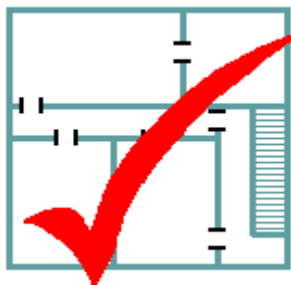
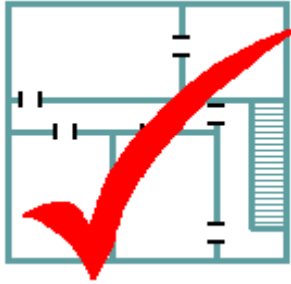


Safety Checks in a domain of collaborating(?) applications

Dr David Greaves
Systems Research
Group
djg@cl.cam.ac.uk





Project

CMI Pebbles and Goals

Groups

Cambridge: AutoHAN (SRG)

MIT CSAIL: Oxygen

Local People

David Greaves

Tope Omitola

Daniel Gordon

Atif Alvi

Computer Laboratory

- Staff
 - 30 academic staff,
 - 20 support staff, and
 - 25 affiliated research staff.
- Students
 - 100 research students (PhD),
 - 20 Diploma Students,
 - 20 Mphil in Speech and Language,
 - 3x90 Undergraduates.



Talk Overview

- Reliable Computing (historical)
- The new environment (ubicomputing)
- RBC / Pushlogic project

Highly-reliable Computer Systems

- Pre-1970, hardware used to cause the problems.
- *“A common view of software reliability is that it is ensured by solely ensuring it is free from bugs.*

...software errors are seen as design errors and cannot arise from component ageing.” -- Randell 1971

Traditional Reliability Dimensions

- High availability (many 9's)
- Graceful failure
- Automatic recover after reboot
- Hardware interlocks
- Safe defaults
- Voting

Use a *good* language

- C/C++ (eg. MISRA profile)
 - Popular, ANSI reformed, but...
 - Space station dereferences constants!
- ADA plus coding standard (eg. Ravenscar).
- Erlang
- Esterel
- Modellica, Statecharts, SysML.
- Pushlogic

Coding Standards

- MISRA-C
 - 127 rules
 - R104: “Non-constant pointers to functions should not be used”
- ADA Ravenscar 97
 - Standard templates for timing/threading
 - No use of asynchronous select
 - No use of dynamic priorities
 - many other rules

Erlang

```
1 -module(math).
2 -export([areas/1]).
3 -import(lists, [map/2]).
4
5 areas(L) ->
6     lists:sum(
7         map(
8             fun(I) -> area(I) end,
9             L)).
10
11 area({square, X}) ->
12     X*X;
13 area({rectangle,X,Y}) ->
14     X*Y.
```

```

emit WATCH_MODE_COMMAND;
loop
  trap WATCH_MODE in
    loop
      do
        <watch mode -- exit WATCH_MODE on LL>
      upto UL;
      do
        <set-watch mode>
      upto UL
    end
  end;
end;

```

```

emit STOPWATCH_MODE_COMMAND;
do
  <stopwatch mode>
upto LL;
emit ALARM_MODE_COMMAND;
loop
  trap ALARM_MODE in
    loop
      do
        <alarm mode -- exit ALARM_MODE on LL>
      upto UL;
      do
        <set-alarm mode>
      upto UL
    end
  end
end

```

Esterel


```

stategraph graph_name()
{
  state statename0 (subgraph_name, subgraph_entry_state), ... :

    entry: statement;
    exit: statement;
    body: statement;

    statement;
    ... // implied 'body:' statements
    statement;

    c1 -> statename1: statement;
    c2 -> statename2: statement;
    c3 -> exit(good);
    ...
    exit(good) -> statename3: statement;
    exit(bad) -> statename4: statement;
    ...
endstate

state statename1:
  ...
endstate

...
}

```

Hierarchical Stategraph Syntax

(SysML/
Pushlogic)

Reliable Toolchain

- Validated compiler
 - Hard to find one
 - User's program could be wrong
- Validate the object code at point of use
 - Easy to understand code, or
 - Proof carrying code

System Validation

- Simulation/emulation/exercising
 - Hours of switching switches according to printed script
 - Output as expected: yes/no ?
 - Run-time assertion monitors.
- Code coverage.
- Static checking with formal methods.

Automatic(?) Error Recovery

- BPEL4WS and STAC provide
 - exceptions
 - roll back contexts
- New language constructs:
 - Enter a new named context
 - Add a rollback command to the context
 - Pop to named context (forget associated rollbacks)
 - Abandon to named context (unwind, executing rollbacks in reverse order)

New Environment

- Many concurrent applications sharing resources,
- Dynamic population of applications and resources,
- Multiple, overlapping domains,
- Dynamic disconnection and reconnection,
- Device API evolves (new models).



Es wurde ein neues
Gerät gefunden.

Device:
Airbus A310

Soll die Auto-
Konfiguration
gestartet werden ?

API Reflection (mature).

