Dynamic Software Evolution-Issues and Approach

Shinichi Honiden, Yasuyuki Tahara

Background: Software evolution

- Software evolution: activity for adapting to requirements changes
  → Play central role in overall software lifecycle

- Recent topics: continuous software evolution
  - **Continuous delivery**
    - Reliable Software Releases through Build, Test, and Deployment Automation
    - Background: continuous evolution to satisfy frequently-changed user requirement

Continuous Delivery Case Studies

Easy: Continuous Innovation with 3 Releases Every Year

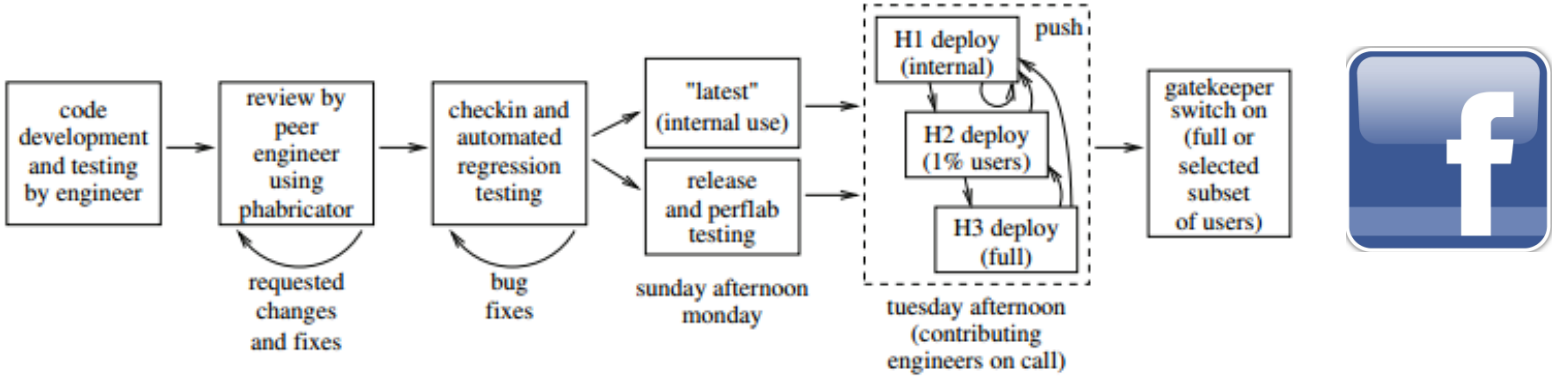
Seamless, Automatic Upgrades

40 Major Releases

Every customization & integration automatically upgraded

Includes features sourced by customer community

salesforce

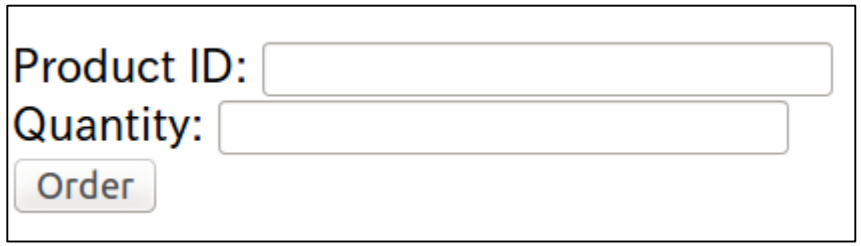


Motivating Example

- Online shopping system
  - Current version: No security
- Evolving two times
  - First evolution: to add the authentication function with IDs and passwords
  - Second evolution: to add the two-factor authentication function requiring users to exchange additional secret codes using smart phone applications or e-mails

