Hauptseminar Aspektorientierte Systemprogrammierung

Dynamisches Aspektweben in C und C++: DAO C++ und µDyner

Andre Zweschke

Inhalt

1. Einleitung und Motivation

Die objektorientierte Programmierung gilt als zur Zeit eleganteste Programmierung: mit ihr lassen sich zusammengehörende Belange (Daten und darauf arbeitende Funktionen) zusammenfassen (kapseln). Doch es gibt Fälle, in denen ein zusammengehörender Belang sich durch viele Teile des Programms zieht und unmöglich in ein Objekt gefasst werden kann. Man spricht von einem *„crosscutting-concern“*. Ein einfaches Beispiel wäre eine Debugging-Funktionalität, die bei jedem Betreten und Verlassen einer Funktion eine entsprechende Ausgabe macht. Die aspektorientierte Programmierung bietet hier die Möglichkeit, einen solchen Belang doch zusammenzufassen und zu kapseln. Mittlerweile ziemlich bekannt ist AspectJ, welches diese Funktionalität für Java bereitstellt, doch auch für C++ gibt es eine entsprechende Erweiterung. Diese beiden Spracherweiterung bieten allerdings „nur“ fest eingewobene (statische) Aspekte an. Im Folgenden soll es darum gehen, wie Aspekte zur Laufzeit in C oder C++ eingewoben werden können (dynamische Aspekte).

Web-Caches schaffen heute die Möglichkeit im ständig wachsenden Internet den Netzwerkverkehr zu verringern und können auch die Zugriffszeit auf Dokumente, die bereits im Cache liegen, wirksam reduzieren. Allerdings befinden sich längst nicht alle angeforderten Dokumente bereits im Cache. Um dem entgegenzuwirken könnte man sämtliche mit dem Dokument verlinkten Seiten ebenfalls anzufordern (*„Prefetching“*). Allerdings führt das zu einem erhöhten Datentransfer, dem die Caches ja gerade entgegenwirken sollten.
Es muss also eine vernünftige Prefetching-Strategie gefunden werden.

Wenn ein Client eine Seite über einen Web-Cache aufruft, laufen auf dem Server folgende Schritte ab (Abb. 1): Sollte die Seite im Cache gespeichert sein, so sendet er sie an den Client zurück. Ansonsten muss er die Seite zuerst anfordern. Das Dokument wird im Cache abgelegt, dazu muss er eventuell Platz schaffen und über die Seitenersetzungsstrategie die zu verdrängenden Seiten auswählen.

