Designing Components versus Objects: *A Transformational Approach*

David H. Lorenz
College of Computer Science
Northeastern University
Boston, MA 02115
`lorenz@ccs.neu.edu`

Joint Work With

John Vlissides
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598
`vlis@us.ibm.com`

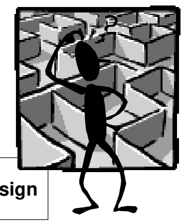
Northeastern University College of Computer Science — ICSE 2001 — IBM Research

Motivation: Components versus Objects

- **What is a "*component*"?**
  - ***Components are binary units of independent production, acquisition, and deployment that interact to form a functioning system.***

Szyperski, Component-Oriented Software, Beyond OOP. Addison-Wesley, 1997.

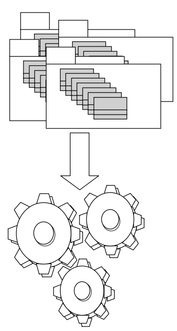
- **"Beyond OOP"**
  - **Decoupling software production from deployment**
  - **Markets for third-party components**
  - **Large scale third-party composition**



☞ **Unexplored issues in software design**

Test Case: Transforming OOD to CBD

- **Input: An OO Design**
- **Output: A CB Design**



Agenda

3 Steps in Component Based Design
■**Objective**: Underscore the importance of design in CBP

❶ **Identify the components and connectors**
- **What do components model in the problem space**
- **What do their communications model**

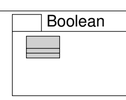
❷ **Characterizing the components and connectors**
- **What are the dimensions of components**
- **Fan-in, fan-out, event types**

❸ **Implementing the components and connectors**
- **How do you trade off space, time, and safety to optimize performance, reusability, modularity, and cost**

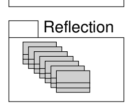
Progression of the Example

Case Study: Transforming OOD to CBD
■Increasingly complex examples

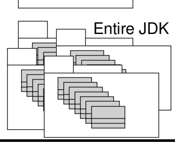
❶ **Identify the components and connectors**
- **Single class example**



❷ **Characterizing the components and connectors**
- **Multi-class**
- **Single package**



❸ **Implementing the components and connectors**
- **Multi-class**
- **Multi-package**



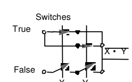
Example 1: Boolean Logic

- **Minimalist Approach: NAND**
  - **{ NAND } is a complete set.**
  - **NAND far outnumber AND, OR in hardware designs.**

Truth Table

X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

Switches

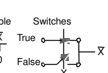


- **Adding only the most important gizmo: NOT**
  - **Cut down on wiring.**

Truth Table

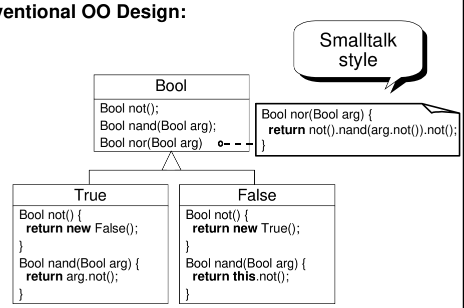
X	$\overline{X}$
0	1
1	0

Switches



① Component-Event Partitioning

- **Conventional OO Design:**

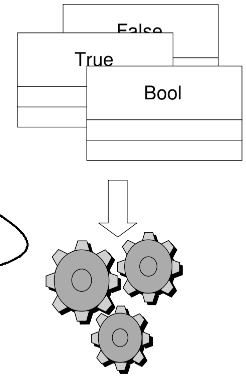


What should a CB version of this design look like?

