Pain-mitigation Techniques for Model-based Engineering using Domain-specific Languages

Benny Akesson[1], Jozef Hooman[1,2], Roy Dekker[1,3],
Willemien Ekkelkamp[1,3], and Bas Stottelaar[4]

[1] Embedded Systems Innovation by TNO
[2] Radboud University
[3] Thales Nederland
[4] Altran

Embedded Systems Innovation BY TNO

Embedded Systems Innovation BY TNO

Trends in Complex System Design

- **Increasing system complexity results in**
  - Longer design times
  - Harder to react to changes
- **Changes to system often results in inconsistent artifacts**
  - E.g. simulation models, production code, and documentation

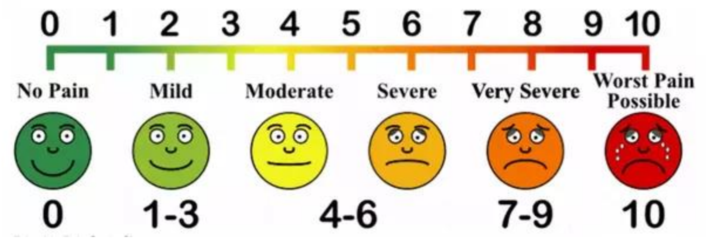
Model-based Engineering using Domain-specific Languages

- **Idea is to reduce design time and improve evolvability using model-based engineering (MBE)**
- **We investigate use of domain-specific languages (DSLs) to specify (parts of) systems**
  - Artifacts are generated from specified DSL instances
- **Supposed benefits:**
  - Allows specification at high level of **abstraction**
  - DSL instance as single source of truth ensures **consistency** among generated artifacts
  - Artifacts can be **quickly regenerated** as system evolves
  - Enables **quick exploration** of components

Problem Statement

- **All design methods come with both pains and gains**

- **Will the pains of the proposed DSL approach offset the gains?**



- **Paper discusses initial steps towards transfer of approach to Thales**
  - We investigate the pains and techniques to mitigate them
  - Results determine if future steps will be taken

- **Current state**
  - Inconsistent simulation models for different frameworks at different levels of abstraction
  - Models often inconsistent with production code

Contributions

The paper has 4 main contributions:

1. List of 14 pains related to MBE from industrial partners
2. Subset of 6 pains positioned with respect to state-of-the-practice
3. Experiences from applying DSL approach to industrial case study and mitigating 6 selected pains
4. List of 3 open issues

Presentation Outline

Introduction

Identification of Pains

Approach

Threat Ranking DSL

Pain-mitigation Techniques

Conclusions

Identified Pains

- **We identified pains relevant to MBE and DSLs based on interactions with partner companies**
  - Inspired by management processes, engineering practices, and experience from senior people

- **The 31 pains have been grouped in 3 main categories:**
  1. Pains related to MBE (14 pains)
  2. Pains related to the introduction of MBE (6 pains)
  3. General pains of the current development process (11 pains)

- **Note that …**
  - the formulation of pains or their classifications are not unambiguous
  - the pains are not laws of nature and may represent unfounded opinions of people critical to MBE
  - the concerns of a partner company needs to be taken seriously either way

Selected Pains

A subset of 6 pains were selected for consideration in this work:

1. No continuity in the development process

7. Difficult to deal with different versions of a component, variability within a component, and different models for one component

