Formal Analysis of Pervasive Computing Systems

Yan Liu[1], Xian Zhang[1], Yang Liu[1], Jun Sun[2], Jin Song Dong[1], Jit Biswas[3], Mounir Mokhtari[4]

[1] National University of Singapore [2] Singapore University of Technology and Design
[3] Institute for Infocomm Research, Singapore [4] CNRS-IPAL/Institut TELECOM

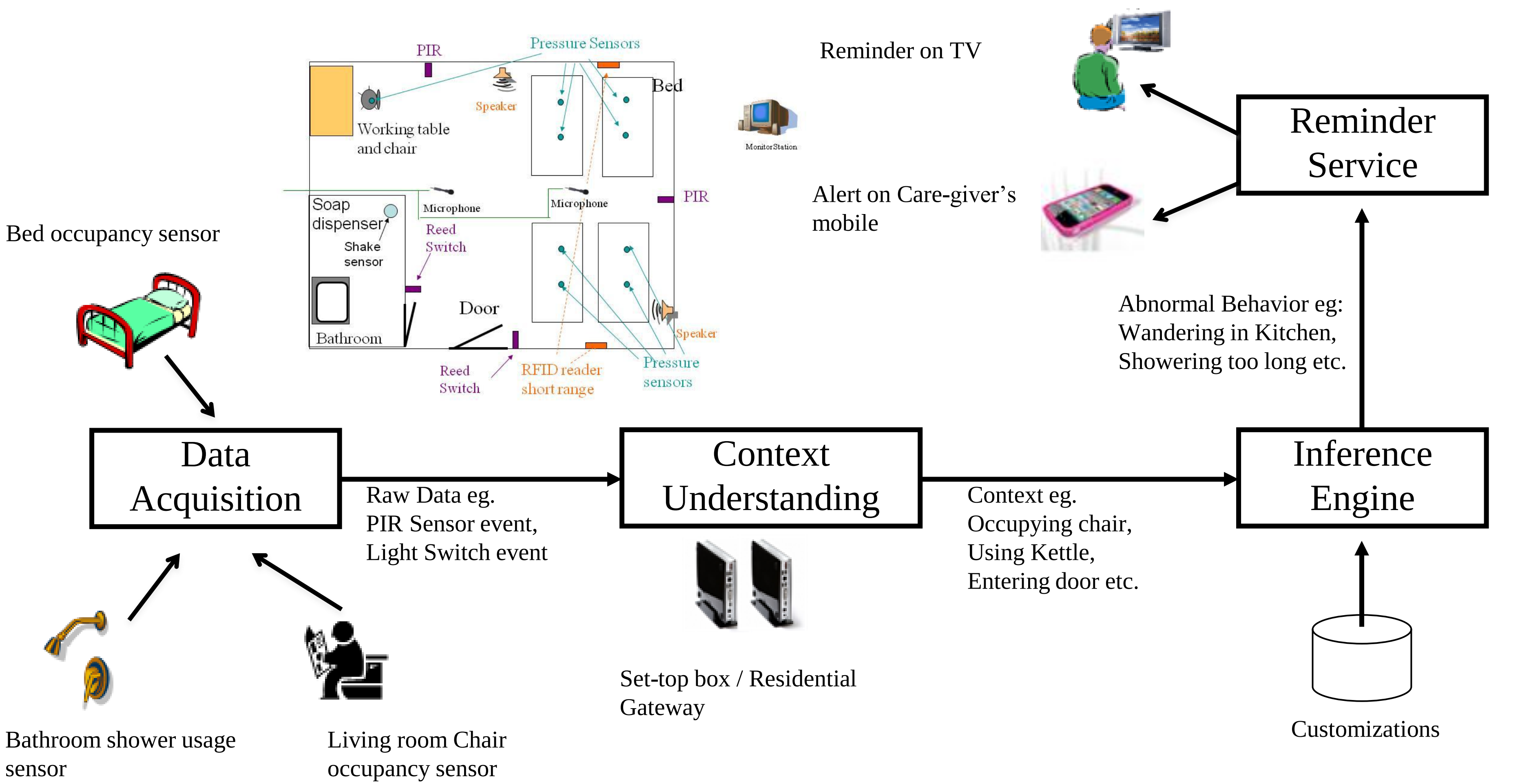
ICECCS 2012

Outline

- Motivation
- Formal Analysis Approach
  - Formal modeling framework
  - Formal specification of critical properties
- Case Study
- Related Work
- Conclusion and Future Work

