Подходы к статическому анализу открытого исходного кода

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

LVEE 2012

В наши дни

- Создается очень много программного обеспечения
- Не все разработчики следят за качеством кода

А потом эти люди его исправляют!

Цели анализа ПО

- Поиск проблемных мест
- Понимание устройства
- Улучшение и расширение

Статический анализ
-
анализ без исполнения кода

Достоинства

- Не требует входных данных
- Не требует особого окружения
- Не зависит от целевой архитектуры

Недостатки

- Доступ к исходному коду
- Большие затраты
- Поиск эвристик
- Ложные срабатывания

Применение

- Поиск ошибок
- Верификация
- Получение информации

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)

Получение информации

- Архитектура системы
- Взаимодействие модулей
- Поиск определённых конструкций

Подходы к анализу

Вручную...

Машинные подходы

- Текстовая обработка
- Модели отражения
- Промежуточные представления

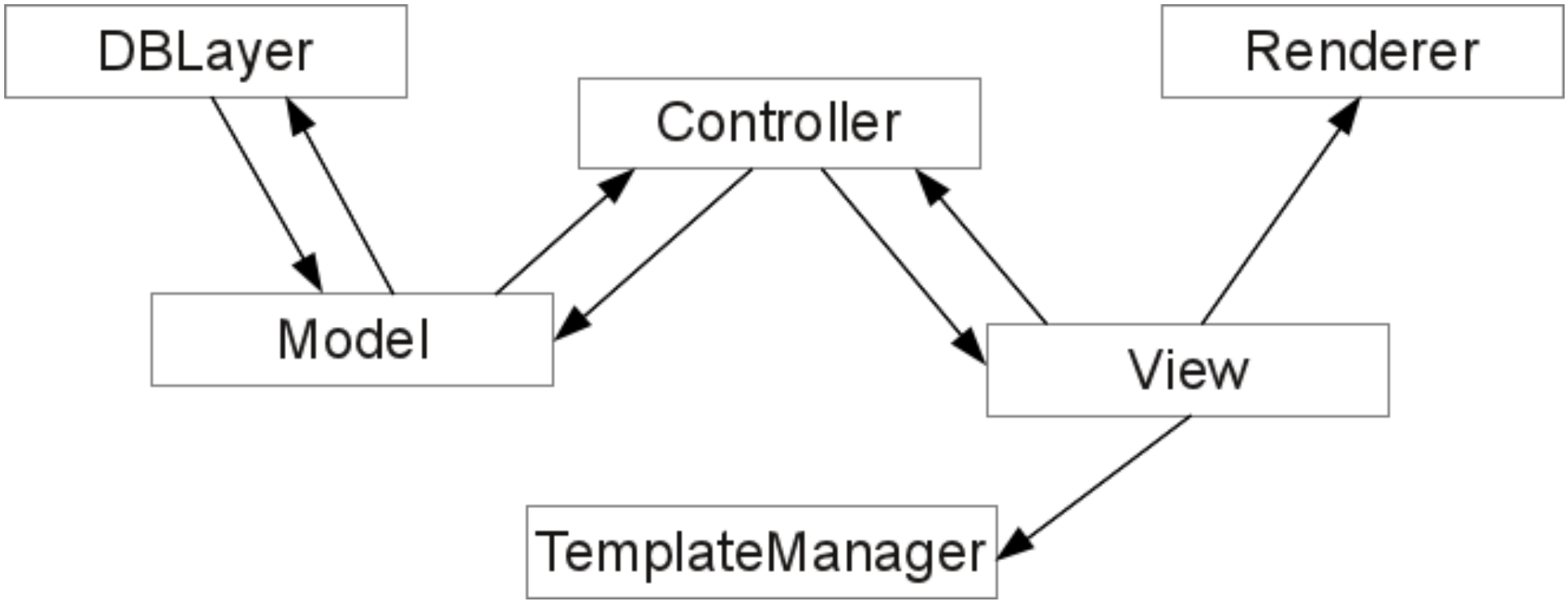
Модели отражения (reflexion models)*

Термин введен G. Murphy и коллегами в 1995 году

Программисты…

- …не любят сложностей*
- …любят наглядность
- …представляют ПО в виде высокоуровневых моделей
- …рисуют прямоугольники со стрелочками!

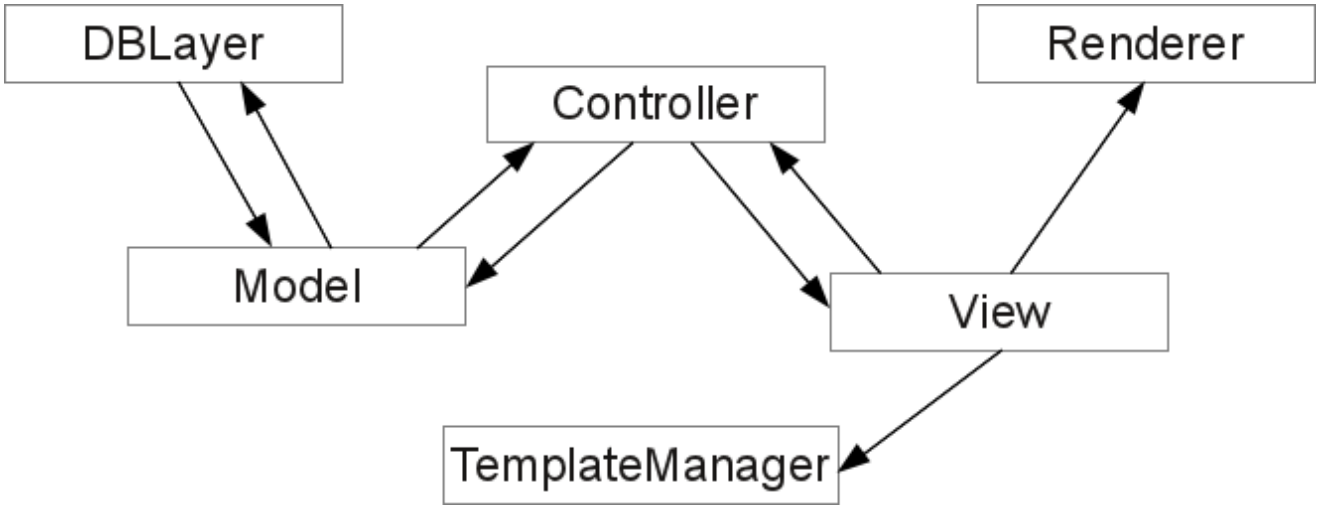
Типичные прямоугольники и стрелочки



А на самом деле это:

- ~100 классов
- ~250 файлов исходного кода
- ~600 функций
- >200 000 строк кода

Вы здесь видите 250 файлов исходного кода?

