

Aspektorientierte Programmierung mit AspectC++ Grundlagen

Georg Blaschke, Dipl.-Inf. Daniel Lohmann, Dr.-Ing. Olaf
Spinczyk

Friedrich-Alexander-Universität Erlangen-Nürnberg
Informatik 4

03.09.2004



Motivation

Aspektorientierten Programmierung (AOP)

AspectC++

Zusammenfassung

Demonstration

Eine Queue Klasse

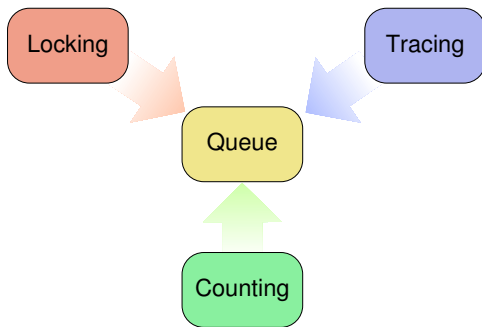
```
namespace util {
class Item {
    friend class Queue;
    Item* next;
public:
    Item() : next(0){}
};

class Queue {
    Item* first;
    Item* last;
public:
    Queue() : first(0), last(0) {}

    void enqueue( Item* item ) {
        if( last ){
            last->next = item;
            last = item;
        } else
            last = first = item;
    }
}
```

```
Item* dequeue() {
    Item* res = first;
    if( first == last )
        first = last = 0;
    else
        first = first->next ;
    return res;
}
}; // class Queue
} // namespace util
```

Anforderungen an Queue



Queue Klasse (mit Tracing)

```
class Queue {
    Item* first;
    Item* last;
public:
    Queue() : first(0), last(0) {
    }
    void enqueue( Item* item ) {
        printf ( " > Queue::enqueue()\n" );
        if( last ){
            last->next = item;
            last = item;
        } else{
            last = first = item;
        }
        printf( " < Queue::enqueue()\n" );
    }
}
```

```
Item* dequeue() {
    printf(" > Queue::dequeue()\n");
    Item* res = first;
    if( first == last )
        first = last = 0;
    else
        first = first->next ;
    printf(" < Queue::dequeue()\n");
    return res;
}; \\ class Queue
```

Queue (mit Tracing+Counting)

```
class Queue {
    Item* first;
    Item* last;
    unsigned int counter;
public:
    Queue() : first(0), last(0) {
        counter=0;
    }
    void enqueue( Item* item ) {
        printf ( " > Queue::enqueue()\n" );
        counter++;
        if( last ){
            last->next = item;
            last = item;
        } else{
            last = first = item;
        }
        printf( " < Queue::enqueue()\n" );
    }

    Item* dequeue() {
        printf(" > Queue::dequeue()\n");
        Item* res = first;
        if( first == last )
            first = last = 0;
        else
            first = first->next ;
        counter--;
        printf(" < Queue::dequeue()\n");
        return res;
    }
    unsigned int count(){return counter;}
}; \\ class Queue
```

Queue (mit Tracing+Counting+Locking)

```
class Queue {
    Item* first;
    Item* last;
    unsigned int counter;
    os::Mutex lock;
public:
    Queue() : first(0), last(0) {
        counter=0;
    }
    void enqueue( Item* item ) {
        printf ( " > Queue::enqueue()\n" );
        lock.enter();
        counter++;
        if( last ){
            last->next = item;
            last = item;
        } else{
            last = first = item;
        }
        lock.leave();
        printf( " < Queue::enqueue()\n" );
    }

    Item* dequeue() {
        printf(" > Queue::dequeue()\n");
        lock.enter();
        Item* res = first;
        if( first == last )
            first = last = 0;
        else
            first = first->next ;
        counter--;
        lock.leave();
        printf(" < Queue::dequeue()\n");
        return res;
    }
    unsigned int count(){return counter;}
}; \\ class Queue
```

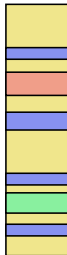
Tangled Code

Resultat:

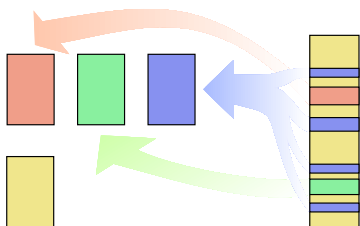
- verhedderter Quelltext (engl. *tangled code*):
Strukturen im Quelltext, die mit der eigentlichen Funktion einer Queue nichts zu tun haben.