1. Early, wire protocol RPC APIs:
 - Sun RPC, CORBA, HAVI, MOSTNET
2. Evolvable XML APIs with Reflection:
 - UPnP, EDDL, (SNMP), XMLRPC, SOAP, WSDL
3. Self-Assembling Directory Services
 - UPnP, RDF, LDAP, SCP, SSDP, INS, ...
4. Namespaces and Ontologies
 - OWL, OWL-S, DAML, RDF

A Device: A collection of Pebbles and a Canned App

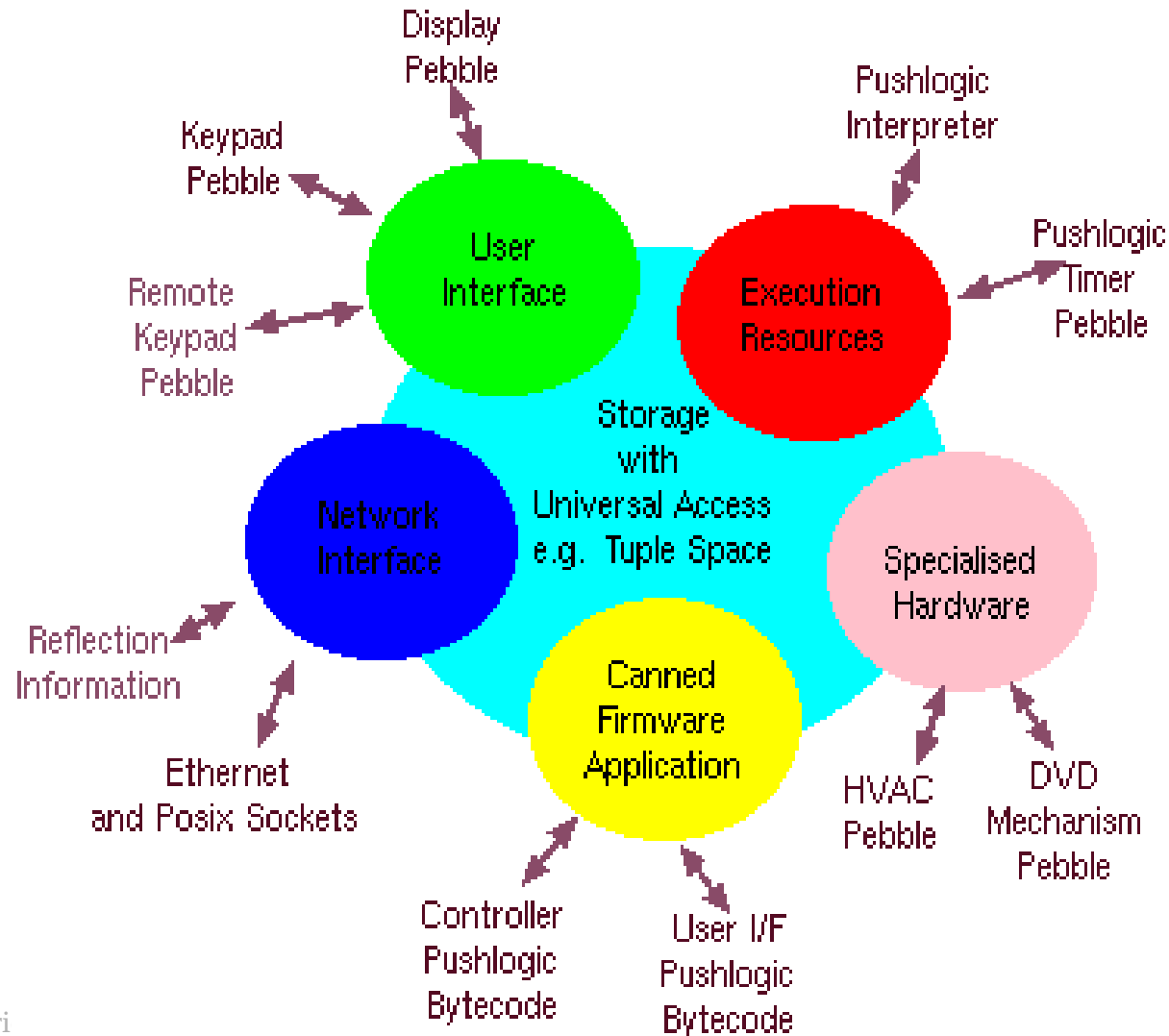
Let's look at what a modern TV set contains:

➤ 1. The following separate devices, each of which can be individually useful in a networked home:

- RF Tuner
- Colour Display
- Ni-Cam Audio Decoder
- Power Amplifier
- Surround Sound Decoder
- IR Receiver
- Teletext Decoder
- MPEG Decoder
- Programming Memory
- Front Panel User Interface

➤ 2. A canned application that joins the components.

Generic Device: Pebble View



Definitions: Pebbles and Applications

- Pebble:
 - A passive network entity (hardware or software) that implements a useful, reusable function and can register and describe itself to its environment.
- Application:
 - A proactive bundle of controlling code that connects pebbles together to achieve some user goal.
- Device:
 - A hardware entity that encompasses some pebbles and bundles

Example User Scripts

- Record both Simpson's shows tonight and charge to Pam's pay-per-view account.
- Create a video call to Peter of best quality.
- Whenever the doorbell is pressed during darkness turn on the porch light for 10 minutes.
- Do not render dialog or other popups when video recording is taking place.

Benz C200 Interior Light

- Interior light rules:
 - On when door open and auto mode (if not night)
 - On when manual mode,
 - Slow fade out when has just been on,
 - On for 15 seconds after remote door unlock at night,
 - On until engine started at night,
 - Otherwise off.
- Darkness sensor should be ignored when any interior light is on.

Code Reflection (new)

- A device must expose the proactive behaviour of its canned application(s)
 - Actual source code (constrained language)
 - Proof carrying actual source code
 - Summary of behaviour
 - E.G. I will not send control messages when I am in standby mode.
 - E.G. I am always off between 1:00 and 5:00.
- Device is banned for remote operations unless proof obligations are met.

Standing Rules (non-disjoint domains).

