

Formal Verification of Robot Movements *– a case study on home service robot HSR*

Moonzoo Kim and Kyo C. Kang
Software Engineering Laboratory, POSTECH

Hyoungki Lee
Samsung Advance Institute of Technology

Pohang University of
Science and Technology
(POSTECH)



Copyright © 2005
SE Lab., Dept. of CSE
POSTECH, R.O.Korea



- Introduction
- Background of Home Service Robot (HSR)
- Background of the Esterel Framework
- HSR Controller Implementation
- Formal Verification of HSR Movements
- Conclusion

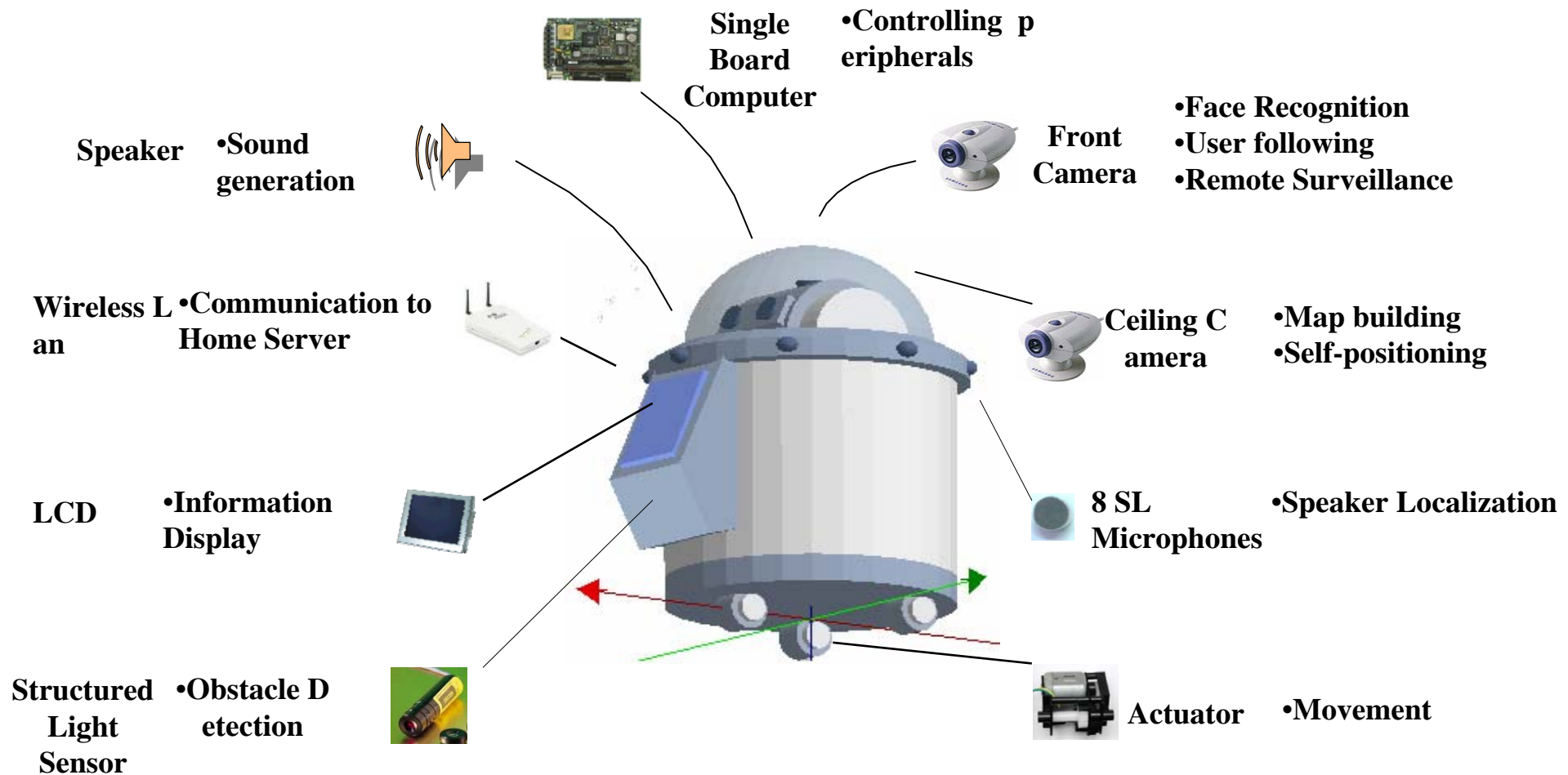


- A robot which performs daily house keeping tasks.
 - Vacuum cleaning, appliance control, home surveillance, etc.
- HSR applications are *complex*.
 - Robots are required to achieve *multiple task goals*
 - Service features tend to be *updated frequently*
 - Robot applications are *concurrent reactive* systems
 - Many applications have stringent *safety requirements*.
- HSRs tend to be developed by *integrating components in an ad-hoc way*
 - Robot developers often concentrate on technology oriented components
 - Feature interactions happen frequently
- It is a challenging task to *validate & verify (v&v) quality*

