The Object Teams Development Tooling:
A high fidelity extension of the JDT

Stephan Herrmann
Technische Universität Berlin
The Characters in this Play

ObjectTeams/Java

ObjectTeams/Java

ObjectTeams/Java

ObjectTeams/Java

ObjectTeams/Java

ObjectTeams/Java

ObjectTeams/Java

ObjectTeams/Java

ObjectTeams/Java

ObjectTeams/Java

ObjectTeams/Java

ObjectTeams/Java

ObjectTeams/Java

ObjectTeams/Java

ObjectTeams/Java

ObjectTeams/Java

ObjectTeams/Java

ObjectTeams/Java

ObjectTeams/Java

ObjectTeams/Java

ObjectTeams/Java

ObjectTeams/Java
OO is not the end of language development

• E.g., inheritance is great, but ...
  
  A text book example:
  - A man/woman is a person, OK
  - An employee is a person, OK?
    ▪ Born as an employee?
    ▪ Dying when loosing the job?
    ▪ Several jobs, yet only one salary?

• What's wrong with inheritance?
  - Missing “become”, “quit” 😞
  - Can't duplicate fields 😞
  - Employee & Person = 1 instance 😏

• Can we do better?
  - Yes:
    - Employee is a Role played by a Person
playedBy relationship

- **Advantages:**
  - **Dynamism:** roles can come and go (same base object)
  - **Multiplicities:** one base can play several roles (different/same role types)

- **Is this completely new?**
  - No, has been around >15 years
  - playedBy is similar to extends
  - how similar?
Comparing at a closer look

• Inheritance
  - Import / acquisition
    - dispatch sub \(\rightarrow\) super
  - Overriding
    - dispatch super \(\rightarrow\) sub
  - Subtype polymorphism
    - substitutability

• Role Playing
Import in OT/J:

- A callout method binding ...

\[
\text{String getName()} \rightarrow \text{String getName();}
\]

... declares that calls to the role should be **forwarded** to its base

- ... can use different names on role / base sides
- ... can adjust signatures
  - implicitly: discard unused values
  - explicitly: parameter mappings
Overriding in OT/J:

• A callin method binding ...

```java
String getPhoneNo() <- replace String getPhoneNo();
```

... declares that calls to the base should be **intercepted** by its role

- ... can use different names on role / base sides
- ... can adjust signatures
  - implicitly: discard unused values
  - explicitly: parameter mappings
- ... comes in one of three flavors: **before**, **replace** or **after**
“... when (s)he is in the office ...”

• *How do you know?*

• **Roles depend on context**

• In OT/J contexts are reified as **Teams**
  - roles are inner classes of a *team class*
  - role instances are inner instances of a *team instance*

• Each team instance can be *(de)activated*
  - (several mechanisms: globally, per thread, implicitly, temporarily ...)

```java
Person c : Company
   name
   hire(Person p)
   if (c.isActive())
      . getPhoneNo()

Employee : Person
   officePhoneNo
   getPhoneNo ← getPhoneNo
```

```java
Employee : Person
   officePhoneNo
   getPhoneNo ← getPhoneNo
   getPhoneNo()
```
Substitutability?

• Are the following assignments legal?

```java
Employee emp = ... 
Person person = ... 
1. person = emp;  // legal?
2. emp = person;  // legal?
```

Normally not, but...

• Idea: roles live (usually) only within the team

• When a role object `leaves` the team
  ▷ it is `lowered` to its base

• When a base object `enters` a team
  ▷ it can be `lifted` to a role
### Comparison

<table>
<thead>
<tr>
<th>Inheritance</th>
<th>Role Playing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import / acquisition</td>
<td>Import / acquisition</td>
</tr>
<tr>
<td>- dispatch sub → super</td>
<td>- dispatch role → base</td>
</tr>
<tr>
<td>Overriding</td>
<td>Overriding</td>
</tr>
<tr>
<td>- dispatch super → sub</td>
<td>- dispatch base → role</td>
</tr>
<tr>
<td>Subtype polymorphism</td>
<td>Translation polymorphism</td>
</tr>
<tr>
<td>Substitutability</td>
<td>lowering role → base</td>
</tr>
<tr>
<td>- pass an instance of sub class where the super class is expected</td>
<td>- lifting base → role</td>
</tr>
<tr>
<td></td>
<td>two-way substitutability!</td>
</tr>
</tbody>
</table>
Summary ObjectTeams/Java

• Role playing combines the powers of inheritance with
  ◆ **Dynamism:**
    roles can come and go (same base object)
  ◆ **Multiplicities:**
    one base can play several roles (different/same role types)

• Teams
  ◆ **encapsulate** a set of interacting roles
  ◆ team **activation** controls the effect of all contained **callin** bindings
  ◆ create **larger structures**  (*stacking / nesting / layering*)
Tooling for ObjectTeams/Java

