Eine Erweiterung der Software *Snoopy* zur Verarbeitung und Verwaltung von Petrinetz-Animationen.

BACHELOR THESIS

Bachelorarbeit zur Erlangung des Grades

Bachelor of Science

am Lehrstuhl für Datenstrukturen und
Softwarezuverlässigkeit an der
Brandenburgischen Technischen Universität Cottbus

Vorgelegt von
Krispin Schulz
Studiengang
Informations- und
Medientechnik
geboren am 9. Oktober 1980
in Nauen

Erstgutachter
Frau Prof. Dr.-Ing.
Monika Heiner

Zweitgutachter
Herr Dipl. Ing. (FH)
Ronny Richter

Eingereicht am
6. Juni 2008

Eidesstattliche Erklärung

Die vorliegende Bachelorarbeit wurde von mir, Krispin Schulz, selbständig angefertigt. Die verwendeten Hilsmittel sind im Quellenverzeichnis vollständig angegeben.

Cottbus, 6. Juni 2008

Inhalt

Inhalt vii

1 Einleitung 1

2 Grundlagen 3
2.1 Graphen 3
2.1.1 Zusammenhang 4
2.1.2 Gerichtete Graphen 5
2.2 Petrinetze 7
2.2.3 Erreichbarkeitsgraphen 8
2.2.4 Gegenbeispiele und Zeugen 9
2.3 Snoopy 10

3 Entwurf 11
3.1 Aufnehmen 11
3.1.1 GUI 17
3.1.2 Anwendungsfall *Clip aufnehmen* 18
3.2 Verwalten 18
3.2.1 GUI 19
3.2.2 Anwendungsfall *Clip verarbeiten* 20
3.3 Speichern und Laden 21
3.4 Importieren 23
3.5 Exportieren 26
3.5.1 Shockwave Flash 27
3.5.2 Java Applet 28
3.5.3 Scalable Vector Graphics 29
3.5.4 Fazit 31

4 Implementierung 33
4.1 Prototyp 34
4.2 Steuerung 38
4.2.1 Skalieren 39
4.2.2 Verschieben 40
4.2.3 GUI 41
4.3 Hierarchie 42
4.3.1 GUI 42
4.4 Animation 43
4.4.1 Optionen 45
4.4.3 GUI 46

5 Zusammenfassung 47
5.1 Ausblick 47

Inhalt

Anhang **A51**
Anhang A – spclip.xsd A51
Anhang B – spped2svg.xsl A53
Anhang C – liveness_1.ltl A64
Anhang D – dekker.cfa A65

Quellen ..... **lxvii**
Literaturverzeichnis lxvii
Abbildungsverzeichnis lxxi
Tabellenverzeichnis lxxv

Einleitung

Kapitel 1

Jeden Tag benutzen Millionen Menschen öffentliche Verkehrsmittel um möglichst schnell, sicher und kostengünstig zu ihrer Arbeit zu gelangen. Damit sie pünktlich dort ankommen ist ein kompliziertes Netz aus Verkehrslinien notwendig. Auf der Arbeit kommunizieren sie dann mit Hilfe des Internets mit Kollegen, Geschäftspartnern und ihrer Familie, wobei das Internet ein heterogenes Netzwerk darstellt, dass die Menschen auf vielfältige Art miteinander verbindet. Endlich beginnt die Mittagspause und der Körper beginnt das Essen und die Getränke durch komplizierte, parallel ablaufende chemische Prozesse in andere Stoffe und Energie umzuwandeln.

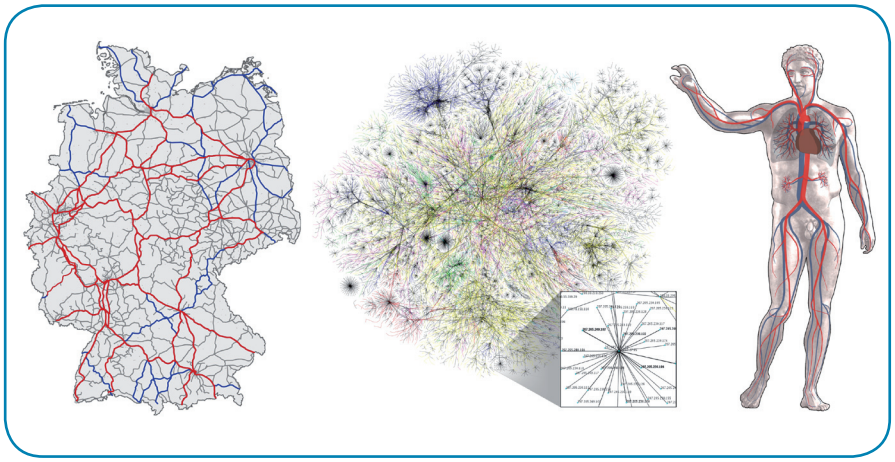


Abb. 1
Beispiele für verschiedene Netzwerke (v.l.n.r.): Eisenbahnnetz in Deutschland, Routen durch Teile des Internets, Blutkreislauf des Menschen

Unsere heutige Welt besteht aus einer Vielzahl von komplexen, nebenläufigen Systemen, die aus unserem Leben nicht mehr wegzudenken sind. Häufig treten diese Systeme in Form von Netzwerken auf. Sogar der Mensch selbst besteht aus solchen Netzwerken, dessen Vorhandensein wir kaum noch bewußt wahrnehmen.

Die Modellierung von Systemen und deren Analyse durch die Netztheorie verlangt stets nach innovativen Software-Lösungen. Viele formale Werkzeuge, die bei der Beschreibung von nebenläufigen Systemen helfen können, wurden bereits entwickelt und erfolgreich eingesetzt. Dabei eignet sich vor allem der Formalismus der Petrinetze um verschiedenste, parallel ablaufende Prozesse zu beschreiben.

Die Software *Snoopy* gestattet es dem Anwender diese Petrinetze zu modellieren und anschließend zu animieren. Leider kann

man die Animationen nicht permanent sichern um sie zum Beispiel für Präsentationszwecke abermals abzuspielen und gegebenenfalls nachträglich zu bearbeiten.

Ziel dieser Arbeit ist die Realisierung der Aufnahme und Verwaltung von Animationen in einem Petrinetz. Dieser auch als *Markenspiel* bekannte Animationsablauf wird im weiteren Verlauf als Clip bezeichnet. Dabei sollen mehrere Clips pro Petrinetz verarbeitet und vom Anwender manuell erzeugt werden können. Um Clips auch ohne *Snoopy* abzuspielen, soll zusätzlich eine Lösung gefunden werden, mit der die Aufnahme auf einer Webseite dargestellt werden kann. Die Struktur des Petrinetzes bleibt dabei unverändert, doch es soll die Möglichkeit bestehen, einzelne Parameter, wie Anfangsmarkierungen, zu ändern. Diese webbasierte Realisierung kann zum Beispiel als Anschauungsmaterial für Lehrveranstaltungen genutzt werden. Weiterhin sollen Gegenbeispiele aus externen Analysewerkzeugen importiert und anschließend animiert werden. Schließlich sollen alle an einer Animation beteiligten Netzstrukturen – also Kanten, Transitionen und Plätze – eingefärbt werden können.