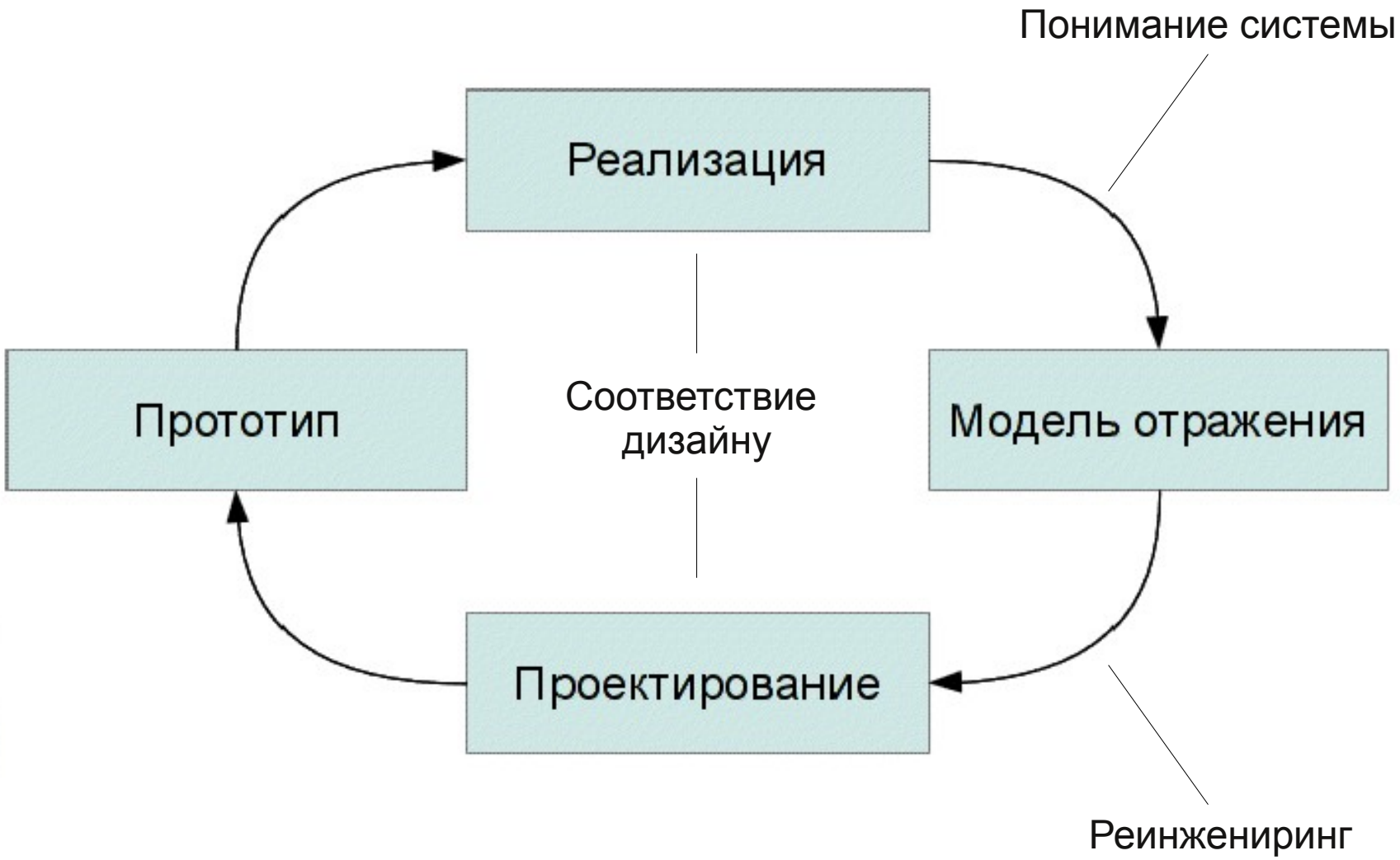


А разработчики видят!

Модель отражения

- Это модель программы, полученная по её коду («прямоугольники со стрелочками»)
- Показывает особенности реализации,выявляет ошибки
- Ориентированный на человека способ анализа!

Применение


Понимание системы
Реализация
Прототип
Соответствие дизайну
Модель отражения
Проектирование
Реинжениринг

