Validation of Requirement Models by Automatic Prototyping

Xiaoshan Li

Department of Computer and Information Science

Faculty of Science and Technology

University of Macau, Macao, China

Email: xsl@umac.mo

Joint work with **Dan Li, Jicong Liu and Zhiming Liu**

International Institute for Software Technology

United Nations University (UNU/IIST)

Motivation

- **Problem**: Early acquired requirements are difficult to validate.
- **Solution**: Prototyping is efficient and effective to expose errors in the early stages of requirements analysis and design.
- **Method**: Automatically generate a prototype from system requirements model

  (***use case model*** and ***conceptual class model***)

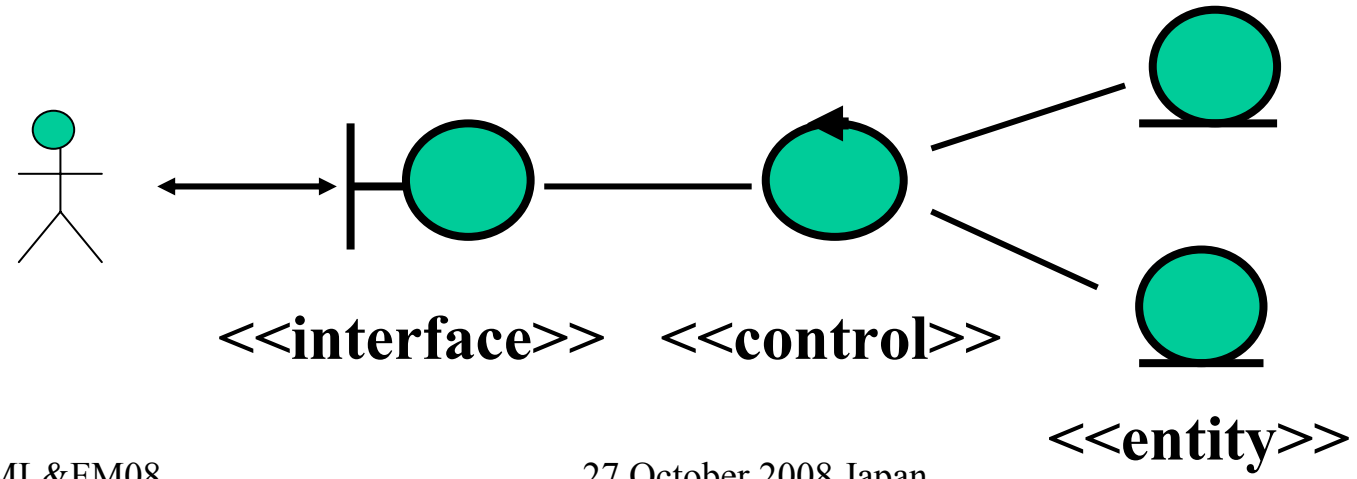
Requirements Model

1. Use Case Model
2. Conceptual Class Model

Use-case → System operations

→Primitive actions (in pre and post conditions)

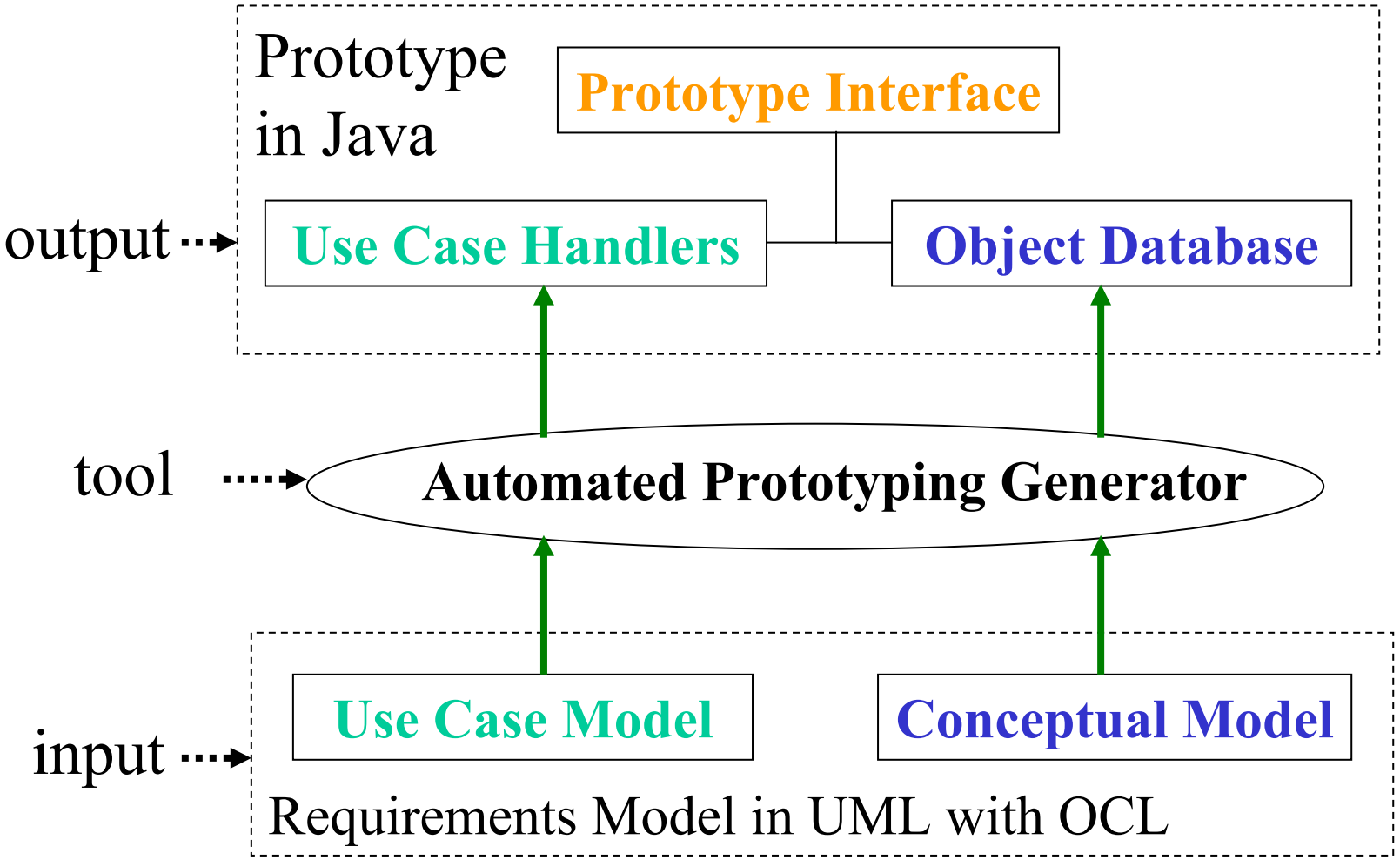
→ Entity object methods (new, get, set …)



Method

- Generate a system entity object database from the semantics of the conceptual class model.
- Generate a system interface which displays the buttons for the use case names in use case model.
- Generate a use-case handler for each use case to handle the execution of specification of the use case.
- A use case handler maps the executable parts of each use case into a sequence of primitive actions on the system entity database.

