Статический анализ ПО с помощью промежуточных представлений и технологий с открытым исходным кодом

Зубов М.В.,
Пустыгин А.Н.,
Старцев Е.В.

ЧелГУ, 2012

Достоинства статического анализа

- Не требует подготовленных или генерируемых входных данных, специально созданного окружения
- Допускает кросс-системный анализ программ, независимо от целевой программной и аппаратной платформы
- Не зависит от частоты появления событий, влияющих на функционирование ПО
- Не требует затрат ресурсов целевой платформы(программно-аппаратных стендов)

Недостатки статического анализа

Требует обработки больших массивов символьной информации
С увеличением объема исходных текстов время анализа может расти быстрее,чем растет объем
Неизбежно использование упрощенных моделей, что ухудшает качество анализа
Не все ошибки могут быть обнаружены, одновременно происходят “ложные срабатывания”

Существующие методы статического анализа ПО

- Анализ текста вручную (неэффективен в силу больших человеческих затрат, однако, дает наиболее точный результат)
- Машинная обработка текста без формирования вспомогательных данных (имеет очень ограниченное применение)
- Обработка с использованием промежуточных представлений (машинное эквивалентное представление кода)

Пример ошибки

- …
- `if ( x == "something") {…}`
- …
- `// Java`

Пример возможной ошибки

- ...
- int x = getValue();
- …
- public Integer getValue(){...}
- …
- // Java

Существующие направления применения статического анализа

- Анализ структуры программных систем
- Тестирование
- Верификация
- Проверка соответствия стандартам разработки
- сопровождение

Верификация

Инструмент	Open source	Особенности	Разработчик
BLAST	+	один из наиболее развитых open source-продуктов	Университет Беркли
Frama-C	+	для описания спецификаций требований используется специальный язык ACSL	INRIA и CEA (Франция)
Linux Driver Verification	+	верификации драйверов Linux (обеспечивает необходимое окружение для проверки инструментами работающими с обычными программами с фиксированной точкой входа)	ИСП РАН
Astree	-	заявляют об обнаружении RTE-ошибок, а не потенциально возможных, а также возможности рассмотрения всех возможных ошибок	ENS при поддержке CNRS (Франция)

Проверка соответствия стандартам разработки

Стандарт	Инструмент	Особенности
MISRA C/C++	LDRA Testbed, PC-Lint, Parasoft Compliance	Ориентирован на улучшение надежности приложений для встроенных систем. Применяется в ряде отраслей промышленности (автомобилестроение, медицина, авиация).
JSF++ AV	LDRA Testbed, Parasoft Compliance	Для C++, ориентирован на вопросы безопасности приложений для военных истребителей-бомбардировщиков НАТО, созданных по программе JSF «единый ударный истребитель».
ISO 26262	Parasoft Compliance, ITEM ToolKit	Адаптация стандарта функциональной безопасности IEC 61508 для автомобильных электрических/электронных систем. Направлен на минимизацию рисков.

Анализ структуры программных систем

- Общее устройство системы
- Взаимодействие модулей

Способы выполнения статического анализа

- Текстовая обработка
- Модели отражения
- Промежуточные представления

Варианты текстовой обработки

- Поиск по регулярным выражениям
- Построение метрик

Модели отражения (reflexion models)*

RM - это структура программы, построенная из её исходного текста

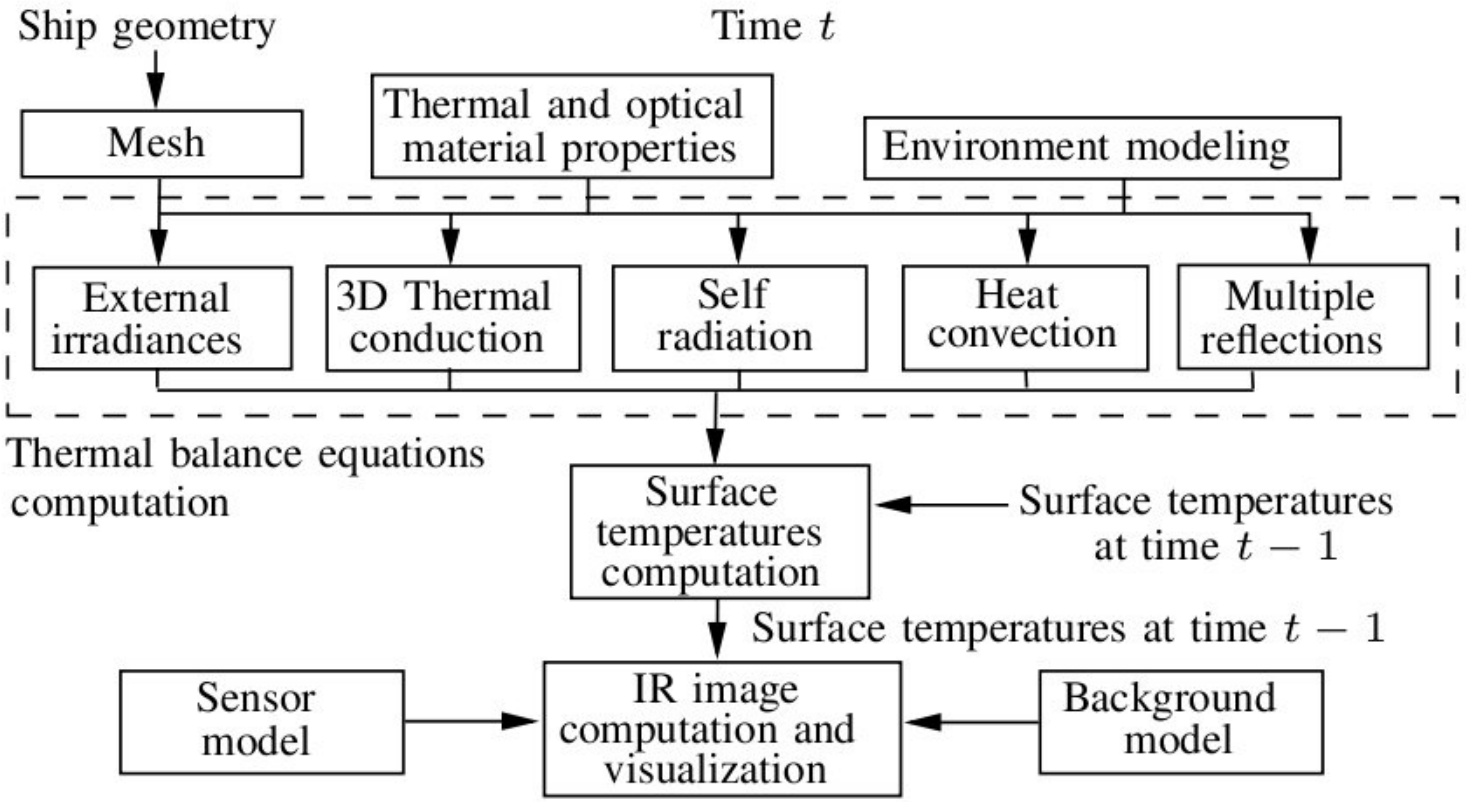
*Термин введен G. Murphy и коллегами в 1995 году

