

Experiments and Results

Mehmet Kaya

Department of computer Science and Electrical
Engineering
Syracuse University
Syracuse, NY, USA
mkaya@syr.edu

James W. Fawcett

Department of computer Science and Electrical
Engineering
Syracuse University
Syracuse, NY, USA
jfawcett@twcny.rr.com

This document demonstrates three of the experiments that we performed for our paper entitled "Identifying Extract Method Opportunities Based on Variable References". Results of these experiments are supported with a source code analysis tool we developed. Here we demonstrate, by giving the original and modified code for the methods that the proposed technique in our paper works effectively with the analysis and visualization tools we developed.

The tools that we developed for our experiments use a rule based ad-hoc parser. This parser analyzes source code extracting only the information we need using rules that work on specific collections of tokens we call "SemiExpressions". We chose this approach over using a traditional parser generator because the results we seek depend on only a small part of the (C++) language grammar.

Experiment 1

First method used for our experiments is shown in Figure 2. Figure 2 shows the placement tree for the method given in Figure 2.

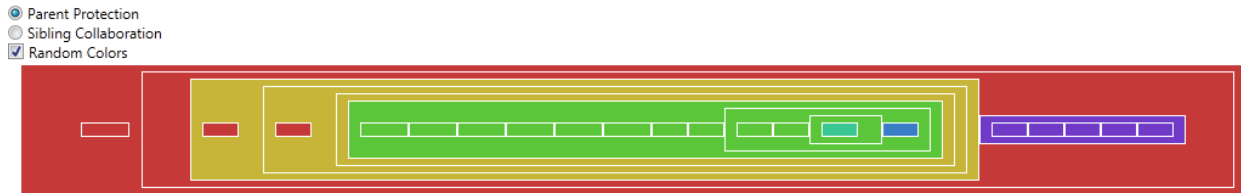


Figure 1 Placement Tree of Original Method

```

1 void doAction(ITokCollection *& pTc){
2     ITokCollection & tc = * pTc ;
3     int _for = tc.find("for");
4     int _if = tc.find("if");
5     int _while = tc.find("while");
6     if(_for < tc.length()|| _if < tc.length()|| _while < tc.length()){
7         DataAnalWithoutEq(pTc);
8         return ;
9     }
10    int eq = tc.find("=");
11    if(eq < tc.length() && tc[tc.length()-1] != "{"){
12        int p, sp, tp ;
13        p = sp = tp = 0 ;
14        bool varFound = false ;
15        if(eq == 1){
16            if(! isSpecialChar(tc[0])){
17                AddAReference(tc[0]);
18            }
19        }
20        else {
21            if(eq == 2){
22                if(! isSpecialChar(tc[1])){
23                    AddAReference(tc[1]);
24                }
25            }
26            else {
27                if(eq > 2){
28                    for(int i = eq-1 ; i >= 0 ; i--){
29                        if(tc[i] == "]"){
30                            sp++;
31                            continue ;
32                        }
33                        if(tc[i] == "="){
34                            p++;
35                            continue ;
36                        }
37                        if(tc[i] == ">"){
38                            tp++;
39                            continue ;
40                        }
41                        if(tc[i] == "["){
42                            sp--;
43                            continue ;
44                        }
45                        if(tc[i] == "("){
46                            p--;
47                            continue ;
48                        }
49                        if(tc[i] == "<"){
50                            tp--;
51                            continue ;
52                        }
53                    }
54                }
55            }
56        }
57    }
58    if(i >= 1 && tc[i-1] == "->"){
59        continue ;
60    }
61    if(i >= 1 && tc[i-1] == "."){
62        continue ;
63    }
64    if(i < tc.length()-1 && tc[i+1] != "("){
65        if(i > 0 && tc[i-1] == ":::"){
66            continue ;
67        }
68        if(i < tc.length()-1 && tc[i+1] == ":::"){
69            continue ;
70        }
71    }
72    if(! isSpecialChar(tc[i])){
73        if(p == 0 && sp == 0 && tp == 0){
74            varFound = true ;
75        }
76        AddAReference(tc[i]);
77    }
78    if(varFound){
79        break ;
80    }
81    }
82    }
83    }
84    }
85    }
86    }
87    }
88    }
89    }
90    }
91    }
92    }
93    }
94    }
95    }
96    }
97    }
98    }
99    }
100    }
101    }