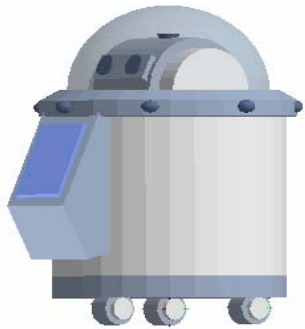
- Traditional testing and validation techniques are **not adequate** for concurrent reactive systems (e.g. robots)
- Other embedded system fields validate and verify requirements using **model checking** techniques
 - Protocol and distributed applications (*SPIN – AT&T (now NASA)*)
 - Automotive controller (*Esterel – INRIA (now Esterel tech)*)
 - Device drivers (*SLAM – Microsoft*)
- Formal v&v techniques put emphasis on **early design stage** where most design flaws are to be captured
 - Reduced overall development time by decreased testing and maintenance costs
 - Increased reliability by proved correctness



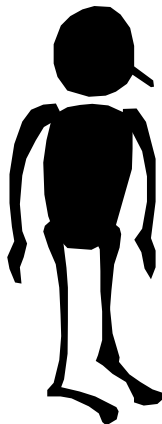
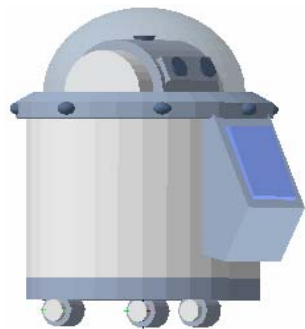
HW Components of home service robots (HSR)



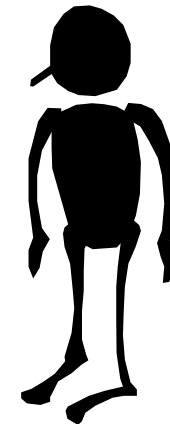
■ Call and Come (CC)



■ User Following (UF)



come



I lost you



Background of the Esterel Framework

The esterel Compiler:

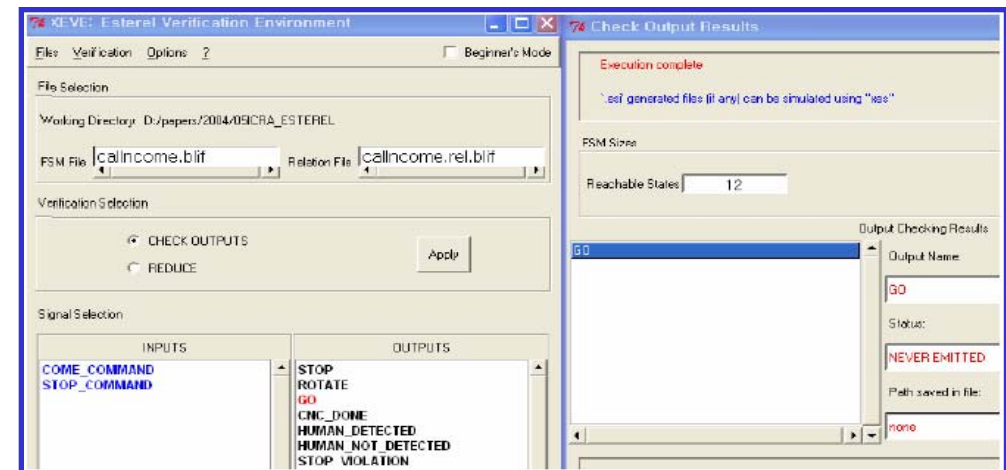
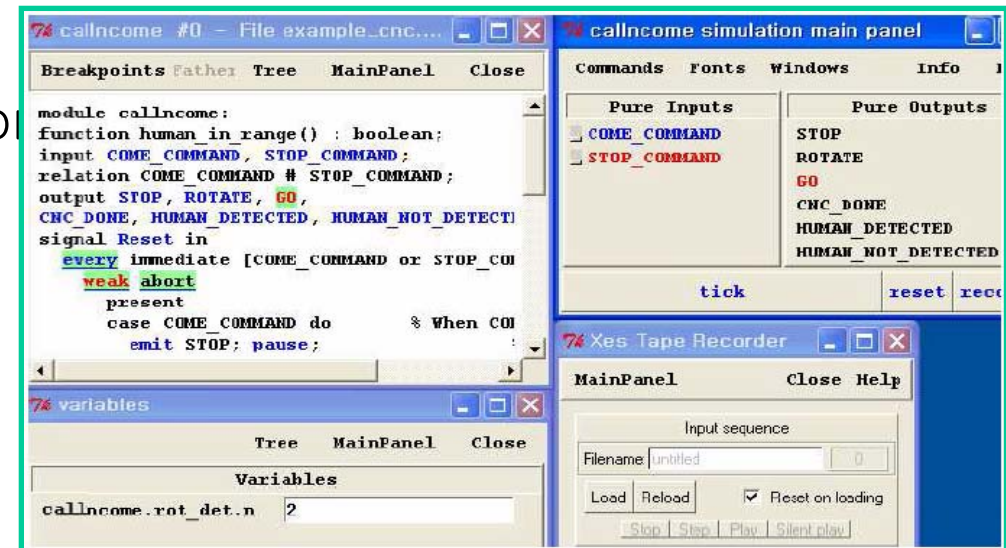
- platform neutral C code generation
- interface between Esterel and C.

