Safety-Critical Real-time Embedded Software Development

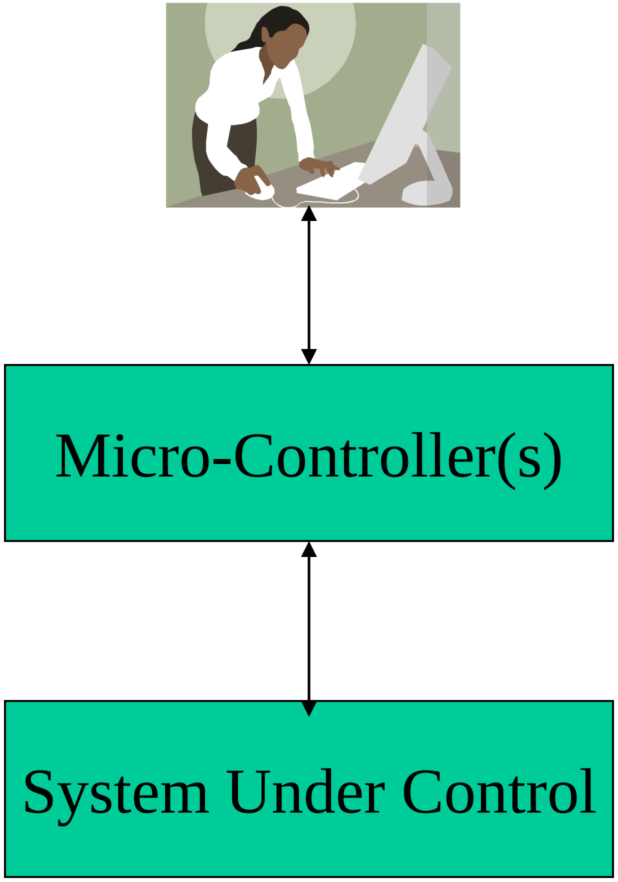
Thiago Carvalho de Sousa
Alstom Transport Brazil – R&D Manager
University of São Paulo – Phd Candidate
June 15, 2011

Introduction

Safety-Critical systems are those systems whose failure could result in loss of life, cause significant property damage or cause damage to the environment. These complex systems tend to have sufficient kinetic or potential energy which can become uncontrollable and thus pose a hazardous condition. Therefore, these systems must be designed in such a way as to guarantee system stability during all of the system operational modes. Furthermore, when a fatal fault occurs, the system safely shuts down.

Definition of Real-time Embedded System

Real-time Embedded System in its simplest form is depicted below:



Definition of Critical Applications

- Computer based systems used in avionics, chemical process, transport and nuclear power plants.
- A failure in the system endangers human lives directly or through environment pollution and Influence is on a large economic scale.

Definition

- **Safety:**

  Safety is a property of a system that it will not endanger human life or the environment.

- **Safety-Critical System:**

  A system that is intended to achieve, on its own, the necessary level of safety integrity for the implementation of the required safety functions.

Developing Safety-Critical Systems

- **To achieve the safety objective:**
  - well-defined system safety requirements (hazards & risks analyzed)
  - quality management (auditing process)
  - design / system architecture (reliability analysis)
  - defined design/manufacture processes
  - certification and approval processes
  - known behaviour of the system in all conditions

Software Development

- **To achieve the safety objective:**
  - Safety requirements which address all system specifications
  - Quality Control Processes for Validation & Verification
  - Software Design Description
  - Certification and approval (according to a guideline)
  - Extensive software development testing (functional and code coverage)
  - Extensive system integration testing (control laws, software and hardware)
  - Complete set of documentation which supports the software development life cycle.

The Need For Certification

As the Embedded systems began to be used for the consumer market, several certification standards for different industries were developed:

IEC 880 - Nuclear Safety - 1986
IEC 601 - Medical Safety – 1996
CENELEC EN 50128 – Railway Safety - 2001
MISRA - Motor Industry Safety – UK 1994
IEC 61508 - Programmable Electronic Safety – Geneva 1998
RTCA DO-178B – Airborne Systems Safety - 1992

Risk Analysis

- **Severity :** - Catastrophic – multiple deaths >10
  - Critical – a death or severe injuries
  - Marginal – a severe injury
  - Insignificant – a minor injury

- **Frequency Categories:**

Frequent	0,1 events/year
Probable	0,01
Occasional	0,001
Remote	0,0001
Improbable	0,00001
Incredible	0,000001

	Consequence			
Likelihood	Catastrophic	Critical	Marginal	Negligible
Frequent	I	I	I	II
Probable	I	I	II	III
Occasional	I	II	III	III
Remote	II	III	III	IV
Improbable	III	III	IV	IV
Incredible	IV	IV	IV	IV

Risk acceptability

Tolerable Hazard Rate (THR) – A hazard rate which guarantees that the resulting risk does not exceed a target individual risk.

