Mapping Context-Dependent Requirements to Event-Based Context-Oriented Programs for Modularity

Tetsuo Kamina (UTokyo)
Tomoyuki Aotani (Tokyo Tech)
Hidehiko Masuhara (Tokyo Tech)
Purpose

- Methodology for context-aware systems from requirements to implementation
- Context-dependent behavior
  - well-studied in implementation
  - identification of contexts and behavioral variations is not trivial

Requirements model and systematic implementation using event-based COP language EventCJ
Context-awareness

- Capability of a system to behave w.r.t. surrounding contexts (outdoors, indoors)

Map: City map, Floor plan
Positioning: GPS, RFID

- Multiple parts of behavior simultaneously change at runtime
Context-awareness

* Capability of a system to behave w.r.t. surrounding contexts (outdoors, indoors)

** Outdoors**

Map: City map, Floor plan
Positioning: GPS, RFID

* Multiple parts of behavior simultaneously change at runtime
Context-awareness

- Capability of a system to behave w.r.t. surrounding contexts (outdoors, indoors)

Indoors

Map: City map, Floor plan
Positioning: GPS, RFID

- Multiple parts of behavior simultaneously change at runtime
Context-Oriented Programming (COP) [Hirschfeld08]

- modularization of context dep. behavior: layer
- disciplined activation of layers

<table>
<thead>
<tr>
<th>Display</th>
<th>Positioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>display()</td>
<td>getPos()</td>
</tr>
</tbody>
</table>

Outdoors

<table>
<thead>
<tr>
<th>Display</th>
<th>Positioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>display()</td>
<td>getPos()</td>
</tr>
</tbody>
</table>

Indoors

<table>
<thead>
<tr>
<th>Display</th>
<th>Positioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>display()</td>
<td>getPos()</td>
</tr>
</tbody>
</table>

call display()
call getPos()
Context-Oriented Programming (COP) [Hirschfeld08]

- modularization of context dep. behavior: **layer**
- disciplined activation of layers

```javascript
with(Outdoors) {
  ...
}
```
Context-Oriented Programming (COP) [Hirschfeld08]

- modularization of context dep. behavior: **layer**
- disciplined activation of layers

```java
with(Indoors) {
    ...
}
```

```
call display()
call getPos()
```
We need to identify:

- Variations of behavior that should be implemented using a layer
- Context that changes behavior
- A layer assumes a context
  - **Outdoors** is active when **the situation is outdoors**
  - Layer
  - Context
- Timing when contexts/layers change
We need to identify:

- Variations of behavior that should be implemented using a layer
- Context that changes behavior
- A layer assumes a context
  - **Outdoors** is active when **the situation is outdoors**
  - Layer
  - Context
- Timing when contexts/layers change

*Do we really know them?*
Questions

- When to use layers?
  - the ways (layers, design patterns, if) affect modularity

- What are contexts?
  - Can a layer always assume only one single context?
  - How relations b/w contexts and layers are defined?

- How can precisely specify when context changes?
Questions

- When to use layers?
  - the ways (layers, design patterns, if) affect modularity

- What are contexts?
  - Can a layer always assume only one single context?
  - How relations b/w contexts and layers are defined?

- How can precisely specify when context changes?

*Methodology is required*
Overview of methodology

- Identifying **contexts** and context-dependent use cases

- Identifying **layers** by grouping use cases

- Identifying **events** that trigger context changes

**Diagram:**
- **Contexts:** outdoors, indoors
- **Layers:**
  - outdoors
  - outdoors
- **Description of behavior:**
  - outdoors
  - outdoors
- **City map:**
  - outdoors
  - outdoors
- **Events:** when GPS value becomes ...
Example use cases

Pedestrian Navigation System:

- If the user is outdoors, it displays a city map using GPS based positioning.
- If the user is indoors, it displays a floor plan using Wi-Fi based positioning.
- If the floor plan is not available, it displays a city map.
- If no positioning is available, it displays a static map and showing an alert message.
Identifying contexts

