FOG

Thomas Grzenkowski

email: thomas@grzenkowski.de

Probleme mit Präprozessor cpp

- Cpp sollte ersetzt werden anstatt beseitigt werden.
- Die Kompilierzeitprogrammierung ist notwendig, um Deklarationen zu konfigurieren.
- Muster und AOP erfordern das Weben.
- Die One Definition Rule muss umgangen werden.
- Introspection ist für einfache Anwendungen nützlich.
- Reflexion ist für hoch entwickelte Anwendungen fast unverzichtbar.

Probleme mit Präprozessor cpp

- Lexikalische Redundanz sollte beseitigt werden.
- Vorhersagbarer Code sollte automatisch zur Verfügung gestellt werden.
- Ein Konzept sollte sich durch einen einzelnen Aufruf realisieren lassen.
- Überlappende Deklarationen sollten erlaubt werden.

ODR und CDR

- One Definiton Rule (ODR): Alles darf nur genau einmal definiert werden.

- Composite Definition Rule (CDR): Weitere Definitionen erweitern die bisherigen, sofern sie sich nicht widersprechen.

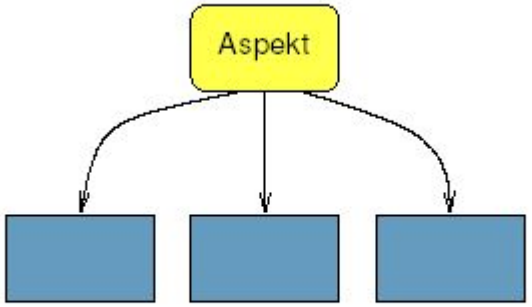
Flexible Object Generator (FOG)

- Verbessert C++ (aufwärts kompatibel, ohne neue reservierte Wörter).
- Übernimmt die Umorganisierung des Quellcodes.
- Übernimmt den Aufbau der Deklarationen.
- Interpretiert Metaprogramme.

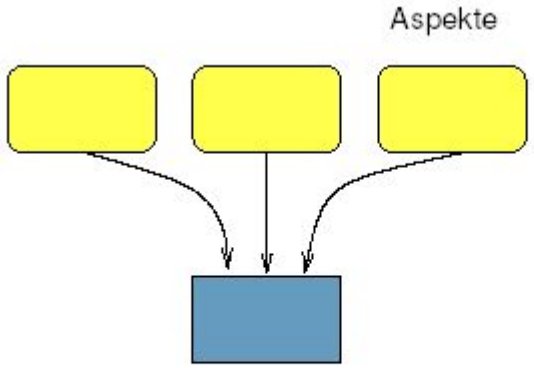
Metaprogrammierung

Herkömmliche Programmierung beschäftigt sicht mit der Entwicklung von Progammen passierend auf deren Verarbeitungsinstanz.
Metaprogrammierung beschäfigt sich mit dem programmieren auf Programminstanzen: die Klassen, Funktionen und deklarierten Variablen die ein Programm ausmachen.

