Formal Verification of Implementability of Timing Requirements

Mark Lawford, Associate Professor
(Joint work with Xiayong (Jason) Hu and Alan Wassyng)

Software Quality Research Laboratory
McMaster University
Hamilton, ON, Canada

CASL University College Dublin 2007

Outline

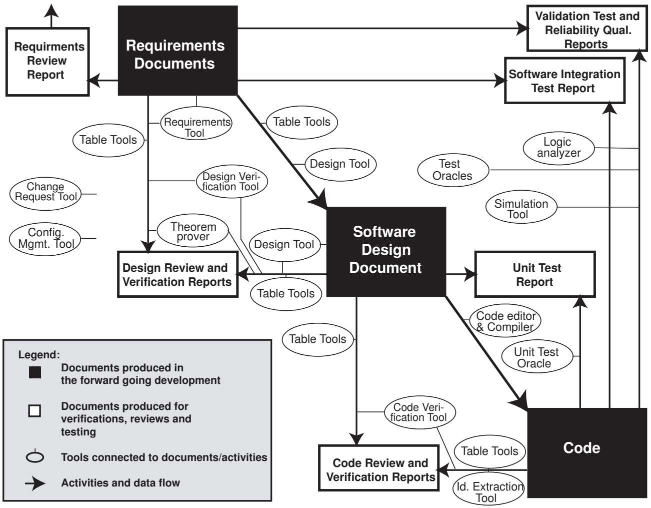
Background

- Built on definitions and analysis developed at Ontario Power Generation for safety-critical software applications in the nuclear power industry.
- In particular, comes from our experiences working in industry on Reactor Shutdown System (SDS).

What is an SDS?

- watchdog system that monitors system parameters
- shuts down (trips) reactor if it observes “bad” behavior
- process control is performed a separate Digital Control computer (DCC) - not as critical

Life-cycle phases, documents and tools[WL03]



Focus on tool support for Systematic Design Verification (SDV)

4-Variable Model (Parnas &Madey) [PM95]

$$\begin{tikzcd} M \arrow[r, "REQ"] \arrow[d, "IN"'] & C \\ I \arrow[r, "SOF"] & O \arrow[u, "OUT"'] \end{tikzcd}$$

M - Monitored Variable statespace
I - Input Variable statespace
C - Controlled Variable statespace
O - Output Variable statespace

M, **C**, **I**, **O** are time series vectors and *REQ*, *SOF*, *IN*, *OUT* are relations.
We use a special case where all relations are functional resulting in proof obligation:

$$REQ = OUT \circ SOF \circ IN \tag{1}$$

Here *REQ* and *SOF* are the one step transition functions of the requirements and design respectively.

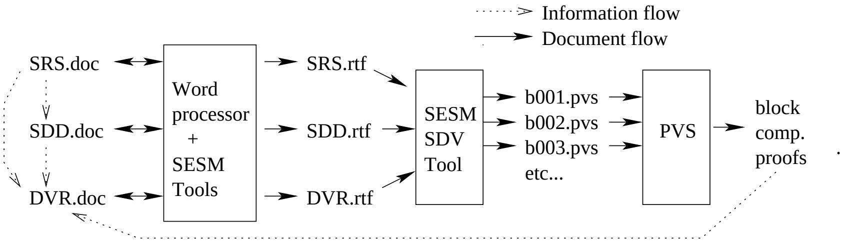
Vertical Decomposition of Proof Obligations [LMFM00]

$$\begin{tikzcd}
M \arrow[rrr, "REQ"] \arrow[d, "IN"'] \arrow[dr, "Abst_M"] & & & C \\
I \arrow[r, "SOF_{in}"] & M_p \arrow[r, "SOF_{req}"] & C_p \arrow[r, "SOF_{out}"] & O \arrow[u, "OUT"'] \\
& & & C \arrow[ul, "Abst_C"', phantom]
\end{tikzcd}$$

$$\begin{aligned}
Abst_{\mathbf{C}} \circ REQ &= SOF_{req} \circ Abst_{\mathbf{M}} && (2)\\
Abst_{\mathbf{M}} &= SOF_{in} \circ IN && (3)\\
id_{\mathbf{C}} &= OUT \circ SOF_{out} \circ Abst_{\mathbf{C}} \ . && (4)
\end{aligned}$$

$\mathbf{M}_p$ is *pseudo-monitored* variables
$\mathbf{C}_p$ is *pseudo-controlled variables*

Tools and Documents Systematic Design Verification



SRS and SDD documents consisted mainly of formal tabular specifications and some informal description.

