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 lets you operate at interaction level

**Motivation**

OOAD

Requirements

Interact-1

Interact-2

Interact-3

Interact-4
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
Problem one addressed

ordinary program

SB program

interaction1

interaction2

interaction3
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
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
Interaction Tangling

ordinary program

SB program

interaction1

interaction2

interaction3
TANGLING  Transportation Tangling

ordinary program

SB program

interaction summary

interaction properties

class diagram
Problem two addressed

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

So, what?
Forget about interaction diagrams?

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

LibrarySystem

Book

books
0..*

User

users
0..*

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

LibrarySystem

Book

Copy

users

books

0..*

book

copies

0..*

copy

items

0..*

CheckOutItem

User
librarySystem

books.Find

copies.Find

users.Find

user.addItem

CheckOutItem.new

items.Add

copy.unAvailable

user
librarySystem : LibrarySystem
book : Book
user : User
item : CheckOutItem
copy : Copy

FindBook

FindCopy

FindUser

addItem()

ew

addItem

setAvailable(false)
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
Where is the graph?
It is a tree

librarySystem.books.Find >book

book.copies.Find >copy

librarySystem.users.Find >user

user.addItem

CheckOutItem.new >item < book,copy

user.items.Add <item

copy.unAvailable
Where is the graph?

It is a tree

> output
< input

librarySystem

<table>
<thead>
<tr>
<th>librarySystem.books.Find</th>
<th>&gt;book</th>
</tr>
</thead>
<tbody>
<tr>
<td>book.copies.Find</td>
<td>&gt;copy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>librarySystem.users.Find</th>
<th>&gt;user</th>
</tr>
</thead>
</table>

user.addItem

<table>
<thead>
<tr>
<th>CheckOutItem.new</th>
<th>&gt;item</th>
<th>&lt;book,copy</th>
</tr>
</thead>
</table>

| user.items.Add    | <item |

<table>
<thead>
<tr>
<th>copy.unAvailable</th>
<th></th>
</tr>
</thead>
</table>

user
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
What are the properties?

CHECKOUT($bn, $uId)

librarySystem.users.Find $user $uId

condition: $uId == $curobj.user_Id

$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.