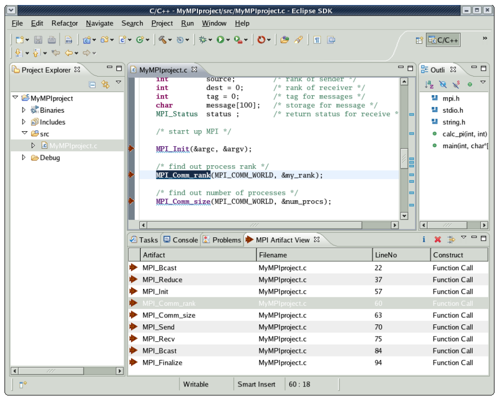
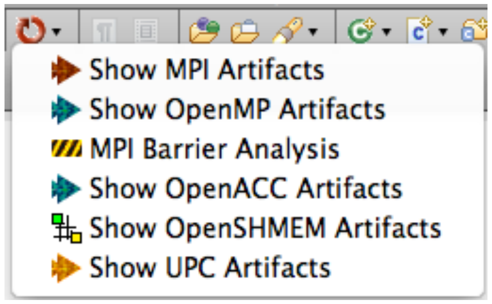
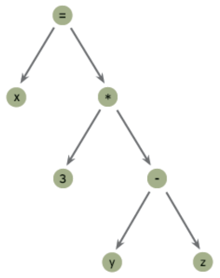


```
    int         source;         /* rank of sender */
    int         dest = 0;       /* rank of receiver */
    int         tag = 0;        /* tag for messages */
    char        message[100];   /* storage for message */
    MPI_Status  status ;        /* return status for receive *

    /* start up MPI */

    MPI_Init(&argc, &argv);

    /* find out process rank */
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    /* find out number of processes */
    MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
```

Artifact	Filename	LineNo	Construct
MPI_Bcast	MyMPIproject.c	22	Function Call
MPI_Reduce	MyMPIproject.c	37	Function Call
MPI_Init	MyMPIproject.c	57	Function Call
MPI_Comm_rank	MyMPIproject.c	60	Function Call
MPI_Comm_size	MyMPIproject.c	63	Function Call
MPI_Send	MyMPIproject.c	70	Function Call
MPI_Recv	MyMPIproject.c	75	Function Call
MPI_Bcast	MyMPIproject.c	84	Function Call
MPI_Finalize	MyMPIproject.c	94	Function Call

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 :
  - 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?

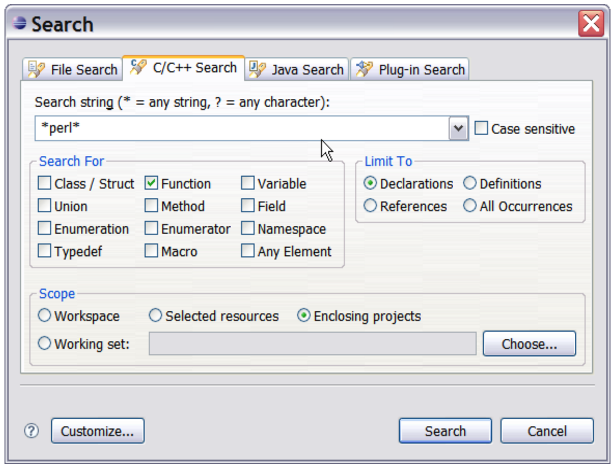
CDT Introspection Components

- Knowledge about the user's source code is stored in the CDT's DOM: Document Object Model
- Two components of DOM
  - DOM **AST** ⟵ concentrate here
    - 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

What is this information used for in CDT?

- Search
- Navigation
- Content Assist
- Call Hierarchy
- Type Hierarchy
- Include browsing
- Dependency scanning
- Syntax highlighting
- Refactoring


Search

File Search | C/C++ Search | Java Search | Plug-in Search

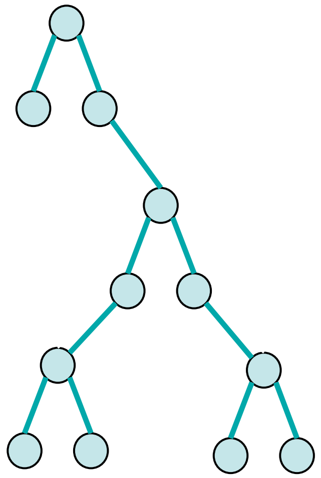
Search string (* = any string, ? = any character): *perl*

