C++ Semantic Interface:  
Idea, Architecture, Implementation

Eugene Zouev

Bergen Language Design Laboratory  
Bergen University  
May 19, 2010
Outline

The idea
Related projects
Semantic representation
Advanced search model
XML representation
Implementation & current state
The idea of C++ Semantic API

The main idea is to provide researchers and programmers with a powerful, flexible and extensible platform for creating a wide range of language-related tools and applications.

(A research task) To experiment with separating C++ syntax from its semantics.

(A research task) To experiment with using XML for representing C++ semantics.
Related Projects

**ASIS**

Ada Semantic Interface Specification (for Ada95): the ISO standard

**SAGE - SAGE II - ROSE** (for C/C++, HPF...)

An open compiler infrastructure for source-to-source transformations

**Pivot** (for C++)

Stroustrup & Dos Reis; “General infrastructure for transformation and static analysis of C++ programs”

Some others...
Advantages of the project presented

**Extensibility**
Both source language and semantic representation are extendable

**Semantic search feature**
Powerful mechanism for investigating programs (including program comparisons)

**Doesn't depend on a third-party front-end**
Comprises parsing routines with name resolution, type checking etc.
The Evolution of the Compiler Architecture 1
The Evolution of the Compiler Architecture 2

Source Program → Compiler → Code Generation

Source Program → Front End Compiler → Intermediate Representation → Code Generation 1 → ... → Code Generation N
The Evolution of the Compiler Architecture

- Source Program
  - Compiler
    - Compiler Tables
  - Code Generation
- Source Program
  - Front End Compiler
    - Intermediate Representation
  - Code Generation 1
  - Code Generation N
- Source Program
  - Semantic Representation
    - Parser (SR Generator)
    - Code Generation
    - Visualization
    - Static Analysis
    - ...
SemantiC++: Common Scheme

Source Program
Code Snippet
Another SR
XML SR

iSource Interface
Semantic Representation
Program

Code Generators
Static Analyzers
Engineering Tools (UML)
Visualizers
Custom Verifiers
Interpreters: C++ Virtual Machines
Optimizers
Converters
SemantiC++: Basic Principles

A rich set of classes each of which represents a particular C++ language notion (class, statement, operator, operand etc.)

The relationships between classes (inheritance, aggregation, delegation) reflect conceptual relationships between corresponding language notions

For a source program, class instances compose an Abstract Syntax Tree for that program
This is not just a structure (like CCI): every class has a functionality for typical operations on ASTs (examples follow)

This is not just a syntax structure: every class has a set of attributes which represent various semantic properties of the notion ("annotated AST")

There is not just 1-to-1 correspondence between source and AST: hidden semantics is represented explicitly (destructor calls, operator function calls)
SemantiC++: Inheritance Class Diagram

(Indentation denotes inheritance)
class COMPOUND_STATEMENT : STATEMENT, iSCOPE {
    // Structure
    public LIST<STATEMENT> statements;
    public LIST<DECLARATION> declarations;
    // Creation
    protected COMPOUND_STATEMENT() ... 
    public static COMPOUND_STATEMENT create() ...
    // Opening
    public static COMPOUND_STATEMENT open
        ( iSource source, iSCOPE context )...
    // Validation
    public override bool check() ...
    public override bool validate() ...
    // Semantic search
    public static COMPOUND_STATEMENT pattern =
        COMPOUND_STATEMENT.create();
    public override bool match ( ENTITY pattern ) ...
    // Attributes
    public ENTITY owner;
    public bool isValid, ischecked, isGenerated;
}
Example of a Class: Some Comments

create(): a way to create a node/subtree from scratch

open(): a common means for reading node or subtree from outside: in particular, from a source text!

check(), validate(): check structural and semantical correctness of the node/subtree

match(): checks whether this node matches the parameter

pattern: common pattern for this node: matches ANY compound statement
SemantiC++: Example of an AAST (simplified)
using Semantic;
...

class Example {
    static void Main() {
        NAMESPACE_DECL ns =
            NAMESPACE_DECL.create(IDENT.create("N"));

        CLASS_DECL cls = CLASS_DECL.open( // Opening
            new FileSource("full-file-name"), ns);
        if (cls == null || !cls.validate()) {
            /* errors in class declaration */
        }

        string source = "int main() { cout << "Hello world!";
            return 0; } ";
        FUNCTION_DECL main = FUNCTION_DECL.openSource(
            new TextSource(source), ns);
        ns.add(cls, main);
        if (ns.validate()) ns.execute();  // ☺
    }
}
Binary and XML Formats: Two Faces of the Same

Both formats have the same rights (both are “first class citizens”)

Both formats are interchangeable

Internally there are converters Binary->XML & XML->Binary
Why XML?

- Open format
- Extensible
- Extremely simple model
- De-facto standard
- Lots of tools & technologies (e.g. XQuery, XSLT) to manipulate on
Why XML?

- Open format
- Extensible
- Extremely simple model
- De-facto standard
- Lots of tools & technologies (e.g. XQuery, XSLT) to manipulate on

- (Hidden idea 😊) To experiment with XSLT technology: is it applicable and useful to manipulate with C++ semantic representation (in XML form)?
Example of the XML representation (simplified)

```xml
<while-statement ln="1" col="1">
  <condition>
    <expression ln="1" col="7"> ... </expression>
  <condition>
  <compound-statement>
    <assignment-expression ln="2" col="4">
      <name ln="2" col="4">x</name>
      <expression ln="2" col="9"> ... </expression>
    </assignment-expression>
    <call ln="3" col="4">
      <name ln="3" col="4">P</name>
      <argument-list>
        <expression ln="3" col="5"> ... </expression>
      </argument-list>
    </call>
  </compound-statement>
</while-statement>
```
XML Based Architecture

- Program
- Semantic Representation
- in XML

Standard Access:
DOM, SAX, XSLT, XQuery

Custom APIs

Client Tools
Semantic Search
Implementation Approach

The project is being implemented on top of .NET in C#: faster programming, easier to maintain, more reliable code.

Interoperability: the SR is accessible from any .NET language (Managed C++, C#, VB, F#, Python, Zonnon).

All SR components have the form of .NET DLL libraries and can be attached to client programs in the standard way (“using xxx.dll”).
Current state of the project

“Semantic” classes, semantic search - completely implemented (not tested yet)

Opening routines for sources (parsing), XML Schema for semantic representation - are being developed

Client tools: (Re)engineering tool for UML - is being developed

Beta testing - planned...
Questions?
Critique?

JAXT – Java Axiom Testing