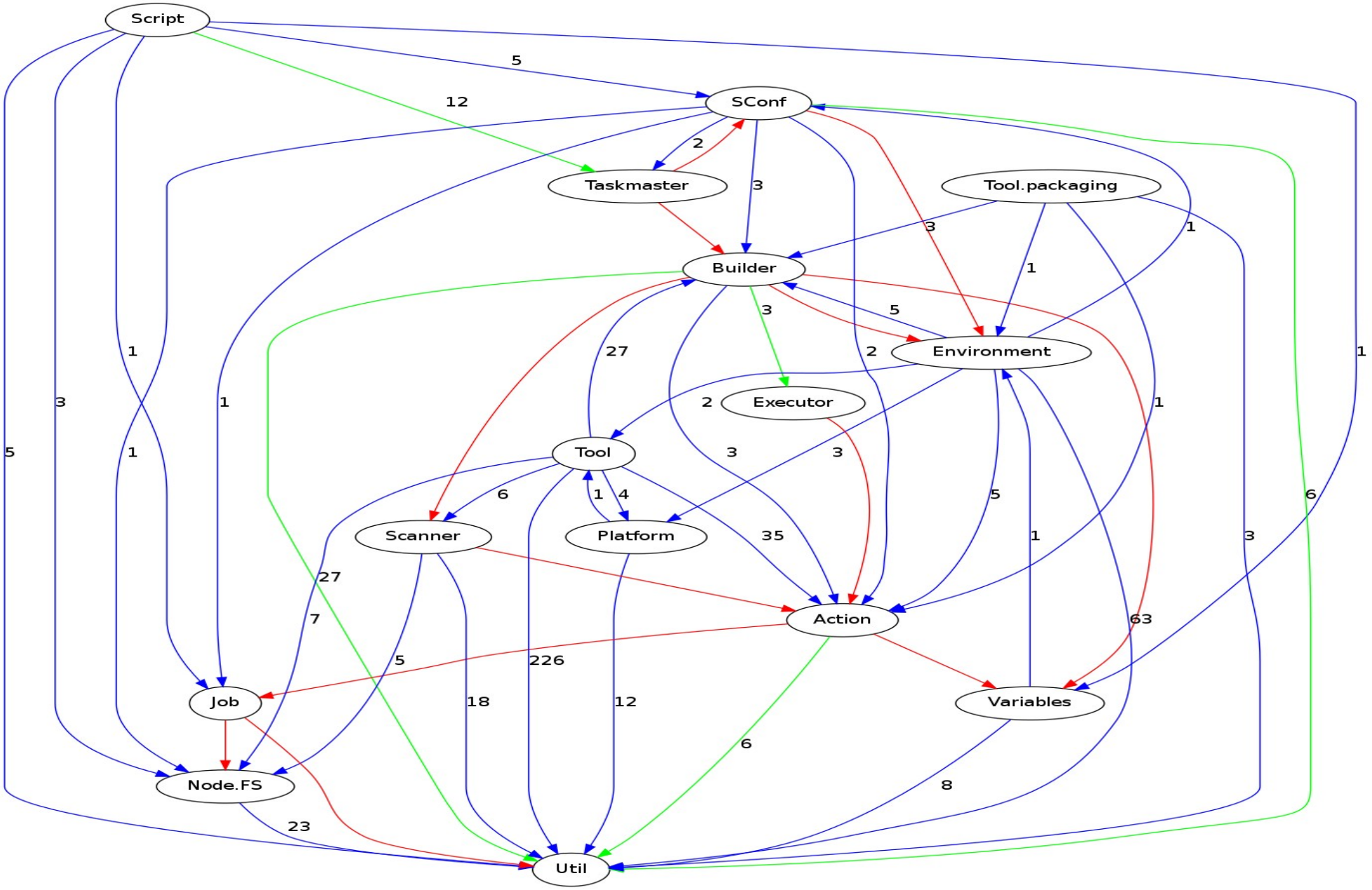
Опыт применения моделей отражения

- Open-source Python-проекты
- Используется logilab-astng (PyLint)
- Представление результатов на основе dot (pydot)