Architecture of Generating Prototype



UML Requirement Model

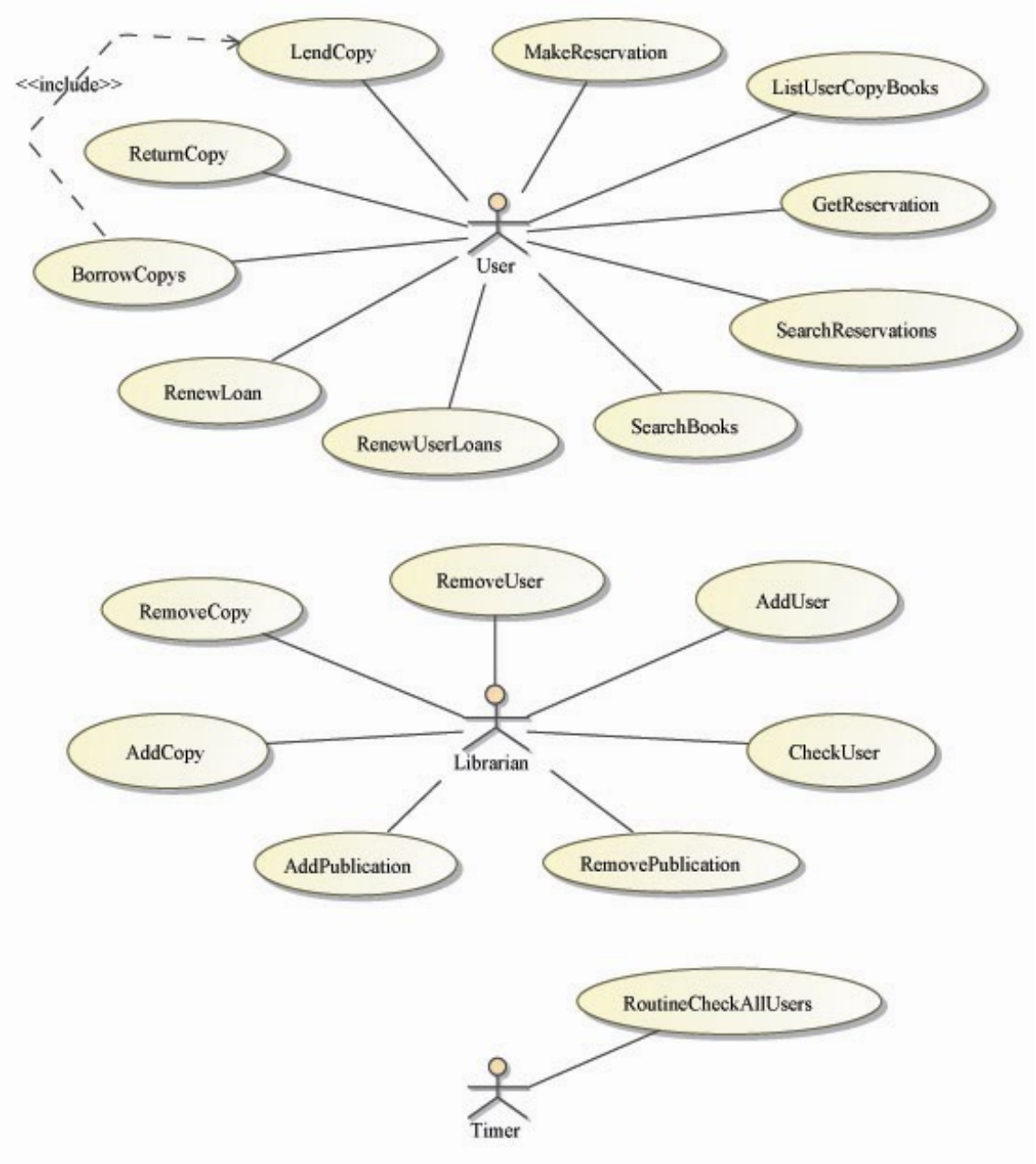
Requirements Model = (UCM, CCM)

UCM (Use Case Model) & CCM (Conceptual Class Model)

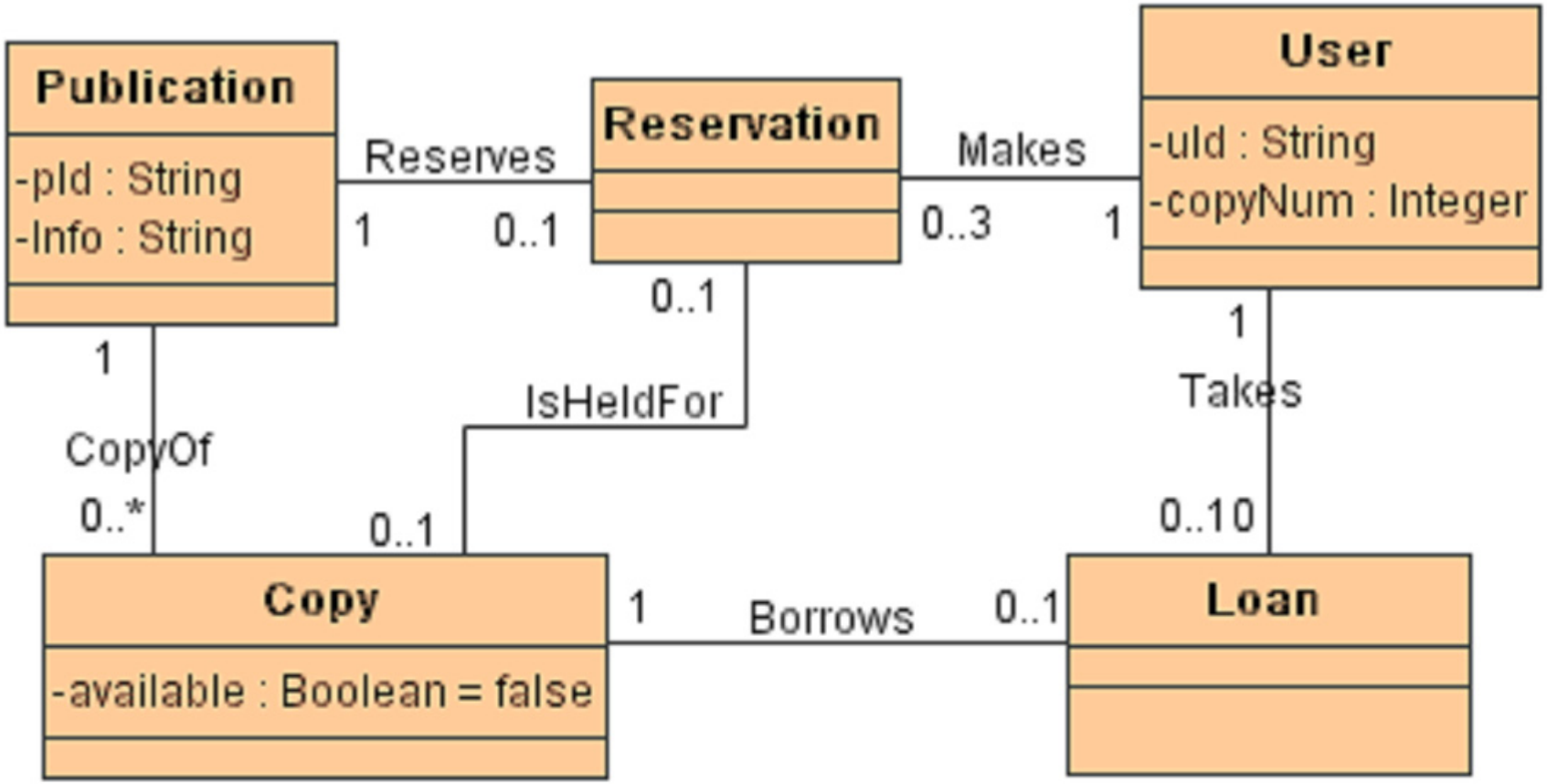
A *step* of a use case (*system operation*) is specified as a *contract* with a pair of pre and post conditions.

***operation*: $p(x) \grave{} R(x, x') =_{def} p(x) \rangle R(x, x')$**

Use Case Diagram of Library System



Conceptual Class Diagram of Library



Conceptual Class Model of Library

CN = { **User, Copy, Publication, Reservation, Loan**}

Attr(**User**) = {<*cid*: **String**>, <*copyNum*: **Integer**>}

Attr(**Copy**)={<*cid*:**String**>, <*available*: **Boolean**>}

AN={**CopyOf**:<**Copy, Publication**>,

Takes: <**User, Loan**>,

Borrows:<**Loan, Publication**>,

Makes:<**User, Reservation**>,

Reserves:<**Reservation, Publication**>,

IsHeldFor: <**User, Reservation**>}

Formal Specification of *LendCopy*

LendCopy = def **pvar** *copy*: **Copy**, *user*: **User**

pre: *copy* ∈ *Copy* /\ *copy.available* = *true* /\

user ∈ *User* /\ *user.copyNum* <10

post: ∃ *loan*: **Loan**. (*loan not* ∈ *Loan*)

