Modulare Analyse Stochastischer Graphtransformationssysteme *

Reiko Heckel Georgios Lajios Sebastian Menge

Lehrstuhl für Software Technologie, Universität Dortmund

Abstract: Analog zu stochastischen Petrinetzen oder Prozesskalkülen wurden stochastische Graphtransformationssysteme entwickelt, um funktionale und nichtfunktionale (insbesondere zeitabhängige und probabilistische) Aspekte von Softwaresystemen integrieren und auf hohem Abstraktionsniveau beschreiben zu können. Diese Kombination eignet sich vor allem für die Modellierung und Analyse mobiler Anwendungen, wo dynamische Rekonfiguration von Netzen an der Tagesordnung ist.

Mit Hilfe von Analysewerkzeugen wie stochastischen Modelcheckern ist es möglich, Eigenschaften dieser Modelle zu überprüfen. Um diese Technik auf Systeme interessanter Größe anwenden zu können, befasst sich die vorliegende Arbeit mit der Modularisierung der stochastischen Analyse. Insbesondere wird die Korrektheit eines entsprechenden modularen Erzeugungsverfahrens gezeigt und von Experimenten berichtet, die die Überlegenheit des modularen Ansatzes belegen.

1 Einleitung

Softwaresysteme basieren in zunehmendem Maße auf global verteilten und mobilen Netzwerken, deren Zuverlässigkeit und Leistung starken Schwankungen unterworfen ist. Damit werden, neben den funktionalen Anforderungen an die zu entwickelnden Systeme, nichtfunktionale Anforderungen immer wichtiger.

Insbesondere bei zeitabhängigen Eigenschaften wie Fehlerraten oder Antwortzeiten sind dabei statistische und stochastische Aussagen interessant, da sich z.B. das Auftreten von Fehlern nicht im Einzelnen vorhersagen lässt. Zur Beschreibung und Analyse solcher Fragestellungen eignen sich mathematische Modelle wie Markovketten (stochastische Transitionssysteme) bzw. stochastische Petrinetze [BK02] oder Prozessalgebren [BH01].

Andererseits haben auch mobile Anwendungen heute eine funktionale Komplexität erreicht, die der klassischer Softwaresysteme nicht nachsteht. Um diese Anforderungen erfassen und analysieren zu können sind Modelle erforderlich, die ein hohes

*Diese Arbeit wurde gefördert durch die DFG, Förderkennzeichen DO 263/8-1 [Algebraische Eigenschaften stochastischer Relationen] und das European Community's Human Potential Programme, Förderkennzeichen HPRN-CT-2002-00275 [SegraVis].

Abstraktionsniveau bieten, intuitiv verständlich und ausdrucksstark genug sind, um das komplexe Verhalten mobiler Anwendungen angemessen beschreiben zu können.

Petrinetze sind aufgrund ihrer starren Struktur dazu nur bedingt geeignet. Prozesskalküle wie der π-Kalkül erlauben die Modellierung mobiler Strukturen, sind aber in Bezug auf ihre Beschreibungsebene eher mit Programmiersprachen vergleichbar und nur bedingt intuitiv zugänglich.

Das wahrscheinlich naheliegendste Modell für Netzwerk- und Softwarearchitekturen ist ein Graph, dessen Struktur sich im Verlauf der Systemoperation schrittweise ändert, etwa durch das Erzeugen einer neuen Netzwerkverbindung, den Ausfall eines Gerätes oder das Aktivieren einer neuen Anwendungskomponente. Im Rahmen von Graphtransformationssystemen lassen sich solche Schritte visuell durch Regeln beschreiben.

Diese Arbeit hat das Ziel, die anschauliche Beschreibung der Funktion mobiler Systeme durch Graphtransformationssysteme mit den Techniken der stochastischen Modellierung und Analyse zu verbinden und so eine integrierte Beschreibung möglich zu machen. Wir benutzen dazu den in [HLM04] vorgestellten Ansatz und beschäftigen uns nach einer kurzen Zusammenfassung grundlegender Konzepte in Abschnitt 2 mit der Zustandsraumanalyse stochastischer Graphtransformationssysteme.

Basierend auf dem Werkzeug GROOVE [Ren04a], das die beschränkte Erzeugung von Transitionssystemen aus Graphtransformationssystemen realisiert, ist es möglich, ein stochastisches Graphtransformationssystem in eine Markovkette zu überführen, die ihrerseits als Eingabe für stochastische Modelchecker geeignet ist.

Das bekannte Problem der Entstehung großer Zustandsräume lässt sich durch eine Zerlegung des Ausgangssystems in verschiedene Sichten vermindern. Aus diesen werden einzelne Transitionssysteme erzeugt, die anschließend zu einem Gesamtsystem komponiert werden.

