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

Typical Occurrence of Rework

Typical Origins of Rework
V-Model

I-Mathic: fix design errors in design phase
I-Mathic view of the world
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

ISystem

IKernel

Kernel Implementation

IEquipment

ITransport
### Sequence Enumeration Example

<table>
<thead>
<tr>
<th>Condition</th>
<th>Accept</th>
<th>Action</th>
<th>Next State</th>
<th>Description</th>
<th>Ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td></td>
<td>m.inputProduction</td>
<td>ILLEGAL</td>
<td>Init succeeded</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>m.outputRecovery</td>
<td>ILLEGAL</td>
<td>Init failed</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>m.initialize</td>
<td>ILLEGAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>m.startProduction</td>
<td>ILLEGAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>m.startProduction</td>
<td>ILLEGAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>m.startProducingMode</td>
<td>ILLEGAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>m.startSystemEngineeringMode</td>
<td>ILLEGAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>m.terminate</td>
<td>ILLEGAL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| IDLE            |        | m.inputProduction                         | 11/DIE     | Abort also possible in idle state     |     |
|                 |        | m.outputRecovery                          | ILLEGAL    |                                       |     |
|                 |        | m.initialize                               | ILLEGAL    |                                       |     |
|                 |        | m.startProduction                         | ILLEGAL    |                                       |     |
|                 |        | m.startProducingMode                      | ILLEGAL    |                                       |     |
|                 |        | m.startSystemEngineeringMode              | ILLEGAL    |                                       |     |
|                 |        | m.terminate                                | ILLEGAL    |                                       |     |

| RUNNING         |        | m.inputProduction                         | 11/DIE     | Abort always makes module go to idle |     |
|                 |        | m.outputRecovery                          | ILLEGAL    |                                       |     |
|                 |        | m.initialize                               | ILLEGAL    |                                       |     |
|                 |        | m.startProduction                         | ILLEGAL    |                                       |     |
|                 |        | m.startProducingMode                      | ILLEGAL    |                                       |     |
|                 |        | m.startSystemEngineeringMode              | ILLEGAL    |                                       |     |
|                 |        | m.terminate                                | ILLEGAL    |                                       |     |

| Terminate       |        | m.terminate                                | 0/INIT     | Terminate, back to INIT, no response  |     |

### Technical Systems

TUE 18 april 2006
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

![Image of debugging design software interface]
Assembléon AX
Schematic overview

1..20 Pick & Place Robots

Transport system containing PCB’s
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.
Technical Details: CSP

\[ P_0 = a \rightarrow b \rightarrow P_0 \]
\[ \downarrow c \rightarrow P_1 \]

\[ P_1 = d \rightarrow b \rightarrow b \rightarrow P_0 \]

\[ R = x \rightarrow a \rightarrow R \]

\[ \text{Sys} = P_0 \parallel R \]
\[ \{a\} \]

\( <x, a, b> \) is a trace of Sys

\( <a, b, x> \) is not
Channels in CSP

P = c?x $\rightarrow$ d!x $\rightarrow$ P  (one place buffer)

Q = c!5 $\rightarrow$ Q

R = d?y $\rightarrow$ R

Sys = Q $\parallel$ P $\parallel$ R
{c}  {d}

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

\[ P = c ? x \rightarrow d !(x+1) \rightarrow P \]

\[ Q = f ? x \rightarrow g !(x*2) \rightarrow Q \]

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

\[ \text{Sys} = P [ d \leftrightarrow f ] Q \]

\(<c.4, g.10>\) is a trace of \text{Sys}\]
Deterministic and non-deterministic processes

\[ P_0 = b \rightarrow P_0 \]
\[ \diamond c \rightarrow P_0 \]
\[ P_1 = b \rightarrow P_1 \]
\[ \Pi c \rightarrow P_1 \]

\( <b, c, c, c, b> \) is a trace of both \( P_0 \) and \( P_1 \),

But they behave differently!

\( P_0 [T= P_1 \) and \( P_1 [T= P_0 \) (equivalent in the Trace model)

\textit{Not} \( P_0 [F= P_1 \) (not equivalent in the Failures model)
## CSP and “normal” software

<table>
<thead>
<tr>
<th>Software</th>
<th>CSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Process</td>
</tr>
<tr>
<td>Method call</td>
<td>Event</td>
</tr>
<tr>
<td>Message passing</td>
<td>Event</td>
</tr>
<tr>
<td>Handle message</td>
<td>Deterministic choice</td>
</tr>
<tr>
<td>Association</td>
<td>Channel</td>
</tr>
</tbody>
</table>
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)
Specifying behaviour

- **Idle**
  - in1.stop/out.stop

- **Running**
  - in1.start/out.start

Technical Systems TUE 18 april 2006
Formal Verification

- expects "stop"
- is not going to output "stop"

Deadlock Detected
Questions
Literature