Жизненный цикл ПО и место RM в нем

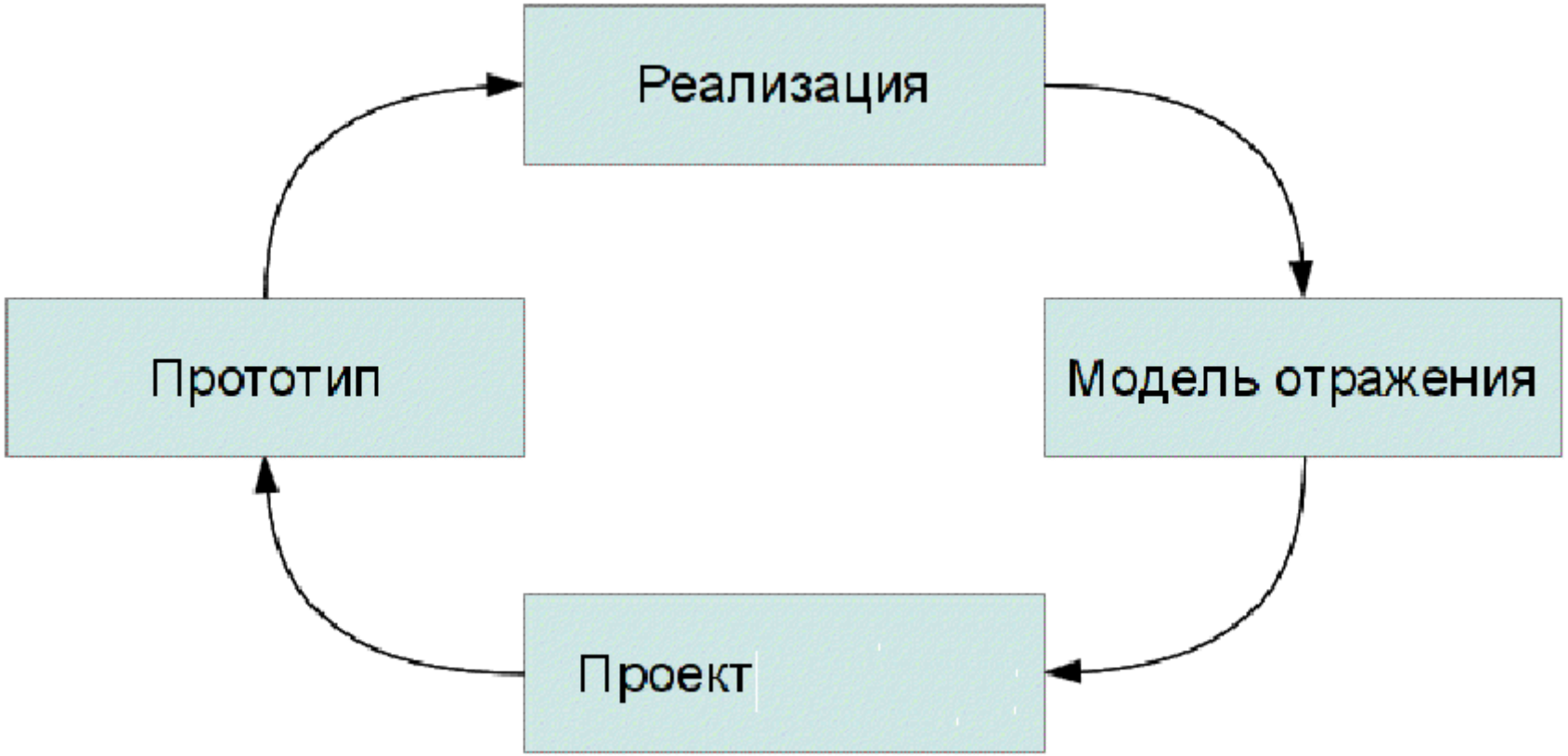
- Проектирование
- Прототип
- Реализация

Проектирование

«прямоугольники и стрелочки»



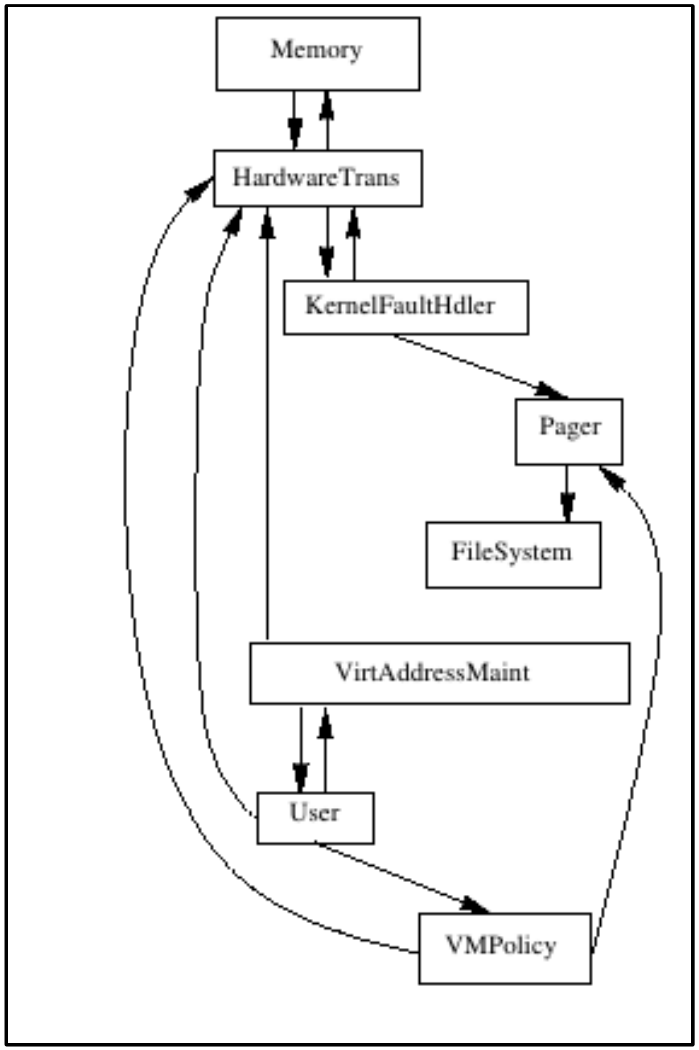
Место модели отражения при анализе ПО



Возможные области применения моделей отражения

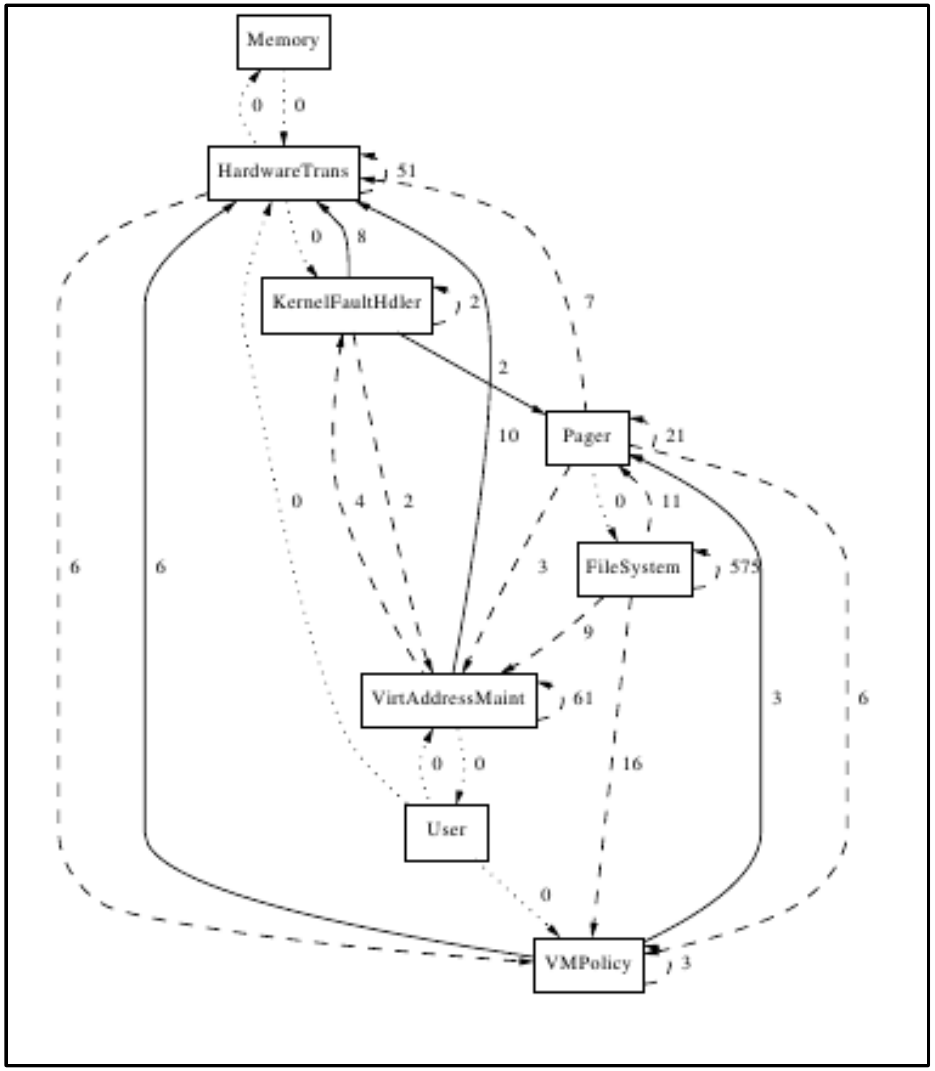
- Изучение истинных связей в структуре программной системы
- Планирование изменений в системе
- Изучение архитектуры системы

