Plugin reuse and adaptation with Object Teams:

Don't settle for a compromise!

Stephan Herrmann
Technische Universität Berlin

www.objectteams.org
innovation

speed

requires: unanticipated

compromise

provides: anticipated

reuse

quality
The Game

innovation

requires: unanticipated

maintenance

provides: anticipated

compromise

speed

reuse

quality

www.objectteams.org
With **Object Teams** it is possible

- to eat the cake
  - apply unanticipated adaptations

- and still have it
  - sustain a well modularized, maintainable design

**OT/Equinox** brings this power to the development of Eclipse plugins
Unanticipated Adaptation

Object oriented adaptation: Inheritance

- program by difference
- choose at instantiation time

This is good, but:

- who controls instantiation?
- behavioral changes after instantiation?
- multiple (independent) adaptations?

We need something similar to inheritance

- apply to objects not classes:
- adapt any time, any number

role objects
team
  collaboration module

role
  members of a team

playedBy
  connect role to base

callout
  forward to base

callin
  intercept base method

decapsylation
  break base encapsulation

(de)activation
  dis/enable all callins

```java
MyTeam
  teamField: someType
  teamMethod(T2): Type2

Role1
  roleMeth1()
  roleMeth2()

BasePkg
  C1
    method1()
    method2()

Role2
  roleMeth1 -> method1
  roleMeth2 <- method2

C2
  «playedBy»

SubTeam
  otherField: otherType
  otherMethod(T3): Type3
```
Inheritance of complex structures

- virtual methods and classes
- propagating specialization

```java
public team class T2 extends T1 {
    @Override protected class R2 {
        @Override void m3() {
            doMyStuff();
        }
    }
}
```
Object Teams

- **ObjectTeams/Java (OT/J)** since 2001
  - Java += roles, teams, aspect bindings

- **Object Teams Development Tooling** since 2003
  - Java Compiler += OT/J constructs
  - JDT for OT/J (code assist, ui, launch ...)

- **OT/Equinox** since 2006
  - Equinox += aspect bindings

- **Application**
  - Case studies (project TOPPrax)
  - Class room
  - OTDT

**WEOODF** (we eat our own dog food)
Plug-in relationships

MANIFEST.MF
... Require-Bundle: B ...

plugin.xml

<extension point="org.objectteams.otequinox.aspectBindings">
  <aspectBinding>
    <basePlugin id="B"/>
    <team class="Team1"
      activation="ALL_THREADS"/>
  </aspectBinding>
</extension point="org.objectteams.otequinox.aspectBindings">

Plug-in A

CA1

CA2

1 «require»

Plug-in B

export

internal

CB1

CB2

CB3

CB4

2 extension point

Plug-in C

Team1

CC1

R1

import base CB1;
import base CB3;

R2

rm←bm

«require»

«aspectBinding»

«playedBy»

«playedBy»
OT/Equinox Hello World

Create a plugin that
- makes “Foo” an illegal Java type name
  - can no longer do demos
  - “AntiDemo”

Prerequisite
- Have identified a join point in
  - `org.eclipse.jdt.core.JavaConventions.validateXYZ()`
Demo 1 – Summary

- Development of OT Plug-ins
  - Wizards, validation, content assist, ...
  - Minimal: 1 aspect binding, 1 team, 1 role, 1 callin

- Running Eclipse with OT/Equinox
  - Enable OT/Equinox

- Inside Eclipse
  - About Plug-ins
    - inspect which plug-ins are adapted
  - OT/Equinox Monitor
    - inspect active team instances
    - dynamically (de)activate

one more lesson we learned ...
Demo 1 – Summary

- Development of OT Plug-ins
  - Wizards, validation, content assist, ...
  - Minimal: 1 aspect binding, 1 team, 1 role, 1 callin

- Running Eclipse with OT/Equinox
  - Enable

- Inside Eclipse
  - About Plug-ins
    - inspect
  - OT/Equinox Monitor
    - inspect
    - dynamically (de)activate

Adapting the core
- not the symptoms
improves consistency

one more lesson we learned ...
Check this consistency constraint

"A plug-in that defines aspect bindings with team activation must have an activation policy (lazy)."

Could write a complete own validator

- parse plugin.xml and MANIFEST.MF
- report errors
- offer quickfix
- get triggered on file changes

Easier: reuse PDE – piggy-back implementation

- find join points
- (just) implement the above rule
Found two interesting methods in `org.eclipse.pde.internal.core.builders`:

- `ExtensionsErrorReporter.validateExtension(Element)`
  - invoked for each extension in `plugin.xml`
  - can be used as trigger for detected aspect bindings

- `BundleErrorReporter.validateBundleActivatorPolicy()`
  - specifically checks the activation policy in `MANIFEST.MF`
  - can be used to add our validation
Found two interesting methods in org.eclipse.pde.internal.core.builders

- ExtensionsErrorReporter

```java
protected class ExtensionAnalyzer playedBy ExtensionsErrorReporter {
    void checkAspectBinding(Element element) <- after void validateExtension(Element element);
    ...
}
protected class BundleErrorReporter playedBy BundleErrorReporter {
    void validateBundleActivatorPolicy() <- after void validateBundleActivatorPolicy();
    ...
}
```

- can be used to add our validation

- But: those two objects don't know each other!
  - what is the common context of both join points?
Bundle Validation – Context

**BundleValidation**

bundleContext: ThreadLocal(BundleCheckingContext)