```

Figure 2 Original Method

Restructured version of the method given in Figure 2 is shown in Figure 3.

```

1 void BeforeEqualSignProcessing(int eq, ITokCollection& tc, int sp,
2     int p, int tp, bool varFound)
3 {
4     for(int i = eq-1 ; i >= 0 ; i--){
5         if(tc[i]==""){
6             sp++;
7             continue ;
8         }
9         if(tc[i]==" "){
10            p++;
11            continue ;
12        }
13        if(tc[i]==">"){
14            tp++;
15            continue ;
16        }
17        if(tc[i]=="["){
18            sp--;
19            continue ;
20        }
21        if(tc[i]=="("){
22            p--;
23            continue ;
24        }
25        if(tc[i]=="<"){
26            tp--;
27            continue ;
28        }
29        if(i >= 1 && tc[i-1]=="->"){
30            continue ;
31        }
32        if(i >= 1 && tc[i-1]=="."){
33            continue ;
34        }
35        if(i < tc.length()-1 && tc[i+1]!="("){
36            if(i > 0 && tc[i-1]=="::"){
37                continue ;
38            }
39            if(i < tc.length()-1 && tc[i+1]=="::"){
40                continue ;
41            }
42            if(! isSpecialChar(tc[i])){
43                if(p == 0 && sp == 0 && tp == 0){
44                    varFound = true ;
45                }
46                AddARefrence(tc[i]);
47            }
48            if(varFound){
49                break ;
50            }
51        }
52    }
53 }
54 void AfterEqualSignProcessing(int eq, ITokCollection& tc)
55 {
56     for(int i = eq ; i < tc.length(); i++){
57         if(i >= 1 && tc[i-1]=="->"){
58             continue ;
59         }
60         if(i >= 1 && tc[i-1]=="."){
61             continue ;
62         }
63         if(i < tc.length()-1 && tc[i+1]=="("){
64             continue ;
65         }
66         if(i < tc.length()-1 && tc[i+1]=="::"){
67             continue ;
68         }
69         if(! isSpecialChar(tc[i])){
70             AddARefrence(tc[i]);
71         }
72     }
73 }
74 void EqualSignProcesing(int eq, ITokCollection& tc, int sp,
75     int p, int tp, bool varFound)
76 {
77     if(eq == 1){
78         if(! isSpecialChar(tc[0])){
79             AddARefrence(tc[0]);
80         }
81     }
82     else {
83         if(eq == 2){
84             if(! isSpecialChar(tc[1])){
85                 AddARefrence(tc[1]);
86             }
87         }
88         else {
89             if(eq > 2){
90                 BeforeEqualSignProcessing(eq, tc, sp, p, tp, varFound);
91             }
92         }
93     }
94 }
95 void doAction(ITokCollection *pTc){
96     ITokCollection & tc = * pTc ;
97     int _for = tc.find("for");
98     int _if = tc.find("if");
99     int _while = tc.find("while");
100    if(_for < tc.length() || _if < tc.length() || _while < tc.length()){
101        DataAnalWithoutEq(pTc);
102        return ;
103    }
104    int eq = tc.find("=");
105    if(eq < tc.length() && tc[tc.length()-1] != "{"){
106        int p,sp,tp ;
107        p = sp = tp = 0 ;
108        bool varFound = false ;
109        EqualSignProcesing(eq, tc, sp, p, tp, varFound);
110        AfterEqualSignProcessing(eq, tc);
111    }
112    else {
113        DataAnalWithoutEq(pTc);
114    }
115 }

```

Figure 1 Restructured Version For Experiment 1

After restructuring we run our tool and get the following placement tree for the new version of the code.

