Public Defence, January 18, 2013

Contributions to the Construction of Extensible Semantic Editors

Emma Söderberg

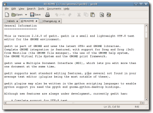
Doctoral Dissertation, 2012

Department of Computer Science
Lund University

Orientation

Today's programming editors

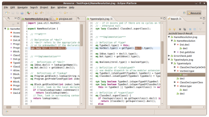
Text editor


README (~/cvs/gnome2/gedit) - gedit
File Edit View Search Tools Documents Help
Open
Save
Undo
NEWS
README
ChangeLog
Ln 15, Col 50
INS

```
General Information
===================

This is version 2.14.3 of gedit. gedit is a small and lightweight UTF-8 text
editor for the GNOME environment.

gedit is part of GNOME and uses the latest GTK+ and GNOME libraries.
Complete GNOME integration is featured, with support for Drag and Drop (DnD)
from Nautilus (the GNOME file manager), the use of the GNOME help system,
the GNOME Virtual File System and the GNOME print framework.

gedit uses a Multiple Document Interface (MDI), which lets you edit more than
one document at the same time.

gedit supports most standard editing features, plus several not found in your
average text editor (plugins being the most notable of these).

gedit plugins may also be written in the python scripting language: to enable
python support you need the pygtk and gnome-python-desktop bindings.

Although new features are always under development, currently gedit has:

    * Complete support for UTF-8 text
```

gedit, Notepad, ..

Semantic editor



```
import java.util.HashSet;

aspect NameResolution {

  // ***API***

  // Declaration of *decl*
  // *decl* refers to the appropriate d
  // or to unknownDecl if the declarati
  syn lazy Decl Access.decl();


  // ***Implementation***

  //   Definitions of *decl*
  eq IdUse.decl() = lookup(getName());
  eq Dot.decl() = getIdUse().decl();

  //   Definitions of *lookup*
  eq Program.getBlock().lookup(String na
  eq Program.getPredefinedType(int inde

  eq Block.getBlockStmt(int index).look
    // First, look in the local declarat
    if (!localLookup(name).isUnknown())
      return localLookup(name);
    // Then, look in surrounding contex
    return lookup(name);
  }
```

```
// - if it exists and if there are no cycles on t
// - otherwise null
syn lazy ClassDecl ClassDecl.superClass();


// ***Implementation***

// Definition of *type*
eq TypeDecl.type() = this;
eq VarDecl.type() = getType().decl().type();

eq IdUse.type() = decl().type();
eq Dot.type() = getIdUse().type();

eq BooleanLiteral.type() = booleanType();

// Definition of *isSubtypeOf*
// use double dispatch to allow modular extension
eq TypeDecl.isSubtypeOf(TypeDecl typeDecl) = type
eq ClassDecl.isSubtypeOf(TypeDecl typeDecl) = type

syn lazy boolean TypeDecl.isSuperTypeOf(TypeDecl 
syn lazy boolean TypeDecl.isSuperTypeOfClassDecl(
  this == typeDecl || typeDecl.superClass() != nu

// Definition of *superClass*
eq ClassDecl.superClass() {
  if (hasSuperclass() && getSuperclass().decl() i
    return (ClassDecl) getSuperclass().decl();
  else
```

Eclipse, NetBeans, ...

Error feedback
Code browsing
Refactoring
Name completion
...

Background:

- Programming: From text to semantic editor
- Not all languages have semantic editors

Problems:

- Construction is time-consuming and complex
- Maintenance may be difficult (extensions)

Challenge:

How can we make it easier to construct
and maintain semantic editors?

Approach

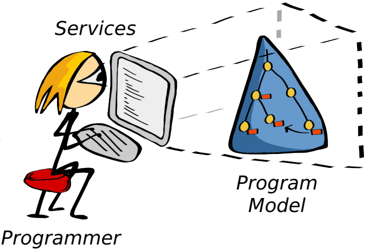
Generate services from specification

Specification: RAGs

- Formalism, specify semantics
- Declarative, easily modularized
- JastAddJ, JModelica, JastAdd

RAGs – *Reference Attribute Grammars*

- *Grammar* – defines program model, abstract syntax tree (AST)
- *Attributes* – computed properties of AST nodes (types, scopes, ..)
- *References*: Attribute values