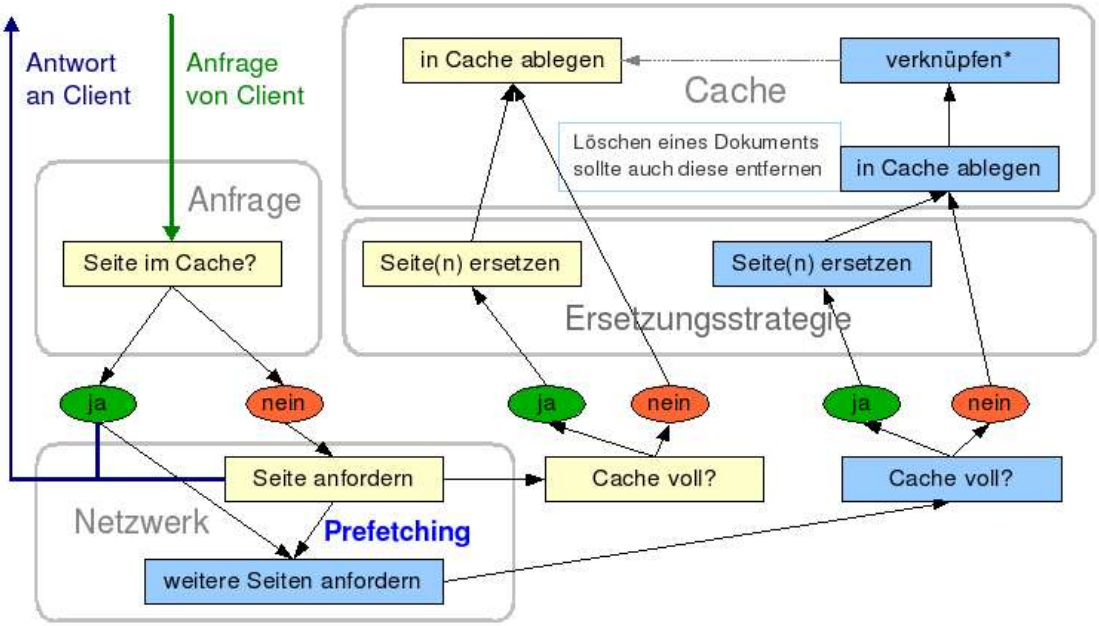

Abb. 1: Anfrageabarbeitung eines Web-Caches

Angestoßen werden sollte das Prefetching sinnvollerweise vom Netzwerkmodul. Im Cache müssen eventuell weitere Seiten gelöscht werden und es macht Sinn Dokumente, die durch Prefetching in den Cache gelangt sind, mit der ursprünglich aufgerufenen Seite zu verknüpfen, damit sie mit ihr zusammen gelöscht werden können.

Es zeigt sich also, dass das Prefetching Einfluss auf die meisten Programmteile des Web-Caches hat, man spricht von einem *„crosscutting concern“*. Die eleganteste Möglichkeit der Implementierung bietet hier die **Aspektorientierte Programmierung.**

Um das Prefetching bestmöglich auf die Anfragen zuschneiden zu können sollte es möglich sein den Prefetching-Algorithmus auszutauschen; mit der aspektorientierten Programmierung lässt sich das am besten realisieren. Allerdings können Web-Caches nicht so einfach abgestellt werden. Der Austausch müsste so schnell erfolgen, dass es für den Nutzer unbemerkt bleibt, am besten wäre es Aspekte zur Laufzeit weben zu können. Nachdem Proxy-Server in der Regel in C oder C++ geschrieben sind, müssen die Sprachen entsprechend erweitert werden. DAO C++ [1] und µDyner [2] sind zwei Projekte, die genau dies tun.

2. Dynamic Aspect Oriented C++ (DAO C++)

Dies ist das erste Projekt zum Weben dynamischer Aspekte in C++.

2.1 DAO C++ Benutzersicht

Zunächst soll einmal ein kurzer Einblick gegeben werden, wie der Programmierer mit dieser Erweiterung umzugehen hat[1]. Um einen Aspekt in DAO C++ zu erzeugen muss der Programmierer eine von `Aspect` abgeleitete Klasse erzeugen:

```
class NewAspect : public Aspect {
   void advice() {
      advice code
   }
}

NewAspect na;
na.setExpression("class_sign, method_sign()")
na.setBefore(true);
na.setAfter(false);
```

Mit `setExpression()` definiert der Programmierer die Pointcuts, mit `setBefore()` und `setAfter()` ob der Aspekt zu Beginn oder Ende einer Funktion eingefügt werden soll.

1. Die Codefragmente zur Veranschaulichung sind aus [1] entnommen

Wie man hier sieht, handelt es sichbei DAO C++ um keine echte Spracherweiterung, das Projekt ist eher als Framework zusehen. Es werden keine neuen Befehle oder Ausdrücke hinzugefügt, sondern die bereits vorhandenen sprachlichen Elemente genutzt: Funktionen und Strings, die zur Laufzeit ausgewertet werden.

Um einen Aspekt in die bestehende Anwendung einweben zu können, muss eine Instanz der Klasse `AOP_Engine` (AOP steht für Aspekt Oriented Programming) erzeugt werden:

```
AOP_Engine aop_engine;
```

Damit lässt sich dann die gewünschte Aktion ausführen:

```
aop_engine.weave(&na);          //Weben
aop_engine.unweave(&na);        //Entfernen
aop_engine.refresh(&na);        //neu Einlesen und Weben
```

2.2 DAO C++ Implementierung

Nach der kurzen Einführung nun zu den Internas und zu Implementierungsdetails von DAO C++.