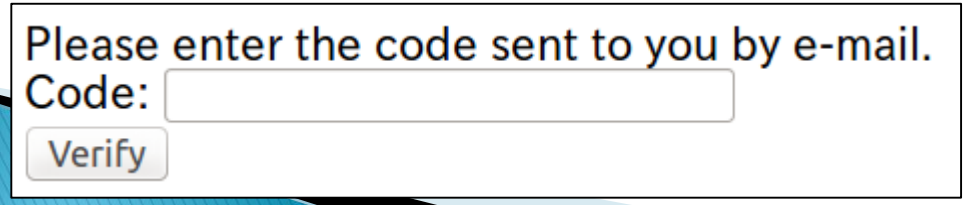
Motivating Example

- Screenshot of browser before evolution



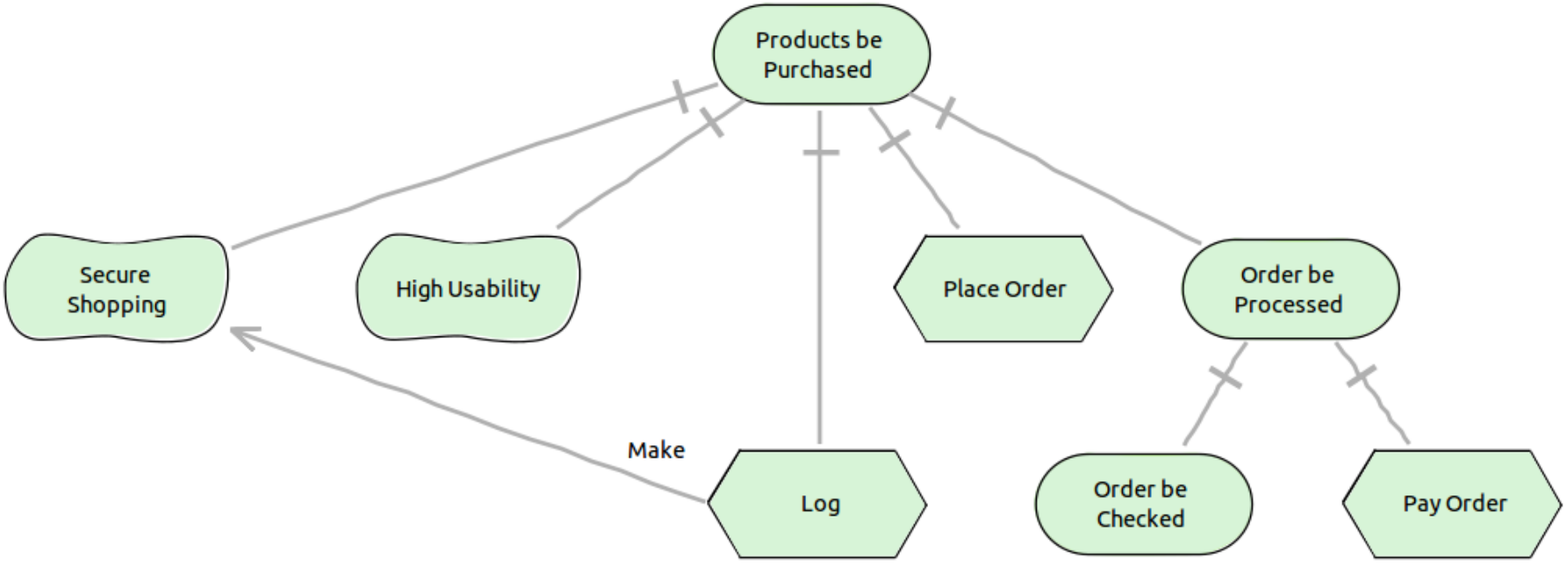
- After the first evolution

- After the second evolution



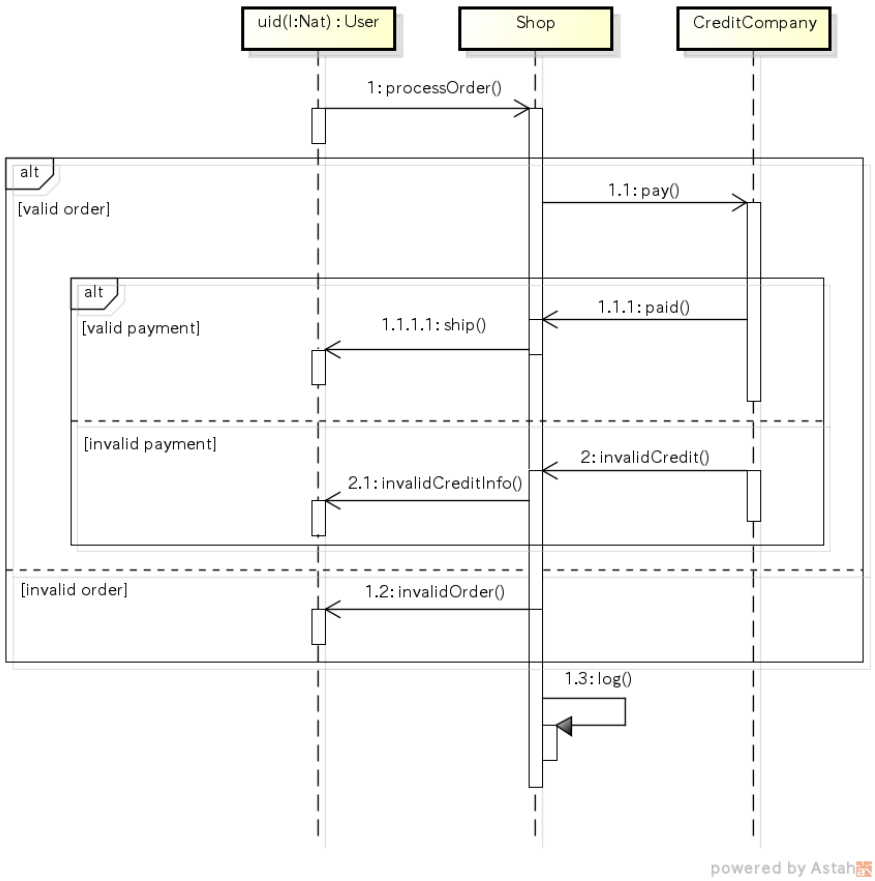
Motivating Example

- Goal model before evolution



