AOP on the C-side

Bram Adams

Ghent University
Department of Information Technology
Sint-Pietersnieuwstraat 41
9000 Ghent, Belgium
http://www.intec.ugent.be

bram.adams@ugent.be
http://users.ugent.be/~badams

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

	T1	T3	T4	T5	A2	A3
AspectC	?	-	-	+	-	+
AspectC++	-	+	-	+	+	+
Aspicere	+	+	-	+	+	+
C4	+	-	-	+	-	+
WeaveC	?	?	-	+	+	+
µDiner	-	-	-	-	-	+
TinyC^2	N/A	-	+	-	-	+
Arachne	N/A	-	+	-	-	+
TOSKANA	N/A	-	+	-	-	+
TOSKANA-VM	+	?	-	-	?	?

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

compile-time weaving

AspectC

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

  /* after-advice analogous */
}
```

C4

Aspect = semantic patch:

- woven C4 written in situ
- AspectC-like unwoven C4 generated and distributed

WeaveC

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

run-time weaving

Aspicere

```
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;
}
```

µDiner

```
tracing :[
 int f(int aa) :[ {
   int res=0;
   printf("before f\n");
   res=continue_f(aa);
   printf("after f\n");
   return res;
 } ] ]
 ...
]
```

"hookable"

AspectC++

```
aspect tracing{
  advice execution("% %(...)"):
   around(){
    char* s=tjp->signature();
    printf("before %s\n",s);
    tjp->proceed();
    printf("after %s\n",s);
  }
};
```

type parameters, ...

Arachne

```
int trace_f(int aa){
  int res=0;
  printf("before f\n");
  res=f(aa); //no proceed()
  printf("after f\n");
  return res;
}
call(int f(int)) && args(a)
 then trace_f(a);
...
```

TinyC^2

```
onentry group * : ( ){
  char* s="before function";
  printf("%s\n",s);
}

/* onexit-advice analogous */
```

TOSKANA

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

ASPECT tracing_before(void){
  char* s="before function";
  printf("%s\n",s);
}

/* tracing_after analogous */
```

VM weaving

Low-Level VM-approach:

- life-long optimalisation
- weave LLVM-bytecode
- extra join point context

TOSKANA-VM