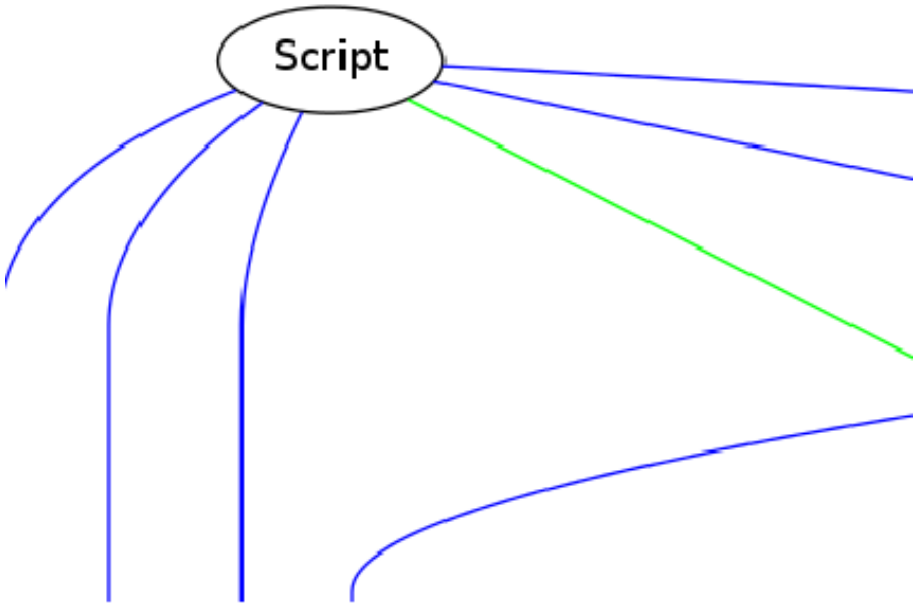
Исследованные проекты

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

Модель отражения для SCons

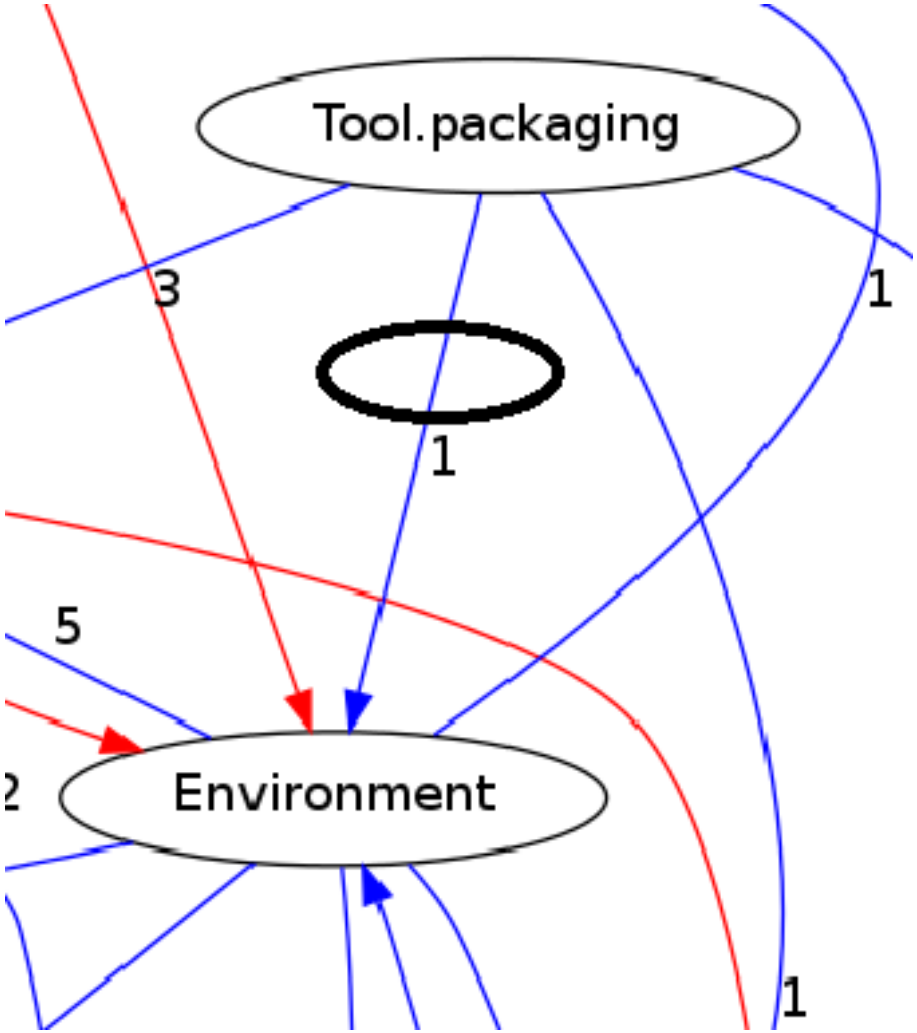


Знание



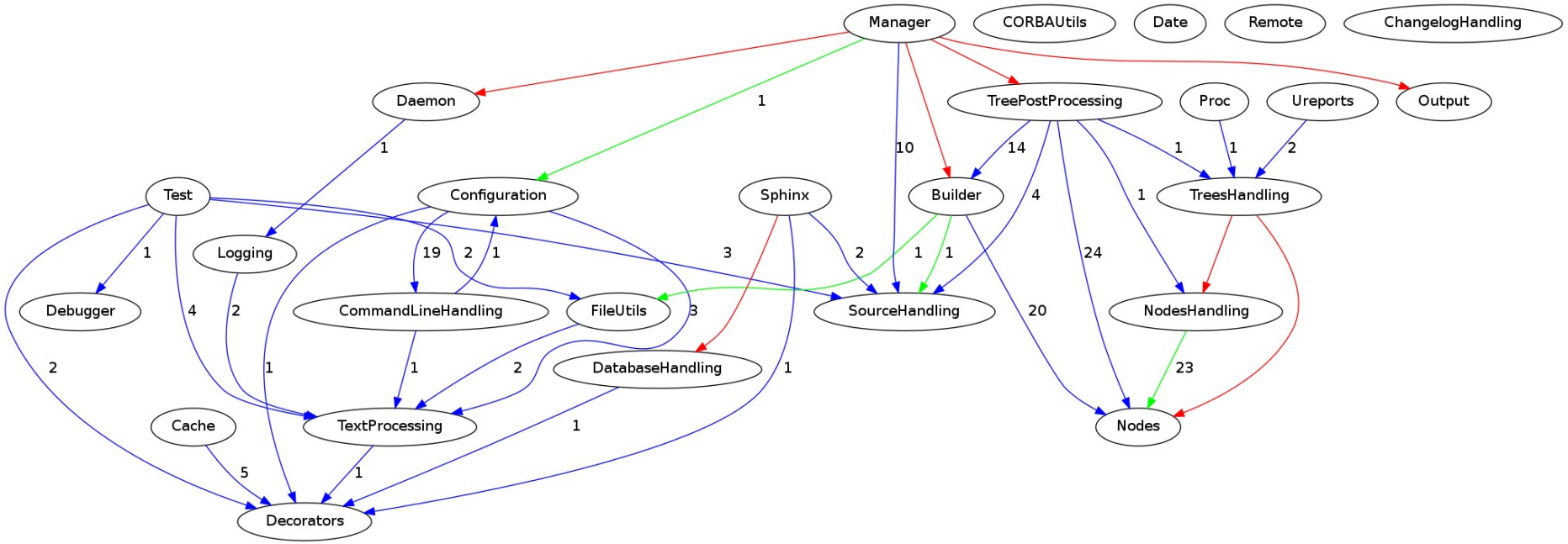
- Script - это точка входа SCons при запуске из внешних оболочек
- Этот факт подтверждает данные из докуметации на SCons

Еще знание

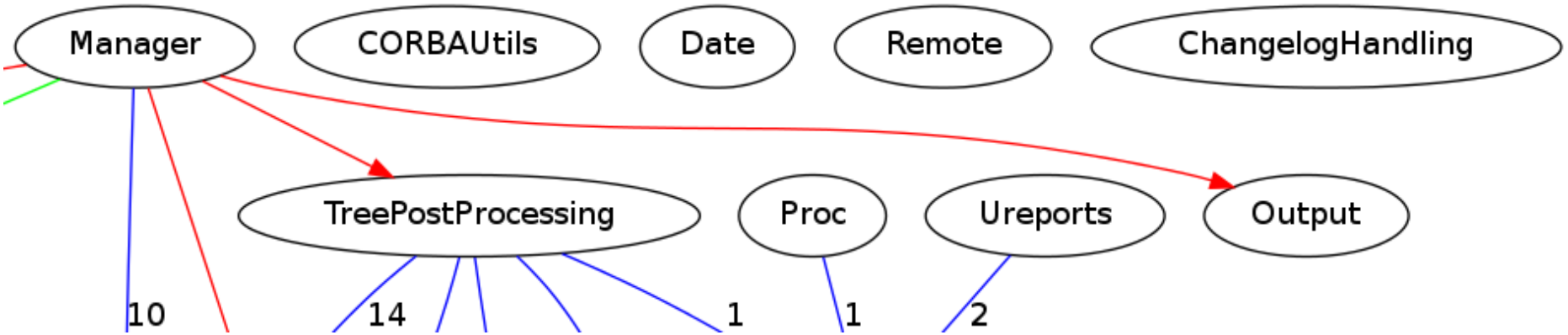


- При сборке rpm-пакетов используется возможность переопределить окружение переданное сборщику
- При сборке других типов пакетов это не используется
- Это может говорить о плохом проектировании (возможно)