NU

"Identify the Components & Connectors"

Transforming OOD to CBD:

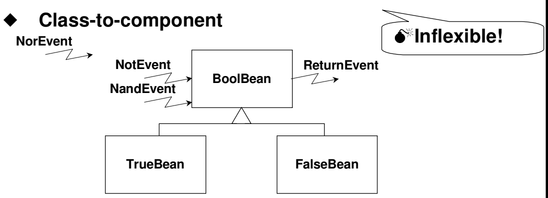


- **Input: An OO Design**
  - **Classes**
  - **Methods**
- **Output: A CB Design**
  - **Components**
  - **Events**

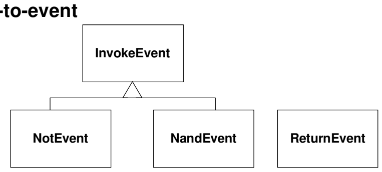
NU

A Naïve Component-Based Design

- **Class-to-component**



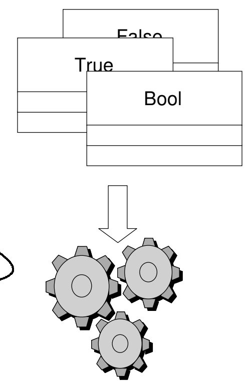
- **Method-to-event**



NU

"Identify the Components & Connectors"

Transforming OOD to CBD:

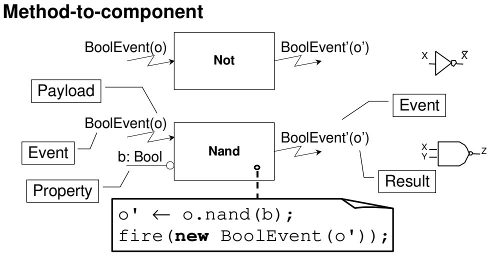


- **Input: An OO Design**
  - **Classes**
  - **Methods**
- **Output: A CB Design**
  - **Components**
  - **Events**

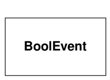
NU

An Improved Component-Based Design

- **Method-to-component**



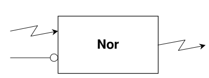
- **Class-to-event**



NU

Adding New Operations

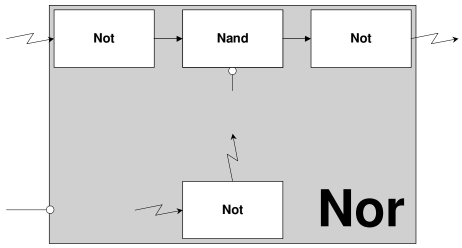
- **nor(x,y)**



NU

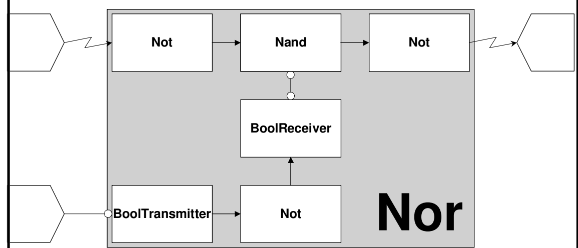
Building the New Operations

- nor(x,y) = not(nand(not(x),not(y)))



NU

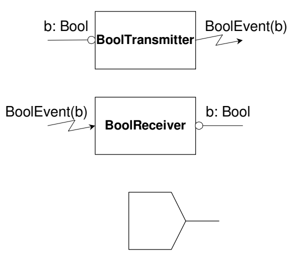
Event-Property Adapters



NU

Additional Required Components

No direct analog in OO design



NU

Steps in Component Based Design

① **Identify the components and connectors**
- **What do components model in the problem space**
- **What do their communications model**

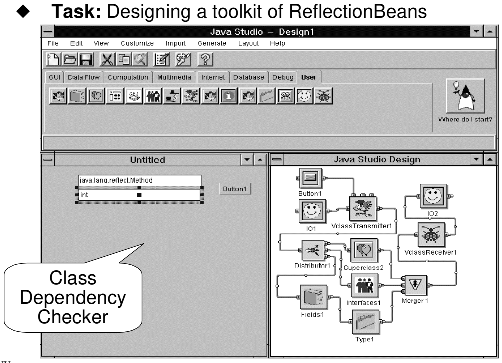
☞ **Characterizing components and connectors**
- **What are the dimensions of components**
- **Fan-in, fan-out, event types**

③ **Implementing the components and connectors**
- **How do you trade off space, time, and safety to optimize performance, reusability, modularity, and cost**

