

# StructureBuilder Tendril Software Inc.

OOPSLA '98 Demo

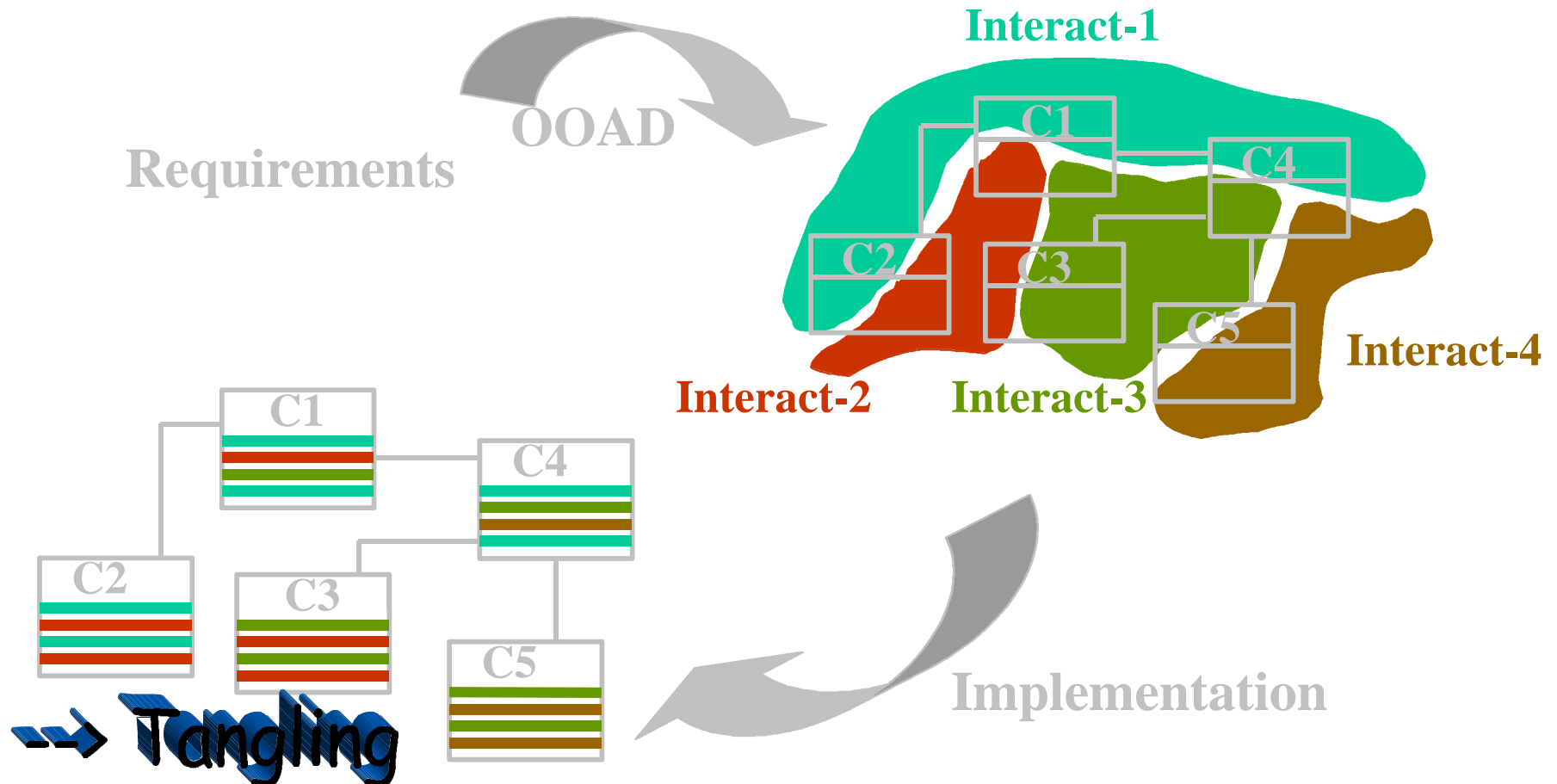
**Neeraj Sangal**, President Tendril Software