/\ *Loan'*=*Loan* ∪ { *loan* }

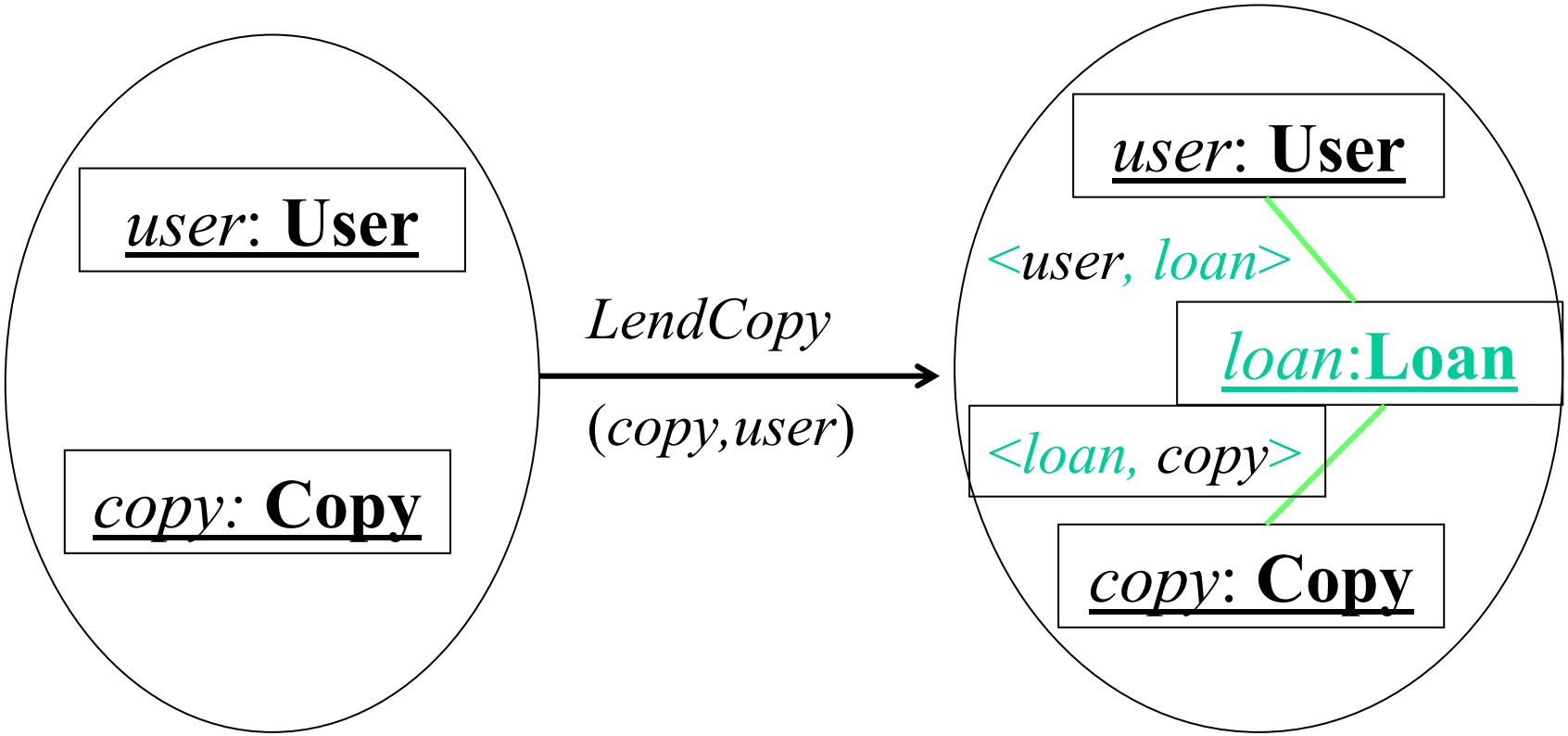
/\ *Borrows'*=*Borrows* ∪ {<*loan, copy*>}

/\ *Takes'* = *Takes* ∪{<*user, loan*>}

/\ *copy.available'*= *false*

/\ *user.copyNum'*=*user.copyNum*+*1*

Pre & Post State Object Diagrams of *LendCopy*



OCL Specification of *LendCopy*

context Usecase::LendCopy(user:User,copy:Copy) :Loan

***pre**: User ->includes(user) and user.copyNum < 10*
and Copy ->includes(copy) and copy.available = true

***post**: user.copyNum = user.copyNum**@pre** + 1*
and copy.available = false and result.oclIsNew() and
result.Borrows = copy and result.Takes = user

Generate Executable Prototype

- Prototyping of requirements analysis should demonstrate the important functional effects of each use case in terms of atomic state changes rather than the detailed algorithms for internal object interactions.

- Primitive actions are ***create an object*** or ***a link***, ***remove an object*** or ***a link***, ***check*** and ***set object attributes***.

Find Objects and Links in Pre & Post Conditions

- **preobject = {*user, copy, copy.available= ture,***
  ***user.copyNum < 10* } ,**
- **prelink = ∅ ,**
- **postobject = {*user, copy ,loan, copy.available'=false,***
  ***user.copyNum'=user.copyNum+1,* },**
- **postlink = {<*user, loan*> , <*loan, copy*>}**

- **createobject = postobject – preobject ={*loan*}**
- **checkattributes ={*c.available= ture, u.copyNum < 10* }**
- **removeobject = preobject – postobject = ∅**
- **createolink = postlink–prelink = {<*user, loan*> , <*loan, copy*>}**
- **removelink = prelink – postlink = ∅**
- **setattributes = {*copy.available'=false,***
  ***user.copyNum'=user.copyNum+1*}**

Algorithm of Source Code Generation

1. Check existing objects of *preobject* set in pre-state.
2. Check existing links of *prelink* set in pre-state.
3. Check conditions about attributes of objects in *preobject*.
4. Create objects of *createobject* set in post-state.
5. Create links of *createlink* set in post-state.
6. Set attributes to force the conditions satisfied in *postobject* set.
7. Remove links of *removelink* set in post-state.
8. Remove objects of *removeobject* set in post-state.

Rules for Transforming OCL Expressions to Primitive Actions

Ten rules can be used to transform OCL expression fragments in AST to primitive actions.

Context ***Usercase:: AddCopy*****(*****copy:*****Copy, *****pub:*****Publication)**

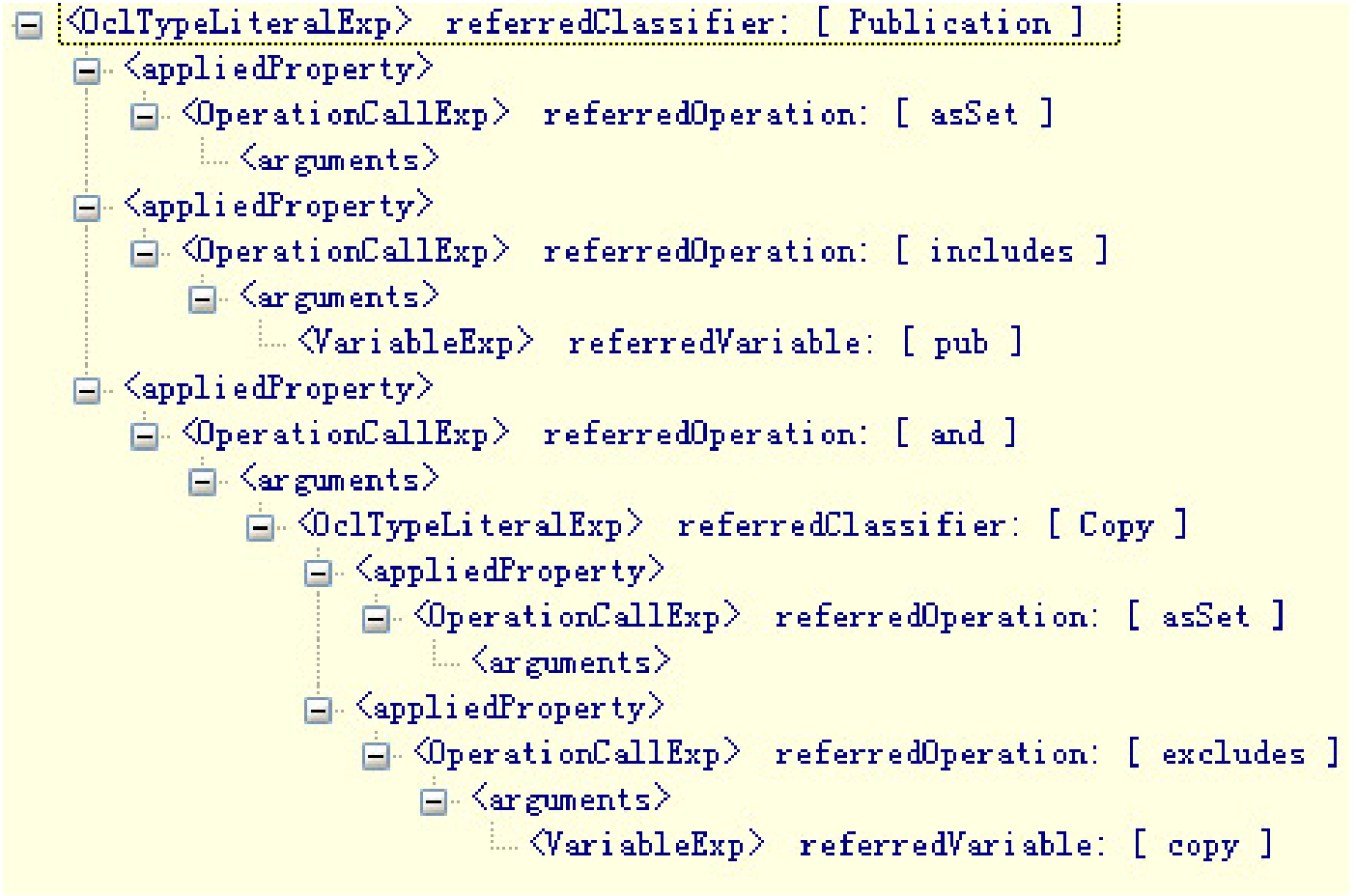
Pre***: Publication->includes(pub)***

and Copy->excludes(copy)

