

Rekonstruktion von Architekturansichten: Erfahrungen mit der Relation Partition Algebra

Kay Schützler, Humboldt-Universität zu Berlin
schuetzl@informatik.hu-berlin.de

1 Einleitung

Symphony, das in [1] beschriebene Framework zur ansichtsorientierten Rekonstruktion von Softwarearchitekturen, basiert auf drei wesentlichen Konzepten: dem Quell-Blickwinkel, dem Ziel-Blickwinkel und der Zuordnung von Elementen des einen zu Elementen des anderen Blickwinkels.

Ausgangspunkt für den Quell-Blickwinkel und seine Ansichten sind Implementationsfakten, die durch geeignete Relationen über Elementen des Quelltextes beschrieben werden und im Idealfall direkt aus den Quellen ableitbar sind.

Der Zielblickwinkel beschreibt eine an den Bedürfnissen des Reengineering-Vorhabens ausgerichtete Darstellung der rekonstruierten Software-Architektur.

Unter den (halb-)automatischen Techniken zur Ableitung der Ziel-Ansichten aus den Quell-Ansichten wird in [1] speziell die Relation Partition Algebra (RPA [2], S. 29ff) herausgestellt, die sich zur Beschreibung von Transformationen zur Gewinnung von Ansichten, zur Überprüfung der Übereinstimmung von Annahmen über die Architektur mit der Realität, aber auch zur Ableitung von Abhängigkeiten auf höheren Abstraktionsebenen aus den Fakten darunterliegender Ebenen eignet.

In diesem Beitrag werden Erfahrungen mit der letztgenannten Anwendung der RPA beschrieben. Untersuchungsgegenstand waren Möglichkeiten zur automatischen Gewinnung abstrahierter Architektur-Informationen für eine bestimmte Klasse von Software-Systemen.

2 Ausgangssituation

Im Rahmen eines Architekturbewertungsprozesses für hardwaresteuernde Software soll unter anderem die tatsächlich vorhandene Software-Architektur des untersuchten Systems bestimmt werden.

Hardwaresteuernde Software ermöglicht einem Nutzer die interaktive Steuerung eines Mikroprozessorbasierten Geräts (auch im Zusammenspiel mit weiteren Geräten). Diese Steuerung umfasst die Konfiguration von Geräte-Parametern, die den Zustand oder das Verhalten des Geräts/der Geräte beeinflussen, die Überprüfung und Veränderung aktueller Geräte-Zustände sowie die Überwachung des Geräte-Verhaltens.

Aus dieser generellen Anforderungsbeschreibung lässt sich bereits eine grundlegende Strukturierung ab-

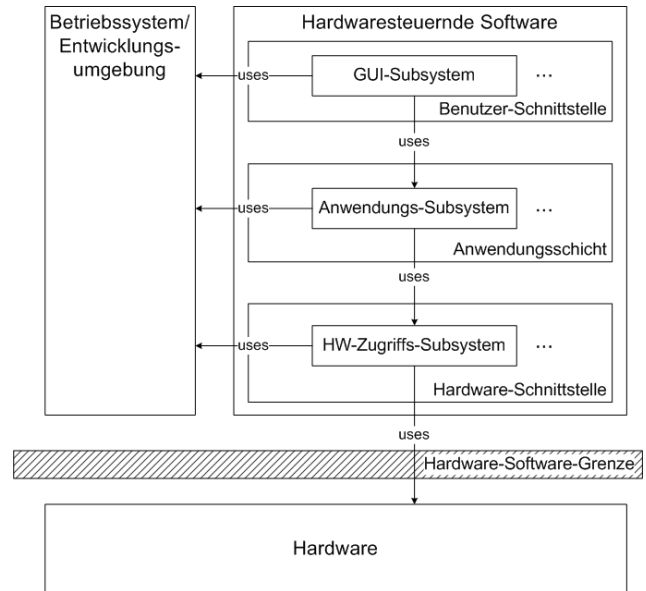


Abbildung 1: Grundlegende Strukturierung hardwaresteuernder Software-Systeme

leiten, die in Abbildung 1 wiedergegeben ist. Die dargestellte „Uses“-Relation zwischen den Subsystemen soll aus den vorliegenden Quelltexten extrahiert werden. Dabei bedeutet „Subsystem A uses Subsystem B“, dass die korrekte Funktion des Subsystems A vom Vorhandensein (und damit ebenfalls von der korrekten Funktion) des Subsystems B abhängt. A kann also nicht ohne B verwendet werden.

Die dargestellte Struktur ist idealisiert und wird nur selten in realen Software-Systemen anzutreffen sein. Dies gilt besonders für die klare Abgrenzung zwischen den einzelnen Schichten und das Vorhandensein einer mittleren Anwendungsschicht, die beispielsweise die softwareinterne Darstellung eines beweglichen Probenhalters von den dabei tatsächlich eingesetzten Motoren abstrahieren lässt.

3 Vorgehen

Zu Beginn ist zu definieren, wie sich die gesuchte „Uses“-Relation in den Beziehungen der Software-Elemente manifestiert. Natürlich sind die entsprechenden, im Quelltext vorzufindenden Relationen sprachabhängig. Für Java beispielsweise wurden bisher die folgenden Relationen betrachtet:

- eine Funktion ruft eine andere Funktion auf oder benutzt (lesend) eine Variable,
- eine Variable wird von einer Funktion gesetzt oder ist von einem bestimmten Typ,
- eine Klasse wird durch eine andere erweitert (Vererbung) oder implementiert ein Interface.

