Using Eclipse CDT/PTP for Static Analysis

Beth R. Tibbitts  IBM STG

tibbitts@us.ibm.com

"This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under its Agreement No. HR0011-07-9-0002"
Outline

• Basics of static analysis
• What CDT provides:
 rama
  • AST: how to inspect it; how to walk it
  • CODAN (Code Analysis) in CDT
• Additional info built by PTP/PLDT for analysis
  • Call graph (incl recursion)
  • Control flow graph
  • Data dependency (partial)

PLDT = Parallel Language Development Tools:
“the analysis part of PTP”
What is static analysis?

- **Static code analysis** is analysis of a computer program that is performed without actual execution - analysis performed on executing programs is known as **dynamic analysis**.
  - Usually performed on some intermediate representations of the source code.
  - Routinely done by compilers in order to generate and optimize object code.
- Motivation:
  - Deriving properties of execution behavior or program structure.
  - Various forms of analysis and refactoring.
  - *Lots more in JDT (Java Development Tools in Eclipse)*

*What can I find out about my C/C++ program?*
CDT Introspection Components

• Knowledge about the user’s source code is stored in the CDT’s DOM: Document Object Model

• Two components of DOM
  - DOM AST
    - Abstract Syntax Tree that stores detailed structural information about the code
  - Index
    - Built from the AST
    - Provides the ability to perform fast lookups by name on elements
    - Persistent index called the PDOM (persistent DOM)

Ref: EclipseCon 2007, “C/C++ Source Code Introspection Using the CDT”, Recoskie & Tibbitts
Using Eclipse CDT/PTP for Static Analysis

What is this information used for in CDT?

- Search
- Navigation
- Content Assist
- Call Hierarchy
- Type Hierarchy
- Include browsing
- Dependency scanning
- Syntax highlighting
- Refactoring
Abstract Syntax Tree: AST

- Maps C/C++ source code info onto a tree structure
  - A tree of nodes, all subclasses of org.eclipse.cdt.core.dom.ast.IASTNode
  - Nodes for: functions, names, declarations, arrays, expressions, statements/compound statements, etc.
  - Src file root: IASTTranslationUnit
    - Correlates to a source file: myfile.c
  - Tree structure eases analysis
  - Knows relationships (parent/child)
  - Easy traversal (ASTVisitor)
  - … etc
Existing CDT views that use structure include....

CDT Call Hierarchy view
Using Eclipse CDT/PTP for Static Analysis

CDT DOM AST View

- Graphical inspection of AST
Using Eclipse CDT/PTP for Static Analysis

CDT’s DOM AST View
- Installation

• Formerly available in CDT Testing feature:
  org.eclipse.cdt.ui.tests package

• But … in CDT 8.1/Juno … **DOM AST View is no longer built/installable with CDT** *(perhaps because its stability is questionable)*
  
  • But you can…

  • Run it in a runtime workspace from the CDT source projects!
    1. Check out the git repository of CDT source code
       ✤ http://wiki.eclipse.org/Getting_started_with_CDT_development
    2. Launch a runtime workspace (with CDT from your dev eclipse install – or source) with these two projects from your workspace:
       ✤ org.eclipse.cdt.core.tests
       ✤ org.eclipse.cdt.ui.tests

Still useful as a tool to understand CDT ASTs
Sample AST (Abstract Syntax Tree)
- tree structure representation of C program

```c
// walkast_edge.c
#include <stdio.h>

3 void edge(int a) {
4   int x,y;
5   if(a>0)
6     x=0;
7   else
8     x=1;
9   y=x;
 }

int foo(int bar){
   int z = bar;
   return z;
}
```

From EclipseCon 2008 reference
Using Eclipse CDT/PTP for Static Analysis

AST Samples

Several examples of using CDT’s AST and walking with the visitor pattern are in “Static Analysis in PTP with CDT” presented at EclipseCon 2008 (B. Tibbitts)

Code for tree walking is in sample plugin
   In PTP git repo: org.eclipse.ptp/tools/samples/
   Project org.eclipse.ptp.pldt.sampleCDTstaticAnalysis

![Image of Eclipse CDT/PTP interface showing AST and call graph options]
Relook at DOM AST View: see depth parsed
Using Eclipse CDT/PTP for Static Analysis

AST: what we do with it
PTP/PLDT provided structures

- Find Location of artifacts (API calls etc): MPI, OpenMP, UPC, OpenSHMEM, OpenACC with AST walking (not a simple text search)
- MPI Barrier Analysis: deadlock detection
Using Eclipse CDT/PTP for Static Analysis

Constructed by PTP’s PLDT:

- Call Graph
  - A partial Call Graph is also constructed by CDT Call Hierarchy view
- Control Flow Graph
- Dependency Graph (Defined/Use Chain: partial)

In order to do:
- MPI Barrier Analysis: detect deadlocks; find concurrently executed statements

Caveats:
- C only (not C++)
- No UI - structures used for analysis only
Control Flow Graph

- A control flow graph (CFG) is a representation of all paths that might be traversed through a program during its execution. Each node in the graph represents a basic block, i.e. a straight-line piece of code with a single point of entry and a single point of exit.

- A Statement Level CFG is a CFG with individual statements instead of larger basic blocks.
  - PLDT builds a statement level CFG as described here
CDT’s Codan

• Codan (Code Analysis) - lightweight static analysis framework in CDT that allows pluggable "checkers" which can perform real time analysis on the code to find common defects, violation of policies, etc.
  - Finds errors as you type
  - Quick fixes often available

• Integrate an external code checker into Eclipse CDT

• External checker cppcheck integrated with codan:
  http://alexruiz.developerblogs.com/?p=2231
Using Eclipse CDT/PTP for Static Analysis

Codan

- Finds errors as you type
- Provides quick fixes
References

• Parallel Tools Platform  eclipse.org/ptp
• C/C++ Development Tools  eclipse.org/cdt

• CDT’s CODAN (Code Analysis)

• Static Analysis with CDT in PTP – EclipseCon 2008
  • [http://www.eclipsecon.org/2008/?page=sub/&id=373](http://www.eclipsecon.org/2008/?page=sub/&id=373)
Summary

• CDT has the basics for Static Analysis, including AST (Abstract Syntax Tree)
• Other useful structures are built by PTP’s PLDT
  - Call Graph, Control Flow Graph, Dependency Graph, etc.
  - These graphs make analysis more straightforward
• CDT provides Code Analysis (Codan) for a framework to provide pluggable static syntax checkers, etc.
  - Quickly notify user of common errors, policy violations, etc.