Reflections on, and predictions for, support systems for the development of programs

Cliff B. Jones

Computing Science
Newcastle University

ASE 2008-09-19

Instead of conventional “Thank you for invite”

Huge Thank you!

My first time at an ASE:
see real conversation between different approaches!

Contents

Me + FM support tools

- contributed to transition from VDL to VDM (language description)
  - we wrote large (including PL/I) definitions with minimal tooling
  - experience: problems mainly hit us when changes made!
  - originated much of "VDM" as for program development
- used support systems since Jim King's "Effigy"
  - I worry they lock user in to one method
  - suspect they constrain thought
  - (but used Effigy for top-down design!)
- "Formal Development Support System" IBM Hursley (1970s)
  - it was so rigid, even I couldn't use it!
- more public is the mural system (below) Manchester (1980s)
- observed use of many TP systems
  - have seen people "hack" without understanding *what* they are proving
- Rodin Toolset (below)

mural [JJLM91]

- order of proof steps was very flexible
- a “logical frame” (e.g. used for LPF)
- focus on building theories
- but only minimal automatic support
- best seen as a UI experiment
- built from VDM spec
- implemented in SmallTalk’80 (turned out to be a mistake)
- kept multiple proof attempts — difficult to delete!
- book now on the web
- yes, it contains the VDM spec (evolved)

Rodin ToolSet

- (EU) project developed tools “Rodin ToolSet”
- open source - available from SourceForge
- kernel + plugins
- Eclipse based
- one key advantage: background proving
- also: nice work on computing impact of changes (minimise re-proof)
- now being used in the (EU) DEPLOY IP project
- “road map” discusses plans; invites input
- tool engenders an approach: everything in Contexts/Machines

Contents

1. Background
2. Arguments
3. An example: ACMs
   - Where to start – a specification
   - Splitting atoms (gently) in abstract state
   - Retaining less history
   - The four-slot representation
   - Conclusions
4. Overall conclusions/summary

Claim: (software) design is hard

- (Yes, I know this is stating the obvious!)
- requires a strange mixture of important (big) insights and detailed symbol pushing
- layers of *abstraction* (backed up by formal rules) are all we've got!
- (for many reasons) we must take our own medicine
    - reluctance to so do: Effigy, . . .
- we must be seen to take our own medicine

Belief: there is a long way to go

- no current "formal methods support system" gives software engineers anything like the support given to hardware designers by their CAD systems
- destruction of design history is intellectual vandalism
- current programming languages are ill-suited to documenting design
- have to stop trying to build "complete" support systems
- build/link components
- care! there are pitfalls here (e.g. different logics)
- "whole" system includes (IMHO) tracking all versions
- . . . and all tests on all versions

Thesis: level of generality

- there are all sorts of things I'd like to prove
- mistake to fix on one method (example below)
- but want more than a general purpose TP system
- there is no point in proving all of the verification conditions for one version of a program and then running a different (buggy) version — so systems have to control all versions, tests, verifications, changes etc.
- might call it a "method frame"
- this can present problems diagnostics (and performance)

My hope for AI contribution

- (discussions with Ireland/Bundy)
- they planning to “mine proofs”
- loses info on how created (order) — cf. mural view
- info on failed attempts long discarded!
- at detailed level, can’t trace what “copied” from where
- system learns high-level strategy (not tactics?)

