Turn Indicator Model Overview

Jan Peleska[1], Florian Lapschies[1], Helge Löding[2], Peer Smuda[3], Hermann Schmid[3], Elena Vorobev[1], and Cornelia Zahlten[2]

[1] Department of Mathematics and Computer Science
University of Bremen, Germany
{jp,elenav,florian}@informatik.uni-bremen.de
[2] Verified Systems International GmbH, Bremen, Germany
{hloeding,cmz}@verified.de
[3] Daimler AG, Stuttgart, Germany
{peer.smuda,hermann.s.schmid}@daimler.com

System Interface. In Fig. 1 the interface between system under test (SUT) and testing environment (TE) is shown. Due to the state-based nature of the hardware interfaces (discretes, periodic CAN or LIN bus messages repeatedly sending state information) the modeling formalism handles interfaces as shared variables.

The TE can stimulate the SUT via all interfaces affecting the turn indication functionality in the operational environment: $\mathsf{in\_CentralLockingRM} \in \{0, 1, 2\}$ denotes the remote control for opening and closing cars by means of the central locking system. Signal $\mathsf{in\_CrashEvent} \in \{0, 1\}$ activates a crash impact simulator, and $\mathsf{in\_EmSwitch} \in \{0, 1\}$ simulates the "not pressed"/"pressed" status of the emergency flash switch on the dashboard. Signal $\mathsf{in\_IgnSwitch} \in \{0, 6\}$ denotes the current status of the ignition switch, and $\mathsf{in\_TurnIndLvr} \in \{0, 2\}$ the status of the turn indicator lever. In special-purpose vehicles (SPV), such as taxis or police cars, additional redundant interfaces for activation of emergency flashing and turn indicators exist (e. g., $\mathsf{in\_EmSwitchSPV} \in \{0, 1\}$). Observe that these redundant interfaces may be in contradicting states, so that the control software has to perform a priority-dependent resolution of conflicts. Inputs to the SUT marked by OPTION specify different variants of vehicle style and equipments, each affecting the behavior of the turn indication functions. In contrast to the other input interfaces to the SUT, options remain stable during execution of a test procedure, since their change requires a reset of the automotive controllers, accompanied by a procedure for loading new option parameters. If the TE component does not contain any behavioral specifications, the test generator will create arbitrary timed sequences of input vectors suitable to reach the test goals, only observing the range specifications associated with each input signal. This may lead to unrealistic tests. Therefore the TE may be decomposed into concurrent components (typically called *simulations*) whose behavior describe the admissible (potentially non-deterministic) interaction of the SUT environment on some or all interfaces. The test generator interprets these simulations as additional constraints, so that only sequences of input vectors are created, whose restrictions to the input signals controlled by TE components comply with the transition relations of these simulations.

SUT outputs are captured in the SignalsOut interface (Fig. 1 shows only a subset of them). The indicator lights are powered by the SUT via interfaces pwmRatio_FL $\in \{0, 120\}, \ldots$ where, for example, FL stands for "forward left" and RL for "rear right". The TE measures the percentage of the observed power output generated by the lamp controllers, in comparison with the expected value, 100% denoting exact identity. System integration testing is performed in grey box style: apart from the SUT outputs observable by end users, the TE also monitors bus messages produced by the cooperating controllers performing the turn indication service. Message tim_EFS $\in \{0, 1\}$, for example, denotes a single bit in the CAN message sent from a central controller to the peripheral controllers in order to indicate whether the emergency flash switch indicator on the dashboard should be activated, and tim_FL $\in \{0, 1\}$ is the on/off command to the controller managing the forward-left indicator light.