Some Statistics from [LFMar]

The complete systems are relatively small.

- Excluding comments and blank lines:

SDS1:	12,000 lines of FORTRAN and assembler
SDS2:	17,000 lines of Pascal and assembler.

- Documentation was extensive:
  - Requirements document (SRS) approx 400 pages
  - Design document (SDD) 500+ pages
  - Design verification report 600+ pages

SDS1:	Generated 11,000+ lines of PVS input in 60 files (SDS1).
SDS2:	Generated 13,000+ lines of PVS input in 102 files (SDS2).

Took several hours on a Windows NT based 75 MHz Pentium system with 32MB of RAM.

More Statistics

- Verification was performed by engineers with no previous experience with PVS or similar proof systems who each received a week-long training course in PVS.
- Block comparison proofs and documentation of any discrepancies uncovered in the process took one person less than 2 weeks for each of the systems.
- This process resulted in roughly 70% of the over 200 functional blocks from the two software designs of the Redesign Project being formally verified using the SDV Tool together with PVS.
- Timing requirements were verified manually.

Example

- To filter noisy signals we may specify that a sensor signal above its setpoint (the event) sustained for 300 ms causes a “trip”.
- If the event is sustained for less than 300 ms, the trip must not occur.
- Similarly, if the event is sustained for 300 ms or longer, the trip must be generated.

What’s needed?

- describe behaviour that includes tolerances in time durations.
- describe timing tolerances that allow deviations from ideal behaviour specified by typical requirements models.

Example

- To filter noisy signals we may specify that a sensor signal above its setpoint (the event) sustained for 300 ms causes a “trip”.
- If the event is sustained for less than 300 ms, the trip must not occur.
- Similarly, if the event is sustained for 300 ms or longer, the trip must be generated.

What’s needed?

- describe behaviour that includes tolerances in time durations.
- describe timing tolerances that allow deviations from ideal behaviour specified by typical requirements models.

Example

- To filter noisy signals we may specify that a sensor signal above its setpoint (the event) sustained for 300 ms causes a “trip”.
- If the event is sustained for less than 300 ms, the trip must not occur.
- Similarly, if the event is sustained for 300 ms or longer, the trip must be generated.

What’s needed?

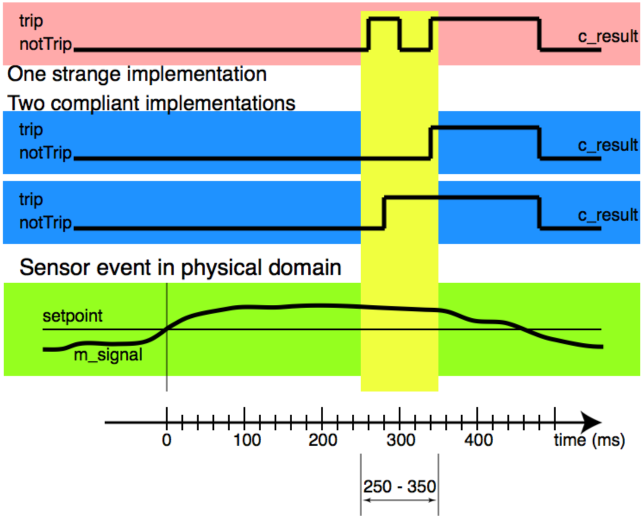
- describe behaviour that includes tolerances in time durations.
- describe timing tolerances that allow deviations from ideal behaviour specified by typical requirements models.