Argument: don't get locked into "legacy code" corner

- (I'm aware there are a lot of "testing" folk at ASE)
- BTW: I started out in IBM's Product Test division
- ideas like "abstract interpretation" ("symbolic execution"): making real progress for non-trivial systems
- handling "legacy" systems presents another set of challenges — here the aim is to accumulate information such as avoidance of certain sorts of bad behaviour; again, such hard won information should not be discarded
- even if can't work on "green fields" projects, look at rational reconstructions

Theses

- model checking not only needs “abstraction” but it should be equipped to use ones that are available from design
- there are enough common problems between the various sorts of tool that interfacing components is imperative — apart from simple syntactic interfaces
- . . . much in the style of the EPFL paper (Beyer?) Wednesday morning (“predicate abstraction” + “explicit analysis”)
- such integration can pose hard semantic challenges

Idea: direct support for SOS

was almost subject of ASE conf paper (now [HJ08])

- do not have complete axiom systems for any widely used programming language (by a big margin)
- might therefore have to reason from, say, an operational semantics
- our paper builds on mural approach
- obviously use Floyd/Hoare-like rules is applicable
- in fact, would be nice if this system supports justification of such rules

Broader worries: industrial perspective

- getting the “right” specification [JHJ07] for a non-trivial system is at least as much an issue as showing that a design matches its specification
- even during design, everything will change (in fact, designing for flexibility is often more important than aiming for efficiency) — systems must maximise what is preserved over such changes
- we have to build our tools so that they can interface with whatever in-house engineering systems are being used by organisations we expect to adopt our formal tools
- . . .

Contents

1. Background
2. Arguments
3. An example: ACMs
   - Where to start – a specification
   - Splitting atoms (gently) in abstract state
   - Retaining less history
   - The four-slot representation
   - Conclusions
4. Overall conclusions/summary

Key abstractions

Argument: flexibility on methods

- Pre/post-conditions (as in VDM/B/. . .)
    - design by *sequential* “operation decomposition rules”
    - Floyd/Hoare-like rules (coping with relational post-conditions)
- Rely/Guarantee “thinking”
    - not (just) a specific set of rules
    - show importance of “frames” (cf. Separation Logic)
    - using “auxiliary variables”
- Abstract objects
    - choice of abstract data objects key for specifications
    - data “reification” (classic-VDM / Nipkow’s rule)
    - link with R/G development
- “fiction of atomicity”
    - “splitting (software) atoms safely” [Jon07]
    - cf. database transactions [JLRW05], . . .
    - cf. POBL [Jon96]

While (operation decomposition) rule

$$\boxed{While\text{-}I}\ \frac{\begin{array}{l} S\ \textbf{sat}\ (P \land b, P \land W) \\ P \ \Rightarrow\ \delta_l(b) \end{array}}{mk\text{-}While(b, S)\ \textbf{sat}\ (P, P \land \neg\, b \land W^*)}$$

There's no reason why a system should hardwire the (standard) Hoare rule the rules should be data (to a method frame)

"posit and prove" is one way of supporting design;
"Verified by Construction" has been shown to be viable for large systems

One R/G rule

$$\boxed{Par\text{-}I}\ \frac{\begin{array}{l}\{P, R \vee Gr\} \vdash sl \ \textbf{sat}\ (Gl, Ql) \\ \{P, R \vee Gl\} \vdash sr \ \textbf{sat}\ (Gr, Qr) \\ Gl \vee Gr \ \Rightarrow\ G \\ \overleftharpoon{P} \wedge Ql \wedge Qr \wedge (R \vee Gl \vee Gr)^* \ \Rightarrow\ Q\end{array}}{\{P, R\} \vdash mk\text{-}Par(sl, sr)\ \textbf{sat}\ (G, Q)}$$

. . . and there are lots more where this one came from

Subtle link between R/G and data reification

cf. [Jon07]

- in $FINDP$
    - we have $t \leftarrow min(t, local)$ in n parallel processes
    - assuming we don't want to "lock" t
    - need a representation that preserves R/G conditions
    - simple to represent as t as $min(et, ot)$
- SIEVE
    - we have to remove an element from a set s
    - assuming we don't want to "lock" s (big!)
    - need a representation that preserves R/G conditions $s \subseteq \overleftarrow{s}$
    - (less obvious) represent s as a bit vector
- Simpson
    - extremely interesting
    - my claim: this is the essence of Simpson's contribution

ACMs: [JP08]

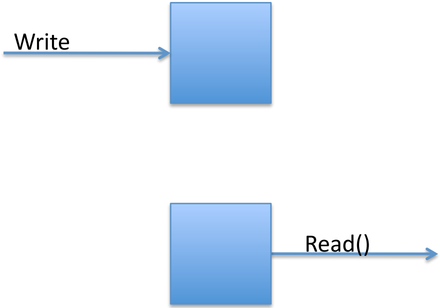
Communication (Atomic?)