Grundlagen

Kapitel 2

Um die nachfolgenden Kapitel besser zu verstehen, werden im Folgenden einige Begriffe der Graphentheorie [DöPe88] und der Petrinetze [Abel90] [Star90] eingeführt. Weiterhin wird die Software *Snoopy* kurz vorgestellt.

2.1 Graphen

Oft wird man mit bestimmten Objekten, Orten oder Personen konfrontiert, die in Beziehung zueinander stehen:
Einige Computer kommunizieren miteinander, zwei Städte sind mit einer Straße verbunden, mehrere Menschen sind über bestimmte Grade verwandt usw. Durch die Angabe der Objekte, als Elemente einer Menge, und derjenigen ungeordneten Paare, die in Verbindung stehen, definiert sich eine netzartige Struktur.

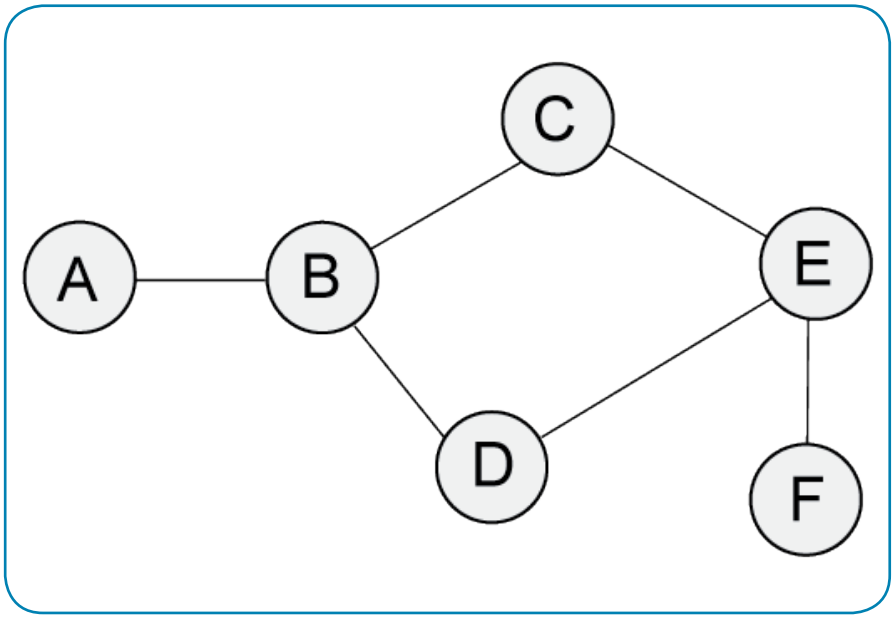


Abb. 2
Visualisierung eines einfachen ungerichteten Graphen mit sechs Knoten und sechs unbenannten Kanten

Diese Struktur kann allgemein und abstrakt zum mathematischen Begriff eines einfachen ungerichteten Graphen zusammengefasst werden:

Ein ungerichteter Graph X besteht aus einer Menge V(X), den Knoten von X, und einer Menge E(X) von ungeordneten Paaren e = [x, y] verschiedener Elemente aus V, den Kanten von X.
Man setzt X = (V, E).

Eine Kante $[x, y]$ verbindet demzufolge ihre Endpunkte x und y. Sind also x, y durch eine Kante verbunden, so heißen x, y *adjazent* und y ist ein Nachbar von x. Da es sich um ungerichtete Kanten handelt ist diese Abbildung symmetrisch. Das heißt, x ist auch ein Nachbar von y, also $[x, y] = [y, x]$. Ein Knoten x und eine Kante $e = [x, y]$ heißen *inzident*. Zwei Kanten werden als *adjazent* bezeichnet, wenn sie einen gemeinsamen Endpunkt besitzen.

Disjunkte Mengen werden auch als elementfremde Mengen bezeichnet. Sie besitzen kein gemeinsames Element, d.h. ihre Schnittmenge ist leer.

Ein Graph X besteht aus zwei disjunkten Mengen V und E und einer Abbildung f von E in die Menge der ungeordneten Paare von Elementen aus V.

Somit wird der allgemeine Fall des Graphen mit Schlingen, d.h. e mit $f(e) = [x, x]$, und Mehrfachkanten, d.h. e_1, e_2 mit $f(e_1) = f(e_2)$, in mathematisch exakter Weise erfasst.

2.1.1 Zusammenhang

Da Graphen in erster Linie die möglichen Verbindungen zwischen den durch die Knoten repräsentierten Objekten darstellen, helfen die folgenden Begriffsdefinitionen dabei den Zusammenhang eines Graphen zu veranschaulichen.

Eine Kantenfolge im Graphen X vom Knoten x zum Knoten y ist eine endliche Folge von Knoten $x = x_1, x_2, x_3, \ldots, x_n = y$, *wobei* $[x_i, x_i + 1] \in E$ *ist für* $i = 1, 2, 3, \ldots, n - 1$. *Die Kantenfolge verbindet x mit y und heißt offen, wenn* $x \neq y$ *ist, und geschlossen, wenn* $x = y$ *ist.*

Bei einer offenen Kantenfolge, in der alle Knoten $x_1, \ldots, x_n$ verschieden sind, hat sich der Begriff „Pfad“ geprägt. Sind alle Kanten einer Kantenfolge verschieden, so heißt sie ein Kantenzug oder auch einfacher Weg. Eine geschlossene Kantenfolge, in der bis auf den ersten und letzten Knoten alle Knoten verschieden sind wird als Kreis bezeichnet.

Die Anzahl der Kanten, die in einer Kantenfolge durchlaufen werden, heißt die Länge der Kantenfolge. Dadurch wird insbesondere auch die Länge eines Weges erklärt. Die kleinste Länge eines Weges von x nach y definiert sich durch den Abstand $d(x, y)$. Sollte es keine Wege zwischen x und y geben, dann wird $d(x, y) = \infty$ gesetzt.

Mit Hilfe dieser Begriffe lässt sich nun eine mathematische Beschreibung für den anschaulichen Begriff „zusammenhängender Graph“ finden:
Ein Graph X heißt zusammenhängend, wenn je zwei seiner Knoten durch einen Weg verbunden ist. Es ist dann d(x, y) endlich für alle $x, y \in V$.

Dabei wird ein Graph mit $|V(X)| = 1$, also ein einzelner isolierter Knoten, ebenfalls als zusammenhängend betrachtet.

2.1.2 Gerichtete Graphen

In vielen Fällen muss allerdings die Orientierung von Kanten berücksichtigt werden, d.h. die Kante hat einen Anfangspunkt und einen Endpunkt und die Paare von Knoten müssen als geordnet definiert werden:

Ein gerichteter Graph X besteht aus einer endlichen Menge V von Knoten und einer Menge von geordneten Paaren (x, y) verschiedener Knoten x, y aus V; also ist $E \subseteq V \times V$. *Die Elemente aus E sind die gerichteten Kanten von X.*

Ist $e = (x, y) \in E(X)$, so heißt x der Anfangspunkt und y der Endpunkt von e. Die Orientierung einer Kante (x, y) wird in der Zeichnung eines Graphen mit einem Pfeil von x nach y angedeutet.