• **Goal:** “A **high-fidelity** extension of the JDT”
  - The same quality (code & user experience)
  - Unrestricted Java-development
  - Seamless support for specifics of OT/J

• **Need to cover (minimum):**
  - Compiler
  - Run/Debug
  - Help
  - Integration with PDE
  - Editor
  - Code Assist
  - Refactoring
  - Wizards
  - Structure Viewers
  - Search / Call Hierarchy

• **Goal:** “A high-fidelity **extension** of the JDT”
  - Maximum reuse
  - Maintainable
  - Evolvable
Working with the OTDT

<DEMO 1> Summary:

• Run as Object Teams Application
  • configuration instantiate & activate a Team

• Wizard: created role with playedBy

• Editor seamlessly highlight etc.

• Compiling seamlessly: incremental, eager ...

• Code assist create OT/J elements
  apply std. assists to OT/J

• Help! link problems to language definition
Working with the OTDT

<DEMO 1> Summary:

- Run
  - configuration
- Wizard:
  - Editor
  - Compiling
  - Code assist
- Help!

- Seamless integration
  - Object Teams Runtime Environment
  - Run as Object Teams Application
- Additional configuration
  - Team Activation
Working with the OTDT

<DEMO 1> Summary:

- Run
  - configuration
- Wizard:
  - Editor
  - Compiling
  - Code assist
- Help!

- Project creation
  - Object Teams Project
  - Object Teams Plugin Project
- Class creation
  - Team
  - Role playedBy Base
  - inline
  - role file
Working with the OTDT

<DEMO 1> Summary:

• Run
  • configuration
• Wizard:
• Editor
• Compiling
• Code assist
• Help!

♦ Syntax highlighting
  • includes other views like Compare
♦ Semantic highlighting
♦ Navigation
  • F3 everywhere!
  • Ctrl-O Ctrl-O
  • Callin markers
  • etc.
Working with the OTDT

<DEMO 1> Summary:

• Run
  ▶ configuration

• Wizard:
• Editor

• Compiling
• Code assist

• Help!

❖ Full Java 5 support
❖ Incremental compilation
❖ Eager & partial compilation
  ▪ sophisticated error recovery (🚀 content assist)

❖ Speed

❖ Scoped keywords:
  ▪ plain identifiers when used outside team
Working with the OTDT

<DEMO 1> Summary:

- Run
  - configuration
- Wizard:
- Editor
- Compiling
- Code assist
- Help!

Completion

- OT/J keywords
- create/complete method bindings (callout/callin)
- support std. completions in OT/J code, too. (work in progress)
Working with the OTDT

<DEMO 1> Summary:

• Run
  - configuration
• Wizard:
• Editor
• Compiling
• Code assist

Quick Fix
  - modifier problems
    - class, method, method binding
  - learn about special OT/J features
  - type errors
  - coding style
  - adapt JDT quick fixes to handle OT/J elements/rules, too:
    - suppress warnings, create method, organize imports

• Help!
Working with the OTDT

<DEMO 1> Summary:

- Run
  - configuration
- Wizard:
- Editor
- Compiling
- Code assist

- New language – new error messages
  - “What did I do wrong??”
  - Let us tell you.
  - Go to Language Definition
    - precise links
    - comprehensive
    - single source: XML → {PDF, XHTML (x2)}

- Tutorial
- Code Samples
- Developers' Guide
Debug

<DEMO 2> Summary:

• Stepping  showing the right code
• Debug View  tell the user what's happening
• Team Activation  show and manipulate program mode
Debug

<DEMO 2> Summary:

- Stepping
- Debug View
- Team Activation

- Source mapping
  - uses JSR 45 / SMAP
- Filter runtime library code
Debug

<Demo 2> Summary:

• Stepping

• Debug View

• Team Activation

♦ Beautify special stack frames
  ▪ revert generated names
  ▪ use custom syntax & coloring
    ✧ dispatch code
      {{Dispatch method foo}}
    ✧ lifting invocations
      {{Lifting to MyRole}}
  ✧ executing declarative bindings
    [foo <- bar]
  ✧ base calls: almost normal method calls
    base.foo()
Debug

<DEMO 2> Summary:

• Stepping
• Debug View
• Team Activation

Team Monitor View
- derived from Variables view
- all known team instances:
  - active
  - implicitly active
  - inactive
- interactively change activation
  - switch program modes

Correspondence:
runtime stack traces ↔ static call hierarchies
Search

• Core
  ♦ find references within OT/J constructs

• UI
  ♦ display found OT/J elements
  ♦ hide generated elements
  ♦ beautify mangled names

Call Hierarchy

• OT/J specific control flows
  ♦ method invocations due to callout/callin method bindings

