Formal Specification, Verification, and Implementation of Fault-Tolerant Systems using EventML

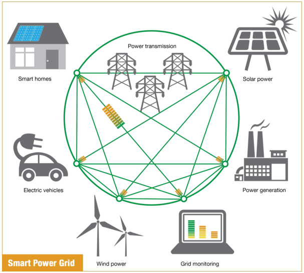
Vincent Rahli, David Guaspari, Mark Bickford and Robert L. Constable

http://www.nuprl.org

securityandtrust.lu

October 7, 2015

Distributed Systems are Ubiquitous



Correctness

What evidence do we have that these systems are correct?

Correctness

What evidence do we have that these systems are correct?

Type checking

Testing

Correctness

What evidence do we have that these systems are correct?

Type checking

Testing

Model checking

Correctness

What evidence do we have that these systems are correct?

Type checking

Testing

Model checking

Theorem proving

New Challenges

Distributed systems are hard to specify, implement and verify.

We need to tolerate failures.

It is hard to test all possible scenarios.

State space explosion using model checking.

Model checking often done on abstractions of the code rather than on the code itself.

Contributions

We use Nuprl as a specification, programming and verification language for asynchronous distributed systems.

Programming interface:
a *constructive specification language* called **EventML**

Verification **methodology**

Nuprl?

Similar to Coq and Agda

Extensional Intuitionistic Type Theory for partial functions

Consistency proof in Coq

Cloud based & virtual machines: `http://www.nuprl.org`

JonPRL: `http://www.jonprl.org`

Contributions

A **logic of events (LoE)** and a **general process model (GPM)** implemented in Nuprl.

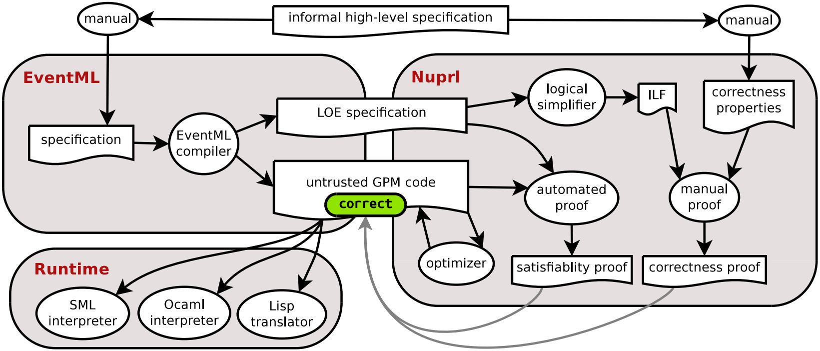
Specified, verified, and generated **consensus protocols** (e.g., 2/3-Consensus & Paxos) using **EventML**.

Aneris: a total ordered broadcast service.

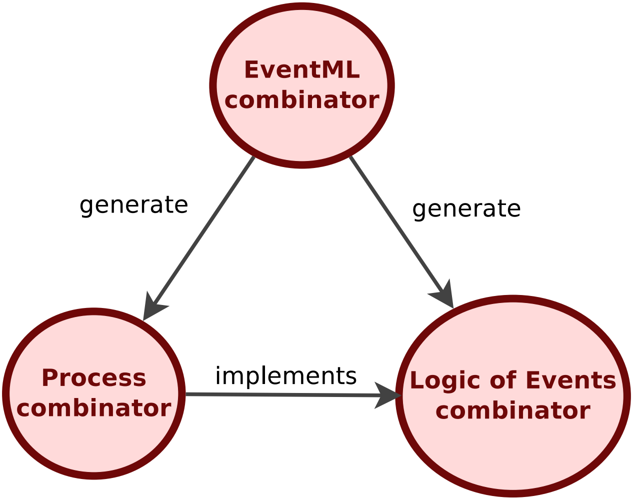
ShadowDB: a replicated database with 2 parametrizable replication protocols (PBR & SMR) built on top of Aneris.

Improved performance without introducing bugs.
We get **decent performance**.

