Local Continuation Join Points
- AOP and Idiom-Based Exception Handling in C -
Bram Adams

int f(int a, int** b){
    int r = OK;
    r = mem_alloc(10, (int**) b);
    return-code idiom
    if(r != OK){
        /* no logging */
        /* no deallocation */
        return r;
    }else{
        if((a < 0)||a > 10)){
            r = PARAM_ERROR;
            LOG(r, OK);
            if(r != OK) mem_free(b);
            return r;
        }else{
            r = g(a);
            if(r != OK){
                LOG(LINKED_ERROR, r);
                r = LINKED_ERROR;
                if(r != OK) mem_free(b);
                return r;
            }else{
                r = h(b);
                if(r != OK){
                    /* no logging */
                    if(r != OK) mem_free(b);
                    return r;
                }else{
                    /* no deallocation */
                    return r;
                }
            }
        }
    }
}

Idiom-Based Software Development

Idiom-based software:
- system-wide programming conventions
- enhances software quality, e.g. return-code idiom (exception handling)
- makes up for lack of direct (legacy) language support

However:
- requires firm developer discipline, i.e. not enforced
- hampers code understandability, readability, ...

AOP can help:
- most invasive idioms are crosscutting concerns
- aspects can reduce error-prone manual approach
- prevents idiom lock-in

Local Continuation Join Points

Problem when modeling return-code idiom in aspect: "abort enclosing procedure execution after a call".

Local continuation of a join point p:
join point representing the future execution after conclusion of p, limited to the control flow of the procedure in which p is active.

Discussion

Cost of our approach:
- build-time overhead (± factor 10)
- run-time overhead (± 10% for example above):
  advice is transformed into procedures
  inlining advice on local continuation join points

Conclusion

- Aspects for idioms improve readability and evolvability
- Local continuation join points are core of our approach
- Performance penalty acceptable ↔ case study required

Implementation in Aspicere2

int around_cflow_transfer(int * R) on Jp:
idomatic_call(JpCall,R)
&!manual(JpCall)
&! local_continuation(Jp,JpCall){
  if(*R==OK)
    return *R;
  else
    return proceed();
}

7 extra aspects + accompanying Prolog files

Code size estimation:
- 122 LOC (aspects)
- base code annotations
- manual recovery code
  ⇒ dramatic code size reduction