Модель отражения для logilab-astng



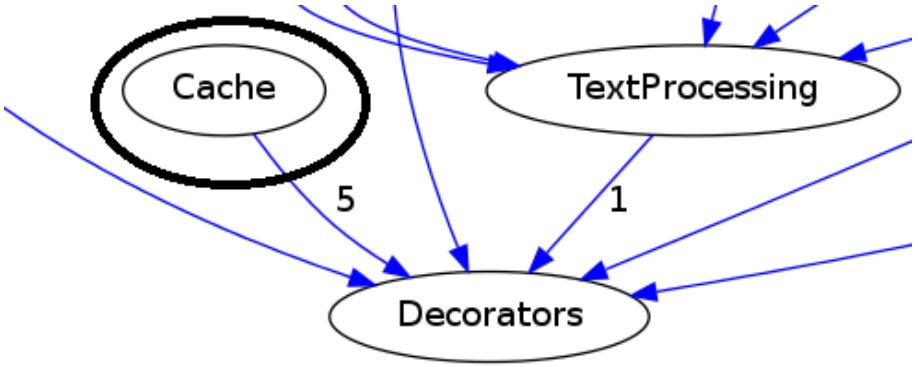
Знание



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

Еще знание

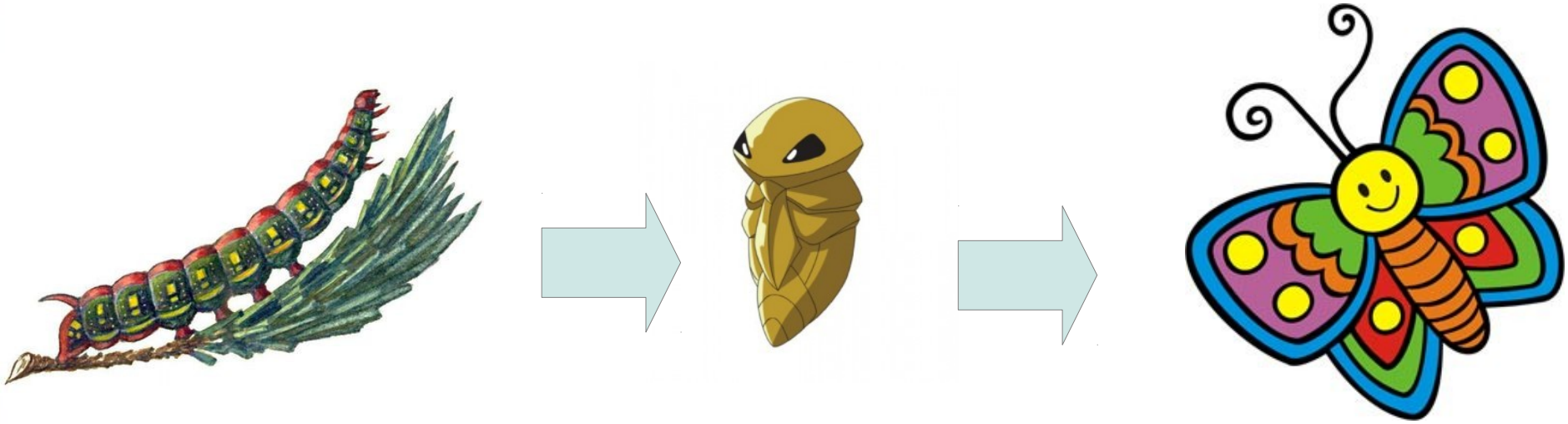
- Компонент Cache не используется внутри проекта (не связан с кэшированием в Manager)



Модели отражения позволяют

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

Промежуточные представления



Основы подхода

- Удобный для анализа набор данных
- Типовые машинные данные
- Представление — именно то, над чем ведётся анализ