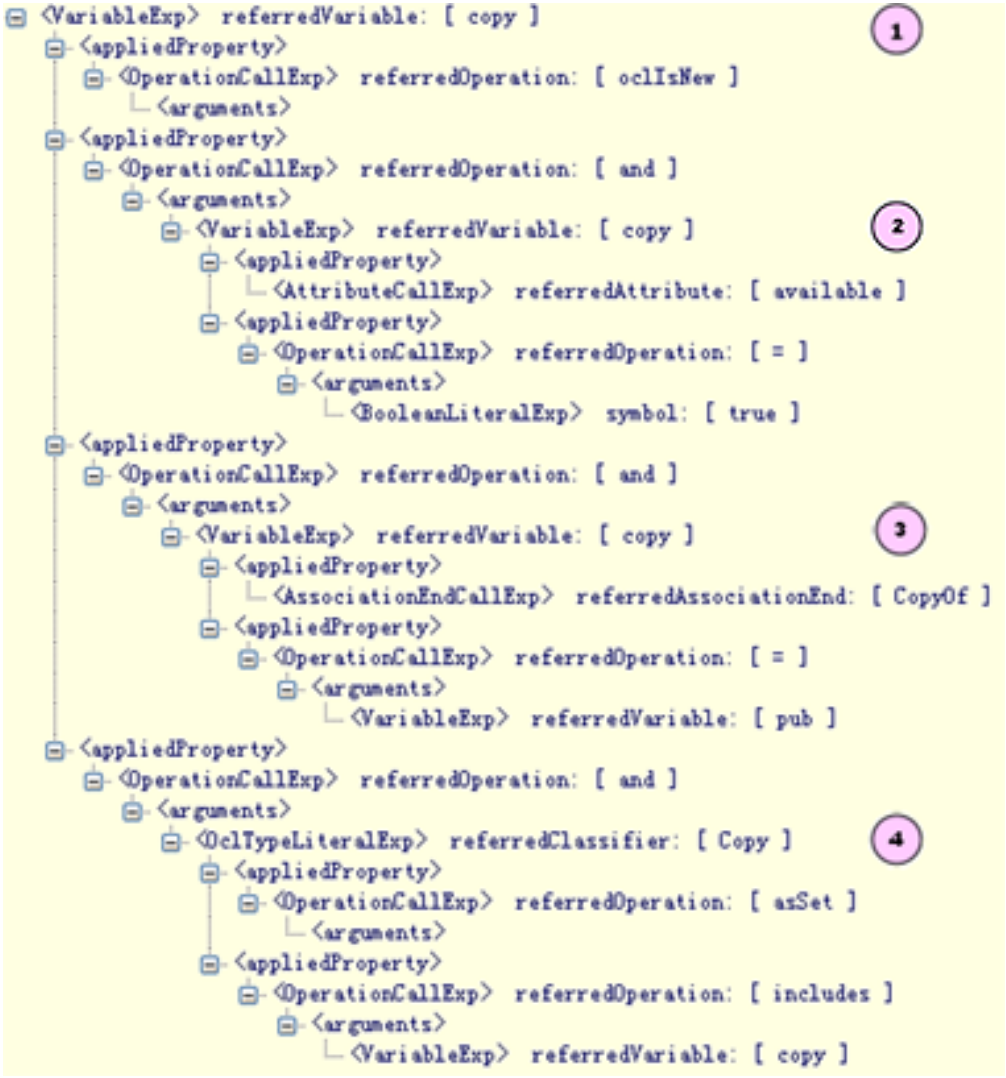
Post***: copyoclIsNew() and copy.available=true***