- No rule should issue a command under the same circumstances where another rule issues the counter-rule.
- Jonny is not allowed to spend more than 2 pounds per day on pay-per-listen.
- Fire Alarm sounding => all music sources muted.
- The front gates must always be remotely openable by some method or other.

Pushlogic

An easy-to-use programming language
that
integrates state, events and error recovery
and is amenable to automated checking,
for
Embedded Applications
and
User Scripting

Pushlogic Aims

- Source language
 - Familiar looking
 - Easy to use (imperative, C-like)
 - Automate error recovery
 - Cleanly integrate state and event
- Do many new things in the compiler
- Carry much more into the object file
- Define a run-time environment for evolvable systems

Application Scenarios

- Consumer and Home Automation
- Automotive
 - CAN car area network
 - Rail: two sets of train carriages join
- Plant and Site Control
 - Fire and intruder alarms
 - Pump and tank monitor (brewery, refinery)
- System On Chip
 - IP assembly and integration

Pushlogic Restrictions

- All integrators must be inside differentiators:

```
if (x != x_last) { sum := sum + 1; x_last := x }
```

- All pointer, arithmetic and time calculations must be reduced to undetermined boolean inputs.
- Dynamic allocation only performed at bundle load time (SPL1).
- All assertions are in CTL.

Event versus State (level)

- Differentiation:
 - If we change state we have an event
- Integration:
 - If we record the last event received we have a state
- Networks are better at carrying events ?
- Safety assertions are best written about state ?

Level and Event Expressions

- Level expressions are functions of state variables using the normal operators
- Event expressions are
 - event variables
 - differentiations
 - disjunctions of event expressions
 - certain conjunctions of level and event expressions
 - certainly not negated event expressions
- Actually defined by elaboration and not syntax

Embedded Assertions

- Three forms:
 - `always <level expression>`
 - `never <expression>`
 - `live <expression>`
- Might add ``a until b'` and other CTL operators?
- Assertions are carried through object file for domain manager use.

Pushlogic Restrictions (2)

- Restricted assignments between state and event expressions

The object-level constraints allow the following four basic source forms, or anything tantamount to them:

```
if (le) lv := le;  
if (le) ev := ee;  
if (ee) lv := le;  
if (le) lv := ee;
```

Rather than assigning to an event variable, emitting an event is possible, described in [§5.8.1](#).

Emit Statement

```
if (<ee>) emit <event-name>;  
if (<ee>) emit <event-name>(args, ...);
```

The `emit` statement is shown in the context of an `if` statement that is guarded by an event expression. Such a guard must normally exist within the surrounding program flow control in some form or another.

If the guard is a level expression, this would allow cause a nominal, continuous stream of back-to-back events to be emitted and would tend to violate idempotency.

Emit Statement (2)

The guarding context may be a level expression if the event being emitted is entirely local and the nominal stream of events is local to the current bundle and is integrated back to being a level expression in all places where it is used.

In the future, it is envisaged that closer integration with UPnP, SOAP and other device control languages will be implemented, and hence the emit statement will be implied by constructs such as

```
if (<ee>
    house.livingroom.curtains.setto(halfway) ;
```

Values

- All values are string constants or integers
- Variables may be
 - level, with safe value(s)
 - event
 - fuse
 - lock
- Variables are currently implemented as part of a global distributed tuple space.

```
pebble heatingpump = tup://128.232.7.22:1080#device;  
input heatingpump#status#temp : { unk -273..1000 };  
inout heatingpump#status#command : { off: 0..9 };
```

Mechanism View of Pushlogic

- Controlled devices can fail or self-reset to a safe value.
- Controlling scripts are *reversible*, so that a failure feeds back to the control source in a defined way.
- Feedback form is intrinsic or explicit.
- System behaves like a '*mechanism*': both the controller and the controlled can push on each other.

Reversible Operation (Pushbacks)

```
{  
  input X#x : { S: US };  
  inout Y#y : { S: US };  
  y := x;  
}
```

The problem is that if $Y\#y$ makes a unilateral change from US to S, which it is free to do, since it is an 'inout', then no push back is possible because $X\#x$ is an 'input' that cannot be changed from inside the bundle.