The xes Graphical Simulator:

- graphical interactive simulation
- session recording/replay.

The xeve Model Checker:

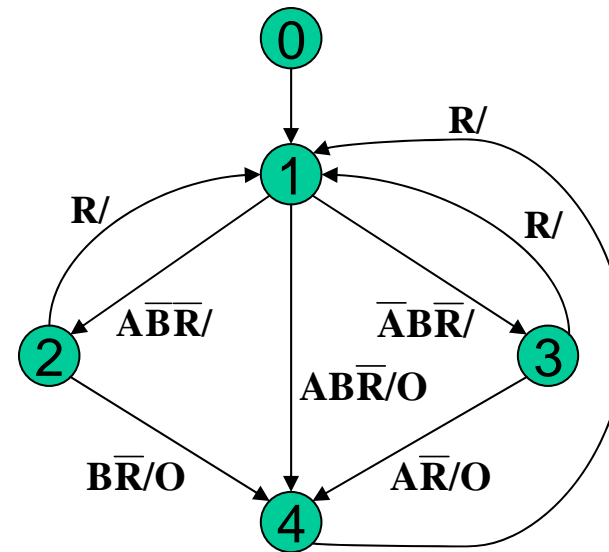
- analyzes an Esterel program.
- check presence of an output signal with given configuration of input signals.



Automata translation

loop

```
[
  wait A
  ||
  wait B
];
emit 0;
halt
every R
```



```
switch(state){
case 0: state=1; break;
case 1: if(!R)if(A)if(B) {0=1; state=4;}
        else state=2;
        else if(B)state=3; break;
case 2: if(R)state=1;
        else if(B){0=1; state=4;} break;
case 3: if(R)state=1;
        else if(A){0=1; state=4;} break;
case 4: if(R)state=1; break;
}
```



■ Structural imperative style

■ Basic constructs

- Classical control flow

$p; q, p || q, \text{loop } p \text{ end}$

- Preemption

$\text{abort } p \text{ when } S, \text{every } s \text{ do } p \text{ end every}$

- Exception handling

$\text{trap } T \text{ in } p \text{ end, exit } T$

- Division into instants:

$\text{pause, await } S$

- Signals:

$\text{signal } S \text{ in } p \text{ end, emit } S,$
 $\text{present } S \text{ then } p \text{ else } q \text{ end}$



Overview of the Previous CC Implementation

- A main control procedure for the *preemptive* CC service

```
01: class CCallComeDlg {
02:     int m_order;
03:     ...
04:     void processState() {
05:         ...
06:         switch(m_order) {
07:             case 0: STOP();
08:                 m_order++;
09:                 break;
10:             case 1: ROTATE();
11:                 m_order++;
12:                 break;
13:             case 2: static int nCount = 0;
14:                 if (abs(m_befO-curO)==0) nCount++;
15:                 else nCount = 0;
16:                 if (nCount > 2) m_order++;
17:                 break;
18:             ...
19:             case 9: CALL_N_COME_FINISHED();
20:                 m_order = -1;
21:                 break;
22:         } /* End of processState()
23:     }
```

- processState() is called periodically once in every 100 milliseconds.
- CC executes through sequential steps identified by the value of m_order
- nCount is declared as a static local variable at line 13

- This straightforward pattern is *error prone*.