Подсистема виртуальной памяти UNIX



C.Murphy,D.Notkin,and K.Sullivan, Software Reflexion models: Bridging the gap between source and high-level models

Модель отражения подсистемы виртуальной памяти NetBSD



C.Murphy,D.Notkin,and K.Sullivan, Software Reflexion models: Bridging the gap between source and high-level models

Правила отображение исходного кода на модель

- `[ file=.*pager.*  mapTo=Pager ]`
- `[ file=vm_map.*  mapTo=VirtAddressMaint ]`
- `[ file=vm_fault\.c  mapTo=KernelFaultHdler ]`
- `[ dir=[un]fs  mapTo=FileSystem ]`
- `[ dir=sparc/mem.*  mapTo=Memory ]`
- `[ file=pmap.*  mapTo=HardwareTrans ]`
- `[ file=vm_pageout\.c  mapTo=VMPolicy ]`

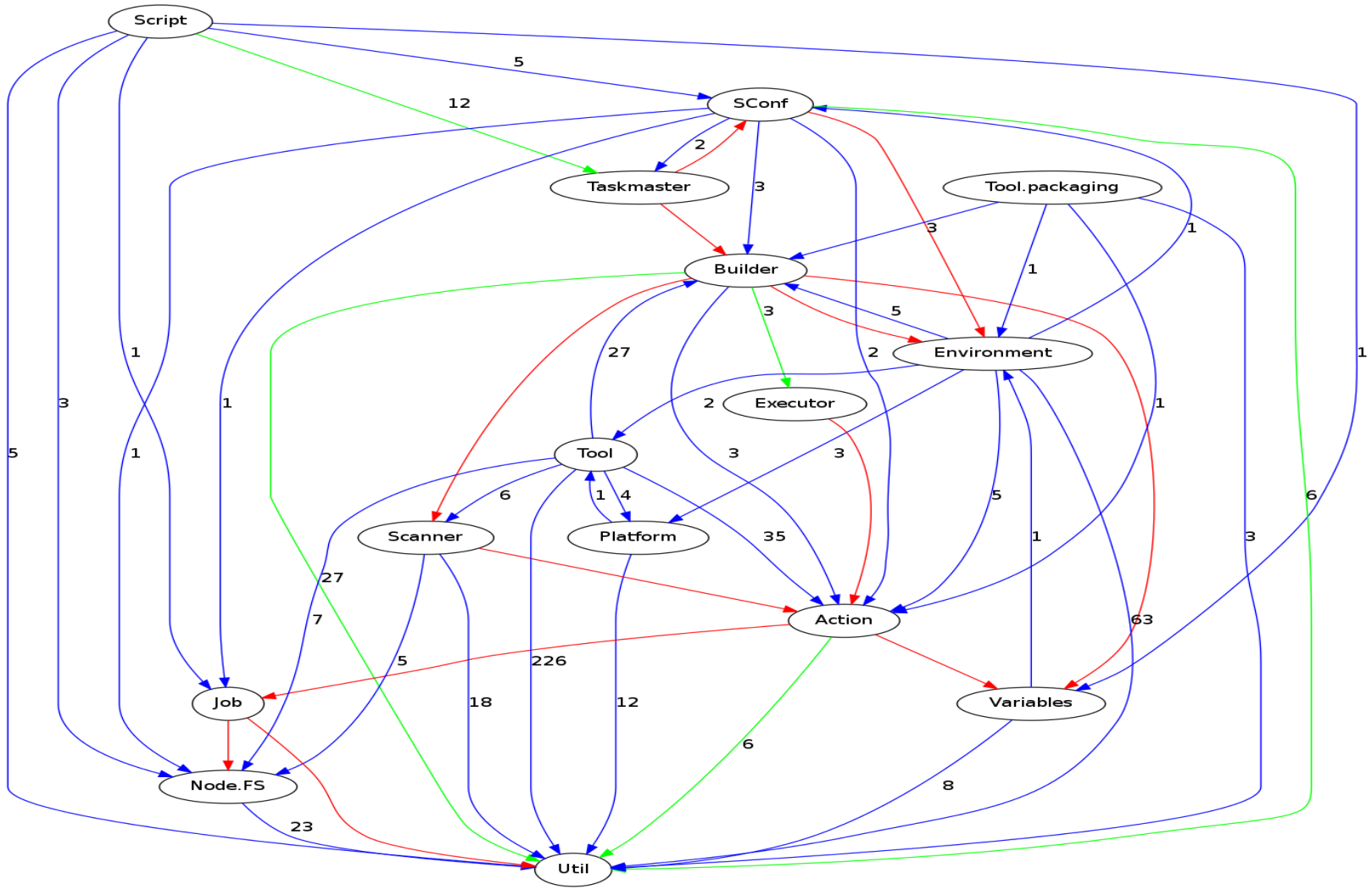
Опыт применения моделей отражения

- Исследуются open-source Python-проекты
- Реализация на основе пакета logilab-astng, на этом пакете основана утилита pylint, предназначенная для поиска ошибок в тексте Python
- Выходные данные генерировались в dot-формате (pydot)

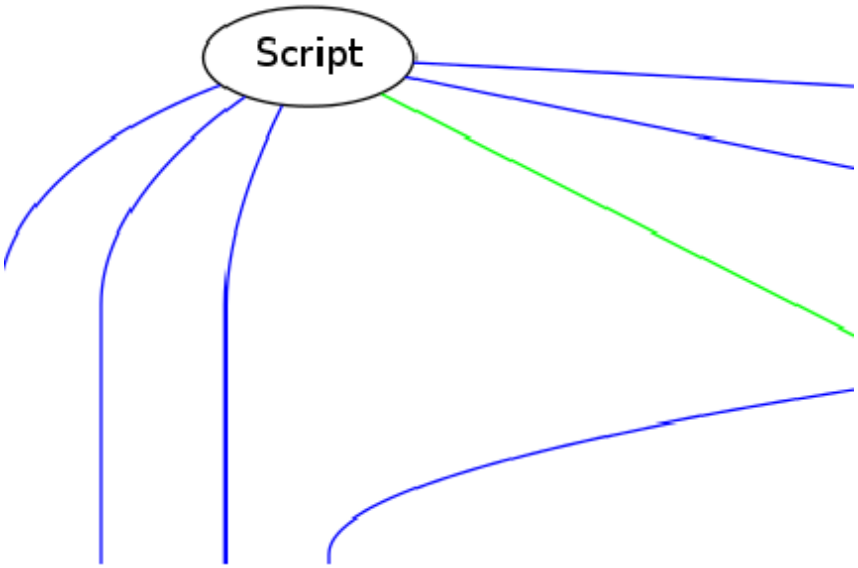
Исследованные проекты

- Scons (версия 2.1.0) -кросс-платформенная система сборки (аналог утилиты make)
- Logilab (logilab-astng версии 0.20.1 и logilab-common версии 0.50.3) — пакет обработки AST-деревьев компилятора Python и вспомогательные средства

