Applied Formal Methods - From CSP to Executable Hybrid Specifications

Jan Peleska
Technologie-Zentrum Informatik TZI,
Universität Bremen and
Verified Systems International GmbH,
jp@verified.de

Overview

1. **Overview: Practice Stimulates Theory**
   Applied CSP and Beyond
2. **Specification-Based Hard Real-Time Testing**
   Test Automation for TCSP
3. **Hybrid Low-Level Language Framework**
   Transformational semantics and hard real-time execution environment for hybrid formalisms
4. **Conclusion**

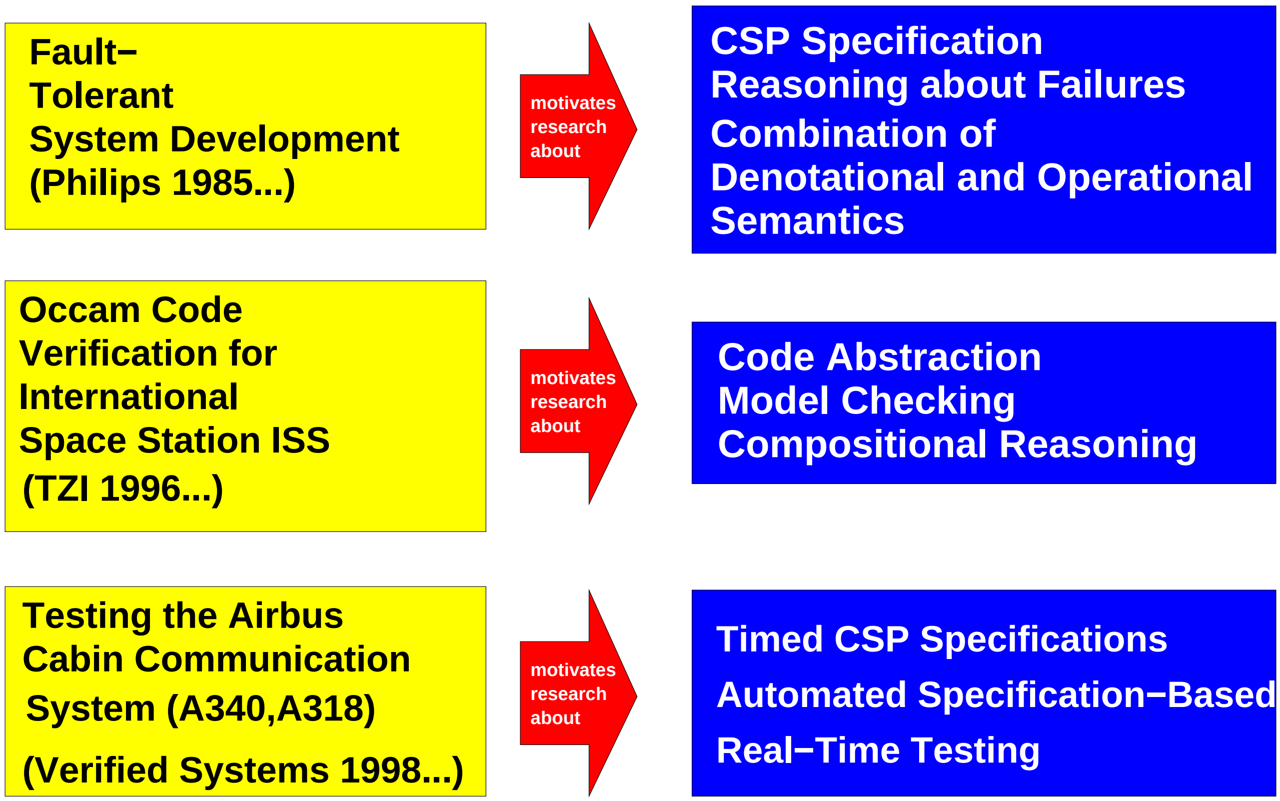
Extended abstract and presentation slides available under

