Tutorial on BoxScript: A Component-Oriented Language

Yi Liu
Department of Computer Science
Outline

- What are Components?
- How to Componentize a System?
- What is BoxScript?
- How to use BoxScript?
Outline

- What are Components?
- How to Componentize a System?
- What is BoxScript?
- How to use BoxScript?
Software Components
A Simple Example

Compositionality
Flexibility
Components – A Closer Look

Interface

Inner component

Component 1

Component 2
## Components vs. Objects

<table>
<thead>
<tr>
<th></th>
<th>Component</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Strong Encapsulation</strong></td>
<td>Yes</td>
<td>Some Yes Some No</td>
</tr>
<tr>
<td><strong>Compositionality</strong></td>
<td>Yes</td>
<td>Need extra programming</td>
</tr>
<tr>
<td><strong>Flexibility</strong></td>
<td>Yes</td>
<td>Need extra programming</td>
</tr>
</tbody>
</table>
Outline

- What are Components?
- How to Componentize a System?
- What is BoxScript?
- How to use BoxScript?
Goals of Component Design

- **Components should be**
  - **cohesive:** all functionality fits together for coherent, easily understandable purpose
  - **independent:** components are decoupled from each other
  - **changeable:** implementation of one component can be changed without affecting others

- **Component system should be robust with respect to change**
  - Likely changes should affect only a few components
  - Unlikely changes might affect overall structure
Design Guidelines

Decomposition:
- If some design issue is likely to change, hide it inside one component – **information hiding**
- Define interfaces of components to be stable in the face of likely changes – **abstraction**

Specification:
- Precisely define everything one component may assume about another
- One component should assume no more than necessary about others
Examples

- Example 1

```java
public interface Price {
    double getPrice(int client, int item, int quantity);
}
```
Examples

- Example2
Outline

- What are Components?
- How to Componentize a System?
- What is BoxScript?
- How to use BoxScript?
BoxScript

- A language for component-oriented programming
Main Purposes of BoxScript

- Introducing component concepts
- Providing simple environment to user
- Supporting main properties of COP
  - Compositionality
  - Flexibility
Outline

- What are Components?
- How to Componentize a System?
- What is BoxScript?
- How to use BoxScript?
Key Concepts

- **Built on top of Java**
- **Component**
  - **Box**
  - **Blackbox entity**
    - No externally visible state
    - Only interfaces exposed
Key Concepts
Interfaces

- **Interface**
  - Java Interface

  ```java
  Price.java
  public interface Price
  {
    double getPrice(int client, int item, int quantity);
  }
  ```

- **Provided interface**
  - Describes operations that a box implements and that other boxes may use

- **Required interface**
  - Describes operations that the box uses and that must be implemented by another box
Key Concepts

Boxes

- General characteristics of boxes
  - Contains the descriptions of provided interfaces and required interfaces (.box)
  - 1..n provided interfaces
  - 0..m required interfaces

- Types of boxes
  - Abstract box
  - Concrete box
    - Atomic box
    - Compound box
Key Concepts
Abstract Box

- Abstract box
  - No implementations of the provided interfaces
  - Should be implemented by concrete boxes
Key Concepts
Abstract Box Example

\textit{PricingAbs.box}
abstract box PricingAbs
\{ provided interfaces Price Pr;
  required interfaces Discount Dc;
\}

Diagram:
- PricingAbs
- Interface type
- Interface Handle
- price Pr
- Discount Dc

21
Key Concepts

Boxes

- Abstract box
- Concrete box
  - Atomic box
  - Compound box
Key Concepts
Atomic Box

- **Atomic box**
  - Does not contain any other boxes
  - Supplies implementations of the provided interfaces

*Pricing.box*

```plaintext
box Pricing implements PricingAbs
{
  provided interfaces Price Pr;
  required interfaces Discount Dc;
}
```

Diagram:
- Pricing
  - Interface type
  - Interface handle
- Price Pr
- Discount Dc
**Key Concepts**

**Interface Implementation of Atomic Box**

**PrImp.java**

Default name for interface implementation:

*Interface handle name + Imp*

```java
public class PrImp implements Price {
    private BoxTop _box;
    Discount dc; // required interface
    public PrImp(BoxTop myBox) {
        _box = myBox;
        InterfaceName name = new InterfaceName("Dc");
        dc = (Discount)_box.getRequiredItf(name);
    }
    public double getPrice(int client, int item, int quantity) {
        double disc = dc.getDiscount(client, item, quantity);
        return PriceList.p[item] * (1 - disc* 0.01) * quantity;
    }
}
```

