Algorithmic Selection of Shared Test Resources

D4.8 Deliverable

PUBLIC

2019-09-30

Edited by:

Andre Bergmann, Julian Becker, Hojat Khosrowjerdi, Karl Meinke, Tommi Oinonen, Stefan Puch

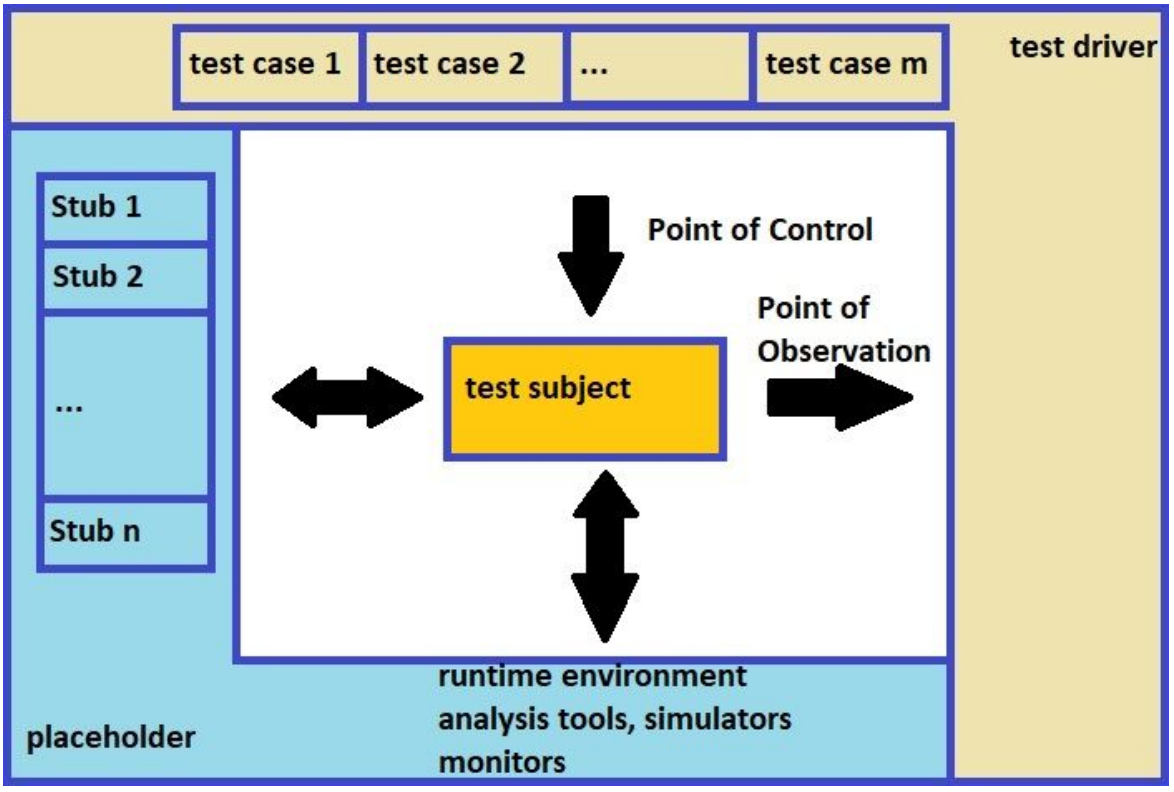
Introduction to algorithmic selection of testbeds

In the advent of newer technologies and more complex applications is inducing an accelerated evolution of the underlying infrastructure and networks. This has created new problems in testing when physical testbeds cannot be used due to the growing size of these structures. Therefore, testbeds in hardware are not practical in some cases. But what in fact is a testbed? The IEEE[1] refers to testbeds as

> *"An environment containing the hardware, instrumentation, simulators, software tools, and other support elements needed to conduct a test."*

However, this definition is rather limited, it is necessary to go into more details. A testbed is a collection of all programs that are needed to execute and to evaluate test cases and to log test protocols. For this reason placeholders and test drivers are needed. The placeholders or stubs simulate the input and output behaviour of the external or not yet implemented parts of the program. Dummies or mocks can be used in exchange of the stubs. A dummy delivers a nearly complete replacement for the actual implementation. A mock delivers additional functionality for test purposes.

Testbeds are programs that enable to execute a test subject, to supply a test subject with test data, and to accept output data from the test subject. Test driver and placeholder together make up the testbed which in combination with the test subject forms the executable program.