DAO C++ besteht aus der gerade schon angesprochenen AOP-Engine und einem Präprozessor, der den Programmcode so modifiziert, dass ein Aspekt eingewoben werden kann. Dazu wurde für die Implementierung OpenC++ [6] verwendet, mit dem sich *„source-to-source“*-Übersetzer und Werkzeuge zur Quellcodeannalyse für C++ entwickeln lassen. Es entsteht also wieder C++-Code, der von jedem Standard-Compiler (z.B. gcc) übersetzt werden kann.

Der Präprozesser hat zwei Aufgaben:

1. Er muss die *„minimal hooks“* in den Programmcode einfügen:

```
int anymethod (............) {
   if (JoinPoints [i].BeforeIsActivated)
       RunAdviceBefore(i)
   method body code
   if (JoinPoints [i].AfterIsActivated)
       RunAdviceAfter(i)
}
```

(Anm.: Die *„minimal hooks“* der After-Advice müssen auch vor jeder `return`-Anweisung stehen.)

2. Er muss für jeden *„minimal hook“* Metadaten erzeugen (siehe Abb. 2):
   - Die Klassen- und Methodennamen werden gespeichert; damit werden bei der Auswertung der *Pointcuts* die Verbindungspunkte (*„Joinpoints“*) gefunden an denen der Aspekt wirken soll.
   - Für jeden *„minimal hook“* muss die Information gespeichert werden, ob ein Aspekt an der Stelle wirkt (`JoinPoints[i].BeforeIsActivated`) und welcher Code in diesem Fall auszuführen ist (`RunAdviceBefore(i)`).

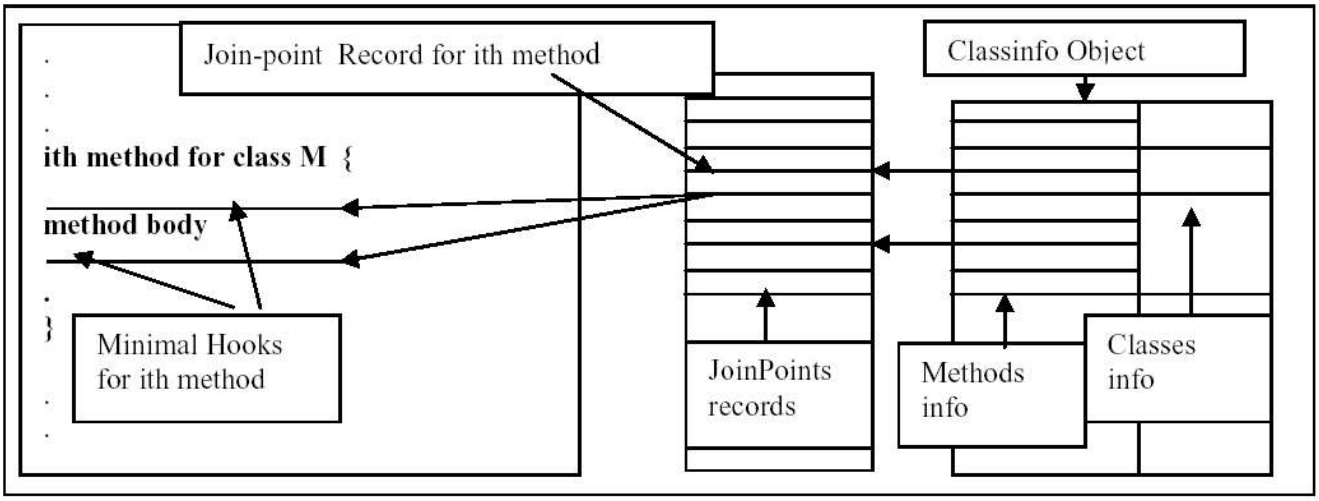

Abb. 2: Strktur der Metadaten für alle „minimal hooks“ (entnommen aus [1])