We identify contexts from behavior
- Documents describing system-to-be (e.g. use cases)
- Prototypes

Conditions are candidates for contexts
- If the use is outdoors, the system displays a city map
- If the use is indoors, the system displays a floor plan
- If the floor plan is not available, the system displays a city map
- If no positioning is available, the system displays a static map

※conditions affecting a number of parts (e.g., external environmental conditions)
Identifying contexts

- We identify contexts from behavior
  - Documents describing system-to-be (e.g. use cases)
  - Prototypes

- Conditions are candidates for contexts
  - If the use is outdoors, the system displays a city map
  - If the use is indoors, the system displays a floor plan
  - If the floor plan is not available, the system displays a city map
  - If no positioning is available, the system displays a static map

※ conditions affecting a number of parts (e.g., external environmental conditions)
Defining contexts

- We define a context in terms of variables
- outdoors/indoors situations are merged

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>situation</td>
<td>outdoors, indoors</td>
</tr>
<tr>
<td>floorPlan</td>
<td>available, unavailable</td>
</tr>
<tr>
<td>positioning</td>
<td>available, unavailable</td>
</tr>
</tbody>
</table>

- A context is a specific setting of value to a variable (a Boolean term)

  e.g. situation=outdoors
Context-dependent use cases

- Defining context-dependent use cases
  - a specialization of use case applicable in specific contexts
- Annotated with proposition of contexts

Diagram:

- **using a map**
  - **using a city map**
    - situation=outdoors \( \lor \) floorPlan=unavailable
  - **using a floor plan**
    - situation=indoors \( \lor \) floorPlan=available
- **showing alert**
  - positioning=unavailable
- **using a static map**
  - positioning=unavailable
Identifying layers

* Layer: a set of use cases with the same proposition

- using a map
  - using a city map
    - situation=outdoors v floorPlan=unavailable
  - using a floor plan
    - situation=indoors v floorPlan=available

- showing alert
  - positioning=unavailable

* a use case scattering over multiple objects may also be identified as a layer (cf. Jacobson, 2005)
Identifying layers

* Layer: a set of use cases with the same proposition

Using a map → showing alert

using a city map

using a floor plan

situation=outdoors v floorPlan=unavailable
situation=indoors v floorPlan=available

positioning=unavailable

* a use case scattering over multiple objects may also be identified as a layer (cf. Jacobson, 2005)
To identify events...

* Contexts are abstract in use cases

* We need to decompose context into more specific states of the machine (sensors)
* State changes are identified as events
Decomposing contexts

- Detailed specification consists of sensors (GPS, Wi-Fi) and external entities (floor plan)
- Some contexts depend on multiple sensors

<table>
<thead>
<tr>
<th>context</th>
<th>detailed context specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>situation=outdoors</td>
<td>GPS=over the criterion value</td>
</tr>
<tr>
<td>situation=indoors</td>
<td>GPS=under the criterion value</td>
</tr>
<tr>
<td>floorPlan=available</td>
<td>The floor plan service exists</td>
</tr>
<tr>
<td>floorPlan=unavailable</td>
<td>The floor plan service does not exist</td>
</tr>
<tr>
<td>positioning=available</td>
<td>GPS=on or Wi-Fi=connected</td>
</tr>
<tr>
<td>positioning=unavailable</td>
<td>GPS=off and Wi-Fi=disconnected</td>
</tr>
</tbody>
</table>
Identifying events

* Specifying how/when the status of detailed context specification changes

<table>
<thead>
<tr>
<th>event</th>
<th>how</th>
<th>when</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>StrongGPS</strong></td>
<td>GPS=under the criterion, GPS=over the criterion</td>
<td>the GPS signal value becomes over XXX</td>
</tr>
<tr>
<td><strong>GPSEvent</strong></td>
<td>GPS=off $\rightarrow$ GPS=on</td>
<td>the GPS device is becoming on</td>
</tr>
<tr>
<td><strong>WifiEvent</strong></td>
<td>Wi-Fi=disconnected, Wi-Fi=connected</td>
<td>the Wi-Fi device is connected ...</td>
</tr>
</tbody>
</table>
We have obtained so far..
We have obtained so far..

layers/context-dep. use cases representing context-dep. behavior

using a city map
We have obtained so far..