Algorithmic Selection

[1] https://ieeexplore.ieee.org/document/159342 (24.7.2019)

Expleo: Vision of Algorithmic Selection of Testbeds

While technologies have become more complex and their underlying infrastructure has grown very fast, Expleo had to find its way to accomplish the more demanding requirements on test systems. Therefore, an inhouse tool named MESSINA was developed for test automation and virtual validation of embedded systems.
MESSINA is a powerful real-time test system for the validation of ECU functions in early phases. Its scope of application includes the test of models, software components, and control units on the HiL test bench. For integrated testing of ECU functions with different maturity levels, MESSINA allows to test them in any combination. This unique consistency of MiL, SiL, and HiL tests as well as the flexible configuration of integration test enables an efficient and cost-optimized test process. In addition to a real time runtime system and an intuitive user interface, MESSINA offers standard interfaces such as FMI or EtherCAT to integrate models, simulations, or hardware components for the test.
There are several test systems with MESSINA:

- Camera-HiL: Testing of driver assistance systems based on cameras such as road sign detection and lane departure warnings.
- MESSINA RS: Test automation of infotainment, interior, and general UIs with robots.
- ANCONA: Modular-HiL: Validation of software based systems and control modules in real time.
- ANCONA: EtherCAT-based HiL test system with a wide range of components for signal conditioning as well as modules for the connection with various sensors and actuators.

Due to the steady growing size of test cases and testbeds, an automation of testbeds is a possible way to go at Expleo. The underlying idea is the following:
It should be possible to generate a testbed for the test subject which means that the test drivers and the stubs for the external functions are provided and the test cases are logged. For writing the tests platform independent we are considering to use a “Target Deployment Port” (TDP) which simulates the compiler, linker, debugger, and the target test environment. The tests themselves are then independent of the TDP. Therefore, they do not have to be changed when the test environment is changing.
The isolation of the module to be tested will be done automatically. During the creation of a new project, all external interfaces of the module to test will be searched and with the help of this information stubs and test drivers will be generated. The external interfaces must therefore be presented in included header files otherwise there could be failures.
During the creation of the testbed, a test file will be generated where the necessary stubs are declared and basic tests are included. The basic test scripts are elementary tests that consist of a call of the functions to test. An expected result will not be declared such that tests can only fail when stubs are called. For this reason tests are only good as a starting base for a test creation. Fortunately, Expleo delivers other tools (TESTONA and MODICA) that can be used to create good test cases.
For every module to test, a test node will be created during the testbed generation. This node contains all child nodes tests that want to proof the functionality of this module. More

program code has then to be inserted in dependence of the runtime analysing features to calculate the required information.

AKKA’s Vision

Testing in the automotive industry has a long tradition. Every feature is tested as intense as possible on hardware test beds which simulate the real car as detailed as needed. Setting up and automating hardware test beds is very time consuming and costly. Thus each testbed has a different focus and use and their number is limited.

Nowadays, the development of automotive features becomes more and more software based. The software becomes more complex and solid testing is as crucial as in the past. But its complexity increases and it is hardly possible to keep up the intensity of tests.

CI / CD concepts of classical software development are applied, to allow the software development to be fast and efficient. Fast build times, instant feedback for coverage and quality metrics, integration into the final hardware and of course complete test results are desired. This has lead to huge systems, which can fulfill most of the software needs. When it comes to executing test on hardware test beds, there is no real solution. Car manufactures need test results on systems, which are as close as possible to the final product. Advanced and complex software needs a lot of tests and executing those on hardware test beds takes a lot of time.

