An Exercise in Iterative Domain-Specific Language Design

Marcel van Amstel
Mark van den Brand
Luc Engelen
Introduction

• Simple Language for Communicating Objects (SLCO)
  • What are the main influences on the evolution?
  • What effect did these influences have?
  • How can we improve the development process?
Introduction

• We developed a language to describe concurrent, communicating objects
• This language is meant for
  • Specification
  • Simulation – *by transformation to POOSL*
  • Execution – *by transformation to NQC*
  • Verification – *by transformation to Promela*

Short history:
1. Alternative syntax for a simulation language to debug software
2. Feasibility study concerning code generation
3. Investigated model checking to detect design flaws

• Target platforms were leading in the development
Background

- Using DSL’s to avoid learning formal languages
  - Generating executables from graphical models
    Vanderlande Industries
  - Generating POOSL simulations from graphical models
    ASML

- Different views on similar problem domains
  - DIVINE model checker
    Masaryk University
  - SPIN model checker
    Bell Labs
  - CHI, POOSL and mCRL2
    Eindhoven University of Technology
  - Uppaal
    Uppsala and Aalborg University
Domain-Specific Language (DSL)

- Graphical + textual syntax
- Concurrent communicating objects
  - Synchronous signals
  - State machines
Domain-Specific Language (DSL)

- Graphical + textual syntax
- Concurrent communicating objects
  - Synchronous signals
  - State machines

```
model LegoCase {
  classes
  Simple {
    ports
      Middle Sensor Motor

    state machines
      SM {
        initial Start

        state Running Block

        transitions
          Block from Running to Block {
            trigger Block() from Sensor
          }

          Run from Block to Running {
            trigger
              BlockPassed() from Sensor
            effect
              send Off() to Motor;
              send Block() to Middle;
              send On() to Motor
          }
      }
  }
}
```

/ Software Engineering and Technology
Purpose of the DSL

- **Aims**
  - **Specification** – *Concise description of systems*
  - **Simulation** – *Executable models*
  - **Execution** – *Code generation*
  - **Verification** – *State-space exploration / Model checking*

- **Users specify concise models**
- **Code for simulation, execution and model checking is automatically generated**
Purpose of the DSL
Purpose of the DSL

/ send On() to Motor

Running

Block() from Sensor

Block

BlockPassed() from Sensor

/ send Off() to Motor;
send Block() to Middle;
send On() to Motor

/ 

BlockPassed() from Sensor

/ send Off() to Motor

Idle

Block() from Left

/ send Right() to Motor

Running

Block() from Right

/ send Left() to Motor

Block

/ Software Engineering and Technology

TU/e
Technische Universiteit Eindhoven
University of Technology
Purpose of the DSL

/ send On() to Motor

Running

Block() from Sensor

Block

BlockPassed() from Sensor
/ send Off() to Motor;
send Block() to Middle;
send On() to Motor

Idle

Block() from Left
/ send Right() to Motor

Running

Block() from Right
/ send Left() to Motor

Block

BlockPassed() from Sensor
/ send Off() to Motor

Software Engineering and Technology
One component
- 6 concurrent tasks
- 10 shared variables for communication between tasks (state machines)
- 425 lines of code (including comments)
Purpose of the DSL

One component
- 6 concurrent tasks
- 10 shared variables for communication between tasks (state machines)
- 425 lines of code (including comments)
## Approach

- **Target languages**
  - POOSL for simulation
  - NQC for execution
  - Promela for simulation and model checking

<table>
<thead>
<tr>
<th></th>
<th>(A)Synchronous signals</th>
<th>Channel directionality</th>
<th>Channel reliability</th>
<th>Number of concurrent objects</th>
<th>Number of channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCO</td>
<td>Synchronous</td>
<td>Bidirectional</td>
<td>Lossless</td>
<td>Unlimited</td>
<td>Unlimited</td>
</tr>
<tr>
<td>POOSL</td>
<td>Synchronous</td>
<td>Bidirectional</td>
<td>Lossless</td>
<td>Unlimited</td>
<td>Unlimited</td>
</tr>
<tr>
<td>Promela</td>
<td>Both</td>
<td>Unidirectional</td>
<td>Lossless</td>
<td>Unlimited</td>
<td>Unlimited</td>
</tr>
<tr>
<td>NQC</td>
<td>Asynchronous</td>
<td>Bidirectional</td>
<td>Lossy</td>
<td>Limited</td>
<td>1</td>
</tr>
</tbody>
</table>
Approach

- **Target languages**
  - POOSL for simulation
  - NQC for execution
  - Promela for simulation and model checking

<table>
<thead>
<tr>
<th></th>
<th>(A)Synchronous signals</th>
<th>Channel directionality</th>
<th>Channel reliability</th>
<th>Number of concurrent objects</th>
<th>Number of channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCO</td>
<td>Synchronous</td>
<td>Bidirectional</td>
<td>Lossless</td>
<td>Unlimited</td>
<td>Unlimited</td>
</tr>
<tr>
<td>POOSL</td>
<td>Synchronous</td>
<td>Bidirectional</td>
<td>Lossless</td>
<td>Unlimited</td>
<td>Unlimited</td>
</tr>
<tr>
<td>Promela</td>
<td>Both</td>
<td>Unidirectional</td>
<td>Lossless</td>
<td>Unlimited</td>
<td>Unlimited</td>
</tr>
<tr>
<td>NQC</td>
<td>Asynchronous</td>
<td>Bidirectional</td>
<td>Lossy</td>
<td>Limited</td>
<td>1</td>
</tr>
</tbody>
</table>

bridging all gaps at once
Approach

- Target languages
  - POOSL for simulation
  - NQC for execution
  - Promela for simulation and model checking