**Diagram:**

- `Pr` (Pricing)
- `Dc` (Discount)
- `price`

`PrImp.java` class implements the `Price` interface and uses a `BoxTop` to get a required interface named `Dc` for pricing.
**Key Concepts**

**Compound Box**

- Composed from atomic boxes or other compound boxes
- Follow composition rules
  - By default, all provided interfaces are hidden unless explicitly exposed
  - Expose a required interface of a constituent if not wired to a provided interface of another
Key Concepts
Composition Strategy

Compose to

Box1
P11 P12
R11 R12 R13

Box2
P21 P22
R21 R22

Box1_2
P11
P12
R11 R12 R13

P21 P22
R21 R22

Box1
R11 R12
Box2
R22
Key Concepts

Compound Box Example

- Two boxes

  **[PricingAbs.box]**

  abstract box PricingAbs
  {  provided interfaces Price Pr;
      required interfaces Discount Dc;
  }

  **[DiscountingAbs.box]**

  abstract box DiscountingAbs
  {  provided interfaces Discount Dis;
  }
Compose two boxes

box `CalPrice` implements `CalPriceAbs`

{ composed from `PricingAbs boxP`, `DiscountingAbs boxD`;
  provided interfaces `Price tPrice` from `boxP.Pr`;
  connect `boxP.Dc` to `boxD.Dis`;
}

Diagram:

- Box `CalPrice`
- Box `PricingAbs boxP`
- Box `DiscountingAbs boxD`
- Price `Pr`  <->  `tPrice`
- Discount `Dc`  <->  `Dis`
Box Elements

- **For all boxes**
  - Interfaces (.java) *by user*
  - Box description file (.box) *by user*

- **For atomic boxes only**
  - Interface implementation (.java) *by user*

- **For concrete boxes only**
  - Configuration information (.conf) *by user*
  - Box manager code (.java) *by compiler*
If abstract boxes participate in composition

box CalPrice implements CalPriceAbs
{   composed from PricingAbs boxP,
    DiscountingAbs boxD;
    provided interfaces Price tPrice from boxP.Pr;
    connect boxP.Dc to boxD.Dis;
}

Box Handle       Directory for PricingAbs          Directory for Pricing
[CalPrice.conf]  (boxP, "D:\warehouse_root\boxes\PricingAbs\", "Pricing\Pricing");
(boxD, "D:\warehouse_root\boxes\DiscountingAbs\", "Discounting\Discounting");
Box Elements
Configuration Information

- If concrete boxes participate in composition

```java
box CalPrice implements CalPriceAbs
{   composed from Pricing boxP,
    Discounting boxD;
    provided interfaces Price tPrice from boxP.Pr;
    connect boxP.Dc to boxD.Dis;
}
```

[CalPrice.conf]

```java
(boxP, "D:\warehouse_root\boxes\PricingAbs\Pricing\Pricing");
(boxD,
"D:\warehouse_root\boxes\DiscountingAbs\Discounting\Discounting");
```
Box Elements
Configuration Information

- Configuration for interface implementation when implementation file name is not default

[Pricing.box]
box Pricing implements PricingAbs
{
  provided interfaces Price Pr;
  required interfaces Discount Dc;
}

Box Pricing has Priceimp.java implementing interface type Price.

[Pricing.conf]
(Pr, “Priceimp”);
Box Elements

Box Manager Code

- **Generated by compiler**
  - To initiate box instances
  - To assign references to interface handles
Box Run-time Structure

<<interface >> Price
<<interface >> Discount
<<Java class >> PrImp
<<Java class >> DisImp
<<Java class >> Pricing
constructor Pricing
instantiates
<<Java class >> Discounting
constructor Discounting
instantiates
references
CalPrice
constructor CalPrice

implementes
instantiates
references

<<Java class >> Pricing

<<Java class >> Discounting
constructor Discounting
instantiates
<br>
Box Processing Stages

- Locate
- Compile
Locate

warehouse_root

boxes  interfaces  datatypes
Locate

Warehouse_root
boxes

CalPriceAbs
CalPrice
CalPrice.box
CalPrice.conf
CalPrice.java

PricingAbs
Pricing
Pricing.box
PrImp.java

DiscountingAbs
Discounting
Discounting1
Discounting1.box
Disimp1.java

Discounting
Discounting.box
DisImp.java
For an atomic box, the implementation of an interface needs to

