I-Mathic

Formal Methods in Software Engineering Practice

Emile van Gerwen
Emile.vanGerwen@imtech.nl

Overview

1. Why I-Mathic
2. Sequence Enumeration
3. Verification
4. Case
5. Code Generation
6. CSP
7. Future developments

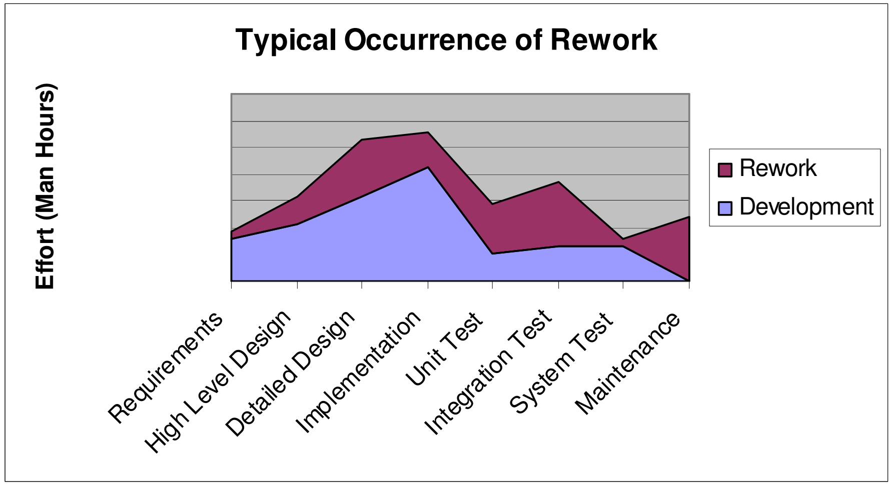
Why I-Mathic

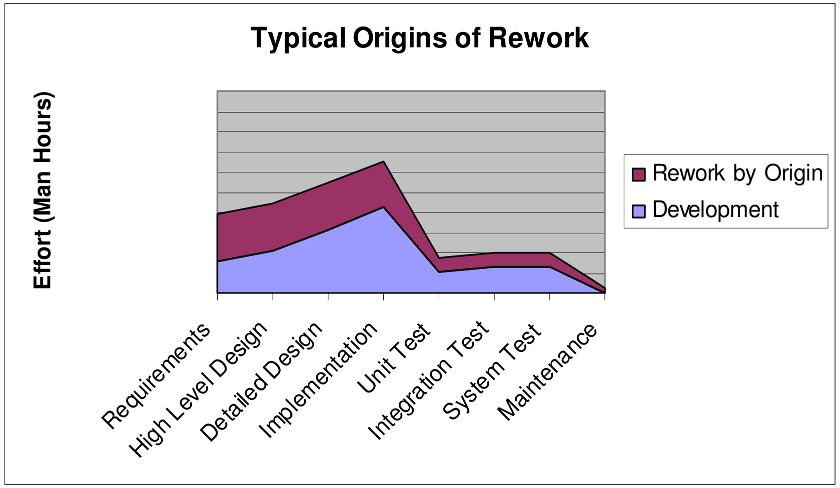
- We see
  - Distributed applications
  - Integration of complex units into even more complex units
- Testing all scenario’s is impossible
- Any alternative must be cost effective (formal methods?)

Formal Methods

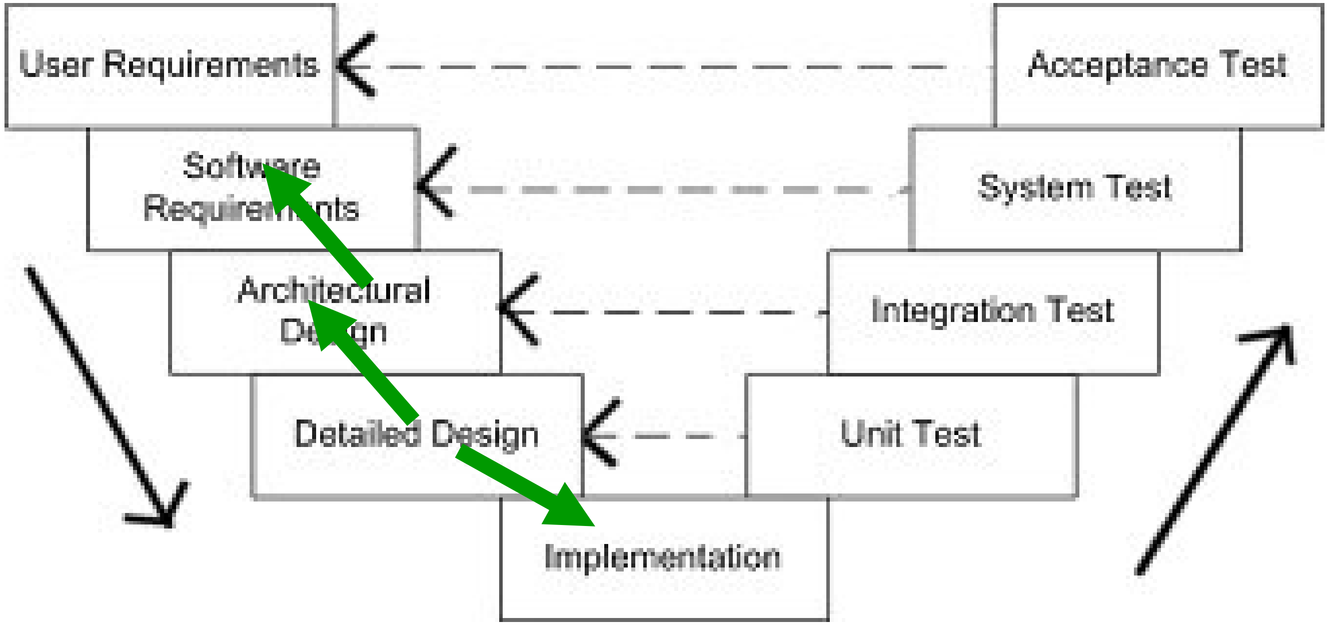
- Formal Methods have promised much and delivered little:
  - The solution is often more complicated than the problem
  - Formal specifications use difficult notations and require extensive mathematical background
  - Critical Stakeholders - Business Analysts, Domain Experts and Customers - cannot understand the formal specifications