Классификация представлений по типам данных

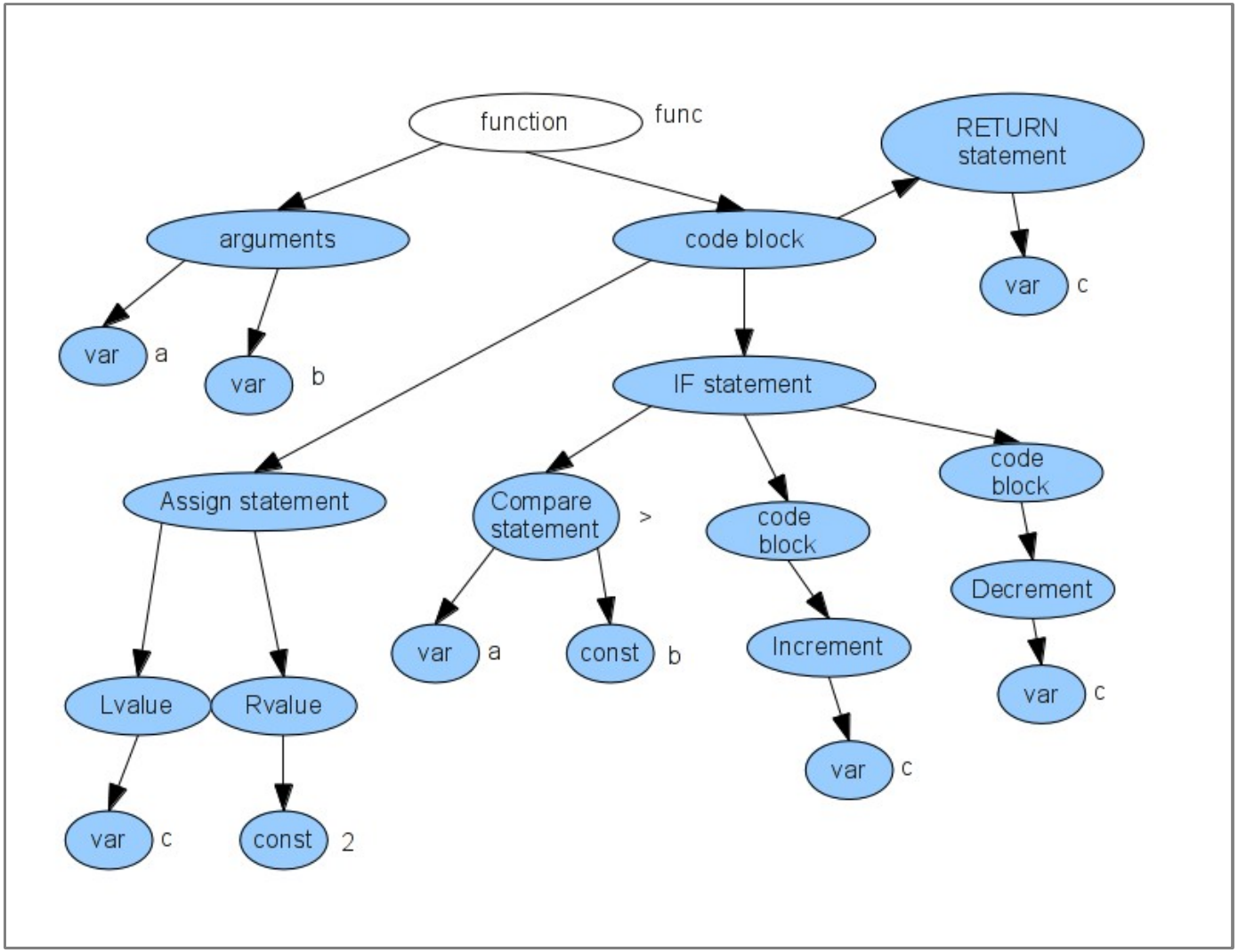
Абстрактное синтаксическое дерево (AST)

- 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

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

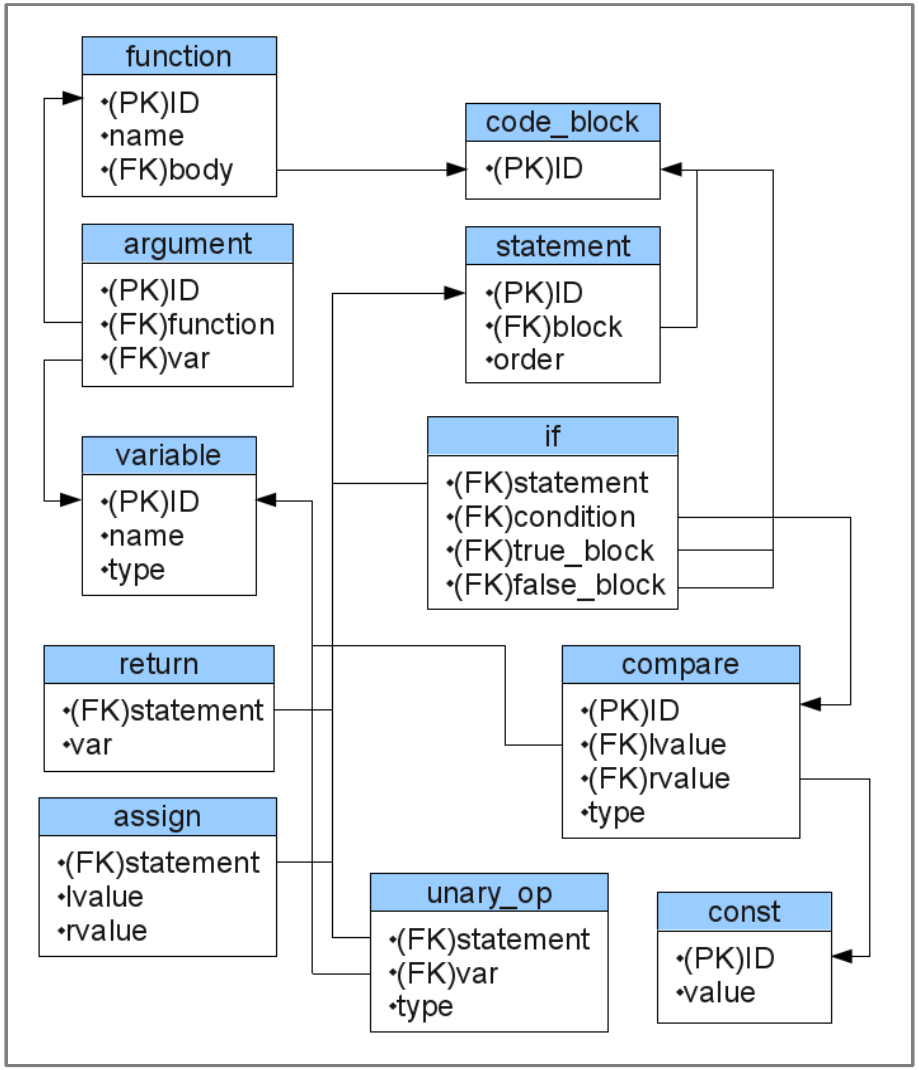
Реляционное представление

- 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.