AOP und SOP



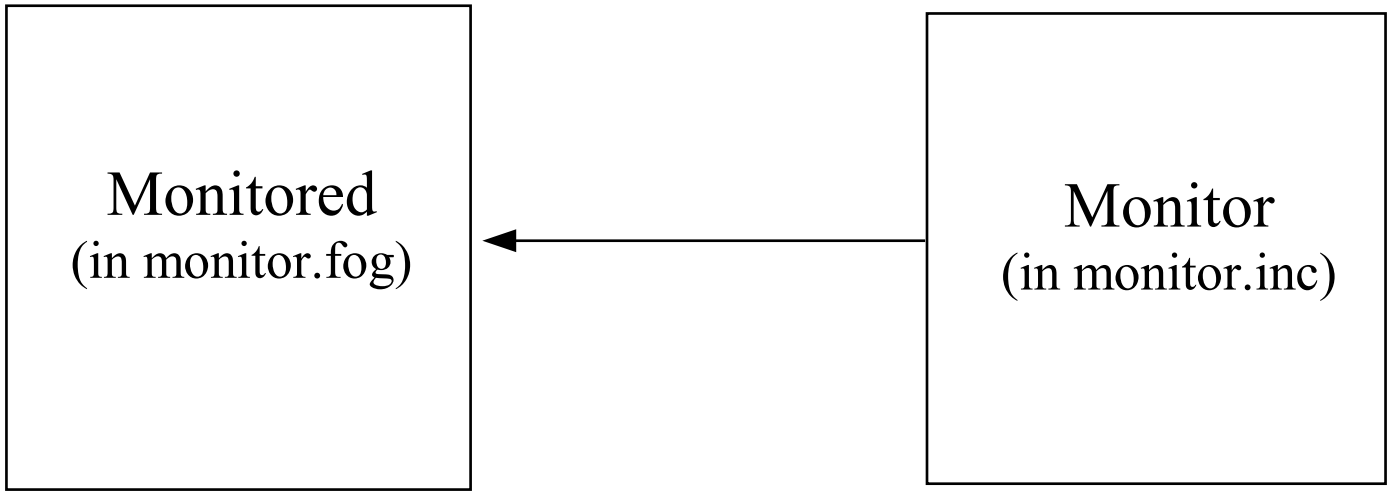
(a) Crosscutting



(b) Subjektorientierte Programmierung

1. Beispiel: Monitor-Aspekt

Es soll zu der Klasse Monitored ein Monitor Aspekt hinzugefügt werden:



monitor.fog

```
using "monitor.inc";

class Monitored
{
    $DerivedMonitor::install();
private:
    int *_i;
public:
    int *f() { return _i; }
    const int *fc() const { return _i; }
    const volatile int *fcv() const volatile { return _i; }
    volatile int *fv() volatile { return _i; }
};
```

monitor.inc – Teil: DerivedMonitor

```
class DerivedMonitor : public Monitor
{
    export/interface Monitor;
    export/implementation Monitor;
};
```

monitor.inc – Teil: ~Monitor()

```
auto Monitor::~Monitor()
{
    auto if (has_monitor)    // Avoid execution for Monitor and its derived classes.
    {
        auto for (iterator f = $functions(); f; ++f)
        {
            auto if (f->is_static())
                ;
            else if (f->is_volatile())
                ;
            else if (f->is_const())
            {
                $f->specifier() :{ entry { ReadOnlyLock aLock(_monitor); } };
            }
            else
            {
                $f->specifier() :{ entry { ReadWriteLock aLock(_monitor); } };
            }
        }
    }
}; // [[derived tree]];
```

monitor.inc – Teil: Monitor

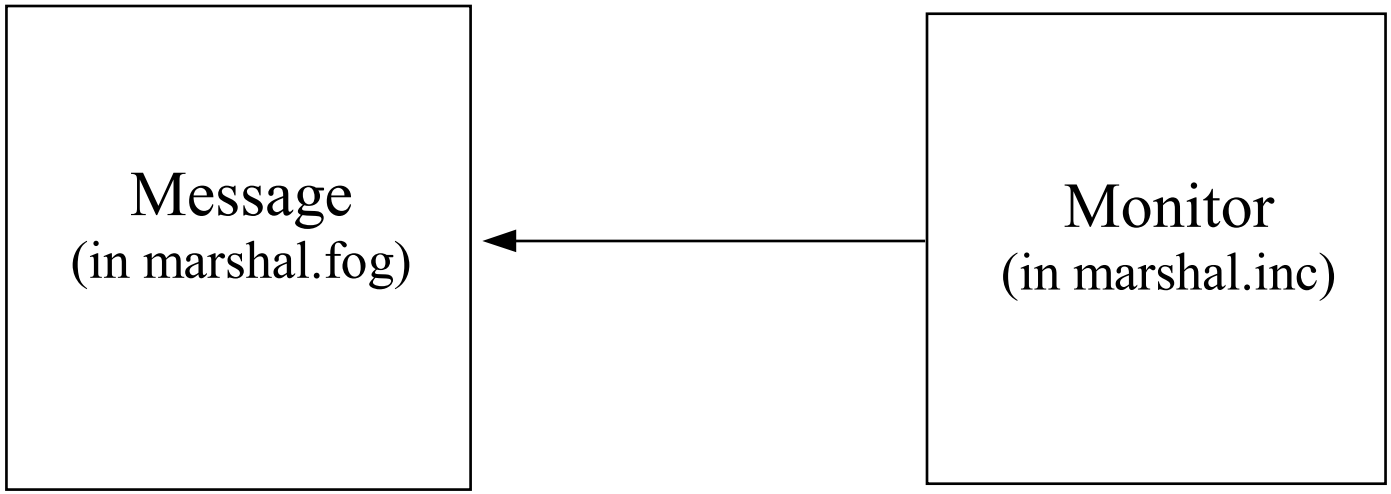
```
class Monitor
{
    auto number has_monitor = false;
public:
    class ReadOnlyLock
    {
    private:
        Monitor& _monitor;
    public:
        inline ReadOnlyLock(Monitor& aMonitor) : _monitor(aMonitor) { _monitor.enter_shared(); }
        inline ~ReadOnlyLock() { _monitor.leave(); }
    };
    friend class ReadOnlyLock;
    class ReadWriteLock
    {
    private:
        Monitor& _monitor;
    public:
        inline ReadWriteLock(Monitor& aMonitor) : _monitor(aMonitor) { _monitor.enter_exclusive(); }
        inline ~ReadWriteLock() { _monitor.leave(); }
    };
    friend class ReadWriteLock;
private:
    void enter_exclusive() { /* ... */ }
    void enter_shared() { /* ... */ }
    void leave() { /* ... */ }
};
```