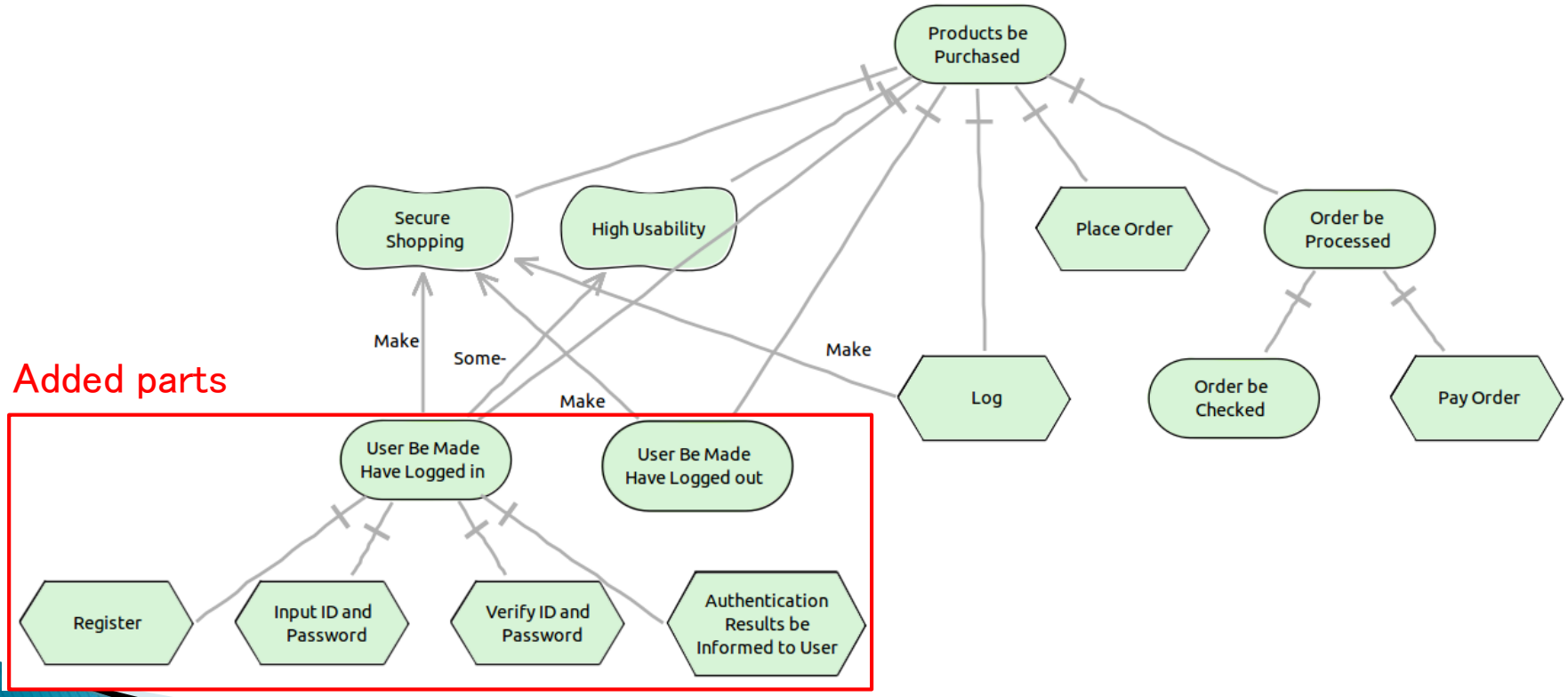
Motivating Example

- Sequence diagram before evolution



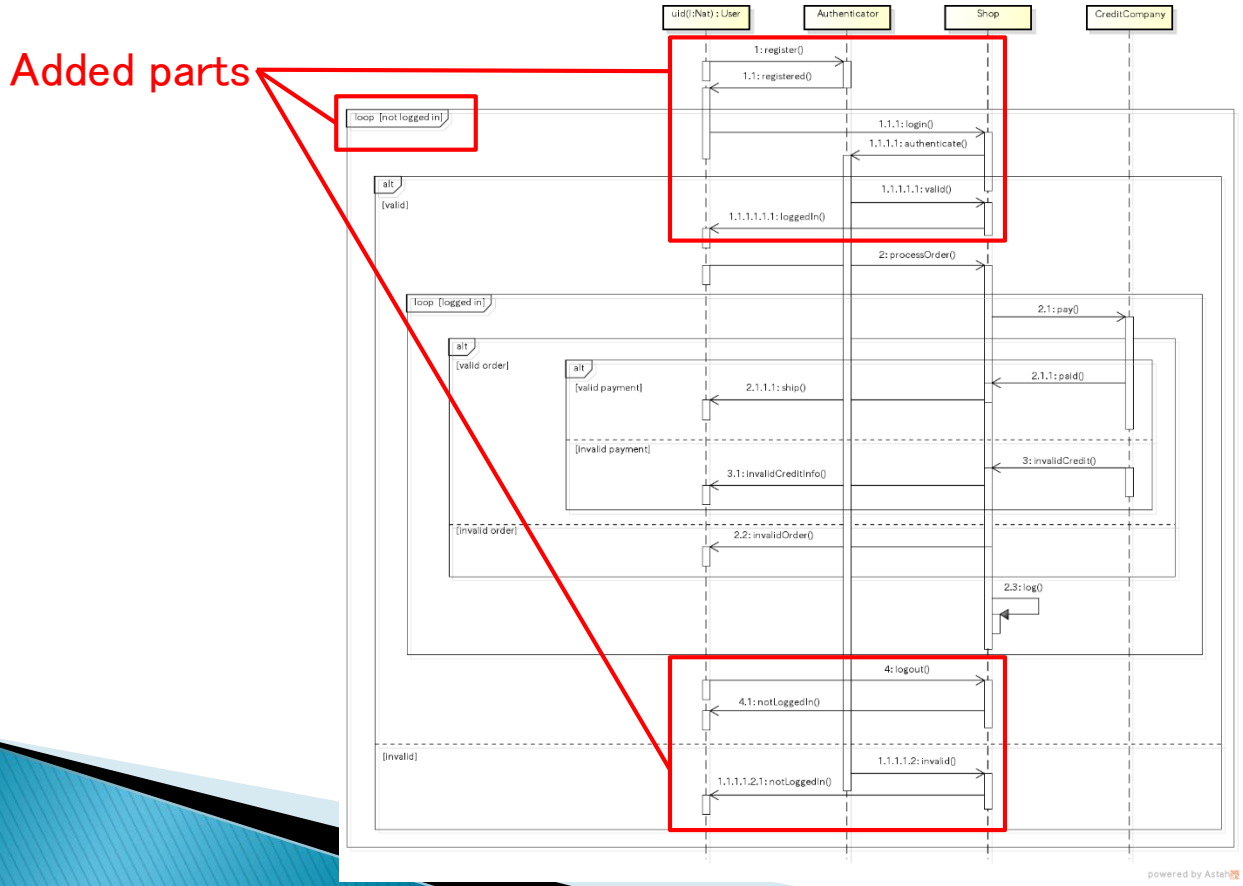
Motivating Example

- Goal model after the first evolution



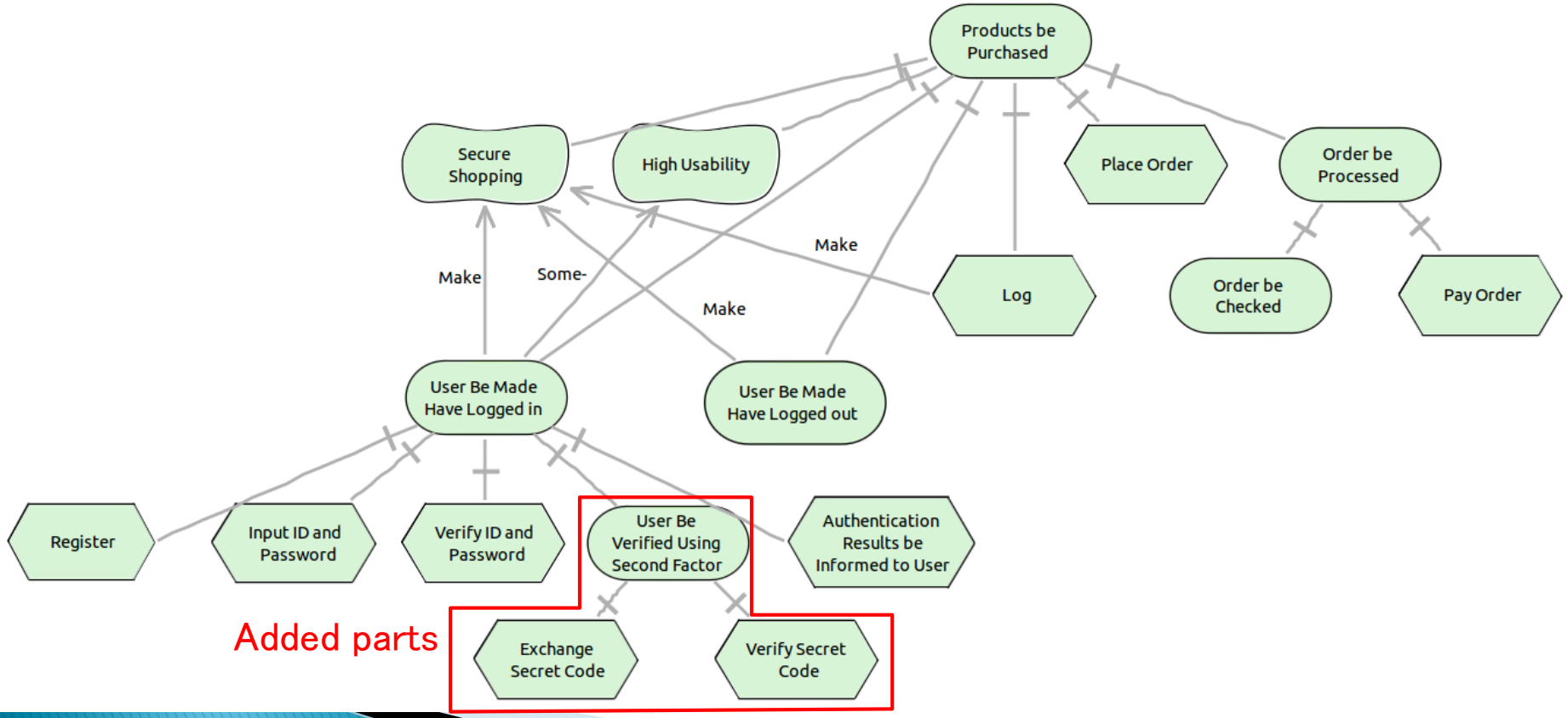
Motivating Example

- Sequence diagram after the first evolution



