Bad smell detection

Jiacheng Qi

January 7, 2020

1 Introduction

This project aims at using ontology to detect bad smells in java code[1].

2 Ontology creation

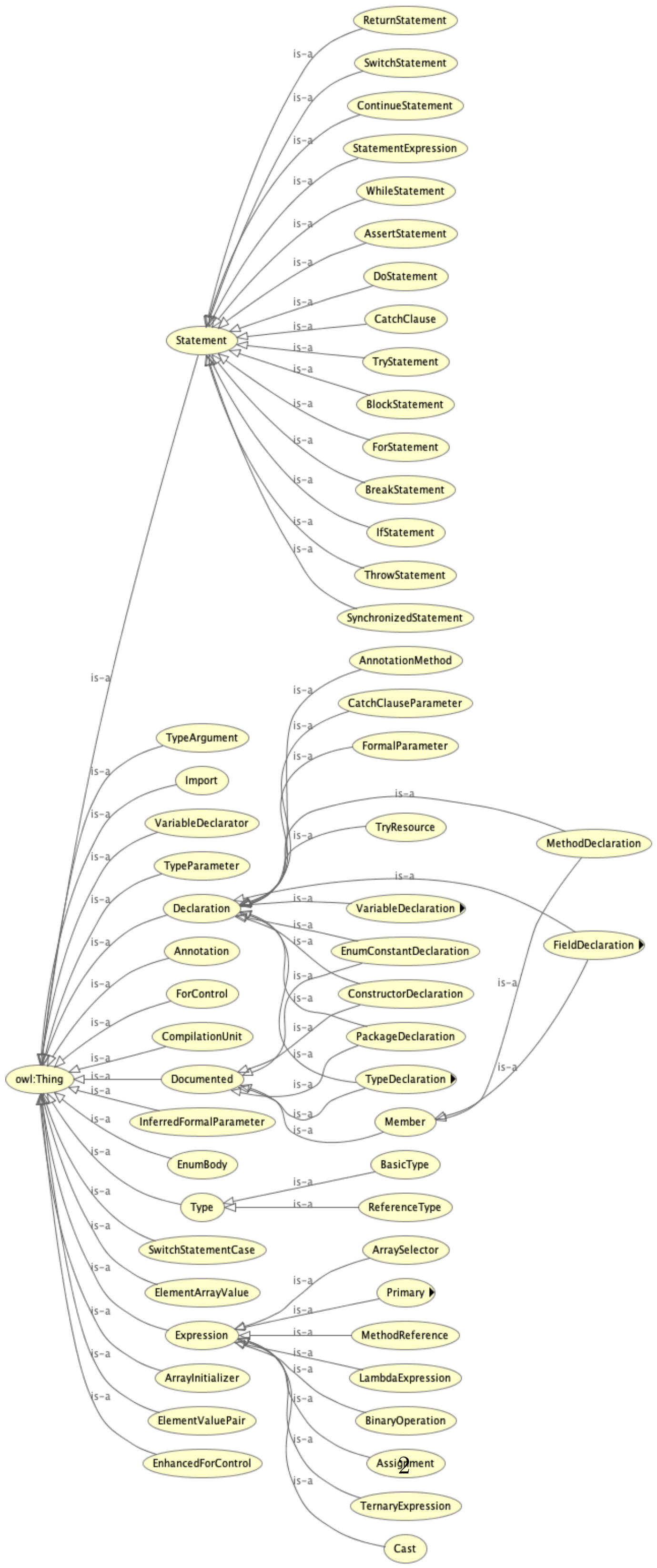
2.1 code description

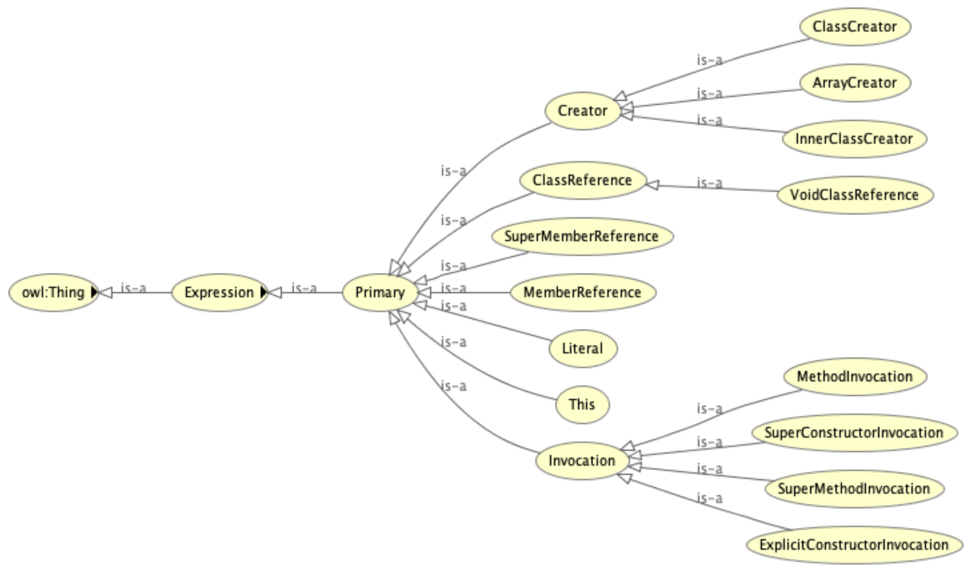
First create a empty ontology by using `get_ontolog`, and then uses `ast.parse` to parse the `tree.py`file and get the abstract syntax tree. Afterwards, by using class `Visitor` to visit the AST. During the visit of AST, if the type of the nodes matched with `Classdef`, then populate `Thing` in the ontology for the superclass Node(id matched with `Node`) or populate the id of the node in the ontology. During the visit of each `Classdef` create a new property in th ontology for each `Assign` in the body. Given an expression in body, if type(x) is ast.Assign, iterate over x.value.elts and use attribute s of each element to get a string representation of the elements in the right hand side tuple of the assignment. In order to create a property for each of them, use `new_class("property", (ObjectProperty,))`. For object properties, use `new_class("property", (DataProperty,))`. For datatype properties, rename property "name" to "jname" to avoid conflicts with the predefined "name" attribute of ontology instances. Finally save the ontology to tree.owl.