monitor.inc – Teil: Monitor::install()

```
auto declaration Monitor::install()
{
    class $This : auto $Dynamic
    {
        private $Dynamic _monitor;
        auto number has_monitor = true;
    };
}
```

2. Beispiel: Marshalling Aspekt

Es soll zu einer Message-Klasse ein Marshalling-Aspekt hinzugefügt werden:



marshal.fog

```
class Message
{   /* ... */ };

class StockReport : public Message
{
    export/interface Message; export/implementation Message;
private:
    unsigned long _item_number;
    short _stock_level;
    // ...
};

using "marshal.inc";

class Message
{
    $Marshal::install();
};
```

marshal.inc - 1. Teil: Marshal::install

```
auto class Marshal {};
typedef unsigned long size_t;
auto declaration Marshal::install()
{
    auto type MessageClass = $Scope;
    auto static statement switchBody =
    {
        default:
        return 0;
    }
    auto number byte_count = 0;
    protected enum MessageTypes {};
```

marshal.inc - 2. Teil: Marshal::install

```
public virtual size_t marshal(unsigned char dataBuffer[]) const
:{
    derived(true) entry
    {
        unsigned char *p = dataBuffer;
        *p++ = MESSAGE_@Scope;
        *p++ = @byte_count;
    }
    derived(true) exit
    {
        return p - (dataBuffer+2);
    }
};

public static !inline $Scope *unmarshal(unsigned char dataBuffer[])
{
    switch (dataBuffer[0])
        @switchBody;
}
```

marshal.inc - 3. Teil: Marshal::install

```
public static @Scope *make(unsigned char dataBuffer[])
:{ derived(true)
    {
        if (*p != @byte_count)
            return 0;
        else
            return new @{Scope}(dataBuffer);
    }
};

auto ${Scope}()
{   protected enum ${MessageClass}::MessageTypes { MESSAGE_$Dynamic };
    auto switchBody +=
        case MESSAGE_$Dynamic:
            return ${Dynamic}::make(dataBuffer);
    private inline/implementation ${Dynamic}(unsigned char dataBuffer[])
    {
        unsigned char *p = dataBuffer + 2;
}   }
```

marshal.inc - 4. Teil: Marshal::install

```
    auto ~${Scope}()
    {
        auto for (iterator i = $variables(); i; i++)
            auto if (!i->is_static())
                $i->type().marshal($i->id());
    }
}
```

marshal.inc – Teil: long marshal(...)

```
auto declaration unsigned long::marshal(expression name)
{
    byte_count += 4;
//  using marshal
    public virtual size_t marshal(unsigned char dataBuffer[]) const
    {
    *p++ = ($name >> 24) & 0xFF;
        *p++ = ($name >> 16) & 0xFF;
        *p++ = ($name >> 8) & 0xFF;
        *p++ = $name & 0xFF;
    }
//  using ${Scope}
    private ${Scope}(unsigned char dataBuffer[])
    {   {
            unsigned long temp = *p++;
            temp = (temp << 8) | *p++;
            temp = (temp << 8) | *p++;
            temp = (temp << 8) | *p++;
            $name = temp;
}   }   }
```

marshal.inc – Teil: short marshal(...)

```
auto declaration short::marshal(expression name)
{
    auto byte_count += 2;
//  using marshal
    public virtual size_t marshal(unsigned char dataBuffer[])
const
    {
        *p++ = ($name >> 8) & 0xFF;
        *p++ = $name & 0xFF;
    }
//  using ${This}
    private ${Scope}(unsigned char dataBuffer[])
    {   {
            unsigned long temp = *p++;
            temp = (temp << 8) | *p++;
            $name = temp;
}   }   }
```