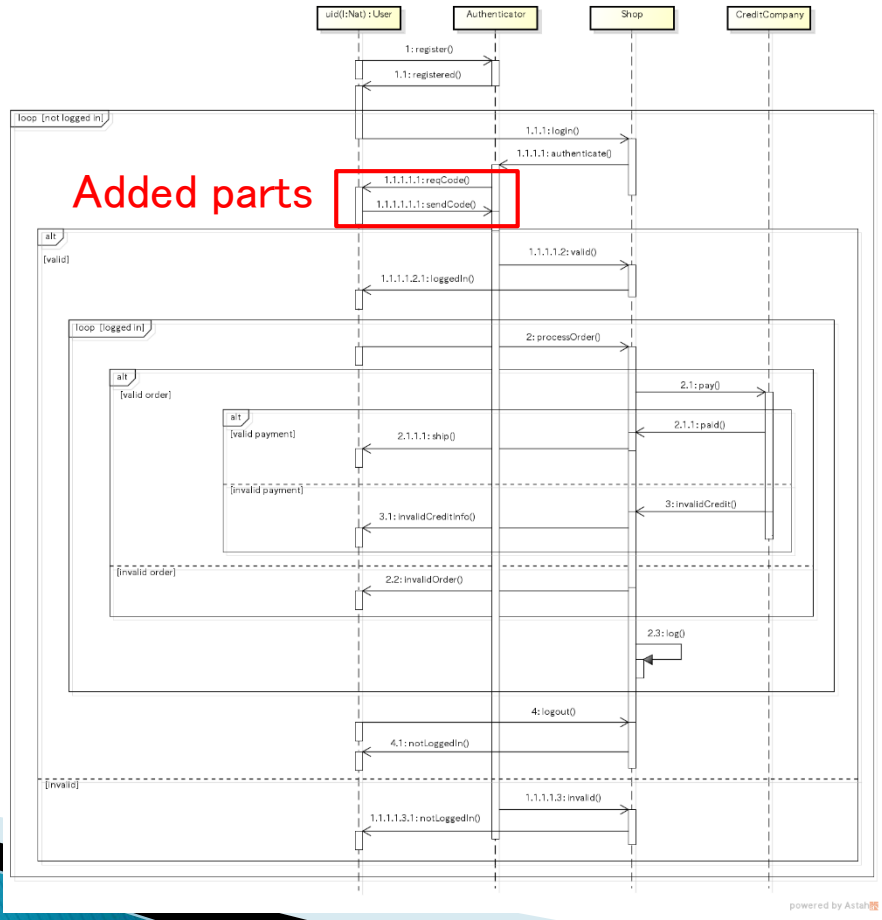
Motivating Example

- Goal model after the second evolution



Motivating Example

- Sequence diagram after the second evolution



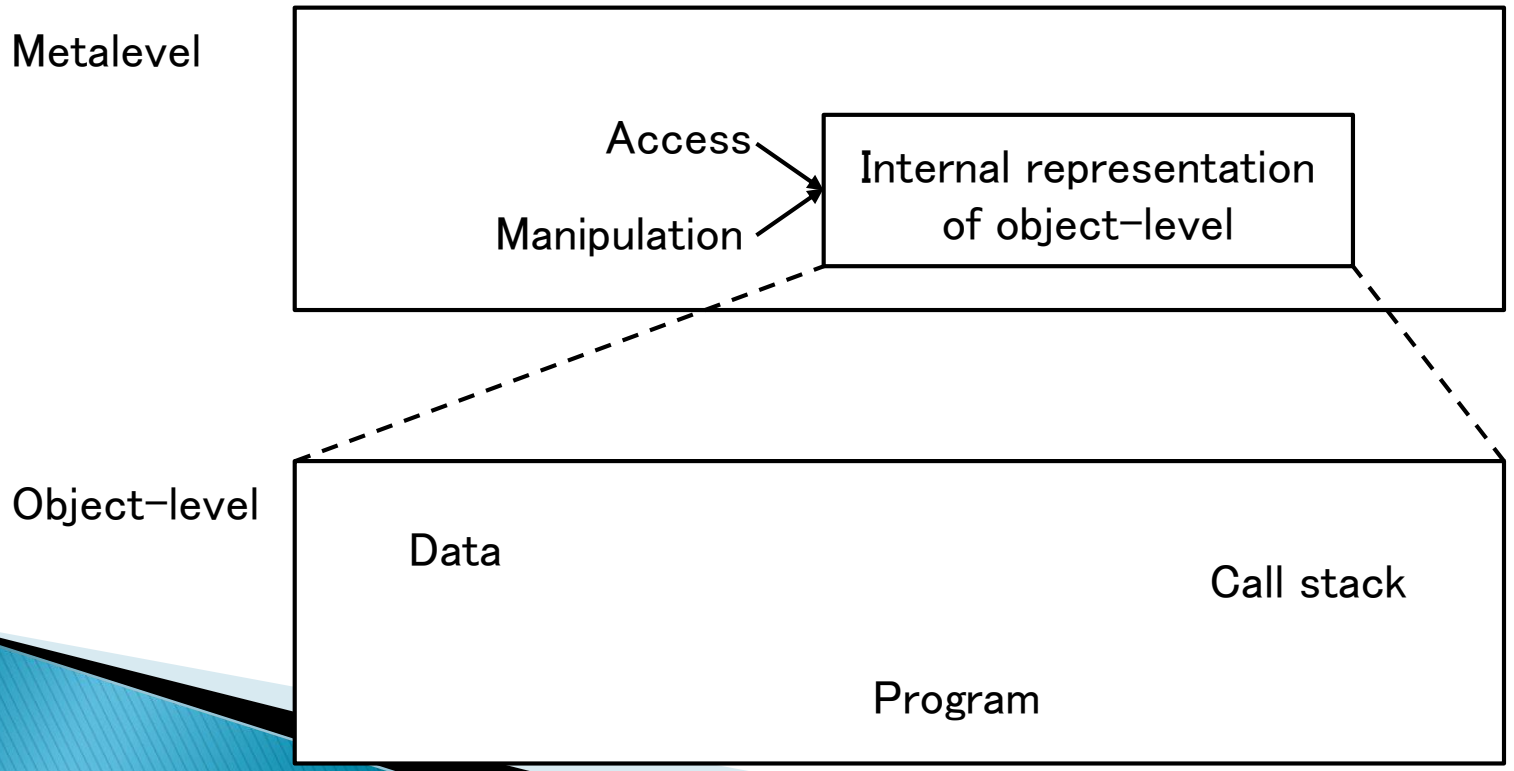
Proposed Approach

- How to implement dynamic evolution?