What everybody knows

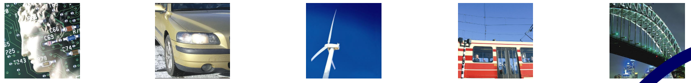


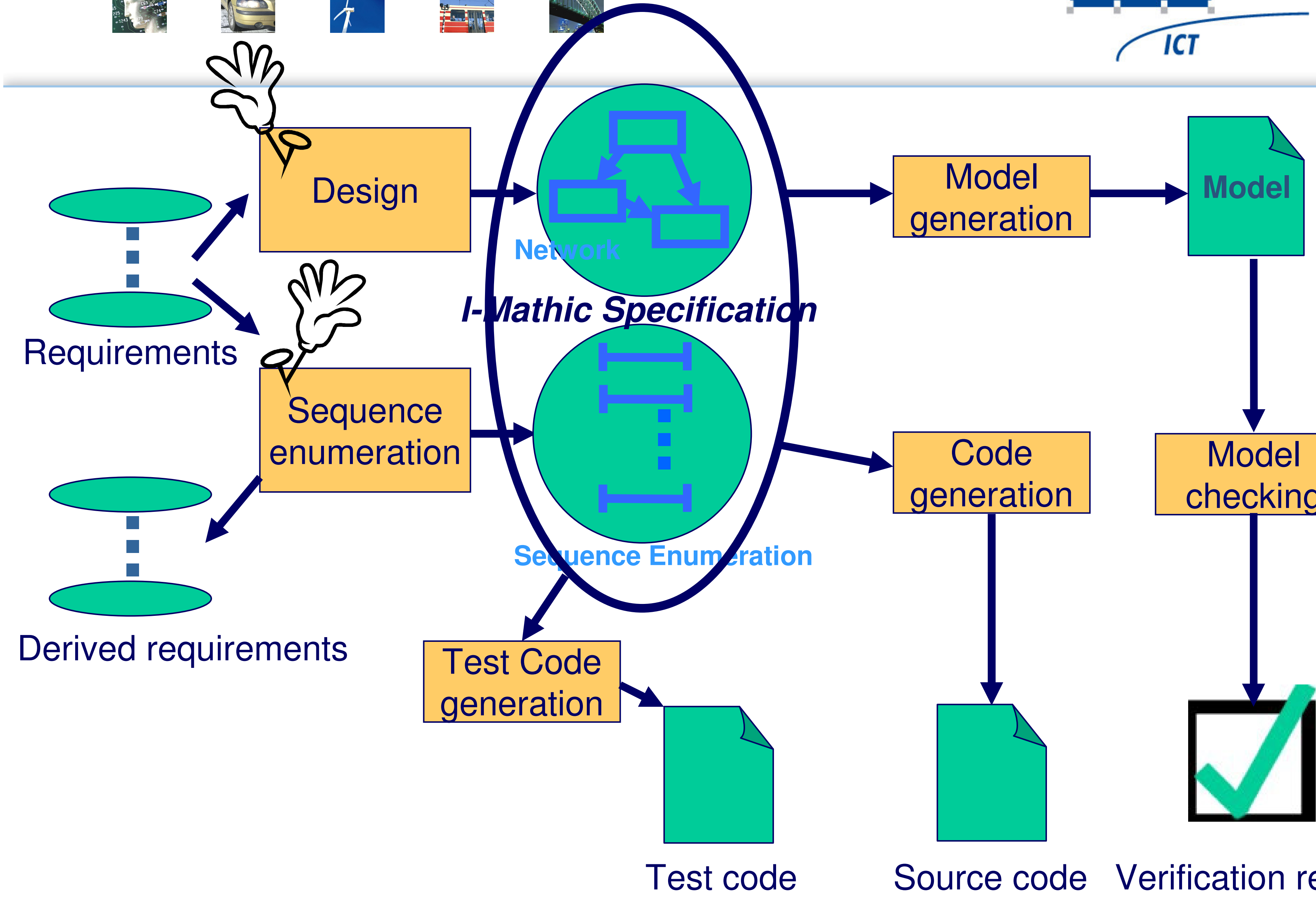


V-Model

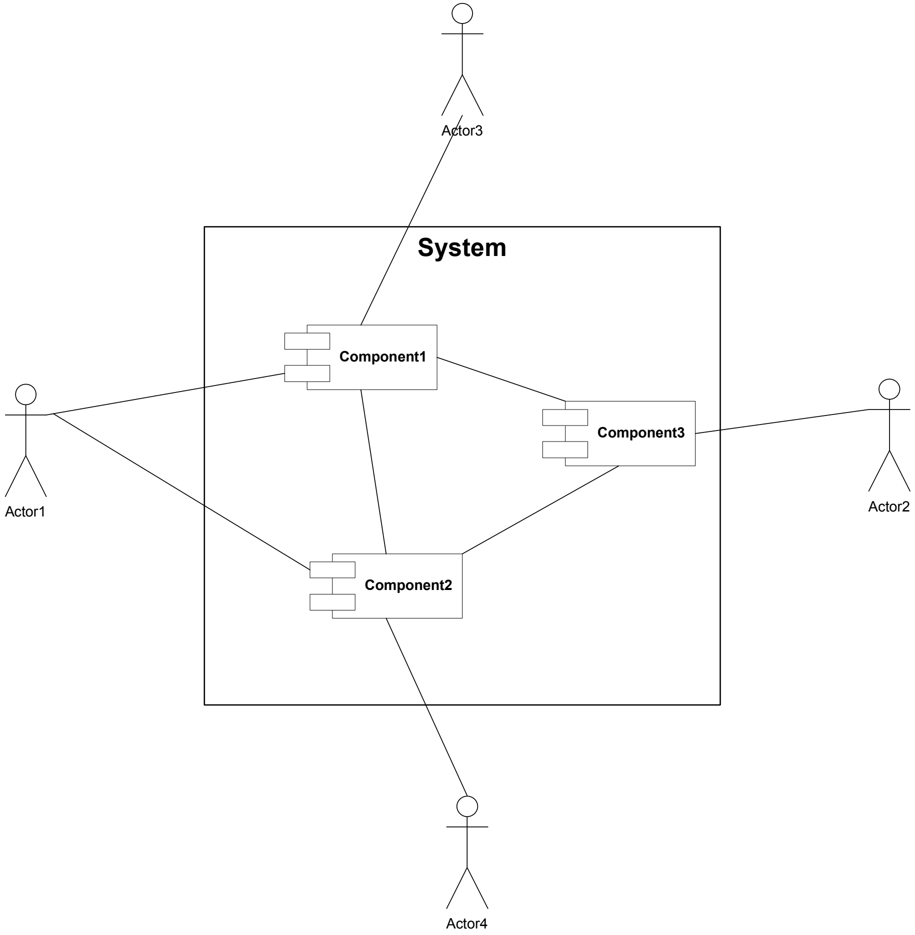


I-Mathic: fix design errors in design phase



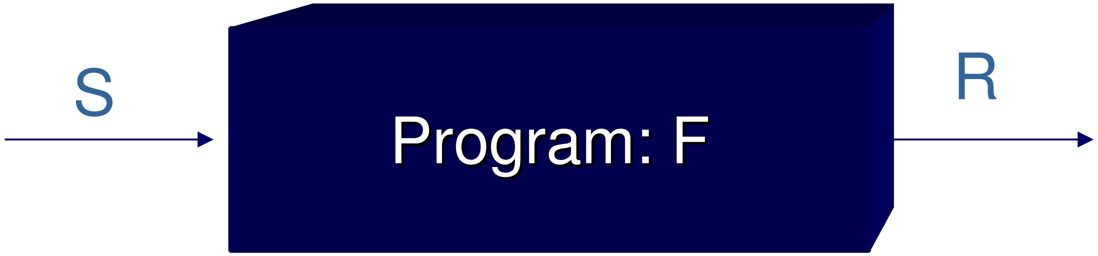

Design
Requirements
Sequence enumeration
Derived requirements
Network
I-Mathic Specification
Sequence Enumeration
Test Code generation
Test code
Model generation
Model
Code generation
Model checking
Source code
Verification result

I-Mathic view of the world



Black Box

What is a software system?



F: S^+ -> R or F: (S^*, S) -> R

Where,

S is a finite set of Stimuli (input)
R is a finite set of Responses (output)

