
Formal Construction and Verification of Home Service Robots: A Case Study

Moonzoo Kim and Kyo Chul Kang

CSE Dept.

Pohang University of Science & Technology, Korea

<http://www.postech.ac.kr/~moonzoo>

moonzoo@postech.ac.kr



■ What we have done

- To re-engineer a prototype of home service robot (SHR100) developed by Samsung Advanced Institute of Technology (SAIT) for higher reliability
 - Re-designed software architecture
 - Re-implemented the core part in Esterel
 - Verified critical safety properties by model checking and found/fixed a feature interaction problem

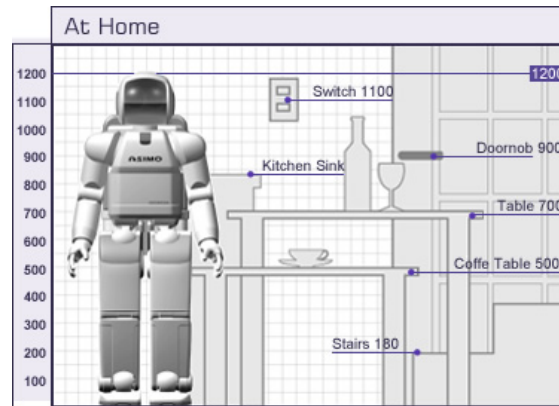


- Background on Home Service Robots
- Project Background
- Description of SHR100
- Re-engineering of SHR100
- Formal Verification of SHR100
- Lessons Learned
- Conclusion

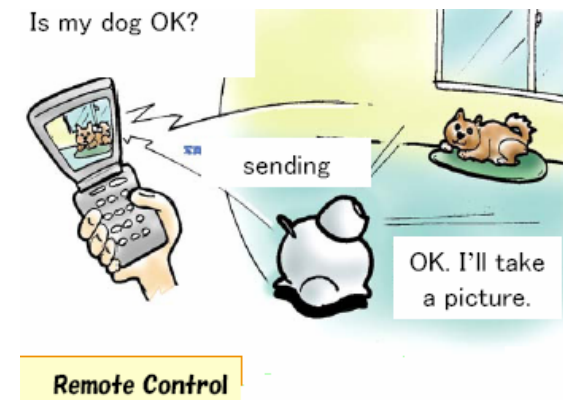


Home Service Robots

- Designed for providing various services to human user
 - Service areas : home security, patient caring, cleaning, etc



* The above heights are examples to serve as a reference(mm).