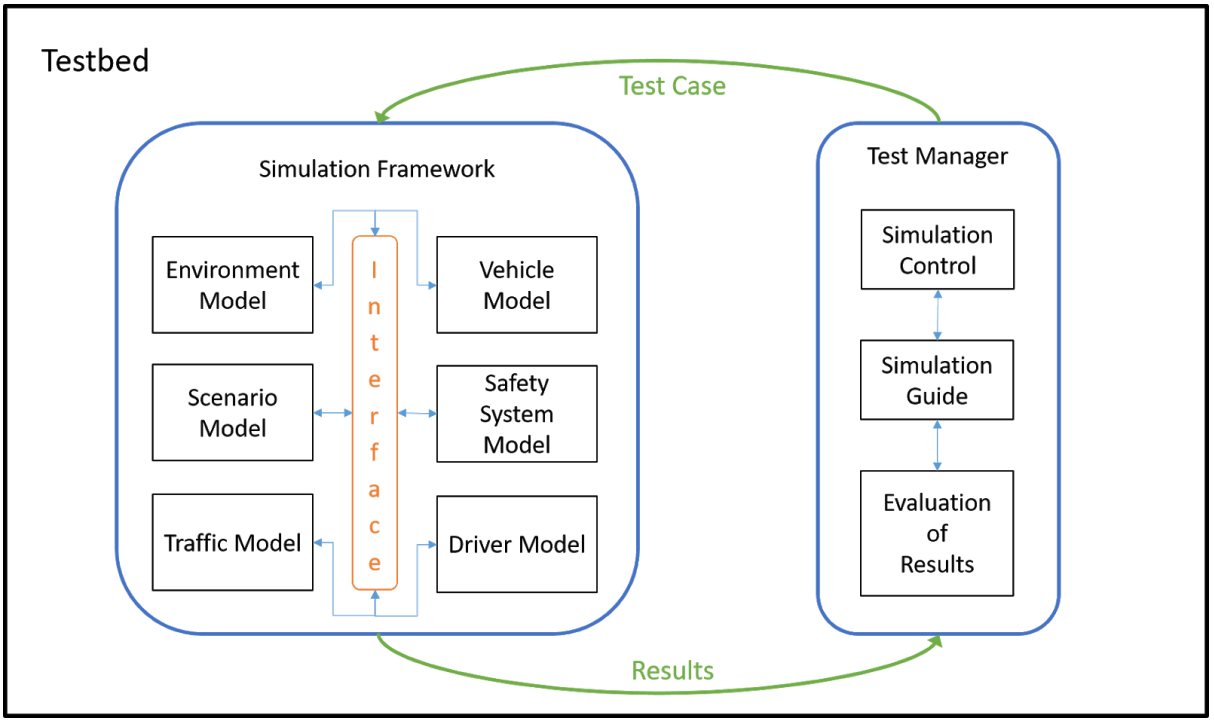
While it is impossible to easily increase the number of test beds until all needs are fulfilled, other solutions are searched for.

In AKKA's vision, there are methods to decide, which tests gets executed on hardware test beds and which do not. The method distributes the selected tests on the available test beds and ensures, that the response time is quick. Most importantly it needs to ensure, that the selected tests are enough and prove the software to be valid.

More research is needed to develop algorithms to achieve this goal.

OFFIS: The Concept of Guided Simulation within Testbeds

After the given introduction into testbeds and the presentation of an in-house test system for test automation this chapter presents a concept to use the simulation time available for automated testing as efficiently as possible. The idea behind the concept is to generate suitable test cases depending on the respective test objective and to spend as little simulation time as possible on meaningless test cases. Depending on the context, different strategies and algorithms are suitable for this purpose.



Schematic structure of a testbed consisting a Simulation Framework and Test Manager

In the figure above a schematic overview of a testbed is shown. On the left side it contains an exemplary Simulation Framework with different types of models. Depending on the context, each model or all models together can represent the system under test. This chapter focuses on the Test Manager depicted on the right side. The Test Manager creates test cases for the framework and gets the simulation results back for evaluation or further processing. While the simulation control is responsible for the technical sequence such as starting, stopping, pausing, etc. of the testbed, the simulation guide takes over the actual task of intelligent simulation control. It can use different algorithms to generate new test cases and thus optimize the necessary simulation time with regard to a desired goal. Apart from classic naive Monte Carlo simulation, test cases with previously defined faults can be injected into the simulation framework (fault injection), or methods for limiting or covering a large state space can be implemented.

A special case of the simulation guidance and therefore extremely challenging is the generation of test cases where rare events have to be taken into account. The analysis of

rare events is an essential task, whether in the automotive sector, in aviation or in the maritime domain. Rare events often mark peripheral areas, but in the context of complex systems they are safety relevant and may have fatal consequences if not handled properly. Depending on the application context or available simulators, the simulation guide can use algorithms based on importance sampling or importance splitting to visualize rare events (i.e. probabilities of 10^{-6} and less) within the given simulation framework. As examples for importance sampling TUTS and cross entropy method can be named, for importance splitting the (adaptive) restart method.