Probleme:

- schwerere Lesbarkeit
- höhererer Wartungsaufwand
- eingeschränkte Wiederverwendbarkeit
- schlechtere Konfigurierbarkeit



Prinzip der Trennung der Belange



„Dinge, die nichts miteinander zu tun haben, sind auch getrennt unterzubringen“

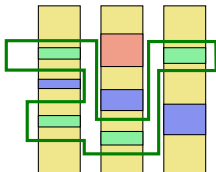
Trennung der Belange (engl. *separation of concerns*)

Kapselung jedes einzelnen Belanges in einem eigenen Softwaremodul (z.B. Funktionen, Klassen, Aspekte,...)

Problem:

Querschneidende Belange

Querschneidende Belange



Querschneidender Belang (engl. *crosscutting concern*)

Belang, dessen Implementierung über mehrere Stellen im Programmcode verteilt ist

Aspektorientierten Programmierung (AOP)

- Erweitert objektorientierte/prozedurale Programmierung um das Konzept der Aspekte

Aspekt

Softwaremodul zur Kapselung eines querschneidender Belangs

Komponente

Softwaremodul zur Kapselung eines herkömmlicher Belangs

Aspektweber

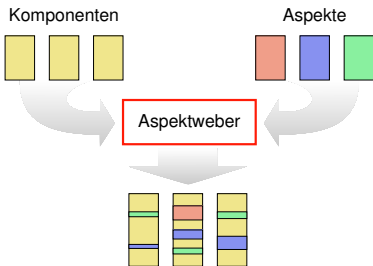
Komponenten



Aspekte

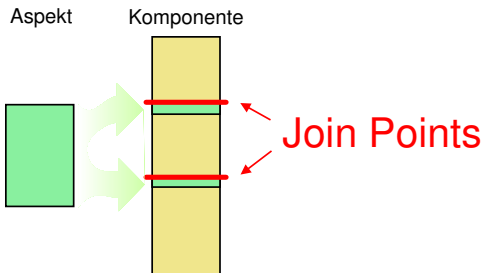


Aspektweber(2)



- Verknüpfung (*Weben*) des Aspektcodes mit dem Komponentencode
- Weben möglich zur Übersetzungszeit, Ladezeit oder Laufzeit

Verbindungspunkt



Verbindungspunkt (engl. *join point*)

Stelle im Komponentencode, wo der Aspektweber Aspektcode hineingewebt; wird im Aspekt spezifiziert.

AspectJ

- Bekanntes System für AOP
- Erweiterung der Programmiersprache Java
- Eingängige Syntax
- Benötigt keine spezielle Java-Laufzeitumgebung
- Starke Förderung durch IBM
- Werkzeugunterstützung:
Kommandozeile, Emacs, Eclipse und JBuilder

AspectC++

- Syntax und Semantik angelehnt an AspectJ
- Transformiert C/C++- und Aspekt-Quelltext nach C++
- Overhead vergleichsweise gering
- Werkzeugunterstützung:
Kommandozeile, VisualStudio.NET, Eclipse

Aktuelle Version 0.9pre1

- Noch Mangelnde Unterstützung einiger C++
Sprachkonstrukte

Queue Klasse (mit Tracing)

```
class Queue {
    Item* first;
    Item* last;
public:
    Queue() : first(0), last(0) {
    }
    void enqueue( Item* item ) {
        printf ( " > Queue::enqueue()\n" );
        if( last ){
            last->next = item;
            last = item;
        } else{
            last = first = item;
        }
        printf( " < Queue::enqueue()\n" );
    }
}
```

```
Item* dequeue() {
    printf(" > Queue::dequeue()\n");
    Item* res = first;
    if( first == last )
        first = last = 0;
    else
        first = first->next ;
    printf(" < Queue::dequeue()\n");
    return res;
}; \\ class Queue
```

Einfache Implementierung eines Tracing Aspekts

tracing.ah

```
aspect Tracing{
    advice execution("% util::Queue::%(...)"): before(){
        printf("--> %s\n",JoinPoint::signature());
    }
    advice execution("% util::Queue::%(...)"): after(){
        printf("<-- %s\n",JoinPoint::signature());
    }
};
```