- :BundleCheckingContext
  - isAspectBundle
  - hasTeamActivation

- :Extension-Analyzer

- :Bundle-ErrorReporter

- :Manifest-Consistency-Checker
  - validateFiles

- :Extensions>ErrorReporter

- :Bundle-ActivatorPolicy
  - validateBundle-ActivatorPolicy

- Manifest-Consistency-Checker

- Extension

- Bundle-ErrorReporter

- Bundle-ErrorReporter

- Bundle-ActivatorPolicy

- set(this)

- set(null)

- get()

- get()
protected class BundleErrorReporter playedBy BundleErrorReporter

    base when (BundleValidation.this.bundleContext.get().isAspectBundle)
Team – Context – View

- A **team** defines a **view**
  - map only relevant elements
    - classes using `playedBy`
    - events using `callIn`
    - provided actions using `callOut`
  - team & roles is a self-contained world

- A **team** and its roles define **context**
  - store context specific state
  - context can be defined ...
    - by a team instance
    - by a role instance / a graph of ...
    - per thread
    - per control flow
    - ...

Roles may
- start disconnected
- discover each other later
- superimpose structure
A complex features applies only conditionally

- identify the relevant situation with a control-flow
- define two teams
  - **feature team** is inactive by default
  - **guard team**
    - observes initial trigger
    - instantiates and activates feature team
- use this pattern:

```java
trigger ← after someEvent;
calling void trigger() {
   within (new FeatureTeam()) {
       base.trigger();
   }
}
```

- thread safe
- exception safe
From-Scratch vs. Piggy-Back

Don't bother with ...
- locating files (plugin.xml and MANIFEST.MF)
- parsing XML and manifest syntax
- mechanics of re-writing the manifest
- receiving triggers on file changes

All this is already implemented => re-use it!!

Only implement the net value
- less code to write
- less code to read
- it's all in one place

Find join points to hook into
- this task is new
Abstracting to a general pattern

Observer-Mediator-Actuator

Mediator

mediatorStorageField : DataStructure

ObserverRole

ActuatorRole

MediatorRole1  MediatorRole2

write  read

additionalReference

write

intercept  decorate  intercept  decorate

(inter non-modifying) (modifying)

adaptation

base application

C1  C2  C3  C4
Example Call Hierarchy

From 3.4 New&Noteworthy

- call hierarchy view works with fields ...
Example Call Hierarchy

- **From 3.4 New&Noteworthy**
  - call hierarchy view works with fields ...
  - first released in the OTDT
  - later refactored and contributed to Eclipse

- **More control flows:** 🛡 callin & ⛽ callout
  - prerequisite: can already search for callin & callout bindings

ui part: adaptation using ...
  - 1 team adapting 1 base plugin (org.eclipse.jdt.ui)
  - 6 roles adapting 6 base classes
  - 4 callin bindings, 12 callout bindings

365 LOC
Example Call Hierarchy

- **From 3.4 New&Noteworthy**
  - call hierarchy view works with fields ...
  - first released in the OTDT
  - later refactored and contributed to Eclipse

- **More control flows: \(\text{callin} \& \text{callout}\)**
  - prerequisite: can already search for callin & callout bindings

**UI part: adaptation using ...**

- 1 team adapting 1 base plugin (org.eclipse.jdt.ui)
- 6 roles adapting 6 base classes
- 4 callin bindings, 12 callout bindings

- **Future: declaratively induced control flows**
  
  - plugin activator: \(\text{MyActivator.start()}\)
  
  - aspect binding: \(\text{new MyTeam()}\)
From 3.4 New & Noteworthy
- call hierarchy
- first released
- later refactored

More control flows: callin & callout
- prerequisite: search already finds callin & callout

UI part: adaptation using...
- 1 team adapting 1 base plugin (org.eclipse.jdt.ui)
- 6 roles adapting 6 base classes
- 4 callin bindings, 12 callout bindings

Future: declaratively induced control flows
- plugin activator: → MyActivator.start()
- aspect binding: → new MyTeam()
Example: Launching with OT/J

First approach:
new launch configuration type
- replace Main
- augment classpath
- change program arguments

Duplicate for each of
- JUnit Test launches
- Eclipse Application launches
- OSGi Framework launches
- JUnit Plugin Test launches
- ...

www.objectteams.org
Second approach: add OT-capability to existing launches
- new checkboxes next to JRE selection
  - defaults from the project context
- add team activation tab
- adapt the command line behind the scenes

Benefits
- less code
- more consistency
- composable launches

Stats:
- adapting 7 plugins
- 8 teams, 16 roles, 660 LOC

Adapting the core
- not the symptoms - improves consistency
Second approach: add OT-capability to existing launches

- new checkboxes next to JRE selection
  - defaults from the project context
- add team activation tab
- add

Benefits:

- less code
- more consistency
- composable launches

Stats:

- adapting 7 plugins
- 8 teams, 16 roles, 660 LOC

Adapting the core – not the symptoms – improves consistency
Meaningful modules
- comprehensible: only one feature
- comprehensive: all of one feature

Join points may change – seldomly
- mostly “syntactical” changes: ~ refactoring
- very few “semantical” changes
  - join point called differently
  - need to refine detection of relevant situation
  - OT provides sufficient means
- embrace change with agility

OTDT built on OT/Equinox since 2006
- migrating the OT-plugins: minimal effort
Conclusion

Find something similar to what you need?

- use it! adapt it to exactly your requirements!
- near miss is no excuse!
- make the adaptation a meaningful module!
- piggy-back adaptation where suitable!
- feature = module = context = team!
- quickly innovate!

No Compromise!

Coming next

- combining OT with generative techniques
  - modeling (e.g., UML 2 tools)
  - IMP?