A Typical Example: AMUPADH

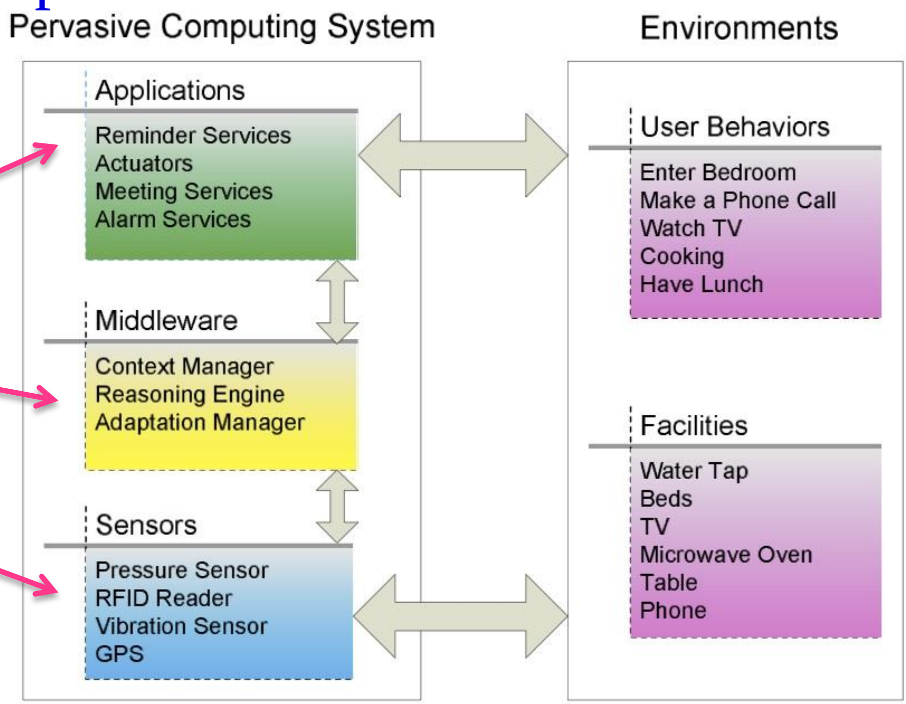
-- Activity Monitoring and UI Plasticity for supporting Ageing with mild Dementia at Home



Motivation

A. PvC systems are safety-critical and their correctness should be verified, but they are complex:
 – Ad hoc interactions among layers
 – Unpredictable environment inputs
 – Faults in multi-layers