Die vorliegende Arbeit zeigt die Korrektheit dieses modularen Ansatzes und berichtet von Experimenten die anhand eines einfachen Modells eines mobilen Netzwerks seine Überlegenheit demonstrieren.

2 Stochastische Graphtransformation

2.1 Beispiel: Mobile Systeme

Wir modellieren mobile Systeme als getypte Graphen, deren Knoten Zellen und Geräte darstellen. Das Klassendiagramm TG im linken Bereich von Abb. 1 legt fest, dass zwischen zwei Zellen eine Kante *neighbor* existieren kann, die wir als geographische Nachbarschaftsbeziehung interpretieren, und ein Gerät D sich im Bereich einer oder mehrerer Zellen befinden *(located)* und Verbindungen zu ihnen besitzen kann *(connected)*. Die Zellen besitzen ein boolesches Attribut *ok*, das angibt, ob

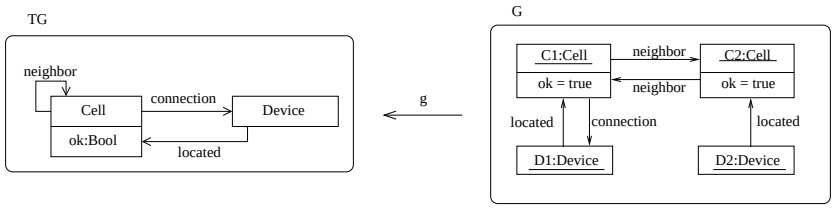

Abbildung 1: Klassendiagramm TG und beispielhaftes Objektdiagramm $< G, g >$

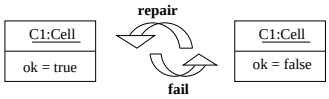

Abbildung 2: Ausfall- und Reparatur-Regeln

sie funktionstüchtig sind oder nicht. Ein getypter Graph ist ein Homomorphismus $g : G \rightarrow TG$ von einem Objektdiagramm G in das Klassendiagramm TG (letzteres nennen wir auch *Typgraph*). Abbildung 1 stellt einen getypten Graphen dar, der aus zwei Zellen und zwei Geräten besteht.

Mithilfe von Graphtransformationsregeln modellieren wir Ereignisse im mobilen System, wie Ausfall und Reparatur von Komponenten, ihre Bewegung von einer Zelle in die andere, die Herstellung und den Abbruch von Verbindungen, u.A. Wir verwenden dazu Graphtransformationsregeln nach dem Doppel-Pushout-Ansatz [Roz97, Kap. 3]. Im Folgenden notieren wir die Regeln semiformal durch graphische Darstellungen. Eine Regel $p : L \rightarrow R$ ist auf einen Graphen G anwendbar, wenn ein Morphismus $m : L \rightarrow G$ von der linken Regelseite L nach G existiert, der im allgemeinen Fall gewissen Verträglichkeitsbedingungen genügen muss [Roz97, 3.3.4], die in unserem Beispiel trivial erfüllt sind. Der Morphismus m definiert den *Ansatz* der Regelanwendung.

Ausfall und Reparatur von Zellen werden durch die Regeln in Abb. 2 ausgedrückt. Mobilität von Geräten wird durch die Regeln *moveIn, moveOut* und *move* in Abb. 3 dargestellt. Die negative Anwendungsbedingung in der Regel *move*, die durch eine durchgestrichene Kante gekennzeichnet ist, stellt sicher, dass es höchstens eine

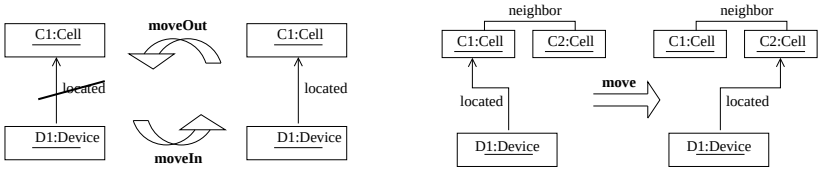

Abbildung 3: Mobilitäts-Regeln

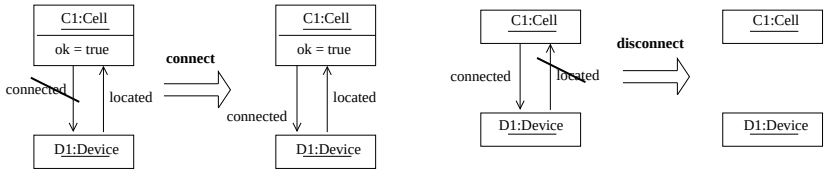

Abbildung 4: Herstellung und Abbruch einer Verbindung

Kante vom Typ *located* zwischen einer Zelle und einem Gerät geben kann. Neben den Regeln *moveIn* und *moveOut*, die das Hinein- und Herausbewegen in bzw. aus einer Zelle modellieren, stellt die Regel *move* den Übergang von einer Zelle zu einer benachbarten dar.