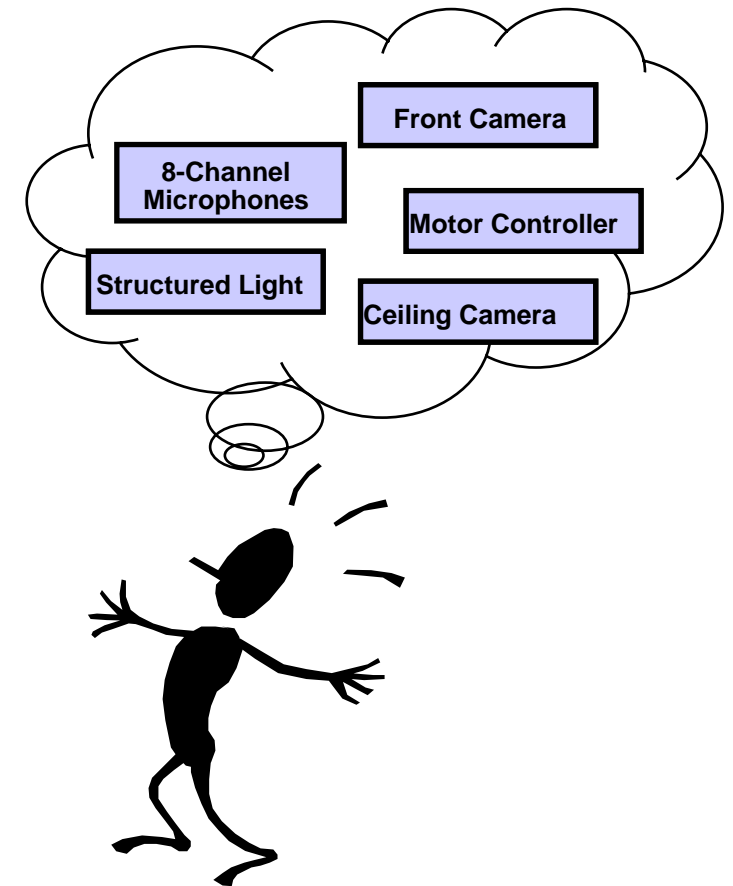
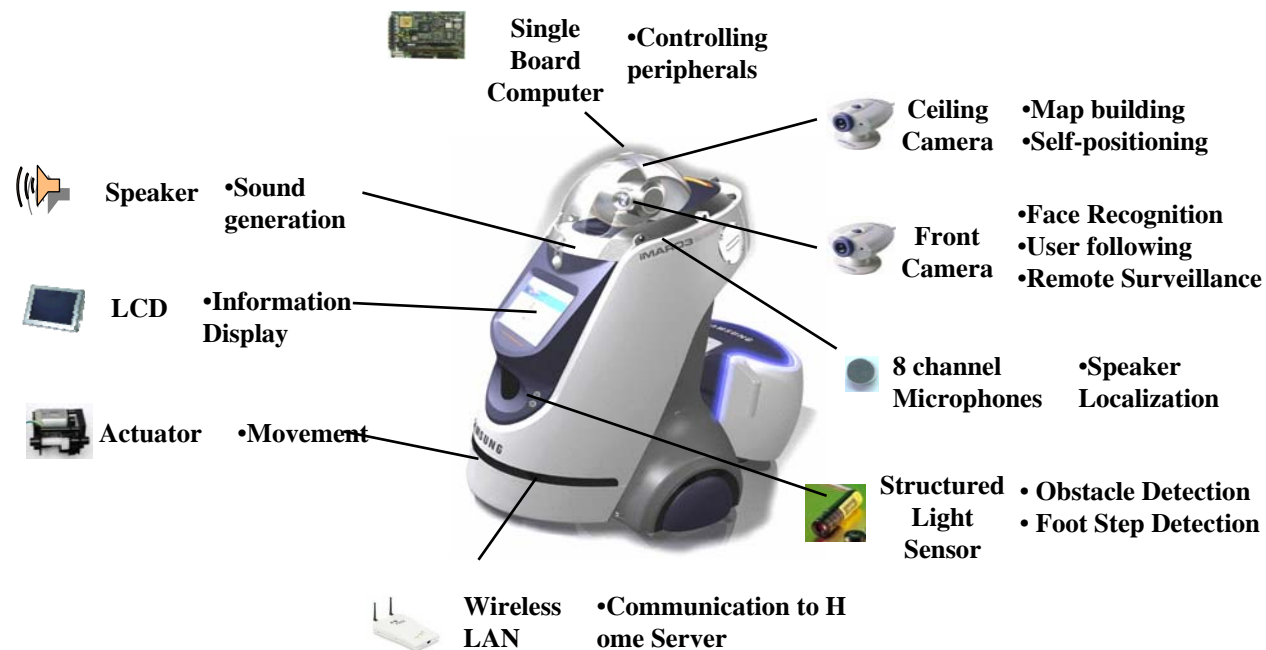


- SAIT started development of SHR00 from 2002
 - ✚ 4 separate teams (13 persons)
 - Vision recognition, speech recognition, simultaneous localization and mapping (SLAM), actuator
- Both SHR00 and SHR50 suffered feature interaction problems
 - ✚ SAIT decided to develop SHR100 from scratch
- SAIT requested POSTECH to improve the reliability of SHR100 in **six months**
 - ✚ SHR100 is written in 17K line of C/C++