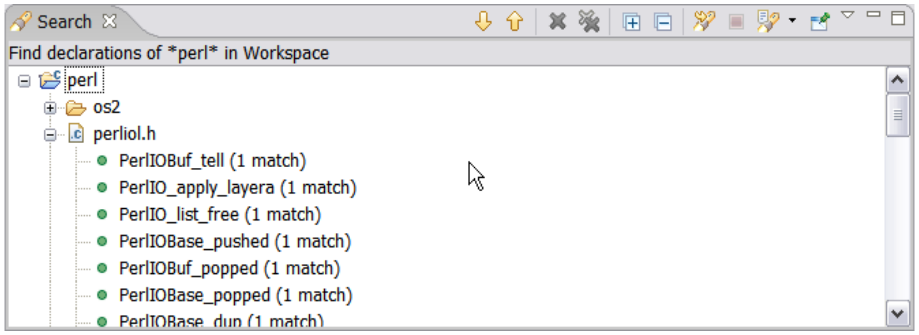
[ ] Case sensitive

Search For: [ ] Class / Struct [✓ Function] [ ] Variable [ ] Union [ ] Method [ ] Field [ ] Enumeration [ ] Enumerator [ ] Namespace [ ] Typedef [ ] Macro [ ] Any Element

Limit To: [✓ Declarations] Definitions References All Occurrences

Scope: Workspace Selected resources [✓ Enclosing projects] Working set:



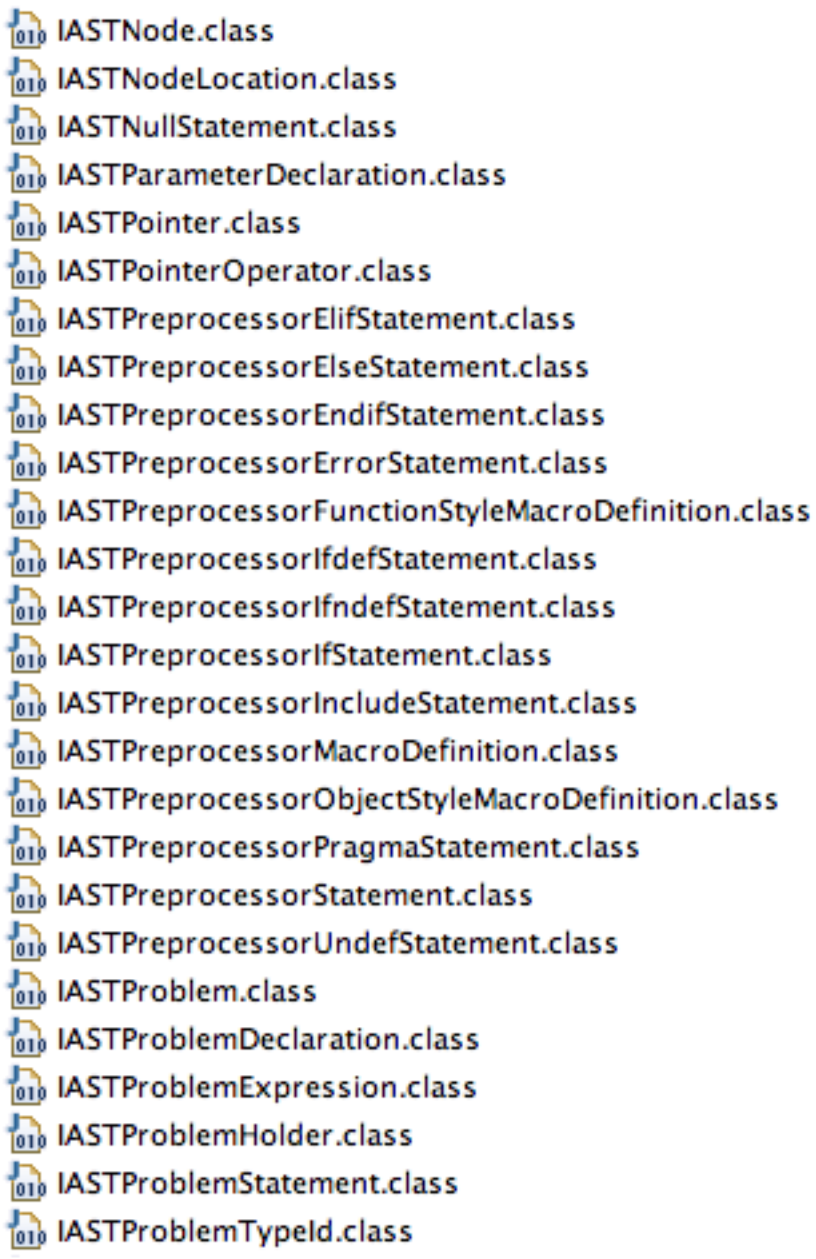

Search

Find declarations of *perl* in Workspace

- perl
  - os2
  - perliol.h
    - PerlIOBuf_tell (1 match)
    - PerlIO_apply_layera (1 match)
    - PerlIO_list_free (1 match)
    - PerlIOBase_pushed (1 match)
    - PerlIOBuf_popped (1 match)
    - PerlIOBase_popped (1 match)
    - PerlIOBase_dup (1 match)