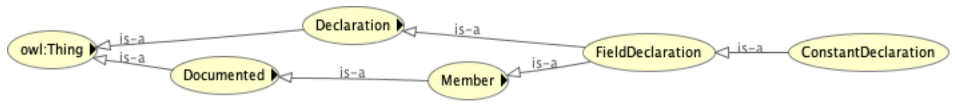
2.2 Class hierarchy

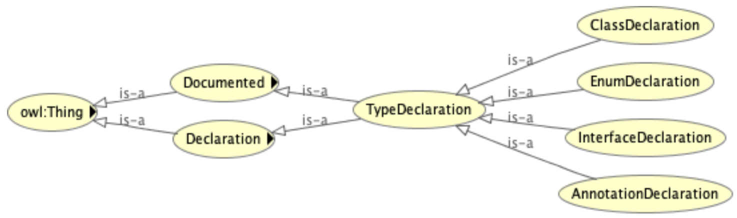
The whole class hierarchy is shown as follow:

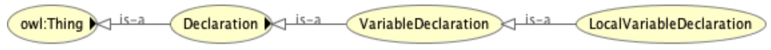
[1] https://github.com/jcarolus/android-chess


owl:Thing
Statement
ReturnStatement
SwitchStatement
ContinueStatement
StatementExpression
WhileStatement
AssertStatement
DoStatement
CatchClause
TryStatement
BlockStatement
ForStatement
BreakStatement
IfStatement
ThrowStatement
SynchronizedStatement
TypeArgument
Import
VariableDeclarator
TypeParameter
Declaration
AnnotationMethod
CatchClauseParameter
FormalParameter
TryResource
MethodDeclaration
VariableDeclaration
EnumConstantDeclaration
FieldDeclaration
ConstructorDeclaration
PackageDeclaration
TypeDeclaration
Member
Annotation
ForControl
CompilationUnit
Documented
InferredFormalParameter
EnumBody
Type
BasicType
ReferenceType
SwitchStatementCase
ElementArrayValue
Expression
ArraySelector
Primary
MethodReference
LambdaExpression
BinaryOperation
Assignment
TernaryExpression
Cast
ArrayInitializer
ElementValuePair
EnhancedForControl
is-a









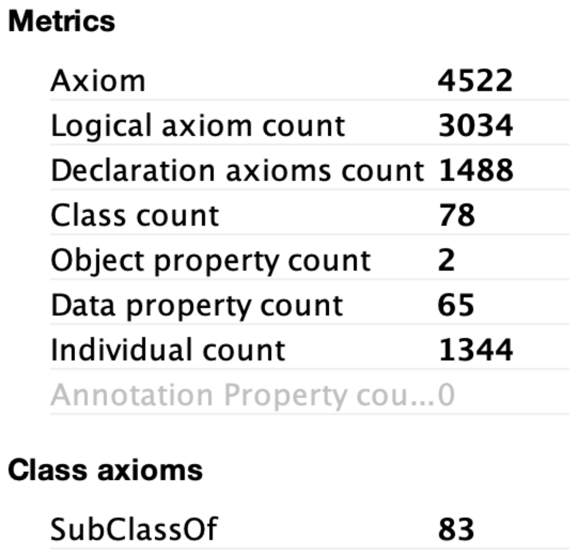
As we can see, the resulting class hierarchy provides capability to modeling abstract syntax tree for java files in a more explicit way.

