Formal Analysis of Pervasive Computing Systems

Yan Liu¹, Xian Zhang¹, Yang Liu¹, Jun Sun², Jin Song Dong¹, Jit Biswas³, Mounir Mokhtari⁴

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

Data Acquisition
- Raw Data eg. PIR Sensor event, Light Switch event

Context Understanding
- Context eg. Occupying chair, Using Kettle, Entering door etc.

Inference Engine
- Set-top box / Residential Gateway

Reminder Service
- Reminder on TV
- Alert on Care-giver’s mobile

Abnormal Behavior eg: Wandering in Kitchen, Showering too long etc.

Customizations
- Bathroom shower usage sensor
- Living room Chair occupancy sensor

Bed occupancy sensor

A Typical Example:
--- AMUPADH ---
Activity Monitoring and UI Plasticity for supporting Ageing with mild Dementia at Home

Formal Analysis of Pervasive Computing Systems

7/19/2012
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

Collect Information from Stakeholders

Description:
- user behaviors
- system design

Formal Modeling:
- Environment
- System Design

Model Checker

Counterexamples

Description:
critical requirements

Formal Specification:
- Safety property
- Liveness property etc.
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:
    • $\text{Patient\_proc}(id) = \text{activity1.id}\rightarrow\text{location\_1}(id)$
    • $[] \text{activity2.id} \rightarrow \text{location\_2}(id)$;
    • event prefixing & choice constructs
  – Environment constraints:
    • Synchronized behaviors
      • $\text{Bed1()} = \text{activity1.0} \rightarrow \text{Bed1\_Occupied}(0)$
        $[] \text{activity1.1} \rightarrow \text{Bed1\_Occupied}(1)$;
    • event synchronization and choices
  – Multi-user sharing environment:
    • $\text{Env}() = (\text{Patient\_proc}(0) ||| \text{Patient\_proc}(1)) || \text{Bed1()}$
    • parameterized processes, interleaving($|||)$ and Parallel
Formal Modeling Framework

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

- 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.
  - \([\mathbb{L}(\text{FoodDeliveryPerson}) \rightarrow \neg (\text{AccessPatientProfile})]\)
Formal specification of properties

- **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 & unreachable rules etc.
## Case Study: Verification Results

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

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

**Reminder fails to send**

**Inconsistency & Reminder Conflicts**
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*
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
References

Tool Introduction—
Process Analysis Toolkit (PAT)

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

<table>
<thead>
<tr>
<th>Distributed Algorithms, Web Services, Bio-systems, Security Protocols, Sensor Networks, etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Modeling</strong></td>
</tr>
<tr>
<td><strong>Compiling</strong></td>
</tr>
<tr>
<td><strong>Intermediate Format</strong></td>
</tr>
<tr>
<td><strong>System Analysis</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

7/19/2012
Formal Analysis of Pervasive Computing Systems
Tool Introduction—

**Process Analysis Toolkit (PAT)**

- PAT is available at [http://www.patroot.com](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