Aspect Orientation in the Procedural Context of C

ir. 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

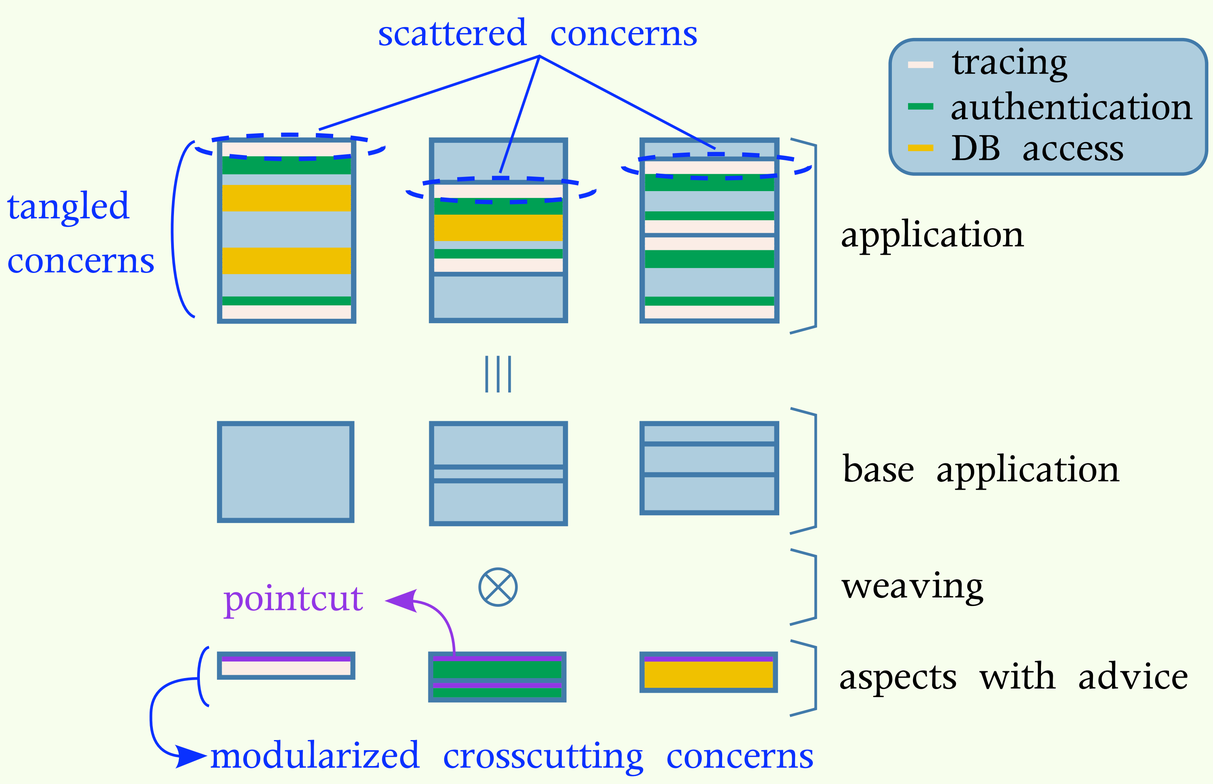

Fig. 1: The essence of Aspect Orientation.

```
static FILE* fp2=0;
static void close_file(void) { ... } /* atexit */
static FILE* init_file(char* name) { ... }

RetType around tracing (RetType,FileStr) on (Jp):
  call(Jp,"^.*/$")
  && type(Jp,RetType)
  && !str_matches("void",RetType)
  && logfile(FileName)
  && stringify(FileName,FileStr) {
    RetType i;
    FILE* fp=(fp2==0)?init_file(FileStr):fp2;

    fprintf(fp,"before ( %s in %s ) \n",
      Jp->functionName,Jp->fileName);
    fflush(fp);
    i = proceed();
    fprintf(fp,"after ( %s in %s ) \n",
      Jp->functionName,Jp->fileName);
    fflush(fp);

    return i;
}
```

A B C D

Fig. 2: A generic tracing aspect for non-void methods in Aspicere.