Модель отражения для SCons

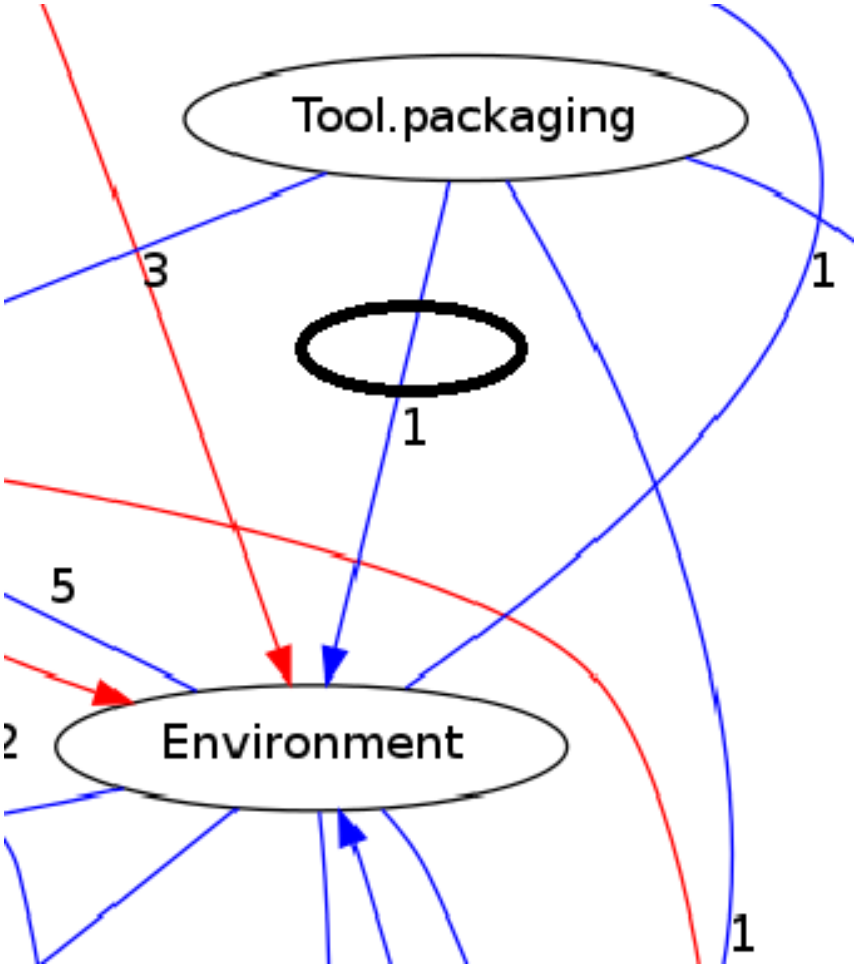


Знание



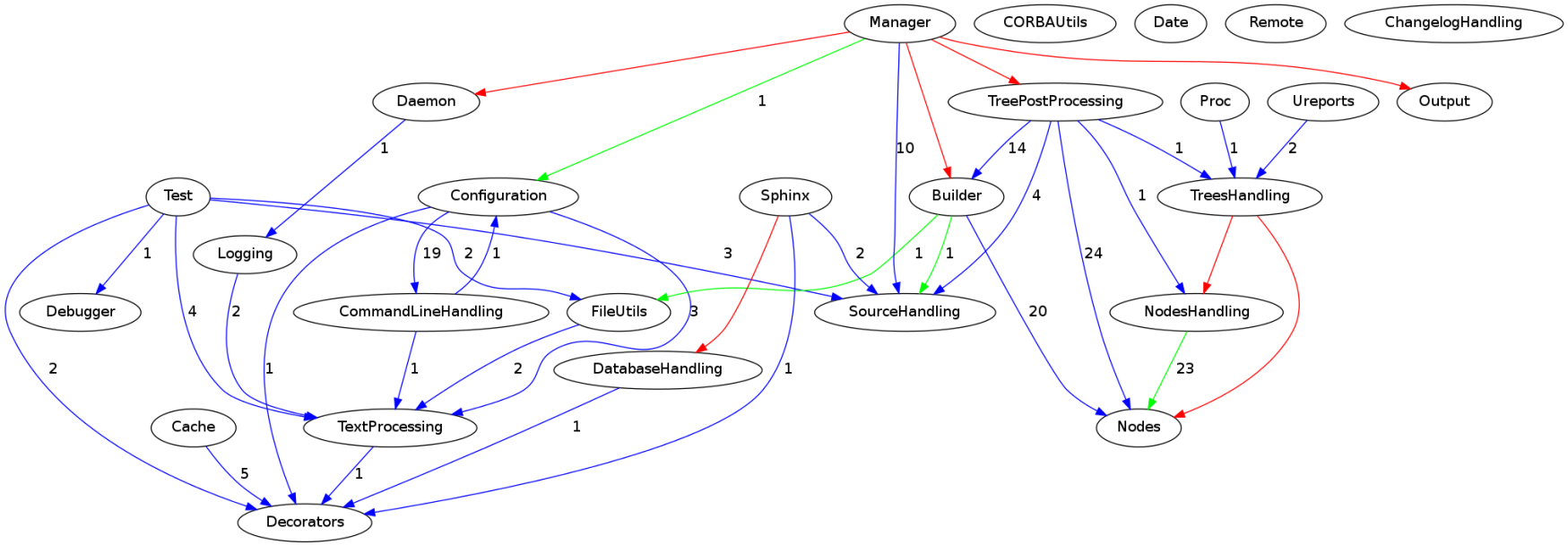
- Script - это точка входа SCons при запуске из внешних оболочек
- Этот факт подтверждает данные из документации на SCons

Еще знание

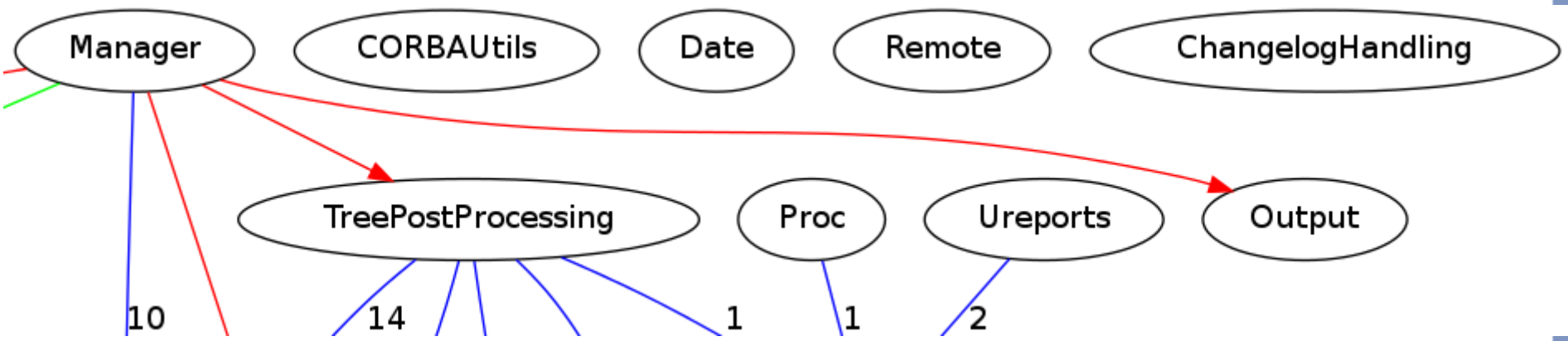


- При сборке rpm-пакетов используется возможность переопределить окружение переданное сборщику
- При сборке других типов пакетов это не используется
- Это может говорить о плохом проектировании (возможно)

Модель отражения для logilab-astng

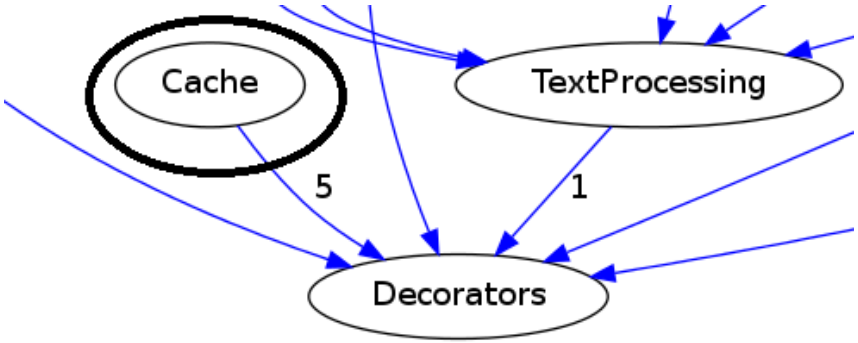


Знание



- Модули CORBAUtils, Date, Remote, ChangelogHandling, Output — независимы

Еще знание



- Компонент Cache не используется внутри проекта (не связан с кэшированием в Manager)

Модели отражения позволяют

- Сравнить проект с реализацией
- Найти ошибки в структуре системы
- Получить дополнительные знания