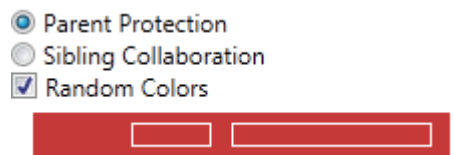


Figure 2 Original Method After Refactoring

Following figures show the placement trees for the new extracted methods.

- Parent Protection
- Sibling Collaboration
- Random Colors



- Parent Protection
- Sibling Collaboration
- Random Colors



- Parent Protection
- Sibling Collaboration
- Random Colors



Experiment 2

Second method used for our experiments is shown in Figure 5.

```

1 void doAction(ITokCollection & pTc){
2     if(hasOnlyOneItem(pTc)){
3         return ;
4     }
5     ITokCollection & tc = * pTc ;
6     if(tc.length() > 0 && tc[0] == "a"){
7         return ;
8     }
9     int eq = tc.find("a");
10    if(eq > tc.length()-1 && tc.length() >= 2 && tc[tc.length()-1] == ";" && tc[tc.length()-2] == "){
11        return ;
12    }
13    if(tc.find("for") < tc.length()){
14        int deletel = 0 ;
15        int _for = tc.find("for");
16        int _eq = tc.find("=");
17        int _sc = tc.find(";");
18        int StartIndex ;
19        if(_eq < _sc && _eq < tc.length()){
20            StartIndex = _eq ;
21        }
22        else {
23            if(_sc < _eq && _sc < tc.length()){
24                StartIndex = _sc ;
25            }
26            else {
27                return ;
28            }
29        }
30        if(StartIndex == _for+3){
31            return ;
32        }
33        for(int i = StartIndex-1 ; i >= 0 ; i--){
34            if(! isSpecialChar(tc[i])){
35                std::string type ;
36                for(int j = _for+2 ; j < i ; j++){
37                    type.append(tc[j]);
38                    type.append(" ");
39                }
40                FoundLocal(type,tc[i]);
41                return ;
42            }
43        }
44    }
45    if((tc[tc.length()-1] == ";" && eq < tc.length()) || (tc[tc.length()-1] == "(" && eq == tc.length()-2)){
46        int p = 0 ;
47        int s = 0 ;
48        for(int i = eq ; i >= 0 ; i--){
49            if(tc[i] == ")"){
50                p++;
51            }
52            else {
53                if(tc[i] == "("){
54                    p++;
55                }
56                else {
57                    if(tc[i] == "["){
58                        s++;
59                    }
60                    else {
61                        if(tc[i] == "]"){
62                            s--;
63                        }
64                    }
65                }
66            }
67        }
68        if(p == 0 && s == 0){
69            if(i == 1){
70                return ;
71            }
72            if(i == 2 && ! isSpecialChar(tc[i-1]) && (tc[i-2] == "" || tc[i-2] == ">" || tc[i-2] == "&")){
73                return ;
74            }
75            if(i > 1 && ! isSpecialChar(tc[i-1]) && (tc[i-2] == "" || tc[i-2] == ">" || tc[i-2] == "&")){
76                int t = tc.find("");
77                if(t < tc.length() && t > 0){
78                    FoundLocal(GetFullType(pTc,t-1),tc[i-1]);
79                    break ;
80                }
81            }
82            else {
83                int star = tc.find("");
84                int amp = tc.find("&");
85                int typeIndex ;
86                if(star < amp){
87                    typeIndex = star ;
88                }
89                else {
90                    typeIndex = amp ;
91                }
92                if(typeIndex < tc.length() && ! isKey(tc[typeIndex-1])){
93                    FoundLocal(GetFullType(pTc,typeIndex-1),tc[i-1]);
94                    break ;
95                }
96            }
97        }
98        if(i > 1 && ! isSpecialChar(tc[i-1]) && ! isKey(tc[i-2])){
99            FoundLocal(GetFullType(pTc,i-2),tc[i-1]);
100            break ;
101        }
102    }
103    else {
104        if(eq > tc.length()-1 && tc[tc.length()-1] == ";"){
105            std::vector < std::string > temp ;
106            std::string type = "unknown" ;
107            for(int i = tc.length()-1 ; i >= 0 ; i--){
108                if(! isSpecialChar(tc[i])){
109                    temp.push_back(tc[i]);
110                }
111                if(tc[i] == "" || tc[i] == "&" || tc[i] == ">"){
112                    int star = tc.find("");
113                    int amp = tc.find("&");
114                    int typeIndex ;
115                    if(star < amp){
116                        typeIndex = star ;
117                    }
118                    else {
119                        typeIndex = amp ;
120                    }
121                    if(typeIndex < tc.length() && ! isKey(tc[typeIndex-1])){
122                        type = GetFullType(pTc,typeIndex-1);
123                        break ;
124                    }
125                }
126                if(i == 1){
127                    if(! isKey(tc[0])){
128                        type = GetFullType(pTc,0);
129                    }
130                    break ;
131                }
132            }
133            for(int i = 0 ; i < temp.size(); i++){
134                FoundLocal(type,temp[i]);
135            }
136        }
137    }
138    }
139 }

