TUCSON, ARIZONA OCTOBER 19-26, 2012

DSM 2012

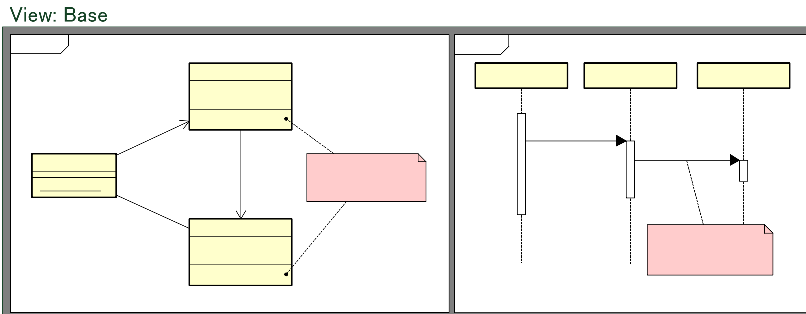
UML4COP: UML-based DSML for Context-Aware Systems

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

October 22, 2012

Overview

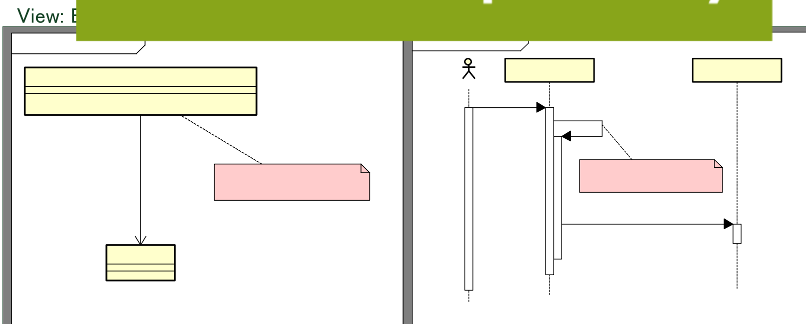
View: Base



View: Address Layer

View Modeling
Each context is modeled separately

View: E



MDSOC (Multi-Dimensional Separation of Concerns)

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, <u>it is not easy to design and implement such a context-aware system</u>, 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.

UML4COP

Hirschfeld, R., Costanza, P., and Nierstrasz, O.:
Context-oriented Programming,
Journal of Object Technology (JOT), vol. 7, no. 3, pp.125-151, 2008.

Example: ContextJ*

Employer

Person

Address Layer

Employment Layer

Name: Tanaka; Address: Kyoto

Name: Tanaka; Address: Kyoto;
[Employer] Name: Suzuki; Address: Tokyo

```
public class Employer imple
layers.define(Layers.Addres
  public String toString() {
    return layers.next(this) + "; Address: " + address;}});
```

```
ess: " + address;}});
```

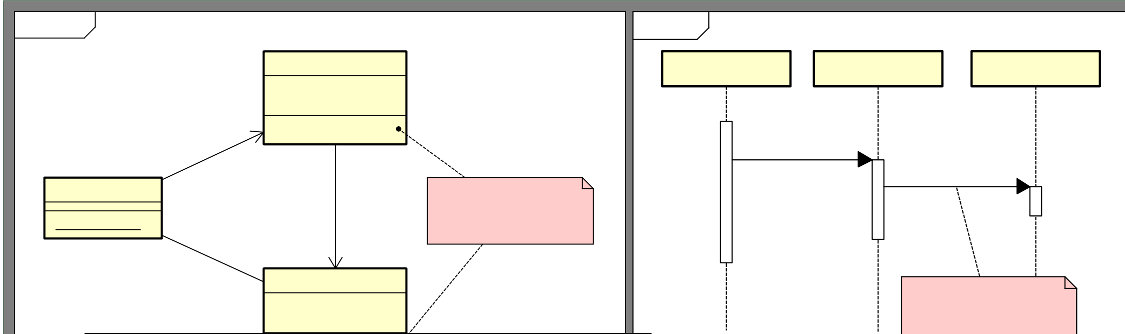
```
  public void eval

with(Layers.Addre
  public void eval(
```