- Our Approach: use of Javassist that is a class library providing reflection functionalities for Java programs

Backgrounds

- Dynamic evolution using reflection
  - Reflection: System accesses to and manipulates itself from the metalevel to the internal representation of object-level



Backgrounds

- Dynamic evolution using reflection
  - Rewrite programs without interrupting system operation
  - Javassist: Java class library for operations on Java byte code
    - Java programs can rewrite themselves at run time
    - Example of use of Javassist

```
public static void main(String[] args) throws Exception {
    ClassPool cp = ClassPool.getDefault();
    CtClass hs = cp.getCtClass("javax.servlet.http.HttpServlet");
    CtClass sfa = cp.makeClass("jp.ac.uec.tahara.eShop.SecondFactorAuthenticater", hs);
    CtMethod m = CtMethod.make(
            "public static String generateCode() {¥n"
                    // omitted
            + "    }", sfa);
    sfa.addMethod(m);
```

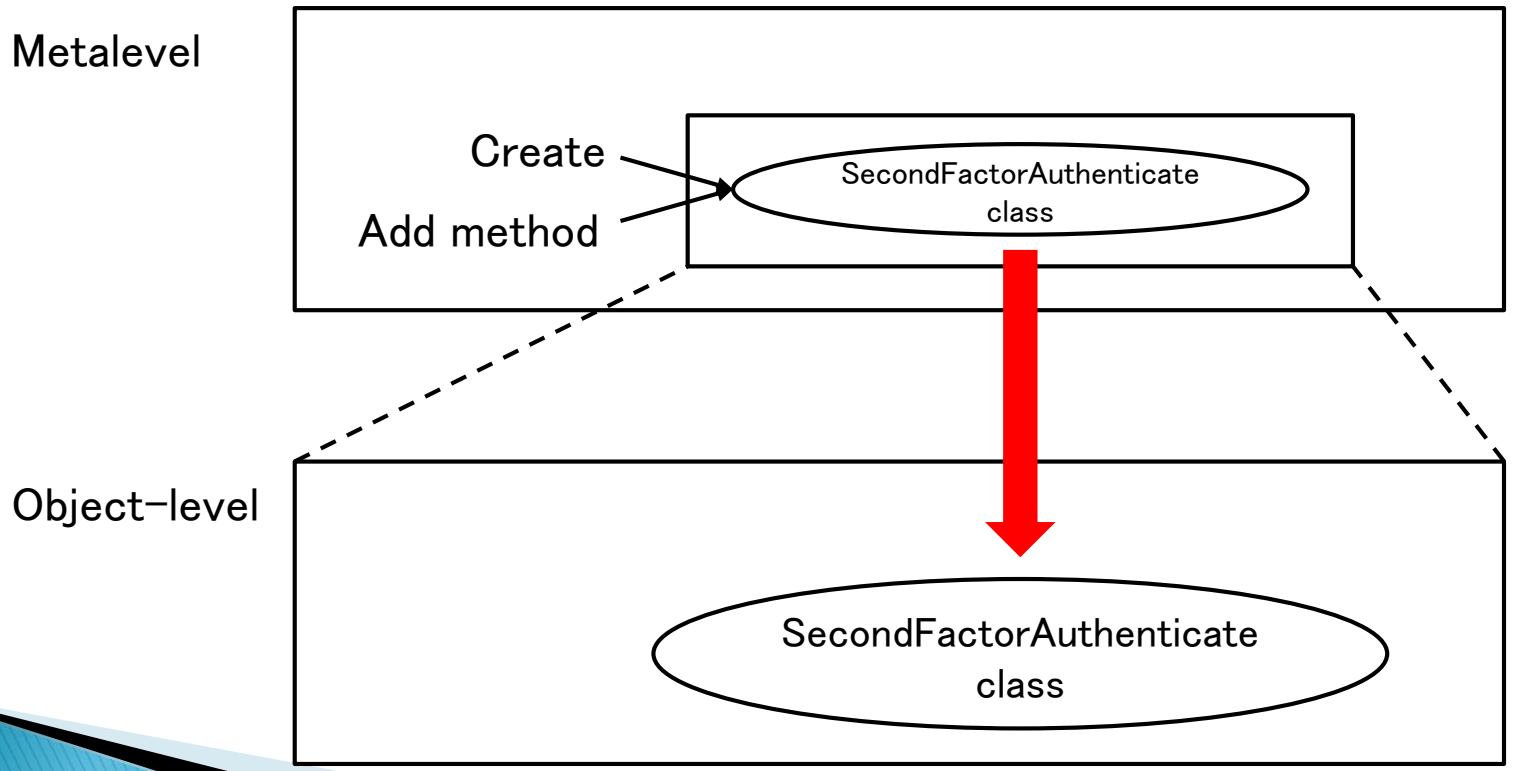
Create a new class

Create a new method

Add the method

Backgrounds

- Dynamic evolution using reflection
  - Example of use of Javassist



Backgrounds

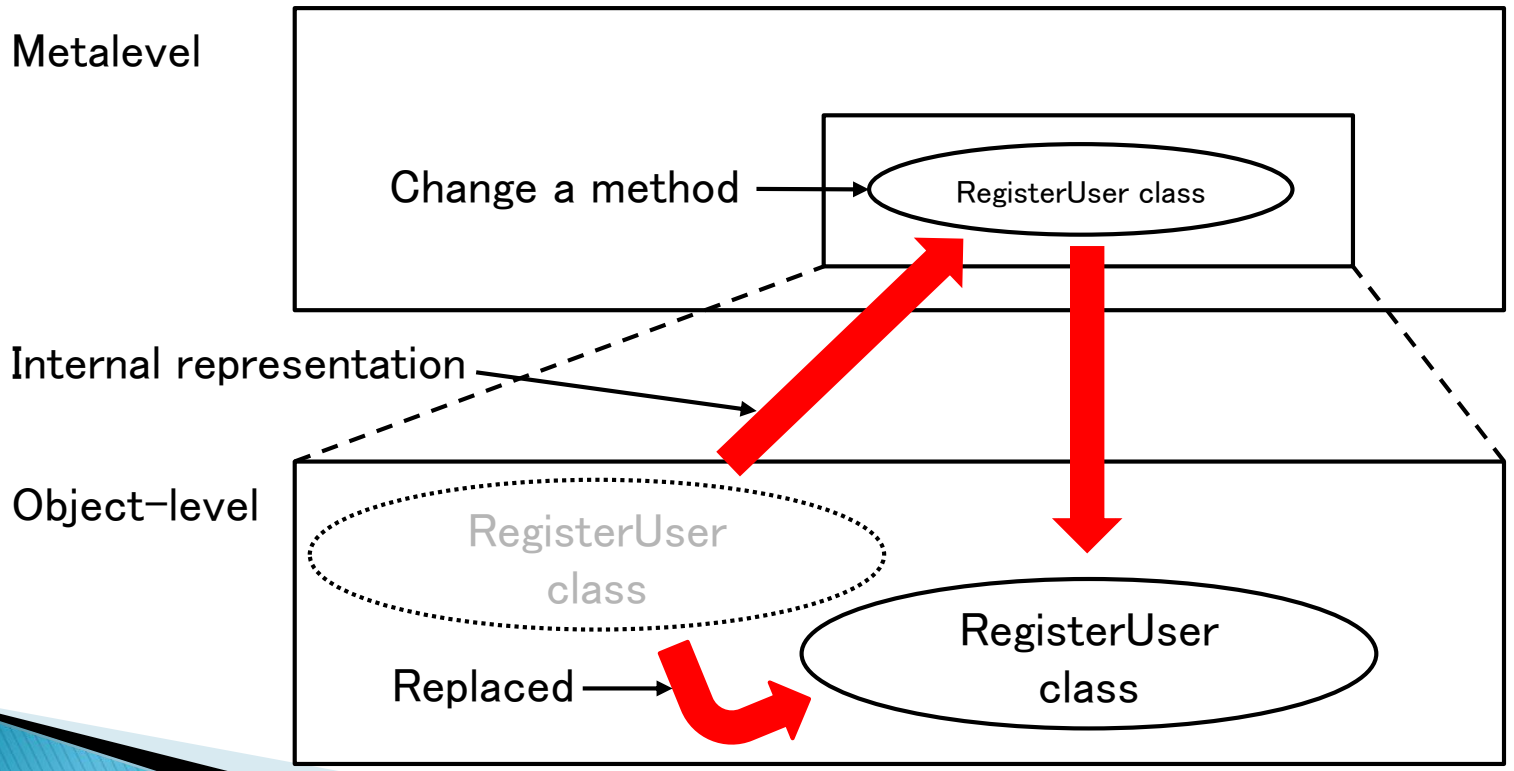
- Dynamic evolution using reflection
  - Example of use of Javassist (cont'd)