SIL 4 = $10^{-9} < THR < 10^{-8}$

SIL 3 = $10^{-8} < THR < 10^{-7}$

SIL 2 = $10^{-7} < THR < 10^{-6}$

SIL 1 = $10^{-6} < THR < 10^{-5}$

SIL = Safety Integrity Level

Safety-Critical Software Specification

Technique	SIL 1	SIL 2	SIL 3	SIL 4
Structured Methodology	HR	HR	HR	HR
Computer-aided Tools	R	R	HR	HR
Semi-Formal Methods	R	R	HR	HR
Formal Methods	NC	R	R	HR

HR = Highly Recommended; R = Recommend;
NR = No Recommendation; NC = No Comment

Safety-Critical Software Design

Technique	SIL 1	SIL 2	SIL 3	SIL 4
Fault detection & diagnosis	NR	R	HR	HR
Error detecting codes	R	R	R	HR
Programming with assertions	R	R	R	HR
Diverse programming	R	R	R	HR
Recovery blocks	R	R	R	R

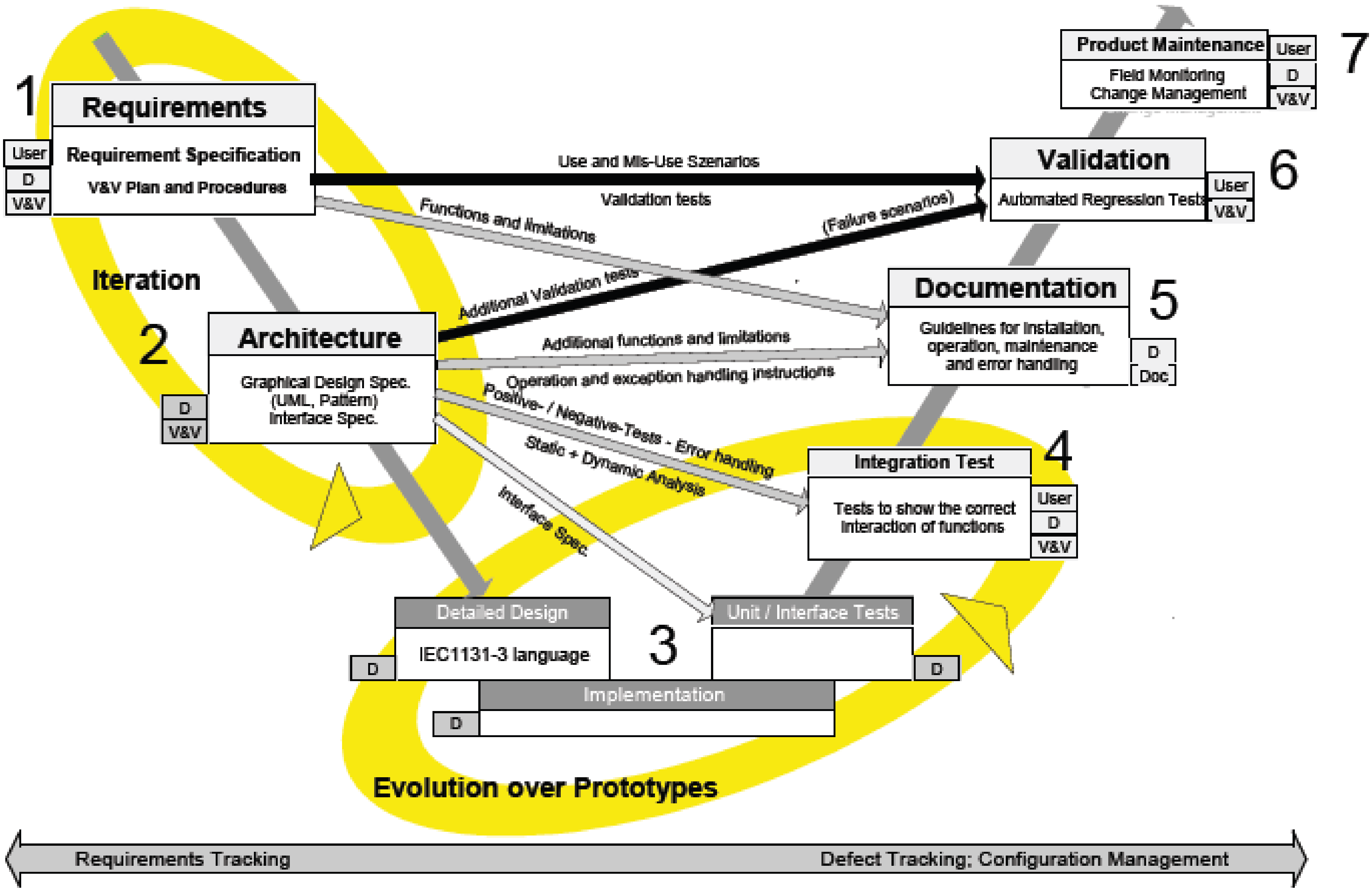
Safety-Critical Software Implementation

Technique	SIL 1	SIL 2	SIL 3	SIL 4
Modular approach	NR	R	HR	HR
Defence programming	NC	R	HR	HR
Code standards	R	HR	HR	HR
Analysable programs	R	HR	HR	HR
Suitable programming language	HR	HR	HR	HR
Language subset	NC	NC	HR	HR
Certified translator	R	HR	HR	HR
Verified library modules	R	HR	HR	HR