**Karl Lieberherr**, Northeastern University (presenter)

# Overview

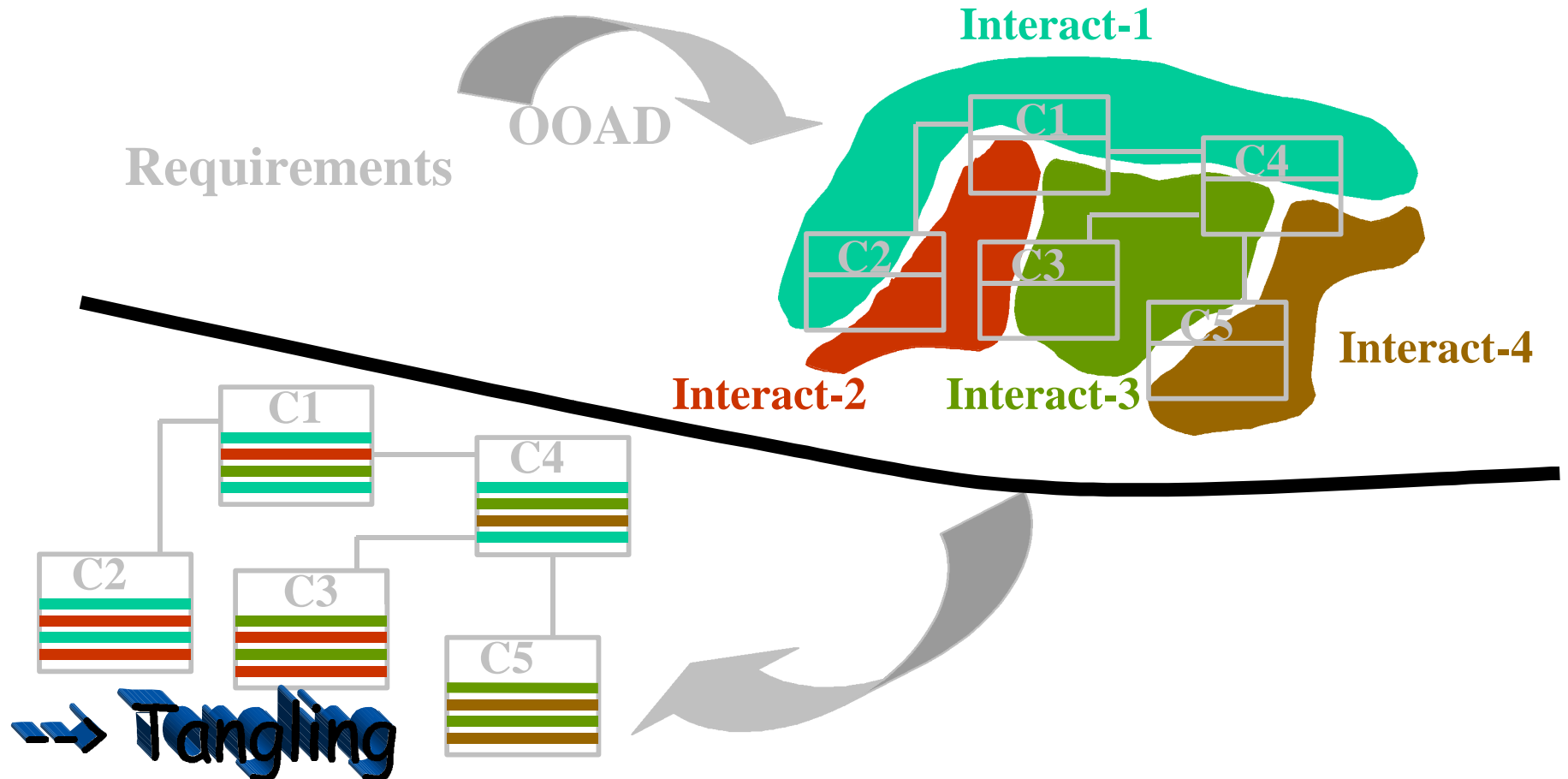
- Problems Addressed
- The Concept: Interaction Graphs
- The Tool: Structure Builder (SB)
  - Generation of executable code from
    - UML class diagrams
    - Generalized UML sequence diagrams (interaction graphs)
  - Other capabilities

# Motivation



Structure Builder  
let's you operate at  
interaction level

# Motivation



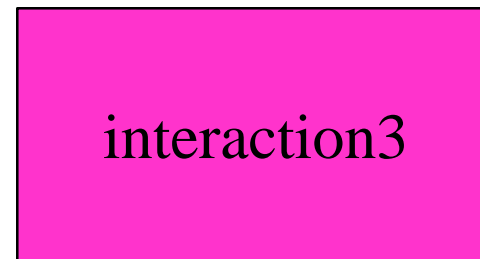
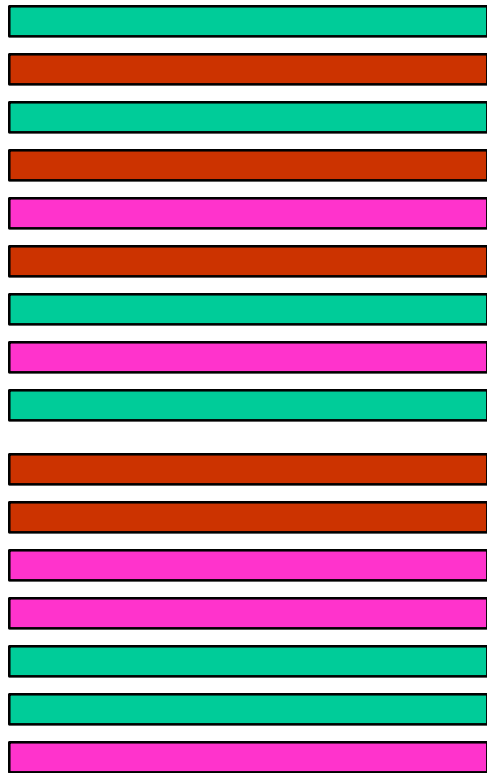
# Four problems addressed