- using a city map
- layers/context-dep. use cases representing context-dep. behavior
- context changing layer activation
- situation=outdoors
- floorPlan=unavailable
We have obtained so far..

- **using a city map**
  - OR
  - **situation=outdoors**
  - **floorPlan=unavailable**
    - **GPS=under criterion**
    - **GPS=over criterion**
      - **event: StrongGPS**
  - layers/context-dep. use cases representing context-dep. behavior
  - context changing layer activation
  - events changing contexts
Translating to implementation

Translating specifications to corresponding constructs in EventCJ [Kamina11]

using a city map

OR

situation=outdoors

floorPlan=unavailable

GPS=under criterion

GPS=over criterion

event: StrongGPS
Translating to implementation

Translating specifications to corresponding constructs in EventCJ [Kamina11]

using a city map

layers are directly mapped

layer CityMap
  when Outdoors || !FPEexists
  { .. }

OR

situation=outdoors

floorPlan=unavailable

GPS=under criterion

GPS=over criterion

contexts are encoded in composite layers

event: StrongGPS
Translating to implementation

Translating specifications to corresponding constructs in EventCJ\cite{Kamina11}

using a city map

layers are directly mapped

layer CityMap
  when Outdoors || !FPExists
  { .. }

contexts are encoded in composite layers

events are encoded in layer transition rules

event GPSEvent ...
transition StrongGPS:
  !Outdoors ? -> Outdoors;

situation=outdoors

floorPlan=unavailable

GPS=under criterion

GPS=over criterion

event: StrongGPS
EventCJ: event-based layer transition

Layer switching is triggered by events

```
event GPSEvent(Navigation n)
  :after call(void *.onStatusChanged())
  && target(n) && if(GPS.isAvailable())
  :sendTo(n);
```

Specify when to generate events using AspectJ-like pointcut language

Layer switching is specified by rules

```
transition GPSEvent: !GPSon ? -> GPSon
```
EventCJ: composite layers

Composite layers are implicitly activated when the condition on other layers holds

```java
layer StaticMap when !GPSon && !WifiConnected {
    /* static map functions */
}
```

StaticMap is inactive
EventCJ: composite layers

[Kamina13]

* Composite layers are implicitly activated when the condition on other layers holds

```plaintext
layer StaticMap when !GPSon && !WifiConnected {
  /* static map functions */
}
```

StaticMap is active
EventCJ: composite layers

[Kamina13]

- Composite layers are implicitly activated when the condition on other layers holds

```c
layer StaticMap when !GPSon && !WifiConnected {
    /* static map functions */
}
```

StaticMap is inactive
Discussion

- Systematic identification of context-related requirements
- Use cases: useful tool to find contexts
- Identification of layers
- Stepwise elicitation of events
- Translation preserves separation of concerns
- More sophisticated case studies are in paper
  - Conference guide system
  - Program editor
Related work

- Jacobson's AOSD (2005)
  - Use case driven methodology
  - A use case scattering multiple classes is implemented by an aspect
  - Mapping "extension points" in use cases to pointcuts in AspectJ
  - Dynamic deployment of behavior is not discussed

- Requirements engineering [Salifu07, Sutcliffe06, Lapouchnian09]
  - Focusing only on requirements variability
  - Lacks viewpoint of detailed context specification
  - Lacks viewpoint of modular implementation
Conclusions

- Use case driven methodology for developing context-aware systems
- Organizing requirements specifications
- Identifying contexts from behavior
- Classifying variations of behavior
  - Identification of layers in use cases
- Stepwise elicitation of details of contexts
- Systematic implementation preserving SoC