Weiterhin führen wir in Abbildung 4 zwei Regeln zur Herstellung und zum Abbruch von Verbindungen auf. Ein Gerät, das sich im Bereich einer Zelle befindet und keine Verbindung zu ihr besitzt, kann eine solche herstellen. Existiert dagegen eine Verbindung, obwohl das Gerät sich nicht im Bereich der Zelle befindet, so geht diese verloren.

2.2 Erzeugung des Transitionssystems

Während ein Graphtransformationssystem alle Graphstrukturen herzuleiten gestattet, zählt das *Graphtransitionssystem* alle Graphen und direkten Ableitungen explizit auf. Ein Graph wird nun ausdrücklich als Zustand angesehen und eine direkte Ableitung als Zustandsübergang. Dabei ergibt sich ein Problem: verschiedene isomorphe Graphen stellen den gleichen Zustand dar, sollen aber im Transitionssystem nur einmal vorkommen. Wir wählen daher aus jeder Isomorphieklasse $\{H \mid H \cong G\}$ einen sogenannten *Standardrepräsentanten* $S(G)$ aus [CEL$^+$92] und nehmen nur diesen in das Transitionssystem auf. Definiert man auch die Morphismen in verträglicher Weise, so erhält man eine Kategorie **SGraph**, in der es keine Isomorphismen zwischen zwei verschiedenen Objekten gibt [Men04].

Wir können nun ein Graphtransitionssystem definieren, in dem keine zueinander isomorphen Zustände vorkommen. Es kann aber durchaus verschiedene Übergänge zwischen zwei Zuständen geben, daher markieren wir jeden Zustandsübergang nicht nur mit der Regel selbst, sondern auch mit ihrem Ansatz.

Definition 1 (Graphtransitionssystem). *Ein Graphtransformationssystem* $\mathcal{G} = \langle G_0, P, \pi \rangle$ *mit einem Startgraph* G_0 *erzeugt ein* Graphtransitionssystem $LTS(\mathcal{G}) = \langle L, T, S \rangle$*:*

1. *S ist die Menge aller Standardrepräsentationen von Graphen, die von G_0 aus durch Transformationen aus P erreichbar sind.*
2. *L ist die Menge der direkten Ableitungen (p, o), wobei p eine Regel ist und*

$o : L \to G$ die Einbettung der linken Regelseite in den zu transformierenden Graphen (der Ansatz der Regel).

3. *$T \subseteq S \times L \times S$ ist die Übergangsrelation, wobei ein Tripel $\langle s_1, l, s_2 \rangle$ in T ist, wenn eine direkte Ableitung $G \overset{(p,o)}{\Longrightarrow} H$ existiert, so dass $S(G) = s_1$ und $S(H) = s_2$ gilt.*

Das Graphtransitionssystem besteht also aus allen Zuständen, unabhängig von ihrer Repräsentation als Graph, und allen Transformationen mit ihren Ansätzen als Zustandsübergängen.

2.3 Zuweisung von Raten und Erzeugung der Markovkette

In [HLM04] haben wir *stochastische Graphtransformationssysteme* eingeführt. Wir weisen jeder Regel p eine exponentialverteilte Schaltzeit zu, indem wir eine Abbildung $\rho : P \to \mathbb{R}^+$ definieren, die den Parameter $\rho(p)$ der Exponentialverteilung $e^{\rho(p) \cdot t}$ festlegt. In unserem Beispiel haben wir folgende Raten gewählt:

Regel p	Rate $\rho(p)$	Regel p	Rate $\rho(p)$	Regel p	Rate $\rho(p)$
moveIn	1	moveOut	1	move	100
repair	500	fail	1		
connect	10000	disconnect	10000		

Nimmt man *pro Tag* als Zeiteinheit, so bedeutet dies, dass beispielsweise durchschnittlich ein Ausfall pro Tag auftritt und eine Reparatur 3 Minuten dauert – dies allerdings stochastisch und exponentialverteilt, d.h. es handelt sich um Erwartungswerte; in der konkreten Ausprägung können beliebige Zeiten auftreten, die allerdings unterschiedlich wahrscheinlich sind.

Die Raten für die Regeln induzieren Übergangsraten im Transitionssystem LTS, wobei geeignete Summationen durchgeführt werden müssen, wenn für eine Regel mehrere Ansätze existieren [HLM04]. Ein zeitbehaftetes Transitionssystem, dessen Übergangszeiten exponentialverteilt sind, ist nichts anderes als eine *zeitkontinuierliche Markovkette* [Nor97]. Wir erhalten somit aus einem stochastischen Graphtransformationssystem eine *induzierte Markovkette*.