Abstract Syntax Tree: AST

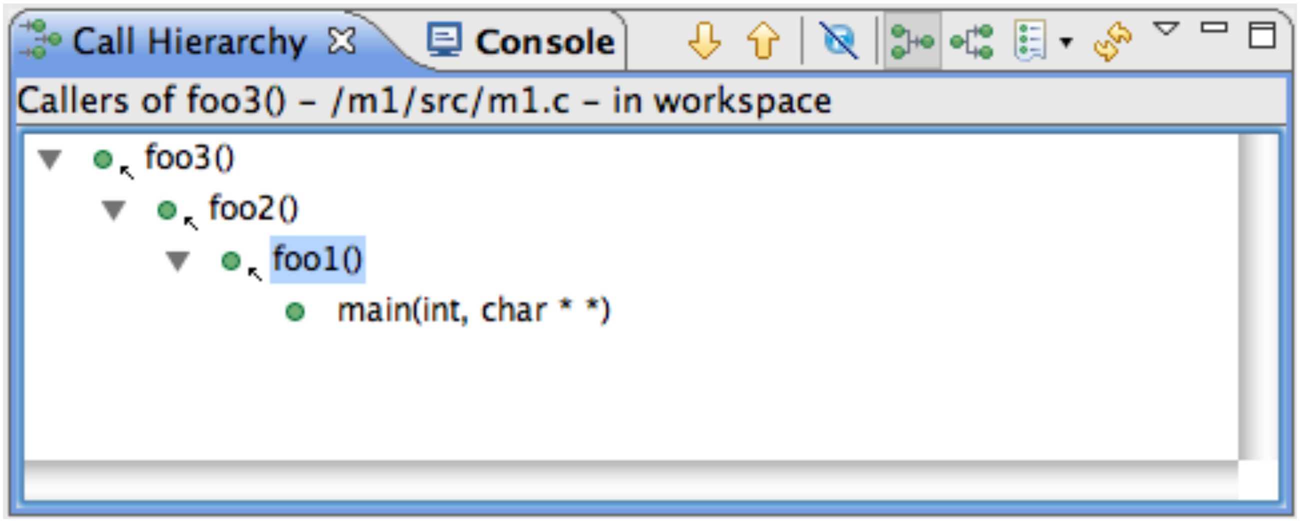
- Maps C/C++ source code info onto a tree structure
  - A tree of nodes, all subclasses of
    - org.eclipse.cdt.core.dom.ast.IASTNode
  - Nodes for: functions, names, declarations, arrays, expressions, statements/compound statements, etc.
  - Src file root: IASTTranslationUnit
    - Correlates to a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



IASTNode.class
IASTNodeLocation.class
IASTNullStatement.class
IASTParameterDeclaration.class
IASTPointer.class
IASTPointerOperator.class
IASTPreprocessorElifStatement.class
IASTPreprocessorElseStatement.class
IASTPreprocessorEndifStatement.class
IASTPreprocessorErrorStatement.class
IASTPreprocessorFunctionStyleMacroDefinition.class
IASTPreprocessorIfdefStatement.class
IASTPreprocessorIfndefStatement.class
IASTPreprocessorIfStatement.class
IASTPreprocessorIncludeStatement.class
IASTPreprocessorMacroDefinition.class
IASTPreprocessorObjectStyleMacroDefinition.class
IASTPreprocessorPragmaStatement.class
IASTPreprocessorStatement.class
IASTPreprocessorUndefStatement.class
IASTProblem.class
IASTProblemDeclaration.class
IASTProblemExpression.class
IASTProblemHolder.class
IASTProblemStatement.class
IASTProblemTypeId.class

Existing CDT views that use structure include….