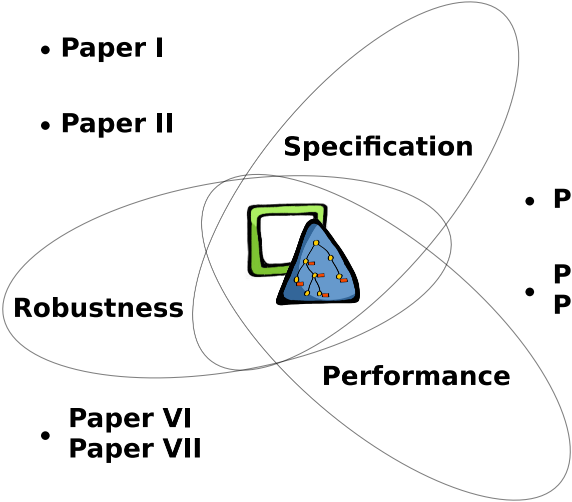


This dissertation:

Extend compiler with editor module

Contributions

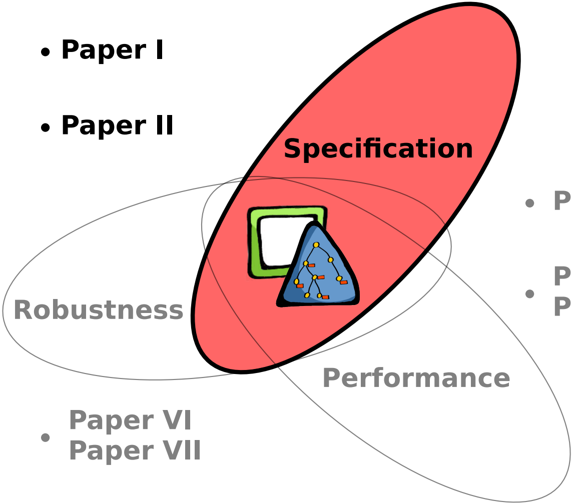
- **Paper I**
- **Paper II**



- **Paper III**
- **Paper IV**
  Paper V
- **Paper VI**
  Paper VII

Contributions

- **Paper I**
- **Paper II**



- Paper III
- Paper IV
  Paper V
- Paper VI
  Paper VII

Specification

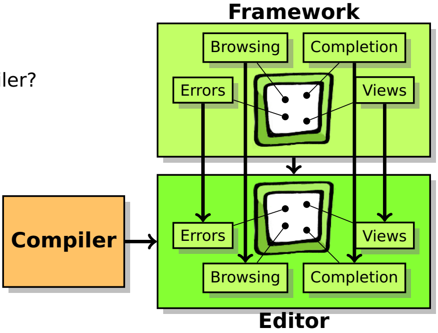
The JastAdd Editor Framework

Problem:

How to add an editor to an existing compiler?

Approach:

- Editor Framework
  - JastAdd: semantics
  - Eclipse: graphical components
- Predefined generic services
- RAG-based compiler extension



Results:

- Two demonstrators (size in LOC):
  - JastAdd: compiler 29, 200, editor 4, 300 (1, 100)
  - PicoJava: compiler 210, editor 600 (420)

Conclusions

- Modularly defined editor services
- Compiler reuse (name analysis, type analysis, ...)

[Paper I]

Specification

Example Service: Dead assignments

Problem:

How to specify flow analysis services?

```
a = 0;
b = 0;
a = b + 5;
a = a + b;
return a;
```

Here "a = 0" is a dead assignment because the value is not used and it could be removed.

Liveness (*textbook*)

Let n be a node and $succ[n]$ the set of successors for the node n:

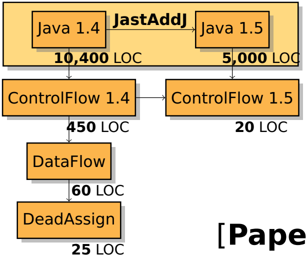
$$in[n] = use[n] \cup (out[n] \setminus def[n])$$
$$out[n] = \bigcup_{s \in succ[n]} in[s]$$

RAGs

```
syn Set CFGNode.in() circular [empty()] =
     use().union(out().compl(def()));

coll Set CFGNode.out() circular [empty()] with add;
Stmt contributes in() to CFGNode.out() for each pred();
Expr contributes in() to CFGNode.out() for each pred();
```

Conclusions:

- Textbook-like definitions
- Flow analysis added modularly with few LOC.
- Precision/performance on par with Soot.