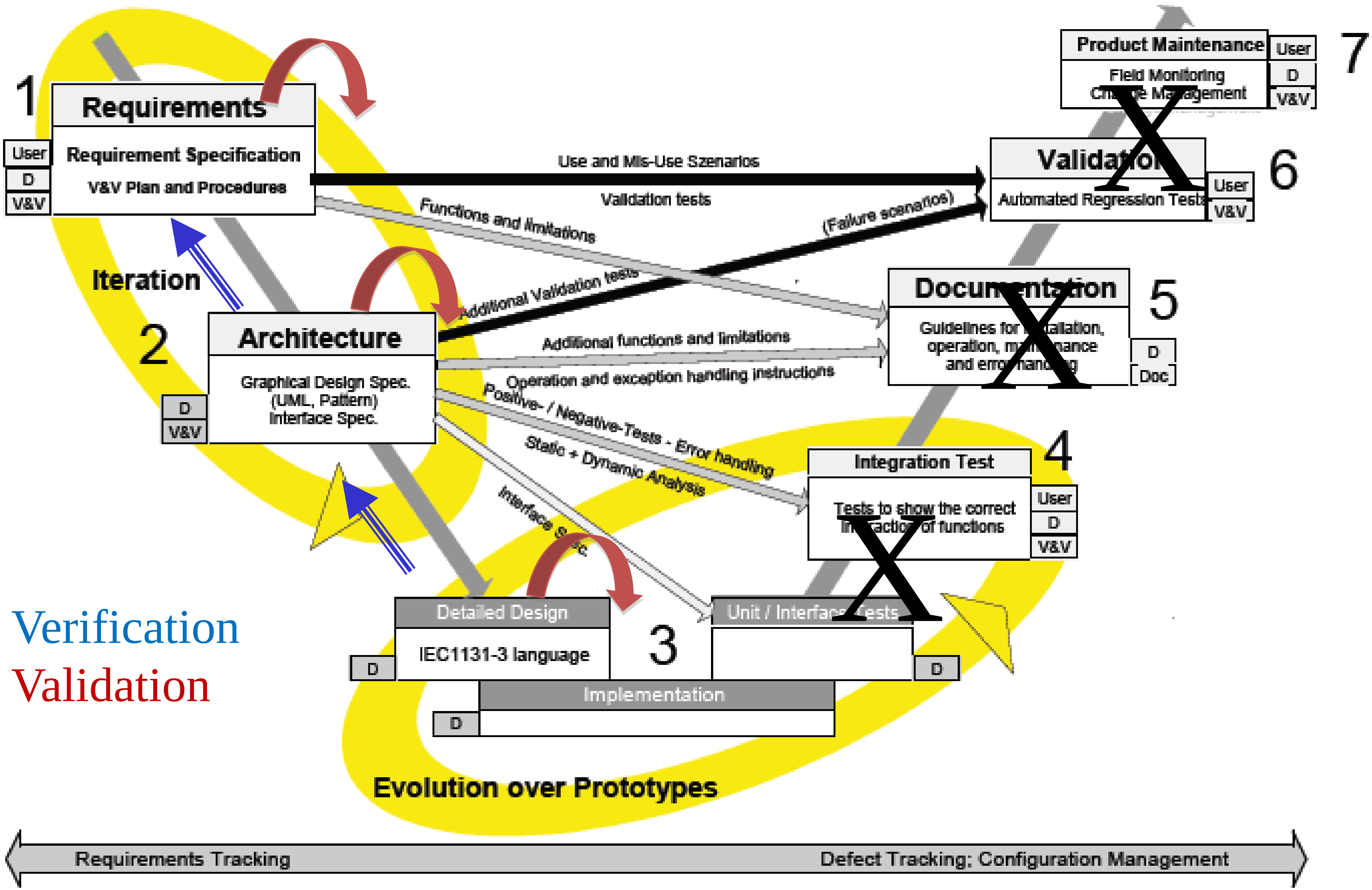
Safety-Critical Software Verification

Technique	SIL 1	SIL 2	SIL 3	SIL 4
Formal Proof	NC	R	R	HR
Probabilistic testing	NC	R	R	HR
Static analysis	R	HR	HR	HR
Dynamic analysis & testing	R	HR	HR	HR
Software complexity	R	R	R	R