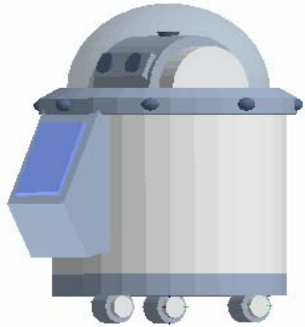
Components of SHR100

- Robots are created based on various **technical components**
 - Speech recognizer, vision recognizer, actuator, etc

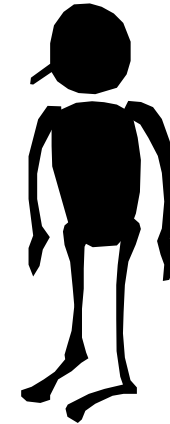


Two Basic Services of SHR100

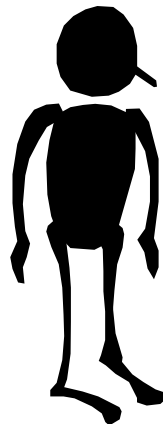
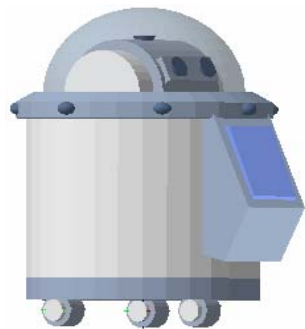
■ Call and Come (CC)



come



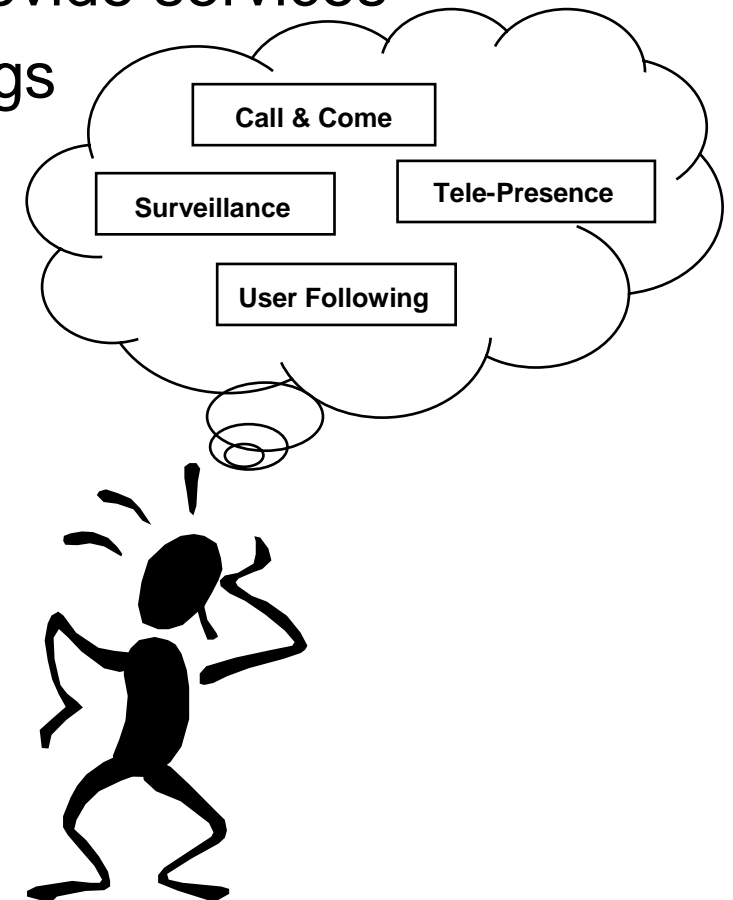
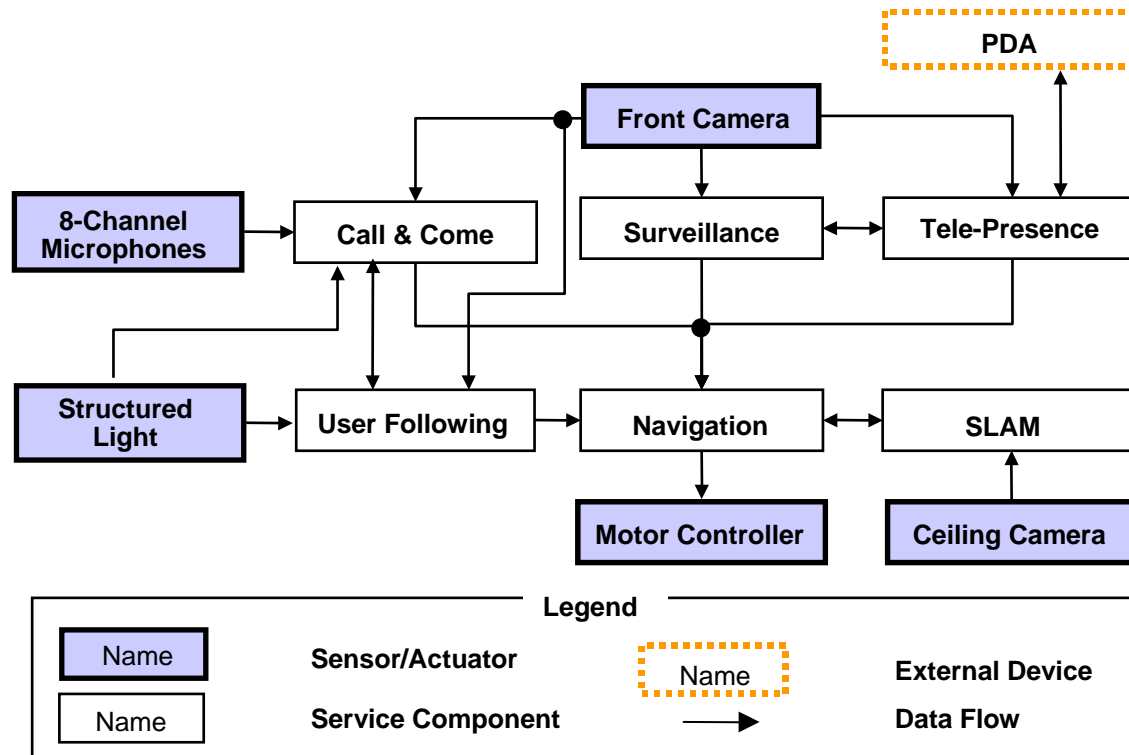
■ User Following (UF)