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

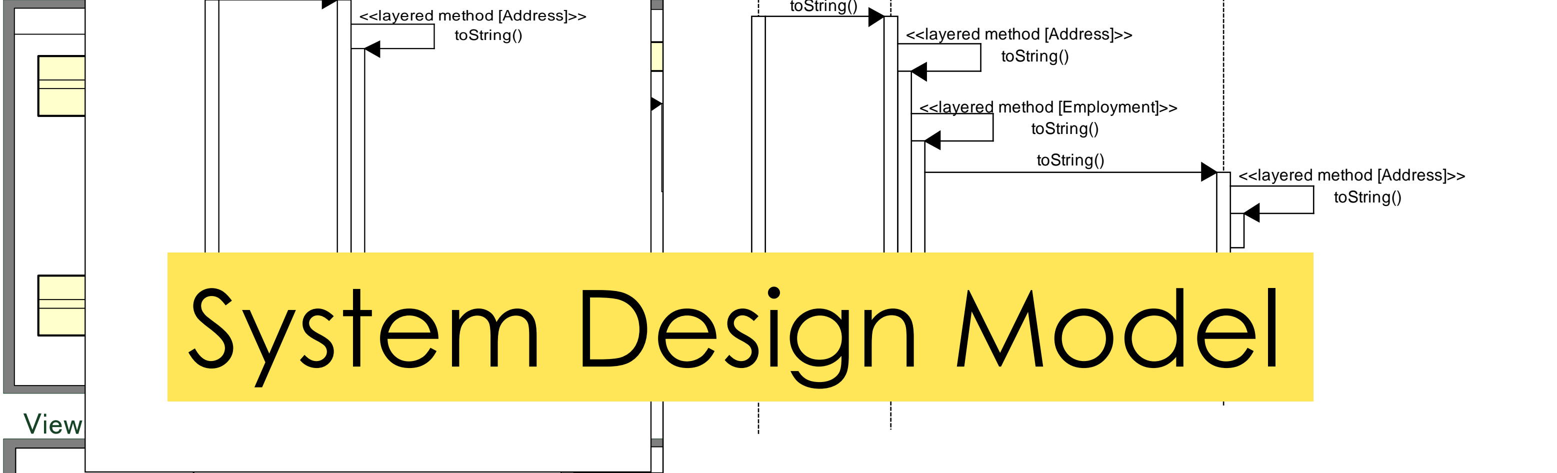
Our Approach: UML4COP

- 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.