and copy.CopyOf=pub

Precondition AST of *AddCopy*



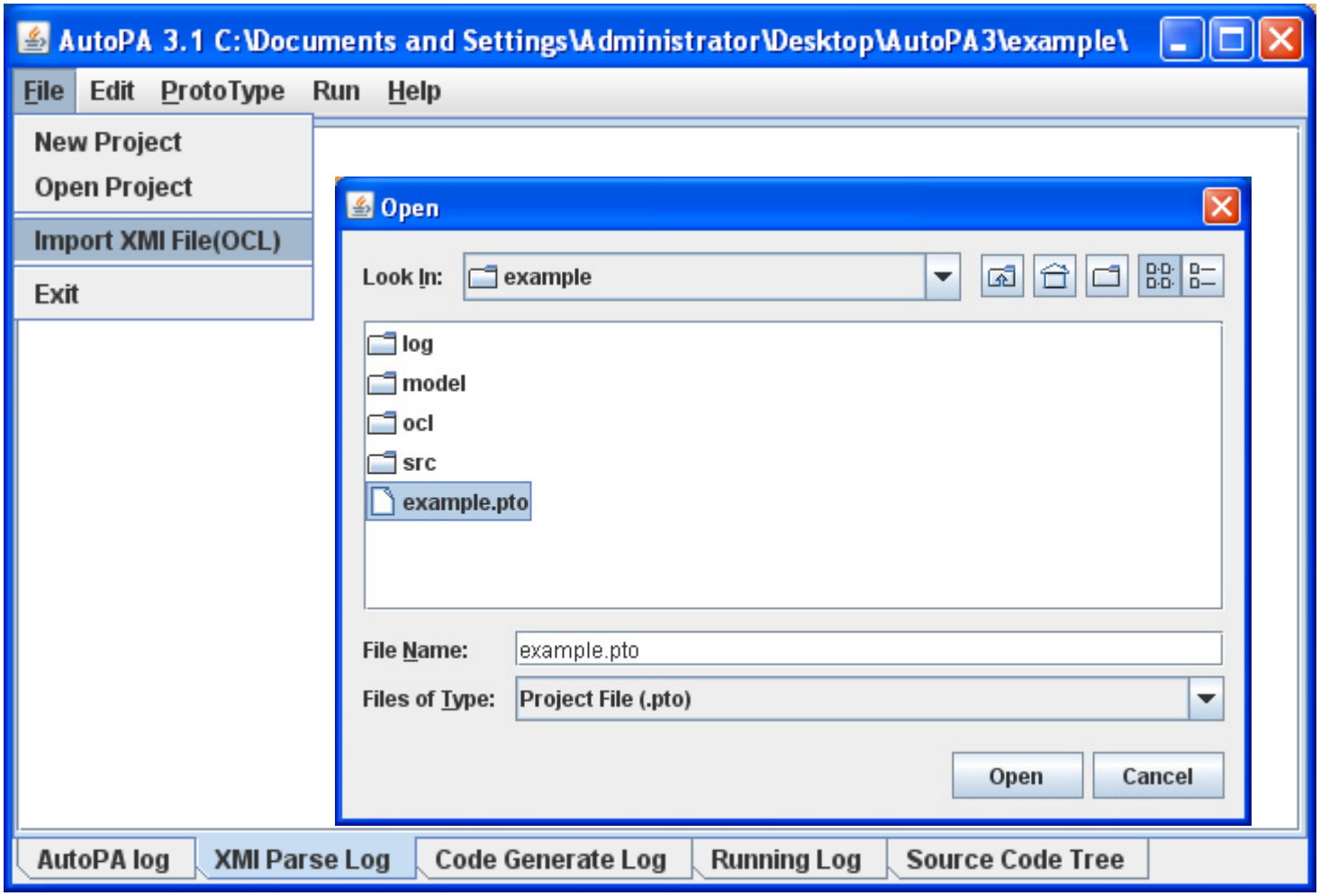
Postcondition AST of *AddCopy*



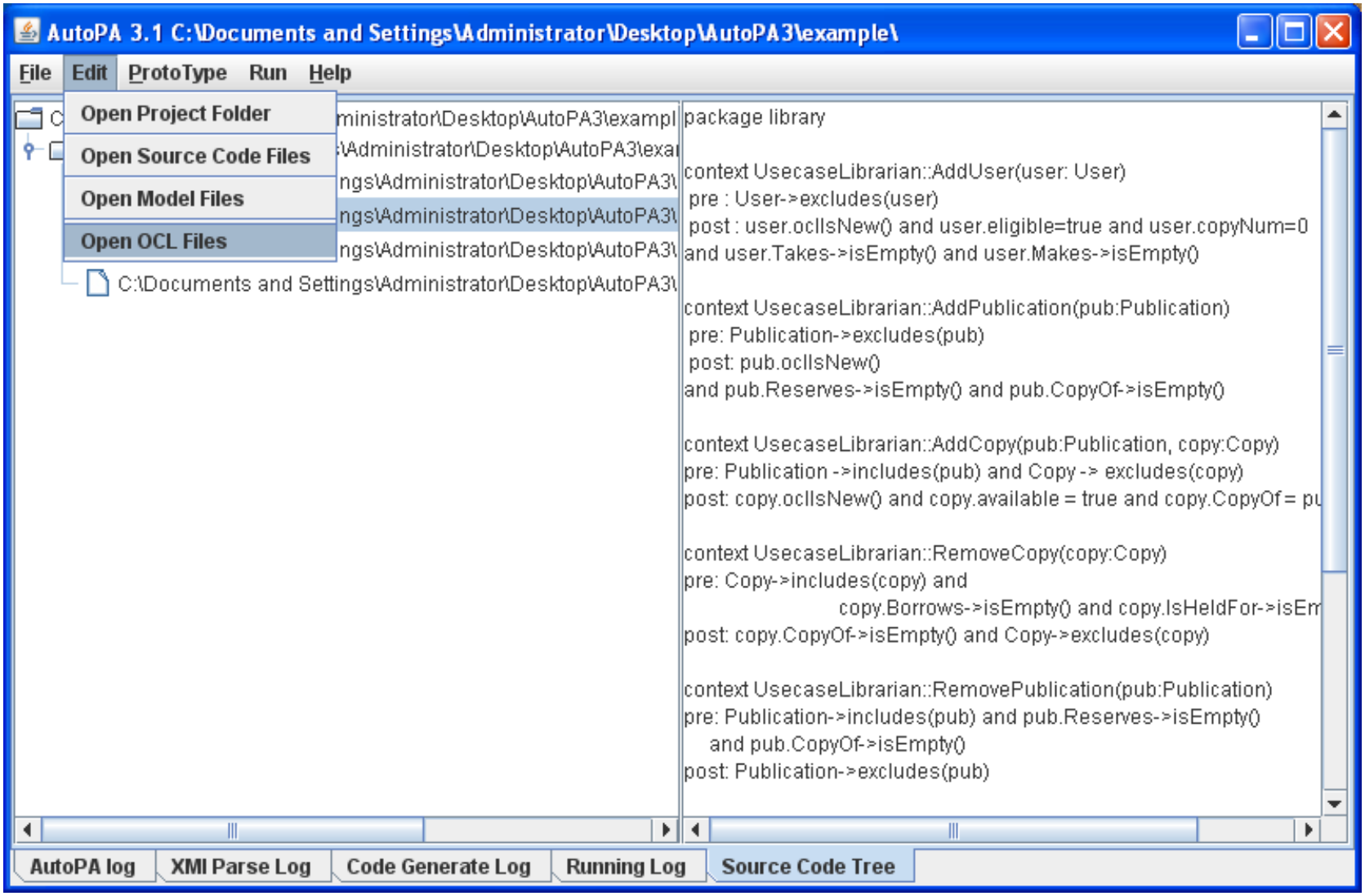
Primitive Actions of AddCopy

1. *FindObject(pub, Publication)*
2. *FindNoObject(copy, Copy);*
3. *CreateObject(copy, Copy);*
4. *SetAttr(copy, available, true);*
5. *CreateLink(CopyOf, copy, pub)*

