Embedding Formal Techniques into Industrial Product Development

Experiences with the DESTECS approach

Marcel Verhoef
Chess eT International B.V.
Marcel.Verhoef@chess.nl

Joint work with John Fitzgerald, Peter Gorm Larsen, Bert Bos, Jan Broenink, Yoni de Witte, Peter van Eijk, Christian Kleijn and (many) others
Overview of this talk

• Challenges in developing dependable embedded systems

• Collaborative modeling: the DESTECS approach

• Example industrial applications

• Live tool demo

• Conclusions
Embedded Systems Development (1)

- Highly competitive marketplace:
  - Requirements are volatile
  - Time to market is key

- Products are complex

- Early design stages are vulnerable to failure:
  - Engineering disciplines have distinct methods & tools
  - Design choices are often implicit or experience based
  - System dynamics are complex to grasp and express
  - Dependability (faults, fault tolerance) is often crucial
• Problem decomposition into disciplines
• Traditional approaches are “one discipline at a time”
• Concurrent engineering required to improve time to market
• ... but important properties are multidisciplinary
• ... and so weaknesses are exposed late (integration)
• So: how to cross the boundaries between disciplines?
Embedded Systems Development (3)

- Design gaps between disciplines lead to errors in designs
- Many of these errors are detected too late: during testing of first physical prototype
- Example: paper path setup
- Paper jams for high speed paper handling
Embedded Systems Development (4)

We offer three kinds of service:

GOOD - CHEAP - FAST
You can pick any two

GOOD service CHEAP won’t be FAST
GOOD service FAST won’t be CHEAP
FAST service CHEAP won’t be GOOD
Industrial “holy grail” : Design Space Exploration

Explore and restrict to feasible solutions
Managing space of models (patterns)
Visualisation of test outcomes
“Ranking”
Managing scenarios
DESTECS (www.destecs.org)

- Bridge design gap between disciplines through co-simulation
- Develop methods and tools
- Modeling of faults and fault tolerance mechanisms

Restriction to discrete-event domain and continuous-time domain
- Industrial Follow Group will monitor results and provide challenges
- EU FP7 project runs from 01-2010 until 12-2012
DESTECS in a nutshell (1)

Model Based Design:
• controller in discrete event domain
• plant in continuous time domain

Co-simulation:
• coupling disciplines
• analysis on virtual prototype

Automated Co-model Analysis

Methodological guidelines
DESTECS in a nutshell (2)

Cause of the problems
• Geometry changes were not adequately communicated
• Errors in acceleration and deceleration paths

Results
• These errors can be detected in an early stage of the design through co-simulation
• Dependability can be assessed by fault injection


Modelling & Simulation

Model

Abstract
Competent if detailed enough for analysis
Variables
Design Parameters fixed per run

Model Interface

Faults – errors – failures
Fault Modelling: including error states & faulty functionality in the model
Fault Injection during a simulation managed by script

Script

Runs a simulation
Initialises variables and design parameters
Forces selections and external updates, e.g. set point
co-modelling & co-simulation

**Shared**
- design parameters
- variables
- events

- **DE Model**
- **Contract**
- **CT Model**

**Co-model Interface**

**Script**
DESTECS Tool Architecture

Discrete-event system  Co-Simulation engine  Continuous-time system

Overture  DESTECS Tool  20-sim

Formally specified semantics of the DE / CT integration (SOS)
Co-Simulation architecture
Example: water tank

\[ \frac{dV}{dt} = \varphi_{in} - \varphi_{out} \]

\[ \varphi_{out} = \begin{cases} 
\frac{\rho \cdot g}{A \cdot R} \cdot V & \text{if valve open} \\
0 & \text{if valve closed} 
\end{cases} \]
Example: water tank

```plaintext
class Controller

instance variables
private i : Interface

operations
async public Open:() ==> ()
    Open() == duration(50)
    i.SetValve(true);

async public Close:() ==> ()
    Close() == cycles(1000)
    i.SetValve(false);

sync
mutex(Open, Close);
mutex(Open); mutex(Close)

end Controller
```
Example: water tank

```
class Controller

  open: () => ()
  close: () => ()

  level: real
  valve: bool

end Controller
```

---

**FlowSource**

- Shared design parameters
  ```
  sdp real maxlevel;
  sdp real minlevel;
  ```

- Monitored variables (seen from the DE controller)
  ```
  monitored real level := 0.0;
  ```

- Controlled variables (seen from the DE controller)
  ```
  controlled bool valve := false;
  ```

- Events
  ```
  event high;
  event low;
  ```

- Link events to operations
  ```
  event high = System.Controller.Open;
  event low = System.Controller.Close;
  ```