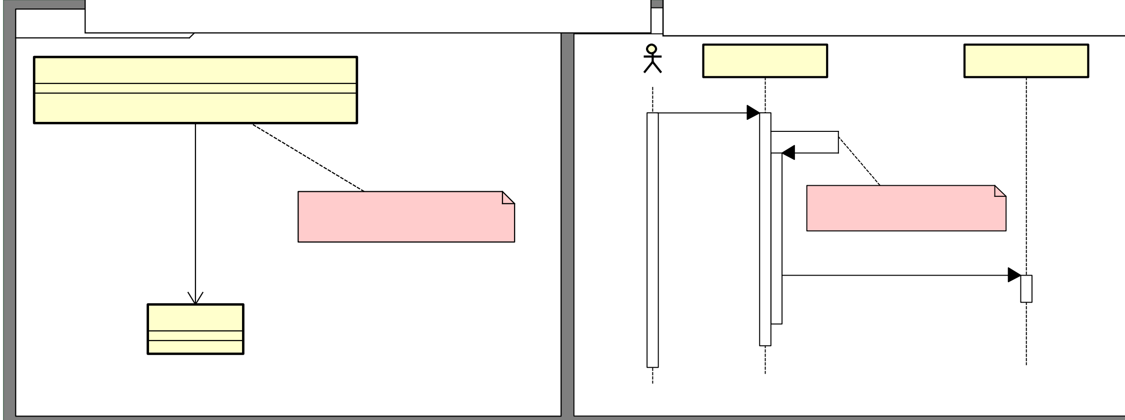
View: Base



Base



ext



Context

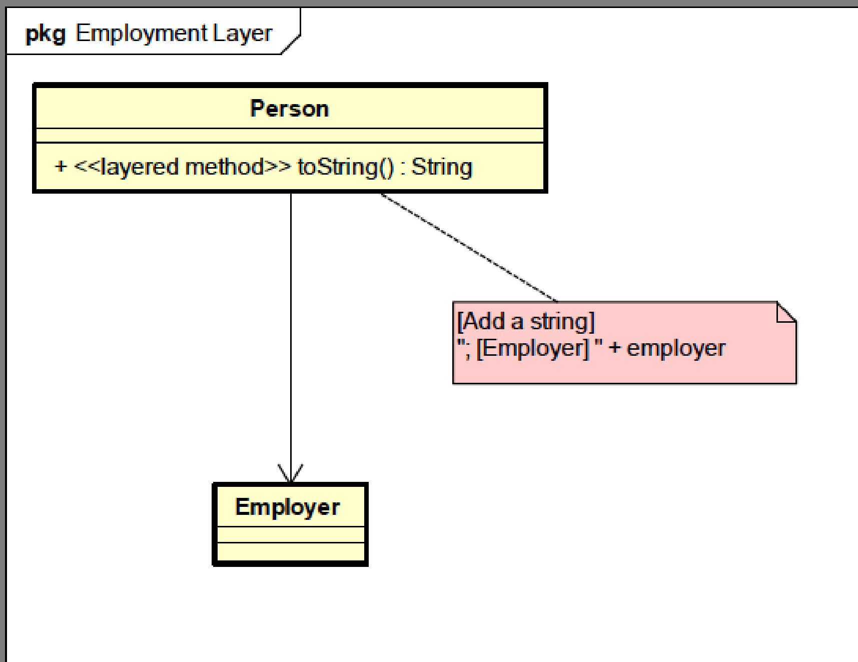
UML4COP

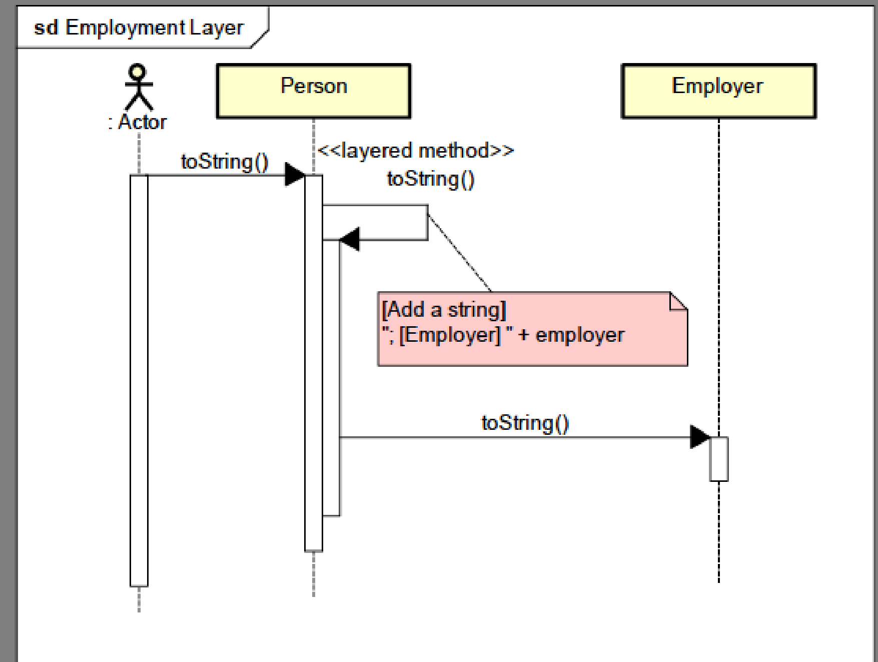
UML4COP Models

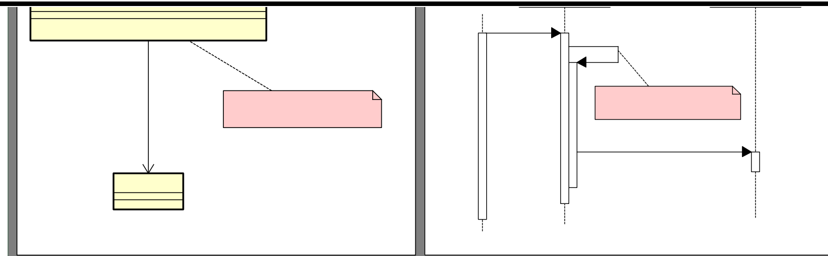
- **View Model**
  - Context representation.
  - Extension of class + sequence diagrams.
  - COP-specific stereotypes.
    - *<<layered method>>*
- **Context Transition Model**
  - Context transitions.
  - Extension of state machine diagrams.
  - Triggered by COP-specific events.
    - *layer in* (entering a layer)
    - *layer out* (exiting from a layer)

