Handle-based models with Handly

Vladimir Piskarev, 1C
pisv@1c.ru

May 27, 2014

(c) Copyright 2014 1C LLC. Made available under the EPL v1.0

It seems we have two firsts here…

- First Xtext conference
- First Handly talk

But wait, there is a third one just around the corner

- First Handly release (0.1) to coincide with Eclipse Luna

Main themes of the talk

- What Handly is about
- Project's status and outlook
- Handly and Xtext: relationship, integration

«You don't understand something until you understand it more than one way»

–Marvin Minsky

Point of view is worth 80 IQ points

- It might be worthwhile to have a look at Handly from multiple points of view

The Java Model.
One of the pillars of JDT

Overview – The 3 Pillars

Java Model – Lightweight model for views
- OK to keep references to it
- Contains unresolved information
- From project to declarations (types, methods..)

Search Engine
- Indexes of declarations, references and type hierarchy relationships

AST – Precise, fully resolved compiler parse tree
- No references to it must be kept: Clients have to make sure only a limited number of ASTs is loaded at the same time
- Fully resolved information
- From a Java file ('Compilation Unit') to identifier tokens

© Copyright 2007 by IBM Corp.; made available under the EPL v1.0

The 3 Pillars – First Pillar: Java Model

Java Model – Lightweight model for views
- Java model and its elements
- Classpath elements
- Java project settings
- Creating a Java element
- Change notification
- Type hierarchy
- Code resolve

Search Engine

AST – Precise, fully resolved compiler parse tree

© Copyright 2007 by IBM Corp.; made available under the EPL v1.0

The Java Model.
Original design motivation

The Java Model - Design Motivation

Requirements for a Java model:

- Light weight
  - Need elements to which a reference can be kept, e.g. to show in a viewer
  - Must work for big workspaces (10'000 types and more). Can not hold on resources, Eclipse is not just a Java IDE
- Fault tolerant
  - Some source does not (yet) compile, missing brackets, semicolons. Tooling should be as helpful as possible
  - Viewers like the outline want to show the structure while typing. Structure should stay as stable as possible

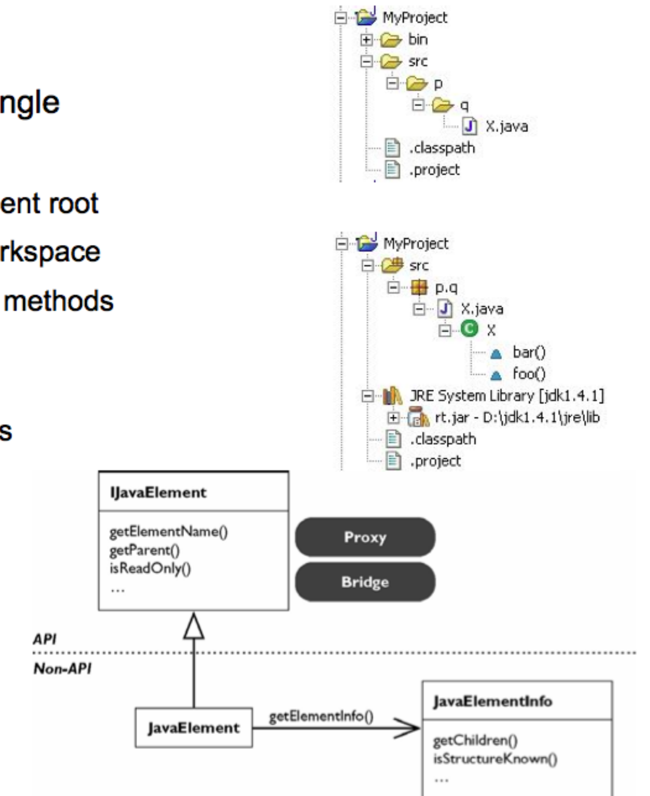
Chosen solution:

- Handle based, lazily populated model
- No resolved information kept
- Wrappers existing resource model