JastAddJ

Java 1.4 — **10,400** LOC

Java 1.5 — **5,000** LOC

ControlFlow 1.4 — **450** LOC

ControlFlow 1.5 — **20** LOC

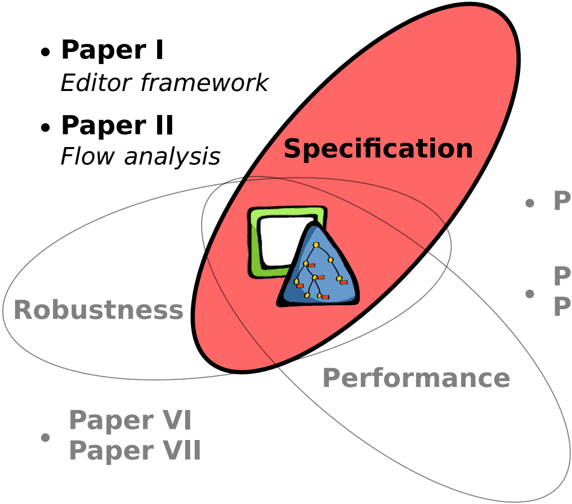
DataFlow — **60** LOC

DeadAssign — **25** LOC

[Paper II]

Contributions

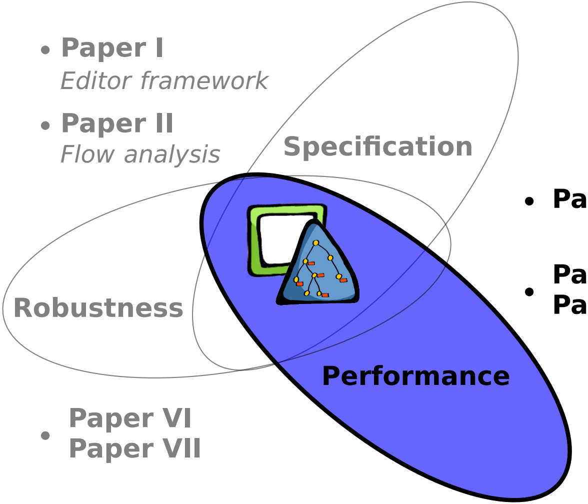
- **Paper I**
  Editor framework
- **Paper II**
  Flow analysis



- **Paper III**
- **Paper IV**
  Paper V
- **Paper VI**
  Paper VII

Contributions

- **Paper I**
  Editor framework
- **Paper II**
  Flow analysis



- **Paper III**
- **Paper IV**
  Paper V
- **Paper VI**
  Paper VII

Performance

Faster evaluation from scratch

Background: At attribute evaluation

- attribute dependencies → call graph
- *no caching* - multiple evaluations → *very slow*
- *full caching* - at most one evaluation → *faster*

Problem: Memory/performance costs

New idea: Selective caching

- based on profiling
- skip caching of some attributes

Results: 20% speedup and 38% memory reduction

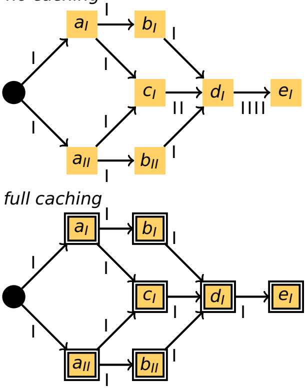
- Compared to full caching
- JastAddJ Java compiler
- Java benchmarks

Reference Attributes

$a = b.c$ $b = d$ $e = ..$

$c = d$ $d = e$

Attribute Instance Call Graphs



no caching

full caching

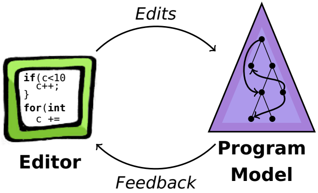
[Paper III]

Performance

Incremental evaluation

Problem:

How to efficiently update the program model after edits?



Edits

Editor

Program Model

Feedback

State of the art:

- Hand-coded solutions, complex, error-prone

Challenge:

- Automatically update model, RAGs

Earlier work:

- Optimal automatic updates for AGs
- No handling of references

New results:

- Dynamic algorithm for RAGs.
  - Build dynamic dependency graph during evaluation
  - Use graph to uncache affected attributes after edits

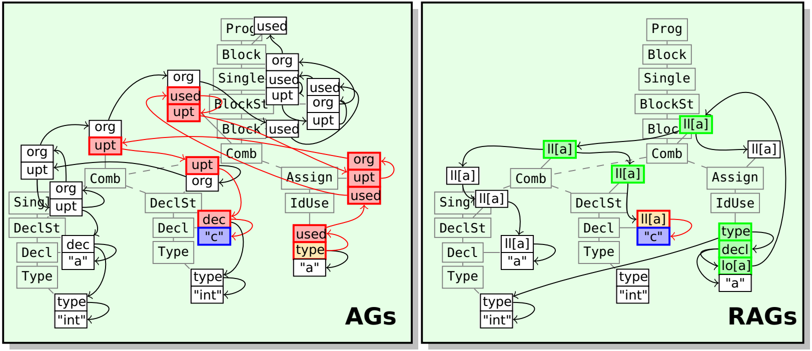
[**Paper IV**]