Note:

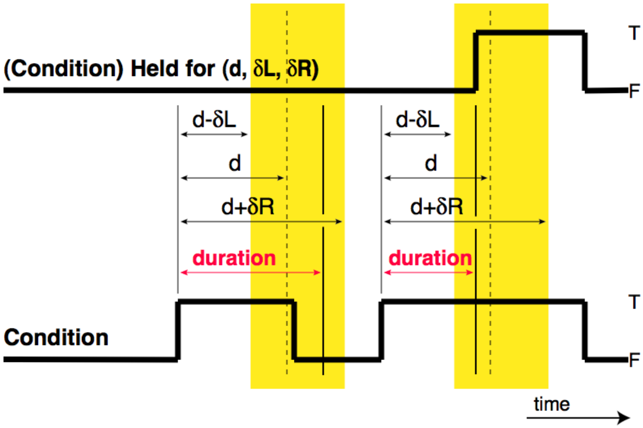
Without tolerances on the time duration, these requirements would be impossible to meet, so the duration may be specified as 300 ± 50 ms for instance

Implementations of example specification.

Informal spec: Trip if *m_signal* is above setpoint for 300 ± 50 ms.

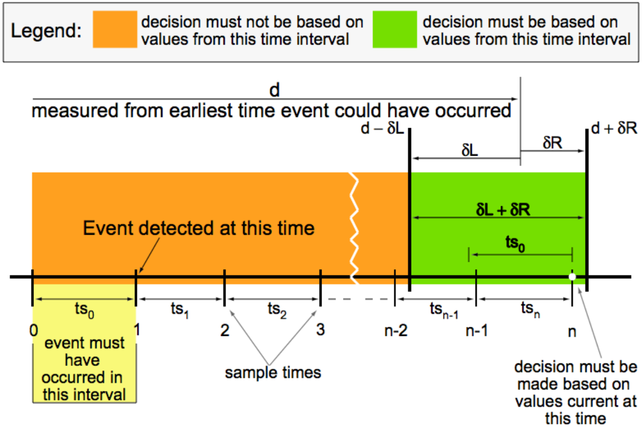


(Condition) Held for (d, δ_L, δ_R) (Infix operator)



Acceptable duration$\in (d - \delta_L, d + \delta_R)$
$d > 0$, is the ideal behavior.

Implementation with Non-uniformly Spaced Samples


Legend:
decision must not be based on values from this time interval
decision must be based on values from this time interval
d
measured from earliest time event could have occurred
d – δL
d + δR
δL
δR
δL + δR
ts0
Event detected at this time
ts0
ts1
ts2
tsn-1
tsn
0
1
2
3
n-2
n-1
n
event must have occurred in this interval
sample times
decision must be made based on values current at this time

When can we detect a sustained condition when samples are not uniformly spaced?

Implementation Situations

Let $\mathit{Sample} : \mathbb{N} \rightarrow \mathbb{R}^{\geq 0}$ be a sequence of sample times such that

$$\forall n : T_{min} \leq \mathit{Sample}(n+1) - \mathit{Sample}(n) \leq T_{max}$$

Different implementation scenarios for sustained events:

a) Omniscient: Know the exact time that the condition becomes true but can only react at sample times,

b) Perfect Clock: Know the value of the condition at sample instances and the exact timing of samples

c) Imperfect Clock: Same as (b) but with access to an imperfect clock (e.g. finite precision, bounded drift) , and

d) No Clock: Knowledge only of T_{min} and T_{max} and the number of samples since the condition became true.

Case (b) Access to Perfect Clock at Sample Times

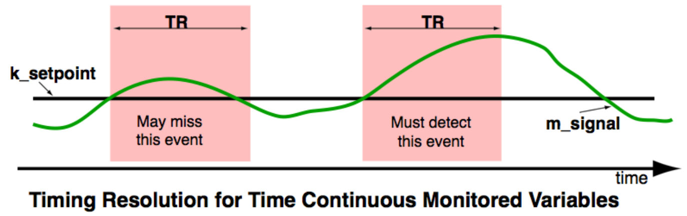
Assumptions:

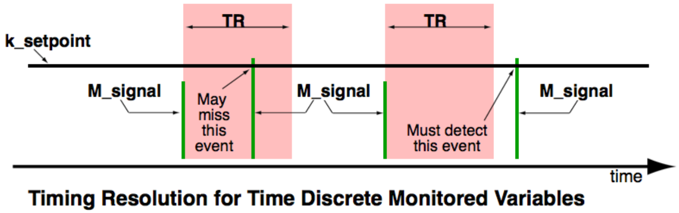
1. No intersample "ripple". (eg. True False True between samples)
2. Bounded $[T_{min}, T_{max}]$ intersample variation (aka jitter).
3. We only know the value of $m\_signal > setpoint$ at sample instances but we know the exact timing of samples.

