ISSN 2658-5014 (Print)
2020. Т. 2, № 1 (3). С. 5–10

СИИТ
СИСТЕМНАЯ ИНЖЕНЕРИЯ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

ISSN 2686-7044 (Online)
http://siit.ugatu.su

УДК 004.65

# System Level Formal Verification as a Service

T. Mancini[1], F. Mari[2], A. Massini[3], I. Melatti [4], E. Tronci[5]

[1]tmancini@di.uniroma1.it, [2]mari@di.uniroma1.it, [3]massini@di.uniroma1.it, [4]melatti@di.uniroma1.it, [5]tronci@di.uniroma1.it

Computer Science Department Sapienza University Rome, Italy

*Поступила в редакцию 18 июня 2020 г.*

**Abstract**. We present SyLVaaS, a Web-based tool enabling Verification as a Service (VaaS) of Cyber-Physical Systems (CPS). SyLVaaS takes as input a high-level model defining the System Under Verification (SUV) operational environment and computes, using parallel algorithms deployed in a cluster infrastructure, a set of highly optimised simulation campaigns, which can be executed in an embarrassingly parallel fashion on a set of Simulink instances, each one defining a copy of the SUV model. As the actual simulation is carried out at the user premises (e.g., in a private cluster), SyLVaaS allows full Intellectual Property protection on the SUV model and the user verification flow. SyLVaaS randomises the verification order of operational scenarios, thus enabling anytime estimation of completion time and computation of Omission Probability, i.e., the probability that there is a yet-to-be-simulated operational scenario violating the property under verification. This information supports graceful degradation in the verification activity.

**Keywords:** modelled, system, methodology, simulation campaigns.

## INTRODUCTION

In this paper, we present SyLVaaS, a Web-based tool enabling Verification as a Service (VaaS) of Cyber-Physical Systems (CPS). SyLVaaS takes as input a high-level model defining the System Under Verification (SUV) operational environment and computes, using parallel algorithms deployed in a cluster infrastructure, a set of highly optimised *simulation campaigns*, which can be executed in an embarrassingly parallel fashion on a set of Simulink instances, each one defining a copy of the SUV model. As the actual simulation is carried out at the user premises (e.g., in a private cluster), SyLVaaS allows full Intellectual Property protection on the SUV model and the user verification flow. SyLVaaS randomises the verification order of operational scenarios, thus enablinganytime estimation of completion time and computation of Omission Probability, i.e., the probability that there is a yet-to-be-simulated operational scenario violating the property under verification. This information supports graceful degradation in the verification activity. Cyber-Physical Systems (CPSs) consist of hardware and software components and can be modelled as hybrid systems (see, e.g., [1] and citations thereof). System Level Verification of CPSs has the goal of verifying that the whole (i.e., software + hardware) system meets the given specifications. Model checkers for hybrid systems cannot handle System Level Formal Verification (SLFV) of actual CPSs. Thus, Hardware In the Loop Simulation (HILS) is currently the main workhorse for system level verification and is supported by Model Based Design tools like, for example, Simulink (*http://www.mathworks.com*) and VisSim (*http://www.vissim.com*). In HILS, the actual software reads/sends values from/to mathematical models (simulation) of the physical systems (e.g., engines, analog circuits, etc.) it will be interacting with.

## MOTIVATIONS

System Level Formal Verification (SLFV) is an exhaustive HILS, where all relevant simulation scenarios are considered. In [24] a methodology has been presented which allows exhaustive HILS. Such methodology works as follows. The System Under Verification (SUV) is a Hybrid System (see, e.g., [1] and citations thereof) whose inputs belong to a finite set of uncontrollable events (disturbances), which model failures in sensors or actuators, variations in the system parameters, etc. The SUV is a deterministic system (the typical case for control systems). Nondeterministic behaviours (such as faults) are modelled with disturbances. Also, sequences of inputs to the SUV are of bounded length, thus the problem addressed is bounded SLFV. Accordingly, in [2–4] a simulation scenario is a finite sequence of disturbances. A system is expected to withstand all disturbance sequences that may arise in its operational environment. Correctness of a system (defined in terms of safety properties) is thus defined with respect to such admissible disturbance sequences.