Performance

Incremental evaluation comparison

Example

int a;
int ~~c~~b;
a = 42;



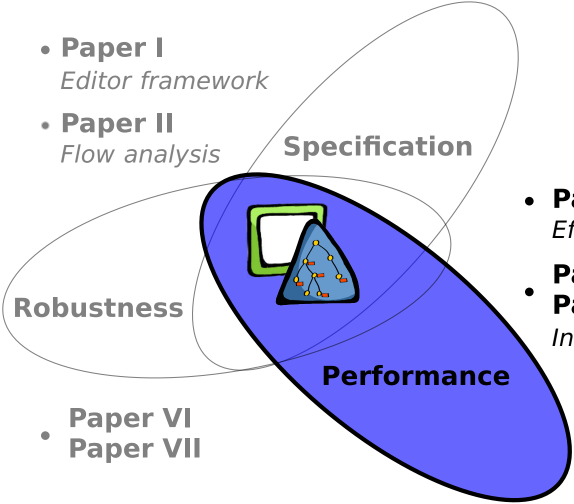
Legend

- Name — *AST node*
- name — *Attribute instance*
- ".." — *Token with value ".."*
- 🟦 *edited*
- 🟥 *affected*
- 🟧 *examined*
- 🟩 *skipped*
- - - *AST edge*
- ⟶ *Attribute dep.*
- ⟶ *Attribute dep. (examined)*

[Paper V]

Contributions

- **Paper I**
  Editor framework
- **Paper II**
  Flow analysis



Specification

Robustness

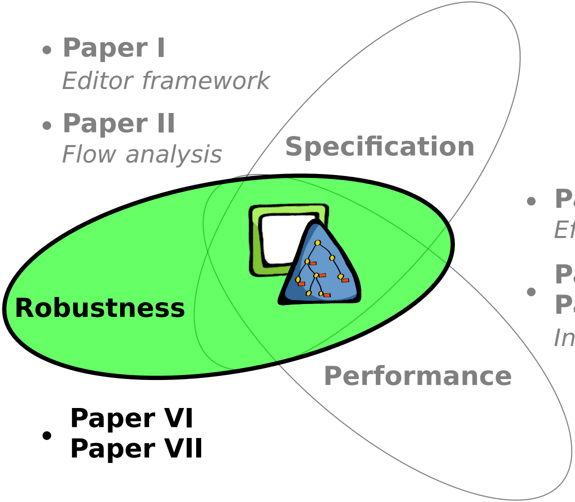
Performance

- **Paper III**
  Efficient caching
- **Paper IV**
  Paper V
  Incremental evaluation

- **Paper VI**
  Paper VII

Contributions

- **Paper I**
  Editor framework
- **Paper II**
  Flow analysis



- **Paper III**
  Efficient caching
- **Paper IV**
  Paper V
  Incremental evaluation

- **Paper VI**
  Paper VII

Robustness

How to handle erronoues input?

Problem:

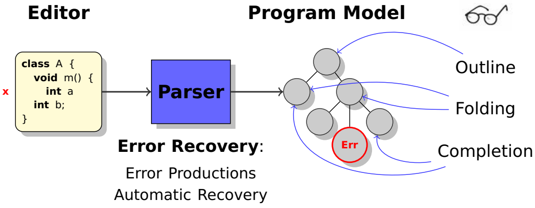
Scope errors cause recovery to fail

Idea:

- Use layout for recovery
- Aid existing recovery with preprocessor

New Algorithm:

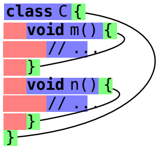
Bridge parsing



```
class A {
  void m() {
    int a
  int b;
}
```

Error Recovery:

Error Productions
Automatic Recovery



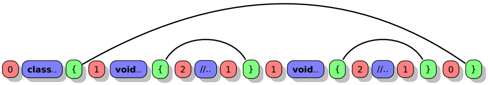
```
class C {
  void m() {
    // ..
  }
  void n() {
    // ..
  }
}
```

Island grammars:

- Islands: Interesting 🟩
- Water: Uninteresting 🟦

Bridge Parsing

- Reefs: Patterns for recovery 🟥
- Bridges – Scopes



[Paper VI]

Robustness

Bridge Parsing Algorithm

```
class C {
  void m() {
    // ...

  void n() {
    // ...
  }
}
```

