IF environment supports three ways of static analysis, *Reach* to remove all the unreachable states, *Slice* to remove the dead code sequences, and *Live* to remove dead variables and/or reset variables when useless in a control state. Figure 4 shows the *VMController* process before (in the right) and after (in the left) from combined reach, slice and static analysis for the VendingMachine case study from Section 2 (see Figure 2). The configuration file sets live the input *Tea*, resulting from slice the reduction of *FillCoffee()* and from reach the reduction of the states *PaidTen*, *MakeCoffee* resulting from static analysis the model with a state space reduction that guard their dependability.

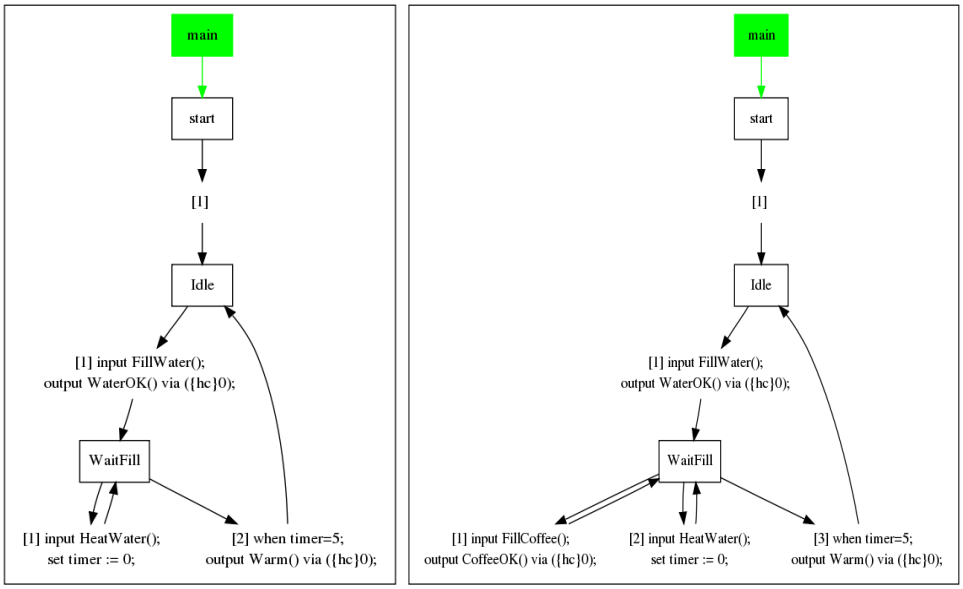

Figure 4: The IF model visualized in Graphviz on VMController process after, (in left) and before the combined Reach, Slice and Live static analysis

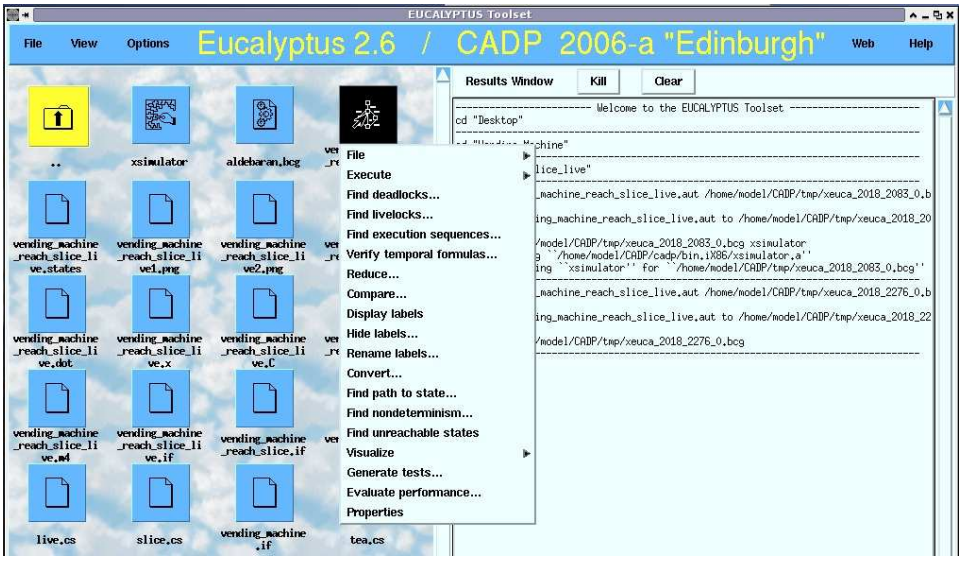

Figure 5: The CADP checking environment and report on Vending_Machine_riche_slice_live.aut