Die Inzidenzmatrix der Markovkette, die sogenannte Q-Matrix, ist eine quadratische reelle Matrix über dem Zustandsraum S des Graphtransitionssystems, deren Nichtdiagonaleinträge die Übergangsraten zwischen zwei verschiedenen Zuständen enthalten (bzw. Null, falls keine Transition existiert), während die Diagonaleinträge aus technischen Gründen so gewählt werden, dass die Zeilensummen verschwinden [Nor97]. Für endliche Zustandsräume ist dies immer gewährleistet, für unendliche sind zusätzliche Bedingungen notwendig.

Eigenschaften zeitkontinuierlicher Markovketten können mithilfe der Continuous Stochastic Logic (CSL) beschrieben werden, einer Erweiterung der Aussagenlogik um zwei Operatoren $\mathcal{S}$ und $\mathcal{P}$ [BHHK00]. Damit können Fragestellungen formuliert werden wie:

- $\mathcal{P}_{<0.01}(true\ \mathcal{U}^{[0,5]}\ err)$ – Ist die Wahrscheinlichkeit für einen Fehler innerhalb der ersten 5 Minuten kleiner als 1 % ?
- $\mathcal{S}_{>0.9}(safe)$ – Befindet sich das System auf lange Sicht mit Wahrscheinlichkeit >90 % in einem sicheren Zustand?
- $\mathcal{P}_{<0.01}(true\ \mathcal{U}^{[0,1]}\ dis.d1)$ – Ist die Wahrscheinlichkeit, dass Gerät d_1 innerhalb einer Zeiteinheit (des Intervalls [0..1]) die Verbindung abbricht, kleiner als 1 % ?

Dazu wird noch eine geeignete Definition der atomaren Propositionen *err, safe, dis* benötigt. Hierzu definieren wir die Menge AP der atomaren Propositionen als die Menge aller Regelnamen mit Parametern (Variablen) $AP = \{p(x_1, \ldots, x_n) \mid p \in P, x_i \in X\}$ und die Menge $L(s)$ der in einem Zustand wahren Propositionen als $L(s) = \{l \mid \exists t \in S\ s \stackrel{l}{\Longrightarrow} t\}$.

Die einer Regel entsprechende Proposition ist also genau dann in einem Zustand wahr, wenn die Regel bezüglich der Parameter anwendbar ist. So gilt *repair(x)* genau dann in $s \in S$, wenn Gerät x in s repariert werden kann, mithin, wenn es defekt ist. Um Eigenschaften zu definieren, die durch die Transformationsregeln nicht erfasst werden, können *Eigenschaftsregeln* eingeführt werden, Regeln mit Rate Null und identischer linker und rechter Regelseite, die nur dazu dienen, ein Muster zu definieren, das ein Zustand erfüllen muss, damit eine Proposition gilt.

Als Werkzeugkette zur Analyse stochastischer Graphtransformationssysteme verwenden wir eine Kombination des Graphtransformationstools GROOVE [Ren04a] mit dem probabilistischen Modelchecker PRISM [KNP02]. Die von GROOVE erzeugten Transitionssysteme versehen wir mit Raten für die Regeln und konvertieren sie so, dass sie das Eingabeformat von PRISM respektieren. Auf diese Weise können wir Eigenschaften stochastischer Graphtransformationssysteme in CSL spezifizieren und diese automatisiert verifizieren.

3 Modularisierung

Mit PRISM wurden in [dAKN+00] Zustandsräume der Größe 10^{30} verifiziert. Die Zustandsräume, die GROOVE erzeugen kann, liegen in Dimensionen von 10^6. Der eigentliche „Flaschenhals“ ist also die Erzeugung des Zustandsraumes eines Graphtransformationssystems. Dies ist aus verschiedenen Gründen ein schwieriges Problem:

1. Die Zustandsräume werden extrem groß. So erhält man in unserem Beispiel zur Mobilität bei weniger als 10 Knoten im Startgraphen bereits Zustandsräume in Größenordnungen von 10^6 Zuständen. Es können also schon bei sehr einfachen Transformationssystemen verhältnismäßig große Zustandsräume entstehen.

2. Es existiert kein Polynomialzeitalgorithmus, um die Isomorphie von Graphen zu testen. Durch die Beschränkung auf Standardrepräsentanten von Graphen wird der Zustandsraum zwar reduziert, aber man muss für jeden neu zu erzeugenden Zustand prüfen, ob nicht schon ein isomorpher Graph existiert. Es gibt zwar gute Heuristiken (z.B. Graphzertifikate [Ren04b]) für das Problem, aber mit der explosionsartig wachsenden Anzahl von Zuständen steigt auch die Wahrscheinlichkeit für ein *worst-case*-Beispiel.