- **TANGLING**
  - Cross-cutting of interactions
- **UPDATE**
  - Difficulty of maintaining UML interaction diagrams
- **TRANSPORTATION**
  - Object transportation code is difficult to maintain
- **CONTEXT DEPENDENCY**
  - Context changes are tedious to make

TANGLING

# Problem one addressed

ordinary program



# Problem one addressed

- Consequences
  - No need to distribute interactions manually over several classes
    - Work at level of interactions
  - No need to identify interactions from tangled code during maintenance
    - The interactions are explicit

## TANGLING

# Problem one addressed

- Related to *Aspect-Oriented Programming* (AOP)
  - Xerox PARC, Demeter Research Group, OOPSLA '92
  - AOP
    - Solves complex tangling problems
  - StructureBuilder
    - Solves two specific tangling problems: Interaction tangling and transportation tangling

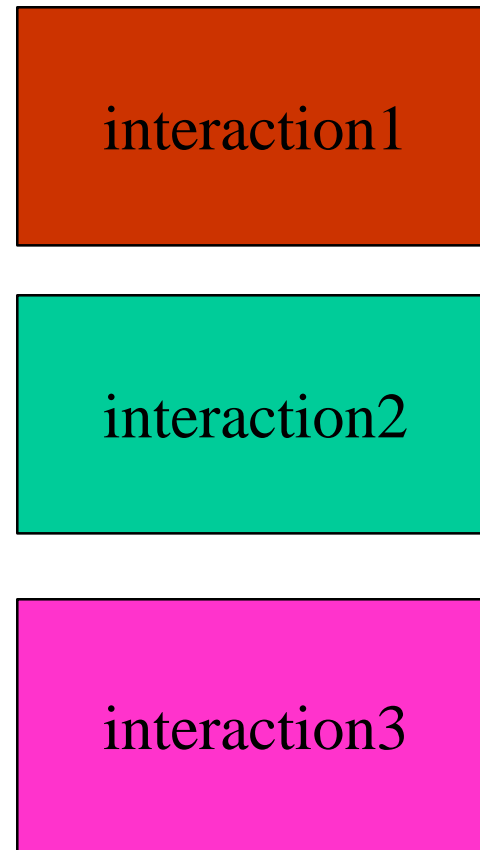
TANGLING

# Interaction Tangling

ordinary program

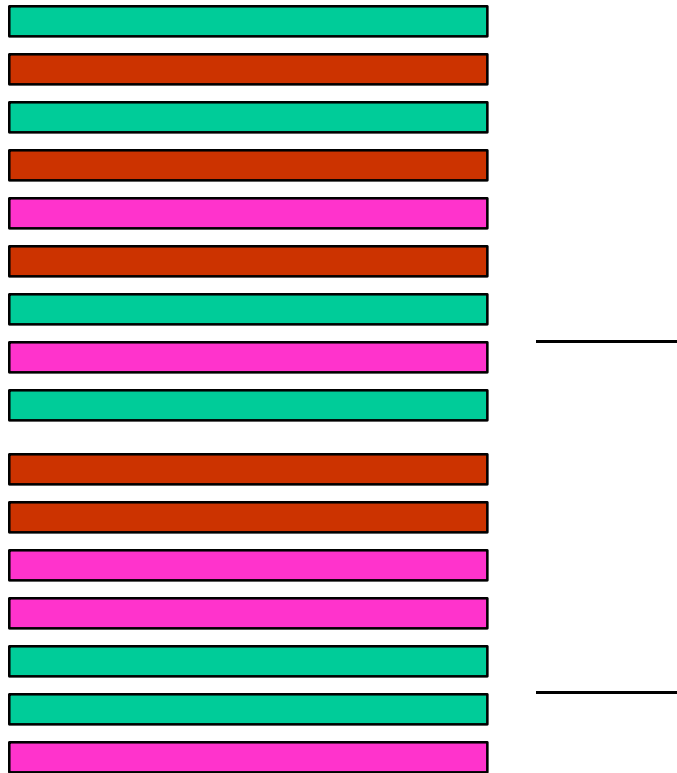


SB program

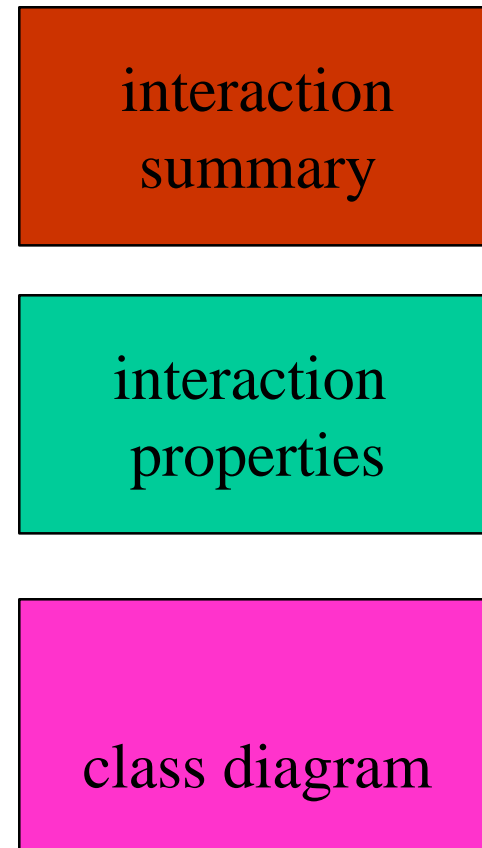


# TANGLING Transportation Tangling

ordinary program



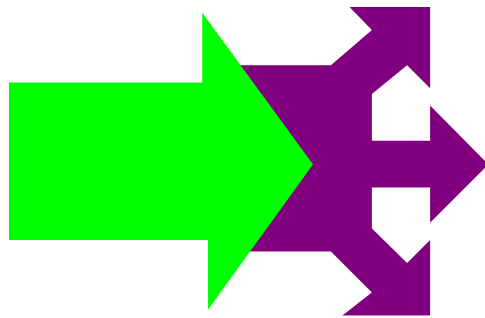
SB program



UPDATE

## Problem two addressed

- UML interaction diagrams are difficult to keep up-to-date with the code.

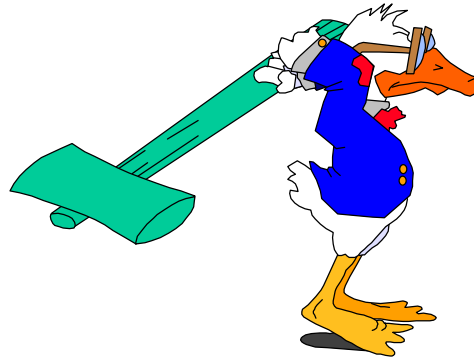


UPDATE

## *Problem two addressed*

So, what?

Forget about interaction diagrams?



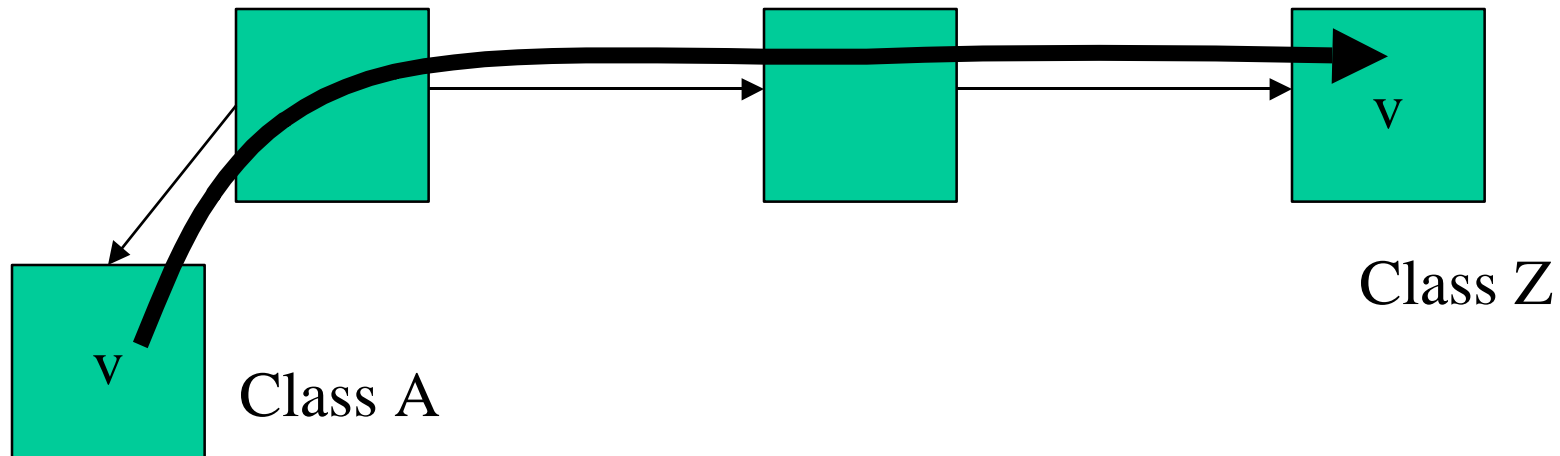
INTERACTION  
DIAGRAMS

No. The point is that they are incomplete.  
Let's make them into into complete specifications.