Siili: Algorithmic Sharing of Testing Resources

In this chapter we give an example for a simple component of algorithmic test bed selection in the context of functional system testing. As explained in the introduction into test beds, an important function of a test bed is to provide the necessary dummy, mock or otherwise suitable interfaces that allow the test subject to perform its required functionality. Here we will discuss this in the context of databases and test data.

Tests in the first phases of an automated testing pipeline (usually unit tests) are executed in an environment where all other interacting components, such as databases, are mocked to reduce complexity of the testing system. This helps by both isolating behavior of the system under test and improving the performance of the test cases. The share of these mocked subcomponents in the system should decrease when moving to later testing phases.

For the final testing phases, all the subsystems should be as close to the production environment as possible. This will provide the maximum confidence that the system under test will behave in the same way during the tests as it would in the production environment. The same rule applies for the test data which means that the final system tests should preferably interact with a copy of the production database.

Producing and maintaining production data or production like data for testing can be impossible or prohibitively expensive. This can mean that test beds providing sufficiently production like data can be a scarce resource for the testers and needs to be shared efficiently. As an example of such a situation there might be a single database of production equivalent user data that contains only a couple user instances suitable for specific test cases. These users and their associated data can not be used simultaneously by multiple tests.

To answer this problem we at Siili developed a very simple tool: Test data server. The tool can be used to either rotate the available test data so that each testing execution can dynamically use available test data or even function as a lock that blocks new executions when no test data is available.

Comiq - Scheduling test execution with limited shared test resources

It is our experience in multiple industries developing systems consisting of hardware and software components is that limited testing resources becomes bottleneck in test automation. This is typically not evident in the beginning when taking test automation in to use but comes apparent when the number of automated test cases and projects using test automation increases. At some point the trivial approach of each project having dedicated testbeds for themselves becomes unfeasible because of the costs associated with testbeds.

In our experience pretty much all the companies in embedded system development have adopted agile methodology and are starting more and more to incorporate DevOps methods as well. These approaches require iterative software development model so that the expected benefits can be realized. On the other hand, for effective iterative software development as fast as possible feedback cycle from code change to testing results is needed. The limiting factor for scaling test automation is the availability of the testbeds and since their scaling is limited other ways to speed up the feedback cycle are needed. With slower release cycles in more traditional software development models this was not as a big problem since test laboratories could be reserved for projects in testing phase.

In our experience the testbeds can be used by many development projects working on different products since testbeds usually have shared basic hardware between product and newer products are quite often backwards compatible with older ones. Currently reconfiguring testbeds is typically done manually or at least not automatically which causes sub optimal utilization of testbed hardware. Our scheduling solution enables dynamically reconfiguring testbeds for various test loads this enables higher testbed hardware utilization. To support iterative software development model our solution supports prioritizing

test case execution so that fast feedback to product development or test automation development can be realized.

Typically test cases are now executed in the industry with test suite as a unit of work. Most often now this unit of work is implemented as a CI job executing one test suite. Since larger test suite execution times can easily be in multiple hours a smaller unit of work is needed. For this reason, we schedule test case execution at single test case level instead of test suite. The test suite results are then aggregated in the end with map reduce type of algorithm but of course can be monitored during the test suite execution.

In the current state of the practice approach the CI job determines the configuration for test suite implicitly by the environment where the executor slave is running. Since this implicit information is not easy to access it is next to impossible track on company levels which teams could share testing resources. This combined with the fact that environment configuration is typically done manually even if the possibility of sharing test resources is known causes very inefficient utilization of testbed hardware. The financial investment in testbeds is very significant composed of hardware expenses, physical space, work taken to build testbeds and maintaining them.

In our solution configuration is explicit when scheduling test suite for execution. This allows utilizing potentially all suitable test beds for given test load. The dynamical reconfiguration of testbed in embedded development typically takes quite a while from some minutes to half an hour or more. On purely digital services this is typically non-issue since time spent on reconfiguration is negligible. For example, time spent reconfiguring service to use different databases running in Docker containers can be done in seconds or alternatively just set-up all configurations in parallel. The reconfiguration time must be measured, tracked and taken into consideration when planning test execution.