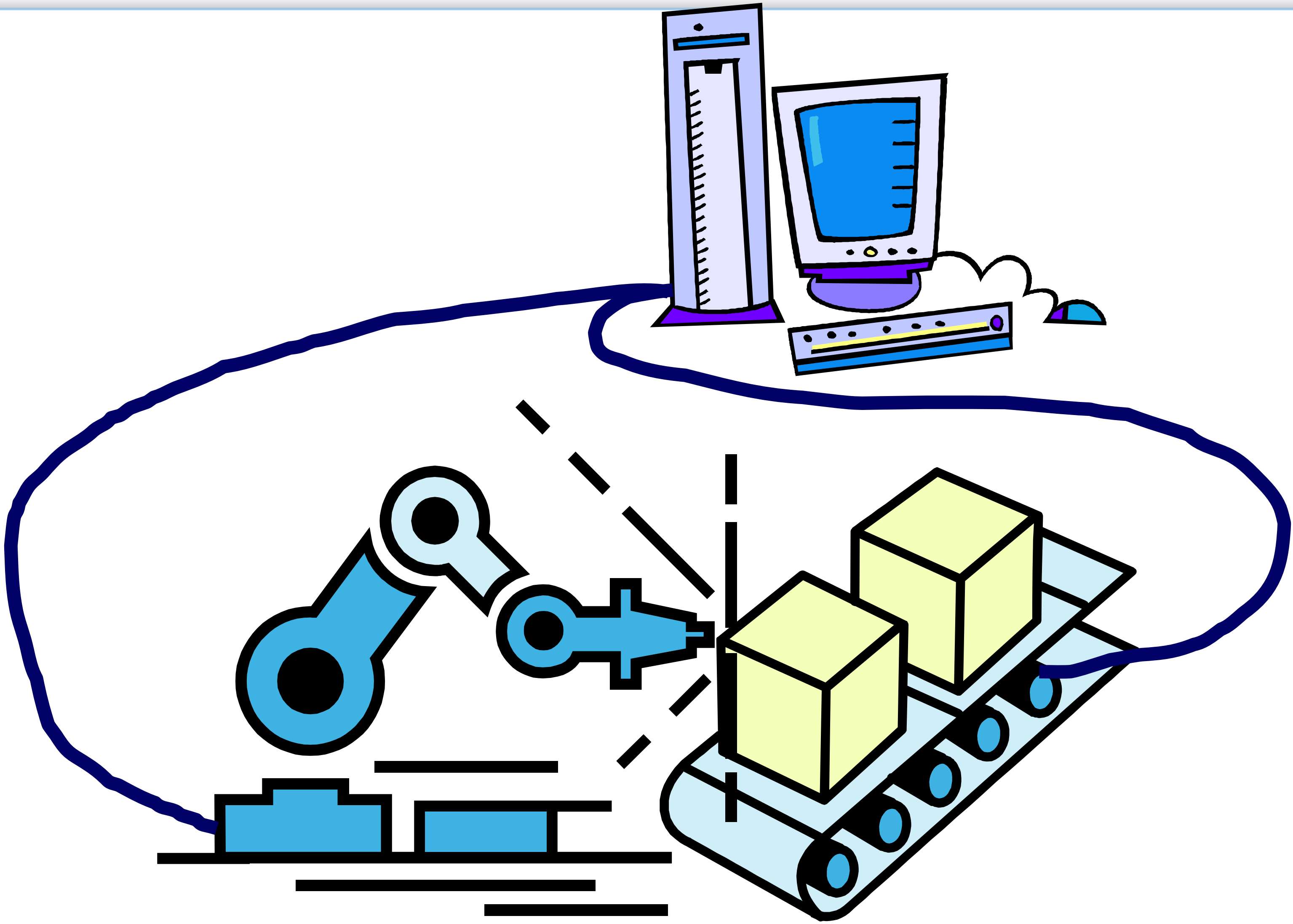
Sequence Enumeration

Define an equivalence relation on S^*:

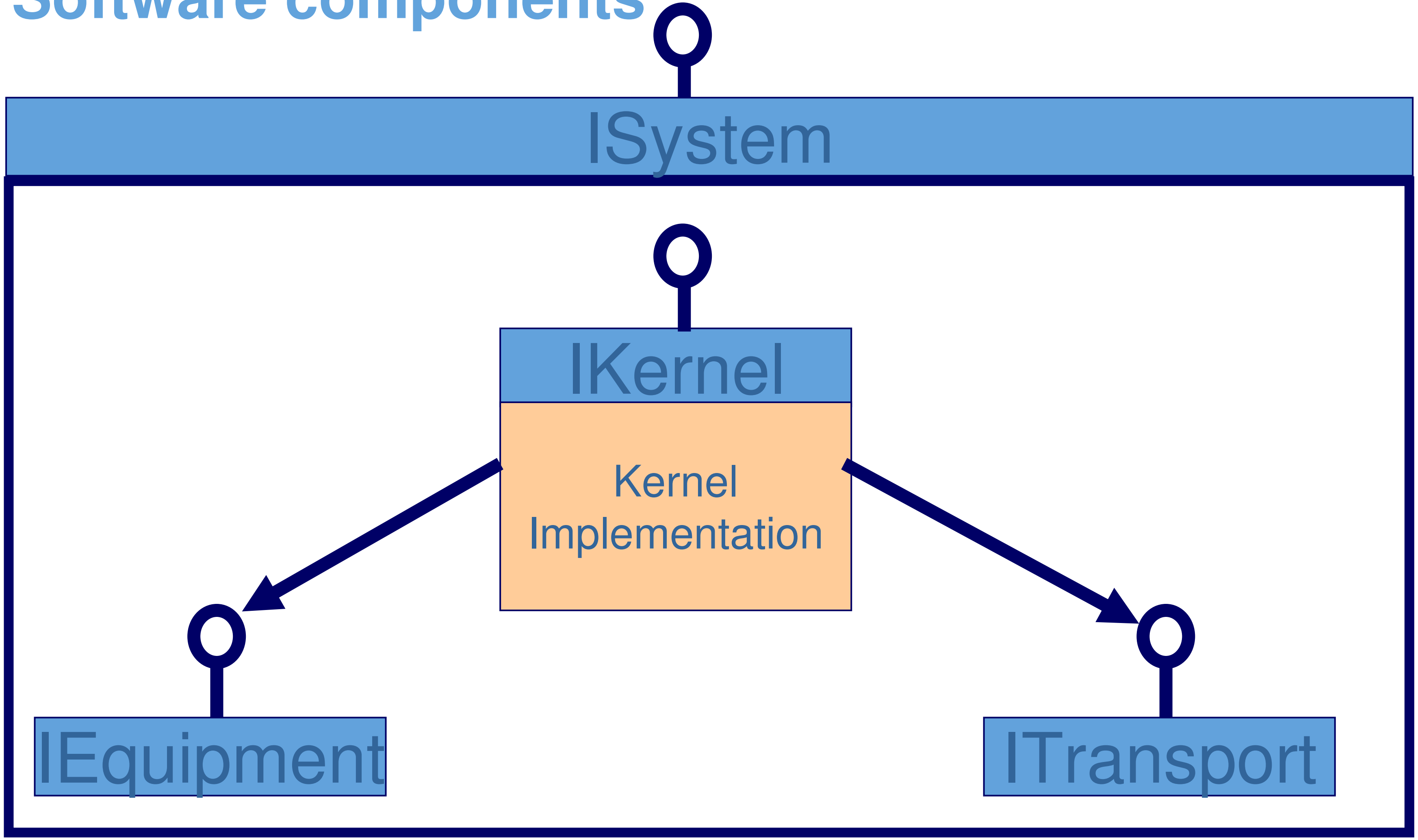
$s_0 \equiv s_1$ if all future behavior from s_0 is equal to that of s_1.