Классификация промежуточных представлений по типам данных реализации

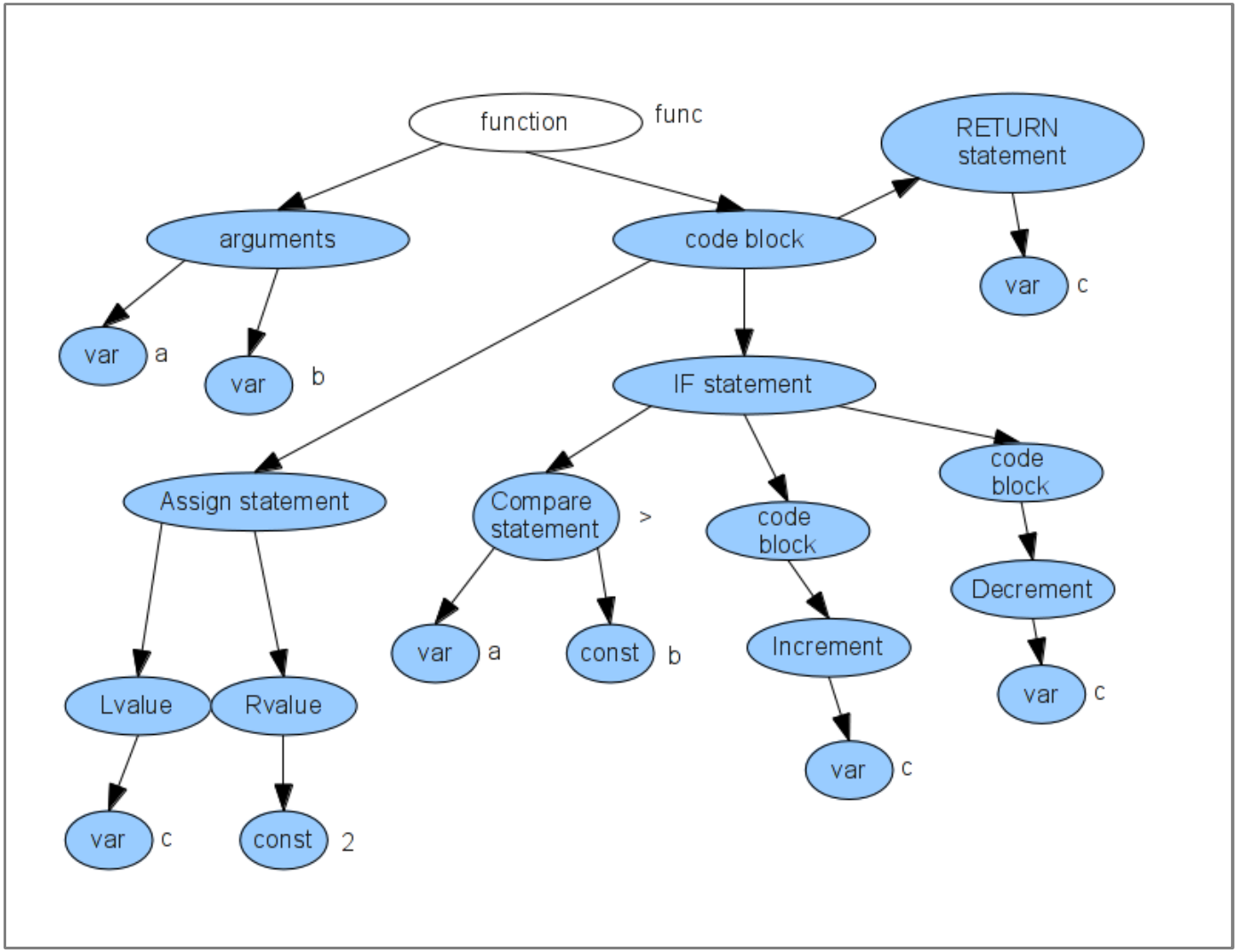
Абстрактное синтаксическое дерево (AST)

- Самое распространённое
- Самое изученное
- Является внутренним набором данных компиляторов

Пример исходного текста на C

```
1  int func (int a, int b){
2      int c=2;
3      if (a>b) {
4          c++;
5      }
6      else {
7          c--;
8      }
9      return c;
10 }
```

AST примера


function
func
RETURN statement
arguments
code block
var
c
var
a
var
b
IF statement
Assign statement
Compare statement
>
code block
code block
Decrement
var
a
const
b
Increment
var
c
Lvalue
Rvalue
var
c
const
2

Пример исходного текста на Python

```
class NextClass:
    def printer(self, text):
        self.message = text
        print self.message
```

AST примера, полученное с помощью logilab

```
Module(simple)
  body = [
  Class(NextClass)
    bases = [
    ]
    body = [
    Function(printer)
      decorators =
      args =
      Arguments()
          args = [
          AssName(self)
          AssName(text)
          ]
          defaults = [
          ]
```

```
body = [
      Assign()
        targets = [
        AssAttr(message)
          expr =
          Name(self)
        ]
        value =
        Name(text)
      Print()
        dest =
        values = [
          Getattr(message)
            expr =
            Name(self)
          ]
      ]
    ]
```

AST примера, полученное с помощью препарата Jython

```
<project name="SimpleProject">
  <package name="__default__">
    <module name="simple.py">
      <classdef name="NextClass">
        <def name="printer">
          <arguments>
            <arg name="self"/>
            <arg name="text"/>
          </arguments>
          <assign>
            <lvalue>
              <field name="message">
                <var name="self"/>
              </field>
            </lvalue>
            <rvalue>
              <var name="text"/>
            </rvalue>
          </assign>
          <print>
            <field name="message">
              <var name="self"/>
            </field>
          </print>
        </def>
      </classdef>
```

Примеры инструментов, использующих AST

Название	Open Source	Особенности	Метод получения AST
Checkstyle	+	Анализатор для языка Java. Ищет ошибки в коде по типовым эвристикам и соответствия стилю оформления кода.	ANTLR
Compass/ROSE	+	Использует инфраструктуру компилятора ROSE. Получает данные напрямую из компилятора. Языки: C, C++, Fortran.	Напрямую из компилятора ROSE
PyLint	+	Проверка качества кода и поиск ошибок в программах на Python.	На основе утилиты logilab
DMS Software Reengineering ToolKit	-	Комплексный коммерческий анализатор. Поддерживает анализ нескольких языков сразу (в одном проекте). COBOL, C, C++, Java, Fortran, VHDL, и т.д..	Собственный парсер

