The PROBAnD
Railway Crossing Specification

Stefan Queins, Andreas Metzger
Department of Computer Science
University of Kaiserslautern, Germany

Introduction

Our Background

• efficient development of large/complex reactive systems
• our "techniques"
  – reuse ⇒ efficiency and product quality
  – prototyping ⇒ verification and early validation
  – object-orientation ⇒ handling complexity
  – domain specialization ⇒ precise definition of development activities and documents

Our Motivation for Using SDL

• executable specification ⇒ prototype generation
• existing tool support (Telelogic Tau)
• object-oriented concepts (block types)
• stateflow diagrams ⇒ suited for reactive system domain
Specification Method

The PROBAnD Requirements Analysis Method

- Environment Description
- Problem Description
- Needs
- Object Structure Specification
- System Description
- Task Description
- HTML Object Type
- Object Structure
- System Modeling
- SDL Block Type
- Requirements Specification
- Prototype Generation
- Control System Prototype

Object Structure Specification
Object Structure Specification

Aggregation to System

RWCrossing

Controller-Side (Ctrl)

GateCtrl

CrossingCtrl

TrackCtrl

RoadCtrl

TrainSensCtrl

SignalCtrl

gate1

track[1-n]

road1

ts[1-2]

sig1

white: static instantiation, green: dynamic instantiation

Environment-Side (Env)

EnvGate

EnvTrainSens

EnvTrain

EnvRoad

ec1

Gate1

Track[1-n]

Road1

ts[1-2]

train[1-n]

sig1

ts[1-2]

Communication Means

Object Structure Specification

RPC-Communication

Controller-Side (Ctrl)

CrossingCtrl

GateCtrl

TrackCtrl

RoadCtrl

TrainSensCtrl

SignalCtrl

gate1

track[1-n]

road1

ts[1-2]

sig1

Environment-Side (Env)

EnvCrossing

EnvGate

EnvTrain

EnvTrainSens

EnvRoad

ec1

SigntCommunica

gate1

track[1-n]

road1

ts[1-2]

train[1-n]

sig1

ts[1-2]
Task and System Description

HTML Object Type Document: SignalCtrl

```
<table>
<thead>
<tr>
<th>Document</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DocName</td>
<td>Project</td>
</tr>
<tr>
<td>SignalCtrl</td>
<td>RailwayCrossing</td>
</tr>
</tbody>
</table>

Description

Components

<table>
<thead>
<tr>
<th>ModelName</th>
<th>Number</th>
<th>Type</th>
<th>Version</th>
</tr>
</thead>
</table>

Tasks

<table>
<thead>
<tr>
<th>ModelName</th>
<th>Description</th>
<th>StrategyDescription</th>
<th>Strategy Type</th>
<th>Realized Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>SignalCtrlTask1</td>
<td>set state of train signal</td>
<td>After receiving setSignal State, set the physical device to the given value and set curState to the actual parameter provided.</td>
<td>functional</td>
<td></td>
</tr>
<tr>
<td>SignalCtrlTask2</td>
<td>report current state</td>
<td>Upon reception of getSignal State, send newSignal State with actual parameter curState.</td>
<td>functional</td>
<td></td>
</tr>
</tbody>
</table>
```
Task and System Description

HTML Object Type Document: SignalCtrl (cont’d)

<table>
<thead>
<tr>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelName</td>
</tr>
<tr>
<td>curState</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelName</td>
</tr>
<tr>
<td>setSignalState</td>
</tr>
<tr>
<td>getSignalState</td>
</tr>
<tr>
<td>newSignalState</td>
</tr>
</tbody>
</table>

Task and System Description: Controller

**Strategy: SignalCtrl**

“SignalCtrlTask1”

- setSignalState(go)/RPC to EnvSignal
- setSignalState(halt)/RPC to EnvSignal

**Strategy: GateCtrl**

- similar to SignalCtrl

**Strategy: TrainSensCtrl**

- RPC from EnvTrainSens/newTrain
- [last RPC from EnvTrainSens > wheelDelay]
- RPC from EnvTrainSens