Инструменты, использующие реляционное представление

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

Использование логических языков

- В качестве языков запросов часто используют логические языки
- Более удобны для рекурсивных запросов
- Позволяют легко вычислять транзитивные замыкания

Автоматное представление

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

Характеристика автоматного представления

- Верификация
- Огромные аппаратные затраты
- Почти 100% результат
- Покрытие всех участков кода

Специфичные представления

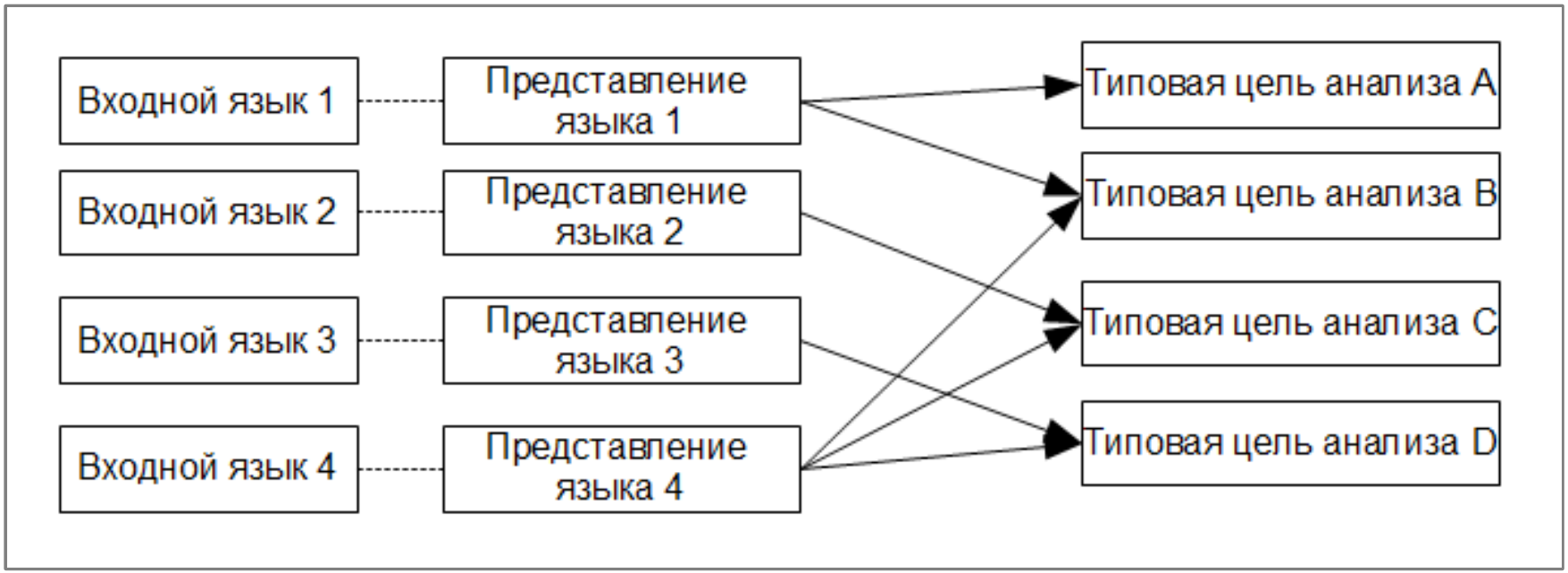
- FindBugs findbugs.sourceforge.net
- Soot sable.mcgill.ca/soot/
- Представление Java байт-кода в виде дерева

Классификация представлений по виду организации

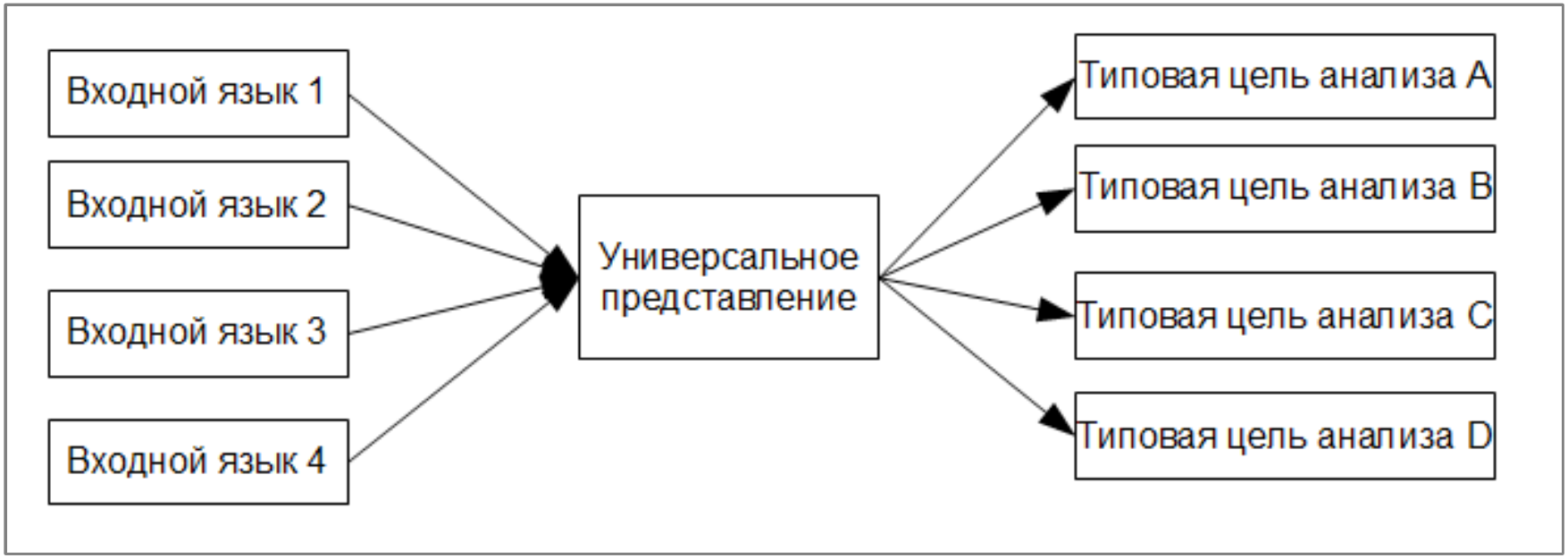
Универсальные и частные

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

Частные представления



Универсальные представления



Перспективы использования универсальных представлений

- Формирование модели как отдельное направление слабо развито
- Позволяют охватить больше входных языков
- Сокращение затрат на анализ