 e.g. :
 - Reminder conflicts
 - False reasoning rules
 - Sensor fails



Motivation

A. PvC Systems are safety-critical and highly complex

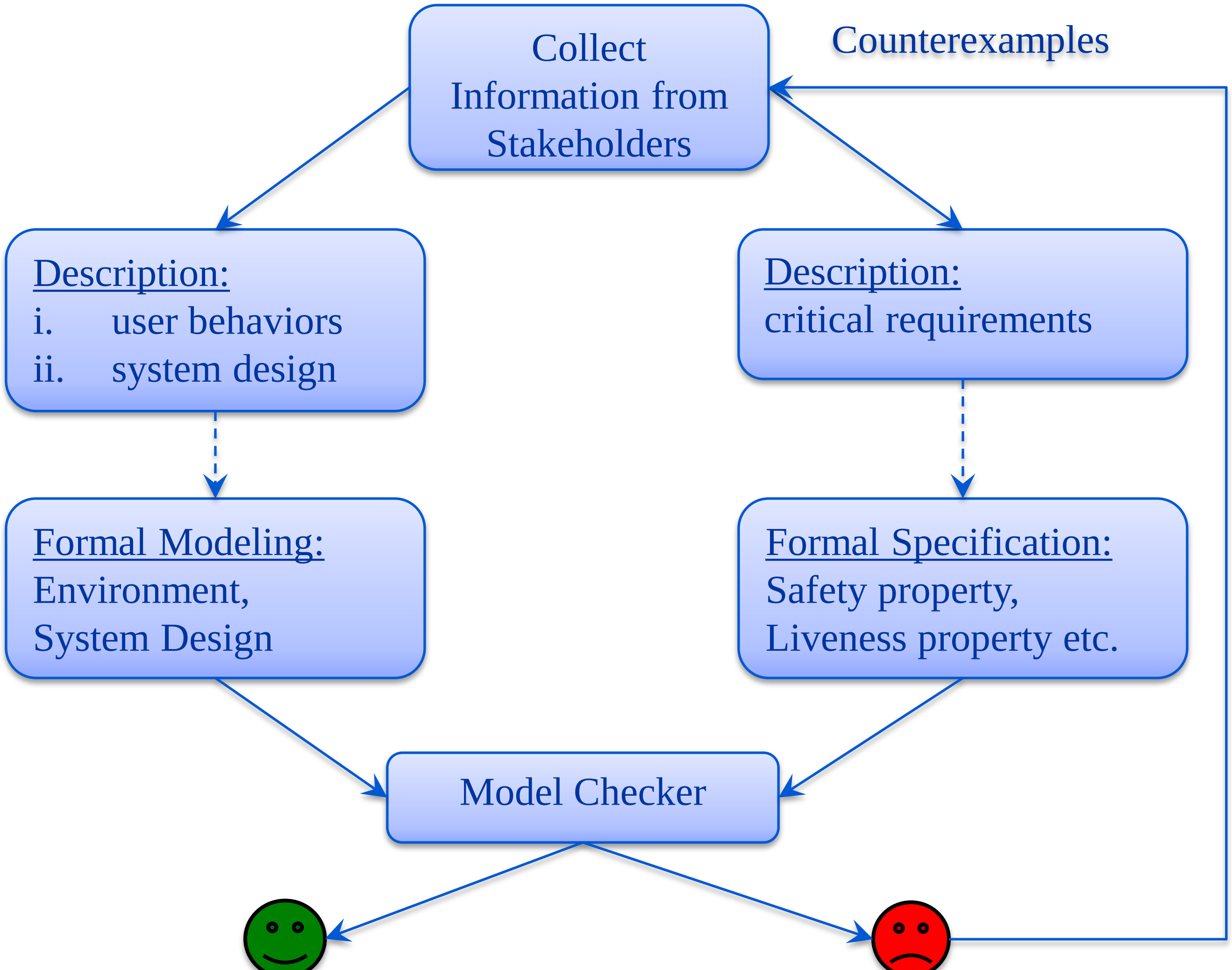
B. Analyze PvC systems via testing is non-trivial

- High cost: deploy the sensors and software system
- Difficult: acting like a real user
- Not complete: explore partial system behavior only
- Hard to debug: no clue for pinpointing source of bugs (manually checking every part of the system)

Our propose: Use formal methods, esp. model checking

- Formalisms for concurrent interaction
- Automatic verification and exhaustive search
- Counterexamples for bug tracking

Formal Analysis Approach: The Process



Formal Analysis Approach: Formal Modeling Framework

- What to model: Critical behaviors & Interactions
- As for a PvC system:
  - It's user centered:
    - Model Environment Inputs: User behaviors & Environment constraints
  - It's a system of systems:
    - Model each sub-system specifically:
      - Sensor layer
      - Middleware layer
      - Application layer
    - Model the compositional structures:
      - Sequential, Interleave and Parallel

Formal Modeling Framework

- Modeling Environment Inputs:
  - User behaviors:
    - *Patient_proc(id) = activity1.id->location_1(id)*
    - *[] activity2.id ->location_2(id);*
    - event prefixing & choice constructs
  - Environment constraints:
    - Synchronized behaviors
    - *Bed1() = activity1.0 -> Bed1_Occupied(0)*
      [] activity1.1 -> Bed1_Occupied(1);
    - event synchronization and choices
  - Multi-user sharing environment:
    - Env() = (*Patient_proc(0) ||| Patient_proc(1)) || Bed1()*
    - parameterized processes, interleaving(|||) and Parallel