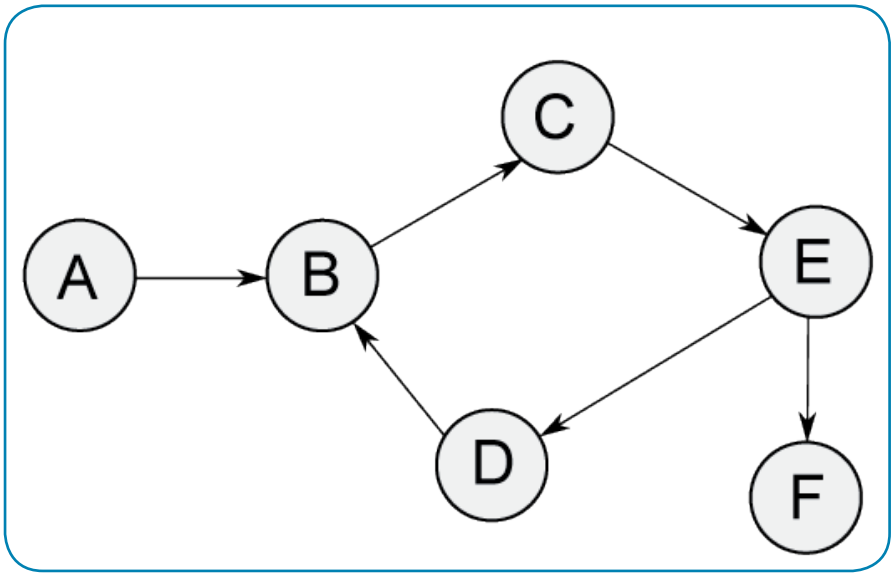


Abb. 3
Visualisierung eines gerichteten Graphen mit sechs Knoten und sechs Kanten mit Anfangs- und Endpunkt

Die Anzahl der Kanten, für die der Knoten x Anfangspunkt ist, heißt der *Außengrad* $d^+(x)$ und entsprechend ist der *Innengrad* $d^-(x)$ die Anzahl der Kanten mit Endpunkt x.
Der Grad $d(x)$ ist dann $d^+(x) + d^-(x)$.

2.2 Petrinetze

Ein Petrinetz kann als bipartiter und gerichteter Graph aufgefasst werden. Er besteht aus zwei Typen von Knoten:

- Plätze
- Transitionen

Bipartiter Graph
ist ein Graph, in dem zwei Partitionen von Knoten existieren, innerhalb derer keine Knoten, die zur selben Partition gehören miteinander verbunden sind.

Die Plätze werden meistens als Kreise und die Transitionen als Rechtecke symbolisiert. Die Plätze können eine Markierung erhalten, die oft als Punkte oder Zahlen innerhalb der Plätze dargestellt werden. Diese Marken werden auch Token genannt und beschreiben den Systemzustand.

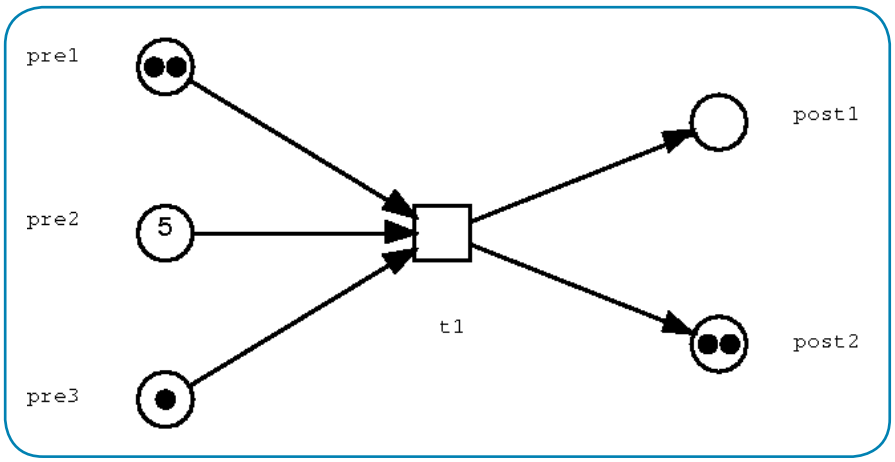


Abb. 4
Transition mit Vor- und Nachbedingungen

Ein Platz beschreibt eine Systembedingung und eine Transition stellt ein Ereignis in diesem System dar. Das Eintreten eines Systemereignises wird durch das Schalten oder auch Feuern der jeweiligen Transition angezeigt. Damit es zum Schalten einer Transition *t* kommen kann, müssen deren Vorbedingungen erfüllt sein. Als Vorbedingung wird ein Platz bezeichnet, der über eine Kante verfügt, die direkt zu der Transition *t* führt. Die Vorbedingung ist erfüllt, wenn sie mindestens einen Token besitzt. Da es in einem Petrinetz auch Kanten mit einer Gewichtung höher 1 geben kann, muss in diesem Falle der Platz mindestens mit dem Wert der Kantengewichtung markiert sein. Äquivalent zur Vorbedingung gibt es auch eine Nachbedingung. Wenn eine Kante von einer Transion zu einem Platz führt, so ist der Platz eine Nachbedingung der Transition. Das Schalten einer

Transition verändert die Markierung des Platzes und somit die Erfüllung der Nachbedingung.

Sollte es keine Transitionen geben, dessen Vorbedingungen erfüllt sind, befindet sich das System in einem toten Zustand, da keine Transition mehr feuern kann.

2.2.3 Erreichbarkeitsgraphen

Alle Markierungen eines Netzes, die von einer gegebenen Anfangsmarkierung aus erreicht werden können, bilden die Erreichbarkeitsmenge des Netzes. Diese Menge kann durch einen gerichteten Graphen, den Erreichbarkeitsgraphen, dargestellt werden.

