Aspect Orientation for C: Express yourself

Aspicere

Bram ADAMS
Software Engineering Lab, INTEC, UGent

Tom TOURWÉ
CWI

Some background

= aspect-language for C (Latin: “to look at”)

History:

- Spin-off from Cobble (Kris De Schutter) [LD05]
- Developed as Master’s thesis (Stijn Van Wonterghem) [V04]
- Further development during PhD research (Bram Adams)

Characteristics:

- Based on Yerna Lindale- and Lillambi-frameworks
- Initially an AspectC-lookalike
- Source code weaving (preprocessor for gcc)

Main challenges:

- Expressive pointcut language → **current work (SPLAT)**
- Adequate weaving process

Parameter checking concern

=coding convention

```
int do_something (char* in, int* out, double** outptr) {
  if (in == (char*) NULL) {
     /*LOG*/
  }
  if (out == (int*) NULL) {
     /*LOG*/
  }
  if (*outptr != (double*) NULL) {
     /*SPECIAL LOG*/
  }
  …
  /*DEREFERENCE of 'in'*/
  /*DEREFERENCE of 'out'*/
  /*DEREFERENCE of 'outptr'*/
}
```

```
if (?var == (?ptrtype) NULL) {
   /*LOG*/
}
```

```
if (*?var == (?basetype) NULL) {
   /*SPECIAL LOG*/
}
```

BUT: only check if not already done earlier in control flow

AspectC

= subset of AspectJ [CKFS01]

➢Consequences for pointcut definitions:

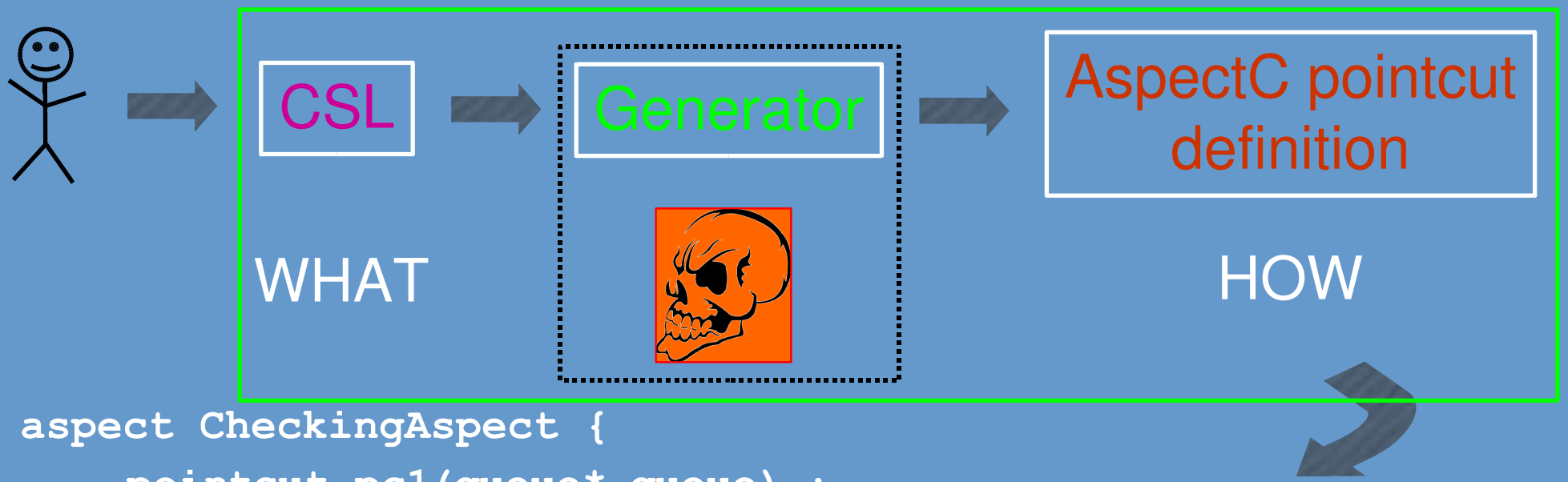
- No static conditions on bound variables
- How to capture a random method argument?
- No explicit, static navigation through code structure
- Name-based matching
- Fixed set of primitive pointcuts/joinpoint types

Lack of expressivity

⇔ Parameter concern needs:

- Structural pattern-based matching
- Static checks

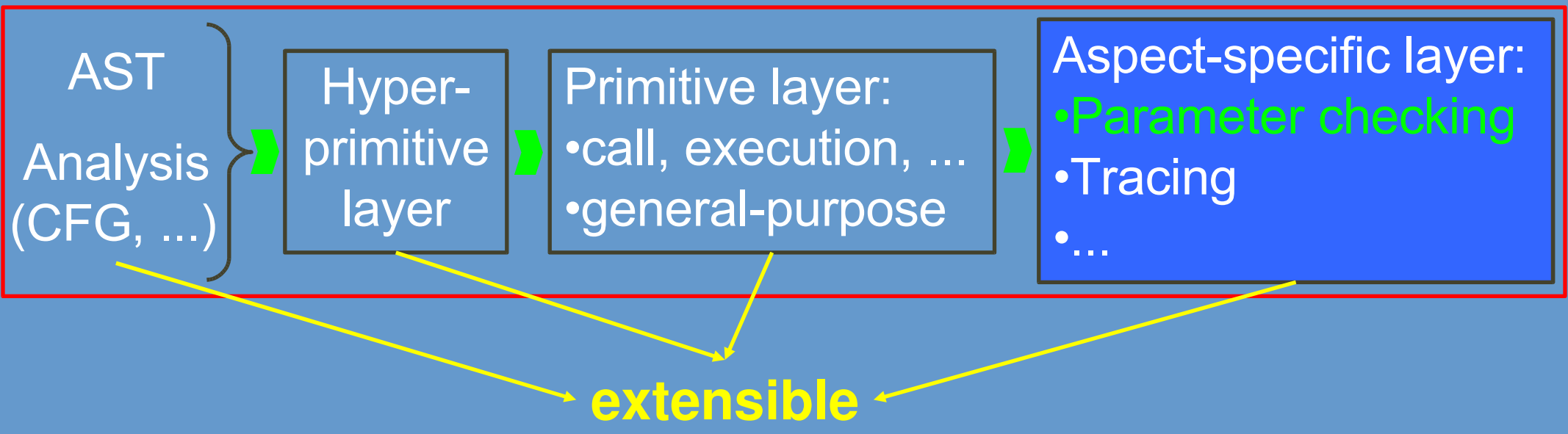
Solution using AspectC




```
aspect CheckingAspect {
    pointcut pc1(queue* queue) :
            args(queue) &&
            (execution(* queue_add(..)) ||
            execution(* queue_pop(..)));
    …
    before(queue* queue) : pc1(queue) {
            if(queue == (queue*) NULL) {
                    LOG(PARAMETER_ERROR)
            }
    }
}
```

Aspicere's design

Needed: pattern matching on structure of base program



Choose logic programming language [GB03]:

- Unification
- Broad range of predicates
  - Selection of joinpoints
  - General-purpose
- Access joinpoint shadows
  navigation through static structure of base program
- Parameterisable pointcut definitions (reuse!)
- Recursion

Taking on parameter concern

```
ioCheck(Jp,[Pointer,Type]):-
        /*gather all needed checkpoints*/.
```

```
before inout(BaseType) on(Jp):
        ioCheck(Jp,[Pointer,Type]) && pointer(Type,BaseType) {
                /*check pointer-parameters using bound BaseType*/
        }
```

CONCLUSION:

- AspectC-like pointcut language is not powerful enough
- Prolog allows extensible language design and expressiveness

- [CKFS01] Y. Coady, G. Kiczales, M. Feeley and G. Smolyn. Using AspectC to improve the modularity of path-specific customization in operating system code. SIGSOFT Software English Notes, 26(5):88-98, 2001.
- [GB03] K. Gybels and J. Brichau. Arranging language features for more robust pattern-based crosscuts. In *AOSD '03: Proceedings of the 2nd international conference on Aspect-oriented software development*, pages 60-69. ACM Press, 2003.
- [LD05] R. Lämmel and K. De Schutter. What does aspect-oriented programming mean to Cobol? In *Proceedings of 4th International Conference on Aspect-Oriented Software Development (AOSD '05)*, March 2005.
- [V04] S. Van Wonterghem. Aspect-oriëntatie bij procedurele programmeertalen, zoals C. Master's thesis, Ghent University, 2004. In Dutch.