8. No consistency between model and realization

10. Incorrect models

12. Code generation leads to low quality code

14. Confusion about the relation between results and versions of component models & tools

Embedded Systems Innovation BY TNO

Presentation Outline

Introduction

Identification of Pains

Approach

Threat Ranking DSL

Pain-mitigation Techniques

Conclusions

Embedded Systems Innovation BY TNO

Context of Case Study

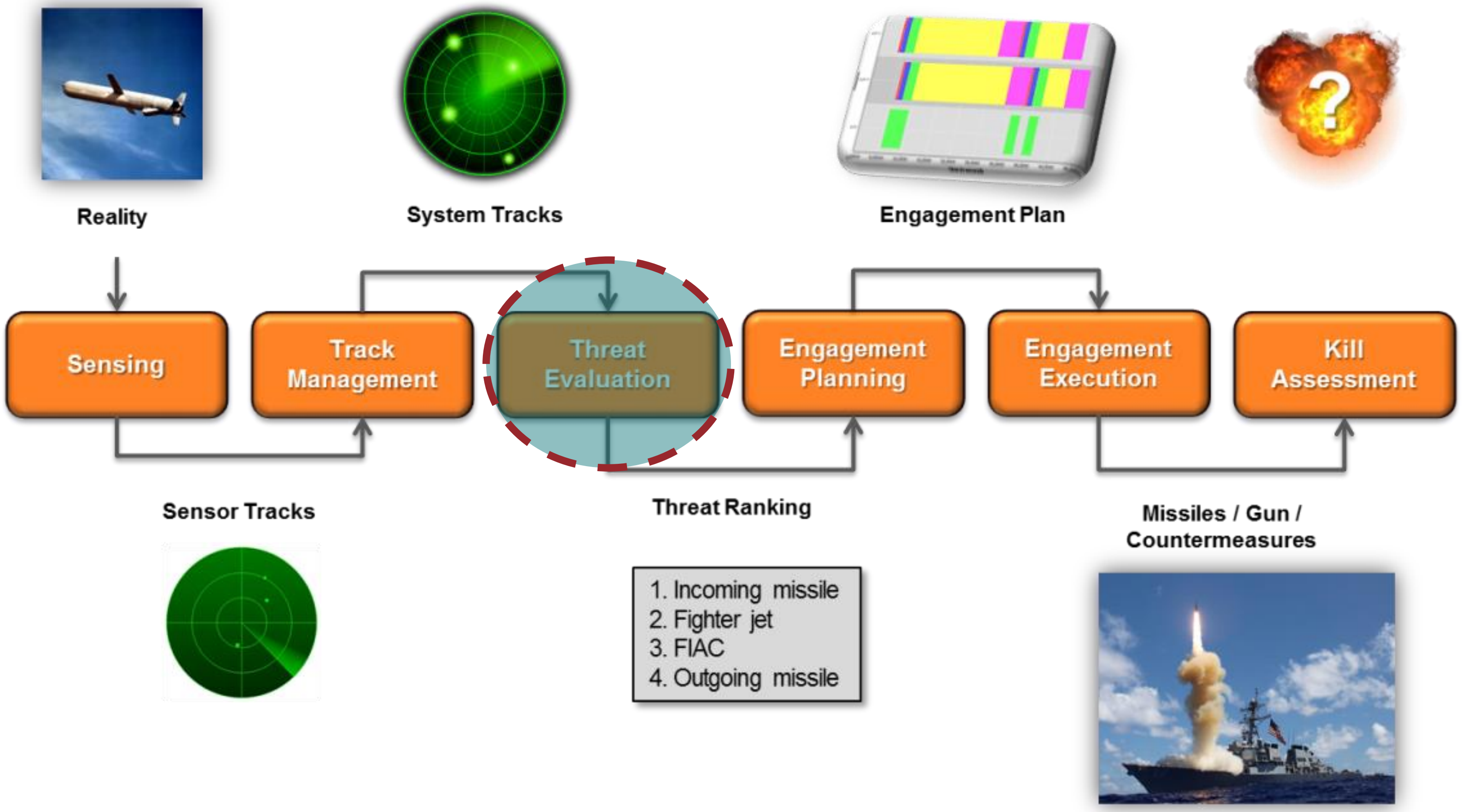
Ship with different capabilities, e.g.

- Surveillance radar
- Tracking radar(s)
- Missile launcher(s)
- Gun(s)

One or more incoming threats, e.g.

- Fast Incoming Attack Craft (FIAC)
- Ballistic/cruise missiles
- Fighter jets

Overview of Engagement Chain



Embedded Systems Innovation BY TNO

Approach of Investigation

- **Xtext is chosen as DSL development tool**
  - Open source framework
  - Previous experience with Xtext both within TNO-ESI and Thales

- **Apply approach to 3 phases of development:**
  1. Design space exploration in Quick Concept Developer (POOSL)
  2. Performance estimation using high-fidelity simulation environment (C++)
  3. Code generation for Combat Management System (choice between **C++**, Ada, Java)

- **Grammars developed in 3 steps to simulate evolution**

Embedded Systems Innovation BY TNO

Presentation Outline

Introduction

Identification of Pains

Approach

Threat Ranking DSL

Pain-mitigation Techniques

Conclusions

Grammar 1: Basic Concepts of Threat Ranking DSL

- **Static threat level per type**
  - None, Low, Moderate, Severe, Critical
- **Dynamic level modifications per threat**
  - Boolean expressions and properties
  - Considers current state of threats
- **Tiebreaker**
  - Breaks ties within threat levels

```
JET assign level SEVERE
MISSILE assign level MODERATE
OTHER assign level NONE

If JET isInbound then INCREASE level
If ANY ownShipDistance < 1 km then assign level CRITICAL

Tiebreaker: timeToOwnShip lowerIsMoreDangerous
```

Embedded Systems Innovation BY TNO

Grammar 2: Custom Metrics and Threat Database

- **Threat database with static information per type**
  - E.g. weapon lethality and keep-out range