3. Es müssen alle Ansätze gefunden werden. Auch wenn durch die Standardrepräsentanten die Anzahl der Zustände reduziert wird, müssen doch alle Ansätze gesucht werden. Die Ansatzsuche ist das Subgraphisomorphieproblem (die linke Regelseite soll isomorph zu einem Subgraph des Graphen sein), das ebenfalls NP-hart ist. Es gelten die gleichen Folgerungen wie in Punkt 2.

Wir sind bei der Laufzeitanalyse von einem *explorativen* Ansatz (z.B. Tiefen- oder Breitensuche) zur Zustandsraumerzeugung ausgegangen, wie er im Werkzeug GROOVE [Ren04b] verwendet wird.

Da also die Erzeugung des Graphtransitionssystems ein sehr aufwändiger Vorgang ist, und der Aufwand mit der Größe des Typgraphen exponentiell steigt, liegt ein *Divide and Conquer* Ansatz zur Erzeugung des Graphtransitionssystems nahe. Anstatt das Graphtransitionssystem für ein komplexes Graphtransformationssystem in einem Schritt zu erzeugen, *dekomponieren* wir es in mehrere kleinere Graphtransformationssysteme und erzeugen für diese die Graphtransitionssysteme mit geringerem Aufwand. Diese komponieren wir anschließend wieder, und induzieren aus dem Ergebnis die Markovkette. Diese können wir dann mit den in [HLM04] vorgestellten Methoden verifizieren.

3.1 Dekomposition von Graphtransformationssystemen

Eine Partition $P(TG)$ eines Typgraphen TG besteht aus einer Anzahl von Typgraphen $TG_i \subseteq TG$, so dass $\bigcup TG_i = TG$ gilt.

Abbildung 5 zeigt eine Partition des Typgraphen aus Abbildung 1. Der Teilgraph *Device* kann nur beschreiben, in welcher Zelle sich ein Gerät befindet (Kante *located*) und in welche Zellen ein Gerät sich bewegen kann (Kante *neighbor*). Der Zustand der Zelle und das Vorhandensein einer Verbindung sind hier irrelevant. Der Teilgraph *Cell* beschreibt das mögliche Verhalten einer Zelle, also deren Zustand und das Vorhandensein einer Verbindung. Geräte und Zellen mit Nachbarschaftsbeziehung kommen in beiden Teilen vor.

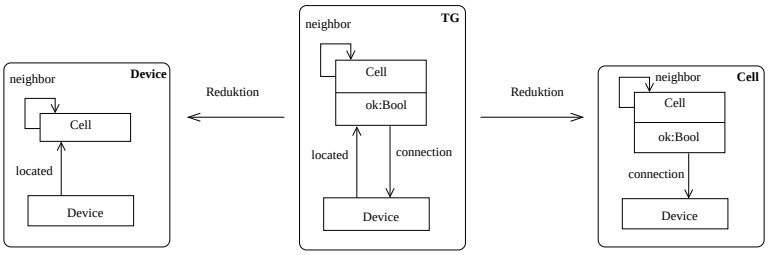

Abbildung 5: Partition des Typgraphen

Durch die Partitionierung entstehen also unterschiedliche *Sichten* auf das System: Der Typgraph *Device* beschreibt die Sicht eines Gerätes auf das System, während der Typgraph *Cell* die Sicht einer Zelle beschreibt.

Um entsprechend der Aufteilung des Typgraphen eine Verteilung von typisierten Graphen zu erzeugen, definieren wir das Redukt eines Graphen auf einen Teilgraphen seines Typgraphen. Das Redukt (oder die Einschränkung) eines TG-typisierten Graphen G mit Typisierungsmorphismus h auf $TG' \subseteq TG$, geschrieben $G|_{TG'}$ („G eingeschränkt auf TG'“), ist definiert als $G|_{TG'} = h^{-1}(TG')$. Anschaulich lassen wir alle Elemente des Graphen weg, deren Typen im kleineren Typgraphen nicht mehr vorkommen. Das Redukt eines TG-typisierten Morphismus $m : G \to H$ auf TG' ist der Morphismus $m|_{TG'} : G|_{TG'} \to H|_{TG'}$, definiert als $m|_{TG'} = m|_{G|_{TG'}}$.

Mit diesen Begriffen ist es möglich, ein Graphtransformationssystem auf einen Teil des Typgraphen einzuschränken, da sich die Einschränkung auf die Regeln, notiert als $p|_{TG'}$ und den Startgraphen $G_0|_{TG'}$ fortsetzt. Eine solche Einschränkung eines Graphtransformationssystems nennen wir auch *Sicht*, da nur die durch TG_i vorgegebenen Elemente sichtbar sind. Da die Sichten verschiedene Aspekte ausblenden, erhalten wir „kleinere“ Graphtransformationssysteme. Es ist aus der Sicht des Gerätes beispielsweise nicht wichtig, ob eine Zelle in Ordnung ist oder nicht.

