UML4COP: UML-based DSML for Context-Aware Systems

Naoyasu Ubayashi (Kyushu University, Japan)
Yasutaka Kamei (Kyushu University, Japan)

October 22, 2012
Development of context-aware systems is not easy!

**UML4COP**: UML-Based DSML for designing context-aware systems

**COP** (Context-Oriented Programming)
Context can be treated as a module!
Outline

- Motivation
- UML4COP
- Program Implementation Based on UML4COP
- Discussion and Future work
Motivation
Motivation

- Context-awareness plays an important role in developing adaptive software.

- However, it is not easy to design and implement such a context-aware system, because its system configuration can be dynamically changed.

- It is hard to check whether a design model is correctly implemented and its behavior is faithful to the design.
COP: New Programming Paradigm

- COP (Context-Oriented Programming) can treat context as a software module.
- Layer-based modularization.
- ContextJ*, ContextJ, Jcop, ContextL.
- We apply the notion of COP to a design method for developing context-aware systems.

public class Employer implements IEmployer {
    layers.define(Layers.Address, new IEmployer() {
        public String toString() {
            return layers.next(this) + "; Address: " + address;
        }
    });

    public void eval() {
        layers.define(Layers.Address, new IEmployer() {
            public String toString() {
                return layers.next(this) + "; Address: " + address;
            }
        });
    }
}

public class Person implements IPerson {
    layers.define(Layers.Address, new IPerson() {
        public String toString() {
            return layers.next(this) + "; Address: " + address;
        }
    });

    layers.define(Layers.Employment, new IPerson() {
        public String toString() {
            return layers.next(this) + "; [Employer] " + employer;
        }
    });

    with(Layers.Address).eval(new Block() {
        public void eval() {
            System.out.println(tanaka);
        }
    });

    with(Layers.Address, Layers.Employment).eval(new Block() {
        public void eval() {
            System.out.println(tanaka);
        }
    });
}

Example: ContextJ*
Our Approach: UML4COP

- DSML (Domain-Specific Modeling Language) for designing context-aware systems.

- Each context is modeled separately from a base design model representing only primary system behavior.

- A system design model at a certain period of time is composed by merging associated contexts.
System Design Model
UML4COP Models

- **View Model**
  - Context representation.
  - Extension of class + sequence diagrams.
  - COP-specific stereotypes.
    - <<layered method>>

- **Context Transition Model**
  - Context transitions.
  - Extension of state machine diagrams.
  - Triggered by COP-specific events.
    - *layer in* (entering a layer)
    - *layer out* (exiting from a layer)
The order of entering a layer can be specified.
We can easily understand system behavior by **composing** views according to context transitions.

Name: Tanaka; Address: Kyoto

Name: Tanaka; Address: Kyoto; [Employer] Name: Suzuki; Address: Tokyo
Program Implementation Based on UML4COP
Translation into COP Languages

- A design model in UML4COP can be easily implemented using COP languages.
- We use ContextJ* whose language features are provided as Java classes.
- Two types of context specification
  - Layer-in-class (ContextJ*)
  - Class-in-layer (similar to AOP)
[List 1]
01: public class Test {
02:     public static void main(String[] args) {
03:         final Employer suzuki =
04:             new Employer("Suzuki", "Tokyo");
05:         final Person tanaka =
06:             new Person("Tanaka", "Kyoto", suzuki);
07:         with(Layers.Address).eval(new Block() {
08:             public void eval() {
09:                 System.out.println(uchio);
10:             }
11:         });
12:     }
13: }

[Layer in]
14: [List 2]
01: public class Layers {
02:     public static final Layer Address =
03:         new Layer("Address");
04:     public static final Layer Employment =
05:         new Layer("Employment");
06: }

[Layered Method]
01: [List 3]
02: public class Person implements IPerson {
03:     private String name;
04:     private String address;
05:     private IPerson employer;
06:     public Person(String newName,
07:                     String newAddress,
08:                     IPerson newEmployer) {
09:         this.name = newName;
10:         this.address = newAddress;
11:         this.employer = newEmployer;
12:     }
13:     public String toString() {
14:         return layers.select().toString();
15:     }
16: }

[Address Layer]
25: [List 4]
01: public class Employer implements IEmployer {
02:     private String name;
03:     private String address;
04:     public Employer(String newName,
05:                     String newAddress) {
06:         this.name = newName;
07:         this.address = newAddress;
08:     }
09:     public String toString() {
10:         return layers.select().toString();
11:     }
12: }

[Layer]
13: [List 5]
26: private LayerDefinitions<IEmployer> layers =
27:     new LayerDefinitions<IEmployer>(new IPerson()) {
28:         public String toString() {
29:             return "Name: " + name;
30:         }
31:     };
32: }

[Layer]
33: }

Discussion and Future work
Everything is OK?

