UCL

Object and Model Management in SDEs

Wolfgang Emmerich
Professor of Distributed Computing
University College London
http://sse.cs.ucl.ac.uk

UCL

Learning Objectives

- To learn about the principle data structures handled by IDEs
- Appreciate the difference between parse trees and abstract syntax trees
- Understand the design rationales of abstract syntax trees and graphs
- Lay the foundation for working with the Eclipse JDT component

UCL

Key requirement for tools in SDEs

- Assist in editing correct formal language
  - Point out syntactic errors
  - Highlight static semantic errors
  - Inform about inter-document consistency constraints
  - Interpret and inspect

Demo

- Program editors are
  - incremental compilers
  - Language run-time environments
- They work on the same data structure as compilers
- Probably need a quick recap…

UCL

Parse Trees

- A tree that represents the syntactic structure of a sentence according to a grammar.
- In a parse tree
  - Inner nodes represent non-terminal symbols of the grammar.
  - Leave nodes represent terminal symbols of the grammar.
- Parse trees are generated by the parser component of a compiler.
- Need to look at an example

4

UCL

Example Parse Tree

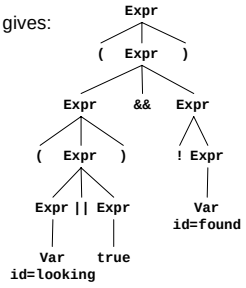
Given the grammar:

```
Expr ::= '(' Expr ')'
     |   Expr '&&' Expr
     |   Expr '||' expr
     |   '!' Expr
     |   Lit .
Lit  ::= Var | 'true' | 'false'.
Var  ::= [a-z][A-Z0-9_]+ .
```

Parsing this string:

```
(( looking || true) && !found )
```

gives:



5

UCL

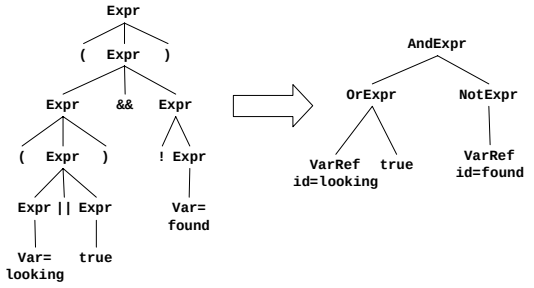
Abstract Syntax Trees

- Parse trees waste a fair amount of space for representation of terminal symbols and productions. In practice tools use abstract syntax trees.
- Abstract syntax trees (ASTs) are built by applying more abstract operators (reflected in inner nodes) and omitting lexical and structuring nodes that have no additional meaning.
- Compilers post-process parse trees into ASTs
- ASTs are the fundamental data structure of IDEs

6

UCL

Abstract Syntax Tree Example



7

UCL

Document Object Model

- A standard of the World-Wide-Web Consortium
- Standardises ASTs of XML documents
- Standardizes the programming interface to manipulate and traverse these ASTs
- DOM trees can be created by any DOM-compliant XML parser
- Given the prevalence of XML, DOM is extensively used in software development environments (and application servers)

8

UCL

Abstract Syntax Graphs

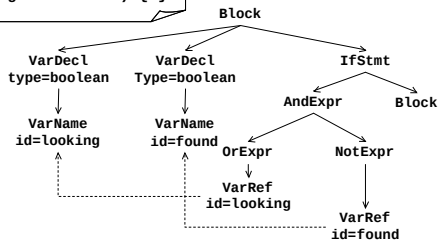
- Problem with ASTs: They do not support static semantic checks, re-factoring and browsing operations, e.g:
  - Have all used variables been declared
  - Have all Classes used been imported
  - Are the types used in expressions / assignments compatible?
  - Navigate to the declaration of method call / variable reference / type
- Abstract Syntax Graphs have additional edges that reflect semantic relationships, e.g. declare/use
- These edges are maintained during static semantic checks
- Static semantic checks might build upon previously established ones
- They are used in re-factoring operations (e.g. renaming a class).

9

UCL

Abstract Syntax Graph Example

```
boolean looking, found;
…
if (looking && !found ) {…}
```



10

UCL

Persistence of ASGs

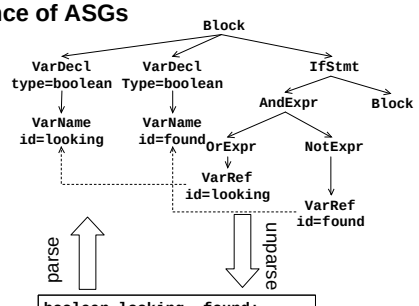
- In SDE research in the 1990s a lot of emphasis on how to store ASTs and ASGs persistently in different forms of databases.
- Today a developer's workstation has sufficient memory to hold ASGs, even of very large projects in main memory.
- Moreover, CPUs are much faster than they were a decade ago.
- Thus persistence is achieved by storage of artifacts in the file system.

11

UCL

Persistence of ASGs

In memory



On file system

```
boolean looking, found;
…
if (looking && !found ) {…}
```

12

Key Points

- Program editors in IDEs are effectively incremental compilers
- They work on abstract syntax trees or graphs as transient representations
- These are persisted by un-parsing into the file system

13

References

- A. Aho, R. Sethi and J. Ullman: Compilers. Addison Wesley. 1977
- M. Nagl (ed): Building Tightly integrated development environments. LNCS 1170. Springer Verlag. pp 32-44. http://dx.doi.org/10.1007/BFb0035684. 1996
- V. Apparao et al. Document Object Model. W3C Recommendation. http://www.w3.org/DOM/DOMTR. 1998

14