Gegeben ein Graphtransformationssystem $\mathcal{G}$ und eine Partition des Typgraphen $P(TG)$, verstehen wir unter der *Dekomposition* von $\mathcal{G}$ die Menge aller Einschränkungen von $\mathcal{G}$ auf TG_i mit $TG_i \in P(TG)$. Dabei synchronisieren wir die Regelanwendungen, indem eine Regel inklusive Ansatz in allen Sichten gleichzeitig angewendet wird. Wir haben damit das Verhalten von $\mathcal{G}$ auf viele kleine Graphtransformationssysteme *verteilt*, und verlieren dabei keine Information, wie wir zeigen werden.

Die Verteilung der Regeln für die Partition des Typgraphen gemäß Abbildung 5 wird in Abbildung 6 dargestellt. Die Reduktion der Regeln *fail* und *repair* hat in der Sicht *Device* keinen Effekt, aber sie erfordert das Vorhandensein wenigstens einer Station. Die Reduktion der Regel zur Mobilität (*move*,*moveIn*,*moveOut*) hat in der Sicht *Cell* keinen Effekt, erfordert aber das Vorhandensein von zwei benachbarten Zellen und einem Gerät. Der Effekt der Regeln ist also nur lokal in einer Sicht, die

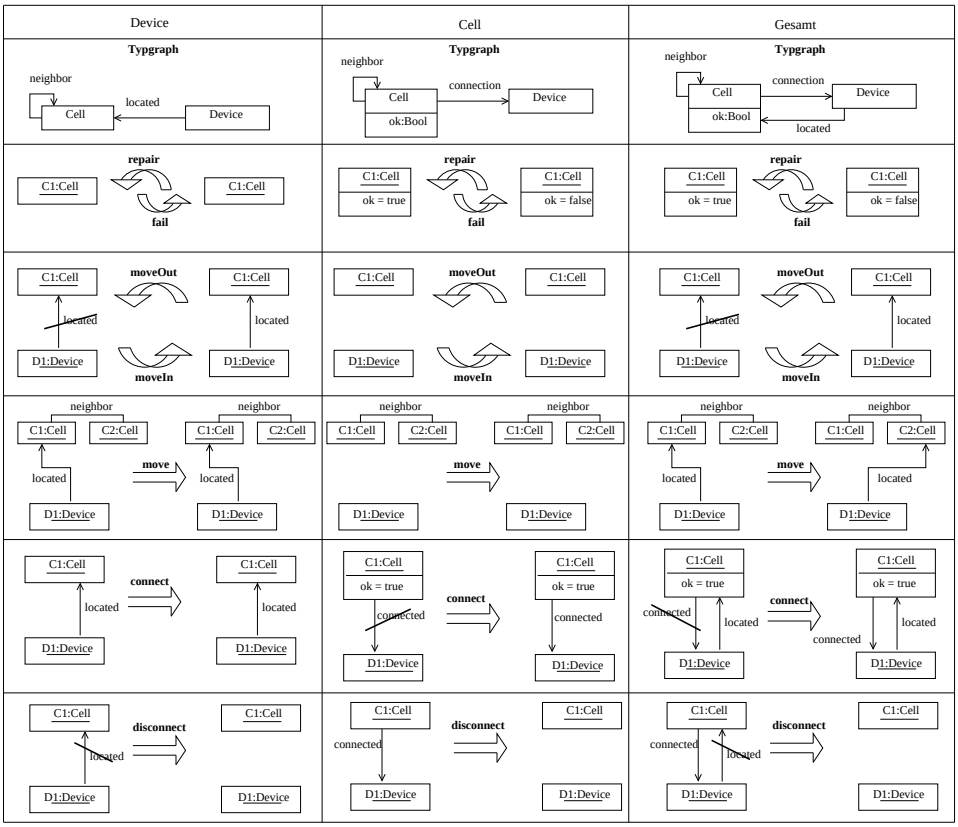

Abbildung 6: Verteilung der Regeln

Anwendbarkeit dagegen wird auf beide Sichten verteilt.

Im Allgemeinen muss das nicht so sein; man kann die Partition des Typgraphen durchaus so wählen, dass auch der Effekt einer Regel auf unterschiedliche Sichten verteilt wird. Dies bleibt dem Modellierer überlassen. Wichtig ist bei der Wahl der Partition aber, dass man in möglichst vielen Sichten Regeln erhält, die keine Zustandsänderung bewirken, da damit der Zustandsraum reduziert wird.

3.2 Kreuzprodukt von Graphtransitionssystemen

Wie in Abschnitt 2.2 gesehen, können wir aus einem Graphtransformationssystem $\mathcal{G}$ das Graphtransitionssystem $LTS(\mathcal{G})$ ableiten. Wenn wir zu einer Partition $P(TG)$ für alle $TG_i \in P(TG)$ die Graphtransitionssysteme $LTS(\mathcal{G}|_{TG_i})$ bilden und diese wiederum geeignet komponieren, entsteht ein Graphtransitionssystem,

das verhaltensäquivalent zu $LTS(\mathcal{G})$ ist.