Свойства AST-представления

- Наглядность
- Лёгкость получения
- Простота обхода
- Затраты на обход

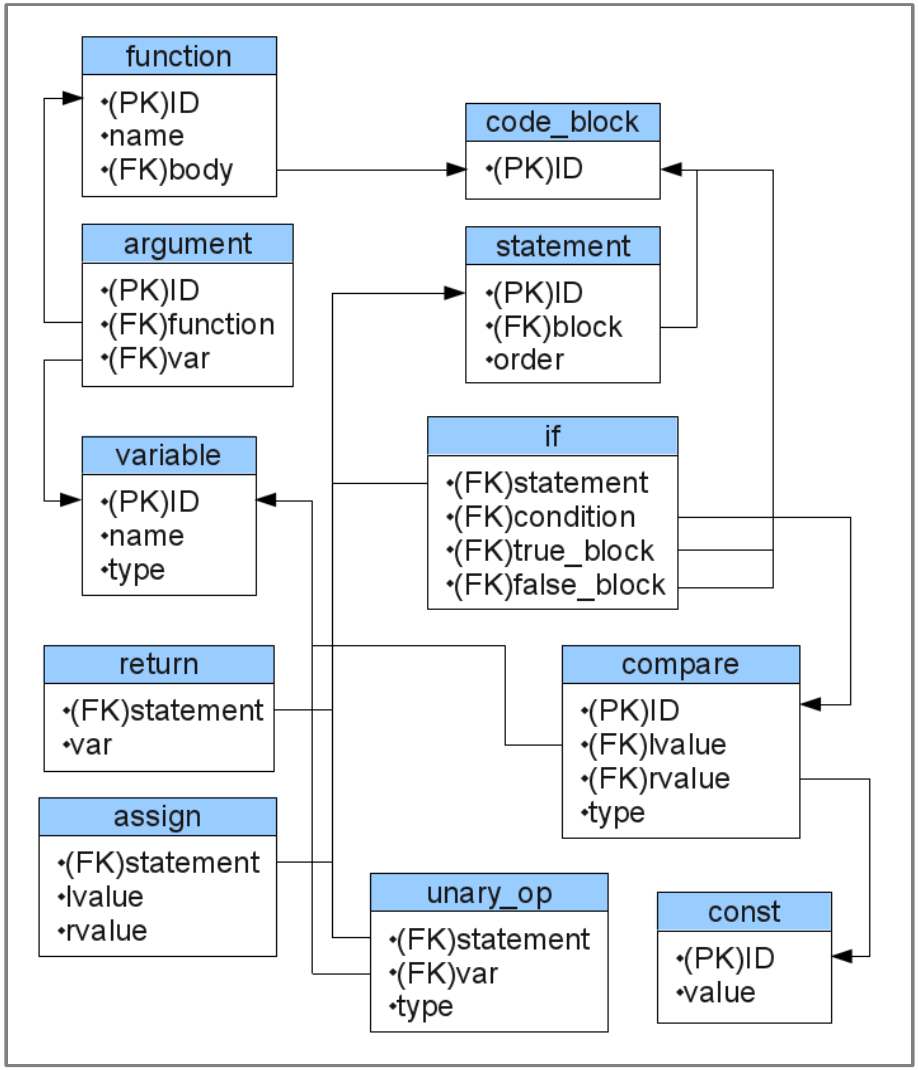
Реляционное представление

- Каждый элемент грамматики — отношение
- Связи грамматики — ключи и ограничения
- Формируется не напрямую из кода

Пример исходного текста на C

```
1  int func (int a, int b){
2      int c=2;
3      if (a>b) {
4          c++;
5      }
6      else {
7          c--;
8      }
9      return c;
10 }
```

Структура БД



Заполнение таблиц БД

function		
ID	name	body
1	func	1

code_block
ID
1
2
3

statement		
ID	block	order
1	1	1
2	1	2
3	2	1
4	3	1
5	1	3

argument		
ID	function	var
1	1	1
2	1	2

const	
ID	value
1	2
2	b

if			
statement	condition	true_block	false_block
2	1	2	3

return	
statement	var
5	c

unary_op		
statemen	var	type
3	3	post++
4	3	post--

variable		
ID	name	type
1	a	int
2	b	int
3	c	int

assign		
statement	lvalue	rvalue
1	3	1

compare			
ID	lvalue	rvalue	type
1	1	2	g

Характеристика реляционного представления

Достоинства	Недостатки
Удобно для хранения больших объемов данных, так как БД оптимизированы для этого.	Достаточно избыточно (свойственно всем БД).
Процедура анализа требует выполнение запросов к БД, что эффективнее обхода AST на больших проектах.	Плохо понятно для человека.
Решает проблему аппаратных затрат на хранение и обход AST.	Выше затраты на создание такого представления, так как требуется дополнительное проектирование структуры БД (таблиц, индексов, ключей) для каждой грамматики.

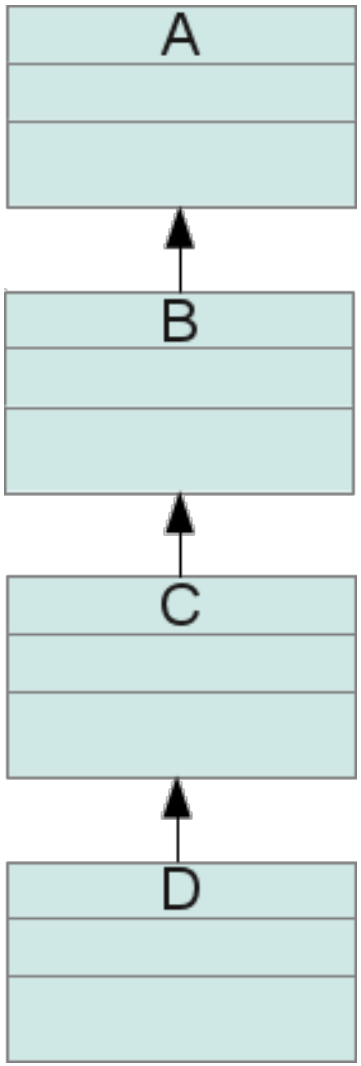
Существующие инструменты, использующие реляционное представление

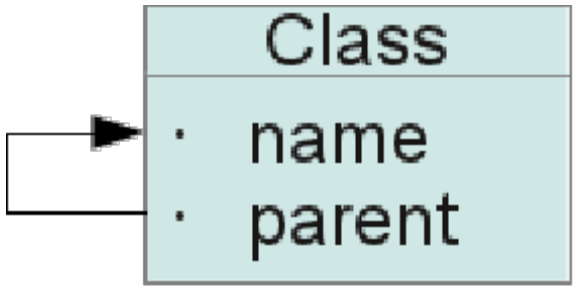
- SemmleCode
  semmle.com/solutions/enabling-tools-for-your-projects/
- CodeQuest
  progtools.comlab.ox.ac.uk/projects/codequest/

Проблема реляционного представления

- Описания могут образовывать циклические зависимости. Анализ таких структур описывается механизмом транзитивных замыканий

Пример транзитивного замыкания на диаграмме классов





Class	
name	parent
A	
B	A
C	B
D	C

Реализация транзитивного замыкания

- SQL:
  - SELECT * FROM Class WHERE name='B' AND parent='A';
  - SELECT * FROM Class AS c1 JOIN Class AS c2 ON c2.parent=c1.name WHERE c2.name='C' AND c1.parent='A';
- Prolog:
  - Parent(X, Y) :- Class(X, Y).
  - Parent(X, Y) :- Parent(X, Z), Class(Z, Y).
  - Parent('D', 'A') будет true.

Существующие инструменты, использующие логические языки в статическом анализе

- CodeQuest (Datalog) progtools.comlab.ox.ac.uk/projects/codequest/
- ASTLog (Prolog) research.microsoft.com/apps/pubs/default.aspx?id=68191

Недостатки использования логических языков для реляционных моделей ПО

- Вызывают рост затрат на реализацию
- Логические языки куда менее распространены, чем языки запросов

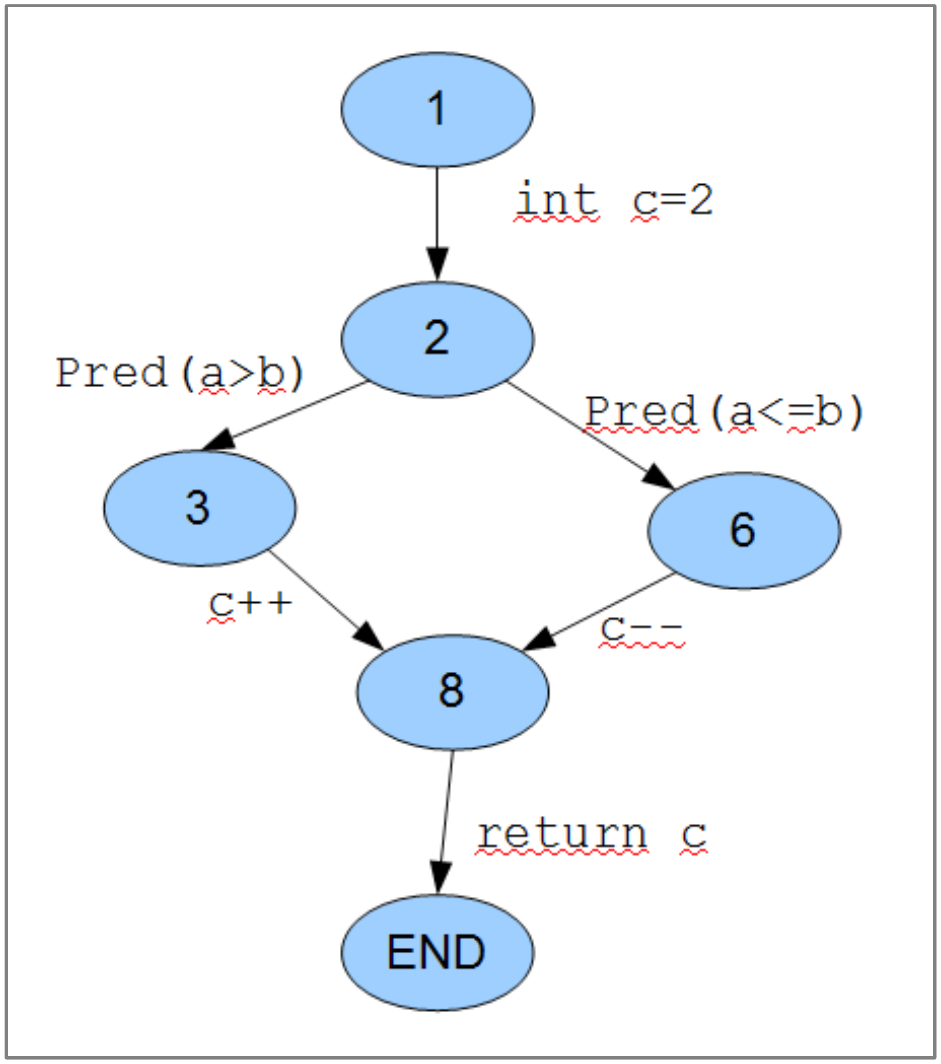
Автоматное представление

- Представление программы в виде конечного автомата
- На каждом шаге исполнения — отдельное состояние
- Вычисляемые условия - переходы

