Interaction-based Programming in Classages

Y. David Liu
Scott Smith

Johns Hopkins University
Interactions Illustrated by UML

CarView
+notify()

Car
+registerView()
+unregisterView()
+getRate()
+axle()

CarVTxt
+notify()

CarV3D
+notify()

Tire
+size()

observe
0..n

compose
1

Car
0..n

0..n

1..n

1..n
Interactions Illustrated by UML

CarView

+notify()

CarV3D

+notify()

CarVTxt

+notify()

Car

+registerView()
+unregisterView()
+getRate()
+axle()

Tire

+size()

Composition: 1

CarView observes 0..n Car

CarComposes 0..n with CarView

CarView communicates with CarV3D and CarVTxt
Interactions Illustrated by UML

CarView
+notify()

Car
+registerView()
+unregisterView()
+getRate()
+axle()

CarV3D
+notify()

CarVTxt
+notify()

Tire
+size()
Interactions Illustrated by UML

CarView
+notify()

Car
+registerView()
+unregisterView()
+getRate()
+axle()

CarVTxt
+notify()

CarV3D
+notify()

Tire
+size()
Class Interaction

- **CarView**
  - +notify()

- **CarVTxt**
  - +notify()

- **CarV3D**
  - +notify()

- **Car**
  - +registerView()
  - +unregisterView()
  - +getRate()
  - +axle()

- **Tire**
  - +size()

Relationships:
- CarView observes 0..n Car
- CarVTxt observes 0..n Car
- CarV3D observes 0..n Car
- Car composes 1 Tire
- Tire has 1..n relationships with Car

Note: The UML diagram represents the class interaction and relationships.
Object Peer-to-Peer Interaction

CarView
+notify()

Car
+registerView()
+unregisterView()
+getRate()
+axle()

CarVTxt
+notify()

CarV3D
+notify()

Tire
+size()
Object Whole-Part Interaction

CarView
  +notify()

CarV3D
  +notify()

CarVTxt
  +notify()

Car
  +registerView()
  +unregisterView()
  +getRate()
  +axle()

Tire
  +size()
Limitations of Mainstream OOPLs on Object Interactions
Limitations of Mainstream OOPLs on Object Interactions

- No explicit support for whole-part interactions.
- Inadequate support for coarse-grained interactions between objects.
- An object has only one encapsulation-enforceable interface for all interactions it might participate in.
- Interaction bi-directional dependencies are not explicit.
Object Whole-Part Interaction

CarView
  +notify()

CarV3D
  +notify()

CarVTxt
  +notify()

Car
  +registerView()
  +unregisterView()
  +getRate()
  +axle()

Tire
  +size()
Object Whole-Part Interaction

Why does it matter?
- memory management
  - garbage collection
- internal representation protection
  - object ownership
  - alias protection

```
Car
+registerView()
+unregisterView()
+getRate()
+axle()
```

```
CarView
+notify()
```

```
Tire
+size()
```

```
0..n
observe
0..n
```
Limitations of Mainstream OOPLs on Object Interactions

- No explicit support for whole-part interactions.
- **Inadequate support for coarse-grained interactions between objects.**
- An object has only one encapsulation-enforceable interface for all interactions it might participate in.
- Interaction bi-directional dependencies are not explicit.
Representing the *observe* Interaction

```
Class: Car
  +registerView()
  +unregisterView()
  +getRate()
  +axle()

Class: CarView
  +notify()

Class: CarVTxt
  +notify()

Class: CarV3D
  +notify()

Class: Tire
  +size()
```

Diagram:
- `CarView` 0..n *observe* 0..n `Car`
- `CarVTxt` and `CarV3D` are children of `CarView`.
- `Tire` 1..n `compose` `CarView`
Representing the *observe* Interaction

### Why does it matter?
- object field is not enough
- related work: relationship representation: RelJ [ECOOP'05]
Limitations of Mainstream OOPLs on Object Interactions

- No explicit support for whole-part interactions.
- Inadequate support for coarse-grained interactions between objects.
- An object has only one encapsulation-enforceable interface for all interactions it might participate in.
- Interaction bi-directional dependencies are not explicit.
Only One Interface for All Interactions

CarView
+notify()

Car
+registerView()
+unregisterView()
+getRate()
+axle()

CarVTxt
+notify()

CarV3D
+notify()

Tire
+size()
Only One Interface for All Interactions

```
CarView
  +notify()

CarV3D
  +notify()

CarVTxt
  +notify()

Car
  +registerView()
  +unregisterView()
  +getRate()
  +axle()

Tire
  +size()
```

Diagram:
- CarView
- CarV3D
- CarVTxt
- Car
  - observe 0..n
- Tire
  - compose 1
  - size 1..n
Only One Interface for All Interactions

Why does it matter?