V - Lifecycle model for SIL3



V - Lifecycle model for SIL4



Verification

Validation

Event-B

- State-transition model (like ASM, B, VDM, Z)
  - set theory as mathematical language
- Refinement (based on action systems by Back)
  - data refinement
  - one-to-many event refinement
  - new events (stuttering steps)
- Proof method
  - Refinement proof obligations (POs) generated from models
  - Automated and interactive provers for POs

Access Control System

- Users are authorised to engage in activities
- User authorisation may be added or revoked
- Activities take place in rooms
- Users gain access to a room using a one-time token provided they have authority to engage in the room activities
- Tokens are issued by a central authority
- Tokens are time stamped
- A room gateway allows access with a token provided the token is valid

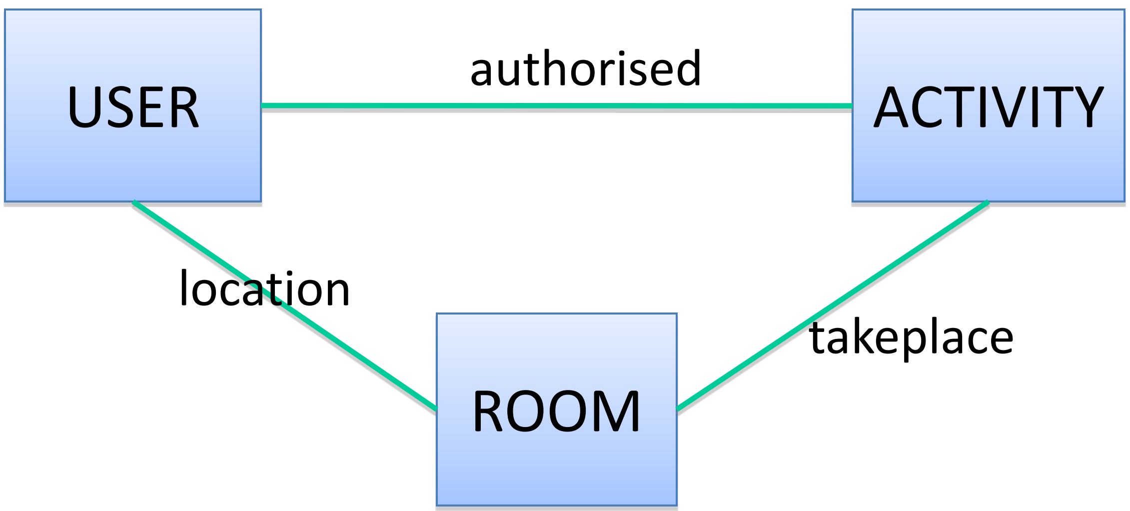
Extracting the essence

- Access Control Policy: *Users may be in a room only if they are authorised to engage in all activities that may take place in that room*

- To express this we only require Users, Rooms, Activities and relationships between them

- Abstraction: focus on key entities in the problem domain

Diagrammatic Representation



Variables and Invariants

Variables of Event-B model

@inv1 $authorised \in User \leftrightarrow Activity$ // relation

@inv2 $takeplace \in Room \leftrightarrow Activity$ // relation

@inv3 $location \in User \pfun Room$ // partial function

Access control *invariant*:

if user *u* is in room *r*,

then *u* must be authorised to engaged in all activities that can take place in r

@inv4 $\forall u,r .\ u \in dom(location) \land location(u) = r \Rightarrow takeplace[r] \subseteq authorised[u]$

State snapshots as tables

User	Activity
u1	a1
u1	a2
u2	a2

authorised

Room	Activity
r1	a1
r1	a2
r2	a1

takeplace

User	Room
u1	r1
u2	r2
u3	

location

Event for entering a room

Enter $\triangleq$

when

grd1 : $u \in User$

grd2 : $r \in Room$

grd3 : $takeplace[\, r \,] \subseteq authorised[\, u \,]$

then

act1 : $location(u) := r$

end

Does this event maintain the security invariant?

Role of invariants and guards

- Invariants: specify properties of model variables that should also remain true
  - violation of invariant is undesirable
  - use (automated) proof to verify invariant preservation
- Guards: specify conditions under which events may occur
  - should be strong enough to ensure invariants are maintained
  - but not so strong that they prevent desirable behaviour

Remove Authorisation

RemoveAuth(u,a) $\triangleq$

when

grd1 : $u \in User$

grd2 : $a \in Activity$

grd3 : $u \mapsto a \in authorised$

then

act1 : $authorised := authorised \setminus \{ u \mapsto a \}$

end

Does this event maintain the security invariant?