METAPROGRAMMIERUNG

Studientag 01853

Moderne Programmiertechniken und -methoden

METAPROGRAMME

- Programme, die andere oder sich selbst bei der Ausführung beobachten oder manipulieren
  - Compiler, Interpreter, Debugger, Refactoring-Tools
- dazu gehört auch selbstmodifizierender Code (s. KI)
  - dieser hat allerdings eingeschränkte Möglichkeiten der Verifikation

VORAUSSETZUNG

- Programme haben die Form von Daten
  - einfacher Fall, wenn es von der Sprache aus keinen Unterschied zwischen Daten und Funktionen gibt (s. funktionale oder logische Sprachen)

ANWENDUNGSGEBIETE I

- Spracherweiterungen und -anpassungen
  - Präprozessoren (C/C++), Smalltalk
- Programmiertools
- KI

ANWENDUNGSGEBIETE II

- dynamische Konfiguration von Systemen und Diensten
  - Schnittstellen sind zum Zeitpunkt der Bindung nicht bekannt
- genetische Programmierung
  - Mutation (zufällige Änderung) des Programms

REFLECTION

- Programm (er)kennt sich selbst
- schlecht lesbar
- schlecht wartbar
- Grenze zwischen Übersetzungs- und Laufzeit verwischt

REFLECTION II

- Introspektion
  - Beobachtung der eigenen Ausführung
  - Ermittlung der beteiligten Klassen und Attribute ohne Zugriff auf den Quellcode
  - dynamischer, vom Compiler nicht geprüfter Methodenaufruf
  - Javas Art der Reflection

REFLECTION III

- Interzession
- Programm kann eigenen Ablauf beeinflussen
- AOP arbeitet auf Basis der Interzession
- in Java nicht direkt möglich, sondern nur über Proxy-Pattern oder Spracherweiterungen

REFLECTION IV

- Modifikation
  - Programm kann eigene Bestandteile Ändern, Ergänzen oder Austauschen
  - Programm muss zugriff auf Compiler/Interpreter haben
  - in Java: Quellcode generieren, über Systemaufruf compilieren und über eigenen Classloader importieren
    - oder gleich Bytecode direkt erzeugen (s. Mock-Objekte)

METADATEN

- Ausgangspunkt
  - bei Programmen, die für Introspektion geschrieben werden, muss sich der Programmierer darüber bewusst sein (z.B. Test-Methoden in JUnit 3), sonst funktionieren sie nicht
  - dem Programm sieht man das aber nicht an
- Metadaten sind nur dann solche, wenn sie nicht auf der selben Logischen Ebene wie das beschriebene Element liegen

METADATEN II

- Annotationen (Java) bzw. Attribute (.NET)
- sind sowohl Typ als auch Objekt
- können Meta-Annotationen besitzen (@Retention)
- Marker-Annotation: hat keine Attribute

METADATEN III

- Annotation Processing Tools (APT)
  - quasi Präprozessor, der zu einer Annotation einen Annotationsprozessor ausführt, welcher ggf. Quellcode anpasst
  - im Compiler: erfordert Anpassung desselben
  - außerhalb des Compilers: doppelte Erstellung des AST

AOP

- Isolation von crosscutting concerns (Logging, Autorisierung)
- Entwicklungsgeschichte
- Strukturierung von Programmen durch Unterprogramme
- dann mit Modularisierung
- dann mit Aspekten: gleichzeitige Strukturierung des Programms nach verschiedenen Ordnungen

AOP II

- Voraussetzung: an mehreren Stellen im Programm wird derselbe konzeptuelle Aspekt auf die gleiche Weise implementiert

- Aspekte sind i.d.R. nichtfunktionale, also Qualitätseigenschaften

AOP III

- Ziel: Vermeidung von
- Verstreuung (Scattering) von Code gleicher Funktionalität
- Verwindung (Tangling) von Code, der mehrere Funktionalitäten abdeckt

AOP IV

- Ansätze zur Vermeidung von Verstreuung und Verwindung
  - symmetrische Teilung: querschneidende Funktionalitäten geben Modularisierung des Programms vor
  - asymmetrische Teilung: eigene Module für Askpekte, die die querschneidende Funktionalität kapseln
- die (a)symmetrische Trennung muss spätestens zur Laufzeit wiederhergestellt werden ➡ Webeplan

WEBEPLAN (WEAVING PLAN)

- extensionale Spezifikation: Angabe der zu verwebenden Stellen
- Bsp. in AspectJ:

```
pointcut what():
call(void Example.test(int));
```

- intensionale Spezifikation: Beschreibung der Webstellen durch Prädikate
- Bsp. in AspectJ:

```
pointcut what():
call(* Example.set*(..));
```

AOP VI

- Aspektorientierung ist also
  - quantification (nur bei intensionaler Spezifikation): potentiell unendlich viele Stellen, an denen Aspekte zum Einsatz kommen
  - obliviousness: von Aspekten betroffene Stelle wissen nichts davon

PROBLEME DER AOP

- AOP unterstützt keine Interfaces
  - Programmteile können sich nicht gegen einweben von Aspekten wehren, dafür können Aspekte keine Interfaces erwarten
- Webeplan verletzt Geheimnisprinzip und muss so bei Änderungen berücksichtigt werden
- ausgelagerte Aspekte brauchen ggf. Zugriff auf den Kontext → es entstehen Abhängigkeiten über Modulgrenzen

PROBLEME DER AOP II

- schränkt Lesbarkeit von Programmen enorm ein, da mit strukturellem Programmfluss komplett gebrochen wird

- sowohl Vorgänger als auch Nachfolger einer Anweisung sind ohne Webeplan nicht auszumachen

- ggf. kann eine neue Methode „zufällig“ und unbeabsichtigt in die intensionale Spezifikation fallen

ASPECTJ

- Klasse kann durch Inductions um Instanzvariablen und Methoden (die im Aspekt notiert sind) erweitert werden
- Funktionalität heißt Advice, Definition der Webestelle heißt Pointcut
- Joinpoint ist mögliche Einwebestelle im Code
- Pointcuts können auf Call-Stack, das aktuelle Empfängerobjekt und die aktuellen Parameter zugreifen

AOP UND ENTWURFSMUSTER

- Entwurfsmuster haben konstanten und variablen Teil
- konstanter Teil lässt sich mit Aspekten umsetzen und so in Klassen injizieren