AOP on the C-side

Bram Adams

bram.adams@ugent.be

1. Legacy C systems and AOP

Legacy systems:
"any information system that significantly resists modification and evolution to meet new and constantly changing business requirements" [Brodie & Stonebraker ‘95]

AOP:
- quantification on properties of base program
- base program oblivious w.r.t. aspects

enable unintrusive reverse engineering of legacy systems

Industrial case study:
- C-system with 407 modules and 269 Makefiles
- feed dynamic analyses with trace generated using AOP

Problem: which C aspect framework to choose and how?

2. Requirements

Tool chain:
T1 handle various "dialects" (ANSI, K&R, GNU, ...)
[ T2 leave base program's semantics intact ]
T3 no special preparation/exploration of source code
T4 minimal preparation/exploration of build system
T5 deployable in other environments (OS, compiler, ...)

Analyses:
[ A1 well-covering execution scenario ]
A2 obtain procedure call-level data + context info
A3 record both procedure call entry and exit
[ A4 analyses need to be scalable ]

3. Comparison

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>A2</th>
<th>A3</th>
</tr>
</thead>
<tbody>
<tr>
<td>AspectC</td>
<td>?</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>AspectC++</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Aspicer</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>C4</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>WeaveC</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>µDiner</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>TinyC2</td>
<td>N/A</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Arachne</td>
<td>N/A</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>TOSKANA</td>
<td>N/A</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>TOSKANA-VM</td>
<td>+</td>
<td>?</td>
<td>-</td>
<td>?</td>
<td>-</td>
<td>-</td>
<td>?</td>
</tr>
</tbody>
</table>

4. Discussion

- requirements target worst case scenarios
- advice reuse (T3 and A2)
- makefiles are composed of crosscutting concerns (T4)
- no general-purpose AOP-workbench for C ...
- ... so C-specific issues are not covered yet

aspect tracing{
  before():
    execution(int f(.))
    | execution(char* g(.))
    | ...
    printf("before function\n");

  /* after-advice analogous */
}

<?xml version="1.0" encoding="UTF-8"?>
<aspect id="tracing">
  <pointcut id="trace_all">
  <elements files="*.c" identifier="function" data=".*"/>
  <advices>
  <advice id="trace_before" type="before"/>
  </advices>
  </pointcut>
  <advice id="trace_before" type="function_call">
  <code>
    <![CDATA[ printf("before $FUNC_NAME\n"); ]]>
  </code>
  </advice>
  </aspect>

Type around tracing(Type) on (Jp):
call(Jp,".*") & type(Jp,Type) & !str_matches("void",Type){
  Type i;
  printf("before %s in %s\n", Jp->functionName,Jp->fileName);
  i = proceed();
  fprintf(fp,"after %s in %s\n", Jp->functionName,Jp->fileName);
  return i;
}

aspect tracing{
  advice execution("% %(...)"; around(){
    type parameters,...
    char* stjp->signature();
    printf("before %s\n",s);
    stjp->proceed();
    printf("after %s\n",s);
  };
}

aspect tracing{
  before():
    execution(int f(.))
    | execution(char* g(.))
    | ...
    printf("before function\n");

  /* after-advice analogous */
}

int f(int a){
  printf("before f\n");
  res = proceed();
  printf("after f\n");
  return res;
}

int trace_f(int aa){
  int res=0;
  printf("before f\n");
  res = continue_f(aa);
  printf("after f\n");
  return res;
}

void aspect_init(void){
  BEFORE(f, tracing_before);
  AFTER(f, tracing_after);...
}

Low-Level VM-approach:
  - life-long optimisation
  - weave LLVM-bytecode
  - extra join point context

VM weaving:

TOSKANA-VM

Aspicer

WeaveC

Aspect = semantic patch:
  - woven C written in situ
  - AspectC-like unwoven C4
  - generated and distributed

run-time weaving

AOP on the C-side

compile-time weaving

http://users.ugent.be/~badams

http://www.intec.ugent.be