Write(42)

x := Read()

ACMs

Atomic and (trying for) Asynchronous



Simpson's algorithm

- Simpson's algorithm
- several other folk still working on this
- run through my "rational reconstruction"
    - "explanation" via layers of abstraction
- essential to get the big steps right before detailed proof
- apologies for so much argument about eight lines of code . . .

formulae in small fount not meant to be read!

Specification

$$
\begin{aligned}
\Sigma^a \;::\;\; & data\text{-}w\colon Value^* \\
& fresh\text{-}w\colon \mathbb{N} \\
& hold\text{-}r\colon \mathbb{N}
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{inv}\ (mk\text{-}\Sigma^a(data\text{-}w, fresh\text{-}w, hold\text{-}r)) \triangleq \\
&\qquad\qquad fresh\text{-}w, hold\text{-}r \in \{1..\textbf{len}\ data\text{-}w\} \wedge hold\text{-}r \leq fresh\text{-}w
\end{aligned}
$$

$$\sigma_0^a = mk\text{-}\Sigma^a([\mathtt{x}], 1, 1)$$

```
while true do
    $start\text{-}Write(v\colon Value)\colon data\text{-}w \leftarrow data\text{-}w \curvearrowright [v]$;
    $commit\text{-}Write()\colon fresh\text{-}w \leftarrow$ len $data\text{-}w$
  od
while true do
    $start\text{-}Read()\colon hold\text{-}r \leftarrow fresh\text{-}w$;
    $end\text{-}Read()r\colon Value\colon r \leftarrow data\text{-}w(i)$ for some $i \in \{hold\text{-}r..fresh\text{-}w\}$
  od
```

Example 3

$$\begin{array}{lcl}
\mathit{start\text{-}Read}() & .. & \mathit{mk}\text{-}\Sigma^a([\mathrm{x}], 1, 1) \\
\mathit{start\text{-}Write}(\mathtt{y}) & .. & \mathit{mk}\text{-}\Sigma^a([\mathrm{x}, \mathrm{y}], 1, 1) \\
\mathit{commit\text{-}Write}() & .. & \mathit{mk}\text{-}\Sigma^a([\mathrm{x}, \mathrm{y}], 2, 1) \\
\mathit{start\text{-}Write}(\mathtt{z}) & .. & \mathit{mk}\text{-}\Sigma^a([\mathrm{x}, \mathrm{y}, \mathrm{z}], 2, 1) \\
\mathit{commit\text{-}Write}() & .. & \mathit{mk}\text{-}\Sigma^a([\mathrm{x}, \mathrm{y}, \mathrm{z}], 3, 1) \\
\mathit{end\text{-}Read}() & .. & r \in \{\mathtt{x}, \mathtt{y}, \mathtt{z}\} \\
\mathit{start\text{-}Read}() & .. & \mathit{mk}\text{-}\Sigma^a([\mathrm{x}, \mathrm{y}, \mathrm{z}], 3, 3) \\
\mathit{end\text{-}Read}() & .. & r = \mathtt{z}
\end{array}$$

Specification in terms of four sub-operations ($Write$)