3 Creation of ontology instances

3.1 code description

First load the ontology from the previous class, then iterate through the given repository. If there are files' name end with `.java`, extract the file path and use `javalang` to parse the java files. For each `ClassDeclaration`in the javalang parse tree, create an instance of ontology class by using `cd=onto['ClassDeclaration']()`. For each class member(MethodDeclaration/FieldDeclaration/ConstructorDeclaration) in the body of a `ClassDeclaration`, create a MethodDeclaration/FieldDeclaration/ ConstructorDeclaration instance and append the member instance to the property "body" of the ClassDeclaration instance. In order to access the each node's body, uses`node.body`, so we only add the right declaration in case of adding inner classes declaration to ouside classes. For the type node euquals to method and constructor declaration, we also add parameters and statements.

3.2 individuals statistics



Metrics

Axiom	4522
Logical axiom count	3034
Declaration axioms count	1488
Class count	78
Object property count	2
Data property count	65
Individual count	1344
Annotation Property cou...	0

Class axioms

SubClassOf	83

4 Bad smells detection

4.1 code description

First query selects all the methods and constructor which contain different kinds of statements(?s a/rdfs:subClassOf* tree:Statement), by adding an constrain that number of statements in each method is more than 20. In this case i also select all the classes and their name to provide more explicit output.

Second query selects all the methods and classes, then count the number of methods for each class, finally filter the count of the methods is more than 10.

Third query selects all the methods and constructor which contain at least one switch statements(in this case just ?s a tree:SwitchStatement . which is different from the first query, since we only need one type of statement instead of different kinds of statements), by adding an constrain that number of switch statement in each method or constructor is more than or equal to 1.

Fourth query selects all the methods and constructor which contain more than 5 parameters.

Fifth query selects all the classes which contain the name of methods start with set or get by using `FILTER (regex(?mn, "set.*"))`, then get the number of methods for each class without filter to compare which class only contains setter/getter.

4.2 bad smells

`long method`

PGNProvider : insert : 31
ChessPuzzleProvider : query : 25
ChessPuzzleProvider : insert : 20
GameControl : loadPGNHead : 26
GameControl : loadPGNMoves : 96
GameControl : requestMove : 76
GameControl : getDate : 26
JNI : newGame : 35
JNI : initFEN : 88
JNI : initRandomFisher : 87

`large class`

GameControl : 63
JNI : 44
Move : 21

`MethodWithSwitch`
PGNProvider : query : 1
PGNProvider : getType : 1
PGNProvider : delete : 1
PGNProvider : update : 1
ChessPuzzleProvider : query : 1
ChessPuzzleProvider : getType : 1
ChessPuzzleProvider : delete : 1
ChessPuzzleProvider : update : 1

`MethodWithLongParameterList`
PGNProvider : query : 5
ChessPuzzleProvider : query : 5
GameControl : addPGNEntry : 5
JNI : setCastlingsEPAnd50 : 6

`dataclass`

Valuation

5 Appendix: Python code

Listing 1: create ontology

```
from owlready2 import *
import ast
import types

ontuple = ()

with open("../dir/tree.py", "r") as source:
    tree = ast.parse(source.read())

onto = get_ontology("http://test.org/tree.owl")



class Visitor(ast.NodeVisitor):
    def generic_visit(self, node):
        print(type(node).__name__)
        ast.NodeVisitor.generic_visit(self, node)
        if type(node) == ast.ClassDef:
            for a in node.bases:
                if a.id == "Node":
                    with onto:
                        types.new_class(node.name, (Thing,))
                else:
                    with onto:
                        types.new_class(node.name, (onto[a.id],))

        if type(node) == ast.Assign:
            for b in node.value.elts:
                if b.s != "body" and b.s != "parameters" and b.s != "name":
                    with onto:
                        types.new_class(b.s, (DataProperty,))
                if b.s == "name":
                    with onto:
                        types.new_class("jname", (DataProperty,))
                if b.s == "body" or b.s == "parameters":
                    with onto:
                        types.new_class(b.s,(ObjectProperty,))


visitor = Visitor()
visitor.visit(tree)
print(onto)
onto.save("tree.owl", format="rdfxml")
```