For scheduling test case execution, the historical test execution times are needed. In some test automation approaches failing test cases are problematic because they have significantly different execution times

compared to passing test cases. Typically, this is faulty test case implementation and might be possible to fix on test case implementation level relatively easily. In cases where the sheer amount of test cases and/or the way in which test cases are implemented prevents this approach. We are experimenting with utilizing historical test case execution times to terminate execution and fail test case if it is taking too long.

We have identified in practice at least three different test suite priority levels for execution present in widely in industry based on how timely the results are needed; as fast as possible, a couple of hours at worst and before the next morning. During the daily development and test automation maintenance activities smaller set of or even single test case executing with fast as possible feedback cycle is needed. Couple of hours is typically acceptable so that the feedback from a larger regression test suite is available on the same day changes have been made. The next morning is often good enough for full regression test suites. Since we are scheduling test case execution per test case and not test suite level the priority of test case execution can be taken into consideration when scheduling which test case should be executed next.

So, what we end up with our current solution is an optimization problem with test priority, test case execution duration, required configuration for execution, available testbeds and knowledge which configurations they support and reconfiguration time of testbed as parameters. Each of the above parameters are time-dependent, since they change with each new test suite scheduled throughout the day or with changes in available testbeds. Since optimal result can rarely be reached the liveliness and attempt to keep deadlines need to be also considered. Liveliness in a sense that even lower priority test suite with rarely used configuration gets executed eventually.

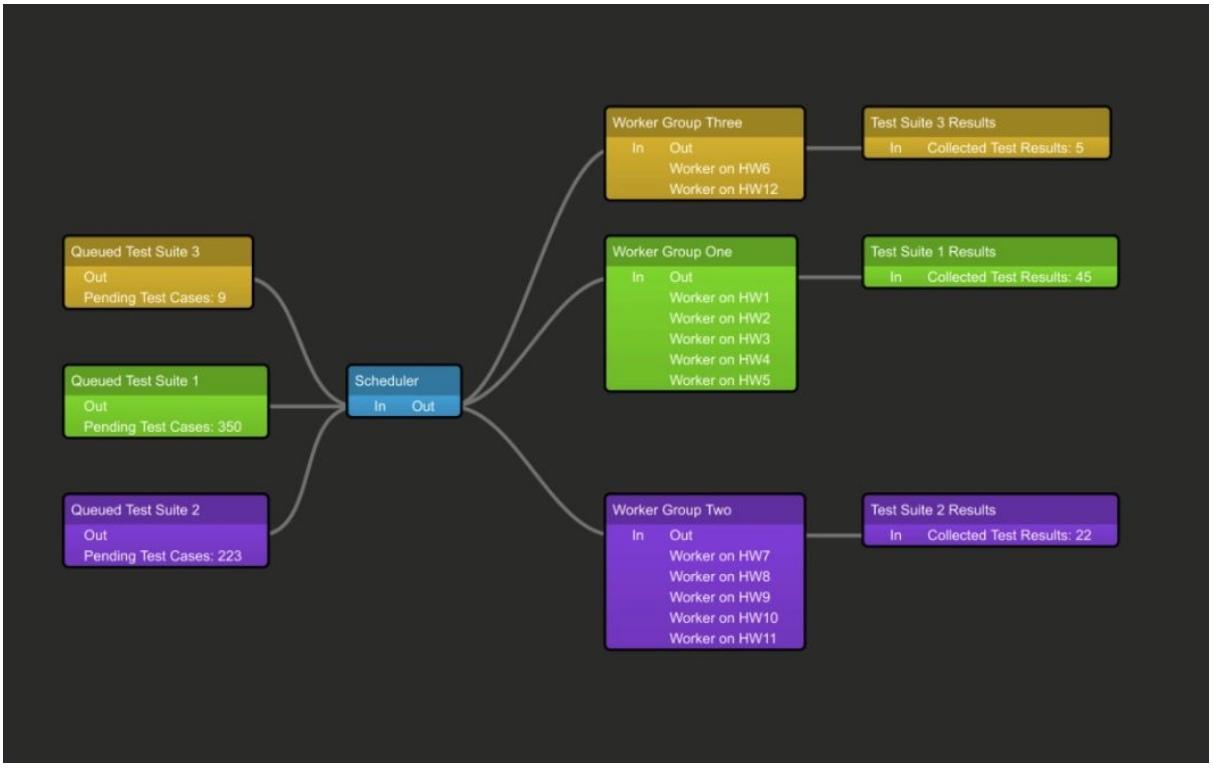


Figure 1: Visualization of scheduler monitoring data

Next we plan to investigate using historical test suite execution data (or/and version control commit data) to estimate test loads from yet to be scheduled test suites. Taking this into consideration should result better throughput of test execution. For example, if there are development teams in multiple time zones; The yet to be scheduled test suites from different time zone are typically targeting mostly different configuration than the already scheduled. Prioritizing test scheduling now on configuration subset not utilized later saves time consuming reconfiguration iterations later.