CDT Call Hierarchy view

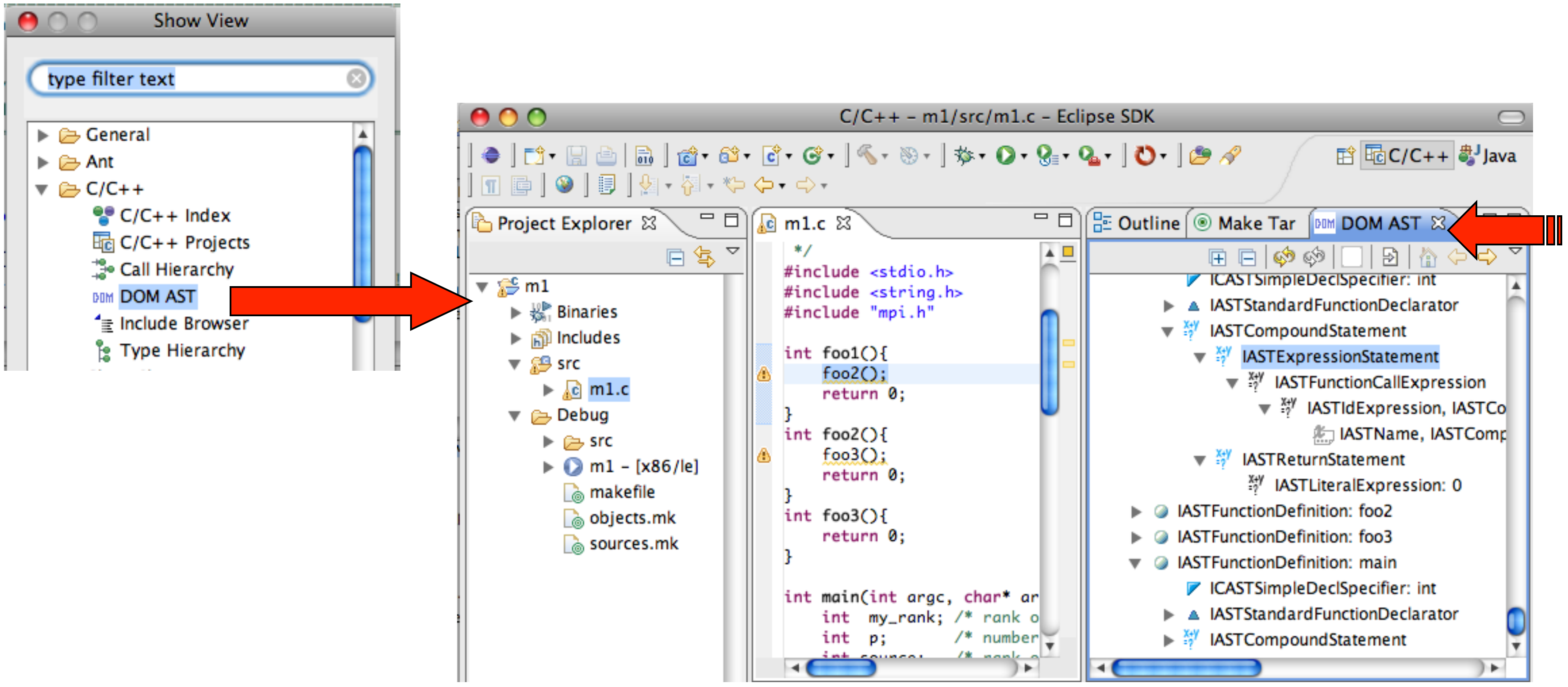


Callers of foo3() – /m1/src/m1.c – in workspace

```
▼ foo3()
    ▼ foo2()
        ▼ foo1()
              main(int, char * *)
```

CDT DOM AST View

- Graphical inspection of AST



```
*/
#include <stdio.h>
#include <string.h>
#include "mpi.h"

int foo1(){
    foo2();
    return 0;
}
int foo2(){
    foo3();
    return 0;
}
int foo3(){
    return 0;
}

int main(int argc, char* ar
    int  my_rank; /* rank o
    int  p;       /* number
```

```
    ICASTSimpleDeclSpecifier: int
  ▶ IASTStandardFunctionDeclarator
  ▼ IASTCompoundStatement
    ▼ IASTExpressionStatement
      ▼ IASTFunctionCallExpression
        ▼ IASTIdExpression, IASTCo
            IASTName, IASTComp
    ▼ IASTReturnStatement
        IASTLiteralExpression: 0
▶ IASTFunctionDefinition: foo2
▶ IASTFunctionDefinition: foo3
▼ IASTFunctionDefinition: main
    ICASTSimpleDeclSpecifier: int
  ▶ IASTStandardFunctionDeclarator
  ▶ IASTCompoundStatement
```