I lost you



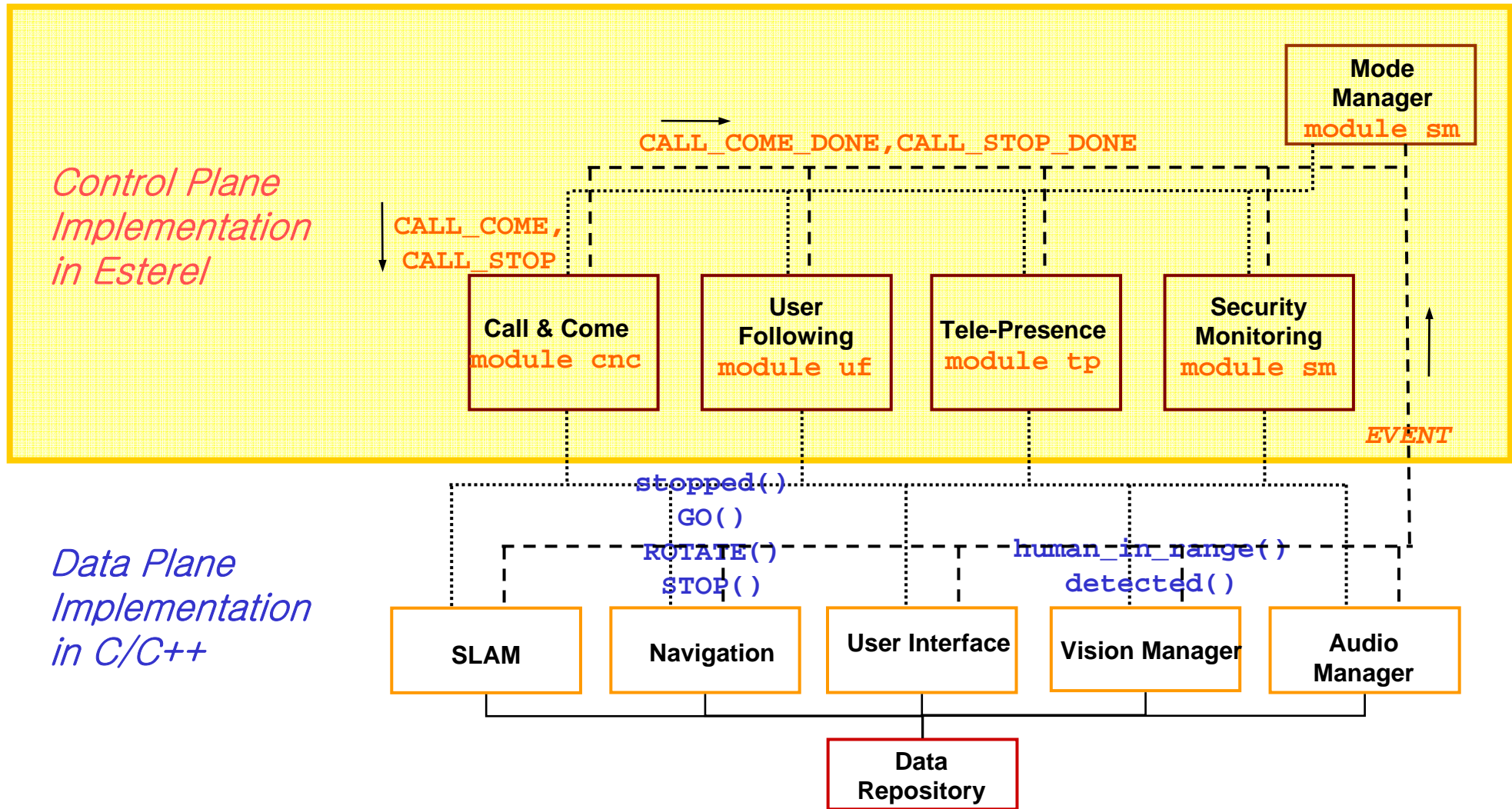
- Robot developers concentrate on technical components only, resulting in **integration in an ad-hoc and bottom-up way**
 - Difficult to coordinate components to provide services
 - Difficult to track down the sources of bugs



- To provide **hierarchical and modular software architecture**
 - Top-down global views
 - Visualization of component interactions
 - [see ICSE05 paper]
- To apply **formal construction & verification** to the core of SW
 - Rigorous and automated debugging support
 - Explicit interaction mechanism among components
 - Compact and easy-to-understand code