<table>
<thead>
<tr>
<th></th>
<th>(A)Synchronous signals</th>
<th>Channel directionality</th>
<th>Channel reliability</th>
<th>Number of concurrent objects</th>
<th>Number of channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCO</td>
<td>Synchronous</td>
<td>Bidirectional</td>
<td>Lossless</td>
<td>Unlimited</td>
<td>Unlimited</td>
</tr>
<tr>
<td>POOSL</td>
<td>Synchronous</td>
<td>Bidirectional</td>
<td>Lossless</td>
<td>Unlimited</td>
<td>Unlimited</td>
</tr>
<tr>
<td>Promela</td>
<td>Both</td>
<td>Unidirectional</td>
<td>Lossless</td>
<td>Unlimited</td>
<td>Unlimited</td>
</tr>
<tr>
<td>NQC</td>
<td>Asynchronous</td>
<td>Bidirectional</td>
<td>Lossy</td>
<td>Limited</td>
<td>1</td>
</tr>
</tbody>
</table>

bridging all gaps at once

1..N transformations for each gap
## Approach

- **Target languages**
  - POOSL for simulation
  - NQC for execution
  - Promela for simulation and model checking

<table>
<thead>
<tr>
<th>(A)Synchronous signals</th>
<th>Channel directionality</th>
<th>Channel reliability</th>
<th>Number of concurrent objects</th>
<th>Number of channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCO</td>
<td>Both</td>
<td>Bidirectional</td>
<td>Both</td>
<td>Unlimited</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unlimited</td>
</tr>
<tr>
<td>POOSL</td>
<td>Synchronous</td>
<td>Bidirectional</td>
<td>Lossless</td>
<td>Unlimited</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unlimited</td>
</tr>
<tr>
<td>Promela</td>
<td>Both</td>
<td>Unidirectional</td>
<td>Lossless</td>
<td>Unlimited</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unlimited</td>
</tr>
<tr>
<td>NQC</td>
<td>Asynchronous</td>
<td>Bidirectional</td>
<td>Lossy</td>
<td>Limited</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

- Bridging all gaps at once
- 1..N transformations for each gap
Transformations

- **Endogenous transformations**
  - Bridge gaps between platforms
  - From SLCO to SLCO

- **Exogenous transformations**
  - Transform DSL to platforms
  - From SLCO to NQC, Promela and POOSL
Target platforms

1. Defined a simple specification language (core)
2. Iteratively:
   1. Chose purpose for the DSL
   2. Chose target language that could fulfil each purpose
   3. Extended the core language
   4. Adapted the core language whenever needed

• The target languages were fixed
• Every allowable SLCO model must have an equivalent counterpart on each target platform
Language evolution

- Phases in DSL design and related activities
  - Analysis
    - Domain analysis
    - Defining mappings from SLCO to target platform
  - Design
    - Defining the language
  - Implementation
    - Implementing the language
    - Implementing transformations
  - Use
    - Creating and using models
    - Interpreting intermediate models
Language features per activity

- Defining the language
  - Synchronous signals
    _Concise specification_

- Defining mappings (for each platform)
  - Asynchronous signals and lossy channels
    _Lego Mindstorms_
  - Triggers and guards
    _Promela_

- Interpreting intermediate models
  - Fine-grained transformations
    _Understandability, state-space generation_
Triggers and guards

\[ s(a, b, c) \text{ from Port1} \]
\[ [a == b + 1] \]

\[ [x == y] \]

\[ s(a) \text{ from Port1} \]
Fine-grained transformations

Specification

Asynchronous communication

Lossless communication over lossy channel

Mutual exclusive access to channel
Fine-grained transformations

Specification → Asynchronous communication

Lossless communication over lossy channel

Mutual exclusive access to channel
Fine-grained transformations

Specification  Asynchronous communication

Lossless communication over lossy channel  Mutual exclusive access to channel
Fine-grained transformations

Specification

Asynchronous communication

Lossless communication over lossy channel

Mutual exclusive access to channel

Software Engineering and Technology
Fine-grained transformations

Specification → Asynchronous communication

enormous state-space

Lossless communication over lossy channel → Mutual exclusive access to channel
Fine-grained transformations

- Merge objects
- Merge channels
- Unified signals
- Unidirectional channels
- Asynchronous signals

ABP

Software Engineering and Technology
Fine-grained transformations

merge objects → merge channels

unified signals → unidirectional channels

ABP

large state-space

enormous state-space
Conclusions

- Major influences:
  - Problem domain
  - Target platforms
  - Model quality
  - Transformation quality
Conclusions

• Quality of Transformations
  • Fine-grained transformations
    understandability, modifiability and reusability
  • Explicit models lead to simpler transformations
    understandability, modifiability and reusability

• Quality of Models
  • Common concepts from program language design (Scoping)
    understandability

• Target platforms
  • Only introduce the concepts you actually need
    Removing constructs that are too general later on is cumbersome