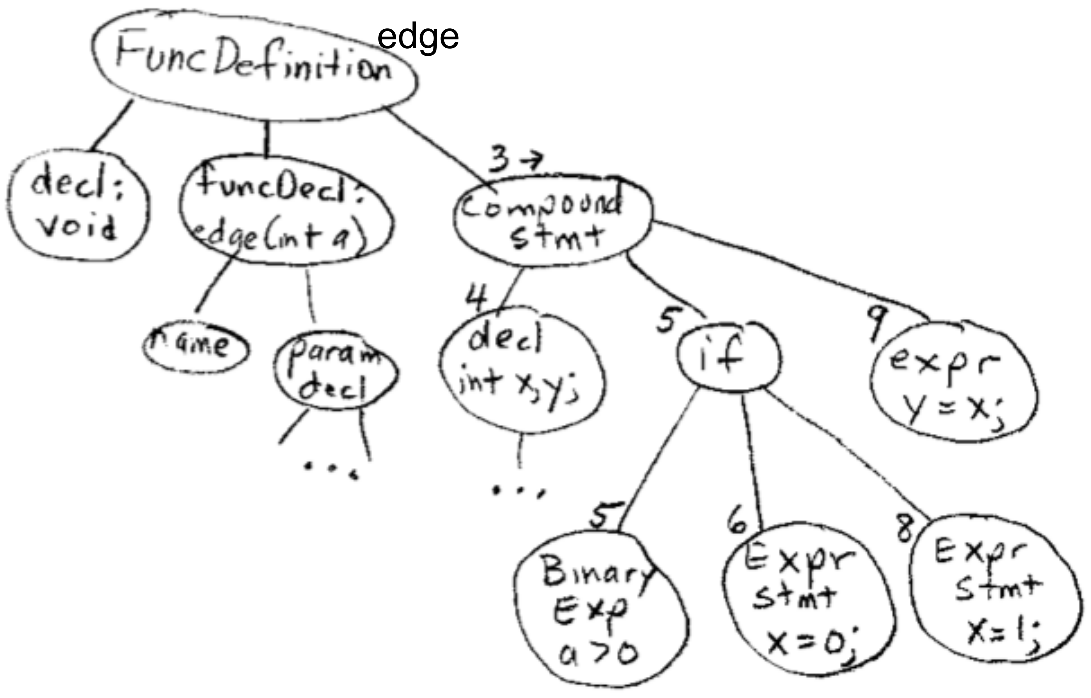
CDT's DOM AST View
- Installation

- 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

```
// 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;
  }
```