Get existing class →
Get existing method →
Create a new method body →
Replaces the method body →

```
CtClass ru = cp.getCtClass("jp.ac.uec.tahara.eShop.RegisterUser");
// omitted
m = ru.getDeclaredMethod("processRequest");
m1 = CtMethod.make(
        "protected void processRequest(HttpServletRequest request,
                                       HttpServletResponse response)¥n"
            // omitted
        + "    }", ru);
m.setBody(m1, null);
```

Backgrounds

- Dynamic evolution using reflection
  - Example of use of Javassist



Backgrounds

- Why reflection?
  - Comparison with other techniques w.r.t. the unit of changes

Techniques	Unit of changes
Design patterns	Classes or methods
Architectural patterns	Components
Autonomic patterns	Resources accessed by actions defined in policies
Middleware-based effectors	Dependent on middleware's functionalities
Dynamic aspect weaving	Aspect
Function pointers	Functions
Reflection	Program of the system itself in detail

  - Reflection is the only technique that enables systems to change their own program in detail

Backgrounds

- Why reflection?
  - Comparison with other techniques w.r.t. the locations of changes

Techniques	Locations of changes
Design patterns	Locations where the patterns are applied
Architectural patterns	Locations where the patterns are applied
Autonomic patterns	Resources accessible by actions defined in policies
Middleware-based effectors	Locations accessible by the middleware
Dynamic aspect weaving	Join points that can be specified by pointcuts
Function pointers	Locations where the functions are called
Reflection	Anywhere in the program

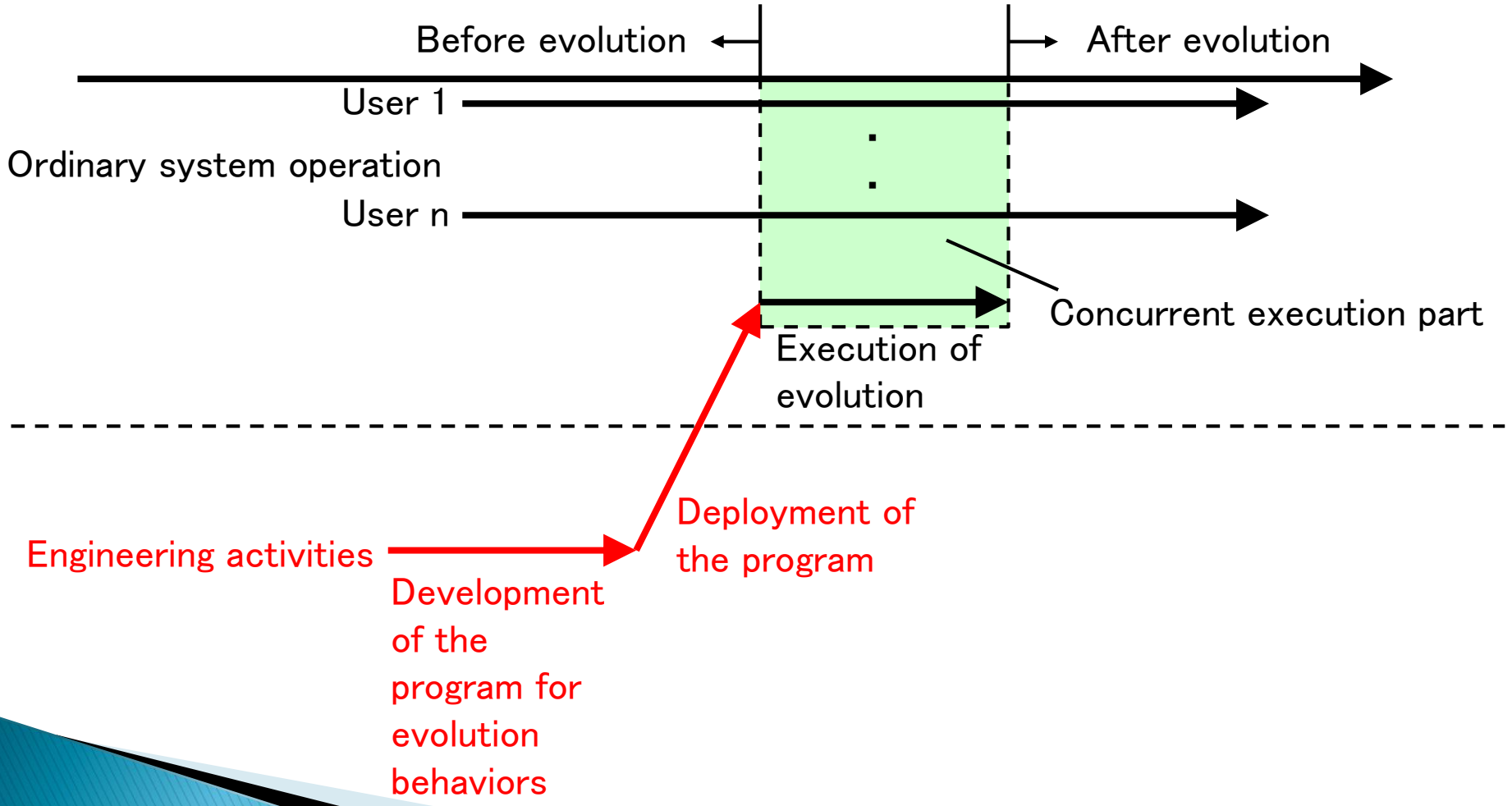
  - Reflection is the only technique that can change anywhere in the program