Listing 2: create ontology instances

```
import os
import javalang.tree
import owlready2

onto = owlready2.get_ontology("tree.owl").load()
file_repo = '../dir/android-chess/app/src/main/java/jwtc/chess'


def extract_field(field, cd):
    for f in field.declarators:
        fd = onto['FieldDeclaration']()
        fd.jname = [f.name]
        cd.body.append(fd)


def extract_statements(node, fd):
    for _ in node.parameters:
        fp = onto['FormalParameter']()
        fd.parameters.append(fp)


def extract_parameters(node, fd):
    for _, statement in node.filter(javalang.tree.Statement):
        if type(statement) != javalang.tree.Statement:
            s_type = statement.__class__.__name__
            s = onto[s_type]()
            fd.body.append(s)


def extract_function(node):
    f_type = node.__class__.__name__
    fd = onto[f_type]()
    fd.jname = [node.name]
    extract_parameters(node, fd)
    extract_statements(node, fd)
    return fd


def extract_member(node, cd):
    for member in node.body:
        if type(member) == javalang.tree.FieldDeclaration:
            extract_field(member, cd)
        elif type(member) in [javalang.tree.MethodDeclaration,
        javalang.tree.ConstructorDeclaration]:
            cd.body.append(extract_function(member))



def class_def(tree):
    for _, node in tree.filter(javalang.tree.ClassDeclaration):
        cd = onto['ClassDeclaration']()
        cd.jname = [node.name]
        extract_member(node, cd)
```

```
for file in os.listdir(file_repo):
    if file.endswith('.java'):
        file_path = os.path.join(file_repo, file)
        with open(file_path, 'rt') as jfile:
            class_def(javalang.parse.parse(jfile.read()))
onto.save("tree2.owl", format="rdfxml")
```

Listing 3: detect bad smells

```
from owlready2 import *
import rdflib.plugins.sparql as sq

onto = get_ontology("file:///Users/jq/Desktop/bad_smell/code/tree2.owl").load()
graph = default_world.as_rdflib_graph()



queries

1.1
q = sq.prepareQuery(
    """SELECT ?cn ?mn ?s (COUNT(*)AS ?tot) WHERE {
    ?c a tree:ClassDeclaration .
    ?c tree:jname ?cn .
    ?c tree:body ?m .
    ?m a tree:MethodDeclaration .
    ?m tree:jname ?mn .
    ?m tree:body ?s .
    ?s a/rdfs:subClassOf* tree:Statement .
    } GROUP BY ?m
    HAVING (COUNT(?s) >= 20)    """,
    initNs={"tree": "http://test.org/tree.owl#"}
)
sys.stdout = open("11.txt", "w")
for row in graph.query(q):
    print(row.cn, ":", row.mn, ":", int(row.tot))
sys.stdout.close()

1.2
q = sq.prepareQuery(
    """SELECT ?cn ?on ?s (COUNT(*)AS ?tot) WHERE {
    ?c a tree:ClassDeclaration .
    ?c tree:jname ?cn .
    ?c tree:body ?o .
    ?o a tree:ConstructorDeclaration .
    ?o tree:jname ?on .
    ?o tree:body ?s .
    ?s a/rdfs:subClassOf* tree:Statement .
    } GROUP BY ?o""",
    initNs={"tree": "http://test.org/tree.owl#"}
)
print("Long_constructor")
sys.stdout = open("12.txt", "w")
for row in graph.query(q):
    print(row.cn, ":", row.on, ":", int(row.tot))
sys.stdout.close()

2

q = sq.prepareQuery(
    """SELECT ?mn ?cn (COUNT(*)AS ?tot) WHERE {
    ?c a tree:ClassDeclaration .
    ?c tree:jname ?cn .
    ?c tree:body ?m .
    ?m a tree:MethodDeclaration .
    ?m tree:jname ?mn .
    } GROUP BY ?cn
    HAVING (COUNT(?m) >= 10)""",
    initNs={ "tree": "http://test.org/tree.owl#" }
)
# print("Large classess: ")
sys.stdout = open("2.txt", "w")

for row in graph.query(q):
    print(row.cn, ":", int(row.tot))
sys.stdout.close()


3.1
q = sq.prepareQuery(
    """SELECT ?cn ?mn ?s (COUNT(*)AS ?tot) WHERE {
    ?c a tree:ClassDeclaration .
    ?c tree:jname ?cn .
    ?c tree:body ?m .
    ?m a tree:MethodDeclaration .
    ?m tree:jname ?mn .
    ?m tree:body ?s .
    ?s a tree:SwitchStatement .
    } GROUP BY ?m
    HAVING (COUNT(?s) >= 1)     """,
    initNs={"tree": "http://test.org/tree.owl#"}
)
sys.stdout = open("31.txt", "w")
for row in graph.query(q):
    print(row.cn, ":", row.mn, ":", int(row.tot))
sys.stdout.close()


3.2
q = sq.prepareQuery(
    """SELECT ?cn ?on ?s (COUNT(*)AS ?tot) WHERE {
    ?c a tree:ClassDeclaration .
    ?c tree:jname ?cn .
    ?c tree:body ?o .
    ?o a tree:ConstructorDeclaration .
    ?o tree:jname ?on .
    ?o tree:body ?s .
    ?s a tree:SwitchStatement .
    } GROUP BY ?o
    HAVING (COUNT(?s) >= 1)     """,
    initNs={"tree": "http://test.org/tree.owl#"}
)
sys.stdout = open("32.txt", "w")
for row in graph.query(q):

    print(row.cn, ":", row.on, ":", int(row.tot))
sys.stdout.close()
```