# TRANSPORTATION

## Problem three addressed

- Code for object interactions includes much low-level object transportation code

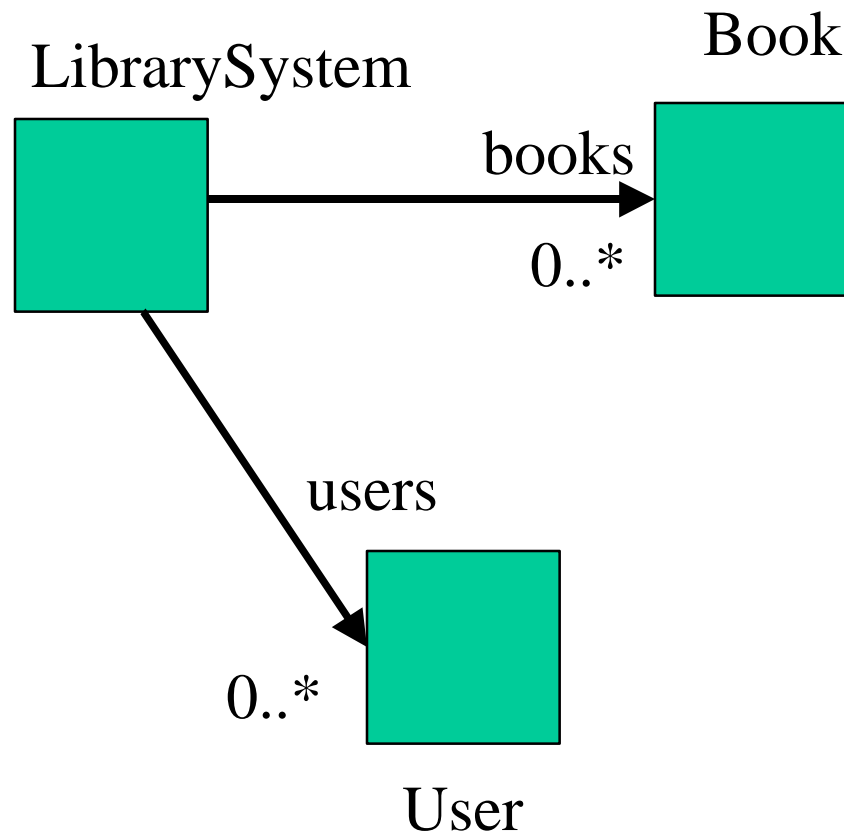


## Problem four addressed

- Code for object interactions includes much context-dependent information that makes the code hard to maintain

## CONTEXT DEPENDENCY

# UML Class Diagram



HashTable or Vector:  
Find operation looks very  
different at code level.

# Solution

- Interaction Graph language
  - Extend interaction diagrams to make them a specification language for object interactions.
- Generate code and interaction diagrams
  - Untangle high-level actions from context information
  - Untangle object transportation glue code from other interaction code

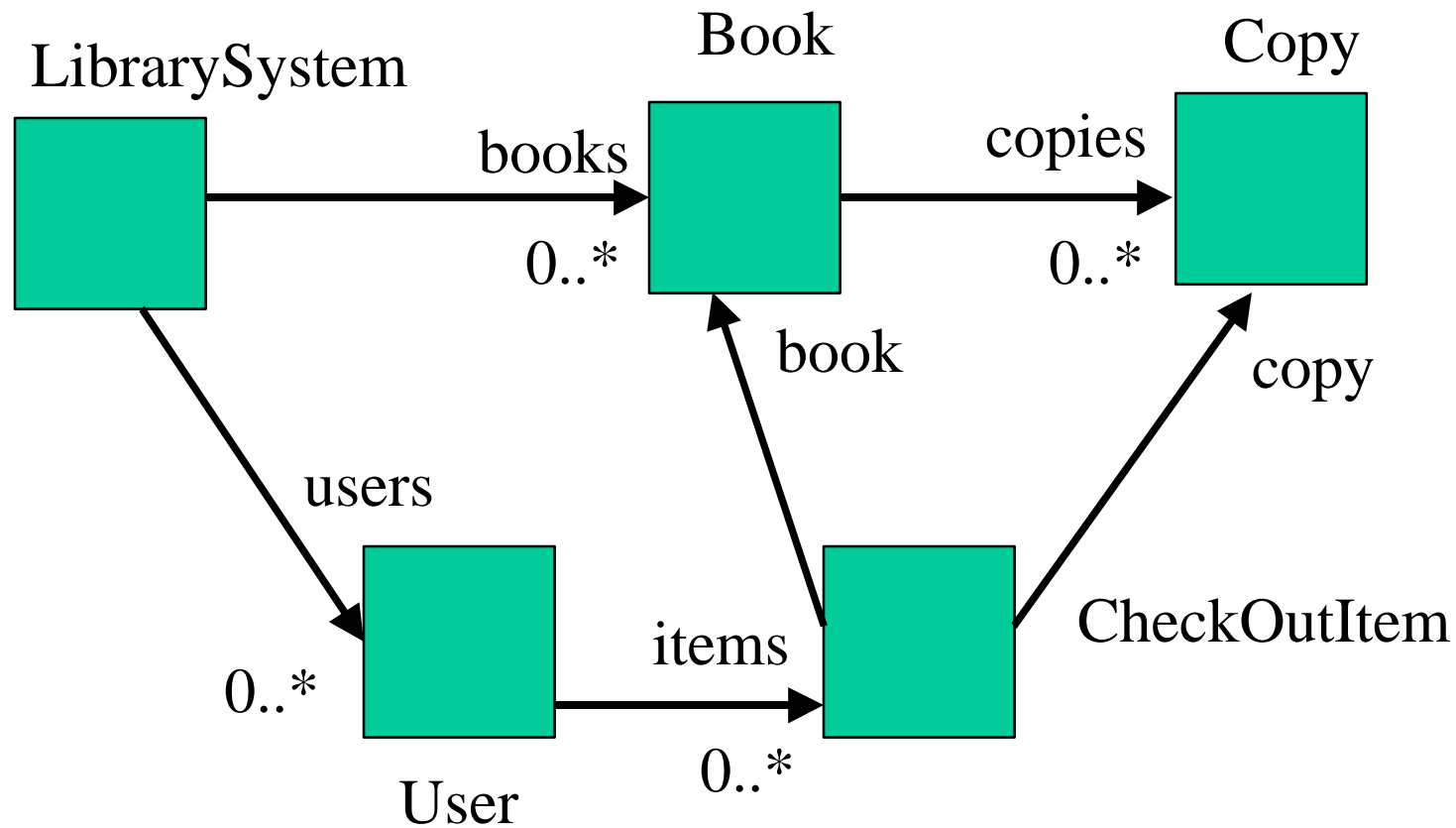
# Solution (continued)