Introduction

- Needs of dynamic software evolution
  - To deal with rapidly changing requirements and environments
  - Without interruptions of system operation
    - Service-down costs several thousands of dollars per minute[*1][*2]

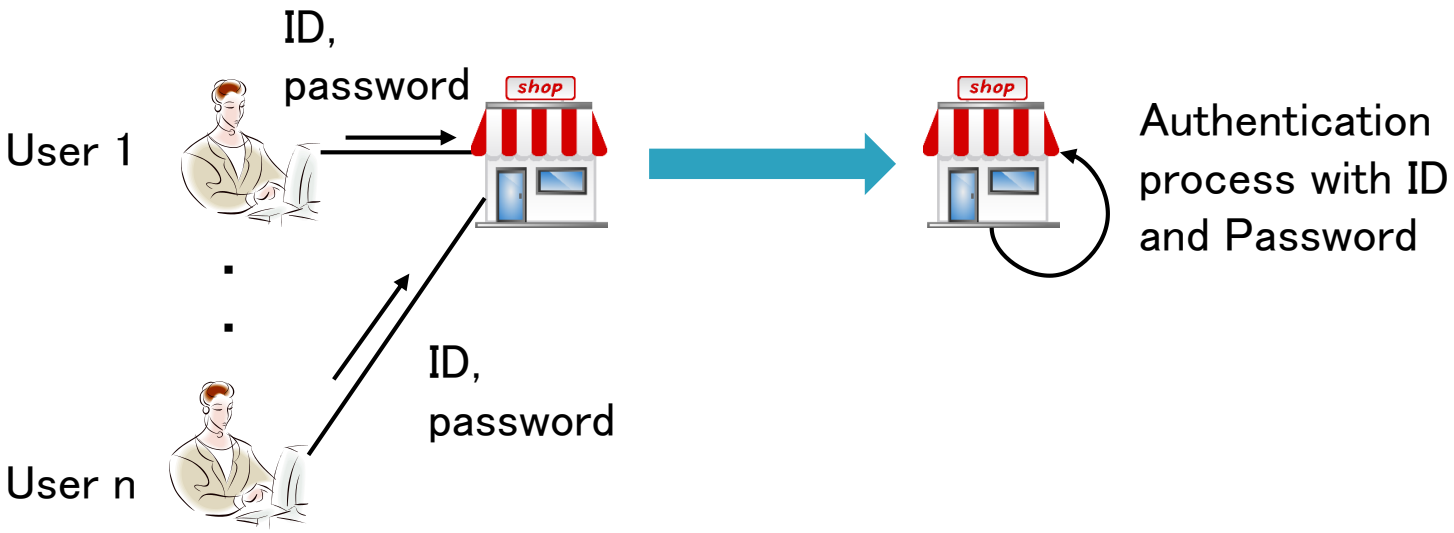
[*1] http://blogs.gartner.com/andrew-lerner/2014/07/16/the-cost-of-downtime/
[*2] http://www.compudata.com/calculating-costs-of-it-downtime/

Dynamic Evolution for Continuous Delivery



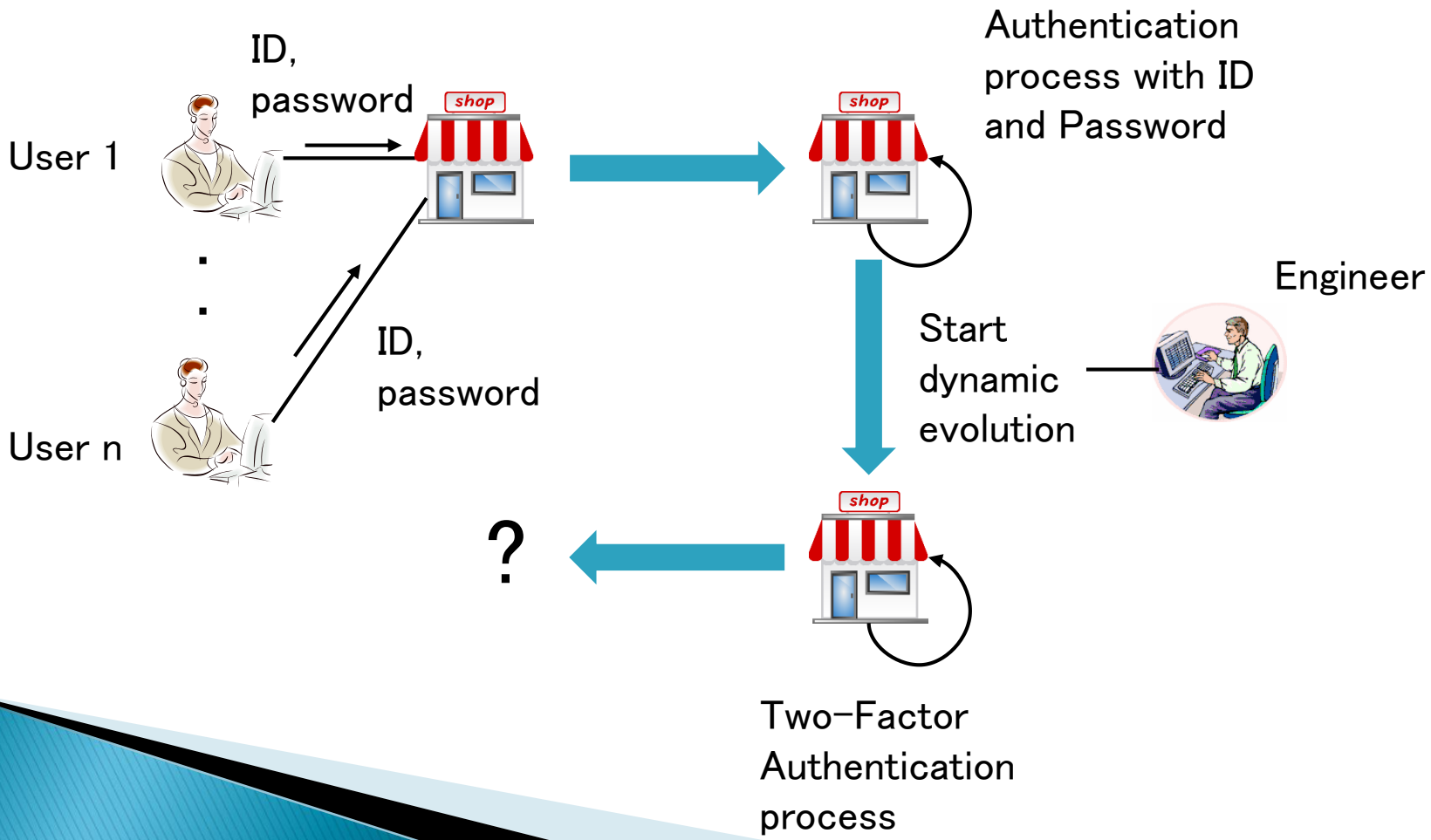
Example of Complicated Behaviors

- In the case of the second evolution



Example of Complicated Behaviors

- In the case of the second evolution



Introduction

- Needs of dynamic software evolution
  - To deal with rapidly changing requirements and environments
  - Without interruptions of system operation
    - Service down costs several thousands of dollars per minute[*1][*2]
- Issue: complicated behaviors
  - Concurrent execution of the ordinary system operations for many users and the evolution behaviors may lead to unexpected states