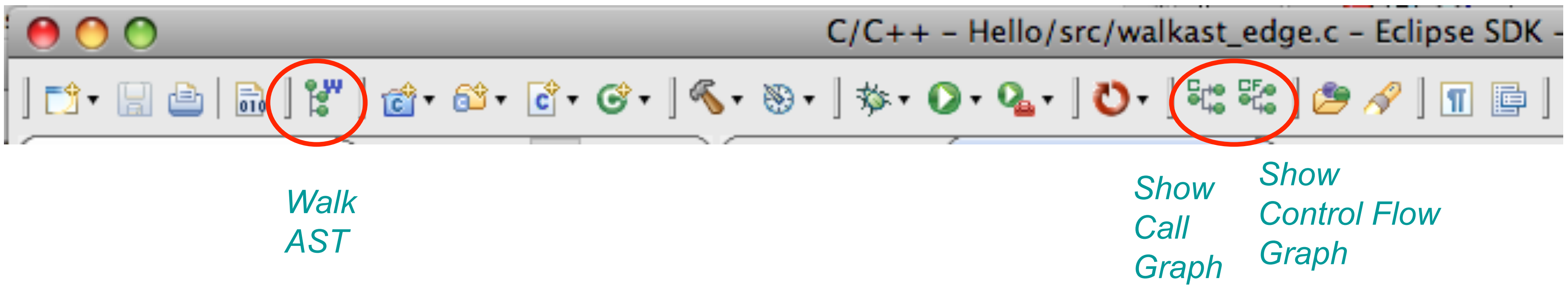


From EclipseCon 2008 reference

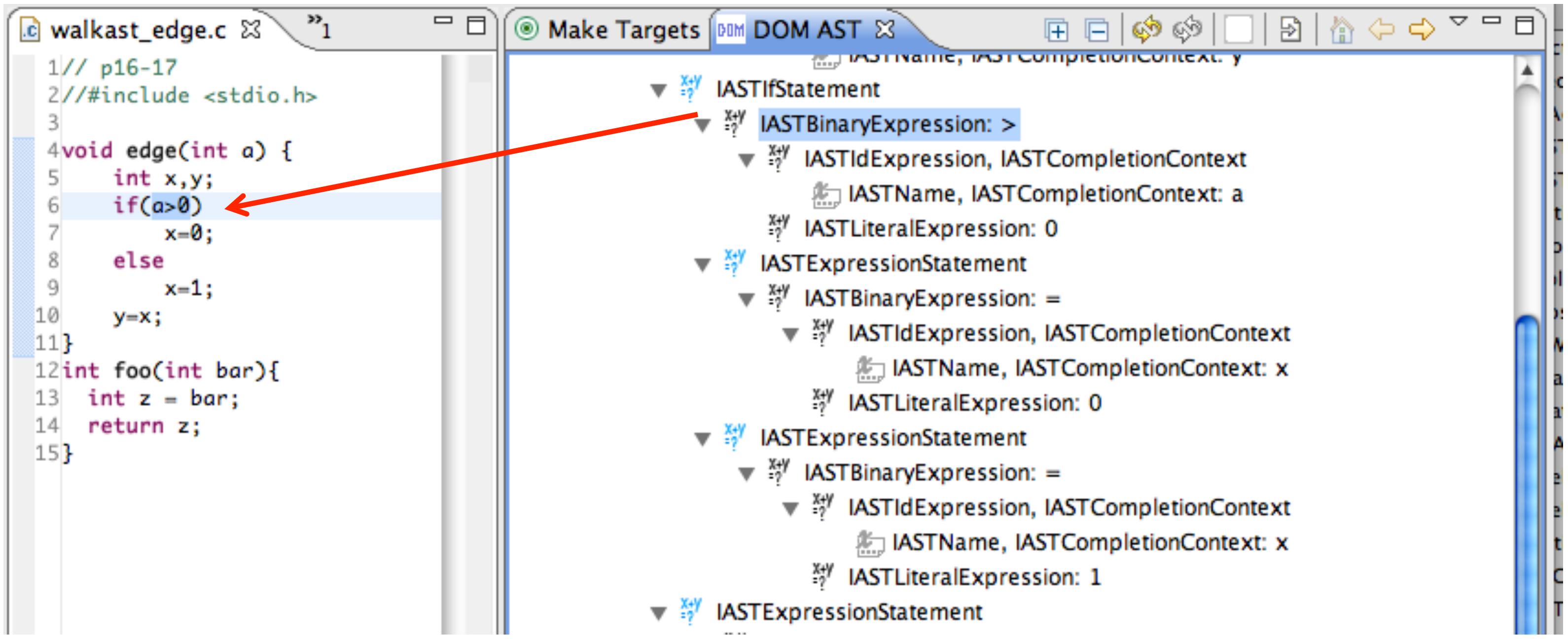
AST Samples

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



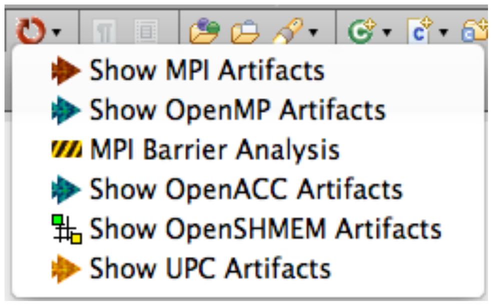
Relook at DOM AST View: see depth parsed



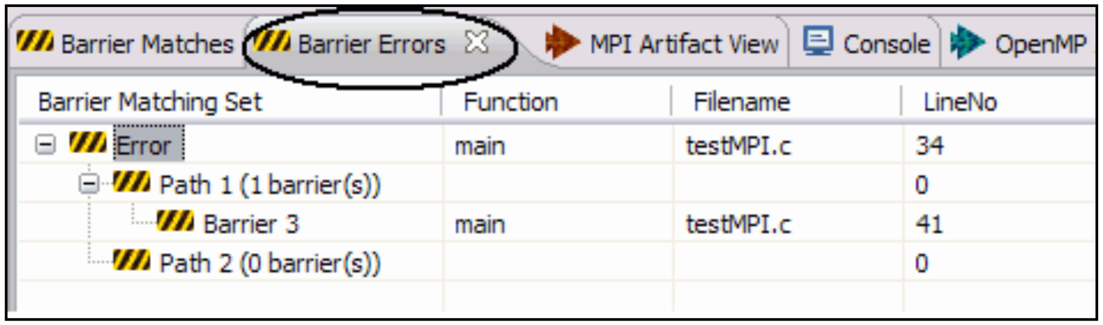
```
// p16-17
//#include <stdio.h>

void edge(int a) {
    int x,y;
    if(a>0)
        x=0;
    else
        x=1;
    y=x;
}
int foo(int bar){
  int z = bar;
  return z;
}
```

```
IASTIfStatement
  IASTBinaryExpression: >
    IASTIdExpression, IASTCompletionContext
      IASTName, IASTCompletionContext: a
    IASTLiteralExpression: 0
  IASTExpressionStatement
    IASTBinaryExpression: =
      IASTIdExpression, IASTCompletionContext
        IASTName, IASTCompletionContext: x
      IASTLiteralExpression: 0
  IASTExpressionStatement
    IASTBinaryExpression: =
      IASTIdExpression, IASTCompletionContext
        IASTName, IASTCompletionContext: x
      IASTLiteralExpression: 1
IASTExpressionStatement
```