Aspect

tracing.ah

```
aspect Tracing{  
    advice execution("% util::Queue::%(...)"): before(){  
        printf("--> %s\n",JoinPoint::signature());  
    }  
    advice execution("% util::Queue::%(...)"): after(){  
        printf("<-- %s\n",JoinPoint::signature());  
    }  
};
```

Advice (before)

tracing.ah

```
aspect Tracing{  
    advice execution("% util::Queue::%(...)"): before(){  
        printf("--> %s\n",JoinPoint::signature());  
    }  
    advice execution("% util::Queue::%(...)"): after(){  
        printf("<-- %s\n",JoinPoint::signature());  
    }  
};
```

Advice (after)

tracing.ah

```
aspect Tracing{
    advice execution("% util::Queue::%(...)"): before(){
        printf("--> %s\n",JoinPoint::signature());
    }
    advice execution("% util::Queue::%(...)"): after(){
        printf("<-- %s\n",JoinPoint::signature());
    }
};
```

Join-Point API

tracing.ah

```
aspect Tracing{
    advice execution("% util::Queue::%(...)"): before(){
        printf("--> %s\n",JoinPoint::signature());
    }
    advice execution("% util::Queue::%(...)"): after(){
        printf("<-- %s\n",JoinPoint::signature());
    }
};
```

Queue (mit Locking)

```
class Queue {
    Item* first;
    Item* last;
    os::Mutex lock;
public:
    Queue() : first(0), last(0) {
    }
    void enqueue( Item* item ) {
        lock.enter();
        if( last ){
            last->next = item;
            last = item;
        } else{
            last = first = item;
        }
        lock.leave();
    }
}
```

```
Item* dequeue() {
    lock.enter();
    Item* res = first;
    if( first == last )
        first = last = 0;
    else
        first = first->next ;
    lock.leave();
    return res;
}; \\ class Queue
```

Implementierung eines Locking Aspekts

locking.ah

```
aspect Locking{  
  
    advice "util::Queue": os::Mutex lock;  
  
    advice execution("% util::Queue::%queue(...)"): around(){  
        JoinPoint::That* currentObject=tjp->that();  
        (currentObject->lock).enter();  
        tjp->proceed();  
        (currentObject->lock).leave();  
    }  
  
};
```

Introduction

locking.ah

```
aspect Locking{  
  
    advice "util::Queue": os::Mutex lock;  
  
    advice execution("% util::Queue::%queue(...)"): around(){  
        JoinPoint::That* currentObject=tjp->that();  
        (currentObject->lock).enter();  
        tjp->proceed();  
        (currentObject->lock).leave();  
    }  
};
```

Advice (around)

locking.ah

```
aspect Locking{  
  
    advice "util::Queue": os::Mutex lock;  
  
    advice execution("% util::Queue::%queue(...)"): around(){  
        JoinPoint::That* currentObject=tjp->that();  
        (currentObject->lock).enter();  
        tjp->proceed();  
        (currentObject->lock).leave();  
    }  
  
};
```

Join-Point API

locking.ah

```
aspect Locking{  
  
    advice "util::Queue": os::Mutex lock;  
  
    advice execution("% util::Queue::%queue(...)"): around(){  
        JoinPoint::That* currentObject=tjp->that();  
        (currentObject->lock).enter();  
        tjp->proceed();  
        (currentObject->lock).leave();  
    }  
};
```

Join-Point API

locking.ah

```
aspect Locking{  
  
    advice "util::Queue": os::Mutex lock;  
  
    advice execution("% util::Queue::%queue(...)"): around(){  
        JoinPoint::That* currentObject=tjp->that();  
        (currentObject->lock).enter();  
        tjp->proceed();  
        (currentObject->lock).leave();  
    }  
};
```

Zusammenfassung

- Kapselung von querschneidenden Belangen mit herkömmlicher Programmierung nicht machbar
- AOP ermöglicht Implementierung von querschneidenden Belangen in eigenständigen Softwaremodulen, den Aspekten
- AspectC++ ermöglicht aspektorientierte Programmierung in C++

Weiterführende Informationen

- Aspect-Oriented Software Development:
www.aosd.net
- AspectC++ :
www.aspectc.org
- Kommerzieller Support und VisualStudio.NET Plugin :
www.pure-systems.com

Demonstration der AspectC++ Erweiterung für Eclipse