---

**Interface**

- Open()
  ```
  duration (50)
  i.SetValve (true);
  ```

- Close()
  ```
  cycles (1000)
  i.SetValve (false);
  ```
Example: water tank

This is VDM-RT: real-time extensions

Interface manages the shared variable for the valve setting.

duration constrain (absolute) time taken by the asynchronous ops.

cycles constrain (relative) time taken by the ops, depending on deployment

Single active thread accessing the valve

class Controller

instance variables
private i : Interface

operations
async public Open():() => ()
  Open() == duration(50)
  i.SetValve(true);

async public Close():() => ()
  Close() == cycles(1000)
  i.SetValve(false);

sync
mutex(Open, Close);
mutex(Open); mutex(Close)

destecs
Example: modelling faults

class ValveActuator

types
ValveCommand = <OPEN> | <CLOSE>;

instance variables
private i : Interface;

operations

public Command: ValveCommand ==> ()
Command(c) == duration(50)

cases c:
<OPEN> -> i.SetValve(true),
<CLOSE> -> i.SetValve(false)
end

post i.ReadValve() ==> c = <OPEN> and not i.ReadValve() ==> c = <CLOSE>

end ValveActuator
Example: modelling faults

A stuck valve ...

class ValveActuator
types
ValveCommand = <OPEN> | <CLOSE>;
instance variables
  private i : Interface;
  private stuck : bool := false
operations
  private SetStuckState : bool == ()
  SetStuckState(b) == stuck := b
  post stuck <= b and not stuck <= not b;

  public Command : ValveCommand == ()
  Command(c) ==
    duration(50)
    if not stuck then
      cases c:
        <OPEN> -> i.SetValve(true),
        <CLOSE> -> i.SetValve(false)
    end
  pre not stuck
  post i.ReadValve() <= c = <OPEN> and
    not i.ReadValve() <= c = <CLOSE>
  errs STUCK : stuck ->
    i.ReadValve() = ~i.ReadValve();
end ValveActuator
Example: modelling faults

A leak in which liquid flows from the tank at a constant rate. Modelling DE-side entails DE accessing flow rate. So this may be more appropriately modelled CT-side. CT-side also allows for more sophisticated fault models, e.g. leak flow rate depends on pressure.
Example: water tank
DESTECS case studies
Chess – Self Balancing Scooter (1)
Chess – Self Balancing Scooter (2)

• ChessWay is a technology and methodology demonstrator
  • First generation: single controller driving both wheels
  • Second generation: two controllers, one driving a wheel each
  • Third generation: wireless communication sensors ↔ controllers

• ChessWay exhibits typically modelling challenges common to many Chess products under development
  • Simple nominal behavior, relatively easy to engineer
  • System behavior becomes very complex when faults and fault tolerance comes into play
  • Managing this complexity is the key to improve productivity (pre-empt cost for complex system integration and validation)

• Typical design questions we want to address a-priori:
  • Can we demonstrate the robustness of the ChessWay design?
  • Can we assess the impact of changes on the current design?
Chess – Self Balancing Scooter (3)
Chess – Self Balancing Scooter (4)

wireless communication
No Wires? Have You Lost Your Mind?

break reliability (response within 250 msec) 99.999999999997% (formally proven)

would you trust this device?

Modeling the SBS (1)

user behavior | controller | interface | plant

discrete event | contract | continuous time
Modeling the SBS – continuous time (2)
Modeling the SBS – continuous time (3)
Modeling the SBS – discrete time (4)
Modeling the SBS – discrete time (5)
Analysis of SBS co-models (1)
Analysis of SBS co-models (2)

CT model running in 20-sim and DE model running in Overture using DESTECS cosim tool
Movie available on http://www.destecs.org and http://www.youtube.com/watch?v=HccXkd4gWys
Verhaert – Dredging Excavator

- Overload and end-stop protection
- Emergency switch and system reset behavior
- Advanced operator assistance (i.e. perform a straight dig)
Observations and conclusions

• **Formal Methods helps to de-risk development**
  • including de-risking detailed formal analysis
  • providing rapid, accurate, but maybe incomplete analyses
  • training and methodological guidelines are crucial
  • start formal, (higher chance to) remain formal

• **What does formalism buy us?**
  • Sound semantic basis for the co-simulation tools & methods
  • Comprehensive analytic solutions are a long way off…
    … so (trustworthy) executable specifications are legit!

• **Co-modelling exposes issues that are often implicit**
  • In individual disciplines (we knew that already!)
  • And across boundaries, e.g. where to model faults
  • Expose potential problems earlier (no-brainer)

• **Co-simulation is enabler for Design Space Exploration**
  • Collaboration (also between researchers and practitioners 😊)
thank you for your attention!

Any questions?

Some pointers to related information resources

http://www.destecs.org  http://www.20sim.com