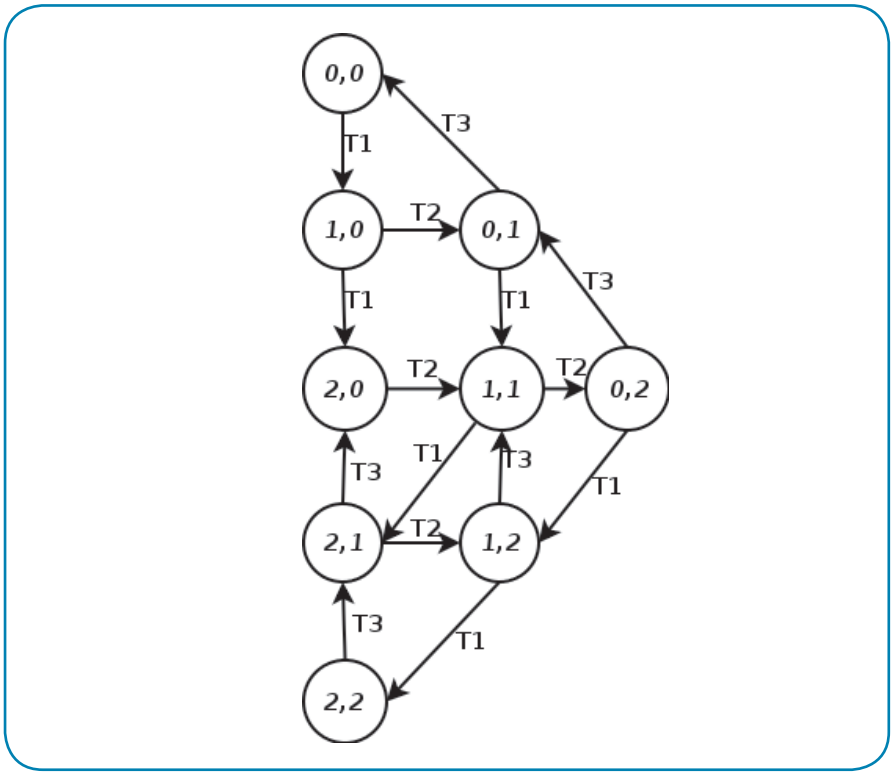


Abb. 5
Erreichbarkeitsgraph eines Petrinetzes mit 9 erreichbaren System-Zuständen

Vorraussetzung für dessen Konstruktion ist, dass die Erreichbarkeitsmenge endlich ist.

2.2.4 Gegenbeispiele und Zeugen

Ein Gegenbeispiel ist ein Pfad im Erreichbarkeitsgraphen eines Petrinetzes. Dieser Pfad widerlegt eine bestimmte System-Eigenschaft.

Ein Zeuge wiederum ist auch ein Pfad im Erreichbarkeitsgraphen, der im Unterschied zum Gegenbeispiel eine System-Eigenschaft bestätigt.

Es gibt verschiedene Analyse-Werkzeuge mit denen die Eigenschaften eines Petrinetzes überprüft werden können. Einige Analyse-Werkzeuge und ihre Ausgaben werden im Kapitelabschnitt 3.4 näher beschrieben.

2.3 Snoopy

Die Software *Snoopy* ist ein Werkzeug für das Entwerfen und Animieren/Simulieren von hierarchischen graph-basierten System-Beschreibungen [Heiner08]. Als Studienprojekt im Jahre 1997 begonnen, entwickelte sich *Snoopy* zu einem komplexen Programm das weltweit Anwendung findet.

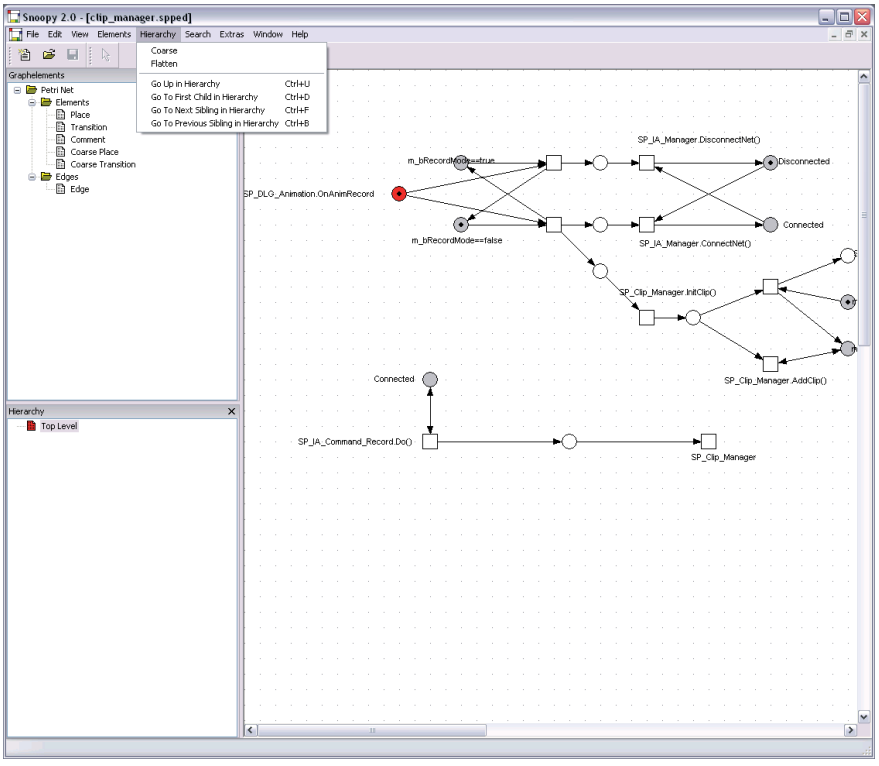


Abb. 6
Das Werkzeug Snoopy in Anwendung auf dem Windows-Betriebssystem

Neben der Erstellung verschiedener Graphen, wie zum Beispiel Erreichbarkeitsgraphen (engl. *reachability graph*) und Fehlerbäumen (engl. *fault tree*), können auch Petrinetze entworfen und animiert werden. Die Petrinetze werden als XML-Datei mit der Endung `spped` abgespeichert.

Entwurf

Kapitel 3

Wie bereits in der Einleitung erwähnt, kann der Anwender von *Snoopy* Petrinetze animieren und einige Parameter vor Animationsbeginn einstellen. Diese Einstellungen gehen aber beim Beenden des Programms verloren. Um nun einen Animationsablauf permanent aufzunehmen, gilt es zu erst zu überlegen, welche Daten des Netzes für die Aufnahme relevant sind. Für die Verarbeitung und Verwaltung der Daten werden geeignete Entwurfsmuster vorgestellt, die anschließend in konkrete Klassendiagramme umgesetzt werden.

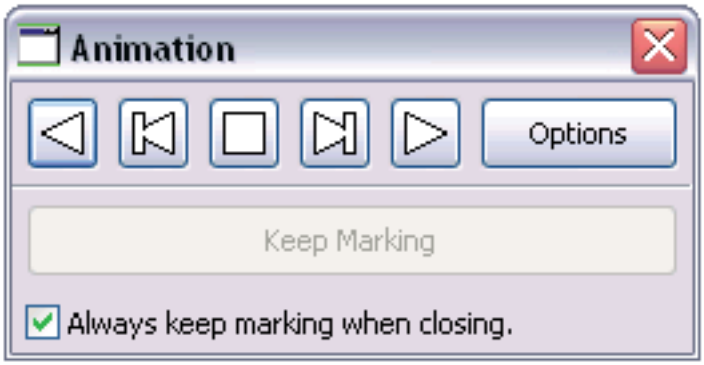
Das Konzept für das Speichern und Laden von Petrinetz-Animationen in *Snoopy* wird ebenso erläutert, wie die Entwicklung eines Konzepts für den Export der Animationen in ein geeignetes Webformat. Auch die Gestaltung und Logik der grafischen Benutzeroberfläche (engl. *Graphical User Interface - GUI*) gehört zu den Überlegungen, die in diesem Kapitel dargelegt werden. Im Sinne einer möglichst hohen Benutzerfreundlichkeit (engl. *usability*) sind in den einzelnen Kapitelabschnitten Anwendungsfälle vorgegeben.