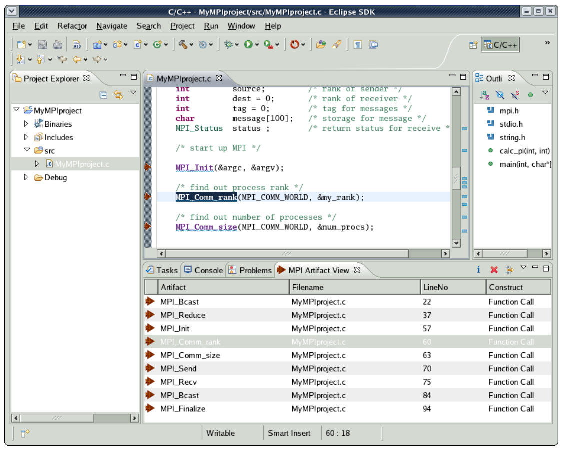
AST: what we do with it
PTP/PLDT provided structures



- 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



Barrier Matching Set	Function	Filename	LineNo
Error	main	testMPI.c	34
Path 1 (1 barrier(s))			0
Barrier 3	main	testMPI.c	41
Path 2 (0 barrier(s))			0



```
int           source;          /* rank of sender */
int           dest = 0;        /* rank of receiver */
int           tag = 0;         /* tag for messages */
char          message[100];    /* storage for message */
MPI_Status    status ;         /* return status for receive */

/* start up MPI */

MPI_Init(&argc, &argv);

/* find out process rank */
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

/* find out number of processes */
MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
```

Artifact	Filename	LineNo	Construct
MPI_Bcast	MyMPIproject.c	22	Function Call
MPI_Reduce	MyMPIproject.c	37	Function Call
MPI_Init	MyMPIproject.c	57	Function Call
MPI_Comm_rank	MyMPIproject.c	60	Function Call
MPI_Comm_size	MyMPIproject.c	63	Function Call
MPI_Send	MyMPIproject.c	70	Function Call
MPI_Recv	MyMPIproject.c	75	Function Call
MPI_Bcast	MyMPIproject.c	84	Function Call
MPI_Finalize	MyMPIproject.c	94	Function Call

Writable | Smart Insert | 60 : 18

Constructed by PTP's PLDT:

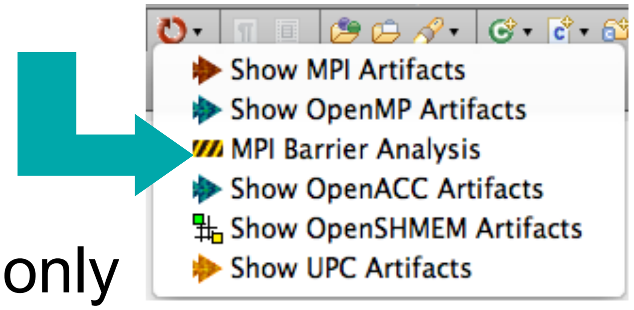
- Call Graph
  - A partial Call Graph is also constructed by CDT Call Hierarchy view
- Control Flow Graph
- Dependency Graph (Defined/Use Chain: partial)