Zum Aspektweben muss die AOP-Engine die Metadaten aller auf den *„Pointcut“* passenden Funktionen modifizieren. Damit es währenddessen nicht zu Inkonsistenzen kommen kann, wird jeder Aspekt als kritischer Bereich betrachtet und mit einer Semaphore geschützt. Während des Aspektwebens befindet sich das Programm also nicht im Aspektcode. Wenn ein Aspekt entfernt werden soll, muss zuerst darauf gewartet werden, bis das Programm alle Bereiche des Aspekts verlassen hat. Dieser Mechanismus sorgt allerdings lediglich für die Konsistenz der von DAO C++ verwalteten Datenstrukturen und verhindert Programmabbrüche. Der Applikationsprogrammierer muss allerdings wissen, dass es weiterhin zu Inkonsistenzen kommen kann, wenn sich zum Beispiel das Programm während des Aspektwebens zwischen den *„minimal hooks“* einer Funktion befindet:

```
int anymethod (............) {
   minimal hook at beginning
   method body code
   minimal hook at the end
}
```

Wenn `Minimal hook at beginning` eine Semaphore setzt, die `minimal hook at the end` wieder freigeben soll, und wird der Aspekt in der Zwischenzeit deaktiviert, so wird der kritische Abschnitt nicht wieder freigegeben.

2.3 Laufzeitverhalten

Nachdem in den C++-Code zusätzlicher Code eingefügt wurde, stellt sich die Frage wie stark dadurch das Laufzeitverhalten beeinflusst wird:
Auch wenn in ein DAO C++-Programm kein Aspekt eingewoben wurde ist zu erwarten, dass das Programm langsamer ist, wenn man es mit dem gleichen C++-Programm vergleicht. Die Ausführungsgeschwindigkeit hängt auch sehr stark von der Programmstuktur ab: bei vielen kleinen Funktionen sind mehr *„minimal hooks“* pro ausgeführtem Programmcode vorhanden, entsprechend langsamer ist die Ausführung.
Vergleicht man ein DAO C++-Programm mit einem C++-Programm, die beide den gleichen Aspekt eingewoben haben, so ist ebenfalls ein spürbarer Overhead zu erwarten.

3. µDyner (steht für Micro Dynamic Weaver)

Das zweite Projekt ist µDyner, welches zunächst einmal dynamisches Weben für C zur Verfügung stellt.

3.1 µDyner Benutzersicht

Auch zunächst einmal, wie der Programmierer mit dieser Erweiterung umzugehen hat[2]:

```
prevent_propagation : [
   int handle_request (char *req) : [
      int relay_request (struct req.data *request) : [
         { #include "prefetching.h"
            if (ie_prefetching_request(request))
               return NO_NEIGHBOR_HAS_FILE;
         } return continue_relay_request(request); }
      ]
   ]
]
```

Es handelt sich hierbei um den Aspekt `prevent_propagation` zum Weiterleiten von Anfragen an andere Caches. `handle_request` kümmert sich um die Abarbeitung einer solchen Anfrage, indem es durch die *„Aspektfunktion“* `relay_request` erweitert wird. In einem C-Programm würde `handle_request` also `relay_request` aufrufen.
Wie man hier sieht handelt es sich bei µDyner um eine echte Spracherweiterung, die durch einen Präprozessor in C-Code umgewandelt werden muss. Damit ist es auch hier möglich einen Standard-C-Compiler zu benutzen und auch hier sorgt ein Tool dafür, dass Aspekte zur Laufzeit geladen werden können, die in Form von Standard-Bibliotheken bereitgestellt werden (s. Abb. 3).