- Programs look like designs
  - UML class diagrams
  - Interaction graphs are similar to UML sequence diagrams

# Interaction Graphs

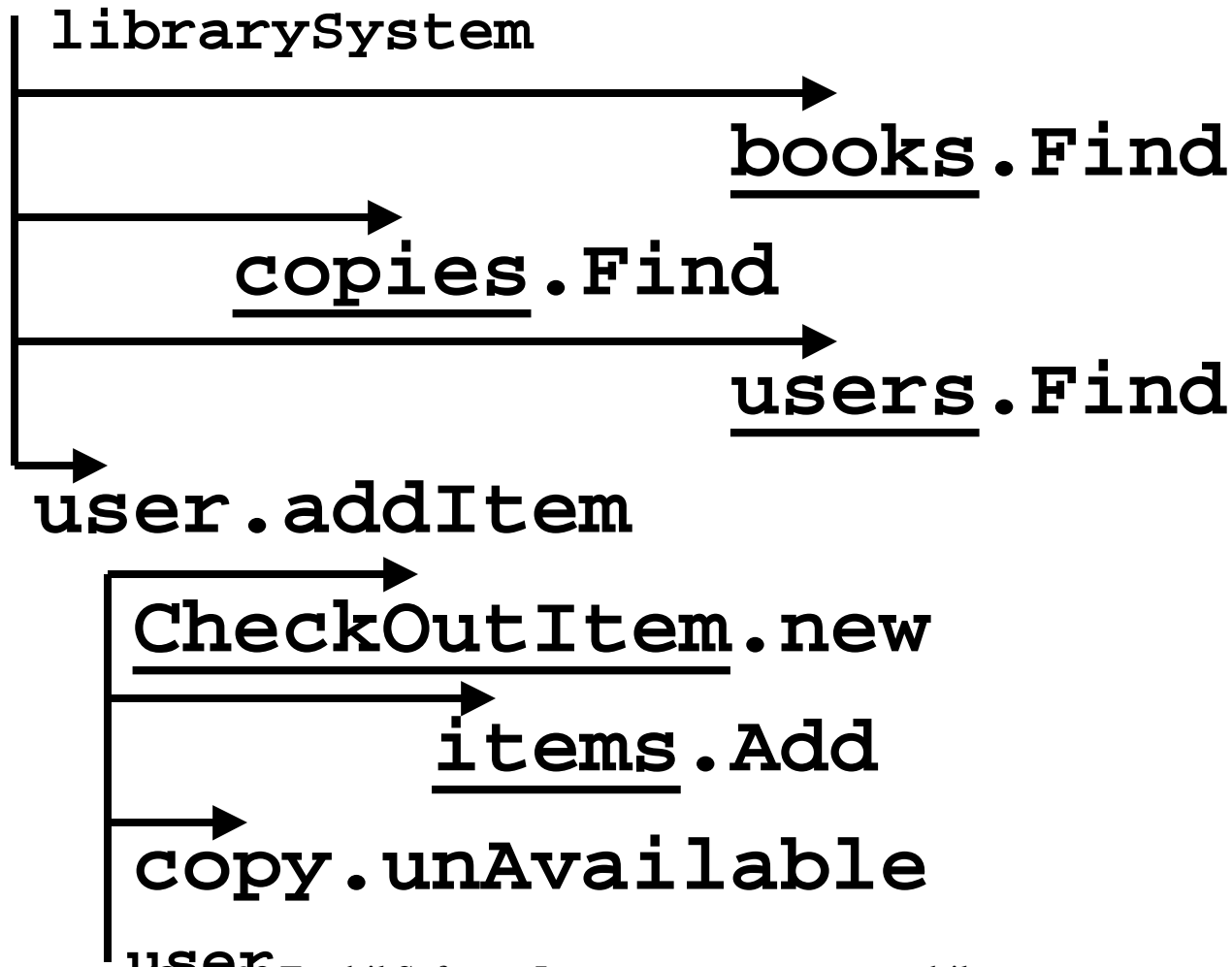
- Nodes
  - Participants/classes: labeled by an access path
- Edges
  - Message sends
- Local variables
  - Communication between participants
- Properties
  - Context of actions