• While we're at it ...
  ♦ control flows resulting in assignment to a given field (see https://bugs.eclipse.org/75800)
Structure Viewers

- Package explorer
- Type Hierarchy
Structure Viewers

- **Package explorer**
  - overlays for packages (package = team, containing role files)
  - switch physical/logical presentation of role files

```
fbapplication
  ▼ GUIConnector.java 1645:
  ▼ fba~.GUIConnector
  ▼ FlightBonusDialog.java 1
  ▼ FlightBonusDialog 164
```

- team = class
- team = package
- role = separate compilation unit
- role = inner class

role itself is a team, containing nested roles

- **Type Hierarchy**
Structure Viewers

- Package explorer
- Type Hierarchy
Structure Viewers

- Package explorer
- Type Hierarchy
  - Team inheritance induces new structure
    - role classes can be overridden (virtual classes)
    - overriding role implicitly inherits from previous version
    - a role may have multiple supers
Binding Editor

- Genuinely new view
- Idea: table based editing of these bindings:
  - playedBy
  - callout
  - callin

- Implemented
  - as a dialog
  - using AST rewriting
Refactoring

• Ensure soundness of existing refactorings
  • Java refactorings must not break OT/J code
  • done for some fundamental refactorings
    ▪ implementation was based on JDT 3.0
    ▪ currently only partly supported
    ▪ migration is work in progress

• New refactorings
  • want to create OT/J structures by refactoring
  • current stage: planning
Facing real problems

• OTDT should be similar to JDT, just different
  Difference is
   not anticipated by JDT developers
   nor likely to be supported by future version of the JDT
    ✦ too many hooks needed,
    ✦ scenarios are too specific
• “I need this hook now.”
  ✦ Copy & paste? Other ways of hacking the base?
• “My adaptation must be maintainable.” – “clean”
  ✦ one module per feature (and vice versa)
  ✦ well-defined / narrow interface to existing plugins.
• That's what OT/J has been developed for!
• Our “trick”:
  ✦ Plugins written in ObjectTeams/Java
Plugin-based Architecture

- Modules
- Relations
Declaring Aspect Bindings

**Extension point** `org.objectteams.otequinox.aspectBindings`

- basePlugin
- team

- Limited privileges
- Deployment
  - Instantiation
  - Activation
E.g., adapting “About Bundles”

```java
public team class BrandingAdaptor {
    protected class AboutBundleAdaptor playedBy AboutBundleData {
        callin String getVersion() {
            String adaptationString = ""; //NON-NLS-1$
            String pluginID = getID();
            if (TransformerPlugin.getDefault().isAdaptedBasePlugin(pluginID)) {
                HashSet<String> reportedPlugins = new HashSet<String>);
                adaptationString = BrandingMessages.BrandingAdaptor_OT_adapted_by;
                for (String element : TransformerPlugin.getDefault().getAdaptingAspectPlugins(pluginID)) {
                    if (!reportedPlugins.contains(element)) {
                        adaptationString += 
                        "* " + element; //NON-NLS-1$
                        reportedPlugins.add(element);
                    }
                }
            }
            return base.getVersion() + adaptationString;
        }
        String getID() -> String getId();
        getVersion <- replace getVersion;
    }
}
```

Override 1 single method
Typical adaptation: extend switch-case

```java
protected class SuppressWarningsAdaptor playedBy SuppressWarningsSubProcessor {
    static callin void addSuppressWarningsProposal(ICompilationUnit cu, ASTNode node,
        String warningToken, int relevance, Collection<ASTRewriteCorrectionProposal> proposals)
    {
        // adding one case block to the front of the original method:
        ChildListPropertyDescriptor property= null;
        String name;
        Object baseElement;
        switch (node.getNodeType()) {
            case ASTNode.CALLIN_MAPPING_DECLARATION:
                property= CallinMappingDeclaration.MODIFIERS2_PROPERTY;
                baseElement= // otj specific code here
                name= // otj specific code here
                break;
            // other similar cases omitted
            default:
                // other cases are already handled by the original method.
                base.addSuppressWarningsProposal(cu, node, warningToken, relevance, proposals);
                return;
        }
        String label= // otj specific code here
        ASTRewriteCorrectionProposal proposal= // otj specific code here
        proposals.add(proposal);
    }
    addSuppressWarningsProposal <- replace addSuppressWarningsProposal;
}  
```
Advanced OT-Plugin

- One team encapsulates many individual adaptations.
- Adapting several plugins involves several teams.
The way we've come

- **ObjectTeams/Java**
  - started in late 2001
- **OTDT**
  - started in 2003
  - 1.0.0 release with Callisto
- **OTDT on OT/Equinox**
  - since mid 2006
  - current version is 1.1.8
- **OT/Equinox for non-IDE use**

see you on [http://www.objectteams.org/](http://www.objectteams.org/)