Given a high-level model (disturbance model) defining the admissible disturbance sequences (disturbance traces), the approach in [2–4] works as follows: *(i)* generates the entire set of disturbance traces, *(ii)* evenly splits such set into $k \in \mathbf{N}^+$ slices in order to allow parallel verification, *(iii)* computes (in parallel) an optimised simulation campaign from each slice, *(iv)* executes (in parallel) the generated simulation campaigns on a set of $k$ independent simulators (e.g., Simulink instances). There, a simulation campaign is a sequence of simulation instructions, which exploits the capabilities of modern simulators to save and restore previously stored simulation states (much as in explicit model checking). In particular, a simulation campaign consists of the following commands: save a simulation state, restore a saved simulation state, inject a disturbance, advance the simulation for a given time length. As soon as one of the simulators (running the simulation campaign corresponding to a slice) finds an error, the whole parallel simulation activity stops, and the disturbance trace which triggered the error is returned as a counterexample. Also, as the generated optimised simulation campaigns (one per slice) randomise the verification order of the traces in the input slice, at anytime during the parallel simulation activity it is possible to compute an upper bound to the Omission Probability (OP), i.e., the probability that an error exists, but no error has been found so far, and give a quite accurate estimation of the completion time. Algorithms for all the activities above have been presented in [2–4]. However, an off-the-shelf tool to effectively support companies working in the CPS business in their everyday SUV verification activities was not available. To provide such a tool is the purpose of [7] and of this paper.

## MAIN CONTRIBUTIONS

We present SyLVaaS (see Figure 1), a Web-based service computing the set of simulation campaigns to be used for a SLFV task. SyLVaaS introduces the new Verification as a Service (VaaS) paradigm, allowing verification engineers (SyLVaaS users) to compute the simulation campaigns needed to their SLFV activities keeping both the SUV model and the property to be verified secret, thus achieving full Intellectual Property (IP) protection. This is mandatory for a VaaS service to be effective and usable, as companies consider the design effort (hence their SUV models) and their verification flow as the core of their IP.

To enable IP protection, SyLVaaS takes as input only a disturbance model, in terms of a CMurphi [5] model describing the admissible operational scenarios the SUV must withstand. The actual verification activity is performed in parallel at the user premises (e.g., in a private

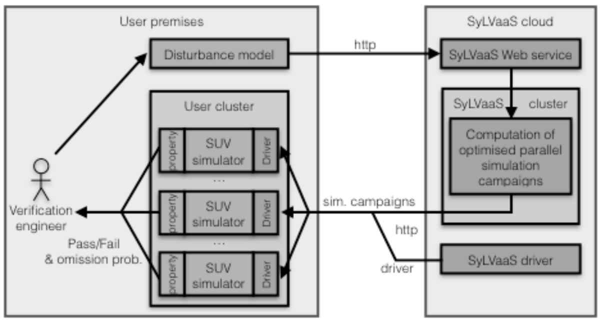

**Fig. 1.** SyLVaaS VaaS architecture

cluster) running an arbitrarily large set of Simulink simulators, using the optimised simulation campaigns computed by SyLVaaS and plugging-in a Simulink driver downloadable from the SyLVaaS Web site.

In case an error is found during verification, a counterexample is generated. Such a counterexample can then be used to correct the SUV and to produce a new SUV model. At this point a new SLFV activity can start. Note that, given that the set of admissible operational scenarios (hence: the disturbance model) has not changed, there is no need to interact with SyLVaaS again, as the previously computed simulation campaigns can be reused. This property also hides the verification flow of SyLVaaS users.

The operational scenario generation algorithm in [2] is a sequential algorithm, taking about half an hour on their case study. Although this time is negligible with respect to the whole HILS activity (which can take weeks of computation), it becomes a major bottleneck in a VaaS context as the one provided by SyLVaaS, as it is the most intensive part of the computation carried out on the SyLVaaS side (i.e., generation of optimised simulation campaigns for parallel HILS, see Figure 1, right). To achieve fast response time, we discuss how we equipped SyLVaaS with a parallel algorithm for the generation of operational scenarios from a disturbance model, whose distributed multi-core implementation has been explicitly designed as to operate efficiently in a cluster of possibly heterogeneous machines.

Our parallel operational scenario generation algorithm consists of an Orchestrator process which governs the exploration of the (state space of the finite state automaton defined by the) disturbance model provided by the user, splitting and delegating the work to a battery of available Slaves, whose work load is dynamically balanced. Slave processes are independent from each other and communicate only with the Orchestrator. This minimises coordination overhead.