- principle of least privilege
- Java interface is not enough
- related work: Encapsulation Policies [Scharli et al, ECOOP'04]
Limitations of Mainstream OOPLs on Object Interactions

• No explicit support for whole-part interactions.
• Inadequate support for coarse-grained interactions between objects.
• An object has only one encapsulation-enforceable interface for all interactions it might participate in.

• Interaction bi-directional dependencies are not explicit.
  • Callbacks
Limitations of Mainstream OOPLs on Class Interactions
Limitations of Mainstream OOPLs on Class Interactions

- **Interaction bi-directional dependencies are not explicit.**
  - Dependencies between superclasses and subclasses are fundamentally bi-directional.

- Interfaces for class interactions are tangled with those for object interactions.

- A superclass has only one interface for all subclasses.
Limitations of Mainstream OOPLs on Class Interactions

- Interaction bi-directional dependencies are not explicit.
- Interfaces for class interactions are tangled with those for object interactions.
  - related work: Traits
- A superclass has only one interface for all subclasses.
Limitations of Mainstream OOPLs on Class Interactions

- Interaction bi-directional dependencies are not explicit.
- Interfaces for class interactions are tangled with those for object interactions.
- A superclass has only one interface for all subclasses.
The Classages Solution
The Simple Example Revisited

CarView
  +notify()

Car
  +notify()
  +registerView()
  +unregisterView()
  +getRate()
  +axle()

CarV3D
  +notify()

CarVTxt
  +notify()

Tire
  +size()

observe
0..n
compose
1
1..n
The Same Example in Classages
Classage Basics: Classages
Classage Basics

Classage Interfaces

GUI3D

updateGUI
updateRate

Super

updateGUI
updateRate

Car

Source

Notifier

CarView

getRate

notify

getRate

notify

size

size

axle

axle

Main

Tires

Tire
Classage Basics

GUI3D
- updateGUI
- Super
- updateRate

GUITxt
- updateGUI
- Super
- updateRate

updateGUI
- Sub
- updateRate

updateGUI
- notify
- getRate

Notifier
- notify
- getRate

Source
- notify
- getRate

Car
- size
- axle
- Main

Tires
- size
- axle

Tire
Classage Basics

Imports, Exports

GUI3D
updateGUI
Super
updateRate

GUITxt
updateGUI
Super
updateRate

CarView
notify
gRate

Source
notify
gRate

Notifier

Car

Tires
size
axle

Main

Super

Super

Super

UpdateGUI

UpdateGUI

UpdateGUI

UpdateGUI

UpdateRate

UpdateRate

UpdateRate

Imports, Exports

Classage Basics
Classage Basics

GUI3D

updateGUI
(updateRate)

Super

mixer

Car

Tires

size

axle

Main

updateGUI

Sub

getRate

updateGUI

getRate

notifier

Notifier

Super

Source

notifier

plugger

CarView

updateGUI

updateRate

updateGUI

updateRate
A Demo
A Demo

CarView

GUI3D

GUITxt

Super

Sub

Source

Notifier

Car

Main

Tires

Tire
classage CarVTxt = GUITxt + CarView with Super >> Sub

At Compile Time...
Mixing: the Class Interaction

GUITxt

CarView

Source

Sub

CarView

Source

Notifier

Car

Tires

Super

GUI3D

Main

Tire
Mixing: the Class Interaction

CarVTxt

GUI3D

CarView

Car

Tires

Super

Sub

Source

Notifier

Main

Tire
Mixing: the Class Interaction

classage CarV3D = GUI3D + CarView
   with Super >> Sub
Mixing: the Class Interaction

Source

GUI3D

CarVTxt

Notifier

Car

Tires

Main

CarView

Tire

Source
Mixing: the Class Interaction
Now the program is running...
Objectage Instantiation

```
o1 = create CarVTxt();
o2 = create CarV3D();
o3 = create CarV3D();
o4 = create Car();
```
Objectage Instantiation