3.1 Aufnehmen

Vor dem Entwurf für das Aufnehmen von Petrinetz-Animationen, wird zuerst analysiert, wie der Anwender von *Snoopy* Petrinetze animieren kann und welche Elemente des GUI ihm dafür zur Verfügung gestellt werden. Darauf aufbauend wird ein Konzept für die Aufnahme und ihre Steuerung entwickelt, das anhand eines möglichen Anwendungsfalles getestet und validiert wird.

Möchte der Anwender von *Snoopy* ein Petrinetz animieren, ruft er das Animationsfenster aus dem Menüpunkt View -> Start Anim Mode auf. Alternativ dazu kann er auch die F5-Taste drücken. Das Programm befindet sich danach im sogenannten Animationsmodus.

Abb. 7
Animationsfenster in *Snoopy*



Wie in Abbildung 2.1 dargestellt besteht das Animationsfenster aus sieben Buttons und einer Checkbox. Die ersten fünf Buttons, von oben links gezählt, werden für die Steuerung der Animation benötigt. Mit dem ersten Button von links wird die Animation rückwärts abgespielt (*Play backward*). Durch erneutes Drücken beim Abspielen wird die Animation unterbrochen (*Pause*). Über den nächsten Button wird das Petrinetz um einen Schritt rückwärts animiert (*Step backward*). Der dritte Button setzt die Markierungen auf den Plätzen wieder auf die ursprüngliche Anzahl, mit der die Animation gestartet wurde (*Reset Marking*). Beim vierten Button wird die Animation um einen Schritt vorwärts abgespielt (*Step forward*). Der Button an fünfter Stelle spielt die Animation vorwärts ab (*Play forward*). Beim erneuten Drücken wird die Animation des Petrinetzes unterbrochen (*Pause*).

Vor Beginn einer Animation kann der Anwender unter dem Button mit der Aufschrift *Options* im Animationsfenster folgende Eigenschaften einstellen:

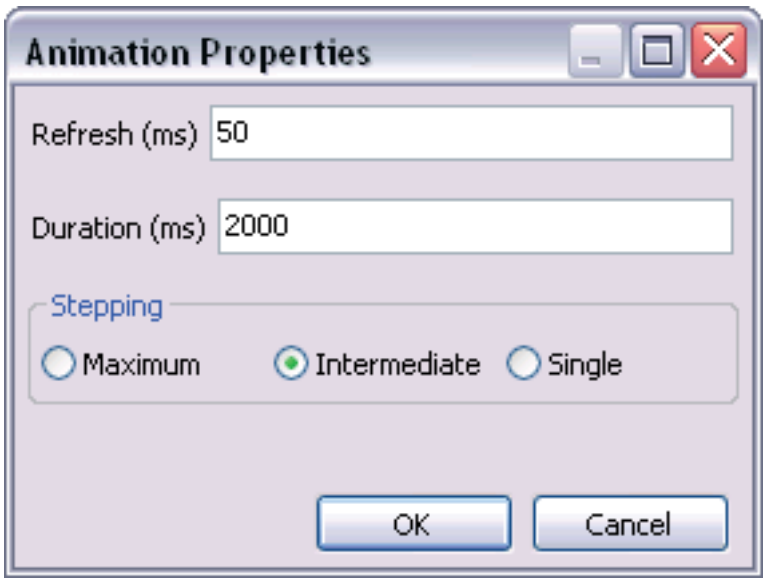


Abb. 8
Fenster-Dialog für das Ändern der Eigenschaften einer Animation in *Snoopy*

Refresh bedeutet, wie häufig die Anzeige aktualisiert werden soll. *Duration* gibt die Dauer eines Animationsschrittes (engl. *step*) in Millisekunden an. Im Bereich *Stepping* wird die Schaltregel festgelegt:

- *Maximum* Es feuert eine maximale Menge von konfliktfreien Transitionen in einem Schritt.
- *Intermediate* Es feuert eine zufällig ermittelte Menge von schaltfähigen Transitionen.
- *Single* Es feuert nur eine schaltfähige Transition in einem Schritt.

Der letzte Button (*Keep Marking*) dient dem Anwender um festzulegen, ob er die Markierungen auf den Plätzen, die sich im Laufe des Markenspiels ändern können, beim Beenden des Animationsmodus behalten möchte. Sobald das Animationsfenster geschlossen wird, ist der alte Zustand des Petrinetzes vor dem Beginn der Animation wieder hergestellt. Dieses Verhalten läßt sich aber ändern, indem die Checkbox (*Always keep*

marking when closing) aktiviert wird. Die Plätze des Petrinetzes behalten dabei automatisch ihre aktuelle Markierung, wenn das Animationsfenster vom Anwender geschlossen wird.

Die folgende Analyse und die daraus resultierende Definition helfen dabei, die Daten zu identifizieren, die für die Aufnahme eines Clips relevant sind.

Der Animationsablauf eines Petrinetzes hat einen zeitlichen Startpunkt T_0 und einen zeitlichen Endpunkt T_n. Der Endpunkt wird automatisch vom Programm oder manuell vom Anwender gesetzt. Die Dauer der Zeitspanne zwischen T_0 und T_n ist abhängig von der Anfangsmarkierung und der Struktur des Netzes. Sie kann sehr lang, sehr kurz oder gleich Null sein. Im Falle, dass die Zeitspanne gleich Null ist befindet sich das Petrinetz zum Zeitpunkt T_0 bereits in einem toten Zustand (engl. *dead state*). In *Snoopy* wird die Animation aus der für Petrinetze üblichen Schaltregel generiert. Das heißt eine Transition schaltet nur, wenn ihre Vorbedingungen erfüllt sind.

Für einen Clip benötigt man die Anfangsmarkierung (engl. *initial marking*) aller Plätze zum Zeitpunkt T_0 und die Transitionen, die pro Animationsschritt schalten. Ein Animationsschritt kann in einer vom Anwender eingestellten Zeitspanne ausgeführt werden. Die Einstellung dieses Parameters (*Duration*) wurde oben bereits besprochen.

Kurz zusammengefasst kann ein Clip in mehrere Animationsschritte unterteilt werden. In einem Animationsschritt können, je nach Schaltregel, mehrere Transitionen feuern. Sollte in einem Schritt keine Transition mehr schalten können, dann ist ein toter Zustand erreicht und die Aufnahme des Clips wird automatisch beendet.

