Formal Analysis of Pervasive Computing Systems

Yan LIU
yanliu@comp.nus.edu.sg
Supervisor: A/P Jin Song DONG
Introduction

• Pervasive Computing System
  — Smart Systems (Aware of context and adaptive to user needs)
  — Machines fit the human environment instead of forcing humans to enter theirs

“Computers disappear from the environment and weave themselves into the fabric of everyday life until they are indistinguishable from it”
— Mark Weiser
Outline

• Introduction
  – Pervasive Computing Systems

• Motivation

• Formal Analysis Approach
  – Formal modeling framework
  – Critical properties
  – Case study

• Conclusion and Future Work
Motivation

A. Pervasive computing systems are highly complex
[EG01, Sat01]

– Heterogeneity and ad hoc interactions
– Various environment inputs
– Context-Awareness
– Unpredictable user behaviors
  • multi-user environment
Motivation

A. Pervasive computing systems are highly complex

B. Traditional validation methods such as simulation and testing only cover partial system behaviors

C. Model Checking is a good candidate
   ✓ Modeling formalisms for concurrent communication
   ✓ Automatic Verification
   ✓ Exhaustive search of system state spaces
Formal Analysis of Pervasive Computing Systems

• A motivating example: AMUPADH system
• A formal modeling framework
• Critical safety and liveness requirements
• Case study of Smart Nursing Home
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
- Set-top box / Residential Gateway
- Context eg. Occupying chair, Using Kettle, Entering door etc.

Inference Engine
- Abnormal Behavior eg: Wandering in Kitchen, Showering too long etc.
- Reminder Service
  - Reminder on TV
  - Alert on Care-giver’s mobile

Customizations
- Bathroom shower usage sensor
- Living room Chair occupancy sensor
- Smart Nursing Home
- Bed occupancy sensor
Formal Modeling Framework

• Modeling Environment Inputs
  – User behaviors (Given by system designer)
  – Environment Objects (Environment constraints)

• Modeling System Design
  – Sensor layer
  – Middleware layer
  – Application layer

• Composing A Complete Model
  – Compositional patterns
Formal Modeling Framework

• Modeling Environment Inputs:
  – User behaviors:
    • $Patient_{proc}(id) = activity1.id -> location_1(id)$
    • $[] activity2.id -> location_2(id);$  
    • event prefixing & choice semantics
  – Interactions between user and environment:
    • $Bed1() = activity1.id -> Bed1_{Occupied}(id);$ 
    • event synchronization semantics
  – Multiple users:
    • parameterized processes

• Modeling System Design
• Composing A Complete Model
Formal Modeling Framework

- Modeling Environment Inputs:
- Modeling System design:
  - Sensor Layer:
    - \texttt{Sensor()} = activity1.id \rightarrow \text{port!sensorId.statusId.id->Sensor();}
    - Concurrent Communications:
      - \textbf{Multi-Party Event Synchronization} for sensor interacts with environment
      - \textbf{Channel Communication} for sensor Interacts with system
    - Refreshing Rates:
      - \textbf{Real time constructs} such as \textit{within}[t] in Stateful Timed CSP
    - Sensor Failure:
      - \textbf{Probabilistic} language constructs such as \texttt{pcase} in PCSP or PRTS
  - Middleware Layer:
    - Shared Contexts: \texttt{global variables}
    - Reasoning Rules: \texttt{guarded processes} or \texttt{conditional statements}
      - \texttt{rule1()} = \texttt{if(conditions){chan!msg -> rule2()}}\texttt{else(rule2())}
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 communications
Revisit

- **System features:**
  - Layered architecture
  - Heterogeneity
  - Interaction patterns
  - Context-awareness and adaptation

- **Critical properties:**
  - Safety properties
  - Liveness properties
Critical Properties: Safety & Liveness

• Safety properties: nothing bad happens
  – Deadlock freeness (**check for dead state**)
  – Error states not reachable (**reachability checking**)
    • Inconsistency between system knowledge and actual environment
    • False adaptation
    • Conflicting services

• Liveness properties: something good eventually happens
  – Guaranteed services (**Linear Temporal Logic**)
    • The system will deliver the service whenever certain situation happens
    • Eg. If patient needs help, the reminder eventually prompts
    • $[(\text{patientInDanger} \rightarrow <>\text{AlertToNurse})$
Case Study: AMUPADH