View Model

View: Employment Layer

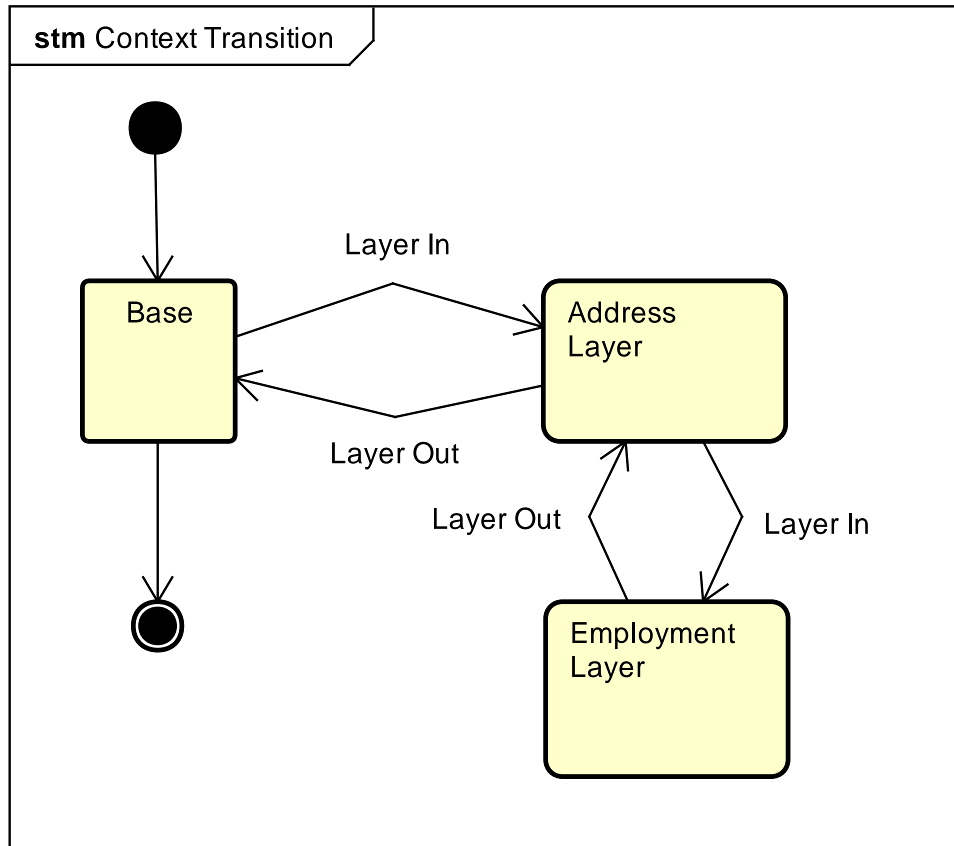






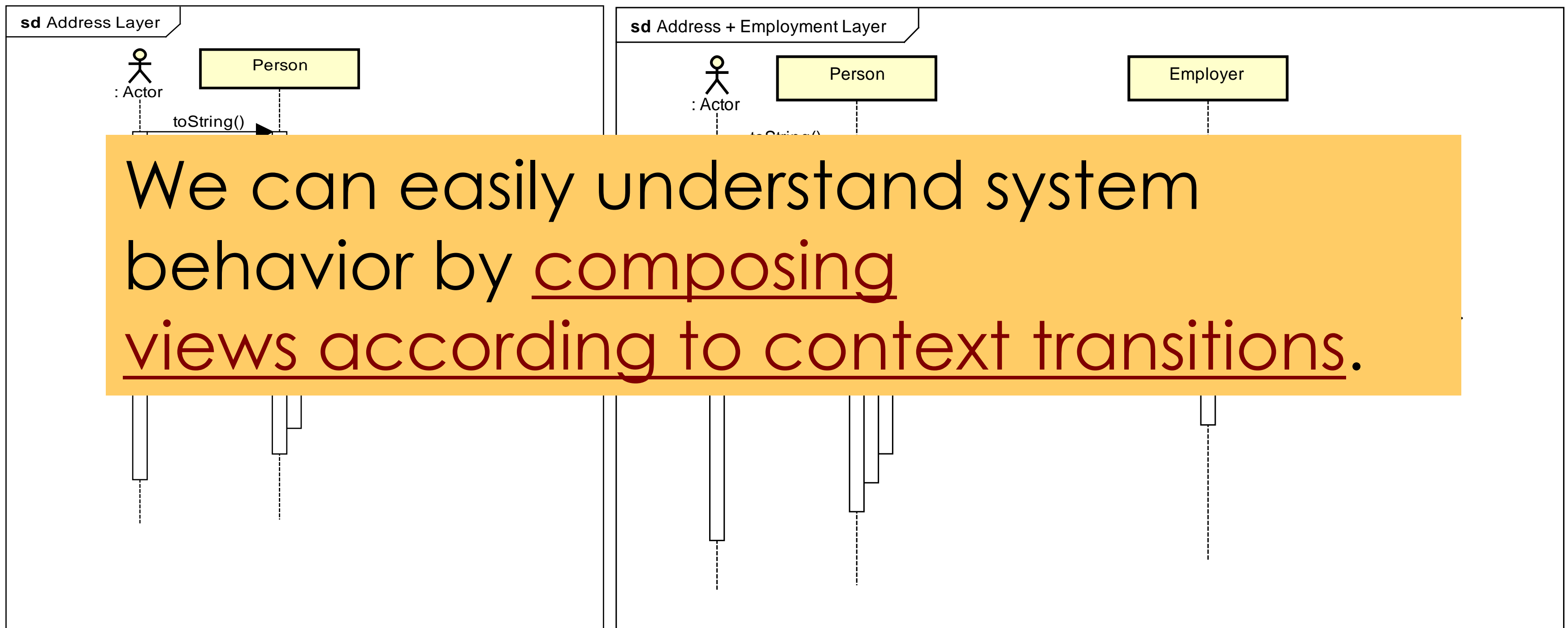
Tarr, P., Ossher, H., Harrison, W., and Sutton, S.M., Jr.:
N Degrees of Separation: Multi-dimensional Separation of Concerns, 21st International Conference on Software Engineering (ICSE'99), pp.107-119, 1999.

Context Transition Model



The order of entering a layer can be specified.