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

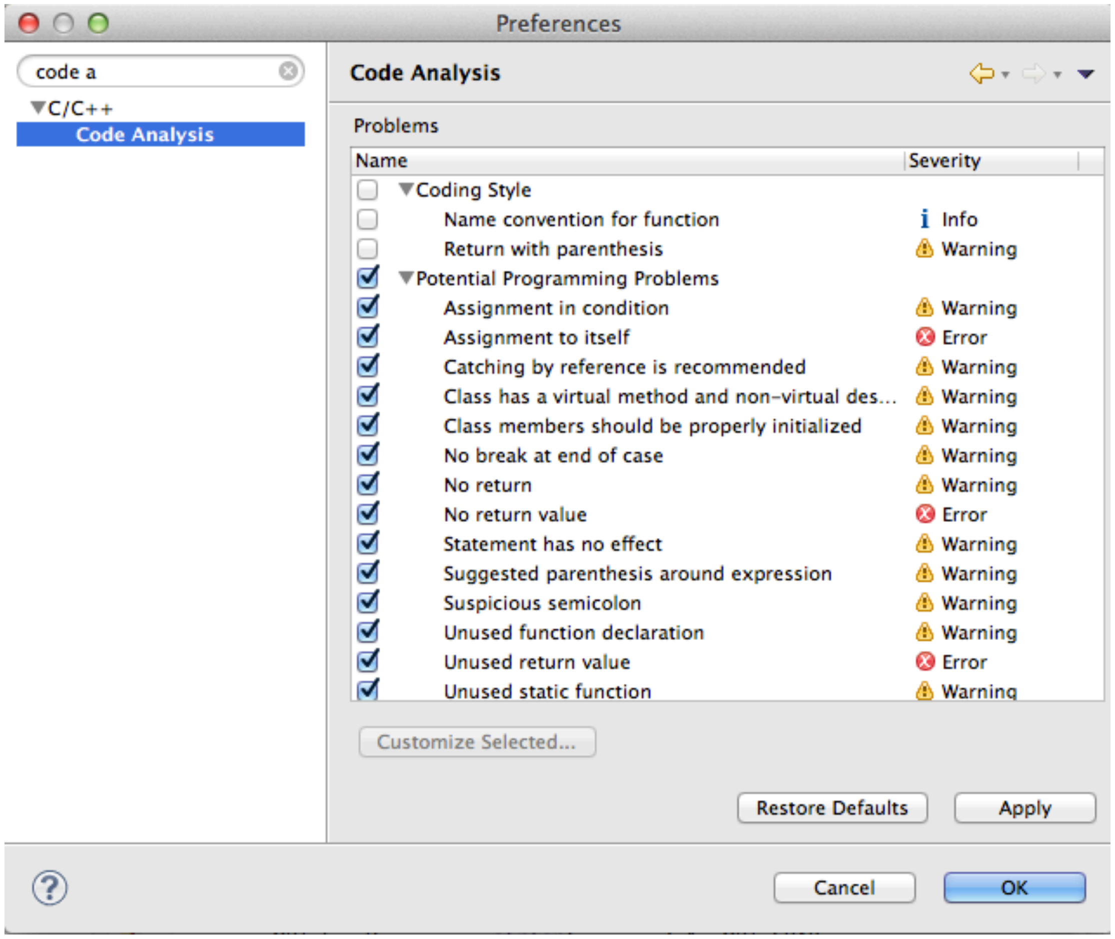
- A **control flow graph (CFG)** is a representation of all paths that might be traversed through a program during its execution. Each node in the graph represents a basic block, i.e. a straight-line piece of code with a single point of entry and a single point of exit
- **A Statement Level CFG** is 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** **https://www.ibm.com/developerworks/java/library/j-codan/**
- External checker cppcheck integrated with codan: http://alexruiz.developerblogs.com/?p=2231

Codan

- Finds errors as you type
- Provides quick fixes



Code Analysis

Problems

	Name	Severity
☐	Coding Style	
☐	Name convention for function	Info
☐	Return with parenthesis	Warning
☑	Potential Programming Problems	
☑	Assignment in condition	Warning
☑	Assignment to itself	Error
☑	Catching by reference is recommended	Warning
☑	Class has a virtual method and non-virtual des...	Warning
☑	Class members should be properly initialized	Warning
☑	No break at end of case	Warning
☑	No return	Warning
☑	No return value	Error
☑	Statement has no effect	Warning
☑	Suggested parenthesis around expression	Warning
☑	Suspicious semicolon	Warning
☑	Unused function declaration	Warning
☑	Unused return value	Error
☑	Unused static function	Warning

References

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

- CDT's CODAN (Code Analysis)
  - http://wiki.eclipse.org/CDT/designs/StaticAnalysis
  - Codan: a C/C++ Static Analysis Framework for CDT – http://www.slideshare.net/laskava/eclipse-con2011-v11
  - Integrate an external code checker into **Eclipse CDT** **https://www.ibm.com/developerworks/java/library/j-codan/**
- Static Analysis with CDT in PTP – EclipseCon 2008
  - 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.