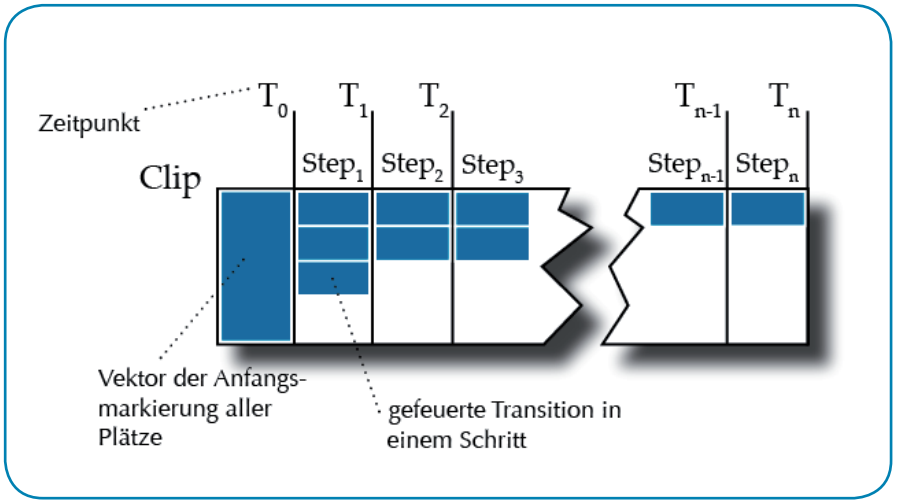


Abb. 9
Schematische Darstellung der Struktur eines Clips

Nach der Betrachtung der Steuerung und einer Analyse von Petrinetz-Animationen in *Snoopy* kann nun ein Konzept entwickelt werden, dass die Aufnahme von Clips gestattet.

Zu Beginn einer Aufnahme muss die Anfangsmarkierung aller Plätze festgehalten werden. Während der Aufnahme müssen alle Transitionen gespeichert werden, die in einem Animationsschritt geschaltet haben. Diese Daten reichen aus, um einen Clip später wieder abzuspielen. Dafür muss die Anfangsmarkierung zum Zeitpunkt T_0 wieder hergestellt werden. Bevor dies geschieht, sollte dem Anwender über einen Fenster-Dialog eine Warnung ausgegeben werden. Hier kann er wählen, ob der Clip abgespielt werden soll. Entscheidet sich der Anwender für das Abspielen wird die Liste der in jedem Schritt gespeicherten Transitionen durchlaufen und abermals zum Schalten gebracht.

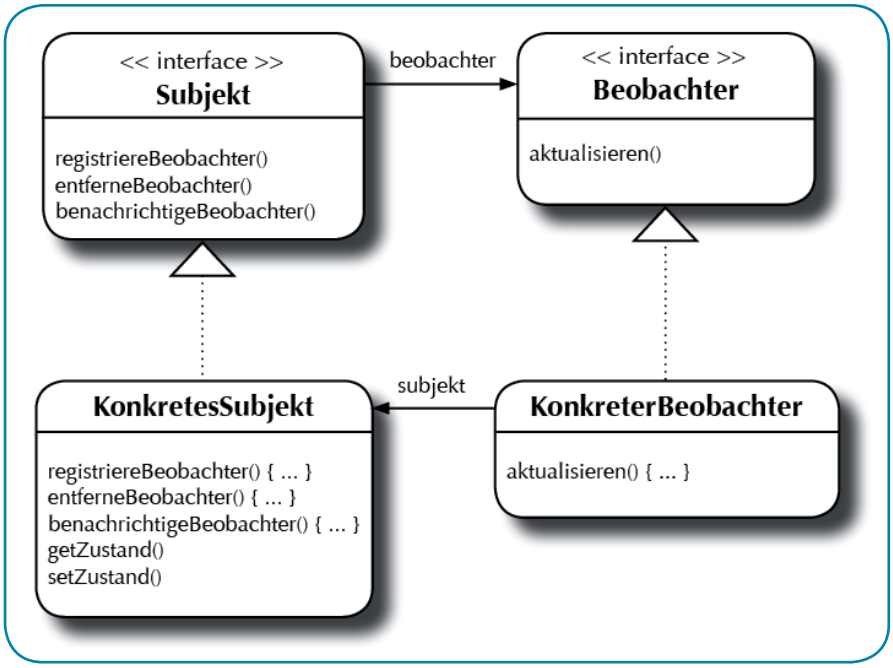
Der Entwurf für das Aufnehmen der betreffenden Transitionen wird mit dem Beobachter-Muster (engl. *observer pattern*) umgesetzt. Dieses Entwurfsmuster definiert sich wie folgt:

> Das Beobachter-Muster beschreibt eine Eins-zu-viele-Abhängigkeit zwischen Objekten in der Art, dass alle abhängigen Objekte benachrichtigt werden, wenn sich der Zustand des einen Objekts ändert [Freeman06].

Die zu benachrichtigenden Objekte sind die Beobachter eines einzelnen Objekts, das Subjekt genannt wird. Sollte sich der

Zustand von dem Subjekt ändern, werden die Beobachter davon in Kenntnis gesetzt. Dadurch entsteht eine lockere Bindung zwischen den Objekten, die sich am besten durch eine Subjekt- und Beobachter-Schnittstelle implementieren lässt. Folgender Klassenentwurf soll dies verdeutlichen.

Abb. 10
Klassendiagramm des Beobachter-Musters



Ein konkreter Beobachter implementiert nur das Beobachter-Interface und das konkrete Subjekt kann ihn registrieren, benachrichtigen und auch wieder aus der Liste der Beobachter entfernen.

Wendet man dieses Muster nun auf den Kontext des Aufnehmens von Clips in *Snoopy* an, dann stellt das zu beobachtende Subjekt eine Komponente dar, die alle Ereignisse in einem Netz überwacht. Der Beobachter wiederum ist eine Komponente, die über das Ereignis des Schaltens einer Transition benachrichtigt wird.

Dieser Entwurf wurde bereits mit dem Konzept der Interaktion von Netzklassen in *Snoopy* von Matthias Dube [Dube07] realisiert. Durch die Verknüpfung von zwei oder mehreren Netzen kann eine Interaktion zwischen den Netzen stattfinden, die über definierte Befehle (engl. *commands*) miteinander kommunizieren. Diese Befehle werden erst durch das Eintreten von bestimmten Ereignissen (engl. *events*) ausgelöst.

Durch die Möglichkeit ein Netz mit sich selbst zu verknüpfen, kann die Aufnahme von Clips realisiert werden. Hierzu wird das Ereignis des Schaltens einer Transition benutzt, um einen Aufnahme-Befehl auszulösen. Der Befehl meldet die betreffende Transition bei der Beobachter-Komponente.

3.1.1 GUI

In Anlehnung an die bereits bestehenden Buttons zur Steuerung der Animation wurde das Konzept eines Record-Buttons entwickelt. Durch diesen wird das Programm in einen Aufnahme-Modus versetzt. In diesem Modus werden bei Betätigung des Play-Forward-Buttons die Animationsschritte aufgezeichnet. Der Aufnahme-Modus ist so lange aktiv, bis der Anwender den Record-Button erneut drückt oder das Petrinetz einen toten Zustand erreicht.