- **Custom Metrics**
  - Allows custom tie-breaker metrics to be defined

```
ANY assign level SEVERE

If ANY keepOutRangeViolated then assign level CRITICAL

Weight a = 1.5
Weight b = 0.9
Metric custom = a * keepOutRange + b * lethality
Tiebreaker: custom higherIsMoreDangerous
```

Grammar 3: High-value Units

- **Objective added to DSL**
  - Ranks threats based on own ship, HVU, or both

```
MISSILE assign level CRITICAL
OTHER assign level NONE

Tiebreaker: timeToOwnShip lowerIsMoreDangerous
Objective: protectHVU
```

Embedded Systems Innovation BY TNO

Presentation Outline

Introduction

Identification of Pains

Approach

Threat Ranking DSL

Pain-mitigation Techniques

Conclusions

Pain 7: Dealing with Change

- **Can old instances of the original DSL still be used?**
  - Instance of DSL1 is valid instance of DSL2/3 (new features are optional with default values).
  - We implemented **model-to-model transformations** to support the general case

```
HELICOPTER assign level MODERATE
OTHER assign level NONE
```

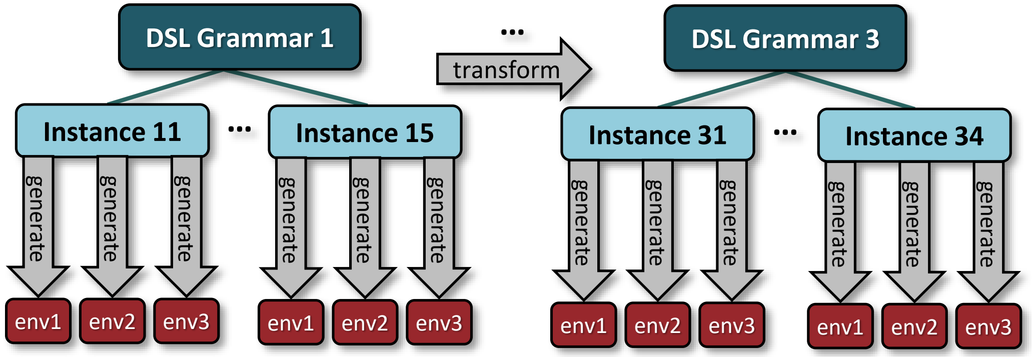
→

```
/* Transformed model from Grammars 1, 2, or 3 now conforming to Grammar 3.
 * Transformer revision: $LastChangedRevision: 1145 $ */

// Static priority assignments for threat types
HELICOPTER assign level MODERATE
OTHER assign level NONE

Tiebreaker: timeToOwnShip lowerIsMoreDangerous
Objective: protectOwnShip
```

Pain 8: Consistency between Model and Realization

- **Simulation models and production code are generated from the DSL instance**
  - POOSL generator for Quick Concept Developer (env1)
  - C++ generator for high-fidelity simulation (env2) and production code (env3)

- **Both simulation models and production code are hence consistent with DSL instance**



Pain 10: Model Quality

- **Validation of algorithm at model level (validation rules)**

```
MISSILE assign level SEVERE
MISSILE assign level LOW
OTHER assign level LOW

If MISSILE ownShipDistance < 100 s then INCREASE level

Tiebreaker: speed higherIsMoreDangerous
```

- **Get insight into ranking through static analysis of tiebreaker metric**

Analysis of custom metric:

Weights: smallNumber := 0.000001
Expression: timeToOwnShip * timeToKOR + keepOutRangeViolated * smallNumber / speed

Ranking by custom metric
(lower is more dangerous):

1) [1.37] 5-MISSILE
2) [2.07] 3-MISSILE
3) [2.08] 1-MISSILE
4) [2.29] 4-MISSILE
5) [2.56] 2-MISSILE

Example: 5-MISSILE

Parameters:
- CPADistance : 48.30 m
- altitude : 19.86 m
- speed : 799.93 m/s
- timeToKOR : 22.82 s
- timeToOwnShip : 0.06 s