Пример исходного текста на C

```
1  int func (int a, int b){
2      int c=2;
3      if (a>b) {
4          c++;
5      }
6      else {
7          c--;
8      }
9      return c;
10 }
```

Автоматное представление примера



Использование автоматного представления

- BLAST mtc.epfl.ch/software-tools/blast — инструмент для верификации

Характеристика автоматного представления

- Огромные аппаратные затраты
- Почти 100% результат при анализе потока управления
- Гарантированное покрытие всех участков кода

Специфичные промежуточные представления

Представление байт-кода Java в виде дерева FindBugs в утилитах findbugs.sourceforge.net

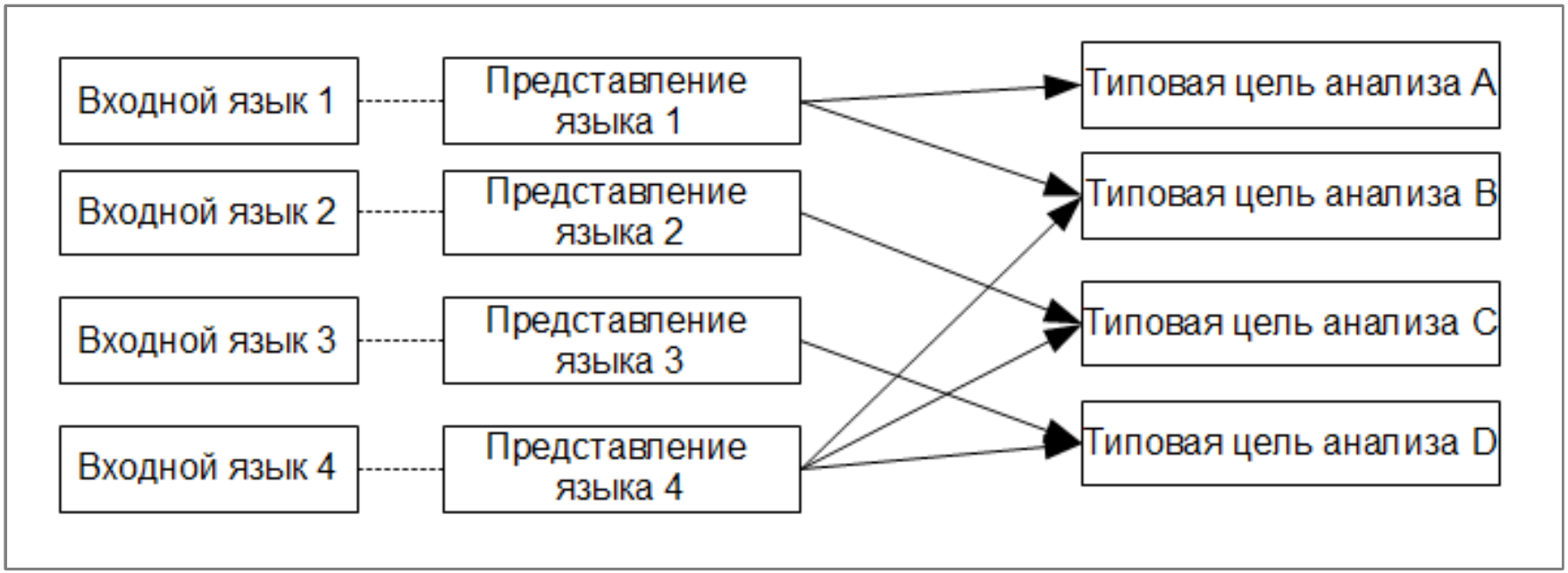
Soot sable.mcgill.ca/soot/

Классификация представлений по виду организации

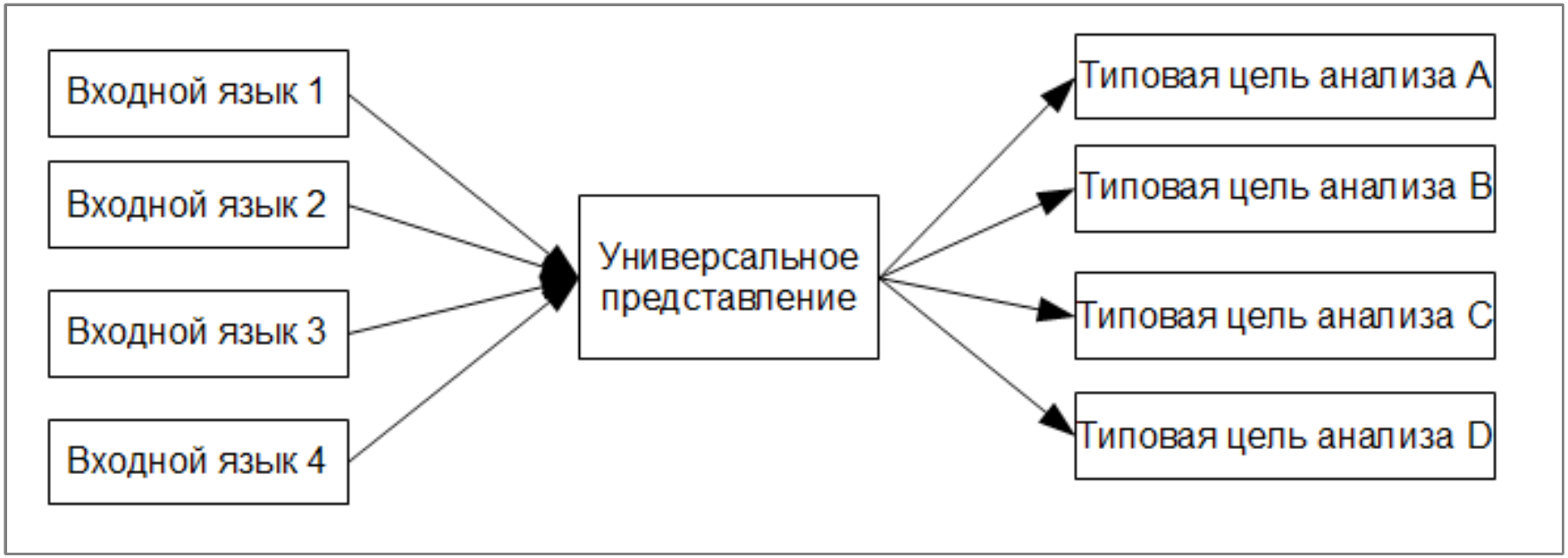
Универсальные и частные

- Универсальные — одно представление на несколько языков
- Частные — для каждого входного языка своё представление

Частные представления



Универсальные представления



Многоуровневые представления

- Берётся лучшее из представлений разной природы
- Разные задачи — разные представления
- Компромисс между затратами на хранение и обработку