**Strategy: RoadCtrl**

- RPC from EnvGate/newCars
Task and System Description: Controller

Strategy: TrackCtrl
“greedy halt”

propagation of signal setting

Task and System Description: Controller

Strategy: CrossingCtrl
“trains take precedence”

“cars take precedence”
Task and System Description: Controller

**Strategy: CrossingCtrl**

“fast trains, then cars take precedence”

- setGate(closed)
- askForGo(\(\text{no cars waiting OR track type = fast}\)) \(\implies\) setSignalState(go)
- askForGo(\(\text{no cars waiting OR track type = fast}\)) \(\implies\) setSignalState(go), setGate(closed)
- newCars(\(\text{no cars waiting}\)) \(\implies\) setSignalState(go) to all "waiting" tracks, setGate(closed)

**further responsibilities**

- “user halt”
- change strategies
- keep track of “trains on track” and “cars waiting”
- (dynamic) instantiation and initialization of the tracks

Task and System Description: Environment

**Strategy: EnvSignal**

- RPC from SignalCtrl/newSignalState(go)
- RPC from SignalCtrl/newSignalState(halt)

**Strategy: EnvGate**

- similar to EnvSignal

**Strategy: EnvTrainSens**

- newTrain/RPC to TrainSensCtrl
- \([\text{last newTrain} > \text{wheelDelay} \times \text{nbrWheels}]\) \(\implies\) RPC to EnvTrainSens

**Strategy: EnvRoad**

- gate closed \(\implies\) increment “nbr of cars”; gate opened \(\implies\) decrement “nbr of cars”
- “nbr of cars” passes threshold \(\implies\) RPC to RoadCtrl
Task and System Description: Environment

Strategy: EnvTrain

"Layout" of Environment

- distances are mapped to durations
  - $ts1Dur$, $ts1SigDur$, $sigTs2Dur$
  - $minDistDur$ = minimal "distance" between trains
  - $brakeDur$ = "distance" from initiation of braking phase until full halt

- position of trains
  - $distDur$ = distance from "start" of track
  - approach: $distDur \leq ts1Dur$
  - signal: $ts1Dur \leq distDur \leq ts1Dur + ts1SigDur$
  - gate: $ts1Dur + ts1SigDur \leq distDur \leq ts1Dur + ts1SigDur + sigTs2Dur$
Task and System Description: Environment

**Strategy: EnvTrack**

responsibilities

- create trains
- propagate newTrainStop and newTrainStart signals
  - each train “knows” if it is followed by another train
- propagate trainArrived and trainPassed to EnvTrainSens instances
  - trainArrived: ts1
  - trainPassed: ts2

**Strategy: EnvCrossing**

responsibilities

- (dynamic) instantiation and initialization of the tracks

System Modeling
**System Modeling**

**Object Types = SDL Block Types**

**Static Instantiation and Initialization of Object Types**
System Modeling

Dynamic Instantiation and Initialization of Object Types

process CrossingCtrl

process TrackCtrlMaster

Realization of Strategies: TrackCtrl

System Modeling

Realization of Strategies: TrackCtrl

process TrackCtrl

1. (4)

DCL instName Charstring;
DCL senderName CharString;
DCL sigState SignalStateType;

init

getSignalState (‘sig1’//instName) via a.i2

init

newSignalState (senderName, sigState)

sigState

halt

go

System Modeling

Realization of Strategies: TrackCtrl

process TrackCtrl

2. (4)

