A Rigorous Approach to the Design of Cyber-Physical Systems through Co-Simulation

John Fitzgerald, **Ken Pierce**, Carl Gamble
(Centre for Software Reliability, Newcastle University, UK)
Overview

• Our view of cyber-physical systems
• DESTECS concepts
• Personal transporter case study
• Error and recovery patterns
• Conclusions and future work
Cyber-physical Systems (1)

- Systems of interacting systems
  - Computing elements
  - Physical elements
  - Human interactions
- Complex, networked character
- Distributed control
- Error detection and recovery
Cyber-physical Systems (2)

- Requires collaborative development
- Analysis of models from different disciplines
  - Diverse cultures, abstractions, formalisms
- Typically tackled separately
- Require early allocation of responsibility to cyber and physical elements
- Need for **design space exploration**
DESTECS Approach
(www.destecs.org)

• Bridge gap between disciplines
  • Through co-simulation
  • Combine DE controller models and CT plant models
  • Collaboration while working with familiar formalism
• Developed a tool
• Developed methodological guidelines, incl. patterns
• Model failures, error detection and recovery
• Industrial partners and follow group
Basic Concepts

**Variables** modified during run

**Design parameters** fixed per run

**Ideal, realistic, faulty behaviours**

**Fault modelling** including error states & faulty functionality in the model

**Fault activation** during simulation managed by the script

Shared:
- design parameters
- variables
- events

Initialise variables and design parameters; swap components
Simulate user input and fault injection
Tool Concepts

Co-model ↔ Co-simulation
Example: Personal Transporter

(a) On/off switch
(b) Safety switch
(c) Direction switch
(d) Accelerometer
(e) Gyroscope
(f) Controller
(g) Wheel / motor
(h) Motor sensor
(i) Motor actuator
Personal Transporter (2)

```
class Controller

instance variables
  -- sensors
  private angle: real;
  private velocity: real;
  -- actuators
  private acc_out: real;
  private vel_out: real;
  -- PID controllers
  private pid1: PID;
  private pid2: PID;

operations
  public Step : () ==> ()
  Step() == duration(20) {
    dcl err: real := velocity - angle;
    vel_out.Write(pid2.Out(err));
    acc_out.Write(pid1.Out(angle));
  };

  public GoSafe : () ==> ()
  GoSafe() == {
    vel_out.Write(0);
    acc_out.Write(0);
  };

thread
  periodic(1E6, 0, 0, 0)(Step); -- 1kHz

end Controller
```
Personal Transporter (3)
What is a Pattern?

• “Each pattern describes a problem, … then describes the core of the solution” [AIS77]

• Applied to
  • architecture [AIS77]
  • object-oriented software [GHJV95]

• Includes: name, motivation, solution, related patterns

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

Ether Pattern (1)

- Model realistic communications
- Explicit model of an *ether*
  - a medium through which data must travel
- Ether can lose, duplicate, corrupt messages
  - can be tailored to offer different guarantees representing different media
Ether Pattern (2)

**Ether**
- lst: `map IListener to DeltaQueue`
  - `Register(IListener)`
  - `Broadcast(Message)`
  - `Step()`

**DeltaQueue**
- `entries: seq of Entry`
  - `push(nat1, Message)`
  - `pop(): seq of msg`

**Controller**
- `ether: Ether`
  - `Receive(seq of msg)`

**Diagram**
- `ether lst`
- `ether ctrl1` to `ether ctrl2`
- `vCPU` to `vBUS` to `CPU1` to `CPU2`
Monitor Pattern (1)

- Small, verifiable monitor that runs as a separate process
- Observes actions of the controller or other components
- Can intervene in undesirable or unsafe situations
- [See also kernel pattern]
Monitor Pattern (2)

```plaintext
class Monitor

instance variables
    ctrl: Controller

operations
    private Step: () => ()
        Step() ==
            if not safe(ctrl) then ctrl.GoSafe()
        post safe(ctrl)

functions
    private safe: Controller -> bool
        notsafe(ctrl) == ...

thread
    periodic(1E5,1,0,0)(Step); -- 10kHz

end Monitor
```
Conclusions

- Design of resilient cyber-physical systems through:
  - the collaborative creation of heterogeneous co-models, and
  - their analysis by co-simulation
- Supported by guidelines, patterns and tools
- Small embedded systems examples
Future Work and Challenges

• From embedded systems to cyber-physical
  • increasing capability of the tool support, e.g. new scripting features, batch execution

• Simulation of multiple systems
  • Issues of abstraction, scale

• Wider range of human behaviours
A Rigorous Approach to the Design of Cyber-Physical Systems through Co-Simulation

John Fitzgerald, Ken Pierce, Carl Gamble
(Centre for Software Reliability, Newcastle University, UK)