Atomic operations — therefore pure pre/post specification

```
while true do
    $start\text{-}Write(v\colon Value)\colon data\text{-}w \leftarrow data\text{-}w \curvearrowright [v];$
    $commit\text{-}Write()\colon fresh\text{-}w \leftarrow \textbf{len}\ data\text{-}w$
  od
||
⋮
```

```
$Write(v\colon Value)$
  $start\text{-}Write(v\colon Value)$
    wr $data\text{-}w$
    post $data\text{-}w = \overleftarrow{data\text{-}w} \curvearrowright [v]$
  $commit\text{-}Write(v\colon Value)$
    rd $data\text{-}w$
    wr $fresh\text{-}w$
    pre $data\text{-}w(\textbf{len}\ data\text{-}w) = v$
    post $fresh\text{-}w = \textbf{len}\ data\text{-}w$
```

Specification in terms of four sub-operations ($Read$)

```
$\vdots$
$||$
while true do
        $start\text{-}Read()\colon hold\text{-}r \leftarrow fresh\text{-}w;$
        $end\text{-}Read()r\colon Value\colon r \leftarrow data\text{-}w(i)$ for some $i \in \{hold\text{-}r..fresh\text{-}w\}$
    od
```

```
$Read()r\colon Value$
    local $hold\text{-}r\colon \mathbb{N}$
    $start\text{-}Read()$
        wr $hold\text{-}r$
        rd $fresh\text{-}w$
        post $hold\text{-}r = fresh\text{-}w$
    $end\text{-}Read()r\colon Value$
        rd $data\text{-}w, fresh\text{-}w$
        post $\exists i \in \{hold\text{-}r..fresh\text{-}w\} \cdot r = data\text{-}w(i)$
```

General messages

- note “algorithmic” specification
- “fiction of atomicity”
  - but single “atomic” variable does not cover all behaviour
- “frames” (for rd/wr access)
  - plus “local”
- data abstraction

Splitting atoms in Σ^a (Write)

Accept overlap (only read/write) — therefore rely/guarantee

$\mathit{Write}(v\colon \mathit{Value})$

  $\mathit{start\text{-}Write}(v\colon \mathit{Value})$

    rd $\mathit{fresh\text{-}w}$

    wr $\mathit{data\text{-}w}$

    rely $\mathit{fresh\text{-}w} = \overleftarrow{\mathit{fresh\text{-}w}} \wedge \mathit{data\text{-}w} = \overleftarrow{\mathit{data\text{-}w}}$

    guar $\{1..\mathit{fresh\text{-}w}\} \lhd \mathit{data\text{-}w} = \{1..\mathit{fresh\text{-}w}\} \lhd \overleftarrow{\mathit{data\text{-}w}}$

    post $\mathit{data\text{-}w} = \overleftarrow{\mathit{data\text{-}w}} \curvearrowright [v]$

  $\mathit{commit\text{-}Write}(v\colon \mathit{Value})$

    rd $\mathit{data\text{-}w}$

    wr $\mathit{fresh\text{-}w}$

    pre $\mathit{data\text{-}w}(\mathbf{len}\ \mathit{data\text{-}w}) = v$

    rely $\mathit{fresh\text{-}w} = \overleftarrow{\mathit{fresh\text{-}w}} \wedge \mathit{data\text{-}w} = \overleftarrow{\mathit{data\text{-}w}}$

    post $\mathit{fresh\text{-}w} = \mathbf{len}\ \mathit{data\text{-}w}$

Splitting atoms in Σ^a $(Read)$

$Read()r\colon Value$
$\quad start\text{-}Read()$
$\qquad$ **rd** $fresh\text{-}w$
$\qquad$ **wr** $hold\text{-}r$
$\qquad$ **rely** $hold\text{-}r = \overleftarrow{hold\text{-}r}$
$\qquad$ **post** $hold\text{-}r \in \{\overleftarrow{fresh\text{-}w}, fresh\text{-}w\}$
$\quad end\text{-}Read()r\colon Value$
$\qquad$ **rd** $data\text{-}w, fresh\text{-}w, hold\text{-}r$
$\qquad$ **rely** $hold\text{-}r = \overleftarrow{hold\text{-}r} \wedge \forall i \in \{hold\text{-}r .. \overleftarrow{fresh\text{-}w}\} \cdot data\text{-}w(i) = \overleftarrow{data\text{-}w}(i)$
$\qquad$ **post** $\exists i \in \{hold\text{-}r .. \overleftarrow{fresh\text{-}w}\} \cdot r = \overleftarrow{data\text{-}w}(i)$