# UML Class Diagram



CHECKOUT

# Sequence diagram





(C) 1998

> output  
< input

## Summary view

of an interaction graph

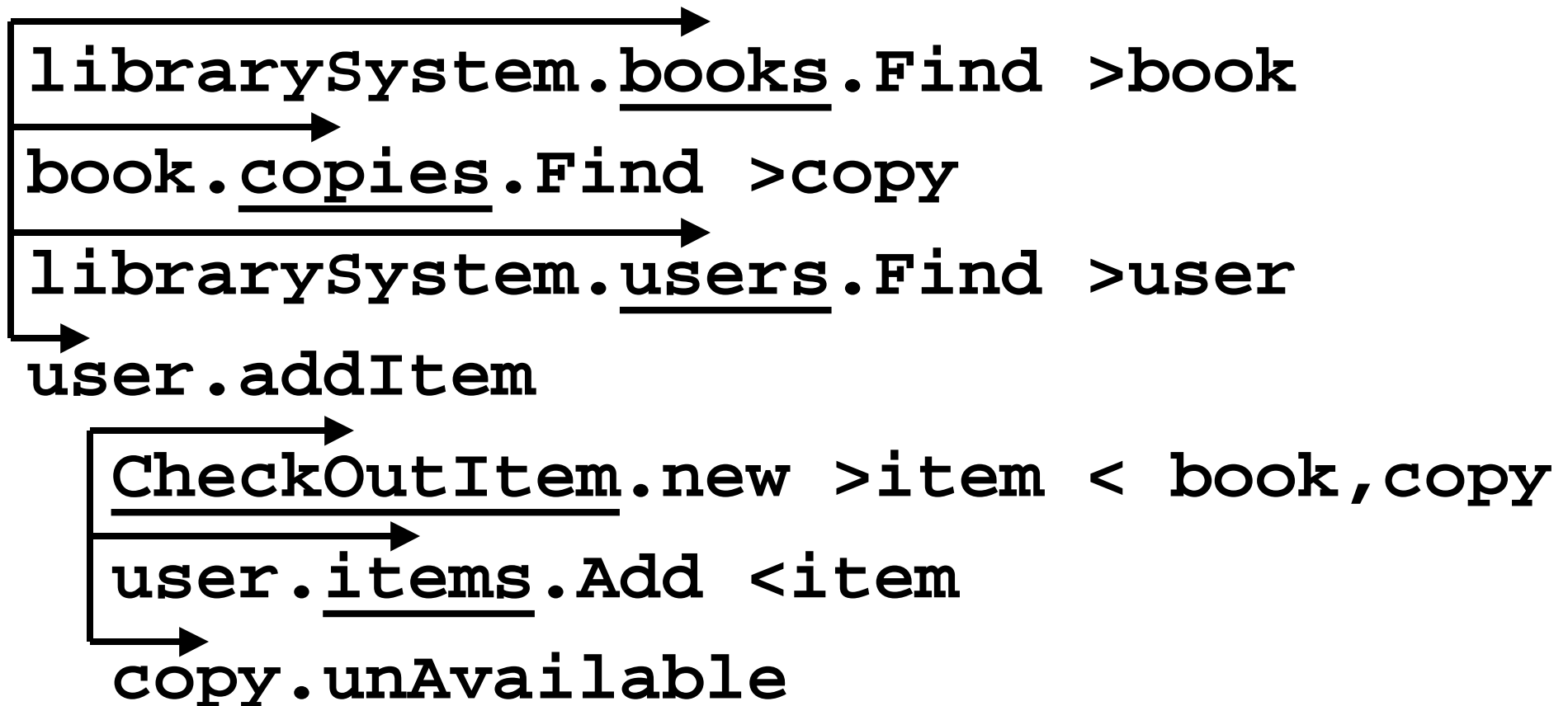
```
CHECKOUT(bn, uid)  
librarySystem.books.Find >book < bn  
book.copies.Find >copy  
librarySystem.users.Find >user < uid  
user.addItem  
  CheckOutItem.new >item < book, copy  
user.items.Add <item  
copy.unAvailable
```