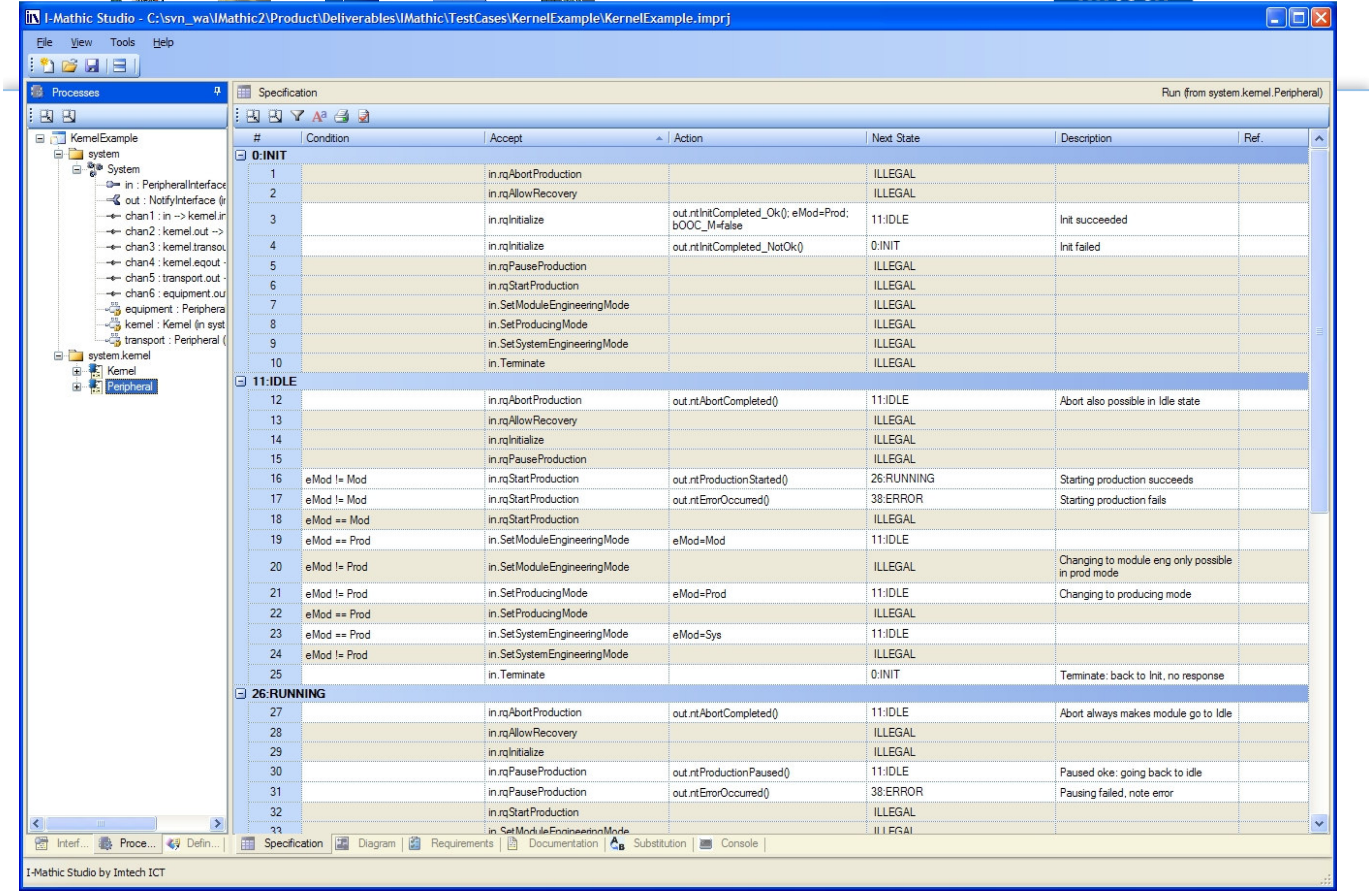
- Equivalence classes are identified by the shortest sequence, called Canonical Sequence.
- Now define F in terms of Canonical Sequences.

Note: All extensions of an illegal sequence are also illegal.



Software components





I-Mathic Studio - C:\svn_wa\IMathic2\Product\Deliverables\IMathic\TestCases\KernelExample\KernelExample.imprj

- KernelExample
  - system
    - System
      - in : PeripheralInterface
      - out : NotifyInterface (ir
      - chan1 : in --> kernel.ir
      - chan2 : kernel.out -->
      - chan3 : kernel.transou
      - chan4 : kernel.eqout -
      - chan5 : transport.out -
      - chan6 : equipment.ou
      - equipment : Periphera
      - kernel : Kernel (in syst
      - transport : Peripheral (
  - system.kernel
    - Kernel
    - Peripheral

Run (from system.kernel.Peripheral)

#	Condition	Accept	Action	Next State	Description	Ref.
0:INIT						
1		in.rqAbortProduction		ILLEGAL		
2		in.rqAllowRecovery		ILLEGAL		
3		in.rqInitialize	out.ntInitCompleted_Ok(); eMod=Prod; bOOC_M=false	11:IDLE	Init succeeded	
4		in.rqInitialize	out.ntInitCompleted_NotOk()	0:INIT	Init failed	
5		in.rqPauseProduction		ILLEGAL		
6		in.rqStartProduction		ILLEGAL		
7		in.SetModuleEngineeringMode		ILLEGAL		
8		in.SetProducingMode		ILLEGAL		
9		in.SetSystemEngineeringMode		ILLEGAL		
10		in.Terminate		ILLEGAL		
11:IDLE						
12		in.rqAbortProduction	out.ntAbortCompleted()	11:IDLE	Abort also possible in Idle state	
13		in.rqAllowRecovery		ILLEGAL		
14		in.rqInitialize		ILLEGAL		
15		in.rqPauseProduction		ILLEGAL		
16	eMod != Mod	in.rqStartProduction	out.ntProductionStarted()	26:RUNNING	Starting production succeeds	
17	eMod != Mod	in.rqStartProduction	out.ntErrorOccurred()	38:ERROR	Starting production fails	
18	eMod == Mod	in.rqStartProduction		ILLEGAL		
19	eMod == Prod	in.SetModuleEngineeringMode	eMod=Mod	11:IDLE		
20	eMod != Prod	in.SetModuleEngineeringMode		ILLEGAL	Changing to module eng only possible in prod mode	
21	eMod != Prod	in.SetProducingMode	eMod=Prod	11:IDLE	Changing to producing mode	
22	eMod == Prod	in.SetProducingMode		ILLEGAL		
23	eMod == Prod	in.SetSystemEngineeringMode	eMod=Sys	11:IDLE		
24	eMod != Prod	in.SetSystemEngineeringMode		ILLEGAL		
25		in.Terminate		0:INIT	Terminate: back to Init, no response	
26:RUNNING						
27		in.rqAbortProduction	out.ntAbortCompleted()	11:IDLE	Abort always makes module go to Idle	
28		in.rqAllowRecovery		ILLEGAL		
29		in.rqInitialize		ILLEGAL		
30		in.rqPauseProduction	out.ntProductionPaused()	11:IDLE	Paused oke: going back to idle	
31		in.rqPauseProduction	out.ntErrorOccurred()	38:ERROR	Pausing failed, note error	
32		in.rqStartProduction		ILLEGAL		
33		in.SetModuleEngineeringMode		ILLEGAL		