Многоуровневые представления

- Лучшее из представлений разной природы
- Разные задачи — разные представления

Пример многоуровневых представлений

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

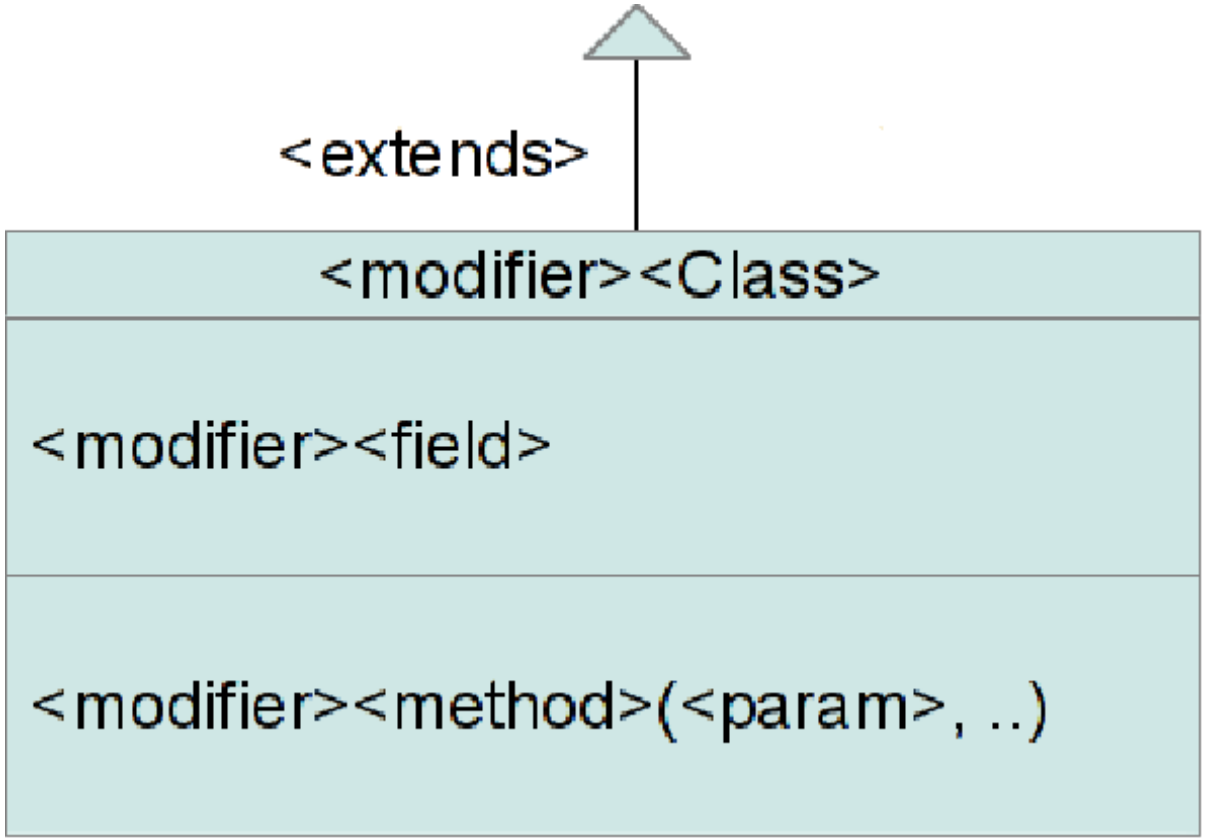
Предположение

- Чем выше уровень абстракции, тем более общим может быть представление
- Использовать универсальные представления на архитектурном уровне

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)
```

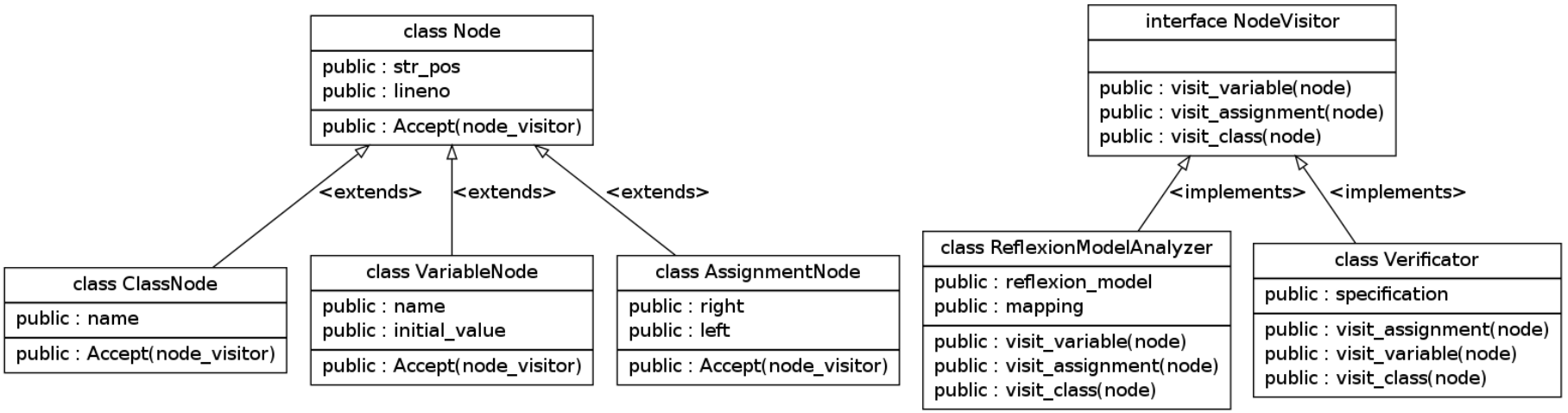
```
class NodeVisitor():
        def visit_class(self,node):
            pass
        def visit_assignment(self,node):
            pass
        def visit_variable(self,node):
            pass

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");
        }
}
```

Промежуточное представление



Достоинства полученного универсального представления

- Ниже затраты на выполнение анализа
- Точно описывает систему на своем уровне абстракции
- Добавление нового языка — одна операция

Спасибо за внимание!

Вопросы ?...