Rekonfiguration durch dynamische aspektorientierte Programmierung

Martin Gumbrecht

13. Juni 2014

Motivation

- Dynamische Rekonfiguration von Softwaresystemen
  - Fehlerbehebung, Sicherheitsaktualisierungen, Weiterenwicklung
  - Ohne Neustart des Systems
- Dynamische aspektorientierte Programmierung
  - Mechanismen zum Einbinden von Code zur Laufzeit

Agenda

Aspektorientierte Programmierung

Dynamische aspektorientierte Programmierung
- Binärcodemanipulation
- Interpreterbasiert
- Quellcode-Instrumentierung

Implementierungen
- Betriebssysteme
- Eingebettete Systeme
- Mehrfädige Anwendungssysteme

Fazit

Agenda

Aspektorientierte Programmierung

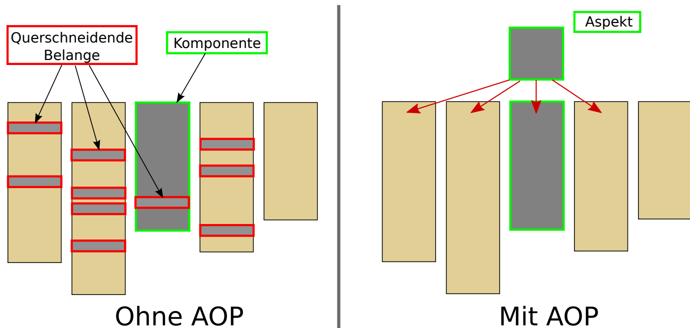
Dynamische aspektorientierte Programmierung
- Binärcodemanipulation
- Interpreterbasiert
- Quellcode-Instrumentierung

Implementierungen
- Betriebssysteme
- Eingebettete Systeme
- Mehrfädige Anwendungssysteme

Fazit

Aspektorientierte Programmierung

- Modularisierung querschneidender Belange
  - z. B. Logging, Fehlerbehandlung, Datenvalidierung
  - Kontextunabhängige Implementierung
- Automatisiertes Einfügen in den Code
  - An definierte Orte: z. B. alle/bestimmte Funktionsaufrufe
  - Verschiedene Typen: z. B. *before*, *after*, *around*
  - Realisiert durch Precompiler



Begriffserklärung

Advice

Modularisierter Aspekt-Code

```
logger.info("Method called")
```

Pointcut

Menge von Punkten, an denen ein Advice eingefügt werden soll

```
pointcut std_func_calls() = call("% std::% (...)");
```

Join-Point

Diskreter Punkt, an dem ein Advice aufgerufen wird

Weaver

Werkzeug zum Einbinden der Aspekte in den Code

Agenda

Aspektorientierte Programmierung

Dynamische aspektorientierte Programmierung
- Binärcodemanipulation
- Interpreterbasiert
- Quellcode-Instrumentierung

Implementierungen
- Betriebssysteme
- Eingebettete Systeme
- Mehrfädige Anwendungssysteme

Fazit

Dynamische aspektorientierte Programmierung

Dynamische AOP: Aspekte zur Laufzeit einbinden

- Nachladen von Aspekt-Code
  - Pointcuts und Advices definieren
- Dynamischer Weaver
  - Einbinden der Aspekte zur Laufzeit

Zur Rekonfiguration von Softwaresystemen:

- Hinzufügen von Funktionalität durch dynamische Aspekte
- Austausch von Funktionen durch Around-Advices

Binärcodemanipulation

- Auffinden der Join-Points im Binärcode
  - z. B. durch Symboltabellen
- Ersetzen von `call`-Instuktionen
  - Aufrufen von nachgeladenem Code
  - Synchronisation erforderlich

```
1 caller:
2    ...
3    call aspect
4    ...
```

```
1 aspect:
2    //before
3    call func
4    //after
5    ret
```

Problem:

Compiler-Optimierungen wie Code-Inlining und Symbol-Stripping

Binärcodemanipulation

+ Effizient
− Plattform- und Compilerspezifisch
− Nicht für alle Programmiersprachen/Compiler realisierbar

Interpreterbasiert

- Angepasste Laufzeitumgebung
  - Identifiziert Join-Points zur Laufzeit
  - Führt bei Bedarf Aspekt-Code aus
- Bei Bytecodeinterpretern i.d.R. Ausführung im Debugging-Modus

+ Sehr mächtig

– Angepasste Ausführungsumgebung erforderlich

– Hoher Ressourcenverbrauch

Quellcode-Instrumentierung

- Einfügen von Join-Points zur Compile-Zeit
  - Durch Precompiler
  - Durch generierte Proxy-Objekte
- Aufrufen eines Aspektmanagers am Join-Point
  - Auf nachgeladene Aspekte prüfen
  - Ausführen der Aspekte
- Dynamische Aspekte beim Aspektmanager registrieren