Formal Modeling Framework

- Modeling Environment Inputs:
- Modeling System Design:
  - Sensor Layer: sensing and data transmission
    - *Sensor() = activity1.id -> port!sensorId.statusId.id->Sensor();*
    - Concurrent Communications:
      - Multi-Party Event Synchronization for sensor interacts with environment
      - Channels "*port*" for sensor interacts with system
    - Refreshing Rates:
      - *TimelySensing() = Sensor() within[10];*
      - Real time constructs such as "*within[t]*" in Stateful Timed CSP
    - Sensor Failure:
      - *FaultySensor() = pcase{ 9: Sensor()*
        1: fail->Skip}; FaultySensor();
      - Probabilistic language constructs such as "*pcase*" in PCSP or PRTS
  - Middleware Layer:
    - Shared Contexts: global variables
    - Reasoning Process (Rules): guarded processes or conditional statements
      - *rule1() = if(conditions){chan!msg -> Skip};*

Formal Modeling Framework

- Modeling Environment Inputs:
- Modeling System design:
  - Sensor Layer:
  - Middleware Layer:
  - Application Layer: channel communication and events
- Composing A Complete Model:
  - Composition patterns in hierarchical modeling languages such as CSP#
    - Sequential Composition(;): workflows
    - Interleave Composition(|||): processes proceed independently
    - Parallel Composition(||): concurrent behaviors

Formal Analysis Approach: Revisit

- Formal modeling framework
  - Environment inputs:
    user behaviors & environment constraints
  - Sensor behaviors:
    sensing behaviors & data transmission
  - Middleware layer:
    shared contexts & reasoning process
  - Application layer:
    service adaptation & channel communication
  - Composition patterns:
    sequential, interleave & parallel
- Next: Formal specification of critical requirements

Formal Analysis Approach: Formal specification of properties

- Desirable properties:
  - Deadlock freeness (**check for dead state**)
    - In a dead state, the system will stop reacting.
  - Guaranteed services (**Linear Temporal Logic**)
    - The system will deliver the service whenever certain situation happens.
    - Eg. If a patient is wandering in a room, the leave-room-reminder should eventually prompt.
    - *[](PatientWandering-> <>LeaveRoomReminder)*
  - Security Related Properties (**Linear Temporal Logic**)
    - Access control of user's confidential profiles
    - Eg. A food delivery person should not have access to the patient's medical records.
    - *[](FoodDeliveryPerson -> not (<> AccessPatientProfile))*

Formal specification of properties

School of Computing

- Testing Purposes (**Reachability checking**):
  - System Inconsistency
    - System knowledge is not consistent with actual environment.
    - Eg. A PIR sensor detects nobody in the room, but the context variable recording user's location shows one in the room.
    - In CSP#, it is defined as:
    - *#define inconsist (PIR_room == **Silent** && LocationUser == **inRoom**);*
    - *#assert system **reaches** inconsist;*
  - Conflicting/ False Service Adaptation
    - Two services resulting conflict consequences adapt in the same time.
    - In multi-people sharing environment, a service adapts to a wrong person.
    - Eg. In AMUPAD, a sit-bed-too-long-reminder is sent to patient 1 who's not in bedroom at the time.
    - In CSP#, *#define FalseAlarm (SBTL_reminder[1] && LocationP1 != Bedroom);*
    - *#assert system **reaches** FalseAlarm;*
  - Anomalies in reasoning rules: *duplications, conflicts & unreachble rules etc.*