2 Dieses Codebeispiel wurde aus [2] entnommen.

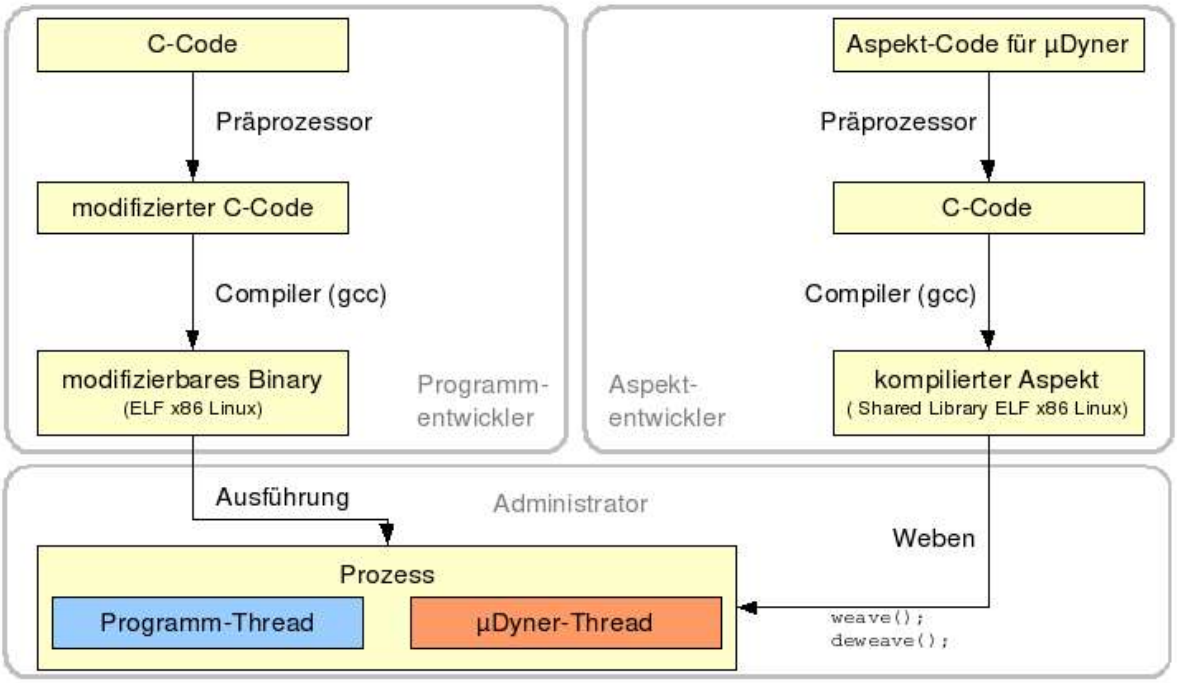

Abb. 3:Entwicklung eines Programms mit µDyner (nach [3])

3.2 µDyner Implementierung

Im Detail zeigen sich eine ganze Menge Unterschiede[3] zu DAO C++:

Am Anfang jeder Funktion werden vor dem ersten Befehl 5 Byte eingefügt. An dieser Stelle steht zunächst nur ein Sprung auf den nächsten Befehl:

```
int anymethod (.............) {
   JMP 3; NOP; NOP; NOP;
   method body code
}
```

Beim Weben wird an diese Stelle ein Sprung zu einem „minimal hook“ eingefügt:

```
int anymethod (.............) {
   JMP Hook;
   method body code
}
```

Dass genau 5 Byte eingefügt werden resultiert daraus, dass ein Sprungbefehl beim Pentium, für den µDyner aktuell entwickelt wird, genau diese Länge besitzt: 1 Byte für den Befehl „JMP“, 4 Byte für die Sprungadresse.
Mit dem Posix-Tool nm werden zur Laufzeit die Symbole und ihre Position ermittelt, deshalb muss auf jeden Fall verhindert werden dass Funktionen vom Kompiler „inlined“ werden.

3 Der nachfolgende Code zur Verdeutlichung ist aus [2] entnommen.

Wenn der Sprungbefehl eingefügt wurde, der „minimal hook“ also aktiviert wurde, springt das Progamm zu einem Stück Code:

```
save registers
if (aspect_loaded && pointcut()) {
   call aspect
   restore registers
   return aspect result
} else {
   restore registers
   perform original effect of join point
return result
}
```

Nachdem dieser nachträglich zur Laufzeit eingebunden wird, handelt es sich hierbei um keine Funktion im eigentlichen Sinne: bei einer Funktion veranlasst der Compiler selbstständig das Sichern und Wiederherstellen des Kontext. Hier müssen deshalb die Register explizit gesichert und wiederhergestellt werden.

`aspect_loaded` ist eine globale Variable, die anzeigt ob der ganze Aspekt gewoben wurde. `pointcut()` überprüft ob auf dem Stack der Kontext des Aspekts und der durch ihn erweiterten Funktion liegt. Nur wenn beide Bedingungen erfüllt sind wird der Aspekt ausgeführt.