Other future area of investigation is leveraging code analysis or historical data of test executions to skip test cases that have a high probability of failing based on the information learned during the test suite execution.

KTH: A Testbed for CO-CPS System-of-Systems Testing

1. Testing CPS software, problems and motivation

Co-operative cyber-physical systems (CO-CPS) require more advanced technologies for testing. They have complex environment which may include other CPSs that interact with each other. Hardware-in-the-loop (HIL) testing is a common practice among different industry domains to achieve high accuracy test results with low damage risks. HILs are powerful hardware platforms that can emulate the real physical environment of a testing object. They are advanced testbeds that can combine model simulations with hardware executions to build a complete test scenario. However, HILs have several limitations.
- They are very expensive and therefore, scarce resources that has to be managed with care.
- Fully automatic test case generation, execution and oracle construction is still a challenge in HILs.
- HILs are not meant to be scalable. This is particularly a bottleneck for testing co-operating CPSs.

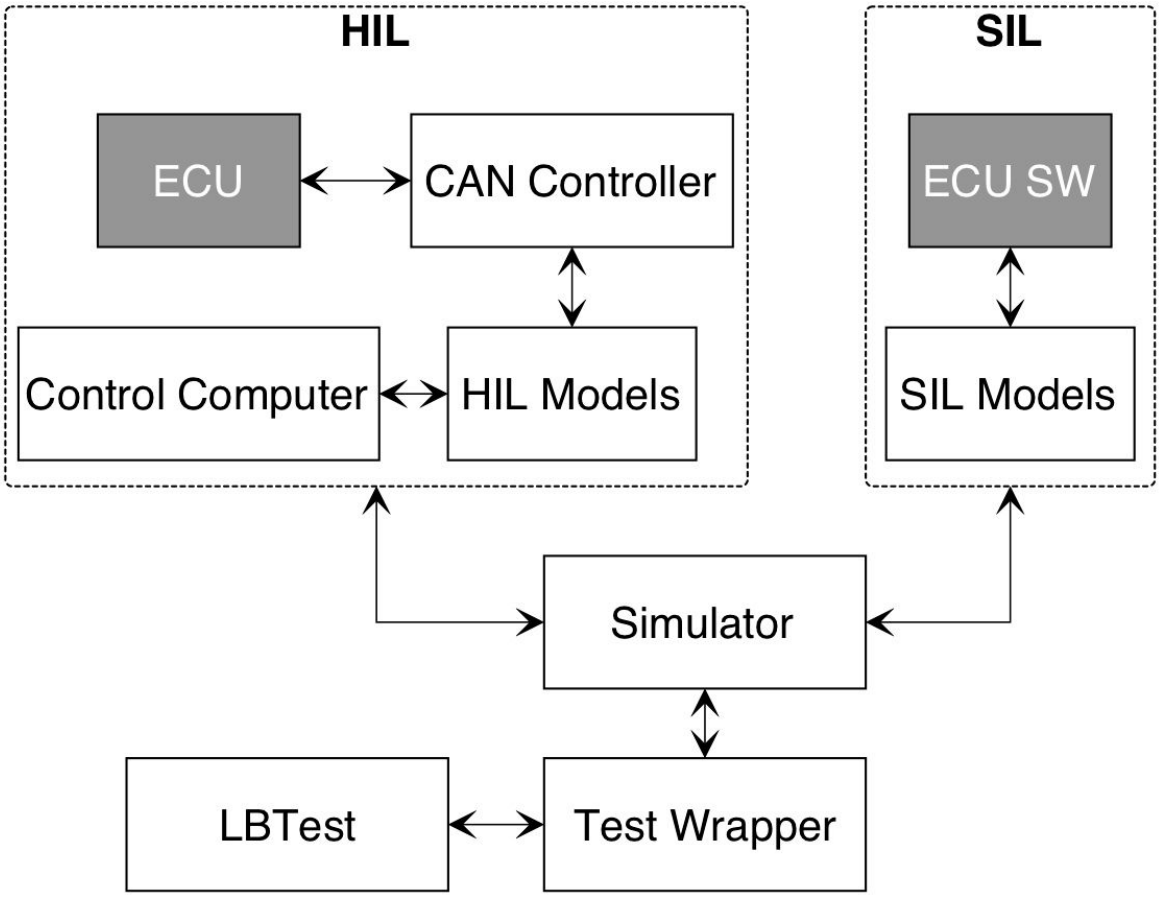
Software-in-the-loop (SIL) testing is an alternative solution in the early development phases. The actual SUT executes in a simulation environment that emulates all the external interfaces to the SUT. Even though a SIL may not be not as accurate as a HIL, it is affordable, scalable and can achieve high test throughput. Scalability is an important feature of SILs towards CO-CPS testing. Consider testing a safety feature of an autonomous platoon of communicating vehicles, or a team of robots. This is way too complex for any available HILs. Therefore, we can ask what should an efficient test framework look like? How can one combine HIL and SIL testing to obtain accurate results with greater scalability to CO-CPS?