Case Study: Verification Results

- Modeling language: Communicating Sequential Program(CSP#)
- Model checker: PAT

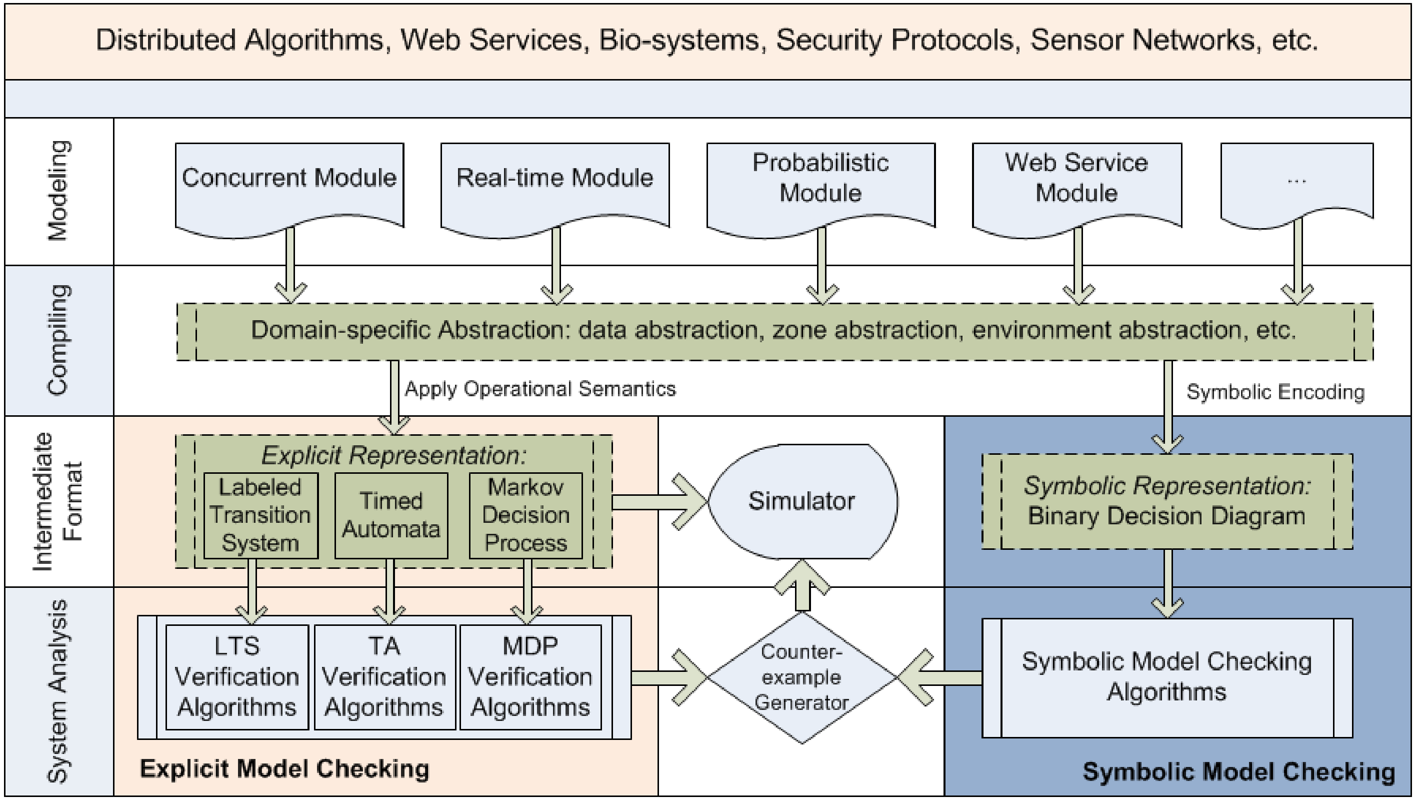
Bugs?	Property	Result	#State	#Transitions	Execution Time(s)
-	P1.1	-	-	-	OOM
☺	P1.2	True	1433654	2038064	815
☺	P1.3	True	10783353	15832370	7045
☺	P2.1	True	1599797	2430351	1945
☹	P2.2	False	68178	130734	39
☹	P2.3	False	2192251	4531005	12414
☹	P2.4	False	832144	1663779	729
☹	P2.5	False	4314	5150	1.6
☺	P2.6	True	1579579	2377381	1913
☹	P3	True	572	745	0.3
☹	P4.1	True	14675	20615	6.1
☹	P4.2	True	2446	3036	1.11
☹	P4.3	True	2332744	3001756	1047

