Gerald Bründl

Was ist ein Zustandsdiagramm?

Das Zustandsdiagramm stellt einen endlichen Automaten in einer UML Sonderform grafisch dar und beschreibt die möglichen Folgen von Zuständen eines Modell-Elements (Objekts) einer bestimmten Klasse während seiner Lebensdauer oder während dieses eine bestimmte Operation ausführt.

Anwendungen für Zustandsdiagramme

Zustandsdiagramme sind geeignet um das Verhalten von Komponenten oder Systemen darzustellen. Außerdem verwendet man sie um die zulässige Nutzung der Schnittstelle eines Systems zu spezifizieren (welche Ereignisse verursachen wann welchen Zustandsübergang).

Woraus besteht ein Zustandsdiagramm?

Die Zustände (Knoten) stellen den aktuellen Systemzustand dar.
Sie werden über Transitionen (Zustandsübergänge) miteinander verbunden die durch Ereignisse ausgelöst werden.
Für jeden Zustandsübergang können Aktivitäten angegeben werden die dann beim Übergang ausgeführt werden zum Beispiel das inkrementieren einer Variable.
Drei vordefinierte Ereignisse sind das entry Ereignis das beim betreten eines Zustands ausgelöst wird, das do Ereignis das ausgelöst wird währen sich System in einem Zustand befindet und das exit Ereignis das beim Verlassen eines Zustands ausgeführt wird.
Jede Transition hat immer einen Quell und einen Zielknoten,
wobei diese auch identisch sein können wodurch man entweder eine Selbsttransition (mit entry/exit Aktivitäten) oder eine innere Transition (ohne entry/exit Aktivitäten) erhält.
Bei jeder Transaktion können durch optionale Guard Ausdrücke zusätzliche Bedingungen gesetzt werden.
Aus einfachen Zuständen lassen sich durch verschachteln komplexe Zustände aufbauen,
dabei müssen die Subzustände disjunkt sein (nur einer aktiv) oder sich in verschiedenen Regionen befinden.
Es gibt einige spezielle Zustände wie den Startzustand bei dem der Ablauf beginnt und der nur eine ausgehende und keine eingehenden Transitionen besitzen darf und den Endzustand der ebenfalls keine ausgehenden Transitionen haben kann.
Daneben verwendet man Historie Zustände, die sich den internen Zustand in einem komplexen Zustand merken von dem aus die letzte Transition ausging und in den man bei bedarf zurückkehren kann.
Der Kontrollfluss kann durch Entscheidungsknoten (if Abfrage) und Parallelisierung und Synchronisierungsknoten gesteuert werden.

Notation einer Transition:

Ereignis(Argumente) [Bedingung] / Aktivität(en)

Geeignete Metriken

Größen Metriken:

Gesamtanzahl der Zustände einer Klasse
Gesamtanzahl der entry Aktivitäten
Gesamtanzahl der exit Aktivitäten
Gesamtanzahl der do Aktivitäten
Gesamtanzahl der einfachen Zustände (inklusive einfachen Zuständen in zusammengesetzten Zuständen)
Gesamtanzahl der zusammengesetzten Zustände
Gesamtanzahl der Ereignisse
Gesamtanzahl der guard conditions

Strukturelle Komplexitäts- Metriken:

Gesamtanzahl der Transitionen
Cyclomatic Number of McCabe: |einfache Zustände| - |Gesamtanzahl Transitionen| + 2

Geeignete Smells

Zustände ohne eingehende Transition
Unbenannte Zustände

Geeignete Refactorings

Fold Incoming/Outgoing Actions
Unfold Entry/Exit Action
Fold Outgoing Transitions
Unfold Outgoing Transitions
Zustand in zusammengesetzten Zustand einfügen
Zustand aus zusammengesetzten Zustand herausholen
Zustand umbenennen
Isolierten Zustand entfernen
Innerer Transitionen hineinziehen falls kein exit oder entry Aktivitäten
Zustände vereinigen
Zusammengesetzten Zustand bilden (gruppieren)

Refactoring Constraints

Die Refactorings können nur durchgeführt werden wenn bestimmte Constraints erfüllt sind. Damit wird sichergestellt das sich Verhalten nach Refactoring nicht verändert hat.

Begründungen für die Auswahl an Metriken Smells und Refactorings

Metriken

Da die Komplexität eines Zustandsdiagramms immer von allen Elementtypen die darin vorkommen abhängt ist es wichtig das Metriken für alle diese Typen vorhanden sind..

Deswegen wurden Metriken gewählt die Anzahl der Aktivitäten, Ereignisse, usw. widerspiegeln. Um etwas über die Strukturelle Komplexität (Quervernetzung) des Zustandsdiagramms aus zusagen werden alle Transitionen gezählt.

Smells

Ein Zustand ohne eingehende Transition wird nicht gebraucht und kann durch Refactoring entfernt werden und ein Zustand ohne Namen kann durch Refactoring benannt werden. Beides verbessert die Verständlichkeit des Diagramms.

Zustand bei dem alle eingehenden Transitionen gleiche Aktivität haben → entry Aktivität.

Zustand bei dem alle ausgehenden Transitionen gleiche Aktivität haben → exit Aktivität.

Alle Transitionen aus Komplexen Zustand zu gleichen äusseren.

Innerer Transitionen → Aktivitäten hineinziehen

Refactorings

Die Möglichkeit Zustände umzubenennen und isolierte Zustände zu entfernen wird benötigt um die Smells zu beheben.

Zustände Vereinigen und Zusammengesetzte Zustände bilden sind wichtige Refactorings um die Übersichtlichkeit zu erhöhen.

Fold und unfold incoming / outcomming actions sowie fold und unfold incomming / outgoing transitions braucht man um Aktionen und Transitionen zusammenfassen zu können womit man die Strukturelle Komplexität des Diagramms durch Verminderung der Transitionspfeile verbessert.. Das Gleiche gilt für das hineinziehen von inneren Transition bei Zuständen ohne entry und exit Aktivitäten.