Zur Komposition von Graphtransitionssystemen verwenden wir eine einfache Kreuzproduktkonstruktion, wie sie aus der Automatentheorie bekannt ist. Ein Zustand des Produktsystems ist also ein Paar von Zuständen der Teilsysteme und eine Transition ist im Produktsystem nur möglich, wenn die Transitionen in den Teilsystemen zueinander *kompatibel* sind.

Wir sagen, dass zwei direkte Ableitungen $G_1 \overset{p_1(o_1)}{\Longrightarrow} H_1$ und $G_2 \overset{p_2(o_2)}{\Longrightarrow} H_2$ kompatibel sind, wenn die folgenden Bedingungen gelten:

1. Aus den Teilgraphen G_1 und G_2 lässt sich ein Gesamtzustand konstruieren.
2. Aus den Teilregeln p_1 und p_2 lässt sich eine Gesamtregel erzeugen.
3. Beide Teilregeln arbeiten auf dem gleichen Ansatz.

Die Bedingungen beschreiben anschaulich die Verträglichkeit von zwei lokalen Aktionen in einem verteilten Prozess. Dadurch wird sichergestellt, dass sich die lokalen Schritte zu einem globalen Schritt $G_a \cup G_2 \overset{p_1 \cup p_2(o_1 \cup o_2)}{\Longrightarrow} H_1 \cup H_2$ zusammenfassen lassen. Formal ist diese Aussage Gegenstand des Verteilungstheorems, vgl. [Roz97], Abschnitt 3.6.

Die Komposition von Graphtransitionssystemen entspricht dem Produkt von Automaten, wobei nur verträgliche Transitionen synchronisiert werden.

Definition 2 (Produkt). *Seien $GS_1 = \langle L_1, T_1, S_1 \rangle$ und $GS_2 = \langle L_2, T_2, S_2 \rangle$ zwei Graphtransitionssysteme. Das* Kreuzprodukt *$GS = \langle L, T, S \rangle = GS_1 \otimes GS_2$ ist folgendermaßen definiert:*

- $S = \{\langle s_1, s_2 \rangle \,|s_1 \in S_1,\ s_2 \in S_2,$ *und* $s_1 \cup s_2$ *wohldefiniert*$\}$
- $L = \{d_1 \cup d_2 \,|d_1 \in L_1,\ d_2 \in L_2,$ *und* d_1 *kompatibel zu* $d_2\}$
- $\langle s_1, s_2 \rangle \overset{d_1 \cup d_2}{\longrightarrow} \langle s_1', s_2' \rangle \in T$ *genau dann, wenn* $s_1 \overset{d_1}{\longrightarrow} s_1' \in T_1$, $s_2 \overset{d_2}{\longrightarrow} s_2' \in T_2$ *und* d_1 *kompatibel zu* d_2.

Um zu zeigen, dass sich die Produktbildung mit der Dekomposition verträgt, haben wir in [Men04] gezeigt, dass die beiden Systeme *bisimilar* sind.

Satz 3 (Verträglichkeit). *Gegeben sei ein Graphtransformationssystem G mit Typgraph TG und eine Partition $P(TG) = \{TG_1, TG_2\}$. Dann gilt:*

$$LTS(G) \sim LTS(G|_{TG_1}) \otimes LTS(G|_{TG_2})$$

Mithilfe des Verträglichkeitstheorems können wir also mit geringerem Berechnungsaufwand große Graphtransitionssysteme erzeugen und aus diesen mit den bekannten Mitteln Markovketten ableiten.

Um die Notwendigkeit eines modularen Ansatzes zur Spezifikation zu untermauern, haben wir das vorliegende Beispiel mithilfe von GROOVE modelliert und die Transitionssysteme für die Sichten und das Gesamtsystem erzeugt. Die Ergebnisse zeigen, dass schon für sehr kleine Beispiele auf Dekomposition nicht verzichtet werden kann. Die Größe der Zustandsräume hängt natürlich stark von der Anzahl der vorhanden Geräte und Zellen ab. Wir haben daher Startgraphen mit verschiedenen Konfigurationen entworfen, und ausgehend von diesen die Zustandsräume erzeugt. In diesen Startgraphen befinden sich die Geräte in keiner Zelle, sind nicht verbunden und alle Zellen sind untereinander benachbart.