`http://www.tzi.de/~jp`

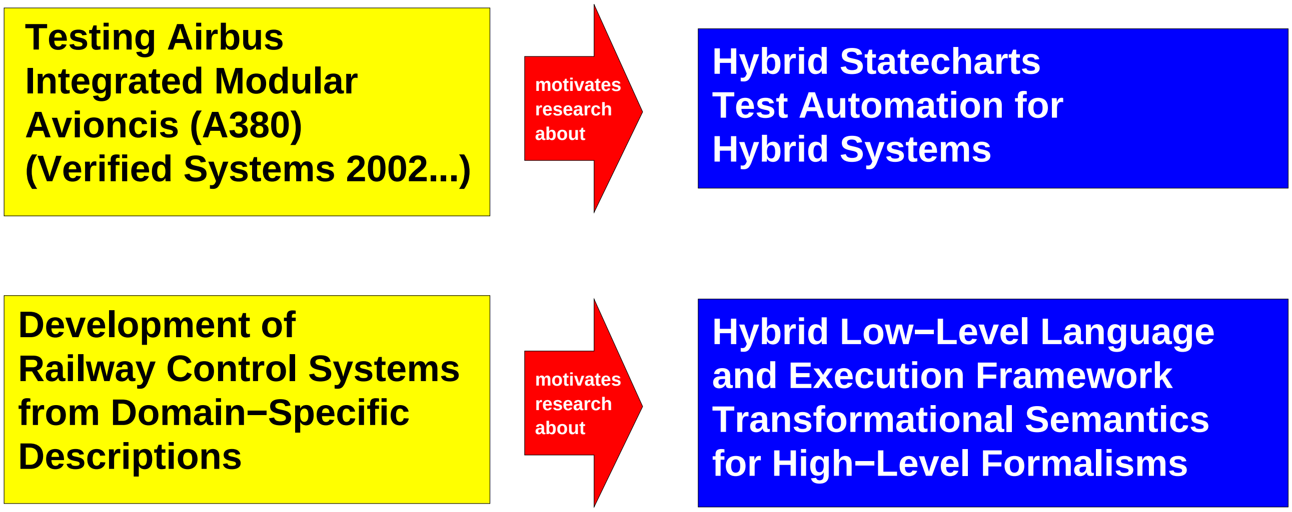
Overview

1. **Overview: Practice Stimulates Theory**
   Applied CSP and Beyond
2. **Specification-Based Hard Real-Time Testing**
   Test Automation for TCSP
3. **Hybrid Low-Level Language Framework**
   Transformational semantics and hard real-time execution environment for hybrid formalisms
4. **Conclusion**

