From Worlds to Machines

Axel van Lamsweerde
University of Louvain (Belgium)
avl@info.ucl.ac.be

ICSE'09, Vancouver, May 2009

A very personal tribute first ...

- 1979: Research on concurrency at Philips Labs
  Fixpoint calculation of safe invariants
  \[ D = D \land I \land \lor_i \text{wp} (S_i, D) \]
- 1980: Teaching COBOL at University of Namur
  ```
  PROGRAM-ID. MYSTIC.
  DATA DIVISION.
  01 PRINT-LINE.
  02 NUM OCCURS 10 PIC ZZZ9 SYNSCHRONIZED.
  PROCEDURE DIVISION.
  PMAIN.
  PERFORM PLINE VARYING LN FROM 1 BY 1 UNTIL LN > 50.
  STOP RUN.
  PLINE.
  MOVE SPACES TO PRINT-LINE.
  PERFORM PROCESS-ELEMENT UNTIL EOF.
  ```
JSP saved me from intellectual trauma

- Yes, we can... (make ugly things elegantly)
- A real *method* for structured programming
- Deep thoughts on resolution of structure clashes
- And ... COBOL as a target language!

### From programs to specs to requirements

**World**

MOVE SPACES TO OUTPUT.
PERFORM PROCESS-EVENT UNTIL TIMEOUT.

**Machine**
This talk

MOVE SPACES TO OUTPUT. PERFORM PROCESS-EVENT UNTIL TIMEOUT.

Problem world vs. machine solution

- **World**: problematic part of real-world
  - Organizational & physical components

- **Machine**: abstraction for what needs to be developed to solve the problem

- Requirements engineering (RE) is concerned with ...
  - the desired machine’s effect on the problem world
  - the relevant properties of this world
Outline

- Anchoring the Machine on the Problem World
- Characterizing the Problem World
- Delimiting and structuring the Problem World
- Chaining satisfaction arguments
- Deriving specifications from requirements
- Questioning statements
- Reusing problem schemas

Anchoring the Machine on the Problem World

- A problem: handbrake release can be inconvenient
- The world and the machine have their own phenomena
Anchoring the Machine on the Problem World

Anchoring through shared phenomena ...
- monitored or controlled: by the world or by the machine
- define the world-machine boundary

Requirements:
constrain world phenomena
MotorRaising → HandbrakeReleased

Specifications:
constrain shared phenomena
motor.Regime = 'up' → handBrakeCtrl = 'off'
Impact

- Clarification of real nature of requirements
  - long-standing confusion: reqs vs. specs,
    stakeholder vocabulary vs. developer vocabulary

- Machine as a refinable abstraction
  - may include software and I/O devices first,
    then only the software to be developed

- Led us to...
  - a realizability criterion for responsibility assignment
  - explicit agent interfaces for inductive inference of
    goals from scenarios

W & M revisited:
the world-as-is and the world-to-be

Domain analysis and requirements elicitation
involve two versions of the world

Concepts, phenomena, rules
about car handbraking

Concepts, phenomena, rules
about automated handbraking
W & M revisited: alternative machine solutions

- Requirements evaluation involves multiple options
- Alternative options yield different shared phenomena

**Option 1**
- `motor.Regime = 'up'`
- `MotorRaising → HandbrakeReleased`

**Option 2**
- `brake.Switch = 'on'`
- `BrakeButtonPressed → HandbrakeReleased`

*The World-Machine boundary is not fixed when RE starts*

---

W & M revisited: world variations and evolutions

- Product lines involve multiple world-machine *variants*
- Requirements prioritization & evolution management involve world/machine *to-be-next*

- Handbraking in Class *E* car
- Handbraking in Class *S* car

- Class *E* handbraking, 2009 model
- Class *E* handbraking, 2011 model
Outline

- Anchoring the Machine on the Problem World
- Characterizing the Problem World
- Delimiting and structuring the Problem World
- Chaining satisfaction arguments
- Deriving specifications from requirements
- Questioning statements
- Reusing problem schemas

Characterizing the Problem World

- Typology of statements about the world
  - **Descriptive**: natural laws, physical constraints, etc
    A car’s motor regime is raising when the air conditioner starts
  - **Prescriptive**: desired, to be enforced
    Handbrake shall be released when the car’s motor regime is raising
  - **Definition**: for concepts/terms, no truth value
    A car’s motor regime is said to be raising if it increases by X above neutral level
- For formalized statements:
  - **Designation**: meaning of atomic predicate/terms, in terms of world objects and phenomena
**Impact**