I-Mathic Studio by Imtech ICT

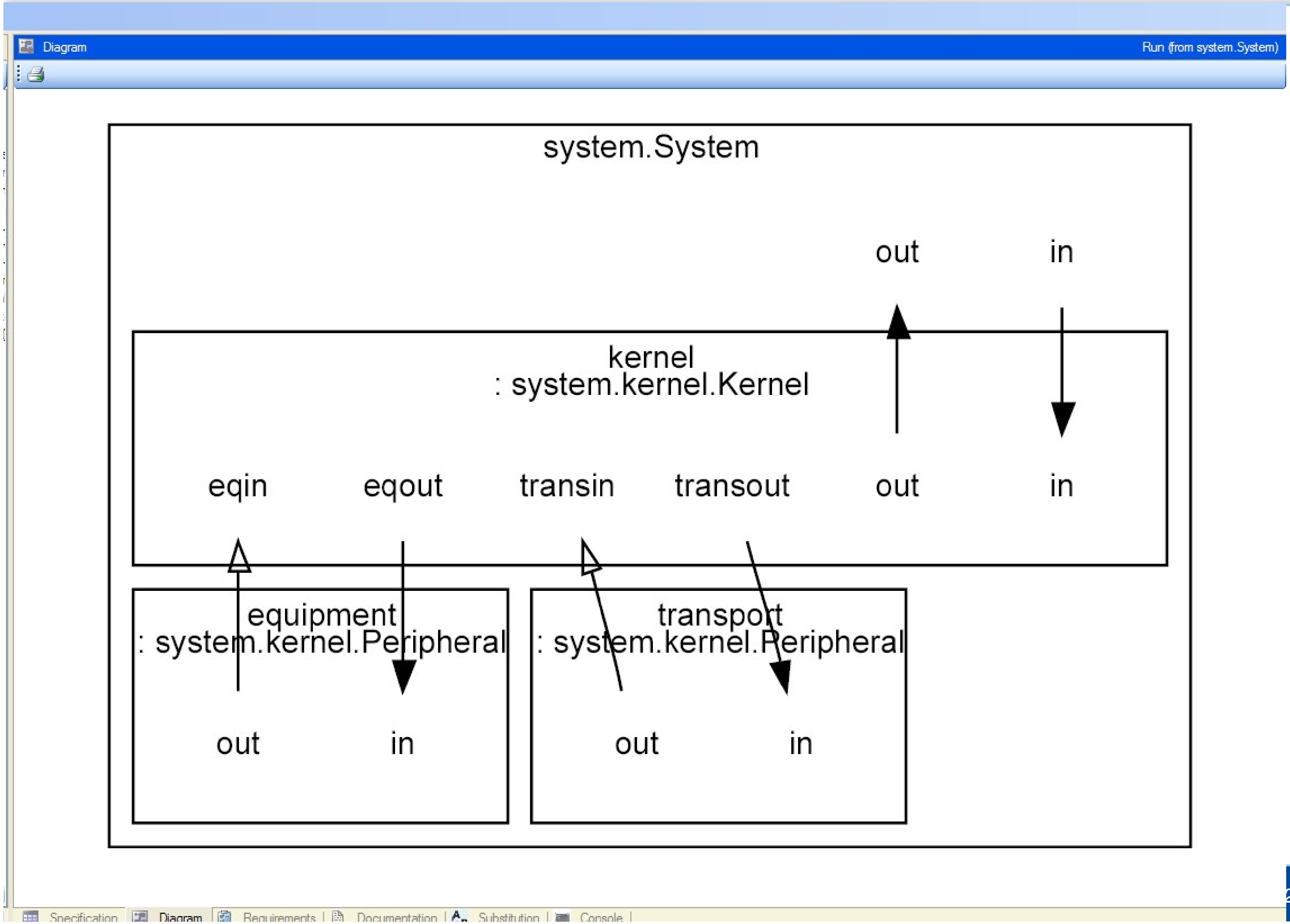


Diagram
Run (from system.System)
system.System
out
in
kernel
: system.kernel.Kernel
eqin
eqout
transin
transout
out
in
equipment
: system.kernel.Peripheral
out
in
transport
: system.kernel.Peripheral
out
in
Specification
Diagram
Requirements
Documentation
Substitution
Console

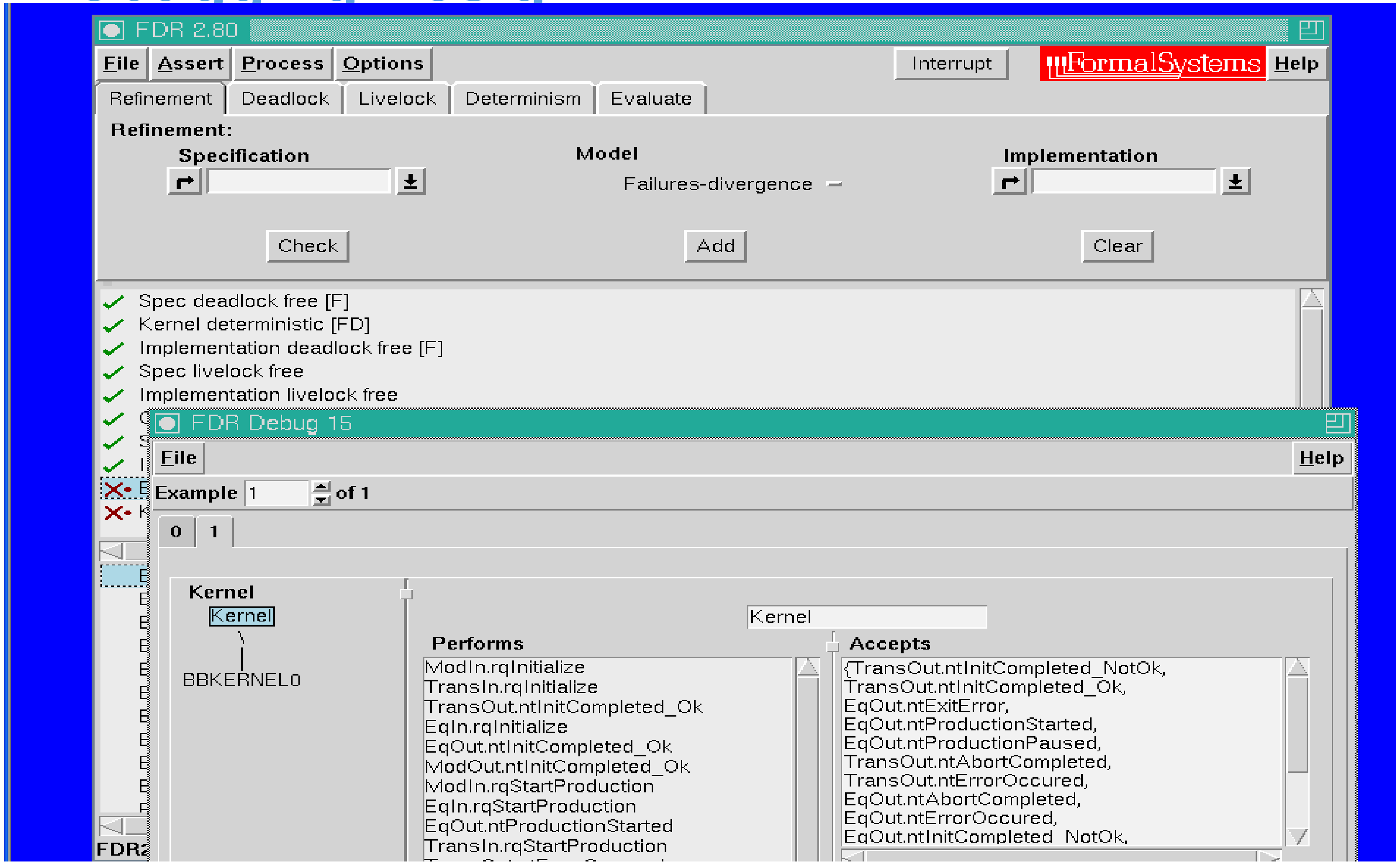
Sequence Enumeration Results

- The Module Function is **complete**
  - Total function: Maps every possible input sequence to response
- The Module is the **right** system
  - Every transition rule justified
  - Full requirements tracing
  - Derived requirements fill the gaps – we do not leave this to the programmer
- Is the Module **correct**?