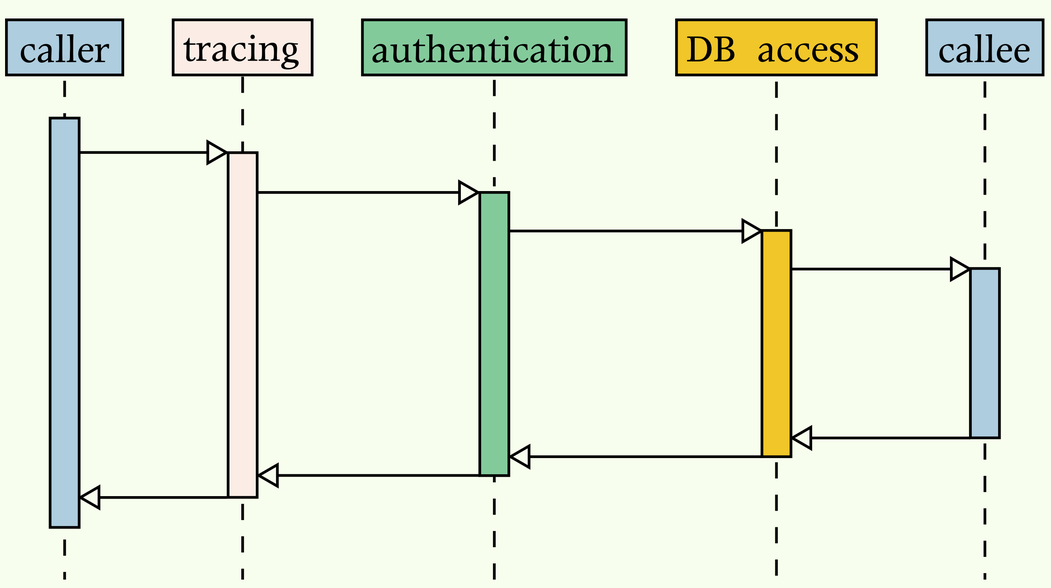

Fig. 3: Advice chain for the three crosscutting concerns of Fig. 1.

Problem Statement

Enterprise software systems ...
= "virtualized" business processes
- massive investments
- acceptable ROI only after long deployment periods

... should be highly and safely evolvable.

BUT:
- first generation maintainers are gone
- legacy hardware and programming languages (Cobol, C, ...)
- huge, ever-growing, undocumented code base

Aspect Orientation (AO)

A new(ish) paradigm tackling the problem of crosscutting concerns, i.e. scattered concerns which are tangled between others. **[Fig. 1]**

Terminology:
- advice: crosscutting functionality, part of aspect
- join point: interesting event in control flow of base program
- pointcut: defines set of join points by quantification on base program's properties
- aspect: advice linked to join points by pointcut

Benefits:
- better SoC, irrespective of paradigm (OO, procedural, ...)
- obliviousness: base program unaware of any aspects

➡ unintrusive reverse engineering of legacy systems

Aspicere, AO for C

Eligible join points: procedure calls and variable accesses

Combines multiple languages/paradigms: **[Fig. 2]**
- C: - leverage existing concepts and scoping rules A
  - less steep learning curve D
- Prolog: - bindings enable generic advice bodies D
  - robust pointcuts C using structure & semantics
- AO: - advice signature B, pointcut C and body D

Weaver transforms base program and aspects into ANSI C code, by chaining relevant advice. **[Fig. 3]**

Validation

Case Study:
- industrial system with 407 C-modules and 269 Makefiles

Goal:
- generate traces with tracing aspect + check pointer args

Results:
- all modules were woven and relevant traces obtained without altering the original source code
- we could replace some hacks with clean, modular code

Problems:
- slow, complex type inference (large pre-ANSI C chunks)
- slow join point matching ➡ scalability or implementation?
- deployment requires altering existing makefile-hierarchy

Legacy systems are killer apps for Aspect Orientation (AO).