- Important clarification: requirements are prescriptive, domain properties are descriptive
  - We can negotiate, weaken, find alternatives to prescriptive statements only
- Definitions & designations are essential for precision and conciseness
- Led us to use descriptive properties for...
  - proving goal refinements & operationalizations
  - deriving obstacles & conditions for conflict

---

**Statement typology revisited:**

**assumptions as first-class citizens**

- Assumptions on problem world are frequently made in RE
- More volatile, more subject to adequacy checking
- Play important role in satisfaction arguments, risk analysis, option selection, traceability management
- Some are prescriptive, others are descriptive
  - The driver shall press the acceleration pedal if she wants to start (expectation)
  - Handbrakes are never used under -40°C (domain hypothesis)
Statement typology revisited

Statement
  
  Prescriptive
  
  Requirement
  
  Specification
  
  Descriptive
  
  Expectation
  
  Domain property
  
  Definition
  
  Domain hypothesis

Expectations on people are often documented in user manuals

Delimiting & structuring the Problem World

- **Scope**: limited to subject matter
- **Structure**: problem diagram showing components, interfaces, requirements

Handbrake Controller

HC! handbrake.Sw

CI motor.Regime

DR! \{pedalPushed, buttonPressed\}

{pedalPushed, buttonPressed}

Driver

Car

{BrakeActivation, BrakeRelease}

Handbrake shall be activated if the brake button is pressed, released if the acceleration pedal is pushed
Issues and perspectives

- Bounding the problem world is sometimes hard
  - Analysis of feature interactions => components beyond subject matter
    e.g. handbrake control + air conditioning control?
  - Security threat analysis => malicious components
    e.g. handbrake release by car robber?
  - Open systems => what foreign components?

- Further structuring of problem diagrams might be required for scaleability
  - Decomposition, specialization, refinement, ...
    + proof obligations

Outline

- Anchoring the Machine on the Problem World
- Characterizing the Problem World
- Delimiting and structuring the Problem World
- Chaining satisfaction arguments
- Deriving specifications from requirements
- Questioning statements
- Reusing problem schemas
Satisfaction arguments

- Need to show that the requirements will be met if the specs are met in view of domain properties and assumptions:

\[
\{\text{SPEC, ASM, DOM}\} \models \text{Req}
\]

**SPEC:**  \text{motor.Regime = 'up' } \rightarrow \text{handBrakeCtrl = 'off'}

**ASM:**  \text{motor.Regime = 'up'} \leftrightarrow \text{MotorRaising}
\text{handBrakeCtrl = 'off'} \leftrightarrow \text{HandbrakeReleased}

\[\text{-----------------------------------------------}\]

**Req:**  \text{MotorRaising } \rightarrow \text{HandbrakeReleased}

Impact

- Important role of satisfaction arguments
  - Verifying spec correctness wrt requirements
  - Making required domain properties & assumptions explicit (and questionable)
  - Managing traceability: if assumption no longer valid, specs linked to it by an argument must change accordingly
Chaining satisfaction arguments into argumentation trees

To show how specifications ensure higher-level concerns, and recursively

\[ \text{MotorRaising} \rightarrow \text{HandBrakeReleased} \]

\[ \text{motor.Regime} = 'up' \rightarrow \text{handBrakeCtrl} = 'off' \]

\[ \frac{\land}{\therefore} \text{motor.Regime} = 'up' \leftrightarrow \text{MotorRaising} \]

\[ \frac{\lor}{\therefore} \text{handBrakeCtrl} = 'off' \leftrightarrow \text{HandBrakeReleased} \]

spec

Chaining satisfaction arguments into argumentation trees

To show how specifications ensure higher-level concerns, and recursively

\[ \text{HandBrakeReleased} \leftrightarrow \text{MotorRaising} \]

\[ \text{HandBrakeReleased} \rightarrow \text{MotorRaising} \]

\[ \frac{\land}{\therefore} \text{MotorRaising} \rightarrow \text{HandBrakeReleased} \]

\[ \frac{\lor}{\therefore} \text{motor.Regime} = 'up' \rightarrow \text{handBrakeCtrl} = 'off' \]

\[ \frac{\land}{\therefore} \text{motor.Regime} = 'up' \leftrightarrow \text{MotorRaising} \]

\[ \frac{\lor}{\therefore} \text{handBrakeCtrl} = 'off' \leftrightarrow \text{HandBrakeReleased} \]

spec
Chaining satisfaction arguments into argumentation trees

To show how specifications ensure higher-level concerns, and recursively

\[
\begin{align*}
\text{HandBrakeReleased} & \leftrightarrow \text{DriverWantsToStart} \\
\text{HandBrakeReleased} & \leftrightarrow \text{MotorRaising} \\
\text{HandBrakeReleased} & \rightarrow \text{MotorRaising} \\
\text{MotorRaising} & \rightarrow \text{HandBrakeReleased} \\
\text{HandBrakeReleased} & \rightarrow \text{MotorRaising} \\
\text{HandBrakeReleased} & \leftrightarrow \text{MotorRaising} \\
\text{MotorRaising} & \leftrightarrow \text{HandBrakeReleased} \\
\end{align*}
\]