NU

Example 2: java.lang.reflect

- **Task:** Designing a toolkit of ReflectionBeans



NU

② Component Design

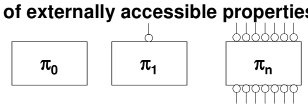
- **A component class exists for every method of every class in `java.lang.reflect`**
- **The resulting number of component types can become unwieldy**
  - **How to reuse abstract components and subcomponents?**
  - **Should a single component carry out several duties?**
  - **How do you structure a potentially large number of components in a way that is easy to navigate and maintain?**

☞ **Need a systematic approach**
- **Degree of freedom JavaBeans offers**
- **Taxonomy for JavaBeans**
- **Mapping methods to components**

NU

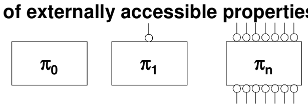
Dimension of JavaBeans

- Number of externally accessible properties

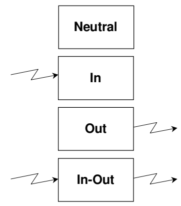


Dimension of JavaBeans

- Number of externally accessible properties

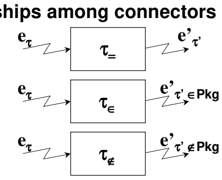


- Bias



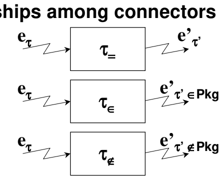
Dimension of JavaBeans (cont)

- Type relationships among connectors
  - Same type
  - Known type
  - Other type

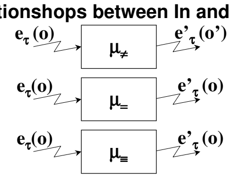


Dimension of JavaBeans (cont)

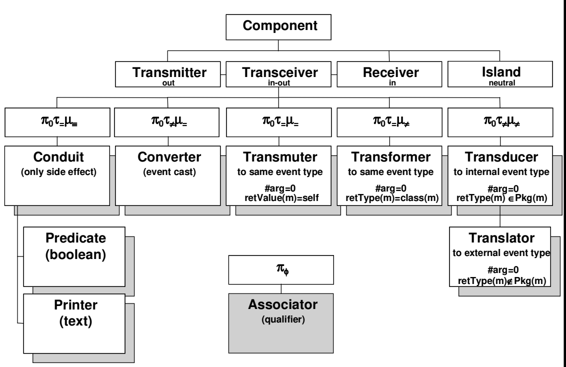
- Type relationships among connectors
  - Same type
  - Known type
  - Other type



- Payload relationshops between In and Out events
  - Other ID
  - Same ID
  - Immutable



Taxonomy of Implicit Invocation



The Class "Class"

```
public final class Class implements Serializable{
 private Class() {}
 public static Class forName(String className)
  throws ClassNotFoundException {…}
  public Object newInstance() throws
   InstantiationException,IllegalAccessException{…}
  public native Class getSuperclass();
  public native Class[] getInterfaces();
  public Field[] getFields() throws
   SecurityException {…}
  public Method[] getMethods() throws
   SecurityException {…}
…
}
```

ClassEvent-Transmiter

ClassEvent-to-ObjectEvent Transducer

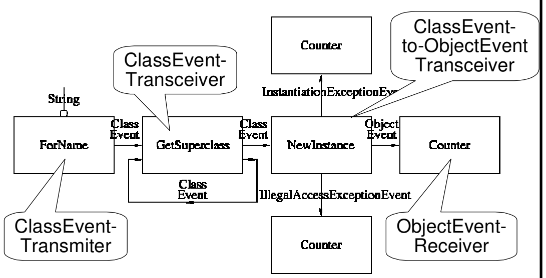
ClassEvent-Transformer

ClassEvent-Transformer

ClassEvent-to-FieldEvent Transducer

ClassEvent-to-MethodEvent Transducer

Reflection components in a Builder



Steps in Component Based Design

① **Identify the components and connectors**
- **What do components model in the problem space**
- **What do their communications model**

② **Characterizing components and connectors**
- **What are the dimensions of components**
- **Fan-in, fan-out, event types**

☞ **Implementing the components and connectors**
- **How do you trade off space, time, and safety to optimize performance, reusability, modularity, and cost**

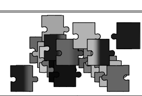
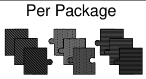
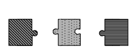
③ Component Implementation