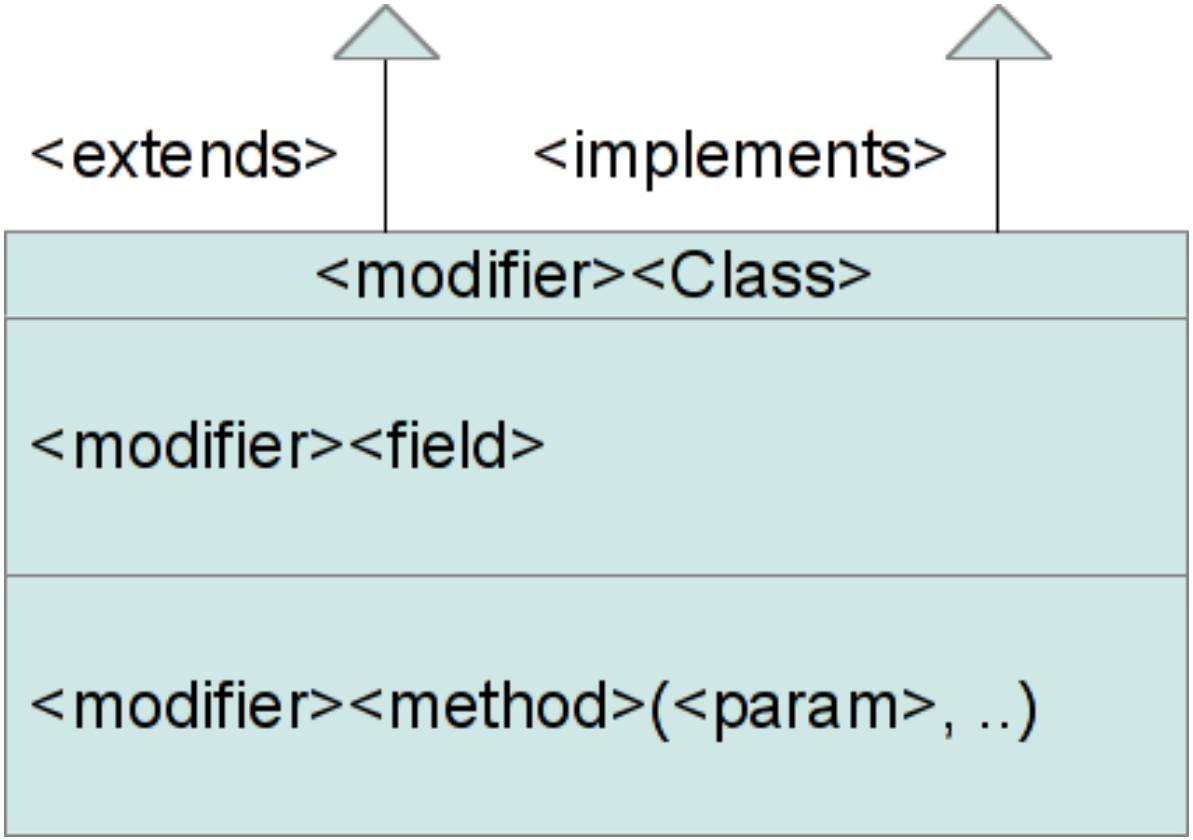
Инструменты, использующие многоуровневые представления

- Bauhaus Project
  bauhaus-stuttgart.de
  - InterMediate Language — описание конструкций языка
  - Resource Flow Graphs — граф взаимодействия модулей, файлов

Универсальное классовое представление

- Все объектно-ориентированные языки используют одинаковые подходы
- Можно выделить общее для всех составных частей классов любого языка

Концепция классового представления



Пример

- Классическая реализация паттерна Visitor

Исходный код на Python

```
class Node():
    lineno = 0
    str_pos = 0
    def Accept(self,node_visitor):
        pass


    def __init__(self):
        pass

class ClassNode(Node):
    name = 'class'
    def Accept(self,node_visitor):
        node_visitor.visit_class(self)

class AssignmentNode(Node):
    left = 'left'
    right = 'right'
    def Accept(self,node_visitor):
        node_visitor.visit_assignment(self)

class VariableNode(Node):
    name = 'var'
    initial_value = 'right'
    def Accept(self,node_visitor):
        node_visitor.visit_variable(self)
```

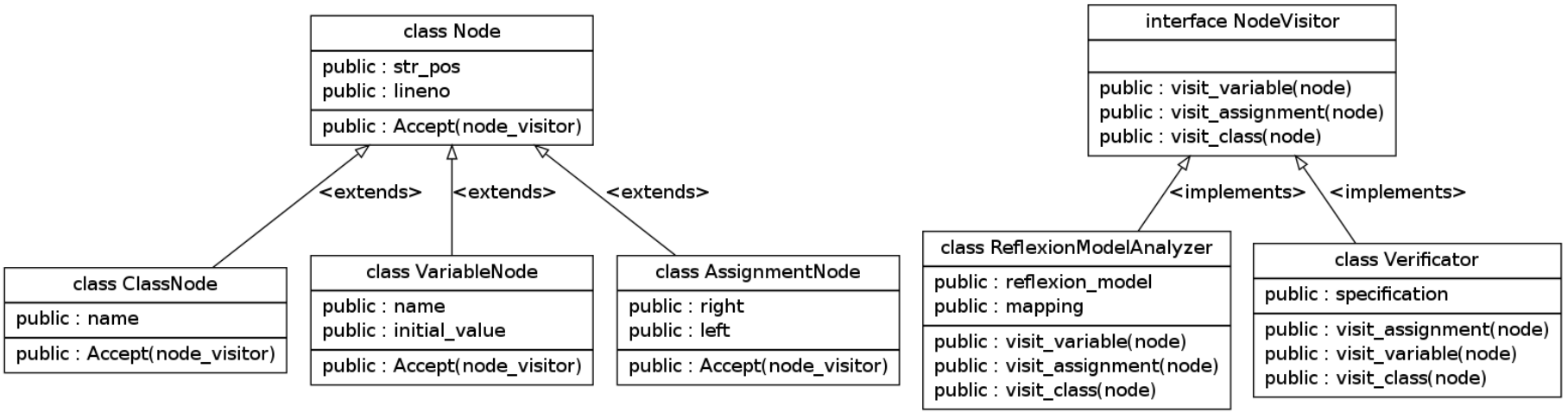
```
class NodeVisitor():
        def visit_class(self,node):
            pass
        def visit_assignment(self,node):
            pass
        def visit_variable(self,node):
            pass

class ReflexionModelAnalyzer(NodeVisitor):
        reflexion_model = None
        mapping = None
        def visit_class(self,node):
            print 'RM visit class'
        def visit_assignment(self,node):
            print 'RM visit assign'
        def visit_variable(self,node):
            print 'RM visit var'

class Verificator(NodeVisitor):
        specification = None
        def visit_class(self,node):
            print 'Verify class'
        def visit_assignment(self,node):
            print 'Verify assign'
        def visit_variable(self,node):
            print 'Verify var'
```

Исходный код на Java

```
abstract public class Node  {
        public int lineno = 0;
        public int strPos = 0;
        abstract public void Accept(NodeVisitor visitor)
  }

public class ClassNode extends Node  {
        public String name = "class";
        public void Accept(NodeVisitor visitor) {
                visitor.visitClass(this);
                }
  }

public class AssignmentNode extends Node  {
        public String left = "left";
        public String right = "right";
        public void Accept(NodeVisitor visitor) {
                visitor.visitAssignment(this);
                }
  }
public class VariableNode extends Node {
        public String name = "var";
        public String initialValue = "right";
        public void Accept(NodeVisitor visitor) {
                visitor.visitVariable(this);
                }
```

```
public interface NodeVisitor  {
        public void visitClass(ClassNode node);
        public void visitAssignment(AssignmentNode node);
        public void visitVariable(VariableNode node);
}

public class ReflexionModelAnalyzer implements NodeVisitor  {
        public Object reflexionModel;
        public Object mapping;
        public void visitClass(ClassNode node){
                System.out.println("RM visit class");
        }
        public void visitAssignment(AssignmentNode node){
                System.out.println("RM visit assign");
        }
        public void visitVariable(VariableNode node){
                System.out.println("RM visit var");
        }
}

public class Verificator implements NodeVisitor  {
        public Object specification;
        public void visitClass(ClassNode node){
                System.out.println("Verify class");
        }
        public void visitAssignment(AssignmentNode node){
                System.out.println("Verify assign");
        }
        public void visitVariable(VariableNode node){
                System.out.println("Verify var");
        }
}
```

Универсальное промежуточное представление



Достоинства полученного универсального предсталения

- Ниже затраты на инструменты анализа
- Точно описывает систему на своем уровне абстракции

Спасибо за внимание!

Вопросы ?...