4 Übungsaufgabe #4: Aspektorientiere Signalverarbeitung & Konfigurierbarkeit

4.1 Aspektorientierte Signalverarbeitung

Nachdem in der Vorlesung die Programmiersprache AspectC++ vorgestellt wurde, sollt ihr nun damit im Rahmen dieser Aufgabe erste Erfahrungen sammeln. Darüberhinaus wurde im Zusammenhang von Softwareentwicklungsprozessen die Softwareentwicklung mit Produktlinien vorgestellt. In dieser Aufgabe sollen Teile aus diesem Prozess eingeübt werden.

4.1.1 Signalverteilung

In Aufgabe 2 wurden verschiedene Varianten der Signalverteilung für unterschiedliche Signale implementiert. Diese Implementierungen sollen nun mit Hilfe von Aspekten so umgebaut werden, dass man durch entsprechende Parametrierung eines Aspekts die Signalverteilung für jede Signalquelle beliebig einstellen kann.
Dabei bietet es sich an für jede (unterstützte) Signalnummer eine eigene Klasse zu erstellen, welche die Unterbrechungsbehandlung für das jeweilige Signal bereitstellen soll. Die Behandlungsfunktion sollte dann eine statische Methode dieser Klasse sein, um dem Interface für `sigaction` zu entsprechen. Die Signatur der statischen Funktion kann nun innerhalb eines *Pointcut*-Ausdrucks verwendet werden, um einen Aspekt an die Behandlung einer bestimmten Signalnummer zu binden.
Durch die Verwendung von *pure virtual Pointcuts* und Vererbung auf Aspekten kann nun erreicht werden, dass die Implementierung des Verteilungsmechanismus von der konkreten Signalquelle getrennt ist und durch Spezialisierung des Verteilungsaspektes ein beliebiges Signal als Quelle ausgewählt werden kann.

4.1.2 Signalbehandlung (optional für 7.5 ECTS)

Ähnliches soll für die in Aufgabe 3 implementierte zweistufige Interruptbehandlung realisiert werden. Die Implementierung der beiden Varianten soll nun in jeweils einen Aspekt refaktorisiert werden. Abhängig davon, welcher der beiden Aspekte aktiviert ist, geschieht die Abarbeitung der *SLIH*s nun auf der unterbrochenen virtuellen CPU, die auch den *FLIH* abgearbeitet hat, oder zentral auf einer vorher ausgewählten.

Aufgaben:

→ Auf beliebige Quellen anwendbare Signalverteilung durch die Verwendung von aspektorientierter Programmierung.

→ Refaktorisierung der zweistufigen Signalbehandlung aus Aufgabe 2 in Aspekte (optional für 7.5 ECTS)

Hinweise:

- Die Implementierung von Aspekten in `*.ah`-Dateien benötigt ebenfalls Include-Guards.
- *Extension-Slices* sollten am besten in einer eigenen `*.ah`-Datei implementiert werden. Zur Verwendung in Aspekten dann einfach mit `#include` in die entsprechende Datei einfügen.
- *Join-Points* für kontrollflussverändernde *Advice* bezieht sich immer auf den Aufruf oder die Ausführung von Funktionen. Man kann also nur am Beginn, Ende oder beim Aufruf von Funktionen Aspektcode einweben. Dementsprechend müssen sämtliche Stellen, die man mittels Aspekten beeinflussen möchte, entsprechende Eigenschaften haben.
- **Toolchain:** Unter `/proj/i4bst/bin` findet ihr den Aspektcompiler. In `/proj/i4bst/tools/aspectc++` könnt ihr euch auch einige AspectC++-Beispiele anschauen. Am einfachsten in euer Makefile lässt sich der `ag++` integrieren. Dazu müsst ihr nur die Übersetzeraufrufe des `g++` durch `ag++` ersetzen. Auch Abhängigkeitsregeln lassen sich mit `ag++` analog zu `g++` mit Hilfe der `-M` und `-MM` Kommandozeilenparameter erzeugen.
- Mit der Option `-p` kann man den Pfad angeben, unter dem der `ag++` die Aspekte sucht, die er in die er in die zu übersetzende Datei einwebt. Gibt man diese Option nicht an, so wird das aktuelle Arbeitsverzeichnis verwendet.
- Dokumentation über AspectC++ findet ihr auf `http://aspectc.org/Documentation.5.0.html`. Nützlich sind vor allem das *AspectC++ Language Quick Reference Sheet*, eine kurz gefasste Übersicht über alle Sprachmerkmale und die *AspectC++ Language Reference*, welche alle Sprachmerkmale detailiert beschreibt.