[*1] http://blogs.gartner.com/andrew-lerner/2014/07/16/the-cost-of-downtime/
[*2] http://www.compudata.com/calculating-costs-of-it-downtime/

Motivating Example

- Online shopping system
  - Current version: No security
- Evolving two times
  - First evolution: to add the authentication function with IDs and passwords
  - Second evolution: to add the two-factor authentication function requiring users to exchange additional secret codes using smart phone applications or E-mails
- Verified property
  - anytime the users can access the shop and the shop properly deals with the users' orders
  - Under the assumption that the system treats all the users fairly (even if more than 100 or 1000 users at the same time)

Proposed Approach

- Issues
  - How to implement dynamic evolution?
    - Our Approach: use of Javassist that is a class library providing reflection functionalities for Java programs
  - How to express the behavior specifications of the dynamic evolution using reflection?
    - Our Approach: use of model checking

Issues of Model Checking

- Concurrent execution of the ordinary system operations and the evolution behaviors
- Various accesses by many users in various timings
  - Before and during evolution

- State space explodes to an enormous size

Proposed Approach

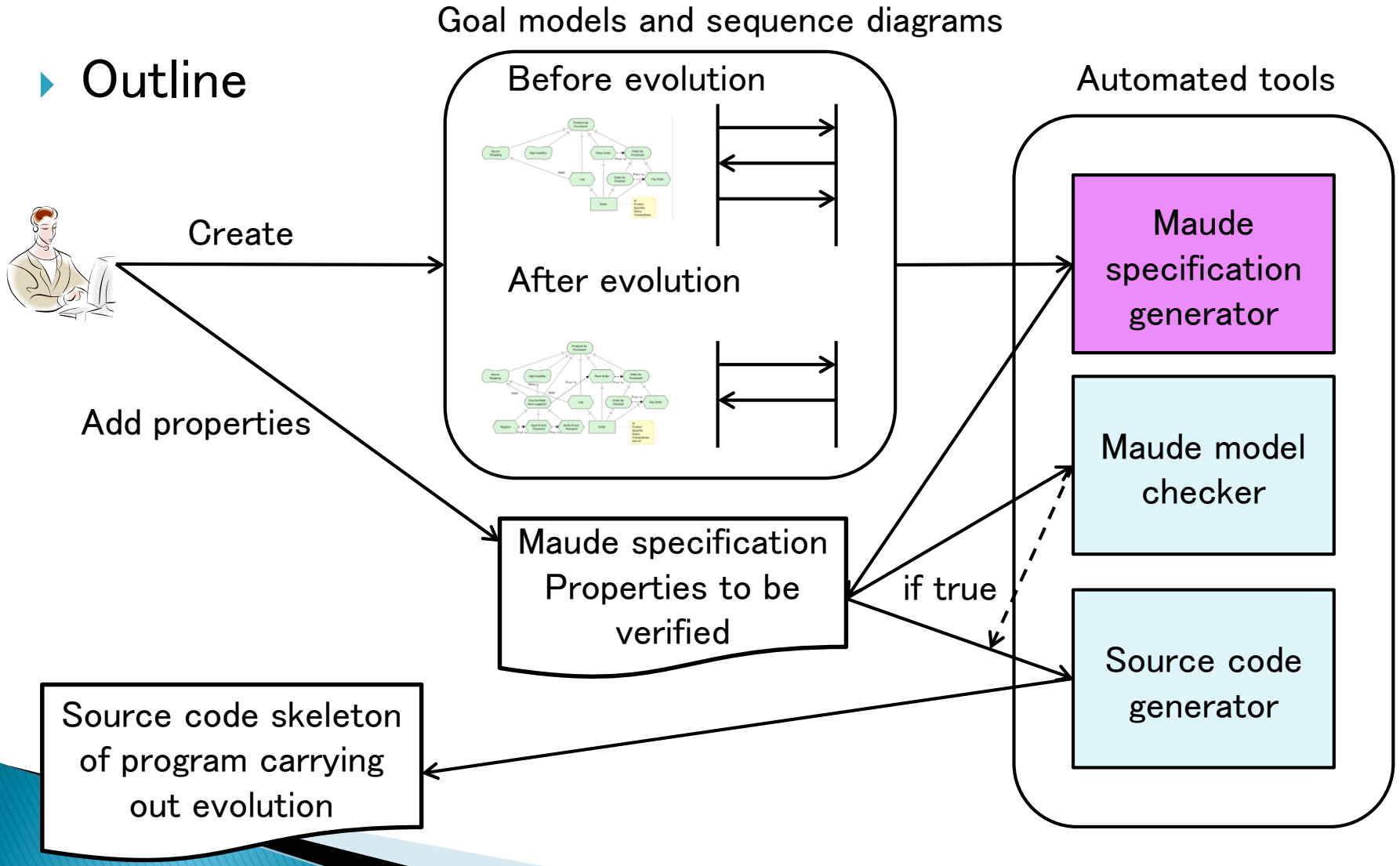
- Model checking would be promising for verification of evolution behaviors
  - Full coverage for possible behaviors
  - Automated verification
- Issues in model checking dynamic evolution
  - Difficult to write behavior specifications
    - Most model checkers cannot deal with dynamic changes of specifications directly
  - State explosion: numbers of states to be explored become enormous for large-scale systems

Maude

- Algebraic specification language
- Useful to write behavior specifications of distributed object-based systems
- Support of reflection
  - Treating constructs of object-level specifications as metalevel terms (representations of data)
  - Metalevel simulates object-level behaviors
- Effective theoretical basis of abstraction
- Model checkers

Proposed Approach

- Outline



Experiments

- First evolution: addition of the authentication functionality
  - Verified property: anytime the users can access the shop and the shop properly deals with the users' orders
    - Under the assumption that the system treats all the users fairly
  - Verification time (in milliseconds)

No. of users	Before evolution	During evolution
1	80	120
2	200	1084
3	2432	42956

Experiments

- Second evolution: addition of the two-factor authentication functionality
  - Verified property: the same
  - Verification time (in milliseconds)

No. of users	Before evolution	During evolution
1	644	696
2	1948	3124
3	43772	117252

Prof. Tahara will present in the next talk

- Details of our proposed approach how to solve issues
  - Procedure
  - Application to the motivating example
  - Theoretical validation of abstraction
- Discussions
  - Advantages and limitations of our proposed approach
  - Comparison with other approaches
  - Future work