© 2005 by IBM; made available under the EPL v1.0

The Java Model

- Render entire workspace from Java angle
  - using '.classpath'
    - classpath entry is package fragment root
    - can even denote JAR outside workspace
  - granularity down to individual fields or methods
- Pure source model
  - accurate independently of build actions
  - fault-tolerant
  - no resolved information
- Handle/Info design
  - scalability: model non exhaustive
  - info lazily populated, LRU cache
  - stable handle



© 2005 by IBM; made available under the EPL v1.0

Language-oriented handle-based models

- Ideally suited for presenting in structured viewers
- Scalable due to virtualization made possible by a handle-based design
- Eventually consistent — need not be consistent all the time
- Can refer to non-existing elements — existence can be tested with exists()
- Tolerant to inconsistencies in the source file (syntax errors, etc.)

Point of view #1

- The Java Model as a model
- Handly supplies basic building blocks that help developers create handle-based models similar in design principles to the Java Model
- Why reinvent the wheel for every new language?

How to stay with the future as it moves?

«The best way to predict the future is to invent it»

–Alan Kay

- Hint: “Go to research labs”

The STEPS project (VPRI)

- Recreating the familiar world of personal computing in 1000 times less the amount of program code
- 20K LOC budget to express all of the “runnable meaning” (executable models) “from the end-user down to the metal”
- Just one example of dealing with languages on a large scale (“language-oriented programming”)
- It may change really everything — tooling in particular

Point of view #2

- Eclipse needs to become a great multi-language IDE platform
- Handly provides a uniform handle-based API that makes it possible to develop common components for a multi-language IDE
- In contrast to other approaches such as DLTK, Handly at its core is designed to be as language agnostic as possible

The missing piece

- Xtext is a really wonderful tool for language engineering
- It covers many important IDE components in a generic and extensible way
- However, one piece seems to be missing for a JDT-like unified IDE experience
- A handle-based model can “glue” it all together into a coherent whole

Point of view #3

- Xtext and Handly: A match made in heaven?
- Handly integrates with Xtext from the very beginning
- Looking for a synergy between the two

Handly. Points of view

- Supplies basic building blocks that help developers create handle-based models similar in design principles to the JDT Java Model
- Provides a uniform handle-based API that makes it possible to develop common components for a multi-language IDE
- Integrates with Xtext from the very beginning