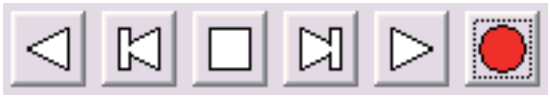


Abb. 11
Buttons zur Steuerung der Animation und Record-Button zur Steuerung der Aufnahme

Während der Aufnahme kann es von Vorteil sein, wenn zu einem beliebigen Zeitpunkt gestoppt werden kann, um später von diesem Zustand aus wieder zu starten. Die bestehende Steuerung gestattet es, eine Animation durch erneutes Drücken des Play-Forward-Buttons zu unterbrechen. Da der Aufnahmeprozess nur auf das Schalten der Transitionen reagiert, ist die Möglichkeit des kurzzeitigen Unterbrechens einer Aufnahme bereits gewährleistet. Allgemein gesagt: Solange der Aufnahme-Modus aktiv ist, sollten alle Schaltvorgänge im Petrinetz registriert und aufgenommen werden. Hierbei sollte es keine Rolle spielen, ob der Anwender die Animation über den Play-Forward-Button oder den Single-Step-Forward-Button abspielt. Auch das manuelle Feuern durch Klicken auf eine Transition ist dabei berücksichtigt.

3.1.2 Anwendungsfall *Clip aufnehmen*

Der Anwendungsfall beschreibt den möglichen Prozess der Aufnahme eines Clips:

Der Anwender betätigt den Record-Button und der Aufnahme-Modus ist aktiv. Durch das Drücken des Play-Forward-Buttons wird die Aufnahme gestartet. Nachdem etwas Zeit vergangen ist und der ungehinderte Animationsablauf immer noch gegeben ist, wird der Play-Forward-Button abermals vom Anwender gedrückt. Die Animation hält an. Der Aufnahme-Modus ist immer noch aktiv und der Anwender schaltet durch den Step-Forward-Button die Animation um einen Schritt weiter. Dieser Schritt führt in diesem Fall zu einem toten Zustand und die Aufnahme wird automatisch vom Programm beendet.

3.2 Verwalten

Der Entwurf eines Objekts zum Verwalten von Clips ist Gegenstand dieses Abschnitts. Dieses Verwalter-Objekt, kurz Verwalter genannt, steuert die Initialisierung, die Manipulation und das Löschen von Clips. Gleichzeitig soll dieser Entwurf auch die Kommunikation zwischen den in *Snoopy* vorhandenen Software-Komponenten mit einschließen. Das wären zum Beispiel die Komponenten des GUI, die die Clips in einer Liste darstellen oder die Komponenten, die für das Speichern und Laden von Clips verantwortlich sind. Das Zusammenspiel der verschiedenen Komponenten kann vom Vermittler-Muster (engl. *mediator pattern*) realisiert werden.

Dieses objektbasierte Verhaltensmuster fördert die lose Kopplung, indem es Objekte davon abhält, aufeinander explizit Bezug zu nehmen. Das Zusammenspiel verschiedener Objekte kann dann unabhängig von ihnen durch den Vermittler variiert werden [Gamma96].

Der Verwalter vermittelt zwischen den Software-Komponenten, welche die Clips für unterschiedliche Zwecke weiter verarbeiten. Genauer sind dies die Komponenten des Speicherns, des Ladens, des Exportierens und des Importierens von Clips. Zusätzlich übernimmt der Verwalter in seiner Rolle folgende

elementare Aufgaben:

- Er muss bei der Erstellung eines neuen Clips für dessen temporäre Speicherung sorgen, zum Beispiel in einer Clip-Liste. In dieser Liste werden die Clips verwaltet und aus ihr können sie auch wieder gelöscht werden.

- Ebenfalls muss der Verwalter eine Methode zur Manipulation des Namens eines Clips bereitstellen, da dies die einzigen Daten sind, die der Anwender nach der Aufnahme eines Clips verändern kann.

- Um die Clips für andere Komponenten von *Snoopy* zugänglich zu machen, müssen entsprechende Methoden vorhanden sein.
  Das sind zum Beispiel Komponenten, die für das Speichern, Laden und Exportieren von Clips verantwortlich sind.

3.2.1 GUI

Sobald der Anwender den Aufnahme-Modus beendet hat, soll der Clip für ihn in einer Auswahlliste sichtbar sein. Der Name des gerade aufgenommenen Clips wird zuerst vom Programm generiert, muss aber jederzeit durch den Anwender änderbar sein. Hierzu bietet sich ein einfaches Texteingabefeld an.

Abb. 12
Erweitertes Animationsfenster mit zusätzlichen GUI-Elementen: Auswahlliste, Texteingabefeld und Button zum Löschen von Clips

Damit der Anwender die Clips in der Liste auch wieder löschen kann, muss ein Button zum Entfernen der Clips vorhanden sein.

3.2.2 Anwendungsfall *Clip verarbeiten*

Folgender Anwendungsfall wäre denkbar:

Der Anwender hat einen Clip aufgenommen. Durch einen einfachen Mausklick auf den Clip innerhalb der Auswahl-Liste wird dieser markiert und gleichzeitig der Wert des generierten Namens (z.B. „Clip1“) in das Textfeld geladen. Jetzt ändert der Anwender im Textfeld den Namen in „Mein Clip“ und betätigt die Enter-Taste. Die Liste wird aktualisiert. Das hieße, dass statt „Clip1“ jetzt „Mein Clip“ in der Liste angezeigt wird.

Nun möchte der Anwender seinen gerade umbenannten Clip abspielen. Dazu markiert er den Clip in der Auswahl-Liste durch einem Doppelklick mit seiner Maus.

Nachdem der Anwender den Clip gesichtet hat, beschließt er ihn zu löschen. Der Anwender markiert den Clip in der Auswahl-Liste durch einen einfachen Mausklick und drückt anschließend den Löschen-Button. Der Clip wird aus der Liste entfernt.

3.3 Speichern und Laden

Für das Speichern und Laden von Clips in *Snoopy* wird ein eigenes Format in XML-Syntax definiert. Dies hat den Vorteil, dass es leicht zu parsen ist. Weiterhin bietet *Snoopy* bereits Komponenten zum Speichern und Laden von XML-Dateien an. Die Klasse `SP_XmlWriter` schreibt die Struktur eines Graphen als XML-Format in eine Datei. Diese Datei kann durch die Klasse `SP_XmlReader` wieder eingelesen werden. Es sind in der Implementierung zwei neue Klassen zu erstellen, die von SP_XmlWriter und SP_XmlReader erben. Die Methoden zum Schreiben und Laden von XML-Dateien werden von den neuen Klassen so überschrieben, dass das Clip-Format ausgegeben wird.

Der Anhang A dieser Bachelorarbeit zeigt eine Struktur-Beschreibung des XML-Dokuments an, in das die Clips gespeichert werden. Diese Beschreibungssprache wird **X**ML-**S**chema-**D**efinition (XSD) genannt und gehört zu den Standards, die das **W**orld **W**ide **W**eb **C**onsortium (W3C) veröffentlicht hat [XSD].