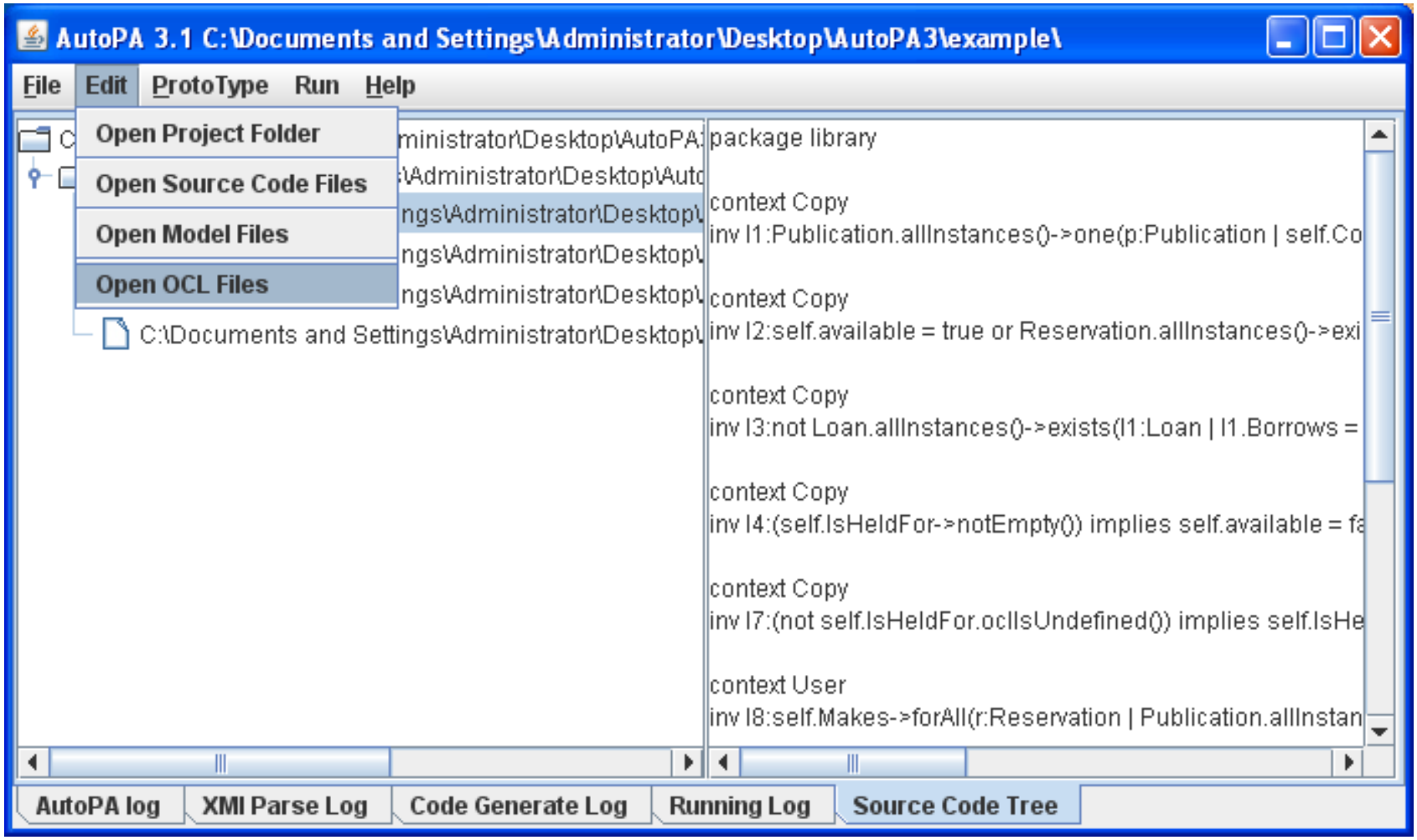
Interface of Prototype Generator Tool



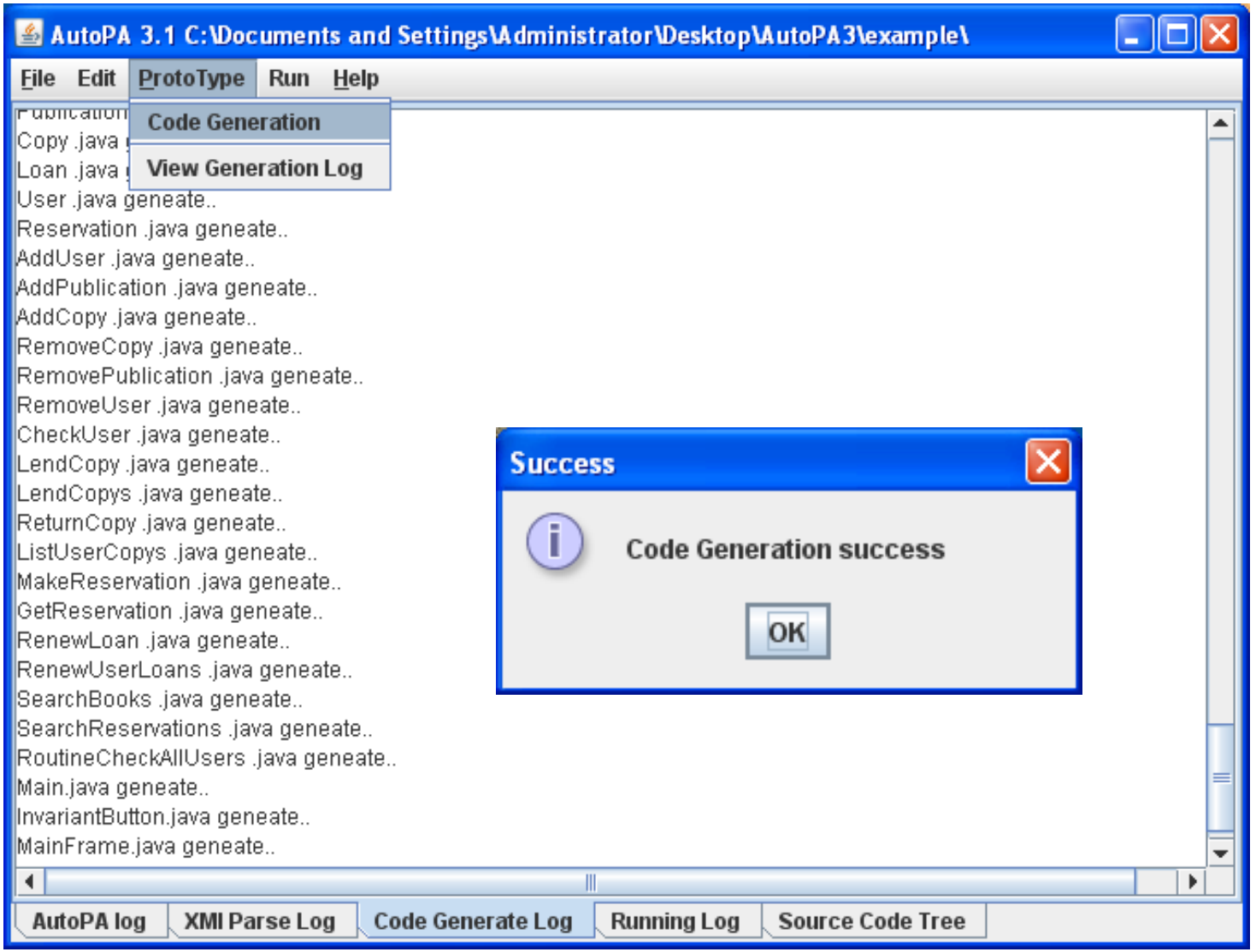
OCL Pre and Post Conditions



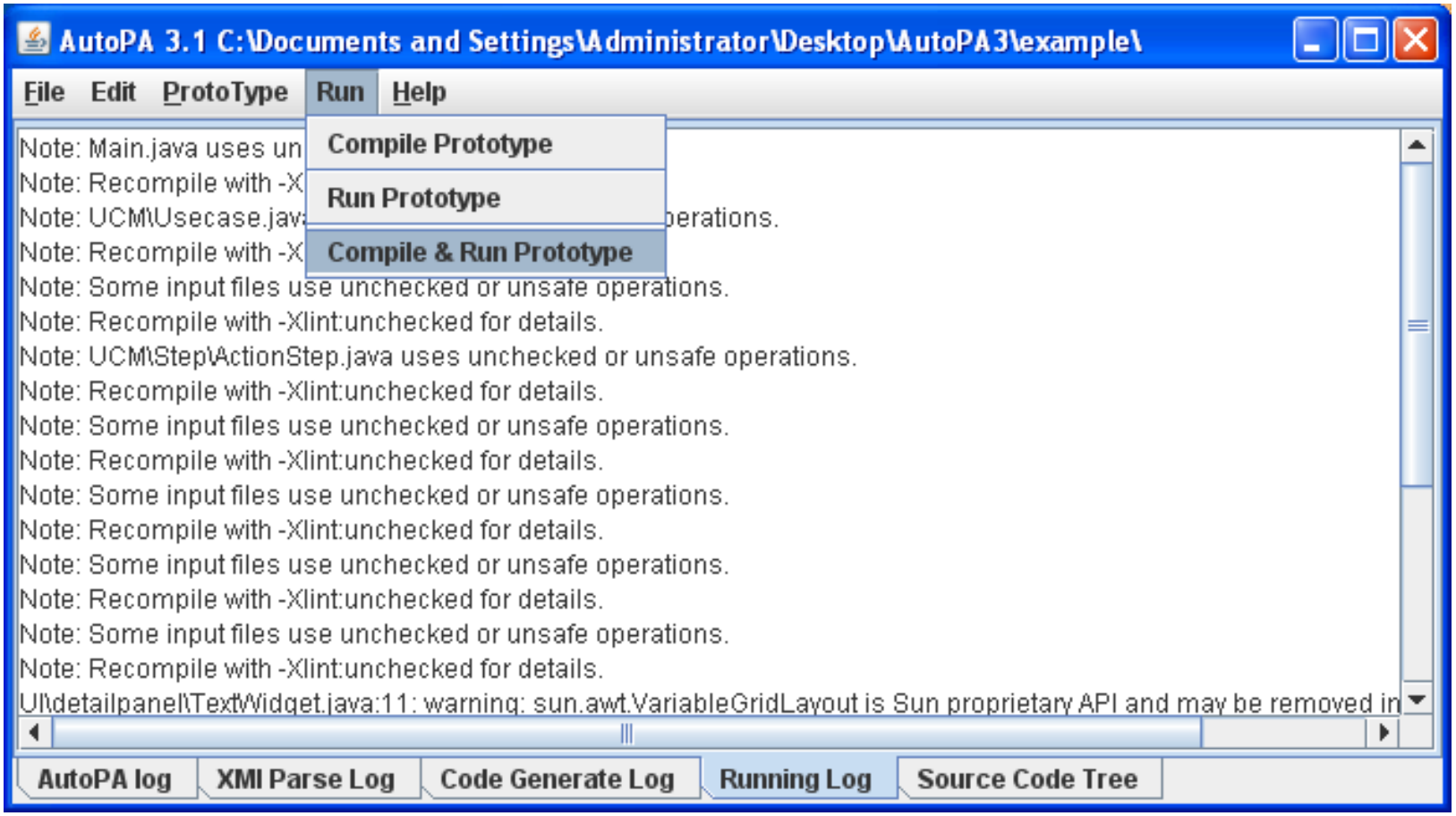
System Invariants in OCL