Practice Stimulates Theory: Applied CSP



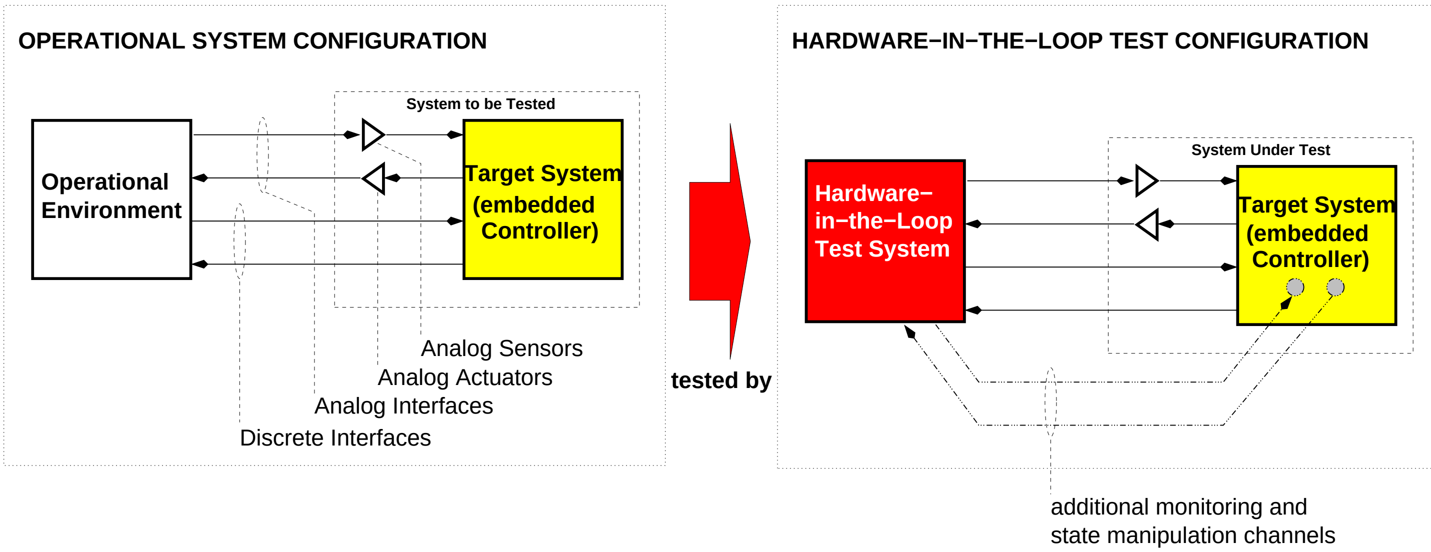
Practice Stimulates Theory: Beyond CSP



Overview

1. **Overview: Practice Stimulates Theory**
   Applied CSP and Beyond
2. **Specification-Based Hard Real-Time Testing**
   Test Automation for TCSP
3. **Hybrid Low-Level Language Framework**
   Transformational semantics and hard real-time execution environment for hybrid formalisms
4. **Conclusion**

Hardware-in-the-loop test configurations



Building blocks of a test automation system

- **Test Generator** creates test cases from specifications. This requires
  - **Environment Specification:** **simulation** of environment behaviour – **stimulation** all “relevant” events, in order to trigger specific reactions of system under test (SUT)
  - **SUT Specification:** to avoid creation of “irrelevant” tests
- **Test Driver** executes test cases in real-time
- **Test Oracle** checks SUT test execution against SUT specification
- **Test Monitor** checks whether test case executions are complete and required test coverage has been achieved

Structural Decomposition Theorem for Networks of Sequential TCSP Processes:

TCSP process P may be decomposed into

```
P' = PU [| {| s0, s1, ..., e0, e1, ... |} |] TIM
```

where PU only contains untimed CSP operators `[| |], |||, \, ->, [], |~|, ;` and `TIM` is an interleaving of **Timer Processes** T following the pattern

```
T = s.t -> ((WAIT t; e.t -> T) [] T)
```

P' is timed-failures equivalent to P.

Examples for structural decomposition: TCSP processes

```
P = WAIT t; a -> b-> P
Q = (a -> Q) [t> (x -> Q)
```

are transformed into

```
PU = s.t -> e.t -> a -> b -> PU
QU = s.u -> (a -> QU [] e.u -> x -> QU)
```

with timers

```
T1 = s.t -> ((WAIT t; e.t -> T) [] T)
T2 = s.u -> ((WAIT u; e.u -> T) [] T)
```

The system

```
SYS = P [| a |] Q
```

is transformed into

```
SYS' = ((PU [| a |] QU)
         [| s.t, s.u, e.t, e.u |]
        (T1 ||| t2))
       \ { s.t, s.u, e.t, e.u }
```

Basic approach to automated specification-based testing with TCSP:

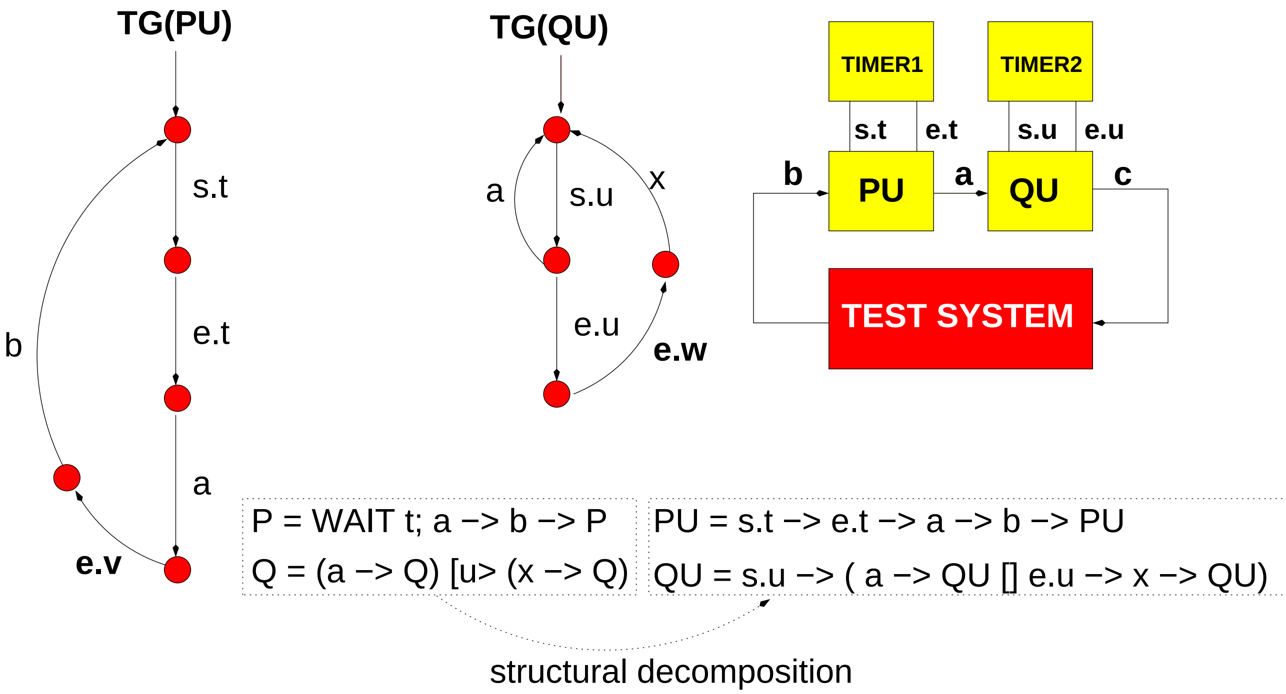
- Use **un-normalised transition graph** representation $TG(SYS')$ for SYS' as defined and implemented for Untimed CSP.
- $TG(SYS')$ **encodes complete timed failures model** of SYS', and therefore of the equivalent original process SYS.
- Test data generation and checking algorithms are based on traversal of $TG(SYS')$

Specification-Based Hard Real-Time Testing

- **Test Oracles** are implemented by **back-to-back checking** of SUT behaviour against $TG(SYS')$.
  - Since $TG(SYS')$ encodes all information about timers, correctness of timed traces can be checked on-the-fly in hard real-time for deterministic SUT $\longrightarrow$ currently applied to development of **built-in-test equipment** of train control systems.
  - Non-deterministic SUT may be checked in soft real-time by maintaining **set S of potential SUT states** in the test oracle. Timed trace behaviour of SUT is correct as long as $S \neq \emptyset$.