The project's scope

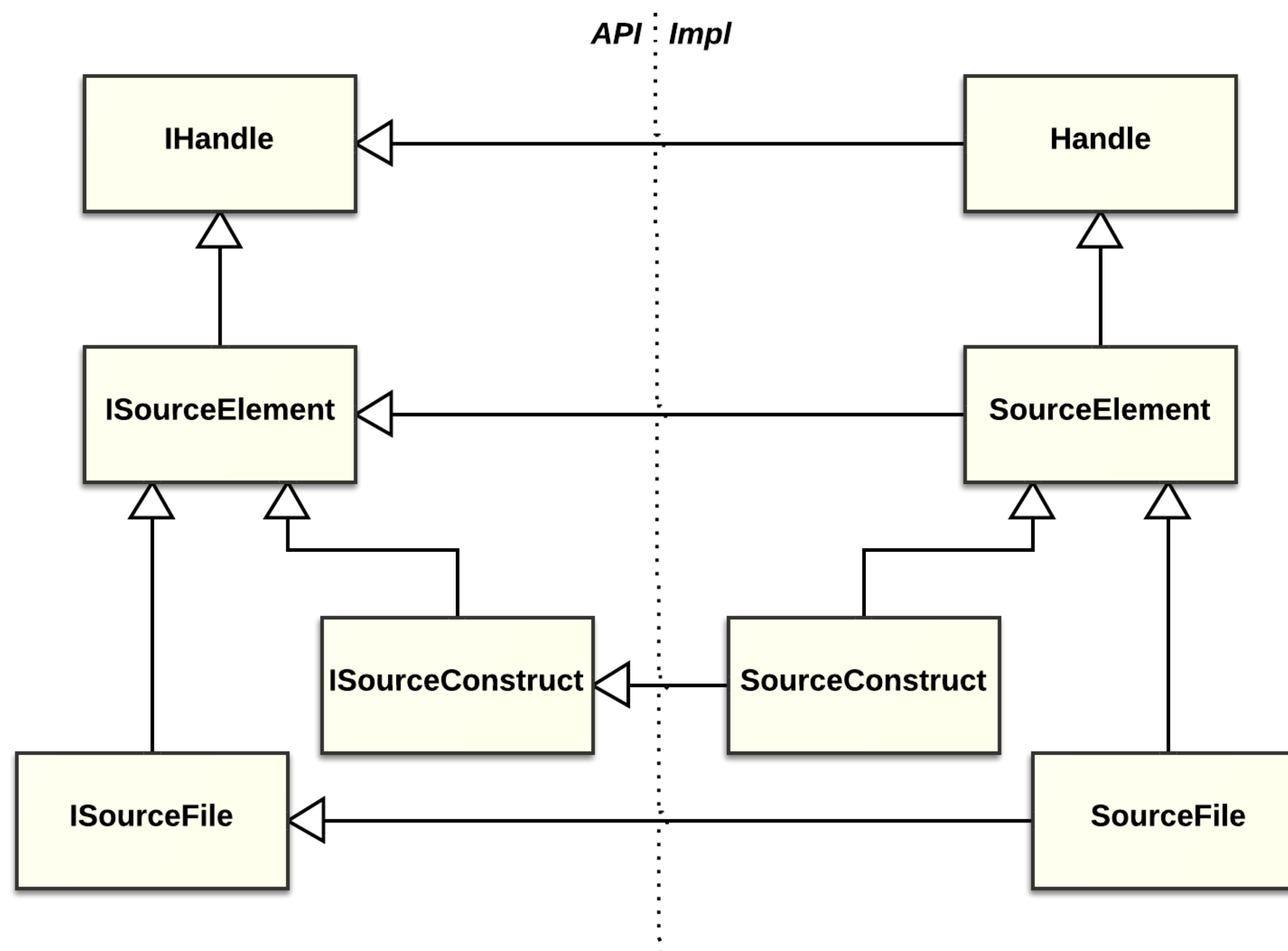
- Core framework
- Integration with other Eclipse projects
    - Xtext integration
- Common UI components
- Exemplary implementations

Core framework. Design motivation

- Make easier development of high-quality handle-based models for various languages
- Retain much of the flexibility associated with creation of such models “from scratch”
- Provide a uniform handle-based API to the models created with the framework
- More “a set of bricks” than “a framework”