```

Figure 3 Original Method

Figure 6 shows the placement tree for the method given in Figure 5.



Figure 4 Placement Tree for Experiment 2

Restructured version of the method given in Figure 5 is shown in Figure 7.

```

1  bool FindStartIndex(int &startIndex, int& _eq,
2  int& _sc, ITokCollection & tc)
3  {
4  if(_eq < _sc && _eq < tc.length()){
5  StartIndex = _eq;
6  }
7  else {
8  if(_sc < _eq && _sc < tc.length()){
9  StartIndex = _sc;
10 }
11 }
12 else {
13 return true;
14 }
15 }
16 return false;
17 }
18 bool FindDataBeforeStartIndex(int& startIndex,
19 int& _for, ITokCollection & tc)
20 {
21 for(int i = StartIndex-1; i >= 0; i--){
22 if(! isSpecialChar(tc[i])){
23 std::string type;
24 for(int j = _for+2; j < i; j++){
25 type.append(tc[j]);
26 type.append(" ");
27 }
28 FoundLocal(type,tc[i]);
29 return true;
30 }
31 }
32 return false;
33 }
34 bool ProcessBeforeEqual(int& eq,ITokCollection & tc, int& p,
35 int& s, ITokCollection *& pTc)
36 {
37 for(int i = eq; i >= 0; i--){
38 if(tc[i]==""){
39 p--;
40 }
41 else {
42 if(tc[i]=="("){
43 p++;
44 }
45 else {
46 if(tc[i]=="){
47 s++;
48 }
49 else {
50 if(tc[i]==""){
51 s--;
52 }
53 }
54 }
55 if(p == 0 && s == 0){
56 if(i == 1){
57 return true;
58 }
59 if(i == 2 && ! isSpecialChar(tc[i-1])&&(tc[i-2]== "" || tc[i-2]== ">" || tc[i-2]== "&")){
60 return true;
61 }
62 if(i > 1 && ! isSpecialChar(tc[i-1])&&(tc[i-2]== "" || tc[i-2]== ">" || tc[i-2]== "&")){
63 int t = tc.find("<");
64 if(t < tc.length())&& t > 0){
65 FoundLocal(GetFullType(pTc,t-1),tc[i-1]);
66 break;
67 }
68 }
69 else {
70 int star = tc.find("");
71 int amp = tc.find("&");
72 int typeIndex;
73 if(star < amp){
74 typeIndex = star;
75 }
76 else {
77 typeIndex = amp;
78 }
79 if(typeIndex < tc.length())&& ! isKey(tc[typeIndex-1]){
80 FoundLocal(GetFullType(pTc,typeIndex-1),tc[i-1]);
81 break;
82 }
83 }
84 if(i > 1 && ! isSpecialChar(tc[i-1])&& ! isKey(tc[i-2])){
85 FoundLocal(GetFullType(pTc,i-2),tc[i-1]);
86 break;
87 }
88 }
89 }
90 return false;
91 }