- **Task:** Expand the scale of application, “JDK Beans”
- **Design challenges**
  - **Method signature**
  - **Component category**
  - **Class and object granularity**
  - **Static safety**
  - **Runtime overhead**

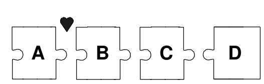
Design Points in the Design Space

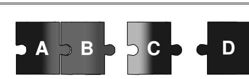
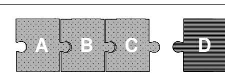
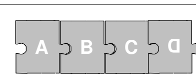
- **Design possibilities**
  - **Hard-wired**
    - “Bool example” approach
    - A custom, hard-wired component for every method signature
  - **ReflectionBeans approach**
    - per package:
      - **A set of custom, hard-wired components per package for every taxonomic category of component**
    - across packages:
      - **A single, parameterized component per category**
  - **Highly parameterized**
    - A single, highly parameterized component covering all methods in all packages

Class Granularity

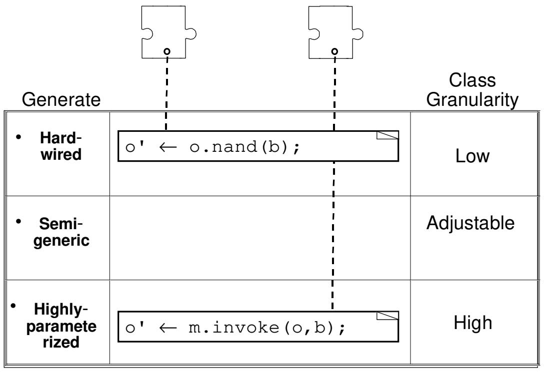
• **Hard-wired**			Finest
• **Semi-generic**	Per Package 	Across Packages 	Adjustable
• **Highly-parameterized**			Coarsest

Static Safety



• **Hard-wired**		Full
• **Semi-generic**		Full for structure; None for data
• **Highly-parameterized**		None

Runtime Overhead

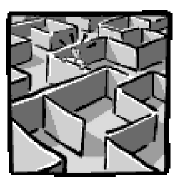


The Design Space for Components

	Hard-wired (point a)	Semi-genetic (point b)	Highly parameterized (point c)
Class Granularity	**Finest**	**Adjustable**	**Coarsest**
Static safety	**Full**	**Full for structure; Adjustable for data**	**None**
Object granularity	**Fine**	**Fine**	**Fine**
Runtime overhead	**Low**	**Adjustable**	**High**

Summary

- **Heighten awareness to CBD as something beyond OOD**
  - **How components expand the design space**
    - A good OOD ≠ A good CBD
  - **Importance of design in CBP**
    - Addressing new design issues ∉ OOD
- **Propose a new taxonomy of implicit invocation**
  - **Mapping of methods-to-components**
- **Illustrate concretely with JavaBeans examples**



Related / Future Work

- **Related Work**
  - **Architectural Styles**
    - Shaw and Garlan
  - **Taxonomy of Connectors**
    - Mehta, Medvidovic, and Phadke
- **Future Work**
  - **Taxonomies**
    - Taxonomy of wrappers.
    - Other component technologies.
    - Documenting design decisions in terms of the proposed taxonomies; Promote collection and dissemination of OOD and CBD expertise alike.
  - **Generator**
    - A component generator with knobs that let a user choose among design decisions
    - Akin to similar work for code generation from design patterns.
  - **ADL**
    - Architectural descriptions for actual development practices.

Conclusion

- **OO Design vs. Component Design**
  - **Much has been written about OOD, with few practical advances in automating the process.**
  - **In CBP, a high degree of automation exists, but little attention is paid to CBD**
- **OOD**
  - **OOL adhere closely enough to a common object model**
  - **A designer can reason about an OO design largely independent of implementation language**
- **CBD**
  - **Commonalities among component architectures are less apparent and agreed upon**
  - **Underlying technology must be taken into greater consideration than in OOD**

The End

More Information:

- **Technical Report NU-CCS-00-01; IBM-RC 21773=98045**
- **URL:** http://www.ccs.neu.edu/home/lorenz/center/
- **Please send comments to:** java-beans@ccs.neu.edu