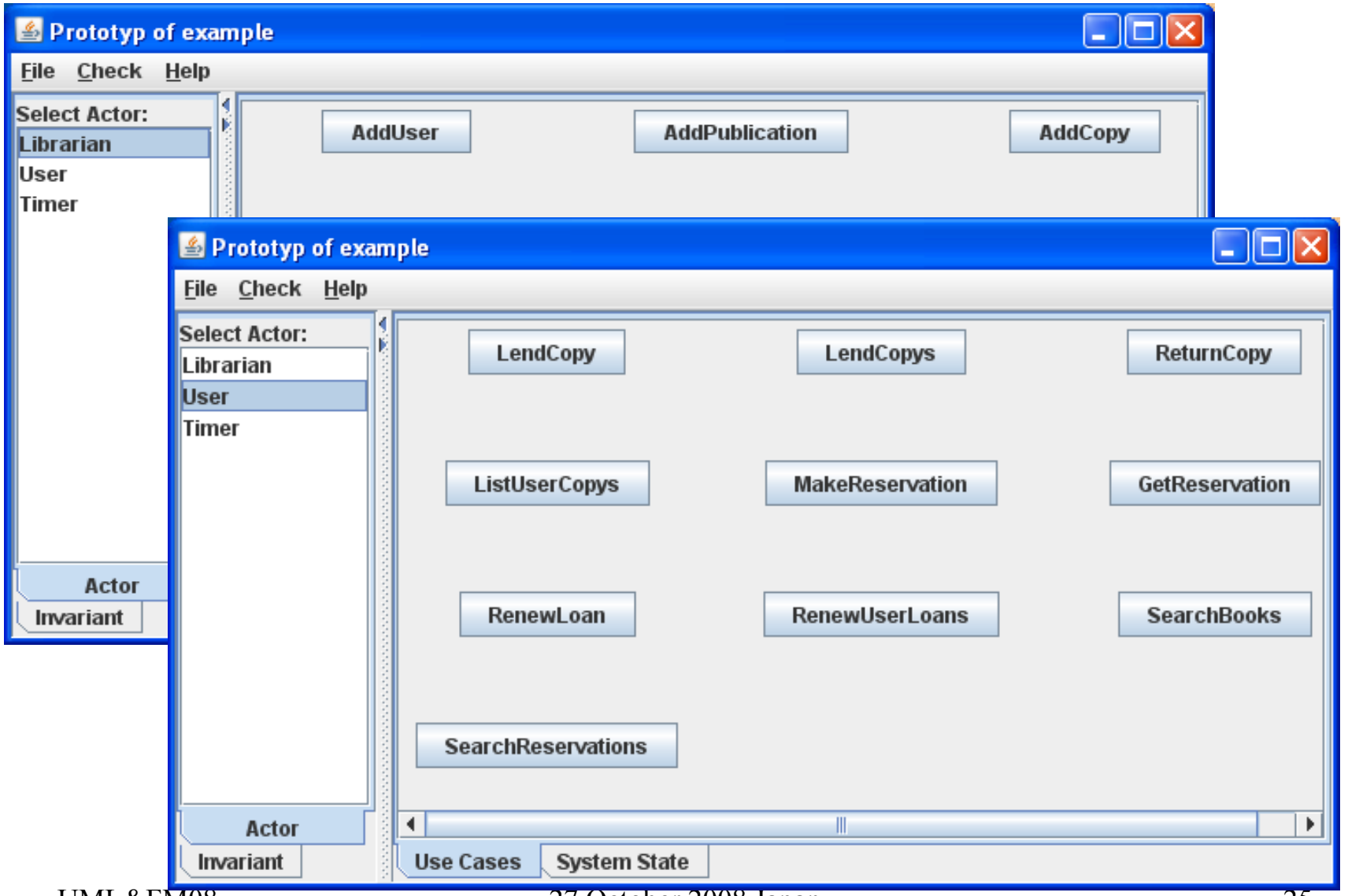
Code Generation of the Prototype



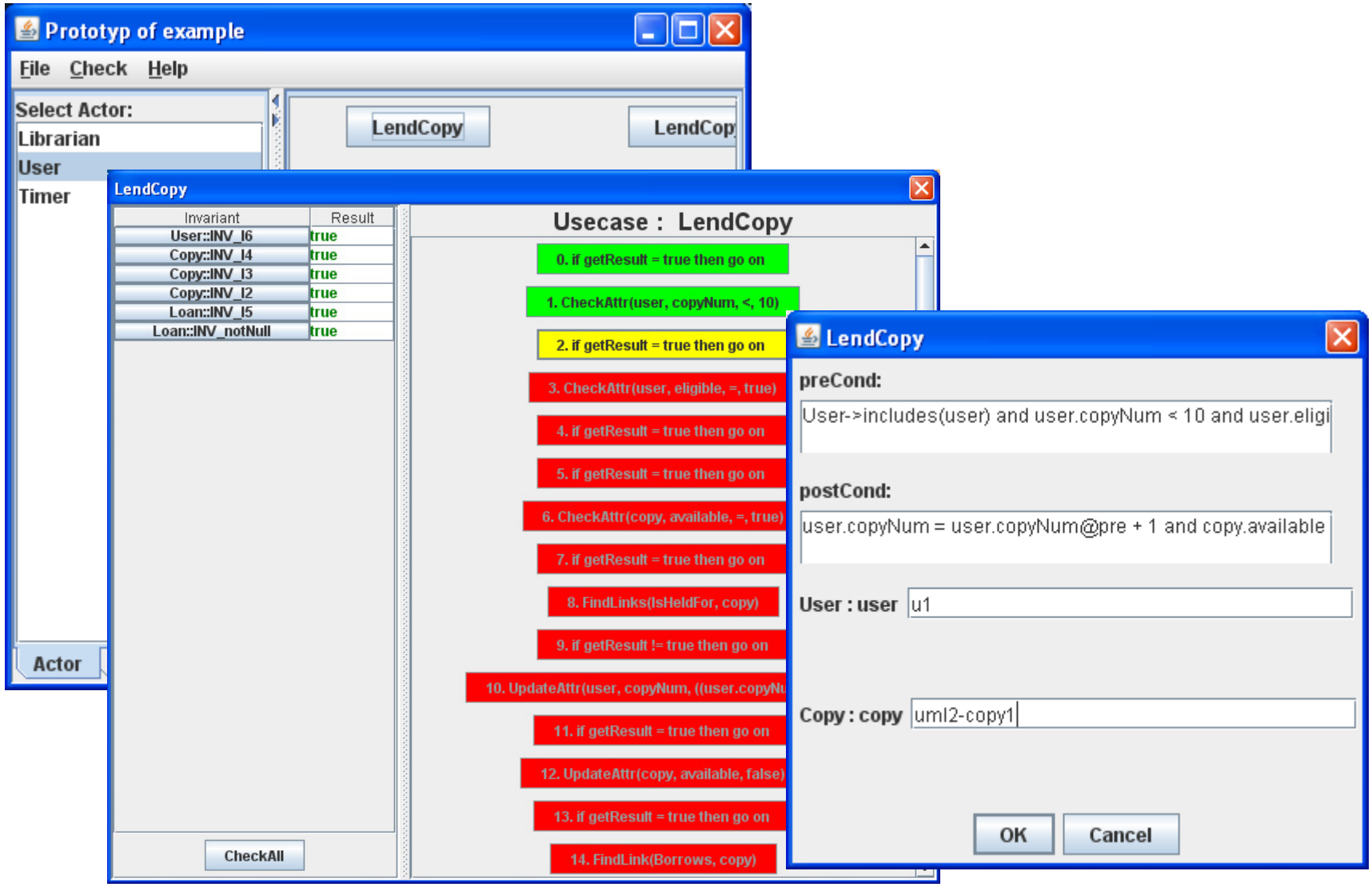
Compile and Run Prototype



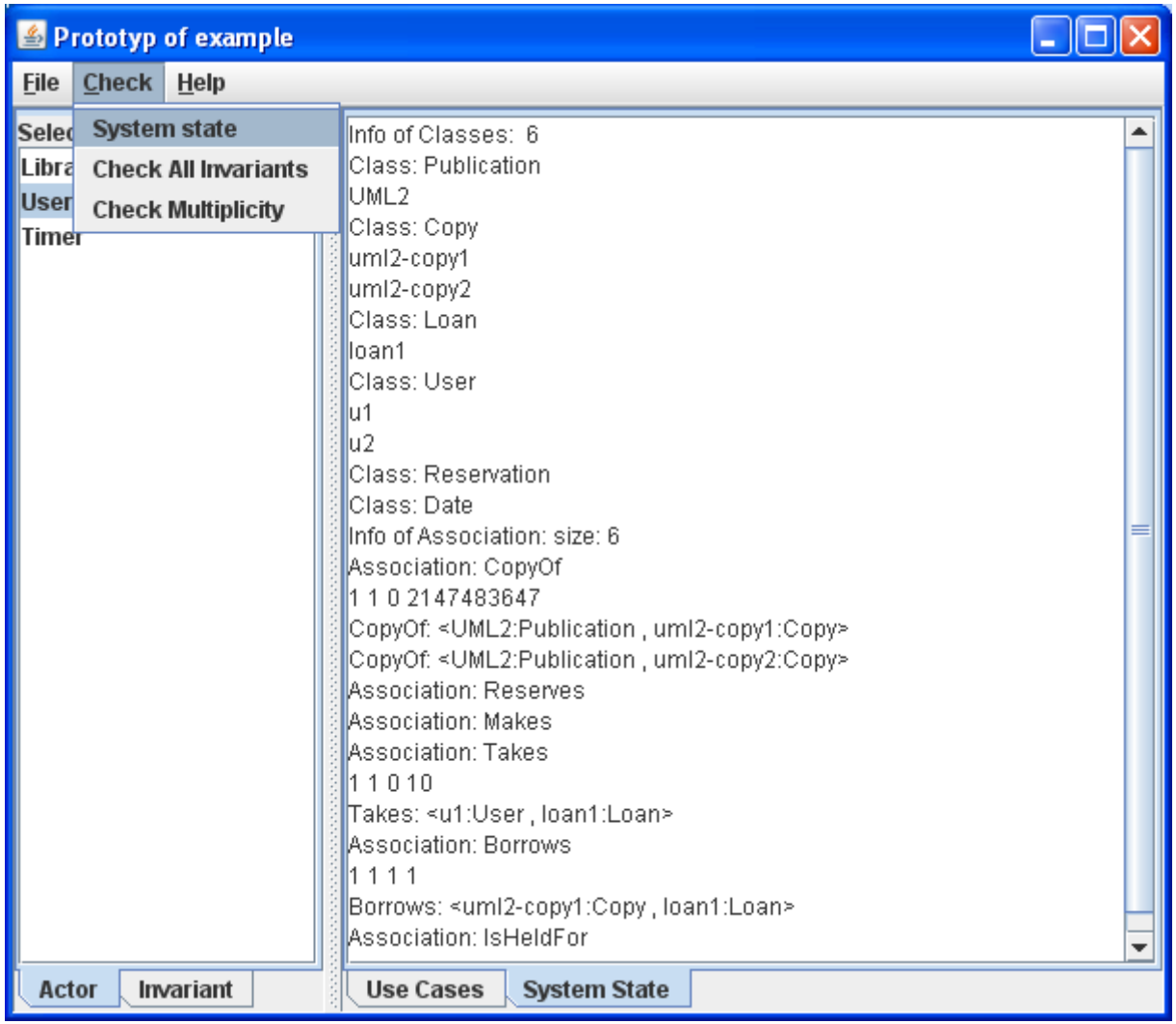
Generated Prototype Interface of Library