Quellcode-Instrumentierung

+ Plattformunabhängig
– Keine Definition von Join-Points zur Laufzeit
– Overhead auch bei ungenutzten Join-Points

Agenda

Aspektorientierte Programmierung

Dynamische aspektorientierte Programmierung
- Binärcodemanipulation
- Interpreterbasiert
- Quellcode-Instrumentierung

Implementierungen
- Betriebssysteme
- Eingebettete Systeme
- Mehrfädige Anwendungssysteme

Fazit

Rekonfiguration von Betriebssystemen

- Framework zur dynamischen Rekonfiguration von Betriebssystemen
  - Nutzt Binärcodemanipulation
  - Mehrere Aspekte pro Join-Point
  - Aspekt-Code als Kernel-Modul nachladen
- Beispielhafte Implementierung für Linux

Aspekt-Code als Kernel-Modul

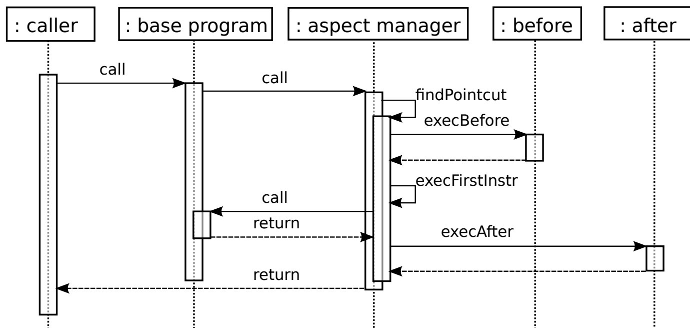
- Precompiler übersetzt Aspekte zu Kernel-Modulen (C-Code)
  - Adresse der Funktionen in Symboltabelle nachschlagen
  - Registrieren und Einbinden des Aspekts beim Laden

Übersetzter Aspekt-Code (vereinfacht)

```
1  #define ADDRESS 0xdeadbeef
2  #define TYPE before
3  void advice(void){
4      printk("this is a before-advice\n");
5  }
6  int init_module (){
7      register(advice, TYPE, ADDRESS);
8      weave(ADDRESS);
9  }
10 void cleanup_module(){
11     unweave(ADDRESS);
12     unregister(advice, TYPE, ADDRESS);
13 }
```

Aspektmanager

- Dynamischer Weaver
  - Erste Anweisung einer Funktion wird mit Aufruf des Aspektmanager ersetzt
- Aspektmanager
  - Alle Pointcuts mit zugehörigen Advices als Multiliste
  - Aspektmanager ruft zum Join-Point gehörende Advices auf



Analyse

- Analyse des Zeitbedarfs fehlt
  - Liste im Aspektmanager: vermutlich $\mathcal{O}(n)$
- Keine Implementierung der Synchronisation bei der Codemanipulation vorgestellt
- Keine Mechanismen zur Änderung von Datenstrukturen
  - Durchaus geeignet für kleinere Anpassungen und Bugfixes

Agenda

Aspektorientierte Programmierung

Dynamische aspektorientierte Programmierung
- Binärcodemanipulation
- Interpreterbasiert
- Quellcode-Instrumentierung

Implementierungen
- Betriebssysteme
- Eingebettete Systeme
- Mehrfädige Anwendungssysteme

Fazit

Rekonfiguration von eingebetteten Systemen

- Framework für ressourcensparende dynamische Rekonfigurierung
  - Produktfamilie mit konfigurierbarem Funktionsumfang
- Gleichartige Implementierung von statischen und dynamischen Aspekten (AspectC++)
  - Entscheidung ob ein Aspekt statisch oder dynamisch eingebunden wird beim Deployment treffen
- Plattformunabhängigkeit durch Quellcode-Instrumentierung

Instrumentationsaspekt

- Statischer Aspekt ruft den Runtime-Manager auf
  - Dieser führt dynamische Aspekte aus
- Pro Join-Point ein eigener Runtime-Manager
  - Ermöglicht konstanten Overhead
  - Realisiert durch Templates
- Mächtige Expression-Sprache zur Definition von Pointcuts
  - Reduziert den Overhead durch überflüssige Join-Points

```
1  aspect instrument{
2  pointcut virtual dynamicJPS = call("% std::%(...)");
3  public:
4      dynamicJPS():before(){
5          ArgsJnPnt<JoinPoint::ARGS> jp;
6          jp.joinpointName = JoinPoint::signature();
7          monitor<JoinPoint::JPID>::BeforeAdvice(&jp);
8      }
9  };
```