Re-engineered SHR100 Architecture



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:     }
```

New
Com-
mands

- 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

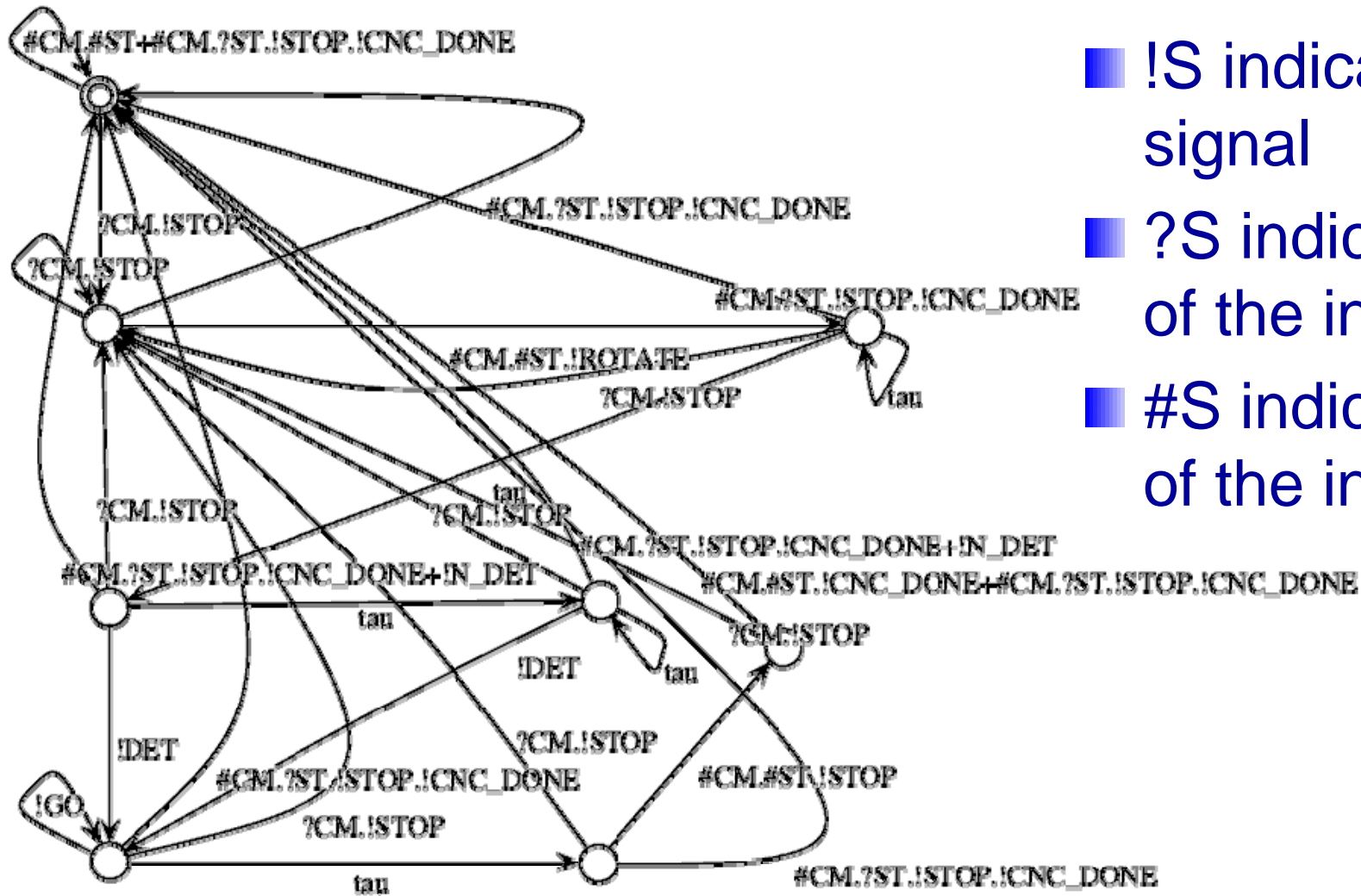
*PRESENT CASE e DO
statements END PRESENT*

- Submodule can be conveniently utilized

RUN module



Behavior of CC



- !S indicates output signal
- ?S indicates presence of the input signal S
- #S indicates absence of the input signal S



