Karlsruher Institut für Technologie

Modularisierung der Reactions-Sprache Code-Review

Praktikum „Software Quality Engineering mit Eclipse“, Wintersemester 2017/18
Lukas Hennig, Betreuer: Heiko Klare

SOFTWARE-ENTWURF UND -QUALITÄT
INSTITUT FÜR PROGRAMMSTRUKTUREN UND DATENORGANISATION, FAKULTÄT FÜR INFORMATIK

Die Reactions-Sprache

- Teil des Vitruvius Frameworks
  - Automatisierte, änderungsgetriebene Konsistenzerhaltung verschiedener Modelle
- Xtext-basierte DSL, .reactions-Dateien
  - **Segmente:** Metamodell-Paar, Reaktionen, Routinen
  - **Reaktionen:** Reagieren auf Änderungen im Quellmodell, ändern Zielmodell, rufen Routinen auf
  - **Routinen:** Ändern Zielmodell, rufen andere Routinen auf
- Aktuelle Einschränkungen:
  - Keine Referenzierung zwischen Reactions-Dateien
  - Konsistenzregeln nicht wiederverwendbar
  - → Copy&Paste von Reaktionen und Routinen + Anpassungen

Modularisierung: Import-Mechanismus

- Segmente können andere Segmente „importieren“
  - Reaktionen werden übernommen
  - Routinen sind aufrufbar, mittels einfachen Namen

- Erweiterungen:
  - Option: Verwendung qualifizierter Routinen-Namen (bei Namenskonflikten)
    - importedSegment.routine()
    - importedSegment.transitiveImportedSegment.routine()
  - Importieren von nur Routinen
  - Überschreibbare Reaktionen und Routinen
    - = Ersetzen von Reaktionen/Routinen (aktuell kein Aufrufen der überschriebenen Reaktion/Routine möglich)
    - Auslagern von „Ketten“ von Routinenaufrufen, mit modifizierter Routine als Element
    - „Deaktivieren“ von Reaktionen durch „leere“ Implementierung

Import-Mechanismus: Reaktionen vs Routinen

- Reaktionen: Nur einmal importierbar in gesamter Importhierarchie
  - Keine Konflikte zwischen Imports aufgrund mehrmals importierter und überschriebener Reaktionen
  - Override-Syntax: ::<ReactionName> { … }

- Routinen: Mehrmals importierbar von verschiedenen Segmenten in Importhierarchie
  - „Routinen-Bibliotheken“
  - Einschränkung: Keine Namenskonflikte erlaubt
  - Überschreibungen wirken nur entlang „Importpfad“
  - Override-Syntax: <ImportPath>::<RoutineName> { … }
    - ImportPath: (.)*

Importierte Routinen: Beispiel

- Importhierarchie: A→{B, C, D}, B→{C}, D→{C}
  - „→“ repräsentiert Routinen-Import mit qualifizierten Namen: Keine Namenskonflikte
  - C = „Routinen-Bibliothek“
  - Annahme: A überschreibt Routine: B.C::eineRoutine() { … }
  - Ergebnis: Überschreibung betrifft Aufrufe entlang Importpfad „B.C“:
    - A→B.C
    - A→B→C
  - Keine Auswirkungen auf:
    - A→C
    - A→D→C

Implementierung: Bisher

- Eclipse Projekt
  - Reactions-Dateien: Segmente
- Pro Segment: **Executor**
  - Liste aller Reaktionen
- Pro Segment: **RoutinesFacade**
  - Enthält Methoden zum Aufrufen von Routinen
- Pro Metamodell-Paar: **ChangePropagationSpecification**
  - Wird von Vitruv Application registiert
  - Wird von Vitruv über Modelländerungen informiert
  - Informiert Executors

- **Vitruv → ChangePropagationSpecifications → Executors → Reaktionen → Routinenfacade → Routinen → Routinenfacade → Routinen → ...**

Implementierung: Importierte Reaktionen

- **Executor**
  - Liste aller Reaktionen
- Tiefensuche in Importhierarchie
  - Liefert: Importierte Reaktionen, Importpfade
  - Backtracking: Austausch von überschriebenen Reaktionen
- Ergänzen eigener Reaktionen in **Executor**
- Importpfade: Spezifizieren verwendete Routinenfacaden (und Überschreibungen)
  - (Siehe: Implementierung überschriebener Routinen)

Implementierung: Importierte Routinen

- **Routinenfacade**
  - Enthält Methoden zum Aufrufen von Routinen
    - Mittels einfachen Namen
- Routinen ohne qualifizierten Namen
  - Ergänzen eigene Routinen in Routinenfacade
- Routinen mit qualifizierten Namen
  - Öffentliche Felder in Routinenfacade:
    - Typ: Routinenfacade des importierten Segments
    - Name: Name des importierten Segments
  - → „importedSegment.routine()“
  - → „importedSegment.transitiveImportedSegment.routine()“

Implementierung: Überschriebene Routinen

- **Erweiterte Routinenfacade**
  - Für jedes Segment, aus dem Routinen überschrieben wurden
  - Erbt von erweiterten Routinenfacade des nächsten Segments entlang Importpfad, das ebenfalls Routine(n) überschreibt
    - Oder von originaler Routinenfacade
  - Enthält Methoden zum Aufrufen der überschriebenen Routinen
    - Überschreiben vorherige/originale Routinen-Methoden in überschriebener Routinenfacade

- Bisher: Reaktionen/Routinen verwenden jeweils eigene Instanz der eigenen Routinenfacade
  - Jetzt: Bekommen Routinenfacaden-Instanz vorgegeben
  - Zuständigkeit: RoutinesFacadesProvider

Implementierung: Überschriebene Routinen

- RoutinesFacadesProvider
  - Für jedes Segment
  - Wird vom Executor erzeugt
    - „Ausführendes Segment“ (Wurzel) gibt verwendeten RoutinesFacadesProvider vor
  - Liefert (erweiterte) Routinenfacade für gegebenen Importpfad

- Verwendet von:
  - Executor: Für enthaltene Reaktionen
  - Routinenfacade: Für aufgerufene Routinen

Und nun ...

Karlsruher Institut für Technologie

zum Quellcode