We show experimental results of our parallel operational scenario generation algorithm on two industry-scale case studies (regarding the Fuel Control System (FCS) in the Simulink distribution) consisting of, respectively, 4,023,955 and 12,948,712 operational scenarios. Results show that our parallel algorithm for operational scenario generation scales well with the number of slaves. As the operational scenario generation is the most computationally intensive computation within the SyLVaaS workflow, and given that the other step performed by SyLVaaS (computation of optimised simulation campaigns) already exploits an embarrassingly parallel algorithm (from [3]), the entire SyLVaaS workflow now benefits from a cluster of machines at the SyLVaaS cloud infrastructure.

## SYSTEM LEVEL FORMAL VERIFICATION

In this section, we describe SyLVaaS in terms of input and output, and describe how to use the system output.

### INPUT

SyLVaaS requires two inputs:

1. An integer $k > 0$ describing the number of computational cores available on the user side for parallel execution of simulation campaigns (hence, for parallel verification);
2. A disturbance model defining the operational environment, i.e., the set of disturbance traces the System Under Verification (SUV) should withstand, along with a bounded horizon $h$.

As it is typically infeasible for a verification engineer to define a SUV operational environment by explicitly listing all its disturbance traces, SyLVaaS, along the lines of [2], takes as input a disturbance model defining a Disturbance Generator (DG) written in the high-level language accepted by the CMurphi [5] model checker.

## OUTPUT

From the value of *k* and the input disturbance model, SyLVaaS produces *k* simulation campaigns, which can be executed in parallel at the user premises over *k* independent simulators, in an embarrassingly parallel fashion. Each simulation campaign verifies, in a highly optimised way, a disjoint and equally sized portion of the disturbance traces entailed by the input disturbance model. Conversely, all disturbance traces entailed by the disturbance model are covered by exactly one simulation campaign. This guarantees that the System Level Formal Verification (SLFV) process is both exhaustive (with respect to the set of disturbance traces entailed by the disturbance model) and non-redundant.

Furthermore, the verification order of the disturbance traces covered by each simulation campaign is randomised. This, according to [3] enables the computation of an upper bound to the Omission Probability (OP) at anytime during the parallel simulation. The *k* simulation campaigns are returned to the user via the Web interface, together with an abstract Simulink driver. Such a driver is a MATLAB script, which reads and executes a SyLVaaS-generated simulation campaign, by sending simulation commands to Simulink. It is *abstract*, as it must be plugged into the SUV Simulink model and configured at the user premises (see Fig. 1).

## WEB INTERFACE

The Web interface of SyLVaaS is hosted at *http://mclab. di.uniroma1.it/sylvaas*. It consists of four main pages: *(i)* a standard login page, *(ii)* a user console page (accessible after login) showing all current, pending, running and completed user jobs, *(iii)* a page to create a new job (providing the required input), *(iv)* a tools page, where the generic driver can be downloaded. Users can download the simulation campaigns for each completed job from their console page.

## HOW TO USE SYLVAAS OUTPUT

Given the output downloaded by SyLVaaS, the verification engineer, in order to actually verify the SUV via exhaustive Hardware In the Loop Simulation (HILS), customises and plugs the abstract Simulink driver into the SUV Simulink model. This task is very easy and consists in properly filling the template files received by SyLVaaS as part of the abstract driver. Such files define: the SUV model, the SUV property to be verified (as a monitor module), the interface between the driver and the SUV, and the mapping between each disturbance (in the CMurphi disturbance model) and its counterpart in the SUV model. At this point, the *k* downloaded simulation campaigns can be executed in parallel on *k* independent simulators. Given the randomisation of the verification order of the disturbance traces within each simulation campaign, at anytime during the simulation process, when values $done_1$, $done_2$, .., $done_k$ (with $done_i \in [0, 1]$ for all *i*) of the disturbance traces covered by each simulation campaign have been verified successfully (i.e., no error has been raised so far), the Omission Probability (OP), i.e., the probability that a future simulation command raises an error, is upper bounded as shown in [3].

## EXPERIMENTS

In this section we experimentally evaluate SyLVaaS. In particular we provide the evaluation of our parallel disturbance generation algorithm and of the cloud deployment of the overall Verification as a Service (VaaS) infrastructure.

## SYLVAAS EXPERIMENTAL DEPLOYMENT

We deployed SyLVaaS on overall 17computational cores allocated to 5 different machines. One core (on a machine equipped with 2 Intel Xeon 2.83GHz CPUs and 8GB RAM) is dedicated to Orchestrator processes, while 16 cores evenly distributed in 4 identical machines (each one equipped with 2 Intel Xeon

2.27GHz CPUs and 24GB RAM) are dedicated to Slave processes. The SyLVaaS web interface application resides on a yet another host (a tiny virtual machine), external to the cluster and directly connected to the Internet.