General messages

- phasing
  - makes clear $start\text{-}Write$ cannot interfere with $commit\text{-}Write$
  - avoids implications in rely conditions
- frames plus phasing significantly simplify R/G assertions
- cf. $rely\text{-}start\text{-}Write$ on Σ^a above

Retaining less history

A data reification exercise — still very general

$$\begin{aligned}\Sigma^i \;::\;& \mathit{data\text{-}w}\colon X \xrightarrow{m} \mathit{Value}\\ & \mathit{fresh\text{-}w}\colon X\\ & \mathit{hold\text{-}r}\colon X\\ & \mathit{hold\text{-}w}\colon X\end{aligned}$$

$$\sigma_0^i = \mathit{mk}\text{-}\Sigma^i(\{\alpha \mapsto \mathtt{x}\}, \alpha, \alpha, \alpha)$$

Relating Σ^i to Σ^a

Using Nipkow's rule

$$r(\sigma_1^a, \sigma_1^i) \wedge post^i(\sigma_1^i, \sigma_2^i) \;\Rightarrow\; \exists \sigma_2^a \in \Sigma^a \cdot post^a(\sigma_1^a, \sigma_2^a) \wedge r(\sigma_2^a, \sigma_2^i)$$

$$r : \Sigma^a \times \Sigma^i \to \mathbb{B}$$

$$\begin{array}{l} r(mk\text{-}\Sigma^a(data\text{-}w^a, fresh\text{-}w^a, hold\text{-}r^a), \\ \qquad\qquad mk\text{-}\Sigma^i(data\text{-}w^i, fresh\text{-}w^i, hold\text{-}r^i, hold\text{-}w^i)) \quad \triangleq \\ \quad \textbf{rng}\; data\text{-}w^i \subseteq \textbf{elems}\; data\text{-}w^a \wedge \\ \quad data\text{-}w^a(fresh\text{-}w^a) = data\text{-}w^i(fresh\text{-}w^i) \wedge \\ \quad data\text{-}w^a(hold\text{-}r^a) = data\text{-}w^i(hold\text{-}r^i) \end{array}$$

Specifications of the sub-operations on Σ^i

Still overlapped — still rely/guarantee

```
$Write(v\colon Value)$
  **local** $hold\text{-}w\colon X$
  $start\text{-}Write(v\colon Value)$
    **rd** $hold\text{-}r, fresh\text{-}w$
    **wr** $data\text{-}w, hold\text{-}w$
    **rely** $fresh\text{-}w = \overleftarrow{fresh\text{-}w} \wedge data\text{-}w = \overleftarrow{data\text{-}w}$
    **guar** $\{\overleftarrow{hold\text{-}r}, hold\text{-}r\} \lhd data\text{-}w = \{\overleftarrow{hold\text{-}r}, hold\text{-}r\} \lhd \overleftarrow{data\text{-}w}$
    **post** $hold\text{-}w \in (X - \{fresh\text{-}w, \overleftarrow{hold\text{-}r}, hold\text{-}r\}) \wedge data\text{-}w = \overleftarrow{data\text{-}w} \dagger \{hold\text{-}w \mapsto v\}$
  $commit\text{-}Write(v\colon Value)$
    **rd** $data\text{-}w, hold\text{-}w$
    **wr** $fresh\text{-}w$
    **pre** $data\text{-}w(hold\text{-}w) = v$
    **rely** $fresh\text{-}w = \overleftarrow{fresh\text{-}w} \wedge data\text{-}w = \overleftarrow{data\text{-}w}$
    **post** $fresh\text{-}w = hold\text{-}w$
```