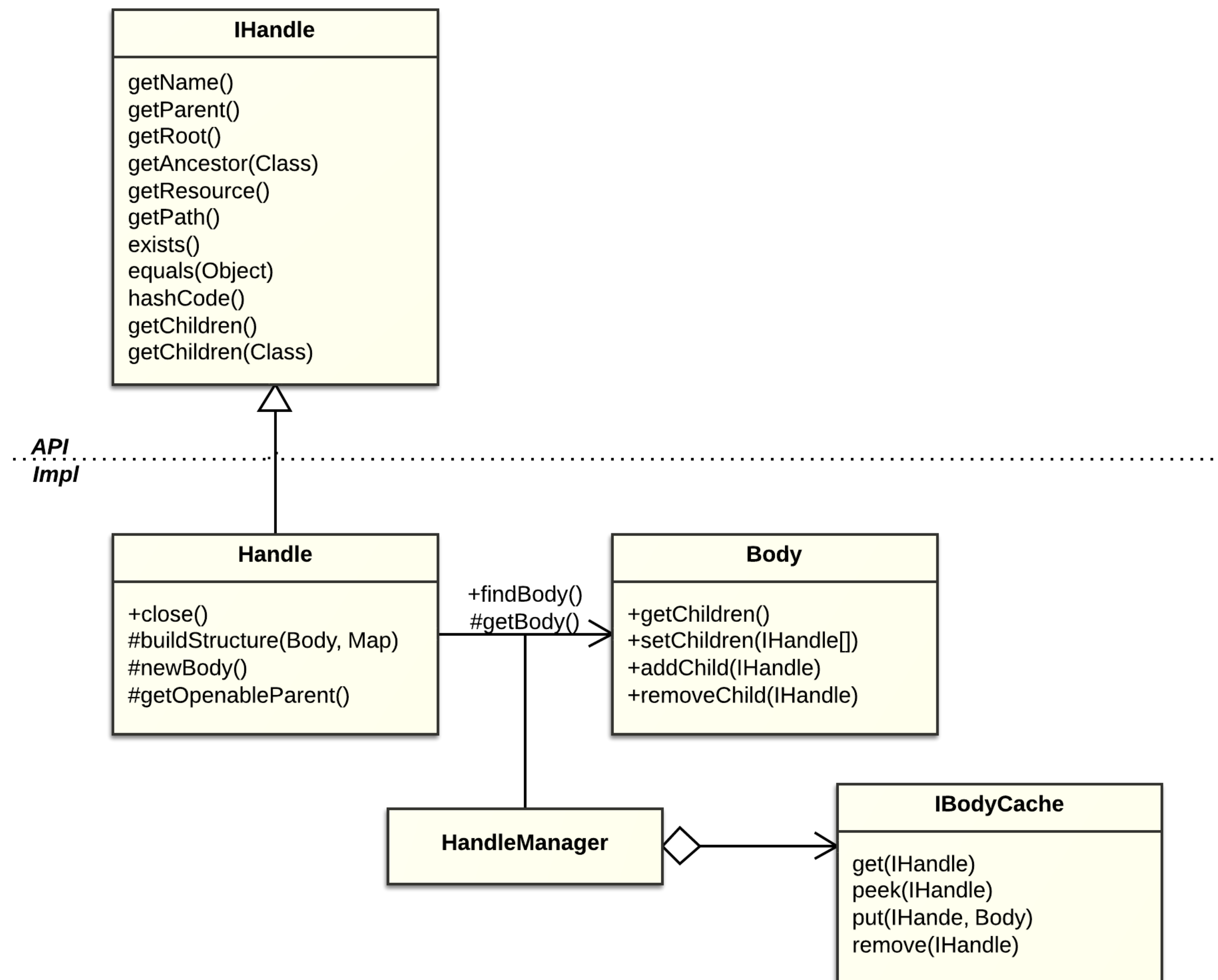
Core framework. Architectural overview

Inheritance hierarchy of the core model elements



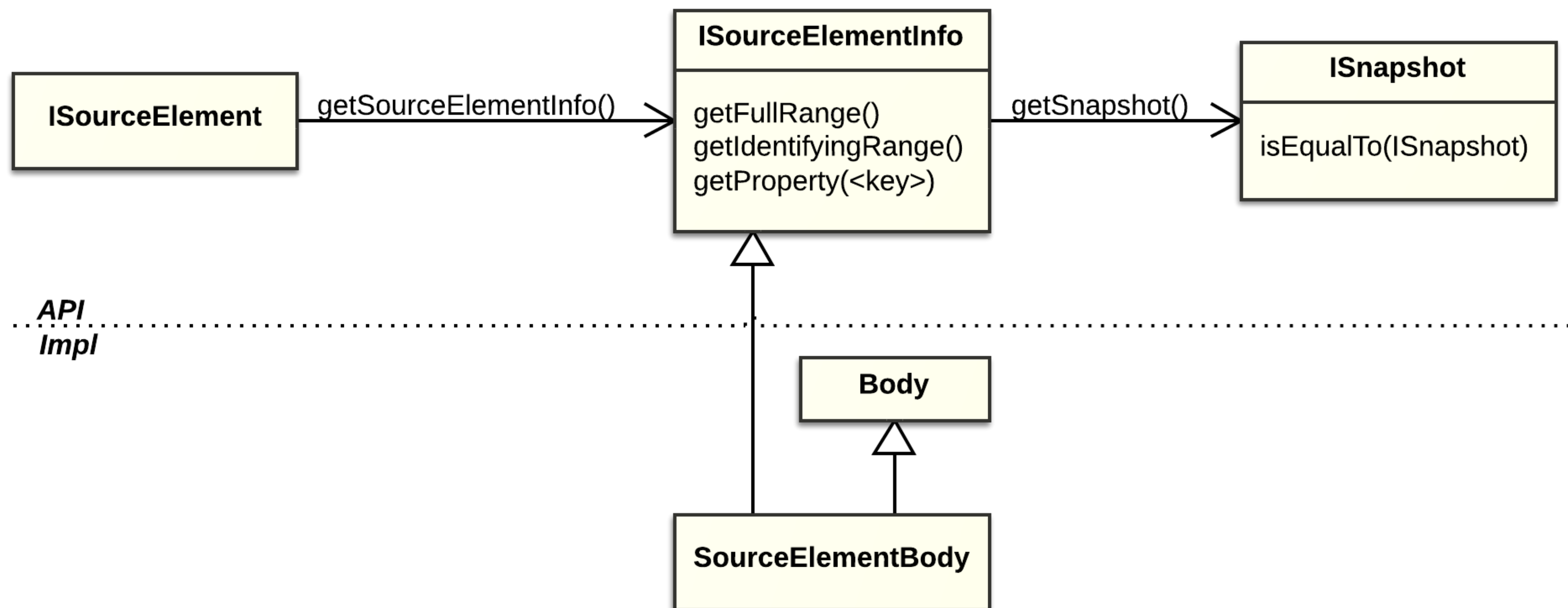
Core framework. Architectural overview

Generalized implementation of the ‘handle/body’ idiom for handle-based models



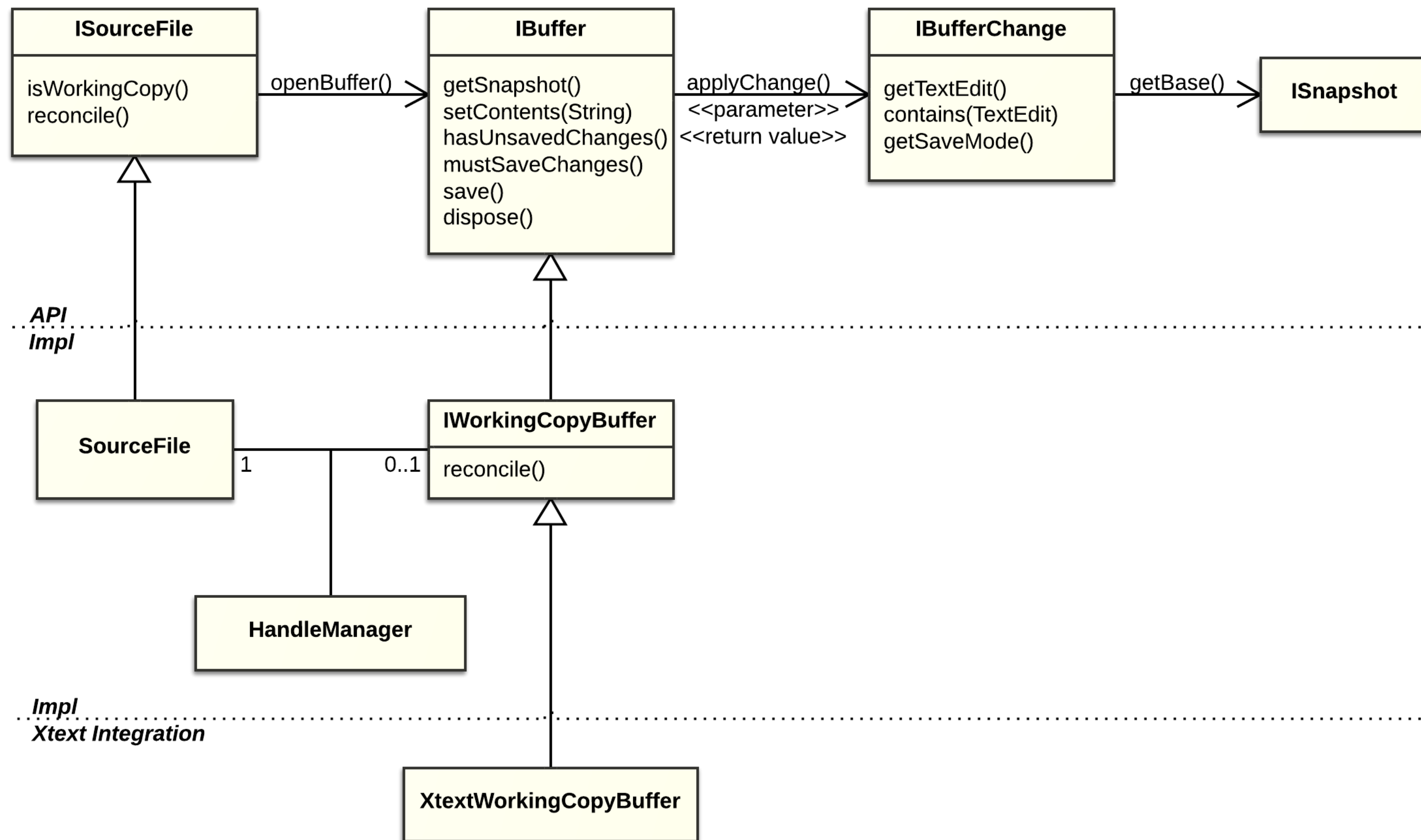
Core framework. Architectural overview

Source Element Info



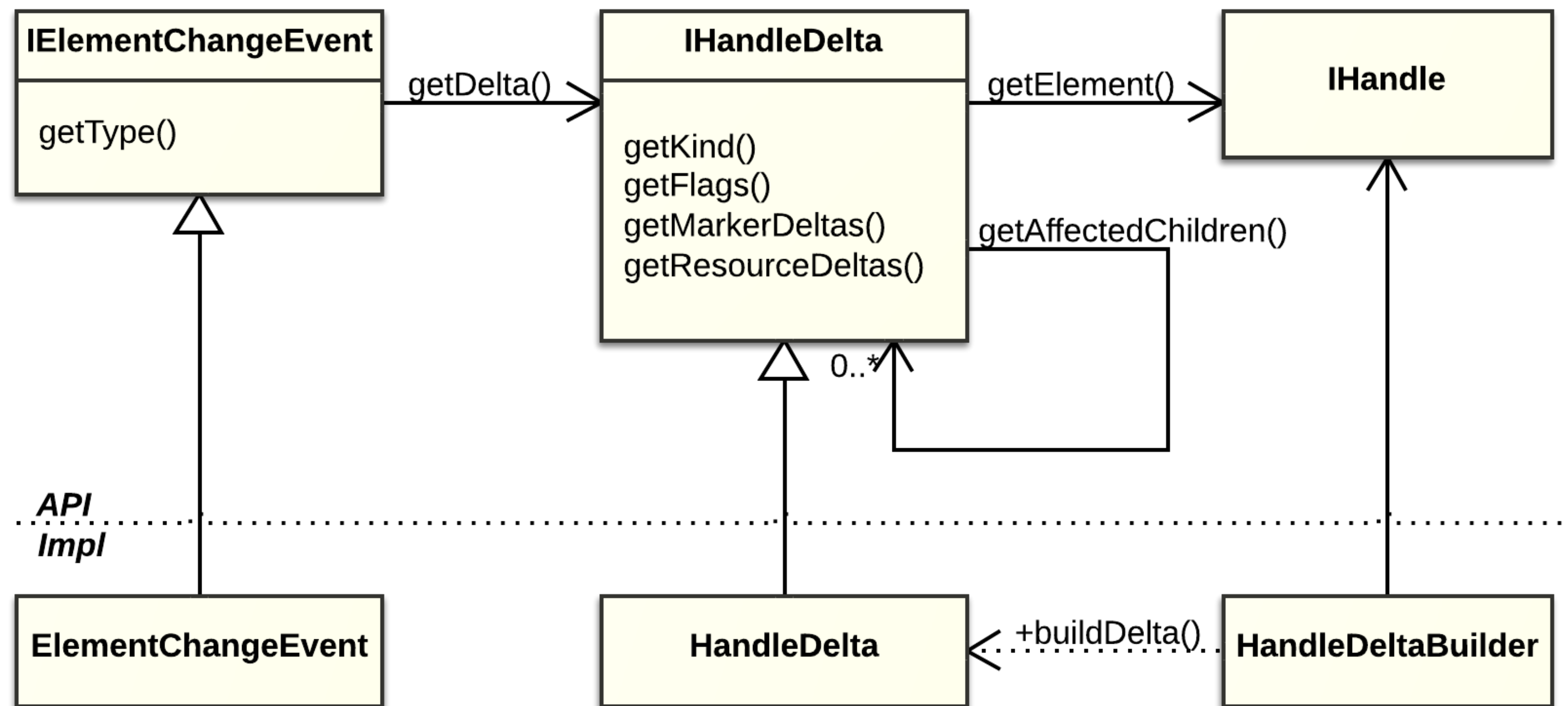
Core framework. Architectural overview

Source File