> output

< input

# Where is the graph?

## It is a tree

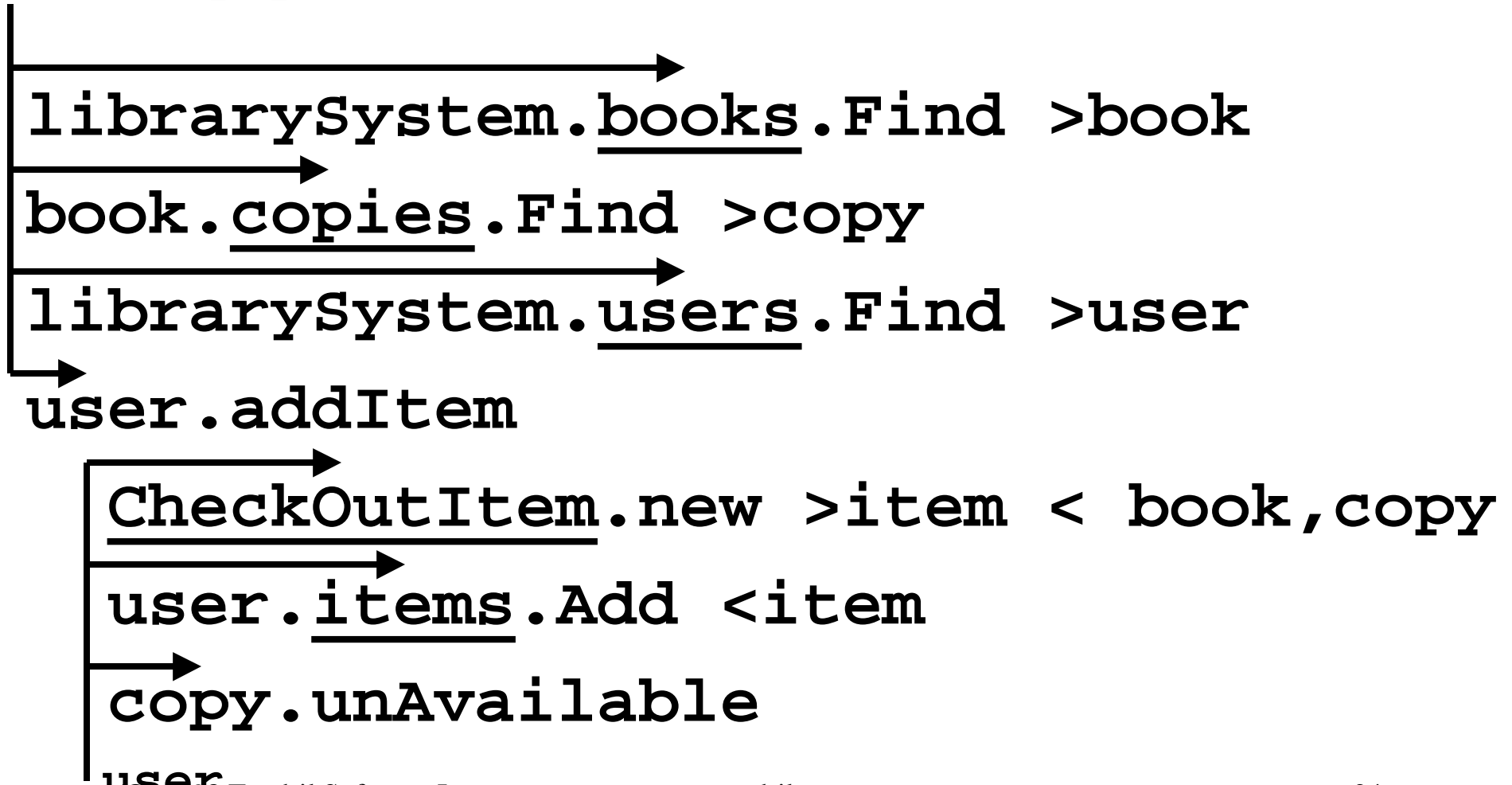


> output  
< input

# Where is the graph?

It is a tree

librarySystem



> output  
< input

## What are the properties?

```
CHECKOUT(bn, uid)
```

```
librarySystem.books.Find >book < bn
```

```
condition: bn == $curobj.bookName
```

```
return: book = $curobj
```

```
book.copies.Find >copy
```

```
condition: $curobj.isAvailable()
```

```
return: copy = $curobj
```

> output  
< input

## What are the properties?

```
CHECKOUT(bn, uId)
```

```
librarySystem.users.Find >user < uId
```

```
condition: uId == $curobj.userId
```

```
user = $curobj
```

```
user.addItem
```

```
CheckOutItem.new >item < book, copy
```

```
item=CheckOutItem(book, copy)
```

```
user.items.Add <item
```

```
copy.unAvailable
```

# Interaction Graph

- Defines properties of actions
- Provides complete specification
- But: is incomplete without a UML class diagram

# Differences Interaction Diagrams

## - Interaction Graphs

- Interaction Graphs
  - Computationally complete
  - Have properties containing code and context information
  - Each action keeps track of input/output

# StructureBuilder capabilities

- Full support for UML class diagrams
  - Reverse engineering of Java code
  - Immediate code generation
  - Code and diagrams are synchronized
  - User can add to generated code but it is not necessary

# StructureBuilder capabilities (continued)

- Support for Interaction Graphs
- Support for Sequence Diagrams
  - Generate sequence diagrams from interaction graphs and vice-versa
  - Object tracker: tells you which variables are currently available
  - Will be demonstrated for library checkout example

# Connection to Demeter

- StructureBuilder was influenced by Demeter/C++ and Demeter/Java
  - Interaction Graphs are related to propagation patterns: both address tangling of interactions
  - Object transportation is used in Demeter/C++

# Next: demo

- Summary so far:  
StructureBuilder introduces a generalization of sequence diagrams to solve four important problems related to OO software development.