Model Composition



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)

Layer in

```
[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:
08:     with(Layers.Address).eval(new Block() {
09:       public void eval() {
10:         System.out.println(uchio);
11:       }
12:     });
13:
14:     with(Layers.Address,
15:          Layers.Employment).eval(new Block() {
16:       public void eval() {
17:         System.out.println(uchio);
18:       }
19:     });
20:   }
21: }
```

```
[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: }
```

```
[List 3]
01: public class Person implements IPerson {
02:   private String name;
03:   private String address;
04:   private IEmployer employer;
05:
06:   public Person(String newName,
07:                 String newAddress,
08:                 IEmployer newEmployer) {
09:     this.name = newName;
10:     this.address = newAddress;
11:     this.employer = newEmployer;
12:   }
13:
14:   public String toString() {
15:     return layers.select().toString();
16:   }
17:
18:   private LayerDefinitions<IPerson> layers =
19:     new LayerDefinitions<IPerson>(new IPerson() {
20:         public String toString() {
21:           return "Name: " + name;
22:         }
23:     });
24:
25:   {   layers.define(Layers.Employment,
26:       new IPerson() {
27:         public String toString() {
28:           return layers.next(this) +
29:                  "; [Employer] " + employer;
30:         }
31:       });
32:
33:       layers.define(Layers.Address,
34:       new IPerson() {
35:         public String toString() {
36:           return layers.next(this) +
37:                  "; Address: " + address;
38:         }
39:       });
40:   }
41: }
```

Layered Method

```
[List 4]
01: public class Employer implements IEmployer {
02:   private String name;
03:   private String address;
04:
05:   public Employer(String newName,
06:                   String newAddress) {
07:     this.name = newName;
08:     this.address = newAddress;
09:   }
10:
11:   public String toString() {
12:     return layers.select().toString();
13:   }
14:
15:   private LayerDefinitions<IEmployer> layers =
16:     new LayerDefinitions<IEmployer>(new IEmployer() {
17:       public String toString() {
18:         return "Name: " + name;
19:       }
20:     });
21:
22:   { layers.define(Layers.Address,
23:       new IEmployer() {
24:         public String toString() {
25:           return layers.next(this) +
26:                  "; Address: " + address;
27:         }
28:       });
29:   }
30: }
```

Address Layer

Discussion and Future work

Everything is OK?

- An essential problem specific to context-awareness still remains.
- Although a UML4COP model is easy to read, <u>it is not necessarily easy to check whether its program execution is faithful to its requirements (e.g., NFR).</u>

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.

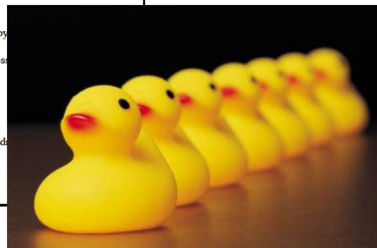
Uchio, S., Ubayashi, N., and Kamei, Y.:
CJAdviser: SMT-based Debugging Support for ContextJ*,
3rd Workshop on Context-Oriented Programming (COP 2011) (Workshop at ECOOP 2011), 2011.

RV4COP

UML4COP
NFR specification

Execution Trace Data

No.	Execution Event (ContextJ*)	Information
01:	[Layer with]	Address
02:	[Method call]	println
03:	[Method execution]	
04:	[Method call]	toString (Person)
05:	[Method execution]	
06:	[Layered method call]	toString (Person's Address layer)
07:	[Layered method execution]	
08:	[Base method call]	toString (Person)
09:	[Base method execution]	
10:	[Layer without]	
11:	[Layer with]	Address
12:	[Layer with]	Employment
13:	[Method call]	println
14:	[Method execution]	
15:	[Method call]	toString (Person)
16:	[Method execution]	
17:	[Layered method call]	toString (Person's Employ
18:	[Layered method execution]	
19:	[Layered method call]	toString (Person's Address
20:	[Layered method execution]	
21:	[Base method call]	toString (Person)
22:	[Base method execution]	
23:	[Method call]	toString (Employer)
24:	[Method execution]	
25:	[Layered method call]	toString (Employer's Add
26:	[Layered method execution]	
27:	[Base method call]	toString (Employer)
28:	[Base method execution]	
29:	[Layer without]	



Logical Formula

+

Logical Formula

SMT Solver

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.