Verification

- CSP: Communicating Sequential Processes
- Model checker explores all state combinations ensuring that:
  - Model is deterministic
  - Model implements interface according to specification
  - There are no deadlocks
  - There are no livelocks
  - Queues never full (processes behave freely)

Debugging Design



FDR 2.80
File Assert Process Options
Interrupt
FormalSystems
Help
Refinement Deadlock Livelock Determinism Evaluate
Check
Add
Clear

Refinement:

Specification

Model

Failures-divergence

Implementation

✓ Spec deadlock free [F]

✓ Kernel deterministic [FD]

✓ Implementation deadlock free [F]

✓ Spec livelock free

✓ Implementation livelock free

Example 1 of 1

Performs

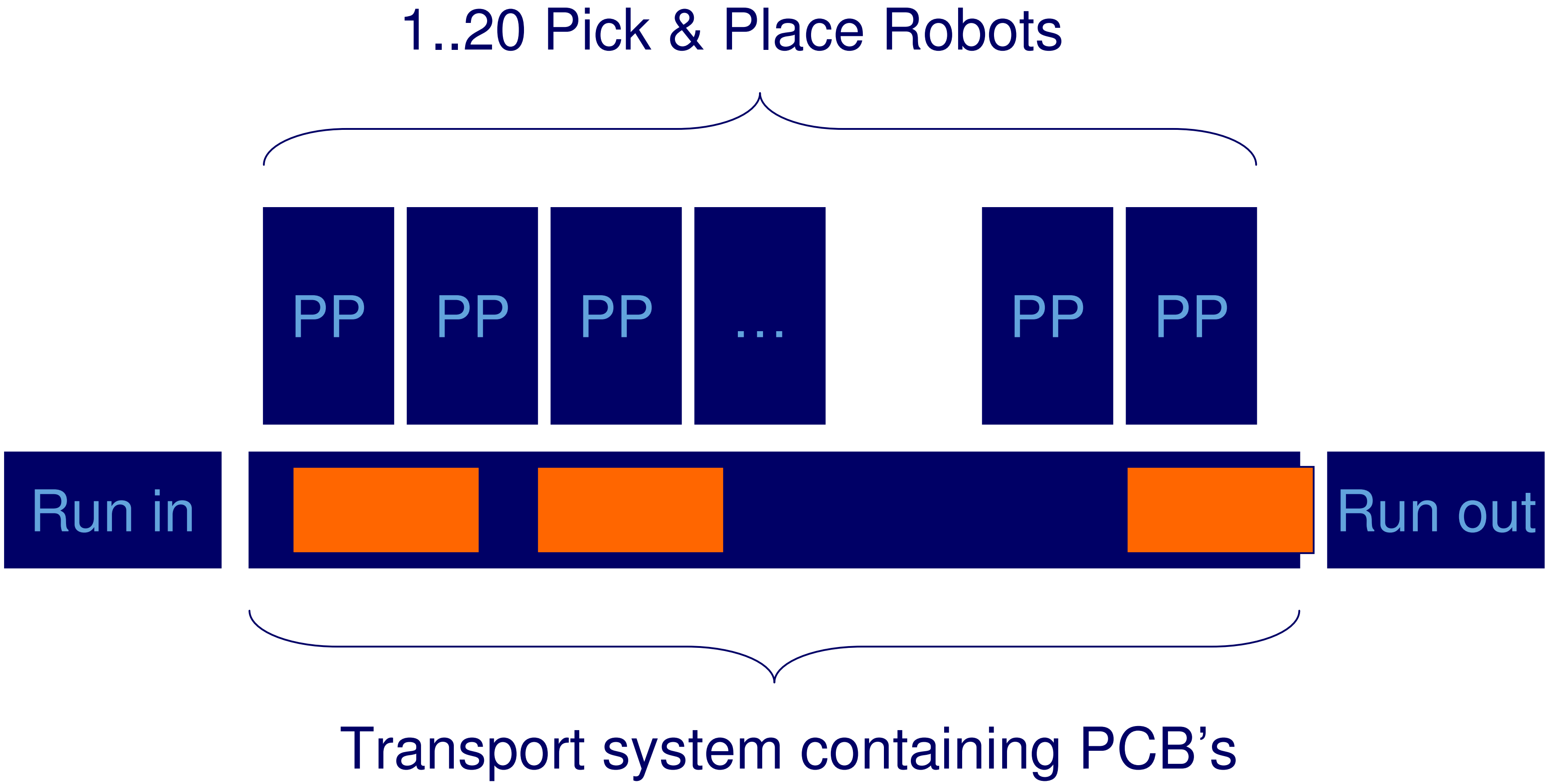
```
ModIn.rqInitialize
TransIn.rqInitialize
TransOut.ntInitCompleted_Ok
EqIn.rqInitialize
EqOut.ntInitCompleted_Ok
ModOut.ntInitCompleted_Ok
ModIn.rqStartProduction
EqIn.rqStartProduction
EqOut.ntProductionStarted
TransIn.rqStartProduction
```

Accepts

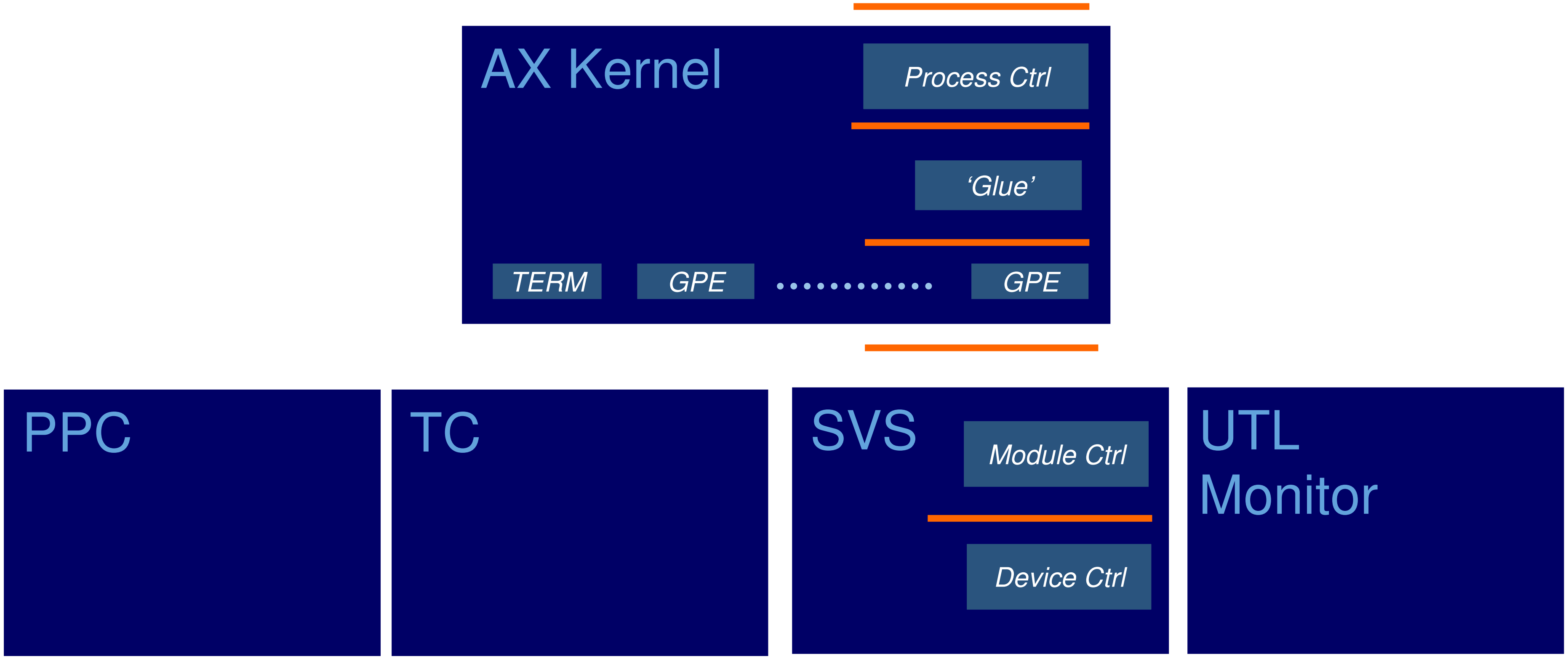
```
{TransOut.ntInitCompleted_NotOk,
TransOut.ntInitCompleted_Ok,
EqOut.ntExitError,
EqOut.ntProductionStarted,
EqOut.ntProductionPaused,
TransOut.ntAbortCompleted,
TransOut.ntErrorOccured,
EqOut.ntAbortCompleted,
EqOut.ntErrorOccured,
EqOut.ntInitCompleted_NotOk,
```

Assembléon AX

Schematic overview



Part of system architecture



Results measured over 3 AX projects

- Number of errors reduced by 40%. Most difficult ones were gone (no more deadlocks or race conditions), only “easy to solve ones” remained.
- Total effort was significantly reduced compared to industry averages.