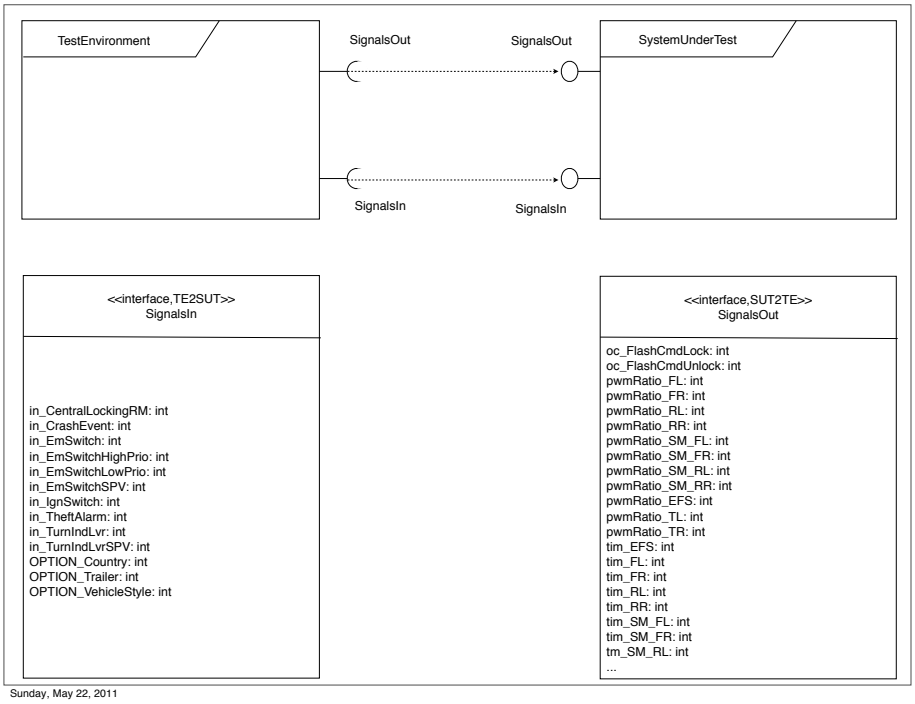

Fig. 1. Interface between test environment and system under test.

First-Level SUT Decomposition. Fig. 2 shows the functional decomposition of the SUT functionality. Component NormalAndEmerFlashing controls left/right turn indication, emergency flashing and the dependencies between both func-

tions (see below). Component OpenCloseFlashing models the indicator-related reactions to opening and closing vehicles with the central locking system. CrashFlashing models indications triggered by the crash impact controller. TheftFlashing controls reactions triggered by the theft alarm system. These functions interact with each other, as shown in the interface dependencies depicted in Fig. 2: the occurrence of a crash, for example, affects the emergency flash function, and opening a car de-activates a theft alarm. The local decisions of the above components are fed into the priority handling component where conflicts between indication-related commands are resolved: if, for example, the central locking system is activated while emergency flashing is active, the open/close flashing patterns (one time for open, 3 times for close) are not generated; instead, emergency flashing continues. Similarly, switching of the emergency switch has no effect if the high-priority emergency interface (in_EmSwitchHighPrio $\in \{0, 1\}$) is still active. Priority handling outputs the function to be performed and relays the left-hand/right-hand/both sides flashing information to the components OnOffDuration and AffectedLamps. The former determines the durations for switching lights on and off, respectively, during one flashing period. These durations depend both on the status of the ignition switch and the function to be performed. The latter specifies which lamps and dashboard indications have to participate in the flashing cycles. This depends on the OPTION_VehicleStyle which determines, for example, the existence of side marker lamps (interfaces pwmRatio_SM_FL, FR, RL, RR), and on the OPTION_Trailer which indicates the existence of a trailer coupling, so that the trailer turn indication lamps (pwmRatio_TL, TR) have to be activated. Moreover, the affected lamps and indications depend on the function to be performed: open-close flashing, for example, affects indication lamps on both sides, but the emergency flash switch indicator (pwmRatio_EFS) is not activated, while this indicator is affected by emergency, crash and theft flashing. The MessageHandling component transmits duration and identification of affected lamps and indicators on a bus and synchronizes the flash cycles by re-transmission of this message at the beginning of each flashing cycle. Finally, component lamp control comprises all output control functions, each function controlling the flashing cycles of a single lamp or dashboard indicator.