Dynamic AspectC++

- Dynamische Aspekte werden zu Shared-Library übersetzt
  - Zur Laufzeit beim Runtime-Manager registrieren
  - Bei Ausführung nachladen

Executable
RuntimeMonitor
RuntimeMonitor
RuntimeMonitor
joinpoint
joinpoint
joinpoint
AdviceRegister
list of before advices
list of around advices
list of after advices
Dynamic Aspect Libraries
compiler
Instrumented Code (C++)
Dynamic Aspects (C++)
static weaver (AspectC++)
compiler (dac++)
Weaver-Binding
Static Aspects
Base Program
Dynamic Aspects (AspectC++)

Analyse

- Effiziente Implementierung
  - Konstanter Laufzeitoverhead
  - 12 Byte zusätzlicher Speicher pro Join-Point
  - Bei vielen Join-Points immer noch problematisch
- Ziel: Nachladen von Funktionalität
  - z. B. Tracing-Aspekt zur Fehleridentifizierung
  - Austausch von Funktionen prinzipiell auch möglich
- Gut in bestehende Systeme integrierbar
  - Keine Voraussetzungen an das Design bestehender Anwendungen

Agenda

Aspektorientierte Programmierung

Dynamische aspektorientierte Programmierung
- Binärcodemanipulation
- Interpreterbasiert
- Quellcode-Instrumentierung

Implementierungen
- Betriebssysteme
- Eingebettete Systeme
- **Mehrfädige Anwendungssysteme**

Fazit

Rekonfiguration von mehrfädige Anwendungssystemen

- Framework zum Hinzufügen, Entfernen und Ändern von Komponenten
  - Komponente: Klasse mit öffentlicher Schnittstelle
  - Öffentliche Schnittstelle darf sich nicht ändern
  - Meachanismus für den Zustandstransfer
- Unterstützung für mehrfädige Programme
- Implementierung für .NET (LOOM.NET)

Rekonfigurationsaspekt

- Rekonfiguration durch Proxies
  - Erzeugt mit dem Factory-Pattern
  - Ergänzt Komponenten um Rekonfigurationsaspekt

```
1  public class ReconfAspect:Aspect,IConfigure{
2      private object target;
3      private RWLock rwlock=new RWLock();

5      [Call(Invoke.Instead)]
6      [IncludeAll]
7      public object Proxy(object[] args){
8          rwlock.AcquireReaderLock(-1);
9          if (target == null){
10             return Context.invoke(args);
11         } else {
12             Context.InvokeOn(target, args);
13         }
14         rwlock.ReleaseLock();
15     }
16     ...
17 }
```

Synchronisation der Rekonfigurierung

- Finden eines Rekonfigurationszeitpunktes durch Reader-Writer-Locks
  - Methodenaufrufe sind Leser
  - Rekonfiguration ist Schreiber
  - Rekursive Locks verhindern Deadlocks

```
1 public class RekonfAspect:Aspect,IConfigure{
2     ...
3     public void ReplaceReference(object target){
4         rwlock.AcquireWriterLock();
5         this.target = target;
6         rwlock.ReleaseWriterLock();
7     }
8 }
```

Zustandstransfer

- Zustandstransfer zwischen alter und neuer Komponente
  - Implementierung einer Export/Import-Funktion
  - Objektgraph traversierender Algorithmus
    - Rekursives Kopieren der Member-Variablen
    - Ersetzen durch neue Versionen der Objekte
    - Erkennen von Zyklen

Analyse

- Konstanter Zeitverbrauch ohne Rekonfiguration
  - Dauer der Rekonfiguration von laufenden Anfragen abhängig
- Grobgranularer Austausch von Komponenten
  - Striktes Komponentenmodell erforderlich
  - Keine Referenzen auf interne Datenstrukturen nach außen geben
- Eigentlich keine dynamische AOP erforderlich
  - Rekonfigurationsaspekt ist zur Compilezeit definiert

Agenda

Aspektorientierte Programmierung

Dynamische aspektorientierte Programmierung
- Binärcodemanipulation
- Interpreterbasiert
- Quellcode-Instrumentierung

Implementierungen
- Betriebssysteme
- Eingebettete Systeme
- Mehrfädige Anwendungssysteme

Fazit

Fazit

- Umfassende Möglichkeiten zur Rekonfiguration durch AOP
  - Zum Teil reicht statische AOP bereits aus
- Jedes Konzept hat Vor- und Nachteile
  - Betriebssysteme: Auch für komplexe Systeme im Nachhinein realisierbar
  - AspectC++: ressourcenschonend
  - LOOM.NET: Austausch von Komponenten

⇒ Sowohl ressourcenschonende als auch mächtige Ansätze

⇒ Noch keine einheitliche Lösung für alle Anwendungsfälle