Handling industrial size systems

We have seen that I-Mathic makes formal methods accessible to software engineers. But can it handle “real” systems?

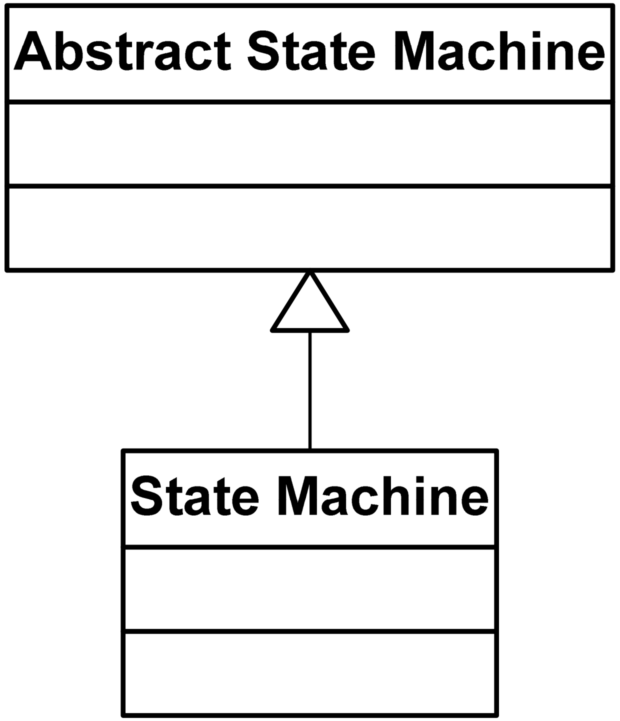
YES! Because of:

- **Algorithms**
- **System architecture**
- **Refinement**

We have demonstrated this on the AX (more than one million lines of code)

Code Generation

- Generation of state machine code from sequence enumeration;
- Separation of interaction from actual implementation to facilitate updates.



Abstract State Machine

State Machine

Technical Details: CSP

$P0 = a \rightarrow b \rightarrow P0$

$\quad\quad \square\ c \rightarrow P1$

$R = x \rightarrow a \rightarrow R$

$P1 = d \rightarrow b \rightarrow b \rightarrow P0$

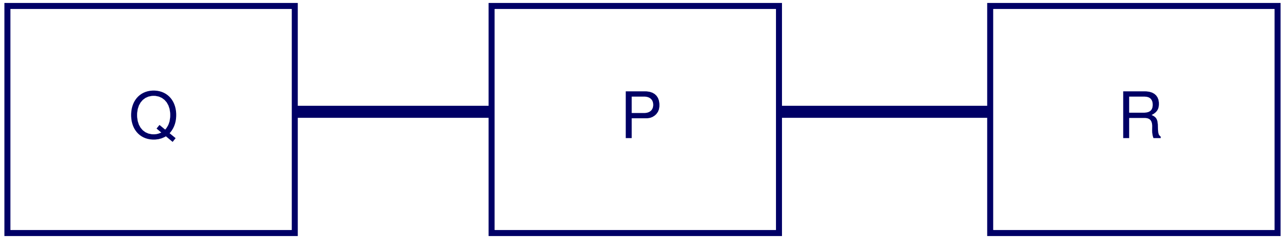
$$Sys = P0 \underset{\{a\}}{||} R$$

<x, a, b> is a trace of Sys

<a, b, x> is not

Channels in CSP

P = c?x → d!x -> P (one place buffer)



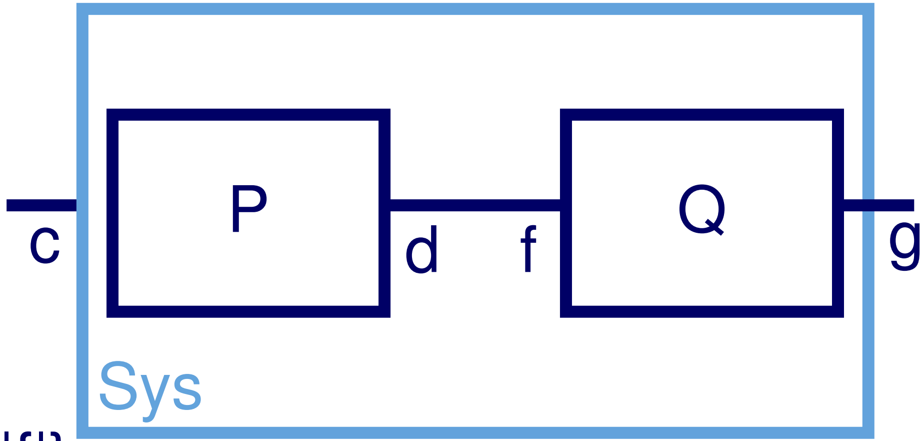
Q = c!5 → Q

R = d?y → R

$$Sys = Q \underset{\{c\}}{||} P \underset{\{d\}}{||} R$$

<c.5, d.5, c.5>
is a trace of Sys

Renaming and Hiding



P = c?x → d!(x+1) → P

Q = f?x → g!(x*2) → Q

Sys = (P [[d<-f]] [{|f|}] Q) \ {|f|}

Sys = P [d <-> f] Q

<c.4, g.10> is a trace of Sys

Deterministic and non-deterministic processes

$$\begin{aligned} P0 = \; & b \rightarrow P0 \\ \square \; & c \rightarrow P0 \end{aligned}$$

$$\begin{aligned} P2 = \; & a \rightarrow b \rightarrow P2 \\ \square \; & a \rightarrow c \rightarrow P2 \end{aligned}$$

$$\begin{aligned} P1 = \; & b \rightarrow P1 \\ \sqcap \; & c \rightarrow P1 \end{aligned}$$

<b, c, c, c, b> is a trace of both P0 and P1,

But they behave differently!

P0 [T= P1 and P1 [T= P0 (equivalent in the Trace model)

Not P0 [F= P1 (not equivalent in the Failures model)

CSP and “normal” software

Software	CSP
Component	Process
Method call	Event
Message passing	Event
Handle message	Deterministic choice
Association	Channel

Threads

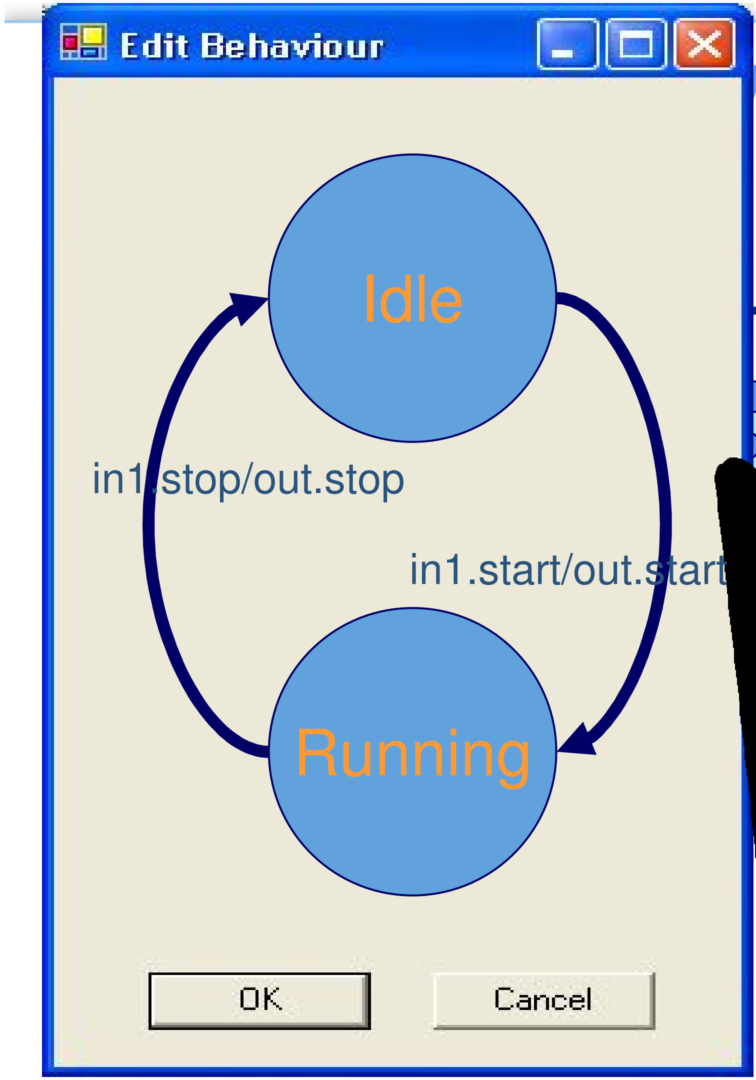
- A CSP process has specific communication points
- Multi-threaded applications are chaotic
- Solutions:
  - Event queue (active object pattern, command pattern)
  - Mutex on component methods
  - Model thread interaction explicitly (shared variables, semaphores…)