## CASE STUDY

We use the same case study of [6, 2- 4], i.e., the Fuel Control System (FCS) model included in the Simulink distribution. The FCS has three sensors subject to faults (disturbances). We used two disturbance models for the FCS, $D_1$ and $D_2$. Model $D_1$ (described in more detail in [2]) has a horizon of $h$ = 100 and defines 4,023,955 disturbance traces. Model $D_2$ is defined extending $D_1$ with more complex operational scenarios and defines 12,948,712 disturbance traces over a horizon of $h$ = 200. A detailed description of $D_1$ and $D_2$ (not relevant for the evaluation of our experiments below) can be downloaded from the SyLVaaS Web site.

## EXPERIMENTAL RESULTS

1. *Parallel Disturbance Trace Generation*: Table 1 shows the time needed by SyLVaaS to generate the disturbance traces entailed by $D_1$ and $D_2$, when using a varying number $S$ of parallel slaves. The level (depth) $L$ to which the Orchestrator bounds its search and triggers a Slave has been fixed at $h/2$ after preliminary experiments. The number of computation bunches executed by the algorithm is 477,727 for disturbance model $D_1$ and 1,681,594 for $D_2$.

**Table 1.** Parallel generation of disturbance traces

| #slaves (S) | disturbance model $D_1$ | | | disturbance model $D_2$ | | |
|---|---|---|---|---|---|---|
| | time (h:m:s) | speedup | effic. | time (h:m:s) | speedup | effic. |
| 1 | 0:32:32 | 1.00 | 100.00% | 4:45:47 | 1.00 | 100.00% |
| 8 | 0:5:32 | 5.88 | 73.50% | 0:43:2 | 6.64 | 83.00% |
| 16 | 0:3:11 | 10.22 | 63.88% | 0:26:16 | 10.88 | 68.00% |

For each value of $S$, Table 1 reports the overall time for generating disturbance traces for both disturbance models (columns "*time*"), as well as *speedup* and *effic.(iency)* with respect to the execution time of the sequential algorithm (the first row in Table 1 referring to $S = 1$). As usual in the evaluation of parallel algorithms, for each value of $S$, the speedup is defined as $t_1/t_S$, where $t_1$ and $t_S$ are, respectively, the execution times of our disturbance trace generation algorithm when using 1 and $S$ parallel slaves. For each value of $S$, the efficiency is computed as the ratio between the speedup and $S$.

2. *Disturbance Trace Slicing*: Table 2 shows the time needed by SyLVaaS to compute $k$ slices from the disturbance traces generated using $S = 16$ slaves from disturbance models $D_1$ and $D_2$, for various values of $k$, which denotes the number of computational cores available at the user side for parallel simulation. To ease comparison of our results with those in [3], we used the same values of $k$ as those used in that paper.

**Table 2.** Results on slicing of disturbance traces

| #slices ($k$) | $D_1$ (h:m:s) | $D_2$ (h:m:s) |
|---|---|---|
| 128 | 0:4:1 | 0:8:7 |
| 256 | 0:4:32 | 0:11:25 |
| 512 | 0:4:52 | 0:13:17 |

3. *SyLVaaS Complete Workflow*: Table 3 reports the time needed to compute (in parallel) the $k$ simulation campaigns (column "*sim. camp. comp. time*") and the overall SyLVaaS response time (summing up trace generation, splitting, and simulation campaign optimisation times, column "*overall time*"), for each disturbance model and each value for $k$. Results in Table 3 have been obtained using $S = 16$ slaves during trace generation and 16 cores to compute the $k$ simulation campaigns (thus, on average, each core computed $k = 16$ campaigns).

**Table 3.** Results on the entire SyLVaaS workflow

| | disturbance model $D_1$ | | disturbance model $D_2$ | |
|---|---|---|---|---|
| #slices (k) | sim. camp. comp. time (h:m:s) | overall time (h:m:s) | sim. camp. comp. time (h:m:s) | overall time (h:m:s) |
| 128 | 0:1:44 | 0:8:56 | 0:4:1 | 0:38:24 |
| 256 | 0:0:42 | 0:8:25 | 0:2:27 | 0:40:8 |
| 512 | 0:0:13 | 0:8:16 | 0:0:24 | 0:39:57 |

4. *Download of Simulation Campaigns*: SyLVaaS stores simulation campaigns computed as above in .zip archives, which are then downloaded by the user. In our experiments, the size of such files is in the order of *a few hundreds of MB*. Hence, their download into the user cluster can be done seamlessly over a standard broadband Internet connection.