Reminder fails to send

Inconsistency & Reminder Conflicts

Tool Introduction—
Process Analysis Toolkit (PAT)

- PAT is a framework of model checkers:
  - Each module is a model checker:



Tool Introduction—

Process Analysis Toolkit (PAT)

- PAT is available at http://www.patroot.com
- Used as an educational tool in NUS and York University
- PAT has 2000+ registered users from 400+ organizations in 52 countries and regions

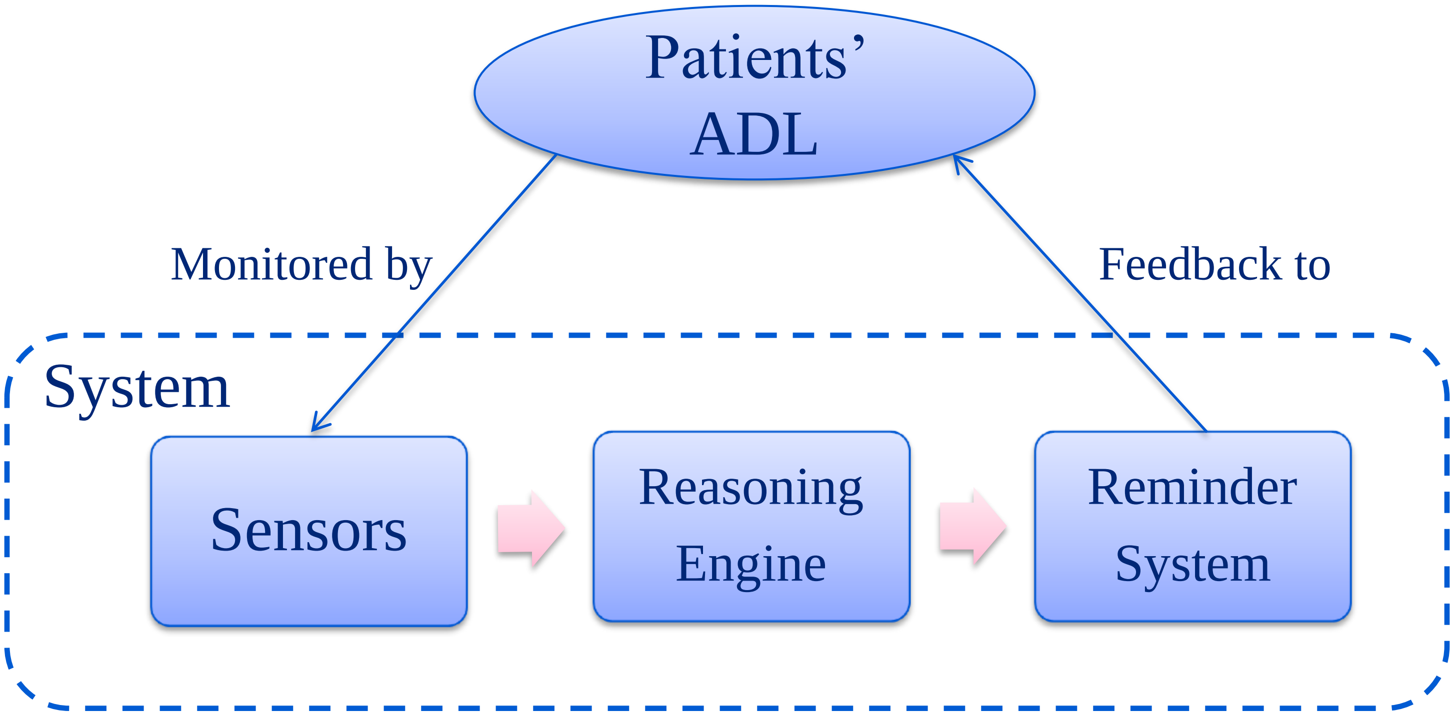
Conclusion & Future Work

- Formal analysis of pervasive computing system:
  - Formal modeling framework
  - Formal specification of critical requirements
  - Case study of a smart healthcare system for elderly dementia people
  - Found bugs!
- In Future: *Handling large state space*
  - BDD encodings of system space: can handle much larger space than explicit state verification
  - Compositional Verification: Verify system property by verified sub-systems

Thank you!