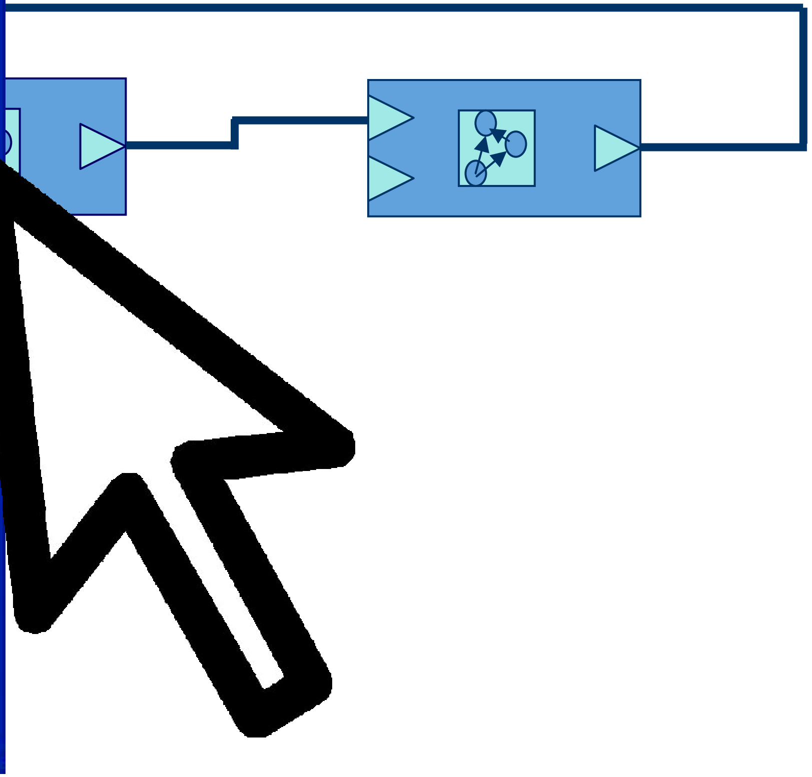
Future Developments

- Better tools, more automation
- Better debug capabilities
- Handle extended finite state machines
- …and many more ideas…

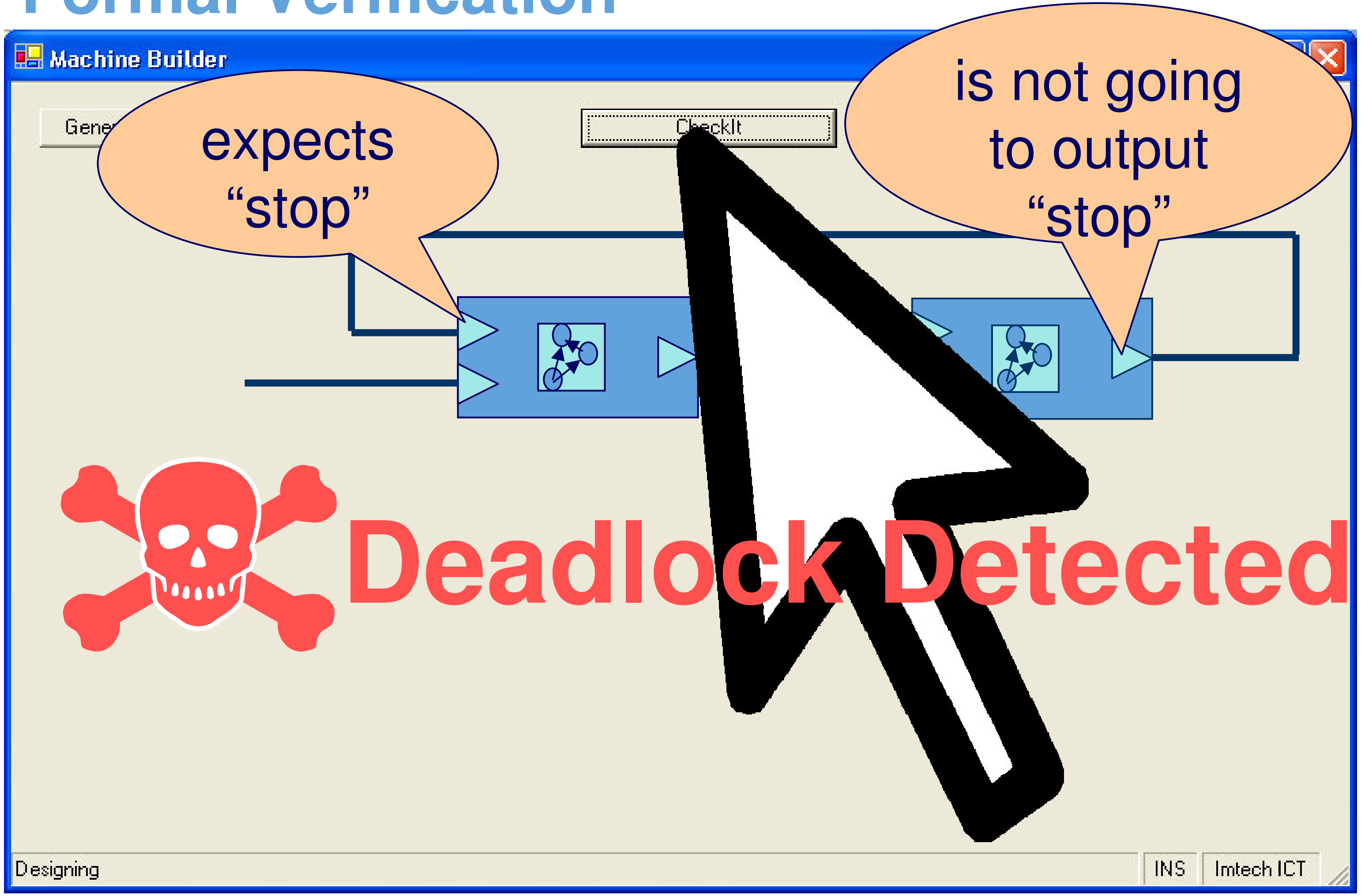
(Interested to help? Emile.vanGerwen@imtech.nl)

viour


Edit Behaviour
Idle
in1.stop/out.stop
in1.start/out.start
Running
OK
Cancel



Formal Verification



expects "stop"

is not going to output "stop"

Deadlock Detected

Questions

?

Literature

- Prowell S., Trammell C., Linger R., Poore J, *Cleanroom Software Engineering: Technology and Process*, Addison-Wesley, 1999.
- Hoare C.A.R., *Communicating Sequential Processes*, Prentice Hall International, 1985.
- Hall A., *Seven Myths of Formal Methods*, in: IEEE Software Vol 7 No 5, 1990.
- *FDR2 User Manual*, Formal systems Europe Ltd, 2003.