TestWeaver 3.2

Higher test coverage with less work

TestWeaver is a tool for automated test and validation of embedded systems using MiL, SiL, HiL simulation.

For testing a system, TestWeaver does not require predefined test scripts. TestWeaver generates, runs and evaluates thousands of tests autonomously - using a unique technology for intelligent test generation and evaluation of reactive systems.

Features
- automated search for worst-case scenarios
- reactive test execution on MiL, SiL or HiL
- automatic evaluation and report generation
- test coverage reports: requirement coverage, source code coverage, operational state coverage

ISO26262-recommended methods
- requirement-based test
- analysis of boundary values
- fault injection tests
- robustness tests
- demonstrate absence of unintended functionality

Support of virtual ECU test with Silver
- flashing of ECU configuration and calibration data
- range monitoring for all ECU signals
- debugging for C/C++ with Microsoft Visual Studio

Problems that can be found with TestWeaver
- coding errors: division by zero, access violation, infinite loops, non-determinism, range violations
- parameter errors: bad configuration/calibration parameter
- algorithmic errors: bad state estimation, oscillations of discrete and continuous signals
- system-level problems: violation of safety conditions, component overheating, loss of comfort.

Coding errors can be found using quite simple models. Quality assessments typically require a closed-loop simulation with a calibrated plant model.

Supported modeling and development environments
- C/C++, for instance Microsoft Visual Studio
- MiL: MATLAB/Simulink, Dymola, SimulationX
- SiL: Silver - virtual ECU
- HiL: dSPACE, ETAS, National Instruments

Connections to traffic & vehicle dynamic simulators
- IPG: CarMaker
- VIRES: VTD
- TASS International: PreScan

Benefit
- fast development: early problem detection
- high test coverage: thousands of high-quality tests
- less work: more automation, less scripting
How to use TestWeaver

The starting point is an executable model of the system under test. This model consists typically of (i) a simulation model of the controlled physical components, e.g., a vehicle model, and (ii) the software controlling the system. In model-based development projects, such models are usually available anyway. You can use a wide range of tools to build the model, see front page.

TestWeaver needs no access to the model source. You can test compiled models without knowing their source code.

1.

Select key inputs and outputs of the system. Typical inputs are acceleration and brake pedal, steering angle, road properties, and variables to control fault injection, e.g., sensor faults. Typical outputs are state variables of the control software, and key variables of the vehicle model, e.g., current and target gears, engine torque, etc. Classify the output values, to enable TestWeaver to distinguish desired from unwanted behavior. Likewise, classify input values, to enable TestWeaver to distinguish nominal inputs from inputs used to activate a component fault, e.g., of a sensor. For real valued inputs and outputs, classification means to partition the real axis in intervals, and assign the above properties to each interval. This turns the infinite space spanned by all inputs and outputs into a grid with a finite number of discrete states.

2.

Connect simulation signals with TestWeaver. Connector libraries are provided, e.g., for C, Simulink, Dymola, Silver and Python.

3.

Run TestWeaver: TestWeaver will autonomously drive the system model using the inputs and observe system response through the outputs. TestWeaver's goal is to drive the system into undesired states and to maximize coverage of the system states in the grid defined above, i.e., to reach every reachable discrete state at least once.

Analyze the problems found by TestWeaver. The problems found and the coverage reached during an experiment is reported using tables and histograms. Every problem found can be replayed in the simulation or development environment for detailed debugging and analysis. Interesting tests can be collected in test databases, e.g., for regression tests, or can be exported into other tools, e.g., for HiL tests.

4.

Our services

- Consulting for test and validation of systems
- Support for developing simulation models
- TestWeaver integration in development projects