```

```

92 void ProcessAStatement(ITokCollection & tc,
93 std::vector < std::string >& temp,std::string& type,ITokCollection *& pTc)
94 {
95 for(int i = tc.length()-1; i >= 0; i--){
96 if(! isSpecialChar(tc[i])){
97 temp.push_back(tc[i]);
98 }
99 }
100 if(tc[i]==" "" || tc[i]=="&" || tc[i]==">"){
101 int star = tc.find("");
102 int amp = tc.find("&");
103 int typeIndex;
104 if(star < amp){
105 typeIndex = star;
106 }
107 else {
108 typeIndex = amp;
109 }
110 if(typeIndex < tc.length())&& ! isKey(tc[typeIndex-1]){
111 type = GetFullType(pTc,typeIndex-1);
112 }
113 break;
114 }
115 if(i == 1){
116 if(! isKey(tc[0])){
117 type = GetFullType(pTc,0);
118 }
119 break;
120 }
121 }
122 }
123 void doAction(ITokCollection *& pTc){
124 if(hasOnlyOneWord(pTc)){
125 return;
126 }
127 ITokCollection & tc = * pTc;
128 if(tc.length() > 0 && tc[0]==" "){
129 return;
130 }
131 int eq = tc.find("=");
132 if(eq > tc.length()-1 && tc.length() >= 2 && tc[tc.length()-1]==" " && tc[tc.length()-2]==""){
133 return;
134 }
135 if(tc.find("for") < tc.length()){
136 int delete1 = 0;
137 int _for = tc.find("for");
138 int _eq = tc.find("=");
139 int _sc = tc.find("<");
140 int StartIndex;
141 if(FindStartIndex(StartIndex,_eq,_sc,tc)){
142 return;
143 }
144 if(StartIndex == _for+3){
145 return;
146 }
147 if(FindDataBeforeStartIndex(StartIndex,_for,tc)){
148 return;
149 }
150 if((tc[tc.length()-1]==" " && eq < tc.length())||((tc[tc.length()-1]==" (" && eq == tc.length()-2)){
151 int p = 0;
152 int s = 0;
153 if(ProcessBeforeEqual(eq,tc,p,s,pTc)){
154 return;
155 }
156 }
157 }
158 else {
159 if(eq > tc.length()-1 && tc[tc.length()-1]==" "){
160 std::vector < std::string > temp;
161 std::string type = "unknown";
162 ProcessAStatement(tc,temp,type,pTc);
163 for(int i = 0; i < temp.size(); i++){
164 FoundLocal(type,temp[i]);
165 }
166 }
167 }

```

Figure 5 Restructured Version for Experiment 2

After restructuring we run our tool and get the following placement tree for the new version of the code.

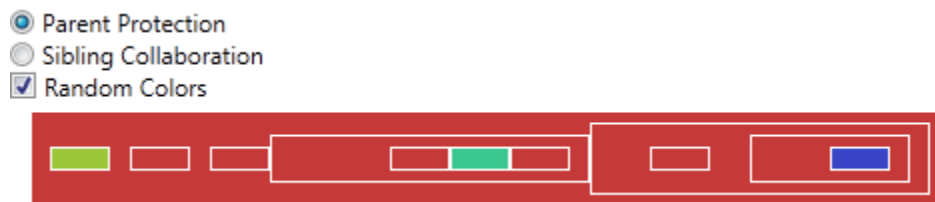


Figure 6 Original Method after Restructuring

Following figures show the placement trees for the new extracted methods.