- o1: CarVTxt
- o2: CarV3D
- o3: CarV3D
- o4: Car

Source

Notifier

Tires

Main

Tire
Plugging: Whole-Part Interaction

Inside o4:
pl₁ = plugin Tire with Tires >> Main;
Plugging: Whole-Part Interaction

Inside o4:

\[ p1 = \text{plugin Tire with Tires} \gg \text{Main}; \]
Plugging: Whole-Part Interaction

Inside o4:
\[ p2 = \text{plugin Tire with Tires} \gg \text{Main}; \]
Plugging: Whole-Part Interaction

Inside o4:
\[ p2 = \text{plugin Tire with Tires} \gg \text{Main}; \]
Plugging Handles

Inside o4:

```java
int s1 = p1.size();
int s2 = p2.size();
```
Inside o1:
\[ c1 = \text{connect } o4 \text{ with Source } >> \text{Notifier}; \]
Connection Established

Inside o1:

c1 = connect o4 with Source >> Notifier;
Inside o2:
\[ c2 = \text{connect } o4 \text{ with Source } \gg \text{Notifier}; \]
Inside o2:

c2 = connect o4 with Source >> Notifier;
Inside o3:
\[ c3 = \text{connect o4 with Source >> Notifier}; \]
More Connections

Inside o3:

c3 = connect o4 with Source >> Notifier;
For All Connections

Inside o4:
forall(c: Notifier) {c->notify(); }
For All Connections

Inside o4:
forall(c: Notifier) {c->notify(); }
Inside `notify` of `o1`:

```java
int r = getRate();
```
Inside `notify` of `o1`:

```java
int r = getRate();
```
If you are Java programmer...

class CarView {
    private Car source;
    ...
    public notify () {
        ...
        int r = source.getRate();
        ...
    }
}

Inside notify of o1:
int r = getRate();
Stateful Connections

Inside `notify` of Source:

```plaintext
::counter = ::counter + 1;
```
Inside o1:

disconnect c1;
Inside o1:
\texttt{disconnect \ c1;}
Inside o4:

unplug p2;
Inside o4:
unplug p2;
Dynamic Dispatch

Sub

CarView

Source

notify
Dynamic Dispatch

![Diagram of Dynamic Dispatch]

- **Sub**
- **Source**
- **notify**

- **GUI3D**
- **CarView**
- **Super**
- **Sub**
- **Source**
- **notify**
Dynamic Dispatch

objectage type CarV3D <: objectage type CarView
Static Dispatch

CarView

Sub

Source

notify

notify
Static Dispatch

GUI3D  notify  CarView

Super         Sub
Source

notify
Overridable Method

Sub

notify

notify

notify

Source

notify

notify

CarView
Dynamic Dispatch: CarV3D

GUI3D  notify  Sub  notify  notify  notify  Source

CarView
Dynamic Dispatch: Objectage CarV3D

![Diagram of object interaction in CarV3D]
Dynamic Dispatch: Objectage CarView

CarView

notify

Sub

Source

notify

notify
The Type System

- static typechecking for (dynamic) interactions: bi-directional interface match with subtyping.

- protecting internal representation: avoiding plugging handles to escape.

- no connection masquerading: avoiding connection handles to escape.
Mixing: The Class Interaction

![Diagram of class interaction with GUI3D and CarView]

Super

Sub

Source
Plugging: The Whole-Part Interaction
Connection: The Peer-to-Peer Interaction
The Type System

• static type checking for (dynamic) interactions: bi-directional interface match with subtyping.

• **protecting internal representation:** avoiding plugging handles to escape.

• no connection masquerading: avoiding connection handles to escape.
Protecting Internal Representation
Protecting Internal Representation
The Type System

- static typechecking for (dynamic) interactions: bi-directional interface match with subtyping.

- protecting internal representation: avoiding plugging handles to escape.

- no connection masquerading: avoiding connection handles to escape.
No Connection Masquerading
Related Work

- explicit interfaces
  - component systems
  - architectural description languages
- object ownership, alias protection
- composition: mixins, Traits, module systems
- relationship representation
- environmental acquisition [Gil & Lorenz], [Cobbe & Felleisen]
Classages Design Principles

- Static interactions and dynamic interactions are fundamentally different.
- Internal interactions and external interactions are fundamentally different.
- Interactions fundamentally have a lifespan.
- Interactions are fundamentally bi-directional.
- Interactions always happen on explicitly defined interfaces.
Download

http://www.cs.jhu.edu/~yliu/Classages