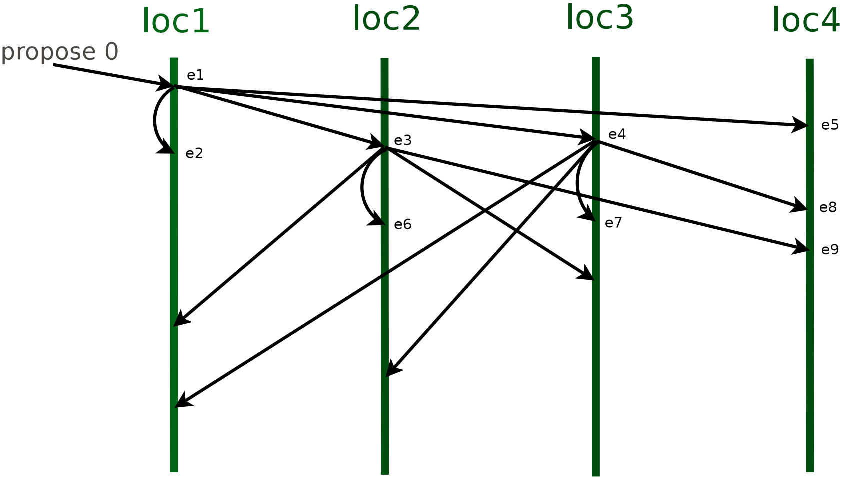
Our Methodology



Our Methodology



Event Orderings (or Message Sequence Diagrams)



Event Orderings

A dependent record

$$EO = \left\{ \begin{array}{ll} Event & : Type \\ loc & : Event \rightarrow Loc \ (e.g., \mathbb{N}) \\ info & : Event \rightarrow Info \ (e.g., input\ \ message) \\ pred & : Event \rightarrow Event \\ < & : Event \rightarrow Event \rightarrow \mathbb{P} \end{array} \right\}$$

plus some axioms

E.g., $<$ is well-founded

Processes and Observers

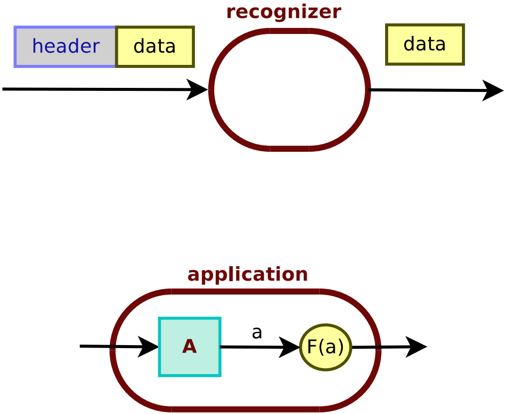
Process (GPM)

$$\texttt{corec}(\lambda P.(A \rightarrow P \times \texttt{Bag}(B)) + \texttt{Unit})$$

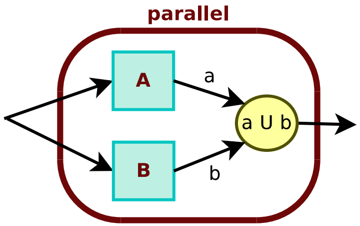
(Programmable) Observer (LoE)

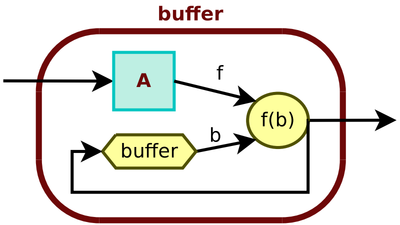
$$eo{:}EO \rightarrow e{:}Event(eo) \rightarrow Bag(B)$$