- Parent Protection
- Sibling Collaboration
- Random Colors



- Parent Protection
- Sibling Collaboration
- Random Colors



- Parent Protection
- Sibling Collaboration
- Random Colors



- Parent Protection
- Sibling Collaboration
- Random Colors



Experiment 3

Third method used for our experiments is shown in Figure 9.

```

1 void doAction(ITokCollection *& pTc){
2     if(p_Repos->scopeStack().size()== 0){
3         return ;
4     }
5     if(BeginningOfScope::GetNonScopePar()){
6         BeginningOfScope::BackInScope();
7         return ;
8     }
9     element * elem = p_Repos->scopeStack().pop();
10    elem->setELine(p_Repos->lineCount());
11    if(elem->getType()=="function"){
12        printElement(elem);
13        std::string fn = p_Repos->Tokenizer()->getCurrFileName();
14        std::string temp = fn.substr(fn.find_last_of('\\')+1,fn.find_last_of('.')-fn.find_last_of('\\'));
15        temp.append("xml");
16        XMLCreator * xc = new XMLCreator(elem,temp);
17        delete xc ;
18    }
19    else {
20        if(p_Repos->scopeStack().size()== 0){
21            return ;
22        }
23        if(! elem->IsInTree()){
24            element * parentelem = p_Repos->scopeStack().pop();
25            parentelem->addChild(elem);
26            elem->setInTree(true);
27            p_Repos->scopeStack().push(parentelem);
28        }
29        if(elem->getName()=="if"){
30            p_Repos->setLatestIf(elem);
31        }
32    }
33 }

```

Figure 7 Original Method for Experiment 3

Figure 10 shows the placement tree for the method given in Figure 9.

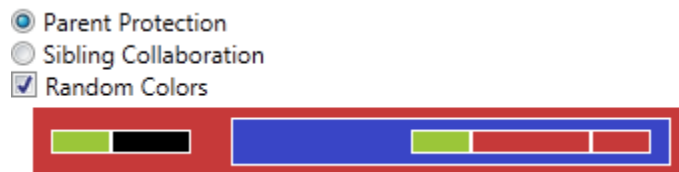


Figure 8 Placement Tree for the Original Method

Restructured version of the method given in Figure 9 is shown in Figure 11.


```

1 void ProcessElementInTree(element * elem)
2 {
3     if(! elem->IsInTree()){
4         element * parentelem = p_Repos->scopeStack().pop();
5         parentelem->addChild(elem);
6         elem->setInTree(true);
7         p_Repos->scopeStack().push(parentelem);
8     }
9     if(elem->getName()=="if"){
10        p_Repos->setLatestIf(elem);
11    }
12 }
13 }
14 bool ProcessFunctionelement(element * elem)
15 {
16     if(elem->getType()=="function"){
17         printElement(elem);
18         std::string fn = p_Repos->Token()->getCurrFileName();
19         std::string temp = fn.substr(fn.find_last_of('\\')+1,fn.find_last_of('.')-fn.find_last_of('\\'));
20         temp.append(".xml");
21         XMLCreator * xc = new XMLCreator(elem,temp);
22         delete xc ;
23     }
24     else {
25         if(p_Repos->scopeStack().size()== 0){
26             return true;
27         }
28         ProcessElementInTree(elem);
29     }
30     return false;
31 }
32 void doAction(ITokCollection *& pTc){
33     if(p_Repos->scopeStack().size()== 0){
34         return ;
35     }
36     if(BeginningOfScope::GetNonScopePar()){
37         BeginningOfScope::BackInScope();
38         return ;
39     }
40     element * elem = p_Repos->scopeStack().pop();
41     elem->setELine(p_Repos->lineCount());
42     if(ProcessFunctionelement(elem)){
43         return;
44     }
45 }

```

Figure 9 Restructured Version for Experiment 3

After restructuring we run our tool and get the following placement tree for the new version of the code.

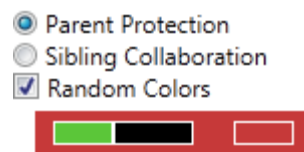


Figure 10 Original Method After Restructuring

Following figures show the placement trees for the new extracted methods.

- Parent Protection
- Sibling Collaboration
- Random Colors



- Parent Protection
- Sibling Collaboration
- Random Colors