Overview of the re-engineered CC Implementation

```
01:module control_plane: % Control Plane
02:input EVENT: integer;
03:output STOP,ROT,GO,CC_DONE,CS_DONE,DET,N_DET;
04:signal CALL_COME, CALL_STOP in
05:run mode_man||run cnc||run uf||run tp||run sm;
06:end signal
07:end module
08:
09:module cnc: % Call and Come service
10:function human_in_range() : boolean;
11:input CALL_COME,CALL_STOP; %come,stop commands
12:output STOP,ROT,GO,CC_DONE,CS_DONE,DET,N_DET;
13:var mv:=false:boolean,n:integer in
14:  every immediate [CALL_COME or CALL_STOP ] do
15:    present
16:    case CALL_COME do % come command
17:      mv := true;
18:      emit STOP; pause;
19:      run rot_det;
20:      ...
21:      emit CC_DONE;pause;
22:    case CALL_STOP do % stop command
23:      emit STOP;
24:      if mv=true then emit CS_DONE;
25:      else mv:=true;pause;run rot_det end if;
26:    end present;
27:    mv := false;
28:  end every
29:end var
30:end module
31:...
```

- Esterel handles a preemptive event e with a preemption operator

*EVERY e DO statements
END EVERY.*

- Interactions among Esterel modules are clearly defined via events

*PRESENT CASE e DO
statements END PRESENT*

- Submodule can be conveniently utilized

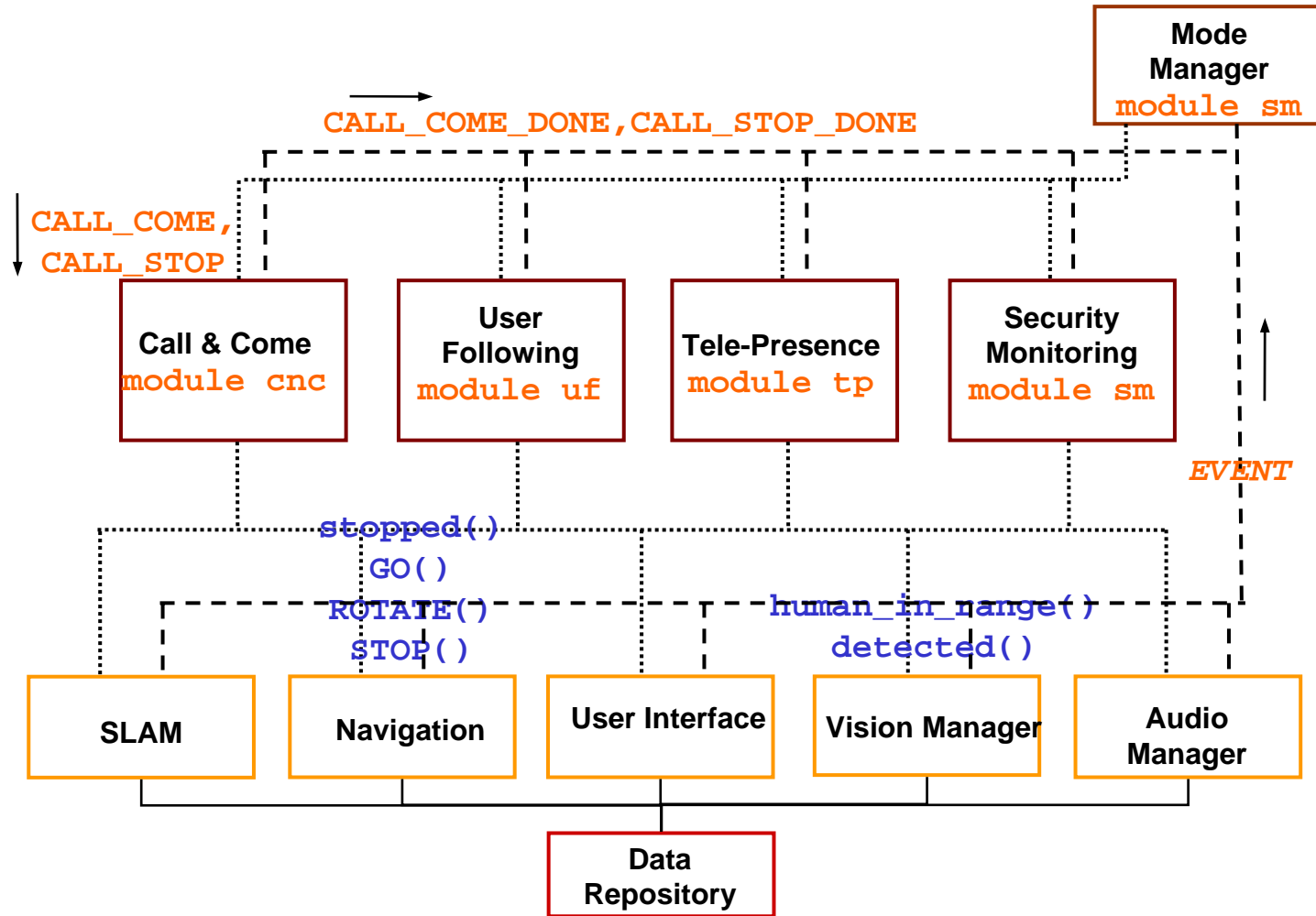
RUN module



Re-engineered HSR Architecture

*Control Plane
Implementation
in Esterel*

*Data Plane
Implementation
in C/C++*



- Stopping behaviors are *safety critical*
- Three properties on the stopping behaviors
 - P1: If a user does not give a command to the robot, the robot must not move.
 - P2: If a user does not give a “come” command, but may give a “stop” command to the robot, the robot must not move.
 - P3: If a user gives a “stop” command, the robot must stop and not move without any new command.
- We verify whether P1,P2, and P3 are satisfied in the following two cases
 - When the CC service runs solely
 - When the CC service and the UF service run concurrently



- We check **P1** by setting
 - Input signals COME_COMMAND and STOP_COMMAND as “always absent”
 - Output signal GO to check.
- Both cases satisfy **P1**

The screenshot displays the XEVE: Esterel Verification Environment interface. The main window is titled "XEVE: Esterel Verification Environment" and includes a menu bar with "File", "Verification", and "Options". The "Beginner's Mode" checkbox is checked. The "File Selection" section shows the "Working Directory" as "D:\papers\2004\OSICRA_ESTEREL", the "FSM File" as "callcome.blif", and the "Relation File" as "callcome.rel.blif". The "Verification Selection" section has two radio buttons: "CHECK OUTPUTS" (selected) and "REDUCE". An "Apply" button is located to the right. The "Signal Selection" section contains two columns: "INPUTS" and "OUTPUTS". The "INPUTS" column lists "COME_COMMAND" and "STOP_COMMAND". The "OUTPUTS" column lists "STOP", "ROTATE", "GO", "CNC_DONE", "HUMAN_DETECTED", "HUMAN_NOT_DETECTED", and "STOP_VIOLATION".