DCL curTrainNbr Integer := 0;

init

newTrain (senderName)

false

curTrainNo = 0

true

false

true

false

false

-
**System Modeling**

**Realization of Strategies: TrackCtrl**

```plaintext
process TrackCtrl

- go
  - newTrain(senderName)
    - senderName = 'ts1'//instName
      - true
      - curTrainNbr = curTrainNbr - 1
          - curTrainNbr = curTrainNbr + 1
          - halt

- curTrainNbr = 0
  - false

- newTrainCtrl
  - curTrainNbr = curTrainNbr - 1
      - curTrainNbr = curTrainNbr + 1
      - noTrains(instName)
          - halt

- printIt('ERROR', NOW)
- setSignalState(sig1//instName, halt)
  - via a.i2

- newSignalState(instName, halt)
  - via oo

- noTrains(instName)
  - halt
```

---

**Prototype Generation and Execution**

```
<table>
<thead>
<tr>
<th>Environment Description</th>
<th>Problem Description</th>
<th>Needs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object Structure</td>
<td>System Description</td>
<td>Task Description</td>
</tr>
<tr>
<td>Specification</td>
<td>HTML Object Type</td>
<td>Tasks</td>
</tr>
<tr>
<td>Object Structure</td>
<td>System Modeling</td>
<td></td>
</tr>
<tr>
<td>Requirements Specification</td>
<td>SDL Block Type</td>
<td>Prototype Generation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Control System Prototype</td>
</tr>
</tbody>
</table>
```
Prototype Generation and Execution

Generation Process

- SDL Specification
- SDL Block Type
- C-Code Generator
- Prototype Code
- C-Code Compiler
- Control System Prototype

Coupling to Environment and Data Logging

- Control System Prototype
- Physical Environment
- Java Panel
- Trace Files
- Log Files

Prototype Generation and Execution

Test Results from Trace Files: “Testcase A”

- cc1: CrossingCtrl
  - setGate (opened)
- gate1cc1: GateCtrl
  - askForGo (‘track1’)
- road1cc1: RoadCtrl
- track1cc1: TrackCtrl
- sig1track1cc1: SignalCtrl
- ts1track1cc1: TrainSensCtrl1
- ts2track1cc1: TrainSensCtrl

- newTrain (‘ts1track1’)
- wheelT(2.0000)
- wheelT(2.0000)
- wheelT(2.0000)

- setSignalState (go)
- setGate (closed)
- setSignalState (go)

- trace files used for error identification (e.g. lost signals, signal propagation)
Prototype Generation and Execution

Test Results from Log Files “fast trains, then cars take precedence”

Specication Effort

Data Acquisition

- HTML forms to specify:
  - execution time, version comments, used documents, responsible developers
- “Version History” (HTML document) is generated from form data

Results

<table>
<thead>
<tr>
<th>Activity</th>
<th>Initial Effort [hrs]</th>
<th>Modification Effort [hrs]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Env</td>
<td>Ctrl</td>
</tr>
<tr>
<td>Object Structure Specification</td>
<td>8.2</td>
<td>2.7</td>
</tr>
<tr>
<td>Task + System Description</td>
<td>17.2</td>
<td>3.3</td>
</tr>
<tr>
<td>System Modeling</td>
<td>25.4</td>
<td>6.0</td>
</tr>
<tr>
<td><strong>∑</strong></td>
<td><strong>25.4</strong></td>
<td><strong>6.0</strong></td>
</tr>
</tbody>
</table>

total effort: **73.4 hrs = 2 person weeks**
Conclusions and Perspectives

Use of MSCs

- detailed trace of prototype/system behavior
- error identification (carried out using textual MSCs and specialized tools)

Use of SDL

- EFSMs = state machines + data structures + control constructs
  - state machines: specification of object states (e.g. Gate: opened/closed)
  - data structures + control constructs: power of programming language (e.g. TrackCtrl: curTrainNbr)
- “active” processes: local control loops (e.g. EnvTrain)
- signals: event-based communication (e.g. newTrain)
- dynamic process instantiation and management (pid = instName)

Potential Benefits of SDL-2000

- agent concept
  - easier dynamic instantiation (automatic “instantiation of sub-agents”?)
  - more comprehensible (less levels of nesting)