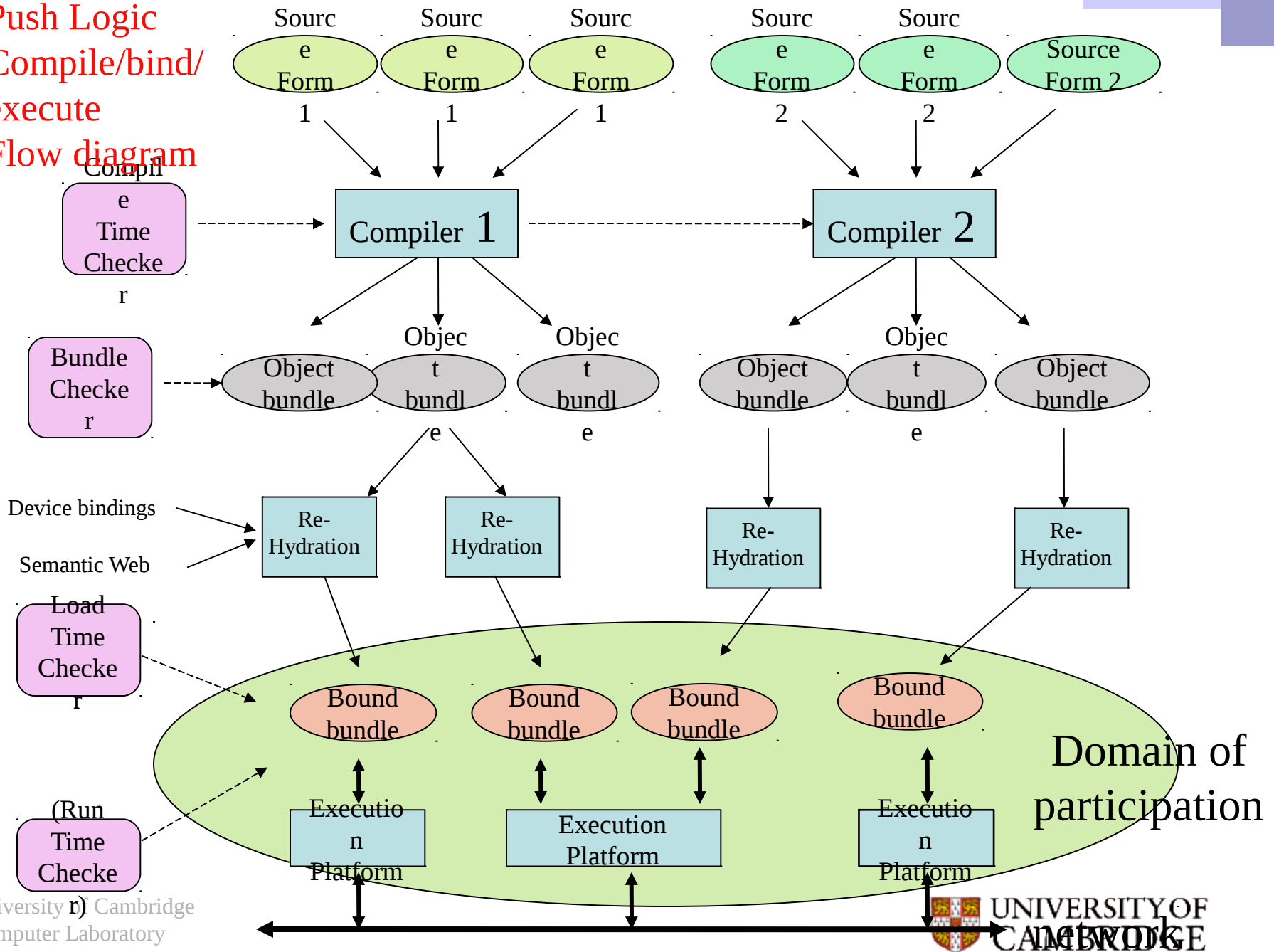
Using a *Fuse* for protection

```
input X#x : { S: US };
inout Y#y : { S: US };
fuse F1;
{ y := x; } fuse F1;

forever { wait F1; sleep_secs(5); F1 := false; }
```

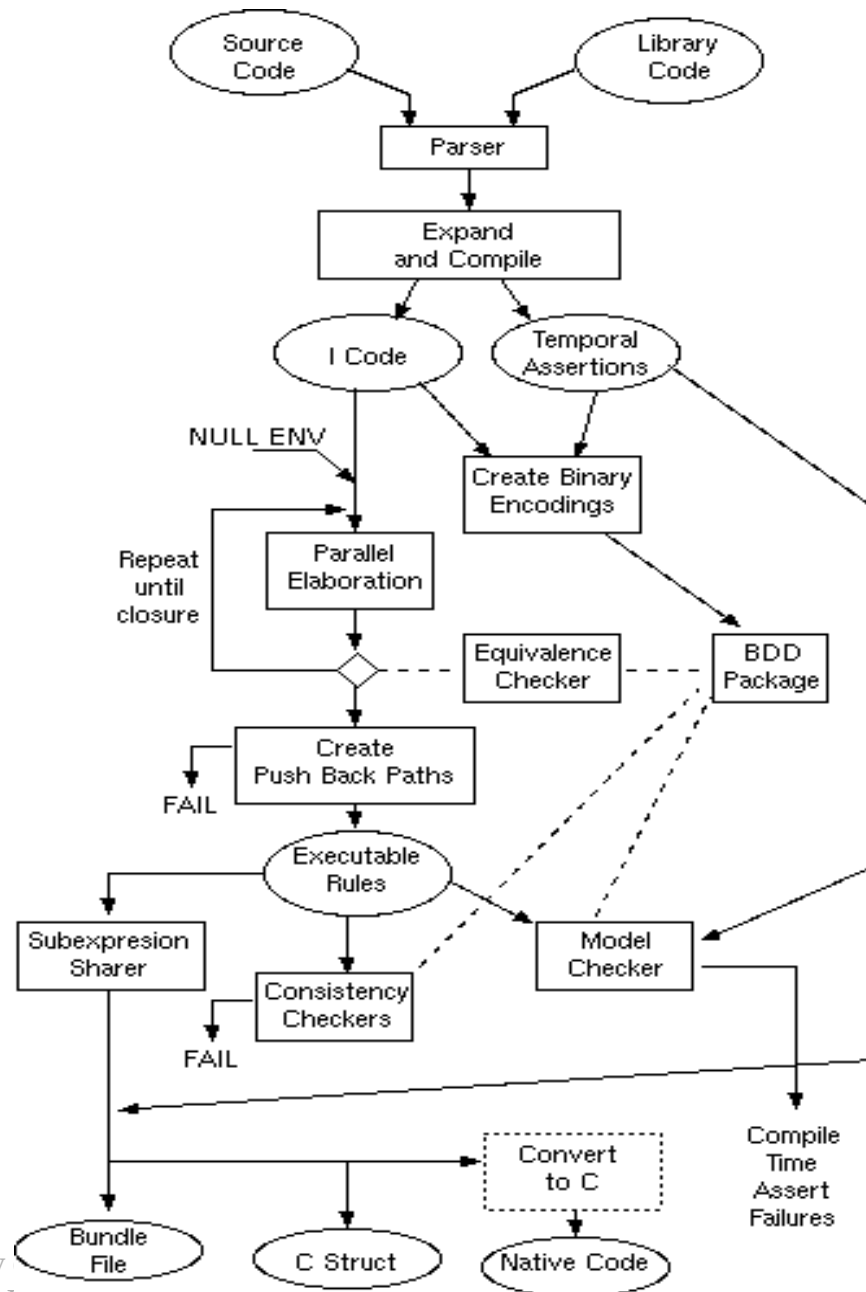
The fuse declaration defines a boolean variable with both values safe and to be set false on bundle load. The fuse statement is just syntactic sugar, because the line ``y := x; fuse F1;`` is rewritten during initial expansion as ``if (!f1) y := x;``. During pushback path creation, the fuse is chosen as