Overlaid on the right side is a "Check Output Results" dialog box. It shows "Execution complete" in red text, followed by the message: ".est generated files (if any) can be simulated using 'xse'". Below this, the "FSM Size" section indicates "Reachable States" as "12". The "Output Checking Results" section features a list box with "GO" selected. To the right of the list box are fields for "Output Name" (containing "GO"), "Status" (containing "NEVER EMITTED" in red), and "Path saved in file:" (containing "none").



Verification Results (cont.)

- The CC service satisfies $P2$, but *not CC and UF together*.
- Verification result said that *ROTATE* and *GO* could be possibly *emitted* when COME_COMMAND command was absent and STOP_COMMAND might be given
 - I.e. *feature interaction* happens
- UF should had been triggered only after a “come” command
 1. We refined CNC_DONE into CNC_COME_DONE and CNC_STOP_DONE.
 2. We modified the UF implementation so that only CNC_COME_DONE could invoke UF.
 3. After this modification, we could see that $P2$ was satisfied by the concurrent CC and UF services.



Verification Results (cont.)

■ The property P3.

- P3: If a user gives a “stop” command, the robot stops and does not move without any new command.

■ To verify P3, we need to build an *observer* to detect violation

```
01:module observer:
02:input STOP_COMMAND,COME_COMMAND,ROTATE,STOP,GO;
03:output STOP_VIOLATION;
04:weak abort
05: every immediate STOP_COMMAND do
06:     present STOP then
07:         loop
08:             present [ROTATE or GO]
09:                 then emit STOP_VIOLATION;
10:             end present;
11:             pause;
12:         end loop;
13:     end present
14:     emit STOP_VIOLATION;
15: end every
16:when COME_COMMAND;
17:end module
```



- Coordinating diverse robot services is a significant challenge
 - We need to apply systematic development and V&V framework.
- We re-engineered existing C/C++ source code into Esterel code for
 - Reactive operators
 - Compositionality
- Through the re-engineering, we could
 - Detect/fix a feature interaction problem
 - Provide high reliability to working controller
- Future Works
 - Resource management problem under heavy resource utilization.
 - Applying Monitoring and Checking framework

