Specifying and Implementing Control Software Systems with Hybrid Paradigms

DISIT-DSI
University of Florence
P. Bellini, P. Nesi, D. Rogai

DISIT Background

- Distributed systems
- Formal specification and execution
- Systems assessment, metrics
- Large project management
- Technology transfer
  - Bringing new technology
  - Pilot cases for Specific Industrial applications
Control systems are complex

- Complex systems
  - many interrelated parts
  - not trivial interactions
  - heterogeneous platforms
- Control systems \((real-time)\)
  - many parts with specific time behaviour
  - timed and scheduled interactions
  - software/hardware components

Dealing with temporal constraints

- Specification
  - describe the temporal constraints
- Design
  - model the temporal behaviour in the system
- Implementation
  - enforce the \(\text{correct}\) temporal behaviour
Temporal behaviour issues

- Description
  - temporal logic
  - abstract models (e.g. process algebra)
- Model
  - state charts, Petri nets, timed automata, process algebra
  - temporal logic (recently)
- Implementation
  - real-time operating systems
  - translation of semantic meaning of temporal constraints

Hybrid methodologies

- A hybrid methodology is capable to describe and model a timed system with a unique formalism
- hybrid provides both high-level denotational formalism for system properties and low-level operational language for the complete system behaviour
- i hybridi also refers to the possibility of integrating formal methods with low-level development framework (traditional programming like C++)
TILCO Framework as hybrid method

- TILCO temporal logic
  - suitable for describing and modeling time constraints (precedence, time-out, periodicity)
- direct execution of specification
  - which enforce a correct implementation
- C++ classes
  - handle to the low-level programming
- Connection predicates between denotational and operational semantics
- Semantic consistency check
  - to help connection between different semantics

TILCO framework overview

Complex System

Temporal/Behavioural requirements

Functional requirements

Architectural specification

Decomposition

Separation of concerns

Automatic support for infrastructure

TILCO time spec.

TILCO Framework

C++ modules
Event-driven and consistency

- Consistency check
  - The framework can assemble different languages and semantics
  - Consistency is managed by common names in both semantic worlds, these provide syntactic controls
  - Linked automatically by code generation

```
danger ⇔ critical@(-5,0]
alert ⇔ since(danger, ¬ack)

Input_critical::eval()
  { some_ext.readstate(); }
Output_danger::action()
  { some_ext.writestate(); }
```

Integrating logic execution

- The specification executor is embedded in a C++ architecture
- From the logic point of view, the C++ world can be read and write with I/O predicates
- From the C++ point of view, the logic engine can trigger events and has special methods to determine values of the predicates
- C++ is capable to handle low-level aspects of the programming
needed framework

- Integration of paradigms at all levels: model, verification, validation, proof, Ö.
- Assisted completion of the specification
- Direct execution of spec.
- Integration of cookbook for systems spec. and dev. with management and assessment

Thanks for your attention

http://www.dsi.unifi.it/~tilco/
Summary

- Control systems and temporal constraints
- Dealing with temporal constraints
- Hybrid methodologies
- TILCO framework
- Conclusions

Required attitudes

- **To do** list in the TILCO framework
  - collect time constraints, organizing them in a hierarchical model
  - list the common entities, preparing an architectural specification
  - specify the time requirements using common entities
  - develop the functional aspects in C++
  - connect the C++ classes with the temporal behaviour using the *event-driven* paradigm
Advantages of the framework

- Temporal constraints are
  - exhaustively validated by theorem proving
  - implemented by real-time execution of specifications
  - integrated with functional aspects by the framework connection C++ classes

- Functional aspects are
  - clearly distinguished from the temporal constraints

- Development and updates are
  - independent from one side to the other since the declared connections are still the same