Push Logic
 Compile/bind/
 execute
 Flow diagram



Compilation Method

- Parse input file(s).
- Break threads into arcs at blocking primitives.
- Guard each arc by a runtime program counter being set to a label constant and create rules to update the program counters.
- Repeated symbolic evaluation of arc set until fixed point reached.
- Perform bundle checks using internal model checker.
- Generate declarative bytecode bundle, containing a mix of
 - Executable rules ($v := e, \dots$)
 - CTL assertions (always, live, until, ...).



Compiler Internal Flow Diagram

World and Plant Models

```
def world name()
{
  input plant#heater#setting : { off: lo, hi };
  output plant#ambient#temperature : { -273 .. 1000 };

  forever
  {
    sleep_seconds(1);
    if (setting==hi && temperature < 90) temperature += 3;
    else if (setting==lo && temperature < 90) temperature += 1;
    else if (setting==off && temperature > 0) temperature -= 1;
  }
}
```

A world model generates bytecode that does not execute on any platform, but which is used for bundle consistency checking.

World and Plant Models (2)



AP

Dynamic Participation Issues

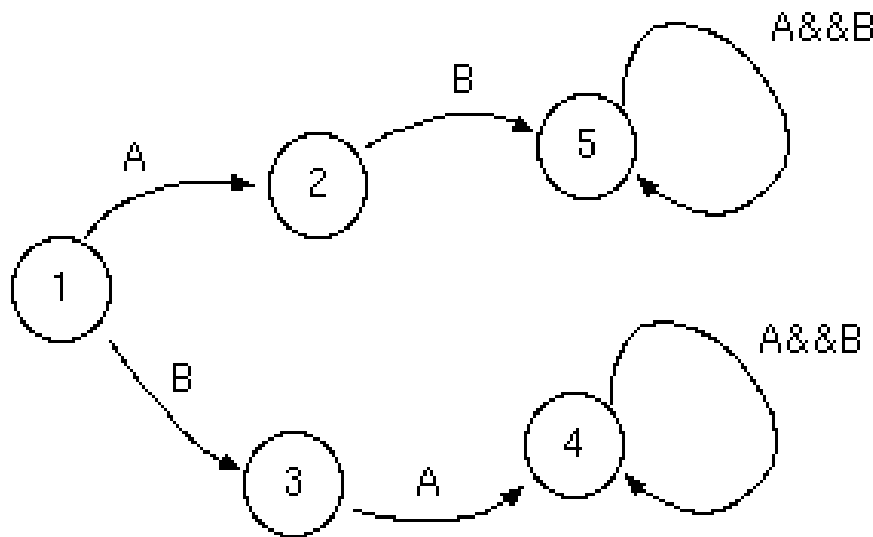
- Monotonicity over bundles present
- Monotonicity over expansion of variable range
 - recorder#quality : { hi : low medium };
- Domain create, refine, merge, divide...

Compile-Time Checks

- Safe Value Check
 - There exists a setting of the variables where each is in a safe state and all executable rules hold.
- Rule Consistency
 - No two rules will try to set the same variable to different values at any one time.
- Idempotency Check
 - No ring of rules exists that causes an observable output to oscillate when rules are obeyed more than once with the same input settings.

Compile-Time Checks(2)

- Hazard/Race Check
 - All inter-leavings of parallel statements must lead to the same result.



```
if (s==1 && A) s := 2;  
if (s==2 && B) s := 5;  
if (s==1 && B) s := 3;  
if (s==3 && A) s := 4;
```

Compile-Time Checks(3)

- Push Back Check
 - For any unilateral change in any output, to any safe value of that output, internal variables or inputs to the bundle can be changed, again to safe values, so that all rules hold
- User's Embedded and Imported CTL Expressions
 - *Safety*, *liveness* and *until* assertions may be embedded in the source.
- Monotonicity Check
 - Rules cannot cease to hold when an un-associated (separate) bundle or device leaves.