4.1

```
q = sq.prepareQuery(
    """SELECT ?cn ?mn ?s (COUNT(*)AS ?tot) WHERE {
    ?c a tree:ClassDeclaration .
    ?c tree:jname ?cn .
    ?c tree:body ?m .
    ?m a tree:MethodDeclaration .
    ?m tree:jname ?mn .
    ?m tree:parameters ?s .
    ?s a tree:FormalParameter .
    } GROUP BY ?m
    HAVING (COUNT(?s) >= 5)     """,
    initNs={"tree": "http://test.org/tree.owl#"}
)
sys.stdout = open("41.txt", "w")
for row in graph.query(q):
    print(row.cn, ":", row.mn, ":", int(row.tot))
sys.stdout.close()
```

4.2

```
q = sq.prepareQuery(
    """SELECT ?cn ?mn ?s (COUNT(*)AS ?tot) WHERE {
    ?c a tree:ClassDeclaration .
    ?c tree:jname ?cn .
    ?c tree:body ?m .
    ?m a tree:ConstructorDeclaration .
    ?m tree:jname ?mn .
    ?m tree:parameters ?s .
    ?s a tree:FormalParameter .
    } GROUP BY ?m
    HAVING (COUNT(?s) >= 5)     """,
    initNs={"tree": "http://test.org/tree.owl#"}
)
sys.stdout = open("42.txt", "w")
for row in graph.query(q):
    print(row.cn, ":", row.mn, ":", int(row.tot))
sys.stdout.close()
```

5

```
q = sq.prepareQuery(
    """SELECT ?mn ?cn (COUNT(*)AS ?tot) WHERE {
    ?c a tree:ClassDeclaration .
    ?c tree:jname ?cn .
    ?c tree:body ?m .
    ?m a tree:MethodDeclaration .
    ?m tree:jname ?mn .
    FILTER (regex(?mn, "get.*"))
    } GROUP BY ?cn""",
    initNs={ "tree": "http://test.org/tree.owl#" }
)
sys.stdout = open("5.txt", "w")

for row in graph.query(q):
    print(row.cn, ":", row.mn,":", int(row.tot))
sys.stdout.close()


q = sq.prepareQuery(
    """SELECT ?mn ?cn (COUNT(*)AS ?tot) WHERE {
    ?c a tree:ClassDeclaration .
    ?c tree:jname ?cn .
    ?c tree:body ?m .
    ?m a tree:MethodDeclaration .
    ?m tree:jname ?mn .
    FILTER (regex(?mn, "set.*"))
    } GROUP BY ?cn""",
    initNs={ "tree": "http://test.org/tree.owl#" }
)
sys.stdout = open("5.txt", "w")

for row in graph.query(q):
    print(row.cn, ":", row.mn,":", int(row.tot))
sys.stdout.close()


q = sq.prepareQuery(
    """SELECT ?mn ?cn (COUNT(*)AS ?tot) WHERE {
    ?c a tree:ClassDeclaration .
    ?c tree:jname ?cn .
    ?c tree:body ?m .
    ?m a tree:MethodDeclaration .
    ?m tree:jname ?mn .
    } GROUP BY ?cn""",
    initNs={ "tree": "http://test.org/tree.owl#" }
)
sys.stdout = open("5.txt", "w")

for row in graph.query(q):
    print(row.cn, ":", row.mn,":", int(row.tot))
sys.stdout.close()
```