Adding on to Photran

Ralph Johnson
johnson@cs.uiuc.edu

PTP User-Developer Workshop
September 19, 2012

Photran

- Made to be expandable
- Still under development
- Thousands of users
- Good documentation
- Hundreds of people have learned how to add a refactoring to Photran

3 Secrets to Extending Photran

- Read the Photran Developer’s Guide
- Follow the Photran Developer’s Guide
- Practice the Photran Developer’s Guide

Facts about Photran

- Written in Java as an Eclipse plugin
- Actually a CDT plugin
  - Many features of Photran are “inherited” from CDT
  - Design of Photran depends on that of CDT
- An extension of Jeff Overbey’s “Rephraser Engine”

From CDT

- Most of UI
- Debugger
- Launcher
- Build (based on make)
- Error parser (interface from CDT)

Purpose of today

- Learn enough about internal representation to make a refactoring
  - Internal representation of Fortran programs
  - How to make a refactoring
- Learn how to connect to new Fortran compiler
  - How to make an error parser
- NOT learning UI

CDT program representation

- Project/Resource
  - Eclipse: represents project and all its files
- CDT Model
  - Provides a simplified view of program for UI, the tree view
- AST
  - Detailed description of a file; produced by parser

Photran Program Representation

- AST
  - Detailed description of a file; produced by parser
- VPG (Virtual Program Graph)
  - Represents entire program
  - Pretends to hold all ASTs, actually just stores summary
  - Produces Fortran version of CDT Model
  - Knows about project, resources

PhotranVPG

- PhotranVPG.getInstance() – singleton
- Has methods for finding parts of program
  - ArrayList<Definition> findAllModulesNamed(String name)
  - ArrayList<Definition> findSubprograms(String name, IFile file)
  - List<IFile> findFilesThatExportSubprogram(String subprogramName)

PhotranVPG

- public Definition getDefinitionFor(PhotranTokenRef tokenRef)
- public Type getTypeFor(PhotranTokenRef tokenRef)
- public Visibility getVisibilityFor(Definition def, ScopingNode visibilityInScope)

PhotranTokenRef

- Represents a token in a file; Editor converts a selection into a TokenRef.

- VPG maps TokenRef to nodes in AST.

- Type
- Definition – Type + array length & dimension
- Visibility – public/private
- ScopingNode – an AST node that represents a scope. The type hierarchy tells you which nodes are ScopingNodes; BlockConstruct, DerivedType, FunctionSubprogram, Interface, Module, …

2 Kinds of Refactoring

- FortranEditorRefactoring
  - Is given the text selection
  - Can change any file
- FortranResourceRefactoring
  - Is given a set of files
  - Will process all these files and can change any
- Choose appropriate superclass and then implement key methods.

Refactoring

- Four key methods
  - checkInitialConditions
    - Very simple checks even before asking for input
  - checkFinalConditions
    - Validate user input, perform more expensive checks
  - createChange
  - getName

User input

- Refactoring classes in org.eclipse.photran. Internal.core.refactoring

- UI classes needed by refactorings are in org.eclipse.photran.internal.ui.refactorings

Register the refactoring

- Add new refactoring as a menu item by putting a line in plugin.xml in org.eclipse.photran.ui.vpg

Adding a new Fortran compiler

- Compiler is invoked by make; just change makefile to use new compiler
- Photran must interpret error messages. Write a new ErrorParser and add it to a plugin for that compiler. See org.eclipse.photran.core.intel for an example.

ErrorParser

- IErrorParser has one method, processLine()
- Method usually contains a large number of nested Ifs that look for special cases.
- Errors reported by generateMarker method of ErrorParserManager, which collects errors.