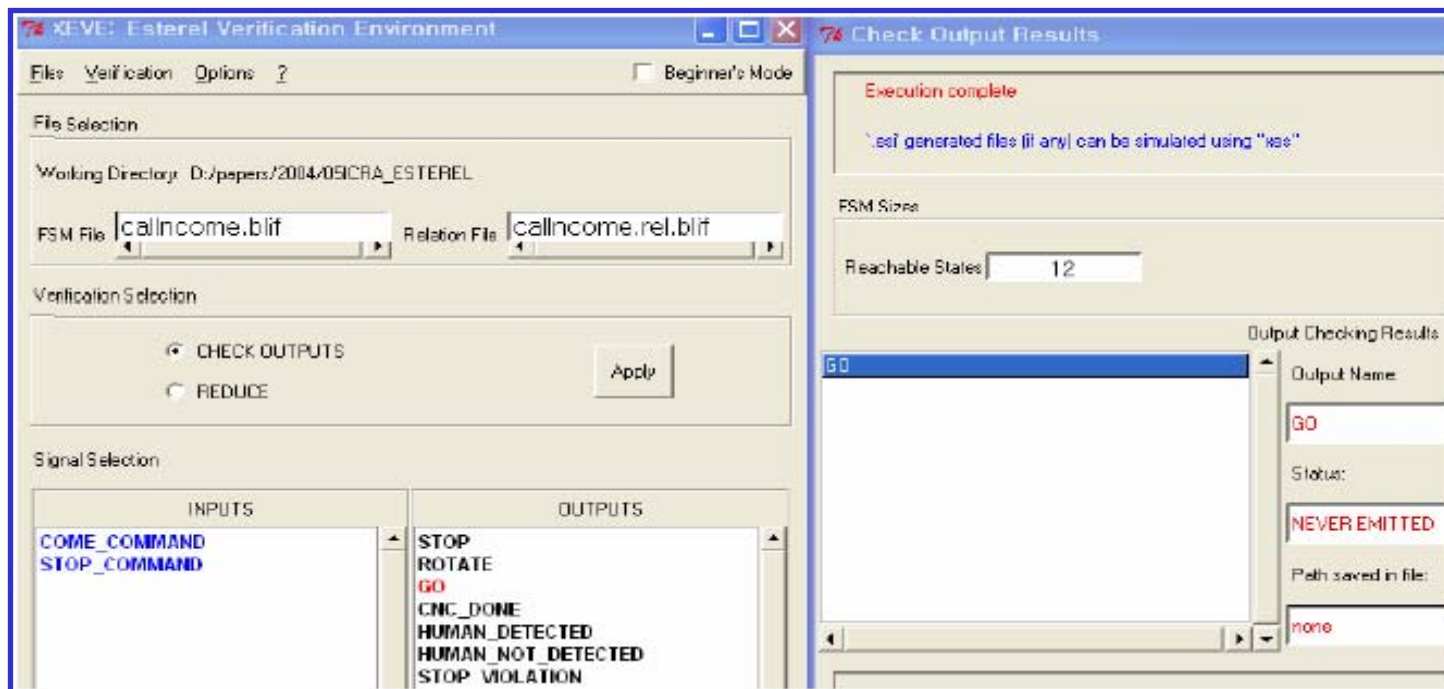
Formal Verification of SHR100

- 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



Verification Result I

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



Verification Result II

- 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.



■ The property P3.

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

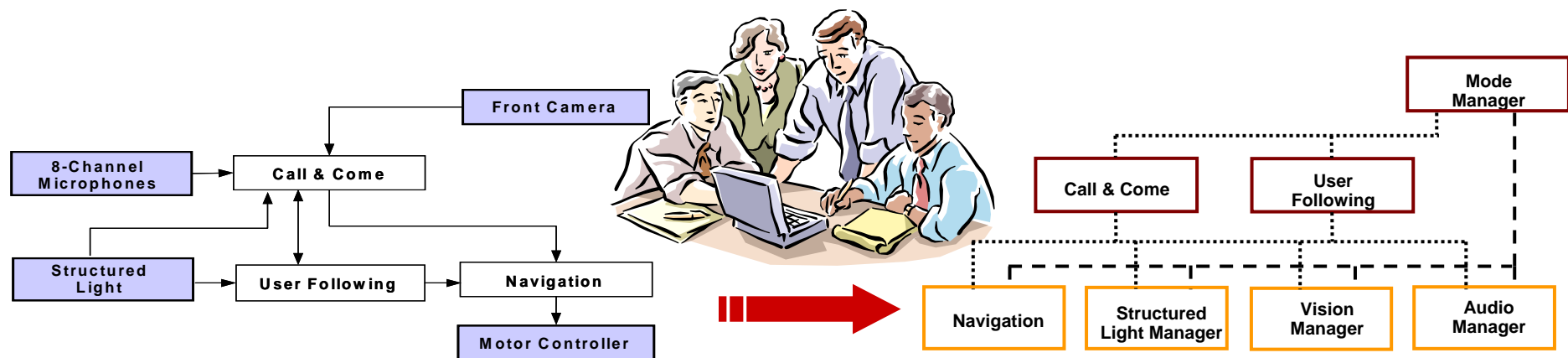
■ To verify P3, we need to build an *observer* to detect violations