• specify the full path of its package name

• import interfaces and datatypes

[d:\warehouse_root\boxes\PricingAbs\Pricing\Prlmp.java]

package boxes.PricingAbs.Pricing;
import interfaces.Price;
import interfaces.Discount;
import datatypes. systems.*;

public class Prlmp implements Price {
    private BoxTop _box;
    Discount dc;    // required interface
    public Prlmp(BoxTop myBox)
    {
        _box = myBox; }

    ....  ....
Locate

Warehouse_root

interfaces

Price.java

Discount.java
package interfaces;

public interface Discount {
    double getDiscount(int client, int item, int quantity);
}
Locate

Warehouse_root

datatypess

PriceList.java
package datatypes;
public final class PriceList extends Object{
    public static double [] p = new double []{
        120.00, 14.45, 16.99, 23.78, 130.89, 239.99,
        18.99, 234.70, 3.99, 6.78, 76.50, 1299.99,
        34.67, 54.20, 67.89, 89.10, 17.50, 22.70
    };
}
Compile

BoxCompiler

- Implemented in BoxScript
- Checks the syntax of the source code
- Generates box manager code

Usage

boxc <box description>

Eg. boxc <dir>CalPrice.box
Box Variants

- Support flexibility

Variants

- One box is variant of another
  if implementations of same abstract box
- One box variant can be substituted by another variant where their abstract box is used
- Variant should conform to its abstract box
Box Conformity

Suppose
- IP2’ extends IP2
- IR1’ extends IR1

B conforms to BAbs

B conforms to BAbs
Box Variant Example

- PricingAbs
  - Pricing
    - variants
  - Pricing2
- DiscountingAbs
  - Discounting
    - variants
  - Discounting2
Replacing Components

box CalPrice implements CalPriceAbs
{   composed from PricingAbs boxP,
    DiscountingAbs boxD;
    provided interfaces Price tPrice from boxP.Pr;
    connect boxP.Dc to boxD.Dis;
}

[CalPrice.conf]
(boxP, "D:\warehouse_root\boxes\PricingAbs\", "Pricing\Pricing");
(boxD, "D:\warehouse_root\boxes\DiscountingAbs\",
    "Discounting\Discounting");

Now, we want to use Pricing1 to substitute for Pricing
Replacing Components

Things we need to do:
- Update conf file for CalPrice
- Re-compile CalPrice

[CalPrice.conf]
(boxP, "D:\warehouse_root\boxes\PricingAbs","Pricing1\Pricing1");
(boxD, "D:\warehouse_root\boxes\DiscountingAbs","Discounting\Discounting ");
Demo

- Make an abstract box
- Make an atomic box
- Make a compound box
Example – KWIC

- **Key Word In Context**
  - *Accepts* an ordered set of lines
    - each line is an ordered set of words
  - *Circularly shifts* each line
    - repeatedly removes first word and appends it at end of line
  - *Outputs* all lines in *alphabetical order*
Example – KWIC

- Key Word In Context

```
BoxScript example
```

Key Word In Context

Circularly shift

```
BoxScript example
In context Key Word
```

output

```
Key Word In Context
```

alphabetize

```
Key Word In Context
Word In context Key Word
context Key Word In
BoxScript example
example BoxScript
```
Example – KWIC

- **Design Decisions [Parnas 72]**
  - Changes in algorithm
  - Changes in data representation
Example – KWIC

- Design I

Diagram:
- Master Control
  - Input
  - Circular Shift
  - Alphabetizing
  - Output

Procedure call
Example – KWIC

- Possible changes
  - Input format
  - Have lines partially stored in memory
  - Take different formats for Word
  - Partially alphabetize lines
Example – KWIC

- Design II

- Master Control

- Input

- Circular Shift

- Alphabetizing

- Output

- Line Storage

- Datatype word
Example – KWIC

- Design II

- Master Control
  - Input
  - Circular Shifter
  - Line Storage
  - Alphabetizer
  - GetLine
  - Get ith Line
  - Output
  - Datatype
    - word
Example – KWIC

- Possible changes
  - Input format
  - Have lines partially stored in memory
  - Take different formats for Word
  - Partially alphabetize lines