Specifications of the sub-operations on Σ^i

```
$Read()r\colon Value$
  $start\text{-}Read()$
    **rd** $fresh\text{-}w$
    **wr** $hold\text{-}r$
    **rely** $hold\text{-}r = \overleftarrow{hold\text{-}r}$
    **post** $hold\text{-}r \in \{\overleftarrow{fresh\text{-}w}, fresh\text{-}w\}$
  $end\text{-}Read()r\colon Value$
    **rd** $hold\text{-}r, data\text{-}w$
    **rely** $hold\text{-}r = \overleftarrow{hold\text{-}r} \wedge data\text{-}w(hold\text{-}r) = \overleftarrow{data\text{-}w}(hold\text{-}r)$
    **post** $r = data\text{-}w(hold\text{-}r)$
```

General messages

- simpler R/G because of read/write frames
- data reification
  - (potentially) reducing non-determinism
  - use of VDM's other reification rule
- still have "bold" atomicity assumptions
  - couldn't update $data\text{-}w$ atomically on any reasonable machine
- still work to be done
- role of data reification in achieving rely conditions
- Simpson's representation crucial

The four-slot representation

Focus on Simpson's inspiration

$$\begin{array}{lll}
\Sigma^r & :: & \mathit{data\text{-}w}\colon P \times S \xrightarrow{m} \mathit{Value} \\
 & & \mathit{pair\text{-}w}\colon P \\
 & & \mathit{pair\text{-}r}\colon P \\
 & & \mathit{slot\text{-}w}\colon P \xrightarrow{m} S \\
 & & \mathit{wp\text{-}w}\colon P \\
 & & \mathit{ws\text{-}w}\colon S \\
 & & \mathit{rs\text{-}r}\colon S
\end{array}$$

where (key assumptions about granularity (ρ)):

$P, S = \mathit{Token}$-**set**

$P = S$
card $P = 2$
$\rho(i) \neq i$

Specifications of the sub-operations on Σ^r

```
$Write(v\colon Value)$
  local $wp\text{-}w\colon P$
  local $ws\text{-}w\colon S$
  $start\text{-}Write(v\colon Value)$
    rd $pair\text{-}r, slot\text{-}w$
    wr $data\text{-}w$
    rely $slot\text{-}w = \overleftarrow{slot\text{-}w} \wedge data\text{-}w = \overleftarrow{data\text{-}w}$
    guar $\{(\overleftarrow{pair\text{-}r}, slot\text{-}w(\overleftarrow{pair\text{-}r})), (pair\text{-}r, slot\text{-}w(pair\text{-}r))\} \lhd data\text{-}w =$
            $\{(\overleftarrow{pair\text{-}r}, slot\text{-}w(\overleftarrow{pair\text{-}r})), (pair\text{-}r, slot\text{-}w(pair\text{-}r))\} \lhd \overleftarrow{data\text{-}w}$
    post $wp\text{-}w = \rho(\overleftarrow{pair\text{-}r}) \wedge ws\text{-}w = \rho(slot\text{-}w(wp\text{-}w)) \wedge data\text{-}w(wp\text{-}w, ws\text{-}w) = v$
  $commit\text{-}Write()$
    wr $pair\text{-}w, slot\text{-}w$
    rely $pair\text{-}w = \overleftarrow{pair\text{-}w} \wedge slot\text{-}w = \overleftarrow{slot\text{-}w}$
    guar $slot\text{-}w(pair\text{-}r) = \overleftarrow{slot\text{-}w}(pair\text{-}r)$
    post $slot\text{-}w(wp\text{-}w) = ws\text{-}w \wedge pair\text{-}w = wp\text{-}w$
```