- **Test data generation for timed-failures testing** is based on a **zone abstraction** of $TG(SYS')$
  - During time intervals where no timer elapses, **SUT behaviour remains stable with respect to refused and accepted events**.
  - Since TCSP asserts maximal progress, **events refused by SUT may be probed** using test patterns like `TEST = (a -> ACCEPTED(a)) [t> REFUSED(a)` for small $t > 0$.
  - By blocking specific inputs to or outputs from SUT, testing environment may **explore SUT behaviour at time boundaries**

Specification-Based Hard Real-Time Testing



Overview

1. **Overview: Practice Stimulates Theory**
   Applied CSP and Beyond
2. **Specification-Based Hard Real-Time Testing**
   Test Automation for TCSP
3. **Hybrid Low-Level Language Framework**
   Transformational semantics and hard real-time execution environment for hybrid formalisms
4. **Conclusion**

Motivation

- Development, test and formal verification of **Hybrid Control Systems**, with discrete and time-continuous observables
- Requirements engineering with physical models requires **global observables**
- Specifications to be developed in **various formalisms**, e. g. Hybrid Statecharts, Hybrid Automata, Duration Calculus, Hybrid CSP, ...

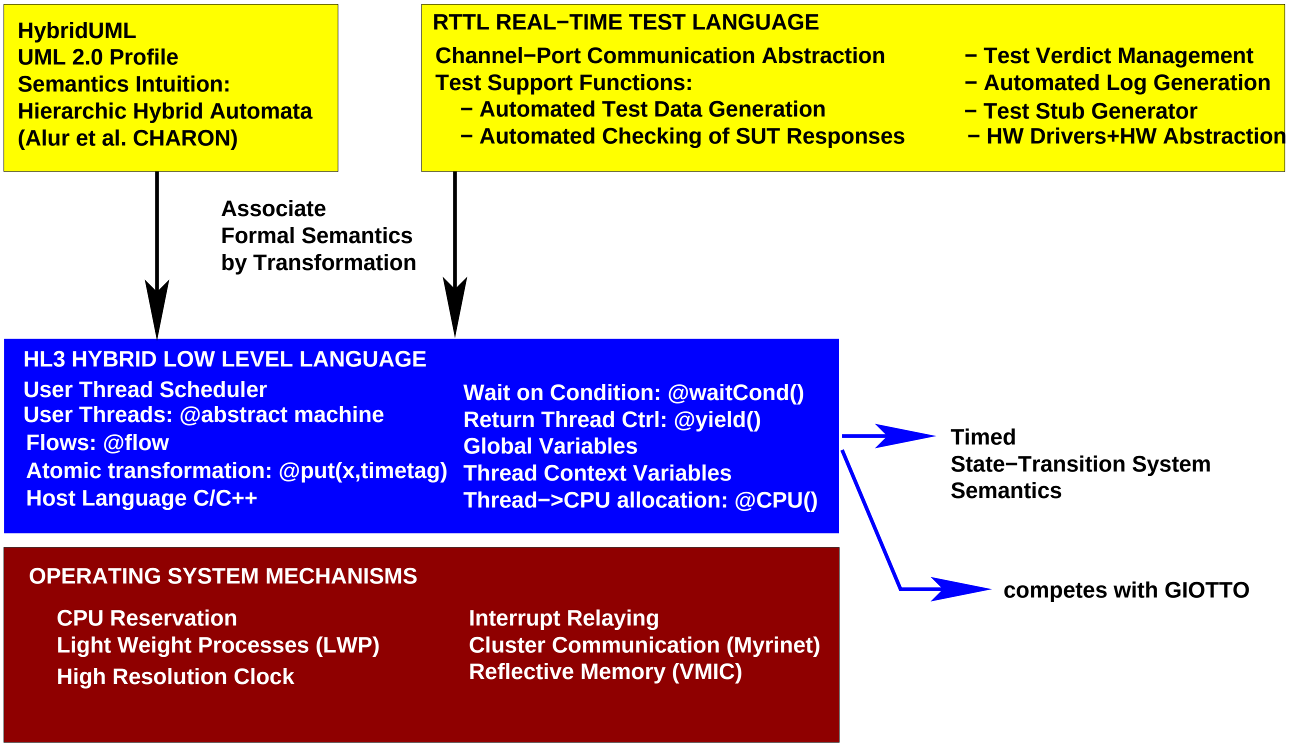
Hybrid Low-Level Language Framework

- **Specifications should be automatically transformed into hard real-time execution environment**, because development by stepwise refinement ...
  - offers too many degrees of freedom,
  - cannot be performed by domain specialists who are not formal methods specialists at the same time.
- **Execution environment should have well-defined real-time semantics**