Question: How do we formally capture if a requirement is feasible in this case?

$$\begin{aligned} Feasible(d) \quad : \quad & bool = \forall Sample : \forall n_0 : \exists n : \\ & \forall (t |\ Sample(n_0) \leq t \leq Sample(n_0 + 1)) : \\ & d - \delta_L \leq Sample(n) - t \leq d + \delta_R \end{aligned}$$

Ignoring Intersample Ripple





Question: In the perfect clock case, what must hold for a sustained condition requirement to detectable and thus implementable (feasible) with non-uniform sampling?

Theorem

Assume that $T_{min} < T_{max}$, $\delta_L > 0$, $\delta_R > 0$, and $d > \max(\delta_R, T_{max} + \delta_L)$. Let $K_{min} = \lfloor (d - delta_L)/Tmax \rfloor$. Then

Case 1 : $T_{max} \leq \frac{\delta_L + \delta_R}{2} \Rightarrow Feasible(d)$

Case 2 : $\frac{\delta_L + \delta_R}{2} < T_{max} \leq \delta_L + \delta_R \Rightarrow$

$$T_{min} \geq \frac{d - \delta_L}{K_{min} + 1} \wedge (K_{min} + 2) * T_{max} \leq d + \delta_R \Leftrightarrow Feasible(d)$$

Case 3 : $T_{max} > \delta_L + \delta_R \Rightarrow \neg Feasible(d)$

The above result has been formalized in the Open Source PVS Theorem Prover.

Benefits of Formalization in PVS

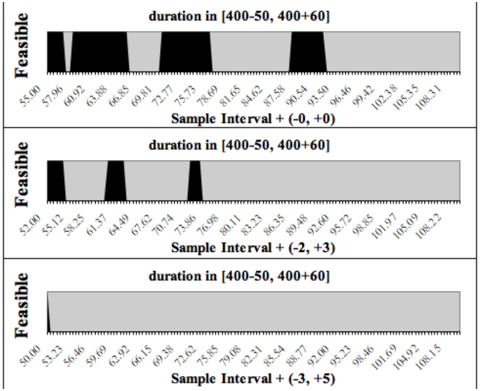
Similar result was originally stated informally in our FM2005 paper [WLH05]. Formalization in PVS

- Detected incorrect boundary case in informal statement - Case 2 in [WLH05] used condition $K_{min} = K_{max}$ as first conjunct which is equivalent to $T_{min} > \frac{d-\delta_L}{K_{min}+1}$
- Forced us to consider and precisely define different implementation situations
- Helps understanding and increased confidence in result
- Will let us automatically verify implementablity of requirements and largely automate Systematic Design Verification

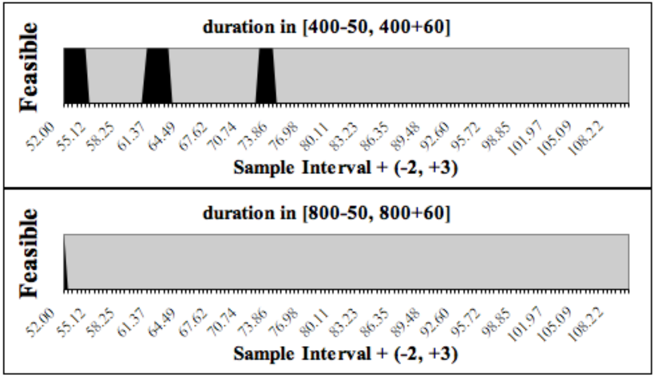
Implications: Slower is not always worse!

For Case 2:

- Increasing jitter decreasing chance of implementing
- Sometimes you can make requirement implementable by sampling slower! (and reducing CPU usage and power consumption!)



Implications: Increasing *d* reduces your chances



Can similarly see that increasing nominal duration *d* of sustained event reduces chance of implementation.

