

Bram Adams

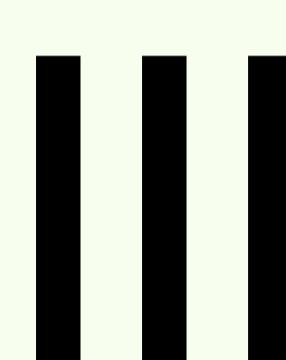
<http://users.ugent.be/~badams/aspicere2>

```

int f(int a, int** b){
    int r = OK;
    r = mem_alloc(10, (int**) b);

    if(r != OK){      return-code idiom
        /* 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;
                }
            }
        }
    }
}
  
```

main concern
error variable
control transfer



```

/*@range("a",0,10)*/
int f(int a, int** b){
    mem_alloc(10, (int**) b);

    /*@log("LINKED_ERROR")*/
    g(a);
    h(b);      base code
}
  
```



Discussion

Cost of our approach:

- build-time overhead (\pm factor 10)
- run-time overhead (\pm 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 \leftrightarrow case study required

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

- aspects are written once
- base code annotations
- configure aspects

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.**

 around-advice
on call's local
continuation
join point

```

int main(void){
    f();
    printf("C");
    return 0;
}

void f(void){
    printf("A");
    do_something();
    printf("B");
}
  
```

 output:
 "AC"

Implementation in Aspicere2

```

int around cflow_transfer(int* R) on Jp:
    idiomatic_call(JpCall,R)
    && !manual(JpCall)
    && local_continuation(Jp,JpCall){
        if(*R!=OK)
            return *R;
        else
            return proceed();
    }
  
```

control flow transfer advice

 7 extra
aspects
+
accompanying
Prolog files

```

idiomatic_call(Jp,R):-
    int_call(Jp,FName),
    %exclude (standard) libraries,
    enclosingMethod(Jp,JpEncl),
    idiomatic_proc(JpEncl),
    property(JpEncl,error_var,R).
  
```

```

idiomatic_proc(Jp):-
    execution(Jp,_),
    %limit scope of modules.
  
```

```

int_call(Jp,FName):-
    invocation(Jp,FName),
    type(Jp>Type),
    type_name(Type,"int").
  
```

Code size estimation:

- 122 LOC (aspects)
 - base code annotations
 - manual recovery code
- \Rightarrow **dramatic code size reduction**