Satisfies guarantee conditions (as well as post)

```
$Write(v: Value)$
  local $wp\text{-}w: P$
  local $ws\text{-}w: S$
    $wp\text{-}w \leftarrow \rho(pair\text{-}r)$;
    $ws\text{-}w \leftarrow \rho(slot\text{-}w(wp\text{-}w))$;
    $data\text{-}w(wp\text{-}w, ws\text{-}w) \leftarrow v$;
    $slot\text{-}w(wp\text{-}w) \leftarrow ws\text{-}w$;
    $pair\text{-}w \leftarrow wp\text{-}w$

$Read()r: Value$
  local $rs\text{-}r: S$
    $pair\text{-}r \leftarrow pair\text{-}w$;
    $rs\text{-}r \leftarrow slot\text{-}w(pair\text{-}r)$;
    $r \leftarrow data\text{-}w(pair\text{-}r, rs\text{-}r)$
```

Conclusions (on example)

- all at ASE probably accept “refinement from abstractions”
- “splitting atoms” – a new/old formal addition
- subsidiary points
  - rely/guarantee “thinking”
  - remember frame descriptions
  - combination with data reification
  - link with “phasing”
  - “auxiliary variables” + Nipkow’s rule
  - tool support
  - try to avoid “coding logic into values”
- these ideas are not (all) in any single “method”

Contents

(My) general conclusions

- OK, tools *do* matter
- no one method solves covers all problems
- must design interworking components
  - XML is not the answer!
  - generality: diagnostics via rules?
  - . . . performance via abstract interpretation?
- I hope to explore "method frame"
  - flexibility
  - way to combine
- GUI does matter
  - view onto huge data structure
  - much of which generated
  - quick/easy check to avoid wasting time trying to prove non-theorems?
- Programming Languages — part of the problem (not solution)
  - "must try harder"
  - old TOPD question

Personal preferences

- (post this meeting) I still plan to work on Verification!
- a couple more examples
  - I got into “formal methods” (1969) because of PL/I-F compiler mess
  - my attempts to prove extant programs always *failed*
  - concede: I didn’t have the good tools available here at ASE 2008
  - but: finding errors late still leaves “scrap and rework” issue
- real message: continue to search for synergy
  - I happen to be on Verification side of the fence
  - I do see the payoff with model checking etc.
  - Confess: VxC suites my personal research tastes!
- . . . and I hope to work with AI people!

References

John R. D. Hughes and C. B. Jones.
Reasoning about programs via operational semantics: Requirements for a support system.
Automated Software Engineering, 10.1007/s10515-008-0036-6, 2008.

Cliff B. Jones, Ian J. Hayes, and Michael A. Jackson.
Deriving specifications for systems that are connected to the physical world.
In Cliff B. Jones, Zhiming Liu, and Jim Woodcock, editors, *Formal Methods and Hybrid Real-Time Systems: Essays in Honour of Dines Bjørner and Zhou Chaochen on the Occassion of Their 70th Birthdays*, volume 4700 of *Lecture Notes in Computer Science*, pages 364–390. Springer Verlag, 2007.

C. B. Jones, K. D. Jones, P. A. Lindsay, and R. Moore.
mural: A Formal Development Support System.
Springer-Verlag, 1991.

C. B. Jones, D. Lomet, A. Romanovsky, and G. Weikum.
The atomicity manifesto.
Journal of Universal Computer Science, 11(5):636–650, 2005.

C. B. Jones.
Accommodating interference in the formal design of concurrent object-based programs.
Formal Methods in System Design, 8(2):105–122, March 1996.

C. B. Jones.
Splitting atoms safely.
Theoretical Computer Science, 357:109–119, 2007.

Cliff B. Jones and Ken G. Pierce.
Splitting atoms with rely/guarantee conditions coupled with data reification.
In *ABZ2008*, volume LNCS 5238, pages 360–377, 2008.

Plug!

ASE community might also like
VSTTE (because the “E” is for “experiments”)
Toronto, October 6^{th}–9^{th}

http://qpq.csl.sri.com/vsr/vstte-08