Observers

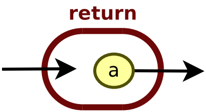


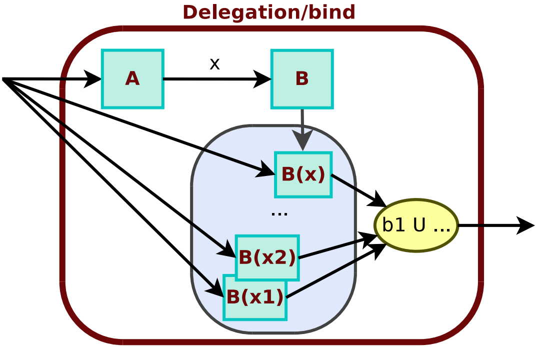
Observers





Observers





Observers in EventML

```
(* ============ Quorum: a state machine ============ *)

(* -- filter -- *)
let new_vote (n,r) (((n',r'),cmd),sender) (cmds,locs) =
  (n,r) = (n',r') & !(deq-member (op =) sender locs);;

(* -- update -- *)
let upd_quorum (n,r) loc ((nr,c),sndr) (cmds,locs) =
  if new_vote (n,r) ((nr,c),sndr) (cmds,locs)
  then (c.cmds, sndr.locs)
  else (cmds,locs);;

(* -- output -- *)
let roundout loc (((n,r),cmd),sender) (cmds,locs) =
  if length cmds = 2 * F
  then let (k,cmd') = poss-maj cmdeq (cmd.cmds) cmd in
    if k = 2 * F + 1 then decided'bcast reps(n, cmd')
    else { retry'send loc ((n,r+1),cmd') }
  else {} ;;
let when_quorum (n,r) loc vt state =
  if new_vote (n,r) vt state then roundout loc vt state else {} ;;

(* -- state machine -- *)
observer QuorumState (n,r) =
  Memory(\loc.([],[]), upd_quorum (n,r), vote'base) ;;
observer Quorum (n,r) =
  (when_quorum (n,r)) o (vote'base, QuorumState (n,r)) ;;
```

Observer Relation

$v \in (X \; eo \; e)$ written as $v \in X(e)$

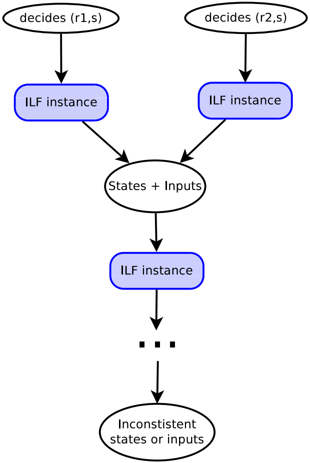
$$v \in X \,||\, Y(e) \iff \downarrow(v \in X(e) \;\vee\; v \in Y(e))$$

$$\begin{array}{c} v \in X >>= Y(e) \\ \iff \end{array}$$

$$\begin{array}{l} \downarrow\exists e' : \{e' : E \mid e' \leq_{\texttt{loc}} e\}. \\ \quad\quad \exists u : A. \\ \quad\quad\quad u \in X(e') \;\wedge\; v \in (Y \; u \; eo.e' \; e) \end{array}$$

Automated Verification

We use causal induction + inductive logical forms (ILFs) + state machine invariants + our brain



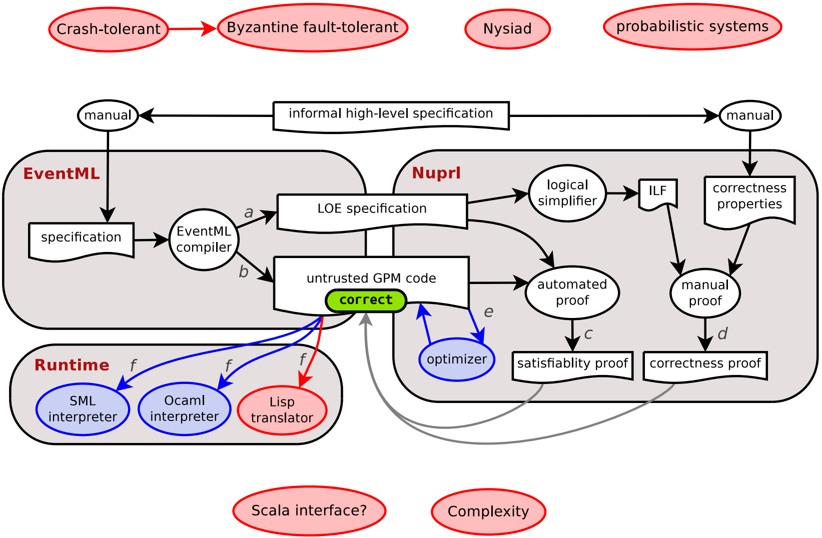
State Machines

```
import no_repeats length
invariant quorum_inv on (cmds,locs) in (QuorumState ni)
  == no_repeats ::Loc locs /\ length(cmds) = length(locs);;

import fseg
ordering quorum_fseg on (cmds1,locs1) then (cmds2,locs2)
      in QuorumState ni
  == fseg ::Cmd cmds1 cmds2 /\ fseg ::Loc locs1 locs2 ;;

progress rounds_strict_inc on round1 then round2
      in (NewRoundsState n)
    with ((n',round'),cmd) in RoundInfo
     and round => n' = n /\ round < round'
  == round1 < round2 ;;

memory rounds_mem on round1 then round2 in (NewRoundsState n)
   with ((n',round'),cmd) in RoundInfo
  == (n = n') => round' <= round2 ;;
```