## CONCLUSION

We have presented SyLVaaS, a Web-based software-as-a-service tool for HILS-based System Level Formal Verification (SLFV). Such a tool allows verification engineers to obtain from a Web service the most important part of their HILS campaigns, i.e., a set of simulation campaigns to exercise the System Under Verification (SUV) on all the relevant operational scenarios (disturbance traces).

As the simulation campaigns are executed at the user premises, SyLVaaS provides full Intellectual Property (IP) protection for both the SUV model, the property to be verified, and the user verification flow. The simulation may be carried out in parallel in a user cluster whose machines have Simulink installed.

To achieve a short response time, the whole SyLVaaS approach benefits of a cluster of machines at the SyLVaaS cloud infrastructure. To the best of our knowledge, SyLVaaS is the first Web-based software-as-a-service tool for HILS-based SLFV.

## ACKNOWLEDGMENTS

This research has received funding from the EU 7th Framework Programme (FP7/2007-2013) under grant agreements n. 317761 (SmartHG) and n. 600773 (PAEON).

## REFERENCES

1. **Alur R.** "Formal verification of hybrid systems". In: Proc. EMSOFT (2011), Taipei, Taiwan, 2011, pp. 273-278.

2. **Mancini T., Mari F., Massini A., Melatti I., Merli F., Tronci E.** "System level formal verification via model checking driven simulation". In: Proc. CAV (2013), Saint Petersburg, Russia, 2013, pp. 296-312.

3. **Mancini T., Mari F., Massini A., Melatti I., Tronci E.** "Anytime system level verification via random exhaustive hardware in the loop simulation" . In: Proc. DSD (2014), Verona, Italy, 2014, pp. 236-245.

4. **Mancini T., Mari F., Massini A., Melatti I., Tronci E.**. "System level formal verification via distributed multi-core hardware in the loop simulation". In: Proc. PDP (2014), Turin, Italy, 2014, pp. 734-742.

5. **Penna G. Della, Intrigila B., Melatti I., Tronci E., Zilli. M. Venturini** "Exploiting transition locality in automatic verification of finite state concurrent systems". STTT, 2004; 6: 320–341.

6. **Zuliani P., Platzer A., Clarke E..** "Bayesian statistical model checking with application to Simulink/Stateflow verification". In: Proc. HSCC (2010), Stockholm, Sweden, 2010, pp. 243-252.

7. **Mancini T., Mari F., Massini A., Melatti I., Tronci E.** "SyLVaaS: System Level Formal Verification as a Service". In: Proc. PDP (2015), Turku, Finland, 2015.

## METADATA

**Title:** System Level Formal Verification as a Service

**Authors:** T. Mancini[1],F. Mari[2],A. Massini[3], I. Melatti[4], E. Tronci[5]
**Affiliation:** Computer Science DepartmentSapienza University Rome, Italy
**Email:** [1]tmancini@di.uniroma1.it, [2]mari@di.uniroma1.it, [3]massini@di.uniroma1.it; [4] melatti@di.uniroma1.it, [5]tronci@di.uniroma1.it

**Language:** English
**Source:** SIIT, vol.2, no. 1 (3), pp. 5-10, 2020. ISSN 2658-5014 (Print), 2686-7044 (Online).

**Abstract:** In this paper, we present SyLVaaS, a Web-based tool enabling Verification as a Service (VaaS) of Cyber-Physical Systems (CPS). SyLVaaS takes as input a high-level model defining the System Under Verification (SUV) operational environment and computes, using parallel algorithms deployed in a cluster infrastructure, a set of highly optimised simulation campaigns, which can be executed in an embarrassingly parallel fashion on a set of Simulink instances, each one defining a copy of the SUV model.

**Key words:** modelled, system, methodology, simulation campaigns.

**About authors:**

**Mancini Toni** Prof. Dr., Computer Science Department Sapienza University Rome, Italy

**Mari Federico,** Assistant Prof. Science Department Sapienza University Rome, Italy

**Massini Annalisa,** PhD Computer Science Department Sapienza University Rome, Italy

**MelattI Igor,** PhD Computer Science Department Sapienza University Rome, Italy

**TRONCI Enrico,** Prof. Dr. Computer Science Department Sapienza University Rome, Italy