Figure 5 shows the checking for deadlock and livelock properties for the model resulted from steps of the static analysis phase.

The Vending_Machine_riche_slice_live.aut Aldebaran format file in the CADP provides information from specification properties after this the search for deadlock and livelock is reported. After the breadth-first search walk in the IF model state space had 606727 state and 2332246 transitions. With live and reach, and without slice static analysis due to CADP to give 204 state and 944 transitions. The reduced specification with slice and following live and reach static analysis due to CADP to give 63 states and 117 transitions only. The performance of deadlock search had increased with processing a successive reach, live and slice analysis, before.

4. Conclusion

We proposed to give experience on tooling and on processing of the verification tasks from several individual tools integrated process-driven in model-based design. The model-based verification tasks with VERIMAG IFx toolset it consist on engineering the verification environment to integrate by IF language some specific-purpose tools for modelling, analysis, and model checking, like ArgoUML, IF ASP, IF Static Analyser, SPIN, and CADP. We did an experiment to well-known vending machine system. We concluded that engineering and performing modular verification tasks for processing a model refinement give more coverage for non-expert verification engineer than the test case based automatic verification. Those can contribute significantly towards the granularity of design model and the performance of modelling with the reuse of test models.

The reason for this is that the built verification environment gives more interaction choice to handle variables and their values in refinement steps with slice and live analysis after the reach static analysis. While the automatic verification of general-purpose Telelogic Tau G2 Model Verifier hide the unreached states or the testing simulation cannot explores all of possible behaviours.

In the future works we plan to experiment importing UML models via an XMI repository for transformations from VERIMAG IFx toolbox, like uml2if and if2pml. The model transformations gives at each refinement of the design model to check it automatically and to set out the method for engineering verification tasks and artefacts for an MDA process at earlier phases of model-building.

References

[1] L. Kozma- L. Varga: A szoftvertechnológia elméleti kérdései, ELTE Eötvös Kiadó, 2003.

[2] Telelogic Tau User Guide, 2008.

[3] ArgoUML, `www.argouml.tigris.org`

[4] SPIN website: `http://spinroot.com/spin/whatispin.html`

[5] CADP-Construction and Analysis of Distributed Processes `www.inrialpes.fr/vasy/cadp/demos.html`

[6] M. Bozga, S. Graf, and L. Mounier. IF-2.0: A validation environment for component-based real-time systems. In Proceedings of Conference on Computer Aided Verification, LNCS 2404, 630-640, CAV'02, Copenhagen, 2002.

[7] M.Bozga, S. Graf, L. Mounier, Il. Ober, Iu. Ober, J. Sifakis: The IF toolset, Formal Methods for the Design of Real-Time Systems, LNCS 3185, pp.237-267, 2004. `http://www-verimag.imag.fr/~async/IF/IFx.html`

[8] Il.Ober, S. Graf, Iu. Ober: Validation of UML models via a mapping to communicating extended timed automata, OMEGA European Project (IST-33522) `http://www-omega.imag.fr`

[9] E. M. Clarke, E. A. Emerson, J. Sifakis Model checking: algorithmic verification and debugging, Communications of the ACM Volume 52 , Issue 11, Pages: 74-84, 2009.

Anna Medve, György Orbán
University of Pannonia, 8200 Veszprém, Egyetem u. 10., I/102/a, Hungary

László Kozma
Eötvös Lóránd University 1117 Budapest, Pázmány Péter sétány 1/C. 2.412/a., Hungary