Case Study: AMUPADH modeling

- Modeling language: Communicating Sequential Program(CSP#)
  - Supports modeling of concurrent interactions and hierarchical structures
  - Supports shared variables and programming features
- Model checker: PAT



Case Study: Property Specification

- **P1: Deadlock freeness**
  - P1.1 #**assert** SmartNursingHome() *deadlockfree;*
  - P1.2 #**assert** SmartBedroom() *deadlockfree*;
  - P1.3 #**assert** SmartShowerRoom() *deadlockfree*;
- **P2: Guaranteed reminder**
  - P2.1 – P2.6 6 reminders: 2 in bedroom and 4 in shower room
- **P3: System inconsistency:**
  - PIR sensor in shower room case
- **P4: Conflicting/False Alarm**
  - **P4.1 Conflicting reminders**:
  - *Shower-Using-Soap-Reminder and Leave-Room-Reminder send at the same time resulting patient to be confused.*
  - **P4.2 False reminder:**
  - *Sit-Bed-Too-Long-Reminder is sent to patient 1 who's not in the bedroom.*

Case Study: Bug Report

- System inconsistency
  - The bug: shower room is empty in real environment, however the location of person 1 remains in Shower Room
  - enterShowerRoom.1 -> turnOnTap -> exitShowerRoom.1 -> port.PIRShowerRoom.Silent
- False alarm
  - The bug: person 1 is not in the bedroom, however sit-too-long reminder is sent to him
  - enterBedroom.2 -> sitOnBed.2.1 -> **promptReminder**
- Conflicting reminders
  - Apply soap reminder and wandering in the shower room reminder both prompted to the same patient
  - enterShowerRoom.1 -> res.Error.WanderInShowerRoom.1 -> promptReminder.Wander -> turnOnTap -> res.Error.ShowerNoSoap.1 -> promptReminder.Soap

Related Works

- Papers:
  - TCOZ model of a smart meeting room
    - *ISoLA'06, Jin Song Dong et al.* *[DFSS06]*
  - Ambient Calculus model for location sensitive smart hospital
    - *TECS 2010, Antonio Coronato et al.* *[CP10]*
  - A-FSM and fault patterns for Context-Aware Adaptive Applications
    - *TSE 2010, Michele Sama et al.* *[SER+10]*
  - Towards Verification of Pervasive Computing Systems
    - *FMIS'09, Myrto Arapinis et al.* *[ACD+09]*
- The modeling languages are not hierarchical
  - *no support for compositional structures/layered system architectures*
- There is no automatic tool support
  - *limited applicability to large PvC systems*

References

- [EG01]W. Keith Edwards and Rebecca E. Grinter. At home with ubiquitous computing: Seven challenges. In *Proceedings of the 3rd international conference on Ubiquitous Computing,* UbiComp '01, pages 256-272, London, UK, 2001. Springer-Verlag.
- [Sat01] M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, 8:10-17, 2001.
- [DFSS06] Jin Song Dong, Yuzhang Feng, Jing Sun, and Jun Sun. Context awareness systems design and reasoning. In *Proceedings of the Second International Symposium on Leveraging Applications of Formal Methods, Verication and Validation*, pages 335-340, Washington, DC, USA, 2006. IEEE Computer Society.
- [CP10] Antonio Coronato and Giuseppe DE Pietro. Formal specication of wireless and pervasive healthcare applications. ACM Trans. Embed. Comput. Syst., 10:12:1-12:18, August 2010.
- [SER+10] Michele Sama, Sebastian Elbaum, Franco Raimondi, David S. Rosenblum, and Zhimin Wang. Context-aware adaptive applications: Fault patterns and their automated identication. IEEE Trans. Softw. Eng., 36:644-661, September 2010.
- [ACD+09] Myrto Arapinis, Muy Calder, Louise Denis, Michael Fisher, Philip D. Gray, Savas Konur, Alice Miller, Eike Ritter, Mark Ryan, Sven Schewe, Chris Unsworth, and Rehana Yasmin. Towards the verication of pervasive systems. ECEASST, 22, 2009.