4.2 Merkmalmodell für die Übungsaufgabe

In dieser und vergangenen Übungsaufgaben wurde die Umsetzung von alternativen Lösungen gefordert. Diese Variabilität soll nun in einem Merkmalmodell, wie sie in der Vorlesung vorgestellt wurden, beschrieben werden. Ein Merkmalmodell ist ein azyklischer Graph, in dem die folgenden Elemente erlaubt sind: Zwingende Merkmale, wahlfreie Merkmale, Gruppen von Alternativen, Gruppen von Anhäufungen sowie weitere Einschränkungen und Abhängigkeiten.
Die Umsetzung erfolgt in der Sprache des Linux Konfigurationswerkzeugs `Kconfig`.

4.3 Das Linux Konfigurationswerkzeug: `make xconfig`

Die Entwicklung mit Produktlinien sieht vor, dass auch nicht-Domänenexperten spezifische Lösungen generieren können. In Linux wurde dazu ein spezielles Werkzeug `Kconfig` entwickelt, welches es den Benutzern erlaubt, den Kern (graphisch) zu konfigurieren. Dieses Werkzeug wird in den Linux Quellen mit den Kommandos `make menuconfig` oder `make xconfig` aufgerufen. `Kconfig` stellt sicher, dass die vom Benutzer getroffene *Selektion* im Variantenmodell *valid* ist. Aus dieser Selektion werden dann einbindbare *Header*-Dateien und *Make*-Fragmente erzeugt, die Sie in Ihren Übersetzungsprozess integrieren können und sollen.
In der Übungsaufgabe wurden soweit alternative Implementierungen von der Signalzustellung (welche Prozessor verarbeitet welche Signale) sowie der Verarbeitungsmodus für SLIH (auf dem selben Prozessor wie der FLIH—**symetrischer** Betrieb—oder auf einem dediziertem, ausgezeichnetem Prozessor—**asymetrischer** Betrieb, *nur 7.5 ECTS*). Beide Entscheidungen über den Implementierungsvarianten werden zur Übersetzungszeit getroffen und sollen nun für den Benutzer ihrer Implementierung mit den (grafischen) Werkzeugen aus Linux bedienbar gemacht werden.
Wie weiter oben bereits gefordert, sind Teile der Lösung mit AspectC++ umzusetzen. Da der Aspektweber grundsätzlich alle Aspektdateien im *Projektverzeichnis* einwebt, muss dieses konfigurationsabhängig konstruiert werden. Hierzu stellen wir euch ein (dokumentiertes) Konfigurationssystem zur Verfügung. Damit könnt ihr ein Feature aus der Kconfig-Beschreibung durch Verwendung eines Familienmodells auf eine Menge von Implementierungsdateien abbilden.

Aufgaben:

→ Modelliert die soweit vorhandene und implementierte Variabilität in Form eines Variantenmodell als Merkmaldiagramm (am Besten auf Papier).

→ Implementiert dieses Modell mit dem Linux-Konfigurationsprogramm `Kconfig`.

→ Integriert das Linux-Konfigurationsprogramm in eure vorhandenen Makefiles.

→ Arbeit euch in das zur Verfügung gestellte Konfigurationssystem ein um mit Hilfe der mit `Kconfig` erzeugten Konfiguration ein Projektverzeichnis zu konstruieren, in dem nur die Aspekte zur Übersetzung verwendet werden, die für die selektierte Konfiguration relevant sind.

Hinweise:

- Eine bereits vorübersetzte ausführbare Version des Linux Konfigurationswerkzeugs Kconfig liegt im CIP-Pool unter `/proj/i4bst/tools/bst-kconfig`. Er wird wie folgt gestartet: `mconf features.fm` und erzeugt dann aus euerer Feature-Auswahl eine `.config` Datei.
- Ebenfalls unter `/proj/i4bst/tools/bst-kconfig` findet ihr das Werkzeug `transform.pl`. Mit diesem könnt ihr einzelne Features auf Dateimengen abbilden. Der Aufbau des Familienmodells, das die Abbildung von Features in Kconfig auf Dateien festlegt, und die Bedienung des Werkzeugs sind in `bst_transform.pdf` dokumentiert. Darüberhinaus sind dort auch zwei Beispiele zu finden, an denen ihr den Aufbau der Konfigurationsdateien und das Zusammenspiel mit Kconfig nachvollziehen könnt.

4.4 Abgabe: am 29.06.2011