Algorithm:

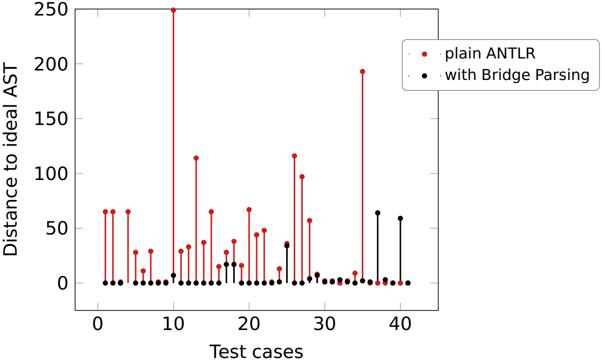
1. Tokenizer
2. Bridge Builder
3. Bridge Repairer

[**Paper VI**]

Robustness

Results from adding bridge parsing

Antlr – a well-known LL-based parser generator



[**Paper VI**]

Robustness

Bridge Parsing ideas in JSGLR

Collaboration: TU Delft

Problem:

Provide error recovery for Scannerless GLR (SGLR)

SGLR:

- *Generalized* LR: Arbitrary CFG
- *Scannerless* – include tokens in the grammar
- Language composition, e.g., Java-SQL and enum
- JSGLR – implementation of SGLR in Java

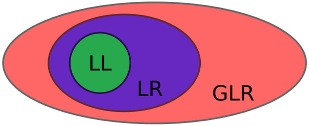
Method:

Recovery using island grammars, layout and bridge parsing

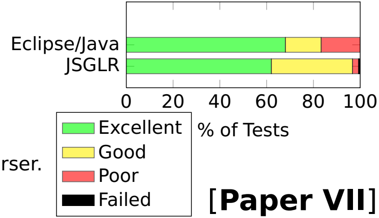
Results:

Recovery quality on par with the Eclipse Java parser.

Context Free Grammars



Recovery Quality

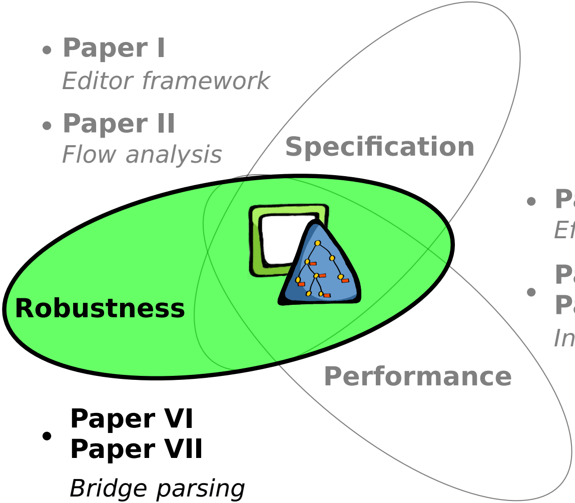


[Paper VII]

Bridge Parser Parts

Contributions

- **Paper I**
  Editor framework
- **Paper II**
  Flow analysis



Specification

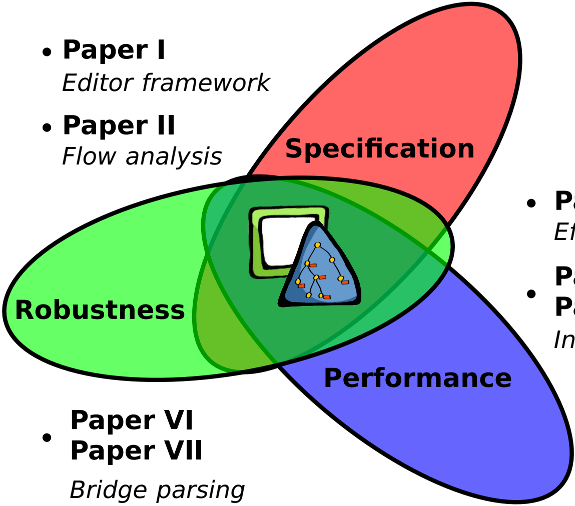
Robustness

Performance

- **Paper III**
  Efficient caching
- **Paper IV**
  Paper V
  Incremental evaluation

- **Paper VI**
  Paper VII
  Bridge parsing
  Layout-sensitive recovery

Contributions



- **Paper I**
  Editor framework
- **Paper II**
  Flow analysis
- **Paper III**
  Efficient caching
- **Paper IV**
  Paper V
  Incremental evaluation
- **Paper VI**
  Paper VII
  Bridge parsing
  Layout-sensitive recovery