Hybrid Low-Level Language Framework

- Semantics for new (high-level) formalisms should be **defined by transformation into execution environment**:
  - Obtain semantic specification model and executable system in a single step
  - Extensions of the new high-level formalism only require extension of the transformation
- **Automatic compilation seems feasible** for specific well-defined application domains, such as railway-control systems

Hybrid Low-Level Language Framework



Features of the Hybrid Low-Level Language Framework HL3.

- Designed as an alternative to Henzinger's GIOTTO – HL3 more flexible with respect to applicable programming paradigms
- **Timed transition system semantics**
- Atomic transformations implemented using **visibility time tags** for global state variables
- Discretised stepwise integration of flows with **guaranteed scheduling precision**
- High-level formalisms are transformed into HL3 Abstract Machines, global state, scheduling conditions and mappings between HW/SW interfaces.

Overview

1. **Overview: Practice Stimulates Theory**
   Applied CSP and Beyond
2. **Specification-Based Hard Real-Time Testing**
   Test Automation for TCSP
3. **Hybrid Low-Level Language Framework**
   Transformational semantics and hard real-time execution environment for hybrid formalisms
4. **Conclusion**

Conclusion

- Since 1985, **CSP has been applied successfully** by the author's research teams at TZI and verifications teams at Philips, DST and Verified Systems International GmbH for
  - **Formal specification and verification** of dependability mechanisms
  - **Code verification** by abstraction, model checking and compositional reasoning
  - Automated **hard real-time testing**
  - Automated generation of **real-time simulations**

Conclusion (. . . continued)

- The main **application domains** for the listed verification projects have been
  - **Fault-tolerant systems** (Philips)
  - **Space applications** (DASA Space Infrastructure)
  - **Avionics** (Airbus)
  - **Railway interlocking and train control systems** (Siemens)
  - **Automotive control** (Daimler Chrysler)

Conclusion (...continued)

- In order to incorporate physical modelling into the development phases, **Hybrid Statecharts** have been designed, combining
  - Global discrete and analogue observables,
  - Hierarchical real-time Statecharts
  - Invariants on global state
  - Flow conditions on time-continuous evolutions

Conclusion (. . . continued)

- To cope with the evolution of (hybrid) real-time formalisms, the **Hybrid Low-Level Language Frame Work (HL3)** has been designed to
  - "Compile" high-level specifications from different formalisms into HL3,
  - Define semantics of high-level specifications by transformation into HL3,
  - Generate executable hard-real time systems by transformation.

Conclusion (. . . continued)

- Ongoing research work focuses on
  - Completion of **theory and tool support for test automation based on TCSP**
  - Development of test **data generation methods and strategies for Hybrid Systems**
  - Combined **model checking and test automation**
  - Automated generation of **executable systems from domain-specific descriptions**

Contributions by . . .

. . . Peter Amthor, Kirsten Berkenkötter, Stefan Bisanz, Jan Bredereke, Bettina Buth, Markus Dahlweid, Christof Efkemann, Hans-Jürgen Ficker, Ulrich Hannemann, Anne E. Haxthausen, Oliver Meyer, Michael O. Möller, Anders Ravn, Willem-Paul de Roever, Raymond Scholz, Michael B. Schronen, Uwe Schulze, Hauke Steenbock, Aliki Tsiolakis, Cornelia Zahlten, . . .