Core framework. Architectural overview

Generalized representation of change notifications for handle-based models



Steps to create a handle-based model with Handly

- Implement **the “handle” part** of the model
    - Inherit from corresponding system-provided interfaces and basic implementations for model elements
- Implement **the “body” part** of the model
    - Implement inherited abstract methods and supply a model-specific implementation of the body cache
- Implement **a resource change listener** for the model
    - Update the model when underlying workspace resources change
- Implement **integration** of the model **with the source file editor(s)**
    - **Already implemented for Xtext editor**. Just bind it in Xtext UI module

Basic example

- Made available under EPL in Handly Examples
- Demonstrates a Handly-based model for a simple Xtext-based language
- The language, called Foo, is contrived, but the model is full-featured

The Foo language

Xtext grammar

```
Module:
    vars += Var*
    defs += Def*
;

Var:
    'var' name=ID ';'
;

Def:
    'def' name=ID
    '(' (params+=ID)? (',' params+=ID)* ')'
    '{' '}'
;
```

Code sample

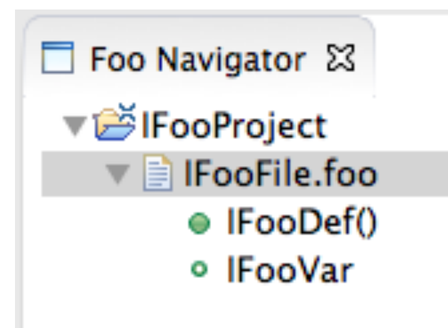
```
var x;
var y;
def f() {}
def f(x) {}
def f(x, y) {}
```

The Foo model

API : Impl

IHandle
IFooModel
IFooProject
ISourceElement
ISourceFile
ISourceConstruct
IFooFile
IFooVar
IFooDef

Handle
FooModel
FooProject
SourceElement
SourceFile
SourceConstruct
FooFile
FooVar
FooDef