Es sind drei Fälle von Aspekten zu unterscheiden:

1. Aufruf einer Funktion

```
save registers used by the hook
if ( aspect_loaded && pointcut() ) {
   restore registers used by the hook
   JMP advice_fn
} else {
   restore registers used by the hook
   JMP fn + 5
}
```

Wenn die Tests am Anfang positiv ausfallen wird die Advice-Funktion ausgeführt, die Aspektfunktion kehrt nach dem Aufruf automatisch zur vom Aspekt erweiterten Funktion zurück. Der Sprung tastet den Stack schließlich nicht an.
Sonst kehrt das Programm zum ersten Befehl (der ja nach dem 5 Byte großen Sprungbefehl steht) der durch den Aspekt erweiterten Funktion zurückgekehrt.

Die beiden anderen Fälle beinhalten den Zugriff auf eine globale Variable. Dazu wird der Zugriff selbst in eine Funktion umgewandelt (im folgenden mit `advice_fn` bezeichnet).

2. Lesen einer globalen Variable

```
save registers
if (aspect_loaded && pointcut() ) {
   CALL advice_fn
   MOVE eax r_tgt
   restore registers except r_tgt
} else {
   restore registers
   MOVE vaddr, r_tgt
}
JMP addr + move_instruction_size
```

Der Rückgabewert der `advice_fn` ist in eax abgelegt, das Programm erwartet den Wert der globalen Variablen im Register `r_tgt`. Es muss also noch die Zuweisung der Variablen erfolgen.
Wenn der Aspekt nicht aufgerufen wird, so wird einfach nur die Zuweisung durchgeführt: `vaddr` enthält die Adresse der Variable im Speicher, der Inhalt wird also ins Register `r_tgt` geladen.

Diese Vorgehensweise muss immer angewandt werden, wenn im Programm auf eine globale Variable zugegriffen wird. Dieser *„minimal hook“* tritt also nicht nur am Anfang einer Funktion auf.

3. Modifizieren einer globalen Variable

```
save registers
if (aspect_loaded && pointcut()) {
   PUSH r_src
   CALL advice_fn
   restore registers
   MOVE vaddr, r_src
} else {
   restore registers
   MOVE r_src , vaddr
}
JMP addr + move_instruction_size
```

Der Wert, der der Variablen zugewiesen werden soll, wird auf den Stack abgelegt und somit der `advice_fn` als Parameter übergeben. Nach Rückkehr aus der Funktion wird der Wert der Variablen zurück in das Register gespeichert, damit das Programm die Variable eventuell weiterverwenden kann (`vaddr` enthält auch hier die Adresse der Variable im Speicher).
Wenn der Aspekt nicht aufgerufen wird, so wird einfach nur die Zuweisung durchgeführt: die Variable wird vom Register in den Speicher kopiert.

Aspekte werden bei µDyner aktiviert indem die Sprungbefehle in den Funktionen verändert werden. Der Pentium kann aber nur 2 oder 4 Byte atomar schreiben, zum Ändern der „minimal hooks“ im Programmcode wird daher zuerst die 2. Hälfte der Sprungbefehle überschrieben.
Bei einem globalen Variablenzugriff muss gesondert verfahren werden: zuerst wird das 1. Byte mit dem INT3-Befehl überschrieben. Sollte also der unwahrscheinliche Fall eintreten, dass die Programmausführung auf Code trifft, der gerade modifiziert wird, so wird die Ausführung des Programms unterbrochen. Erst wenn der „minimal hook“ vollständig eingefügt wurde, wird der INT3-Befehl entfernt.

3.3 Laufzeitverhalten

Beim Einbinden eines Aspekts wird die meiste Zeit zum Laden der Bibliothek benötigt. Zur Laufzeit hat ein Programm nur 1-2% Overhead gegenüber einem C-Programm [2]. Ermittelt wurden die Werte mit der get()-Funktion zum Laden einer Webseite, einmal als C-Funktion, einmal als dynamischen Aspekt für µDyner.