Deriving specifications from requirements

- General idea: incrementally replace non-shared phenomena in requirements by shared “images”
  - using domain properties, assumptions, satisfaction args
  - e.g. \( \text{HandbrakeReleased} \rightarrow \text{handBrakeCtrl = 'off'} \)
- Cf. turnstile control example in [ICSE’95]
- More difficult if requirements language and specification language are not the same
- Can this be made systematic?
One solution: pattern-based derivation

- Patterns encode common refinement tactics
- Proved once for all

\[
\text{ReqOnUnMonitorableCondition} \quad \Rightarrow \quad \text{MonitorableCondition} \leftrightarrow \text{UnmonitorableCondition}
\]

Instantiation

\[
\text{MotorRaising} \rightarrow \text{HandBrakeReleased}
\]

\[
\text{motor.Regime} = 'up' \rightarrow \text{HandBrakeReleased}
\]

Pattern-based derivation (2)

\[
\text{ReqOnUnControllableCondition} \quad \Rightarrow \quad \text{ControllableCondition} \leftrightarrow \text{UncontrollableCondition}
\]

Instantiation

\[
\text{motor.Regime} = 'up' \rightarrow \text{HandBrakeReleased}
\]

\[
\text{handBrakeCtrl} = 'off' \leftrightarrow \text{HandBrakeReleased}
\]

Spec
Questioning statements

- Critical domain properties & assumptions used in satisfaction args must be checked for adequacy
- Cf. A320 braking logic example in ...
- More difficult for probabilistic statements
  \( S \) holds in \( \chi \% \) of cases
- Can this be made systematic?

One solution: obstacle analysis

- Obstacle = condition for statement obstruction
  \[ \{ O, \text{Dom}\} \models \neg S \quad \text{obstruction} \]
  \[ \text{Dom} \models \neg O \quad \text{domain consistency} \]
- Obstacle analysis:
  1. **Identify** obstacles as preconditions for statement negation in view of domain properties
     \( \rightarrow \) formal calculus, obstruction patterns
  2. **Assess** their likelihood and severity
  3. **Resolve** likely/critical ones (using available tactics)
Questioning statements by obstacle analysis

BrakeReleased $\leftrightarrow$ DriverWantsToStart

BrakeReleased $\leftrightarrow$ MotorRaising

MotorRaising $\leftrightarrow$ AccelerPedalPressed

AccelerPedalPressed $\leftrightarrow$ DriverWantsToStart

MotorRaising $\leftrightarrow$ AccelerPedalPressed And Not DriverWantsToStart

AccelerPedalPressed And Not DriverWantsToStart

...
Questioning statements by obstacle analysis

BrakeReleased $\leftrightarrow$ DriverWantsToStart

BrakeReleased $\leftrightarrow$ MotorRaising

MotorRaising $\leftrightarrow$ AccelerPedalPressed

MotorRaising And Not AccelerPedalPressed

AccelerPedalPressed And Not DriverWantsToStart

BrakeReleased $\leftrightarrow$ DriverWantsToStart

Driver killed by his luxurious car on a hot summer day

Outline

- Anchoring the Machine on the Problem World
- Characterizing the Problem World
- Delimiting and structuring the Problem World
- Chaining satisfaction arguments
- Deriving specifications from requirements
- Questioning statements
- Reusing problem schemas
Reusing problem schemas

- Candidate world structures, reqs & specs can be derived by instantiating problem patterns captured by frame diagrams

Commanded behavior frame

Reusing problem schemas

- Candidate world structures, reqs & specs can be derived by instantiating problem patterns captured by frame diagrams

Instantiated frame diagram
**Problem reuse: challenges**

- Schema descriptions should be sufficiently rich to enable reuse beyond structural information.
- Reusable schemas should be sufficiently specific to enable transfer of useful discriminating features.
- Perfect match is unfrequent.
  => support for validation & adaptation of instantiated schemas?
- Problem worlds generally combine multiple types of problem.
  => mechanisms for composing schemas and instantiations?

**Conclusion**

- Major contributions to a reference model for RE.
- Significant clarification of ...
  - the complex relationship between problem worlds and machine solutions.
  - the role of domain properties and satisfaction arguments.
- Increasing impact on RE research, education, practice.
- And... such elegance in thinking and writing...

*Thanks, Michael!*