Load-Time Checks

- All the compile-time checks are repeated but over the union of participating bundles and world/plant models
- Oscillation test

```
a := b;  
b := !a;
```

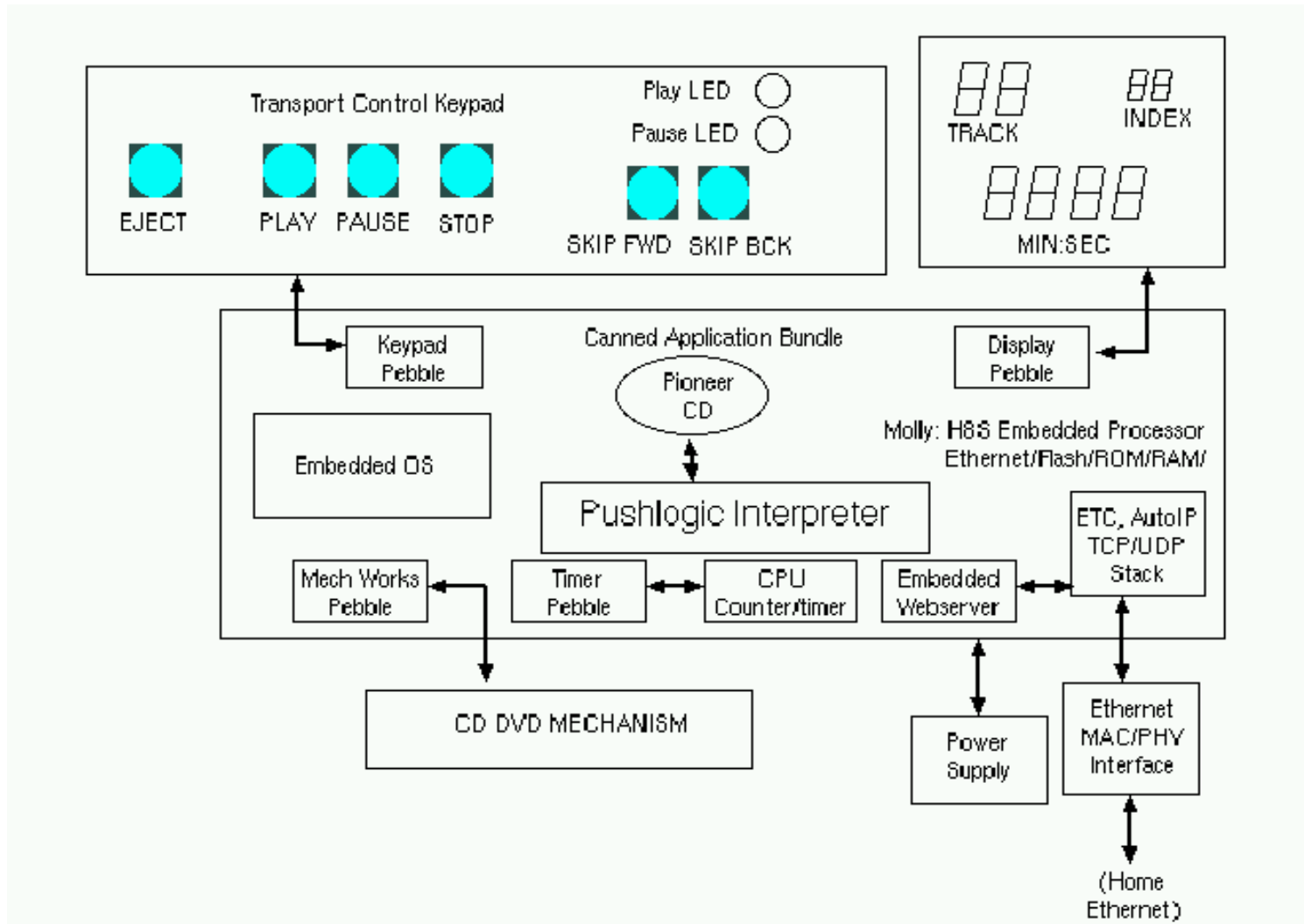
```
with Timer#Countdown if (#atimer == 0)  
{  
    #atimer := 1000; // Delay for one second  
    b_delayed = b;  
}  
a := b_delayed;  
b := !a;
```

- Re-synchronisation constraint:
 - A liveness assertion that any supposedly-coupled systems will re-synch after a network error.

Practical Work Complete

- Have built various hard and soft Pebbles
- Have a compiler
- Don't have a domain manager
- Don't have a re-hydrator
- We use the compiler as the domain checker for inter-bundle checking.

CD Player – Pebble Decomposition



CD Player: Tuple Space View

Keypad Pebble:

```
input devices#keypad#now : { Stop : Play Pause Eject Tfw Trwd};  
output devices#keypad#played : {0 : 1};  
output devices#keypad#pauseled : {0 : 1};
```

Mechanism Pebble:

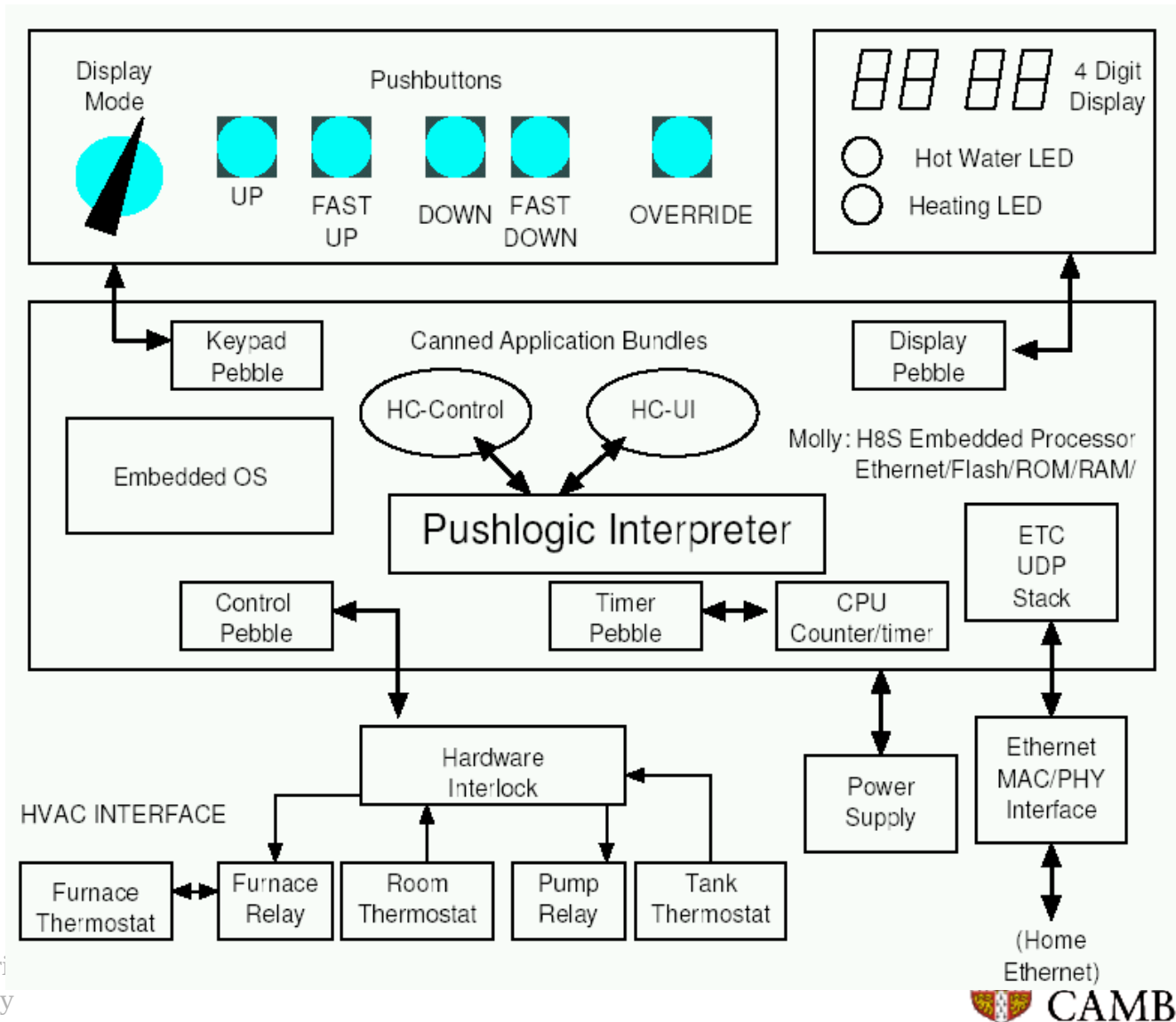
```
output works#cmd : {stop : play pause resume eject};  
output works#sink; // Destination URI for streaming audio.  
input parts#mech#stat#track : {0..99};  
input parts#mech#stat#sec : {0..59};  
input parts#mech#stat#min : {0..99};  
input parts#mech#stat#idx : {0..99};
```

Display Pebble:

```
output parts#disp#track : {0..99};  
output parts#disp#sec : {0..59};  
output parts#disp#min : {0..99};  
output parts#disp#idx : {0..99};
```