3.4 C++-Erweiterung für µDyner

Hier soll nur kurz gezeigt werden, wie µDyner auch mit C++ umgehen kann [5]:

Den einzigen Unterschied, den es zu beachten gilt, ist die Art und Weise wie in C++ Symbole dargestellt werden. Dazu wird das Symbol aus Klassen- und Funktionsname generiert (*name-mangling“* genannt):
Aus der Funktion

```
class Bar {
   public:
        int ival();
  }
```

wird das Symbol `ival_3Bar` erzeugt.

Damit ist µDyner auch in der Lage mit C++-Code umzugehen. Der Präprozessor muss entsprechend modifiziert werden, dass er auch mit Klassen zurechtkommt und die *„minimal hooks“* an den richtigen Stellen in den Methoden einfügen kann. Andere Unterschiede zwischen C und C++ spielen keine Rolle, da zur Laufzeit der Binärcode direkt modifiziert wird. Es werden also wieder die Symbole aus der Binärdatei ausgelesen.

4. Zusammenfassung und kritische Betrachtung

DAO C++ und µDyner ermöglichen es auf den ersten Blick in sehr ähnlicher Weise Aspekte dynamisch in C oder C++-Programme einzufügen. Dazu muss in die Programme vor dem Kompilieren Code eingefügt werden. Bei genauerer Betrachtung zeigt sich, dass µDyner viel ausgefeiltere Methoden verwendet. Das zeigt sich dann auch im Laufzeitverhalten.

Die AOP-Eigenschften, die diese beiden Erweiterungen bieten sind sehr eingeschränkt. Nur die grundlegenden Ansätze zur AOP sind vorhanden: es gibt nur Before-Advice (bei DAO C++ auch After-Advice mit dem angesprochenen Synchronisationsproblem), keine Introduction, keine Aspektvererbung usw.

Bleibt die Frage wie sinnvoll es wirklich ist, dynamische Aspekte zu verwenden. Die Motivation zu einem solchen Projekt war es ja, einen Web-Cache um Prefetching-Mechanismen zu erweitern und dabei den Server möglichst nicht vom Netz nehmen zu müssen. Allerdings dauert es auf heutigen Systemen nur Sekunden das Programm anzuhalten, und eine andere Version (z. B. mit anderen Prefetchingstrategien) des gleichen Programms zu starten Der Anwender bekommt davon wahrscheinlich gar nichts mit. Außerdem ist nicht zu erwarten, dass die Prefetching-Strategien so oft ausgetauscht werden müssen. Mit statischen Aspekten entfällt der Overhead und es stehen sämtliche Features der AOP zur Verfügung.

Quellen:

DAO C++:
[1] Sufyan Almajali and Tzilla Elrad: *A Dynamic Aspect Oriented C++ using MOP with Minimal Hook Weaving Approach*, Proceedings of the 2004 Dynamic Aspects Workshop (DAW04), March 2004, Lancaster, England http://aosd.net/2004/workshops/daw/Proc-2004-Dynamic-Aspects.pdf (Seite 6-13)

µDyner:
[2] Marc Segura-Devillechaise, Jean-Marc Menaud, Gilles Muller Obasco, Julia L. Lawall: *Web Cache Prefetching as an Aspect: Towards a Dynamic-Weaving Based Solution*, Proceedings of the 2nd International Conference on Aspect-Oriented Software Development, March 2003, Boston, MA, USA (Seite 110-119)

[3] Marc Ségura-Devillechaise and Jean-Marc Menaud, *Caching Web Services: Aspect-Orientation to the Rescue International Workshop on Web Engineering at Networking 2002*, Pisa, Italy, May 2002
http://www.emn.fr/x-info/arachne/segura-menaud_webeng-networking2002.pdf

[4] Homepage zur Arache (ehemals µDyner)
http://www.emn.fr/x-info/arachne/pub.html

C++-Erweiterung:
[5] Yan Chen, *Aspect-Oriented Programming (AOP): Dynamic Weaving for C++*. 2003.Vrije Universiteit Brussel and École des Mines de Nantes, Aug 2003
http://www.emn.fr/x-info/arachne/ychen.pdf

OpenC++:
[6] S. Chiba. A Metaobject Protocol for C++. In Proc. ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), page 285-299, October 1995