- Platform
- AMUPADH System Modeling
- Critical Properties
- Verification Results
Case Study: Platform

• Modeling Language: CSP#
  – Short for Communicating Sequential Program
  – Extension of classical CSP [Tony Hoare]
    • Shared variables
    • Low level programming constructs
  – Supports most of the modeling patterns
  – External libraries

• Model Checker: PAT
  – Supports compilation of CSP#
  – Verification algorithms for rich properties
Case Study: AMUPADH

• Architecture:

Patients’ ADL

Monitored by Sensors

Reasoning Engine

Reminder System

Feedback to System
Case Study: System Modeling

• Patient Behaviors
  – Labeled Transition System
Case Study: System Modeling

• **Sensor Behaviors**
  – Bed Pressure Sensor

![Bed Pressure Sensor Behaviors Diagram]

- Event Synchronization
- Synchronized Channel
Case Study: System Modeling

- Rule Based Reasoning Engine
  - Rules Modeling using If-Else Statements

Rule 7: Sit On Bed for Too Long

\[
sensors[\text{Pressure}_1] == \text{SITTING} \quad \text{Duration}[\text{Pressure}_1] > 20
\]

res!\text{ERROR.SitTooLong.1}

Reminder System
Case Study: System Modeling

• Reminder System

res?status.rid.pid

Status == Normal

Deactivate Reminder[rid][pid]

Status == Error

Activate Reminder[rid][pid]
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:** reminders are sent out eventually when they are required to.
  – **Bedroom**
    P2.1 `#assert` SmartBedroom() `|= [](UsingWrongBed -> <>RemindedWrongBed);
P2.6 `#assert` SmartBedroom() `|= [](TroubleSleep -> <>RemindedSleep);
  – **Shower Room**
    P2.2 `#assert` SmartShowerRoom() `|= [](TapNotOff -> <>OffTapReminded);
P2.3 `#assert` SmartShowerRoom() `|= [](WanderingInShowerRoom -> <>WanderReminded);
P2.4 `#assert` SmartShowerRoom() `|= [](ShowerWithoutSoap -> <>NoSoapReminder);
P2.5 `#assert` SmartShowerRoom() `|= [](ShowerTooLong -> <>EndShowerReminded);

• **P3: System inconsistency:** system is wrong about the patient’s location
  – `#define` InConsistency (Pos_Person[1] == SHOWERROOM && ShowerFlag && sensors[PIR] == SILENT);
  – `#assert` SmartShowerRoom reaches InConsistency;
Case Study: Property Specification

• **P4: Conflicting/False Alarm**
  
  – **P4.1 Conflicting reminders**: reminders request different actions
    
    • Apply soap reminder && wandering in the shower room reminder prompt simultaneously.
      
      ```
      # define ConflictReminder (ReminderStage[ShowerNoSoap * 2] != 0 
      && ReminderStage[WanderingInSR * 2] != 0);
      # assert SmartShowerRoom reaches ConflictReminder;
      ```

  – **P4.2 False reminder**: reminders shouldn’t be prompted.
    
    • E.g., reminder in bedroom scenario sends to a person who is not in the bedroom!
      
      ```
      #define FalseReminder (Pos_Person[1] != BEDROOM 
      &&( ReminderStage[LyingWrongbed] != 0 
      || ReminderStage[SitBedLong] != 0 ));
      #assert SmartBedRoom reaches FalseReminder;
      ```
# Case Study: Verification Results

**Processor:** Intel(R) Xeon(R) CPU E5506 @2.13GHz  
**RAM:** 32.0GB  
12414 s ≈ 3.5 hours

<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>
Case Study: Bug Report

- System does not fulfill basic level of confidence

- Bugs found:
  
  - 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

Demo
Conclusion & Future Work

• Formal analysis of pervasive computing system:
  – Formal modeling framework
  – Critical safety and liveness properties
  – Case study on A Typical Smart System
  – Found bugs!

• In Future:
  – Handling large scale state space
    • BDD encodings can handle much larger space than explicit state verification
    • Component Based System Verification Techniques- Verify system property by verify components
References

Thank You!