Da diese Relationen auf sehr niedrigem Abstraktionsniveau liegen, sind die extrahierten Relationen sehr umfangreich und nicht direkt nutzbar. Die Übertragung der Relationen auf Datei- und Paket-Ebene mittels des RPA-Lift-Operators bringt selten mehr Übersicht, da es sich dabei nicht um eine informationserhaltende Transformation handelt. Abbildung 2 verdeutlicht das Problem und einen ersten Ansatz zur Lösung.

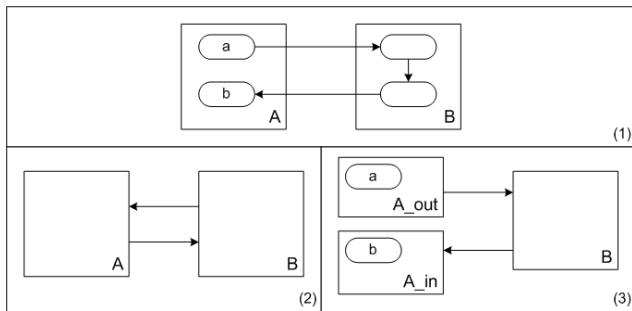


Abbildung 2: Aus der Relation auf unterer Ebene (1) wird durch den Lift-Operator eine gegenseitige Abhängigkeit von A und B erzeugt (2), die sich durch Teilung von A vermeiden ließe (3)

Die Vermeidung der Einführung unnötiger gegenseitiger Abhängigkeiten durch die vorgenommenen Transformationen macht sich besonders bezahlt, wenn die entstandene Relation über den Subsystemen als Abhängigkeits-Graph interpretiert wird und dieser anhand seiner stark zusammenhängenden Komponenten ([3]) in topologisch sortierbare Cluster zerlegt wird. Aufgrund der strukturellen Eigenschaften hardwaresteuernder Systeme sind in einem solchen Graphen die Elemente der Benutzer-Schnittstelle die bezüglich der topologischen Ordnung kleinsten Knoten, die Elemente der Hardware-Schnittstelle finden sich unter den größten Knoten.

Die in Abbildung 2 dargestellte Teilung von „A“ lässt sich mit den Mitteln der RPA automatisch vornehmen. Dazu kann die in [2] (S. 89ff.) beschriebene Technik zur Bestimmung von Used- und Using-Interfaces verwendet werden. Im Beispiel bildet „a“ das Using-Interface und „b“ das Used-Interface von „A“. „A_out“ enthält also „a“, „A_in“ entsprechend „b“.

Bei diesem Ansatz wird jedoch davon ausgegangen, dass die untersuchte Software wenigstens auf Datei-Ebene angemessen strukturiert ist. Speziell C/C++-Fallstudien zeigten jedoch, dass diese Annahme bei real (und bereits länger) eingesetzten Systemen keineswegs zutreffen muss. Daher wurde nach einem Weg gesucht, die angestrebte Aufteilung bereits auf der Ebene der extrahierten Relationen zu realisieren.

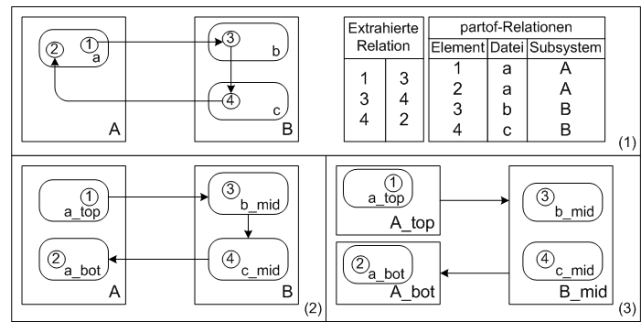


Abbildung 3: Beispiel für die Zerlegung von Dateien (2) und Subsystemen (3) ausgehend von den top- und bottom-Mengen der extrahierten Relationen (1)

Abbildung 3 veranschaulicht an einem Beispiel die gefundene Lösung. Dabei wird für jede Datei die Relation auf diejenigen Elementpaare reduziert, in denen links und/oder rechts ein Element aus der Datei anzutreffen ist. Der top-Menge werden all die Elemente aus der betrachteten Datei zugeordnet, die in der reduzierten Relation nur auf der linken Seite anzutreffen sind. Die bottom-Menge wird von den Elementen, die ausschließlich rechts vorkommen, gebildet. Nicht zugeordnete Elemente kommen in eine dritte Kategorie.

Im Anschluss können die Dateien so aufgeteilt werden, dass sie jeweils nur Elemente einer Kategorie enthalten. Diese Aufteilung kann entsprechend auf Subsystemebene fortgeführt werden.

4 Erfahrungen und Ausblick

Die untersuchten Fallstudien zeigten, dass das beschriebene Defizit des Lift-Operators praktisch relevant ist. Weiterhin zeigte sich, dass diesem Defizit mit dem vorgeschlagenen Vorgehen sogar automatisiert begegnet werden kann.

In weiteren Untersuchungen werden die extrahierten Relationen noch genauer auf ihren jeweiligen Informationsbeitrag untersucht, um mit möglichst wenigen und maximal reduzierten Relationen die für die angestrebte Darstellung (Abbildung 1) benötigten Abhängigkeiten zu erfassen.

Literatur

- [1] van Deursen, A.; Hofmeister, Ch.; Koschke, R.; Moonen, L.; Riva, C.: *Symphony: View-Driven Software Architecture Reconstruction*, Proc. of the 4th Working IEEE/IFIP Conference on Software Architecture, 2004.
- [2] Krikhaar, R. L.: *Software Architecture Reconstruction*, Dissertation an der Universiteit van Amsterdam, 1999.
- [3] Nuutila, E.; Soisalon-Soininen, E.: *On Finding the Strongly Connected Components in a Directed Graph*, Information Processing Letters, 1994.