- An essential problem specific to context-awareness still remains.

- Although a UML4COP model is easy to read, it is not necessarily easy to check whether its program execution is faithful to its requirements (e.g., NFR).
Future Work

- We are developing RV4COP, a runtime verification mechanism based on UML4COP.
- Both a system design model and actual execution trace data at a certain period of time are translated into a logical formula.
- We use an SMT (Satisfiability Modulo Theories) solver, a tool for deciding the satisfiability of logical formulas.

RV4COP

UML4COP
NFR specification

Logical Formula
+ Logical Formula

SMT Solver

Execution Trace Data

<table>
<thead>
<tr>
<th>No.</th>
<th>Execution Error (Concept?)</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>[Layer with]</td>
<td>Address</td>
</tr>
<tr>
<td>02</td>
<td>[Method call]</td>
<td>profile</td>
</tr>
<tr>
<td>03</td>
<td>[Method execution]</td>
<td>textStatus (Persona)</td>
</tr>
<tr>
<td>04</td>
<td>[Method call]</td>
<td>textStatus (Persona)</td>
</tr>
<tr>
<td>05</td>
<td>[Method execution]</td>
<td>textStatus (Persona's Address)</td>
</tr>
<tr>
<td>06</td>
<td>[Layered method execution]</td>
<td>textStatus (Persona)</td>
</tr>
<tr>
<td>07</td>
<td>[Layered method execution]</td>
<td>textStatus (Persona's Address)</td>
</tr>
<tr>
<td>08</td>
<td>[Base method call]</td>
<td>textStatus (Persona)</td>
</tr>
<tr>
<td>09</td>
<td>[Base method call]</td>
<td>textStatus (Persona)</td>
</tr>
<tr>
<td>10</td>
<td>[Layer width]</td>
<td>textStatus (Persona)</td>
</tr>
<tr>
<td>11</td>
<td>[Layer with]</td>
<td>Address</td>
</tr>
<tr>
<td>12</td>
<td>[Layer with]</td>
<td>Employment</td>
</tr>
<tr>
<td>13</td>
<td>[Method call]</td>
<td>profile</td>
</tr>
<tr>
<td>14</td>
<td>[Method execution]</td>
<td>textStatus (Persona)</td>
</tr>
<tr>
<td>15</td>
<td>[Method call]</td>
<td>textStatus (Persona)</td>
</tr>
<tr>
<td>16</td>
<td>[Method execution]</td>
<td>textStatus (Persona's Employee)</td>
</tr>
<tr>
<td>17</td>
<td>[Layered method execution]</td>
<td>textStatus (Persona)</td>
</tr>
<tr>
<td>18</td>
<td>[Layered method execution]</td>
<td>textStatus (Persona's Employee)</td>
</tr>
<tr>
<td>19</td>
<td>[Layered method execution]</td>
<td>textStatus (Persona)</td>
</tr>
<tr>
<td>20</td>
<td>[Base method call]</td>
<td>textStatus (Persona)</td>
</tr>
<tr>
<td>21</td>
<td>[Base method call]</td>
<td>textStatus (Persona)</td>
</tr>
<tr>
<td>22</td>
<td>[Layer width]</td>
<td>textStatus (Persona)</td>
</tr>
<tr>
<td>23</td>
<td>[Layer with]</td>
<td>Address</td>
</tr>
<tr>
<td>24</td>
<td>[Method execution]</td>
<td>textStatus (Persona)</td>
</tr>
<tr>
<td>25</td>
<td>[Method call]</td>
<td>textStatus (Persona's Employee)</td>
</tr>
<tr>
<td>26</td>
<td>[Layered method execution]</td>
<td>textStatus (Persona)</td>
</tr>
<tr>
<td>27</td>
<td>[Layered method execution]</td>
<td>textStatus (Persona's Employee)</td>
</tr>
<tr>
<td>28</td>
<td>[Base method call]</td>
<td>textStatus (Persona)</td>
</tr>
<tr>
<td>29</td>
<td>[Base method call]</td>
<td>textStatus (Persona)</td>
</tr>
</tbody>
</table>

[Diagram of rubber ducks]
Summary

- UML4COP, a UML-based design method for COP, is proposed.

- UML4COP and COP improve the expressiveness for designing and implementing context-aware systems.

- As the next step, we plan to develop RV4COP.
Thank you for your attention.