Inductive Logical Forms

```
∀[Cmd:{T:Type| valueall-type(T)}]. ∀[clients,reps:bag(Id)]. ∀[cmdeq:EqDecider(Cmd)]. ∀[F:ℤ].
∀[f:headers_type{i:l}(Cmd)]. ∀[es:EO]. ∀[e:E]. ∀[i,sender:Id]. ∀[d,n,r:ℤ]. ∀[v:Cmd].
 (<d, i, make-Msg(‘‘vote‘‘;<<<n, r>, c>, sender>)> ∈ main(Cmd;clients;cmdeq;F;reps;f)(e)   1
 ⟺ loc(e) ↓∈ reps  2        ∧ i ↓∈ reps  3        ∧ (d = 0)
    ∧ (↓∃n':ℤ. ∃c':Cmd. ∃e':{e':E| e' ≤loc e }.
         ((((header(e') = ‘‘propose‘‘) ∧ <n', c'> = body(e'))
          ∨ (has-es-info-type(es;e';f;ℤ × ℤ × Cmd × Id)
            ∧ (header(e') = ‘‘vote‘‘)                                     4
            ∧ (n' = (fst(fst(fst(msgval(e'))))))
            ∧ (c' = (snd(fst(msgval(e')))))))
         ∧ (((fst(ReplicaStateFun(Cmd;f;es;e'))) < n')
         ∨ (n' ∈ snd(ReplicaStateFun(Cmd;f;es;e'))))              5
         ∧ (no Notify(Cmd;clients;f) n' between e' and e)            6
         ∧ (((<<<n, r>, c>, sender> = <<<n', 0>, c'>, loc(e)>) ∧ (e = e'))     7
          ∨ (∃r':ℤ. ∃c'':Cmd. ((<<<n, r>, c>, sender> = <<<n', r'>, c''>, loc(e)>)
              ∧ (∃e1:{e1:E| e1 ≤loc e }
                  ((((header(e1) = ‘‘retry‘‘) ∧ <<n', r'>, c''> = body(e1))
                   ∨ (has-es-info-type(es.e';e1;f;ℤ × ℤ × Cmd × Id)
                     ∧ (header(e1) = ‘‘vote‘‘)                                  8
                     ∧ (n' = (fst(fst(fst(msgval(e1))))))
                     ∧ (r' = (snd(fst(fst(msgval(e1))))))
                     ∧ (c'' = (snd(fst(msgval(e1)))))))
                  ∧ (NewRoundsStateFun(Cmd;f;n';es.e';e1) < r') ∧ (e = e1))))))))
```

What next


Crash-tolerant
Byzantine fault-tolerant
Nysiad
probabilistic systems
manual
informal high-level specification
manual
EventML
Nuprl
logical simplifier
ILF
correctness properties
LOE specification
a
specification
EventML compiler
b
untrusted GPM code
correct
automated proof
manual proof
e
c
d
optimizer
satisfiablity proof
correctness proof
Runtime
f
f
f
SML interpreter
Ocaml interpreter
Lisp translator
Scala interface?
Complexity