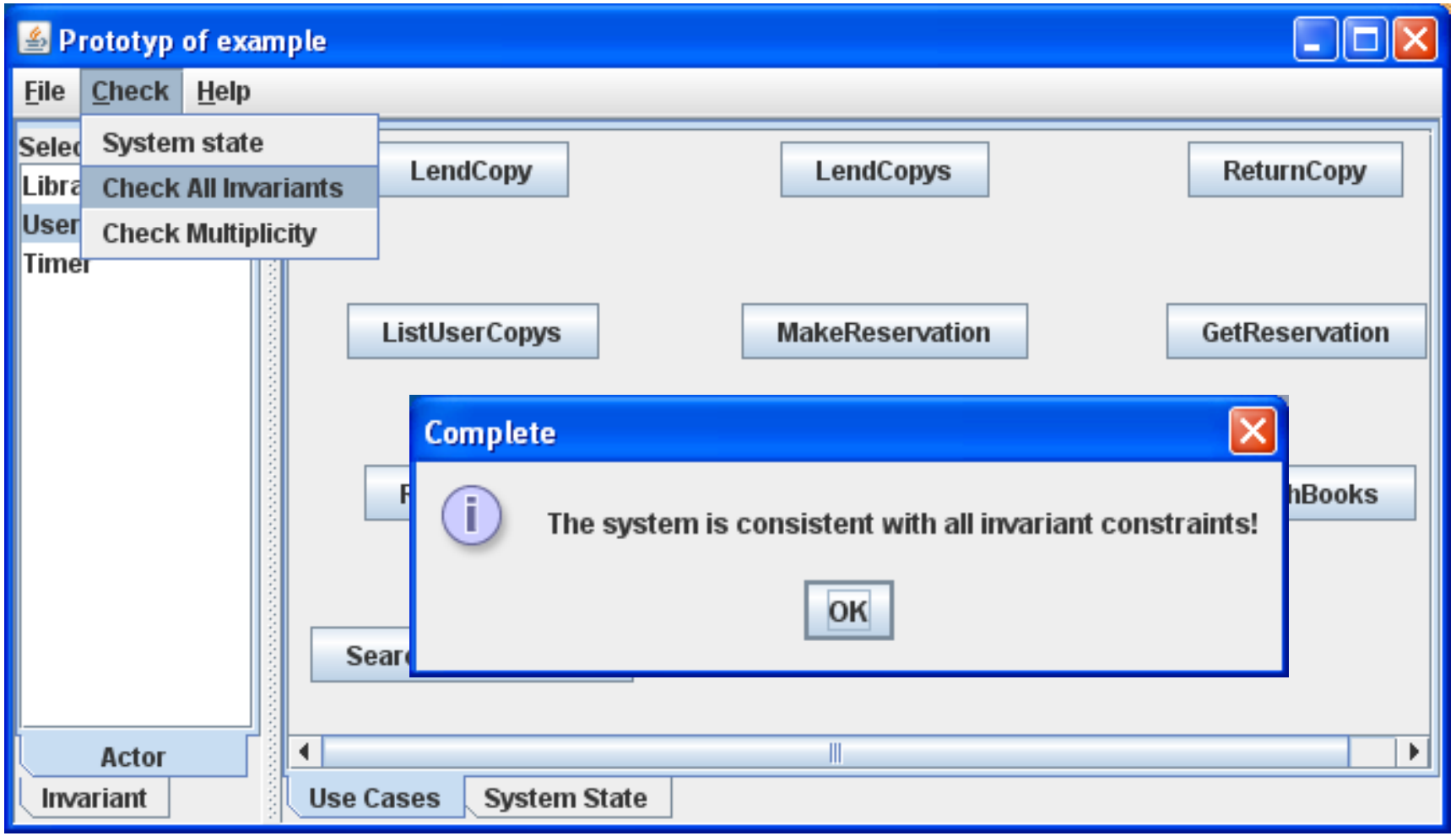
Prototyping Use Case *LendCopy*



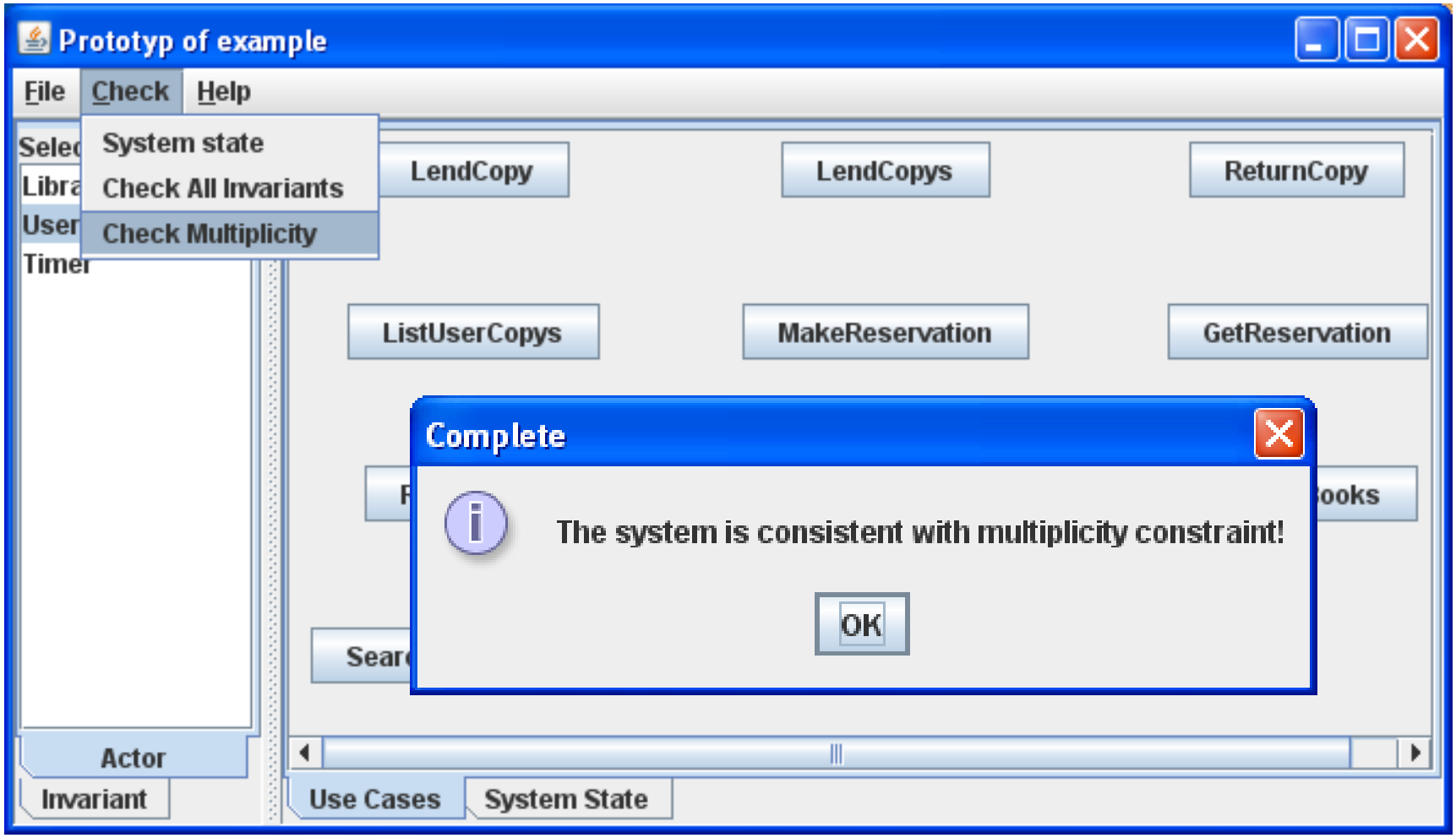
System State



Check Invariants on System State



Check Multiplicity on System State



Executable Problem of Post Conditions

• A complex equation satisfaction in post conditions may not be executable in the implicit style such as

$$F(x,y, x',y')=0 \wedge G(x,y,x',y')=0$$

• They will be left for further refinement into the explicit style such as

$$x'=f(x,y) \wedge y'=g(x,y)$$

Conclusion

1. The key idea is to map a use case defined on the CCM in pre and post conditions to a set of executable primitive actions.
2. We developed the prototype generation and analysis: ***AutoPA3.0***, which can accept the requirements model produced from UML CASE tool with OCL formal specification.
3. Requirements can be validated by the generated prototype.
4. System invariants and multiplicities can be checked dynamically on system state.

Future Work

We hope

• to apply the tool to more case studies and practical applications.

• to extend the functionality of tool to support test case generation and automated testing like the **USE** tool.

• to generate Java code from design sequence diagrams for non-executable parts in requirements.

• to put online for the public.

Thank you!

Question?