Substituted: 0.06 * 22.82 + 0.0 * 0.000001 / 799.93

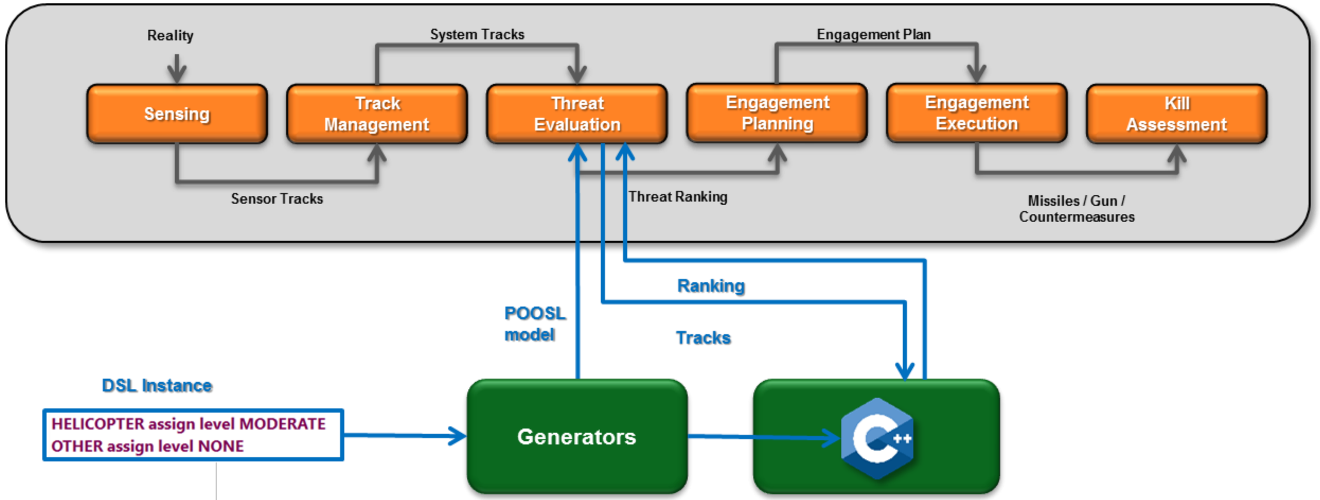
Evaluated: 1.37

Pain 14: Tracking Results and Versions

- **Source code, DSL grammars and instances managed by Subversion**
- **Generated artifacts annotated with version number of generator**
- **After a simulation, we store:**
  - Scenario
  - Configuration information (e.g. ship parameters)
  - Threat Ranking DSL instance
  - Simulation results
  - Version numbers of simulators and other tools
- **This enables tracing and makes deterministic results reproducible**

Pain 12: Quality of Generated Code

- **Ensuring correctness of results across environments is challenging**
  - Results from different simulation/execution environments will be different
- **We use generated C++ as software-in-the-loop to homogenize environment**
  - Results from both implementations should now be **identical**
  - **Automatically tested** by Jenkins server for many scenarios after each commit

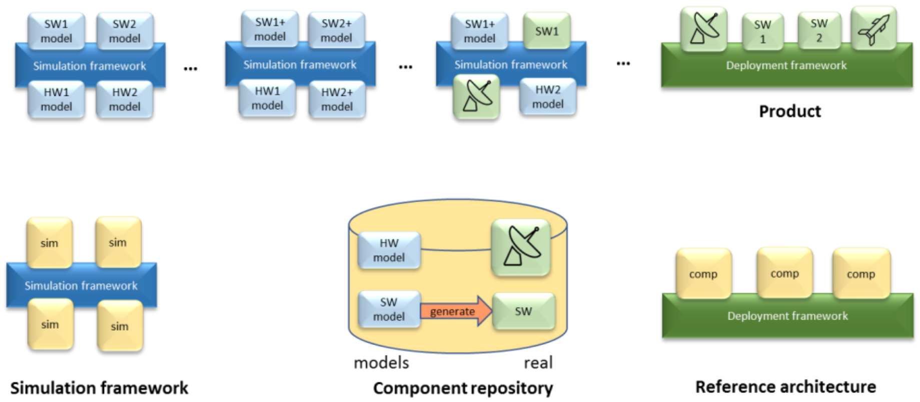


Open Issues

1. **Ensuring semantic consistency of generators by construction**
    - Eliminate problem of manually ensuring semantic consistency across generators

2. **Validation of implementations at different levels of abstraction**
    - Equivalence testing only applies when the same output is expected

3. **Techniques to develop a single framework that can be used throughput development chain**
    - A single model is incrementally refined and used in all stages of design
    - Avoid differences requiring adaptors and wrappers



Presentation Outline

Introduction

Identification of Pains

Approach

Threat Ranking DSL

Pain-mitigation Techniques

Conclusions

Conclusions

- **Paper presents first steps towards transfer of DSL approach to Thales**
  - Goal is to **reduce design time** and **improve evolvability** of system
  - Means to achieve this is to **generate consistent simulation models and code** from DSL instances
- **Problem was to identify the pains related to the approach and propose mitigation techniques**
  - 14 pains related to MBE and DSLs were identified
  - 6 of these were investigated through case study of Threat Ranking component
- **Based on this work, it has been decided to continue the investigation**
  - Scale up approach to a more complex component
  - Further explore identified pains and open issues