Related Work

Implementaion of Timed Automata (TA) [DDR04, DDMR04, DDR05]

- Both the controller and plant are modeled by TA with ASAP semantics.
- implement continuous time controller with a discrete time system
- assume there is a delay Δ associated with the controller's reaction to the environment.
- Objective: ensure that the closed-loop system satisfies a safety properties. by avoiding bad states.
- if all control actions can be delayed by $\Delta > 0$ without violating the safety property, the controller is “implementable”.

Implementaion of Timed Automata (TA) [DDR04, DDMR04, DDR05]

Results:

- PSPACE-complete decision procedure for “implementable”
- [DDR04] semi-decision procedure to compute the maximal delay Δ allowable by the implementation that preserves correctness.
- conclude system is implementable by a CE with loop time upper bound Δ_L and a finite precision clock with a resolution of Δ_P, provided $\Delta > 3\Delta_L + 4\Delta_P$.

Comparison

- Approach is *One size fits all* vs. *per requirement* timing tolerances
- Explicitly addresses clock resolution
- We’re simpler and generally less restrictive

Conclusions and Future Research

Conclusions

- We can precisely define different implementation situations
- We can formally model timing requirements with tolerances and determine when they can be implemented by non-uniform sampling.
- Theorem prover can help to refine and validate results.

Future Work

- Finishing statements and proofs for other implementation situations
- Consider applying different clock rates for data streams in the real-time systems.
- Create a real-time property verification library, including different timing operators.

Other Automated Resoning Work

1. Table Tool 2.0
2. Automated WCET/BCET extraction from microcontroller assembly code
3. Integrating formal methods into the software process
4. Supervisory control of discrete event system (looking at architectural issues for synthesis of correct by construction controllers)
5. Verification of image processing software

M. De Wulf, L. Doyen, N. Markey, and J.-F. Raskin.
Robustness and implementability of timed automata.
In *Proc. of FORMATS04,*, volume 3253 of *Lecture Notes in Computer Science*, pages 152–166, Grenoble, 2004.

M. De Wulf, L. Doyen, and J.-F. Raskin.
Almost asap semantics: From timed models to timed implementations.
In *In the Proc. of HSCC04*, volume 2993 of *Lecture Notes in Computer Science*, pages 296 – 310, 2004.

Martin De Wulf, Laurent Doyen, and Jean-François Raskin.
Systematic implementation of real-time models.
In J. Fitzgerald, I.J. Hayes, and A. Tarlecki, editors, *FM 2005: Formal Methods: International Symposium of Formal Methods Europe Proceedings*, volume 3582 of *LNCS*, pages 139 – 156, Newcastle, UK, July 2005. Springer-Verlag.

M. Lawford, P. Froebel, and G. Moum.
Application of tabular methods to the specification and verification of a nuclear reactor shutdown system.
Accepted for publication in Oct 2004. `http://www.cas.mcmaster.ca/~lawford/papers/`, To appear.

M. Lawford, J. McDougall, P. Froebel, and G. Moum.
Practical application of functional and relational methods for the specification and verification of safety critical software.
In T. Rus, editor, *Proceedings Algebraic Methodology and Software Technology, 8th International Conference, AMAST 2000, Iowa City, Iowa, USA, May 2000*, volume 1816 of *Lecture Notes in Computer Science*, pages 73–88. Springer, 2000.

David Lorge Parnas and Jan Madey.
Functional documents for computer systems.
Science of Computer Programming, 25(1):41–61, October 1995.

Alan Wassyng and Mark Lawford.
Lessons learned from a successful implementation of formal methods in an industrial project.
In K. Araki, S. Gnesi, and D. Mandrioli, editors, *FME 2003: International Symposium of Formal Methods Europe Proceedings*, volume 2805 of *Lecture Notes in Computer Science*, pages 133–153, Pisa, Italy, August 2003. Springer-Verlag.

A. Wassyng, M. Lawford, and X. Hu.

Timing tolerances in safety-critical software.
In J. Fitzgerald, I.J. Hayes, and A. Tarlecki, editors, *FM 2005: Formal Methods: International Symposium of Formal Methods Europe Proceedings*, volume 3582 of *LNCS*, pages 157 – 172, Newcastle, UK, July 2005. Springer-Verlag.