Eclipse CDT code analysis and unit testing

Shaun D’Souza
Agenda

• Compilers and translators
• Eclipse Plugin
• CDT
• AST
• Visitor
Compilers and translators

- Compilers translate information from one representation to another.
- Most commonly, the information is a program.
- Compilers translate from high-level source code to low-level code.
- Translators transform representations at the same level of abstraction.
Stages of Compilation

- AST represents the structure of the source code
- Parser turns flat text into a tree
- AST is a kind of a program intermediate form (IR).

Source code

Lexical analysis

Syntax analysis

Token stream

Abstract Syntax Tree

Semantic analysis
Eclipse

- IDE – Integrated Development Environment
- Support several programming languages
  - C/C++, Java, PHP, XML, HTML
- Multi-platform
  - Windows, Linux
- Supports plug-in functionality
- Open source
- Alternatives – NetBeans, MS Visual Studio, g++
CDT (C/C++ Development Tooling)

- Set of plug-in for developing C/C++ applications
- Edit/compile/debug/run
- CDT parses and analyses the code
- CDT compiles the code into an index file
- Index stores information
  - Identifier Bindings
  - Location source file, offset
  - Macros
  - Include files
- JDT (Java Development Tools)
- PDE (Plug-in Development Environment)
CDT core

- Preprocessor
  - Extra stage between the lexer and parser
  - Converts text into token stream
- Parsers (C and C++)
  - Converts token stream into an AST
- AST
  - Visitor API
- AST Rewrite API
  - Implement refactoring
- Semantic analysis
  - Resolve identifiers
- Indexer
  - Update index file by processing the AST
- Index API
  - Index based tool to query the index
CDT indexer

• Indexer
  • Code traversal and searching

• Refactoring
  • Rename function / method

• Indexer used with static code analysis
AST

- AST represents the structure of the source code
- CDT functionality is based on the AST
  - ~90 classes for C++
- Implement common interfaces
  - Some algorithms depend on specific type – semantic analysis
  - Some algorithms depend on interfaces – outline view
Access to C-model and C-index

• C-Model: ITranslationUnit for a workspace file

```java
ITranslationUnit tu = (ITranslationUnit) CoreModel.getDefault().create(file);
```

• C-Model: ITranslationUnit for file in the editor

```java
IEditorPart e = PlatformUI.getWorkbench().getActiveWorkbenchWindow().getActivePage().getActiveEditor();
ITranslationUnit tu = (ITranslationUnit) CDTUITools.getEditorInputCElement(editor.getEditorInput());
```

• C-Index: IIndex for project

```java
IIndex index = CCorePlugin.getIndexManager().getIndex(project);
```
Binding and types

• Index contains information
  • Include directives and macro definitions
  • Non-local declarations
  • References to macros and non-local declarations
  • File location for each include, macro definition, declaration and reference
  • Binding for each name

• Completely represent C/C++ entities
  • Type of a variable, return type and parameters for a function
  • Fields of a composite type, owner of a field
Creating AST

- Complete AST: IASTTranslationUnit for a workspace file
  ITranslationUnit tu = ...;
  IASTTranslationUnit ast = tu.getAST();

- Index-based AST: IASTTranslationUnit for a workspace file
  IIndex index = ...;
  ITranslationUnit tu = ...;
  index.acquireReadLock(); // we need a read-lock on the index
  try {
    ast = tu.getAST(index, ITranslationUnit.AST_SKIP_INDEXED_HEADERS);
  } finally {
    index.releaseReadLock();
    ast = null; // don't use the ast after releasing the read-lock
  }
Visitor pattern

• Design pattern used to traverse the AST
• Standard easy-to-use API for processing the AST
• Create a visitor object
  • Extends ASTVisitor
  • Implement overloaded visit(IASTXXX) methods for each node type
• Each node class has an accept(ASTVisitor) method (defined in IASTNode)
  • Calls visit(this)
private void walkITU_AST(ITranslationUnit tu) throws CoreException {
    System.out.println("AST visitor for " + tu.getElementName());
    IASTTranslationUnit ast = tu.getAST();
    ast.accept(new ASTPrinter());
}

class ASTPrinter extends ASTVisitor{
    ASTPrinter(){
        this.shouldVisitStatements = true;
        this.shouldVisitDeclarations = true;
    }
    public int visit(IASTStatement stmt) {
        System.out.println("Visiting stmt: " + stmt.getRawSignature());
        return PROCESS_CONTINUE;
    }
    public int visit(IASTDeclaration decl) {
        System.out.println("Visiting decl: " + decl.getRawSignature());
        return PROCESS_CONTINUE;
    }
}

Tree.accept(visitor)
#include <iostream>

int foo() {
    return 0;
}

class a {
    a() {}
    void b() {}
};

int main() {
    std::cout << "foo" << std::endl; // prints foo
    foo();
    a b1;
    return 0;
}
FakeStorageSCSI_DiscoveryAlgorithm::FakeStorageSCSI_DiscoveryAlgorithm()

: StorageSCSI_DiscoveryAlgorithm()

, fake_run("FakeStorageSCSI_DiscoveryAlgorithm::run")

, fake_associate("FakeStorageSCSI_DiscoveryAlgorithm::associate")

, fake_getDuplicatedHardDriveList("FakeStorageSCSI_DiscoveryAlgorithm::getDuplicatedHardDriveList")

, fake_addUniqueHardDrive("FakeStorageSCSI_DiscoveryAlgorithm::addUniqueHardDrive")

, fake_isDuplicateBackplane("FakeStorageSCSI_DiscoveryAlgorithm::isDuplicateBackplane")

}

void FakeStorageSCSI_DiscoveryAlgorithm::verifyFakeMethodUsage(const std::string& testCondition)
{
    TestUtility::verifyFakeMethodUsage(fake_run, testCondition);
    TestUtility::verifyFakeMethodUsage(fake_associate, testCondition);
    TestUtility::verifyFakeMethodUsage(fake_getDuplicatedHardDriveList, testCondition);
    TestUtility::verifyFakeMethodUsage(fake_addUniqueHardDrive, testCondition);
    TestUtility::verifyFakeMethodUsage(fake_isDuplicateBackplane, testCondition);
}

void FakeStorageSCSI_DiscoveryAlgorithm::run(UI_Facade& uiFacade)
{
    return fake_run(uiFacade);
}
StorageSCSI_DiscoveryAlgorithmData

StorageSCSI_DiscoveryAlgorithm_data()
    : fakeDeviceReporter()
    , fakeDiscoveryRepository()
    , fakeIoConnectionOperations()
    , fakeTransportFactory()
    , fakeDiscoveryOperationsFactory()
    , fakeDiscoveredDeviceOperationsFactory()
    , fakeFusionIO_AcceleratorFactory()
    , fakePciOperationsFactoryPtr( new FakePCI_OperationsFactory() )
    , fakeFileSystemOperations()
    , fakeSmbiosOperationsPtr( new FakeSMBIOS_Operations() )
    , fakeIloOperationsPtr( new iLO::Fake_iLO_Operations() )
    , fakeTimeOperationsPtr( new FakeTimeOperations() )
    , failureEventStatus( FakeEvt::failure )
    , goodEventStatus() {}
References

- CDT Project home page - [http://eclipse.org/cdt](http://eclipse.org/cdt)
- CDT Wiki - [http://wiki.eclipse.org/CDT](http://wiki.eclipse.org/CDT)
Questions ?