Zellen / Geräte	Zustände Device	Zustände Cell	Zustände Gesamt
2 / 2	51	76	636
2 / 3	99	430	3738
2 / 4	171	1996	17616
3 / 2	219	430	10090
3 / 3	8241	672	211361
3 / 4	131506	1755	abgebrochen bei 10^7

Diese Experimente wurden auf einem Linux-PC mit 1,5 GHz und 512 MB RAM ausgeführt, und haben bei Zustandsräumen der Ordnung 10^5 länger als 30 Minuten gebraucht. Zustandsräume der Ordnung 10^7 ließen sich aus Speicherplatzmangel nicht erzeugen. Für weitere Details zur Zustandsraumerzeugung mit GROOVE verweisen wir auf [RSV04, Ren04b].

Sind die einzelnen Module erzeugt, stellt es für einen probabilistischen Modelchecker kein Problem dar das Produkt zu bilden. Dies liegt vor allem daran, dass zur Komposition der Systeme die innere Struktur der Zustände (Graphen) nicht mehr benötigt wird. Außerdem stehen sehr effiziente implizite Darstellungen (MTBDDs) zur Verfügung [dAKN$^+$00], mit denen die Produktbildung effizient durchführbar ist. Die Modularisierung (oder Sichtenbildung) dient also vor allem dazu, den „Flaschenhals“ der Erzeugung des Transitionssystems etwas zu weiten.

4 Ausblick

Die stochastische Analyse von Graphtransformationssystemen, die in dieser Arbeit modularisiert wurde, um das Entstehen zu großer Zustandsräume zu verhindern, ist erst seit kurzem Gegenstand der Forschung.

Zukünftige Arbeiten sollen sich damit befassen, die Integration der verwendeten Werkzeuge zu verbessern, etwa durch deren Einbindung in ECLIPSE, sowie weitere Fallstudien und Anwendungen zu untersuchen. Insbesondere sind Untersuchungen zur Modellierung und Analyse von mobilen Ad-Hoc- und Peer-to-Peer-Netzwerken geplant.

Literatur

[BH01] Ed Brinksma und Holger Hermanns. Process Algebra and Markov Chains. In J.-P. Katoen E. Brinksma, H. Hermanns, Hrsg., *FMPA 2000*, number 2090 in LNCS, Seiten 183–231. Springer, 2001.

[BHHK00] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns und Joost-Pieter Katoen. Model Checking Continuous-Time Markov Chains by Transient Analysis. In *Computer Aided Verification*, Seiten 358–372, 2000.

[BK02] Falko Bause und Pieter S. Kritzinger. *Stochastic Petri Nets*. Vieweg Verlag, 2nd. Auflage, 2002.

[CEL+92] A. Corradini, H. Ehrig, M. Löwe, U. Montanari und F. Rossi. Note on Standard Representation of Graphs and Graph Derivations. Bericht 92-25, FB13, 1992.

[dAKN+00] L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker und R. Segala. Symbolic Model Checking of Concurrent Probabilistic Processes Using MTBDDs and the Kronecker Representation. In S. Graf und M. Schwartzbach, Hrsg., *Proc. 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, Jgg. 1785 of *LNCS*, Seiten 395–410. Springer, 2000.

[HLM04] Reiko Heckel, Georgios Lajios und Sebastian Menge. Stochastic Graph Transformation. In *Proceedings of the 2nd International Conference on Graphtransformation, ICGT '04*, Jgg. 3256 of *LNCS*. Springer, 2004.

[KNP02] M. Kwiatkowska, G. Norman und D. Parker. PRISM: Probabilistic Symbolic Model Checker. In T. Field, P. Harrison, J. Bradley und U. Harder, Hrsg., *Proc. 12th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'02)*, Jgg. 2324 of *LNCS*, Seiten 200–204. Springer, 2002.

[Men04] Sebastian Menge. Stochastische Analyse von Graphtransformationssystemen. Diplomarbeit, 2004. Fachbereich Informatik, Universität Dortmund.

[Nor97] J. R. Norris. *Markov Chains*. Cambridge University Press, United Kingdom, 1997.

[Ren04a] Arend Rensink. GRaphs for Object-Oriented VErification (GROOVE), 2004. `http://groove.sourceforge.net`, letzter Besuch: 19.10.2004.

[Ren04b] Arend Rensink. Time and Space Issues in the Generation of Graph Transition Systems. In *International Workshop on Graph-Based Tools (GraBaTs)*, Electronic Notes in Theoretical Computer Science. Elsevier Science Publishers, 2004. To appear.

[Roz97] G. Rozenberg, Hrsg. *Handbook on Graph Grammars: Foundations*, Jgg. 1. World Scientific, Singapore, 1997.

[RSV04] Arend Rensink, Akos Schmidt und Daniel Varro. Model Checking Graph Transformations: A Comparison of Two Approaches. In *Proceedings of the 2nd International Conference on Graphtransformation, ICGT '04*, Jgg. 3256 of *LNCS*. Springer, 2004.