Behavioral Semantics. Model components behave and interact according to a concurrent synchronous real-time semantics, which is close to Harel's original micro-step semantics of Statecharts [1]. Each leaf component of the model is associated with a hierarchic state machine. At each step starting in some model pre-state σ_0, all components possessing enabled state machine transitions process them in a synchronous manner, using σ_0 as the pre-state. The writes of all state machine transitions affect the post-state σ_1 of the micro-step. Two concurrent components trying to write different values to the same variable in the same micro-step cause a *racing condition* which is reflected by deadlock of the transition relation and – in contrast to interleaving semantics – considered as a modeling error. Micro-steps are discrete transitions performed in zero time. Inputs to the SUT remain unchanged between discrete transitions. If the system

is in a stable state, that is, all state machine transitions are disabled, time passes in a delay transition, while the system state remains stable. The delay must not exceed the next point in time when a discrete transition becomes enabled, due to timeout condition. At the end of a delay transition, new inputs to the SUT may be placed on each interface. The distinction between discrete and delay transitions is quite common in concurrent real-time formalisms, and it is also applied to interleaving semantics, as, for example, in Timed Automata [2]. The detailed formal specification of the semantic interpretation of the model is also published on the web site given above.

Deployment and Signal Mapping. In order to support re-use, the model introduced in this section only specifies a *functional* decomposition. Its deployment on a distributed system of automotive controllers, the network topology, and the concrete interfaces implementing the logical ones shown in the model, depend on the vehicle production series. Even the observability of signals may vary between series, due to different increments in the test equipment. For this reason, model interfaces are mapped to concrete ones by means of a signal map: this map associates concrete signal names which may be written to or read from in the TE with the abstract signals occurring in the model and – in case of SUT outputs – specify their acceptable tolerances.

During test executions, the complete SUT model runs on the test engine against the SUT, in order to detect discrepancies between expected and observed behavior on the fly. During this back-to-back execution, all observable SUT outputs are checked against the expected ones calculated according to the model. To this end, all state machines in the SUT-portion of the model are transformed into separate tasks by a model-to-text generator. Depending on the TE infrastructure, these tasks can be distributed on several CPU cores and TE computers. As a consequence, every observable SUT output is continuously checked against its expected value. Values which are not observable are calculated by the corresponding model components and passed on to other components consuming these values, so that they can proceed their computations without the availability of the corresponding value produced by the SUT.

References

1. Harel, D., Naamad, A.: The statemate semantics of statecharts. ACM Transactions on Software Engineering and Methodology 5(4), 293–333 (October 1996)
2. Springintveld, J., Vaandrager, F., D'Argenio, P.: Testing timed automata. Theoretical Computer Science 254(1-2), 225–257 (March 2001)

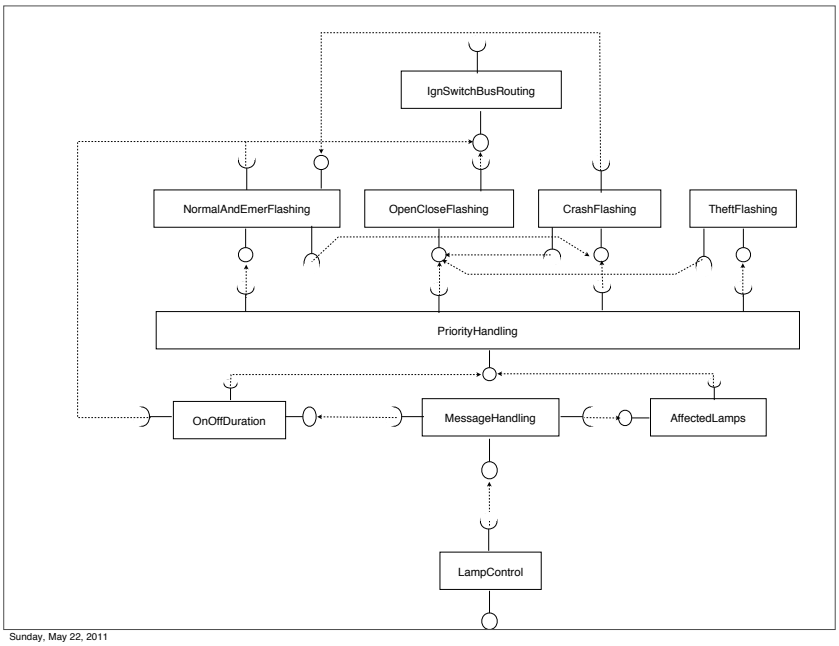

Fig. 2. First-level decomposition of system under test.