Building model structure

```java
// FooModel.java

    @Override
    protected void buildStructure(Body body, Map<IHandle, Body> newElements)
        throws CoreException
    {
        IProject[] projects = workspace.getRoot().getProjects();
        List<IFooProject> fooProjects =
            new ArrayList<IFooProject>(projects.length);
        for (IProject project : projects)
        {
            if (project.isOpen() && project.hasNature(IFooProject.NATURE_ID))
            {
                fooProjects.add(new FooProject(this, project));
            }
        }
        body.setChildren(fooProjects.toArray(new IHandle[fooProjects.size()]));
    }
```

```java
// FooFile.java

    @Override
    protected void buildStructure(SourceElementBody body,
        Map<IHandle, Body> newElements, Object ast, String source)
    {
        XtextResource resource = (XtextResource)ast;
        IParseResult parseResult = resource.getParseResult();
        if (parseResult != null)
        {
            EObject root = parseResult.getRootASTElement();
            if (root instanceof Module)
            {
                FooFileStructureBuilder builder =
                    new FooFileStructureBuilder(newElements,
                        resource.getResourceServiceProvider());
                builder.buildStructure(this, body, (Module)root);
            }
        }
    }
```

Updating model structure (delta processor)

- Have to skip the tedious details
- The code is available in Handly Examples

Xtext integration

- It takes just a few bindings in Xtext UI module to connect a Handly-based model with Xtext editor
- It doesn't matter how dumb or smart the language is...
- The bindings are all the same

Handly bindings in Xtext UI module

```
@Override
public Class<? extends IReconciler> bindIReconciler()
{
    return HandlyXtextReconciler.class;
}

public Class<? extends XtextDocument> bindXtextDocument()
{
    return HandlyXtextDocument.class;
}

public Class<? extends DirtyStateEditorSupport> bindDirtyStateEditorSupport()
{
    return HandlyDirtyStateEditorSupport.class;
}

public void configureXtextEditorCallback(Binder binder)
{
    binder.bind(IXtextEditorCallback.class).annotatedWith(
        Names.named(HandlyXtextEditorCallback.class.getName())).to(
        HandlyXtextEditorCallback.class);
}

public Class<? extends ISourceFileFactory> bindISourceFileFactory()
{
    return FooFileFactory.class;
}
```

Common UI components

- None, currently
- Reserved for future work
- One idea is a common outline framework

Exemplary implementations

- Currently, only a basic example (just seen)
- An interesting example would be to “wrap” the Java Model into Handly API
- Looking for an initial adopter at Eclipse to collaborate on a practical tool built using Handly

«Simple things should be simple,
complex things should be possible»

–Alan Kay

Summing up

- Quite complex things are already possible, but simple things are not yet quite simple
- How can we fight complexity and find the joy of simplicity?

Complexity

«As size and complexity increase,
architectural design dominates materials»

–Alan Kay

- It's all about getting the core right
- Find abstractions that scale
- Beware of the generalization trap (over- and under-generalization)

Simplicity

«Everything should be made as simple as possible, but not simpler»

–Albert Einstein

- Development of a handle-based model for a language still remains an essentially complex task
- What about simplifying things further?
    - Layers on top of the Handly core framework for specific classes of languages? (e.g. for languages sharing the Java project structure)
    - Model-driven approach?
- Your ideas are welcome!

The community is the capacity

- Strong intent for a diverse, community-driven project
- Would like to draw community attention to the important problem area
- Looking forward to community feedback and participation

Now, that's just a vision and a design to start with

- Let's try to envision Handly together to make it really nice!
- You are very much invited to take part in the journey
- Your feedback is essential for setting directions
    - Bugzilla https://bugs.eclipse.org/bugs/enter_bug.cgi?product=Handly
    - Mailing list https://dev.eclipse.org/mailman/listinfo/handly-dev
    - Forum http://www.eclipse.org/forums/eclipse.handly

The Handly Team

Vladimir Piskarev

George Suaridze

<Your Name>

Thank you

Questions?