```
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
```



Lessons Learned I

- From the experience of re-engineering SHR100, we are convinced that **re-engineering is practically beneficial**
 - Developers tend to concentrate only on **technical components at the early state** without considering how they will be integrated.
 - Once feasibility of the project is confirmed through an early prototype, **re-engineering the product at later stage** should be enforced for increased quality of the product.



Lessons Learned II

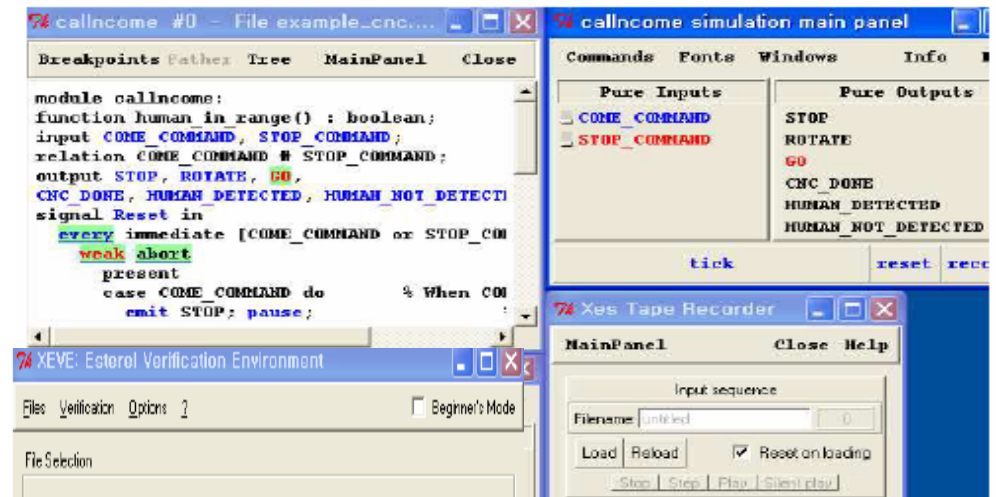
```
m_bHuman = FALSE;
MY_TRACE("[CALL AND COME]#n %d",)
switch (param.order)
{
case 0:
{
if (m_pSLM)
MY_TRACE("CALL PROCE:
m_pSLMain->SetProce:
/**/
m_tracker.cmdTurn(body.d
param.order++;
MY_TRACE("[pInCallCome:Tt
}
break;
case 1:
{
static CPose befPose;
static int mCnt = 0;
CPose curPose = m_tracker
```

■ We uncovered subtle bugs which decrease the accuracy of detecting a user

■ Implementing reactive SW in C is error-prone

■ Esterel allows clean and compact design and implementation

■ Esterel's formal semantics allows rigorous analysis such as model checking



- After all, SAIT decided **not** to adopt re-engineered robot sw in their robot prototype ☹️
- Excuses are
 - ✚ Overhead of using a new language
 - Most robot developers are not from CS field
 - ✚ Inability to optimize final code manually
 - For consumer products, resource constraints are still major issues
 - ✚ Version discrepancy
 - While re-engineering was going on at POSTECH, SAIT constantly add/updated features, which our re-engineered code did not cover



Question ??

Comments ??

Suggestions ??