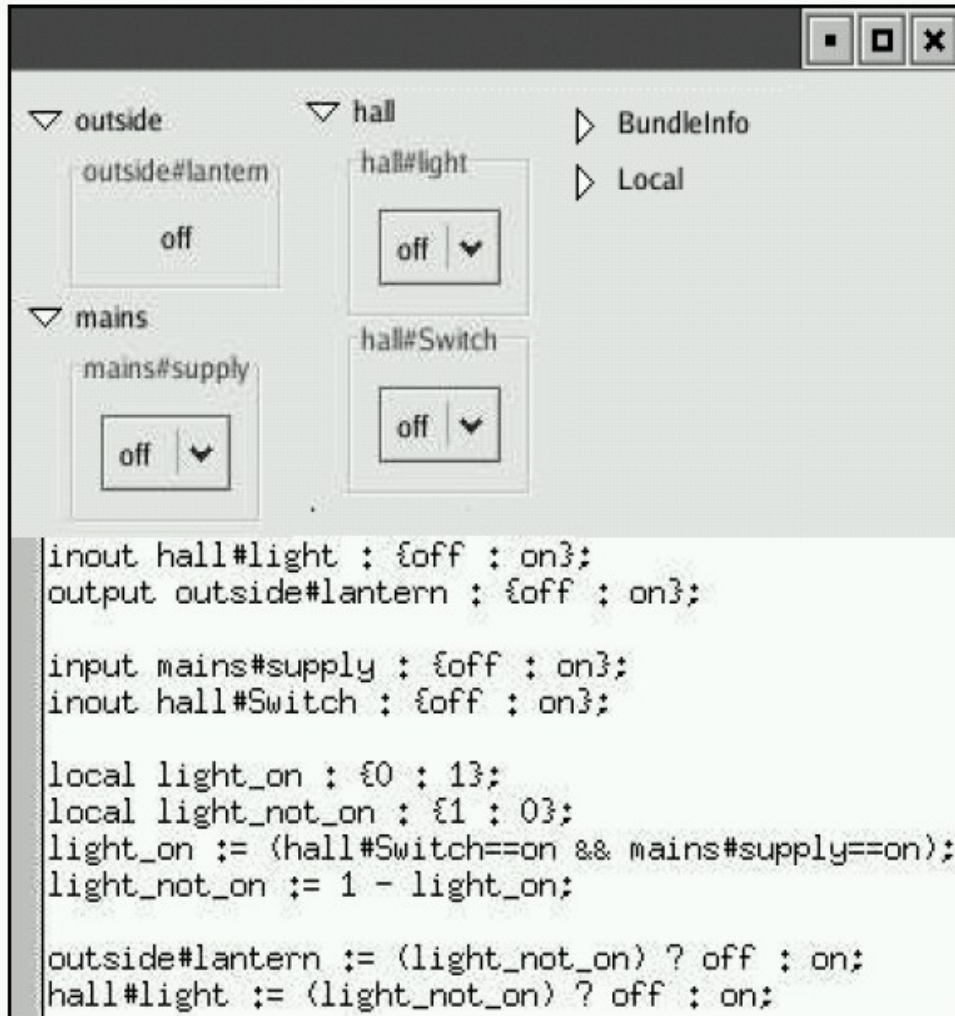
Heating Controller



Heating Controller Prototype



GUI Example



```
inout hall#light : {off : on};
output outside#lantern : {off : on};

input mains#supply : {off : on};
inout hall#Switch : {off : on};

local light_on : {0 : 1};
local light_not_on : {1 : 0};
light_on := (hall#Switch==on && mains#supply==on);
light_not_on := 1 - light_on;

outside#lantern := (light_not_on) ? off : on;
hall#light := (light_not_on) ? off : on;
```

REBOOT

Tuple Timer		
Tuple Timer#Timenow		
Timer#Timenow#second	20	Change ...
Timer#Timenow#minute	0	Change ...
Timer#Timenow#hour	0	Change ...

Tuple Timer#Timenow		
Timer#Timenow#second	20	Change ...
Timer#Timenow#minute	0	Change ...
Timer#Timenow#hour	0	Change ...

Tuple bundles

Tuple platform		
platform#etcActive	1	RO
platform#etcPort	9698	RO
platform#id	No1	Change ...
platform#type	CBG Embedded Push Logic Interpreter	Change ...
platform#substrate	UNIX	Change ...

Pebbles Alarm Clock



```
def bundle PushClock()
{
  pragma elab = false;
  inout Timer#Timencw#hour : {0..23}, Timer#Timencw#minute : {0..59};
  output Display#leds#Hour : {0..23}, Display#leds#Minute : {0..59};
  input buttons#now : {0 : stop hour alarm time minute alarm_hour alarm_minute alarm_stop};
  local Times#Alarm#Hour : {0..23}, Times#Alarm#Minute : {0..59};
  output audio#buzzer : {0 : 1};
  if (buttons#now == alarm)
  {
    Display#leds#Hour := Times#Alarm#Hour;
    Display#leds#Minute := Times#Alarm#Minute;
  }
  else if (buttons#now == alarm_hour)
  {
    Display#leds#Hour := Times#Alarm#Hour;
    Display#leds#Minute := Times#Alarm#Minute;
    Times#Alarm#Hour := (Times#Alarm#Hour == 23) ? 0 : Times#Alarm#Hour + 1;
  }
  else if (buttons#now == alarm_minute)
  {
    Display#leds#Hour := Times#Alarm#Hour;
    Display#leds#Minute := Times#Alarm#Minute;
    Times#Alarm#Minute := (Times#Alarm#Minute == 59) ? 0 : Times#Alarm#Minute + 1;
  }
  else if (buttons#now == alarm_stop)
  {
    Display#leds#Hour := Times#Alarm#Hour;
    Display#leds#Minute := Times#Alarm#Minute;
    Times#Alarm#Minute := 0;
    Times#Alarm#Hour := 0;
  }
  else if (buttons#now == stop)
  audio#buzzer := 0;
  else if ((Timer#Timencw#hour == Times#Alarm#Hour) &&
    (Timer#Timencw#minute == Times#Alarm#Minute))
  audio#buzzer := 1;
  else
  {
    Display#leds#Hour := Timer#Timencw#hour;
    Display#leds#Minute := Timer#Timencw#minute;
  }
}
```

Non-Real-Time Applications:

- EDA: components are brought together:
 - As IP and devices from many suppliers
 - Meta-info ranging from data-sheets to machine-readable formal specs
 - Often a rapid time-to-market requirement
 - Sometimes a live-insertion requirement
- Bringing a new production line online
 - Start the checking the day before!

Future Work

- Compilation of bytecode to ROM-able machine code (PIC) and integrate with CAN car node checking project.
- Some larger examples need exploration.
- Talks with industrial collaborators who might use it ?
- Bundle format optimised for incremental model checking.
- Further work on eventing (GENA) and SOAP integration.
- Complete formal semantics and reference manual
- Further work on disconnection and merging.
- Further HCI and multi-view editing projects.

Questions ?



- David.Greaves@cl.cam.ac.uk
- www.cl.cam.ac.uk/Research/SRG/HAN/Pebbles

Typical Plant Control Stack

DOMAIN-SPECIFIC APPLICATION SCRIPTING
GUI

VIEWER
TOOLS

CONTROLLING
APPLICATIONS

**SYSTEM
LIVE/SAFE
CONSTRAINTS**

DEVICE INVENTORY
DESCRIPTIONS DATABASE

STATUS
DATABASE

Network: LAN or FIELDBUS

PLANT MODEL

**(Known Sensor/Actuator Feedback Paths,
Autonomous Controller Behaviour)**

PLANT DEVICES

(Sensors, Valves, Pumps...)

Problems Doing This Today

- The main technique (so far) is symbolic model checking.
- Model checking is slow and does not scale.
- Can we do anything in real time ?
- How much can be pre-computed ?
- Can we use type-based checks?
- What about non-real time applications ?