2. An Integrated SIL/HIL Testbed

To tackle these issues, the testing research team at KTH has developed a test framework to integrate SIL and HIL testing in one test session(see e.g. [Meinke 2017], [Bergenhem et al. 2018]). In the figure below an example test framework is shown for one or several ECUs excluding interface details. A HIL is mainly composed of one or more physical ECUs under test, a CAN controller, environmental models and external interfaces to the ECU, and a controller to manage test and simulation executions. A SIL usually has a simpler architecture composed of environmental models and external interfaces along with a test harness that allows to execute an ECU software.

The Simulator component is able to manage simulations and test executions in HILs and SILs simultaneously. It has communication links to send test date and read the results. It also synchronizes HIL and SIL environments.

The Test Wrapper is a user-defined software component that creates a mapping between symbolic data types (used in abstract test cases) and concrete data types (used in concrete test cases). It marshals test stimuli into the SUT and returns SUT observations to the testing tool. It has direct control over the Simulator and all other components when starting a test case and can collect the test results. It also performs test setup and tear-down activities necessary to execute individual test cases and a complete test session.



Our test automation tool is LBTest [Meinke, Sindhu 2013]. which is an automated test case generation (ATCG) tool that can fully automate test case generation, execution and verdict construction. It is a dynamic black-box requirements testing tool based on the ideas from machine learning and static analysis (especially model checking). LBTest uses machine learning to generate test cases and reverse engineer a behavioral model of the SUT. Requirements testing is used to validate the functionality of the SUT and generate pass/fail verdicts.

Using the combined HIL/SIL testing framework, we have succeeded to manage shared testing resources more efficiently. Within the Testomat project KTH is attempting to further apply and refine this framework for automotive ECU testing, including using recently developed techniques such as spatio-temporal modeling [Khosrowjerdi, Meinke 2018].

References

[Meinke, Sindhu 2013] Karl Meinke, Muddassar A. Sindhu, *LBTest: A Learning-Based Testing Tool for Reactive Systems*, Sixth IEEE International Conference on Software Testing, Verification and Validation, ICST 2013, Luxembourg, Luxembourg, pp.447--454, 2013.

[Meinke 2017] Karl Meinke, *Learning-Based Testing of Cyber-Physical Systems-of-Systems: A Platooning Study*, Computer Performance Engineering - 14th European Workshop, EPEW 2017, Berlin, Germany, Proceedings, pp.135--151, 2017,

[Bergenhem et al. 2018] Carl Bergenhem, Karl Meinke, Fabian Ström, *Quantitative Safety Analysis of a Coordinated Emergency Brake Protocol for Vehicle Platoons*, in: Leveraging Applications of Formal Methods, Verification and Validation. Distributed Systems - 8th International Symposium, ISoLA 2018, Limassol, Cyprus, Proceedings, Part III, pp.386--404, 2018.

[Khosrowjerdi, Meinke 2018] Hojat Khosrowjerdi, Karl Meinke, *Learning-based testing for autonomous systems using spatial and temporal requirements*, Proceedings of the 1st International Workshop on Machine Learning and Software Engineering in Symbiosis, MASES@ASE 2018, Montpellier, France, 2018, pp. 6--15, 2018.