Folgende Elemente und Strukturen sind in der XSD-Datei beschrieben:

- `<xs:element name=“Snoopy“>`

  Dies stellt das Root-Element der XML-Datei dar und orientiert sich an dem XML-Format der SPPED-Dateien. Auch die in SPPED-Dateien verwendeten Attribute `version` und `revision` sind im Root-Element berücksichtigt. Folgende Sequenz von Elementen ist enthalten:

- `<xs:element name=“reference“ type=“xs:string“>`

  In diesem Element wird der Name der SPPED-Datei, auf den sich der Clip bezieht, gespeichert. Da es im Dateisys-

tem des Rechners mehrere SPPED-Dateien mit gleichem Namen geben kann, wird der absoluten Pfad zu der Referenz hinzugefügt.

- `<xs:element name="properties">`

  Hier werden die Eigenschaften der Animation festgehalten. Das sind im Einzelnen die selben Parameter, die unter dem Fenster-Dialog *Animation Properties* vom Anwender gesetzt werden können:

  ```
  <xs:element name="refresh" type="xs:unsignedInt">
  <xs:element name="duration" type="xs:unsignedInt">
  <xs:element name="stepping">
  ```

  Wobei `<xs:element name="stepping">` die Beschränkung auf `<xs:enumeration value="Single">`, `<xs:enumeration value="Intermediate">` und `<xs:enumeration value="Maximum">` festlegt.

- `<xs:element name="clips">`

  Dieses Element kann eine unbegrenzte Menge an Clips enthalten. Ein Clip-Element `<xs:element name="clip">` wird wie folgt beschrieben:

- `<xs:element name="name" type="xs:string">`

  Der Name des Clips wird in diesem Element festgehalten.

- `<xs:element name="initialmarking">`

  Das Element kann null oder eine unbegrenzte Anzahl von Elementen des Typs `<xs:element name="place">` beinhalten. Konkret beschreibt dies die Menge aller Plätze und ihrer Markierung zum Startpunkt T_0.

- `<xs:element name="steps">`

  Jenes Element beinhaltet die Schritte, die zu einem Clip-Element gehören. Ein Schritt-Element `<xs:element name="step">` wiederum muss aus mindestens einem

Transitions-Element `<xs:element name=“transition“>` bestehen.

Die Komponenten zum Speichern und Laden dieses Clip-Formats in XML-Syntax sollen über das in Abschnitt 3.2 vorgestellte Verwalter-Objekt angesteuert werden.

3.4 Importieren

Der Entwurf für das Importieren von Gegenbeispielen (engl. *counterexamples*) gestaltet sich schwierig, da es keine genormte Syntax für die Beschreibung solcher Strukturen gibt. Die Liste der bekannten Analyse-Werkzeuge, die Gegenbeispiele im Textformat ausgeben sind aber überschaubar:

- PEP
- PROD
- SPIN
- CHARLIE

Diese Werkzeuge [PEP] [PROD] [SPIN] [CHARLIE] können Petrinetze auf ihre Eigenschaften hin überprüfen.

Das Model-Checking Kit [MCKit] der Universität Stuttgart bietet den Komfort die Algorithmen der ersten drei Werkzeuge PEP, PROD und SPIN in einer einzigen Anwendung auszuführen:

Als Beispiel werden die beiden Dateien `dekker.cfa` [Anhang D] und `liveness_1.ltl` [Anhang C] als Parameter übergeben. Das Netz, dass in `dekker.cfa` beschrieben ist, wird mit der temporalen Logik in `liveness_1.ltl` auf Lebendigkeit geprüft:

- **mit PEP-Algorithmus**

  Eingabe:
  ```
  check cfa:pep-ltl dekker.cfa liveness_1.ltl
  ```

  Ausgabe:
  ```
  Time needed: 0.180000 seconds
  Result: NO.
  Counterexample:
  ```

```
dek2: B1 [ flag2=0 --> flag2=1 ] B2
dek2: B2 [ flag1=0 --> flag1=0 ] B3
dek1: A1 [ flag1=0 --> flag1=1 ] A2
*** loop starts here ***
dek1: A2 [ flag2=1 --> flag2=1 ] A5
dek1: A5 [ turn=1 --> turn=1 ] A2
*** loop ends here ***
```

- **mit PROD-Algorithmus**

  Eingabe:
  ```
  check cfa:prod-ltl dekker.cfa liveness_1.ltl
  ```

  Ausgabe:
  ```
  Time needed: 3.330000 seconds
  Result: NO.
  Counterexample:
  dek2: B1 [ flag2=0 --> flag2=1 ] B2
  dek1: A1 [ flag1=0 --> flag1=1 ] A2
  *** loop starts here ***
  dek1: A2 [ flag2=1 --> flag2=1 ] A5
  dek1: A5 [ turn=1 --> turn=1 ] A2
  *** loop ends here ***
  ```

- **mit SPIN-Algorithmus**

  Eingabe:
  ```
  check cfa:spin-ltl dekker.cfa liveness_1.ltl
  ```

  Ausgabe:
  ```
  Time needed: 0.650000 seconds
  Result: NO.
  Counterexample:
  dek2: B1 [ flag2=0 --> flag2=1 ] B2
  dek1: A1 [ flag1=0 --> flag1=1 ] A2
  dek1: A2 [ flag2=1 --> flag2=1 ] A5
  dek2: B2 [ flag1=1 --> flag1=1 ] B5
  dek2: B5 [ turn=1 --> turn=1 ] B6
  dek1: A5 [ turn=1 --> turn=1 ] A2
  dek2: B6 [ flag2=1 --> flag2=0 ] B7
  dek1: A2 [ flag2=0 --> flag2=0 ] A3
  dek1: A3 [ turn=1 --> turn=2 ] A4
  dek2: B7 [ turn=2 --> turn=2 ] B8
  ```

```
dek2: B8 [ flag2=0 --> flag2=1 ] B2
dek2: B2 [ flag1=1 --> flag1=1 ] B5
dek1: A4 [ flag1=1 --> flag1=0 ] A1
dek2: B5 [ turn=2 --> turn=2 ] B2
dek1: A1 [ flag1=0 --> flag1=1 ] A2
dek1: A2 [ flag2=1 --> flag2=1 ] A5
dek1: A5 [ turn=2 --> turn=2 ] A6
dek2: B2 [ flag1=1 --> flag1=1 ] B5
dek1: A6 [ flag1=1 --> flag1=0 ] A7
dek2: B5 [ turn=2 --> turn=2 ] B2
*** loop starts here ***
dek1: A7 [ turn=2 --> turn=2 ] A7
*** loop ends here ***
```

Die gefundenen Gegenbeispiele der drei unterschiedlichen Algorithmen werden in einer einheitlichen Syntax ausgegeben.

Die Ausgabe beim Werkzeug CHARLIE unterliegt wiederum einer anderen Syntax. Als Gegenstand der Analyse wurde ein anderes Netz geprüft:

```
reading net: travel.apnn
net loaded
places: 12
transitions: 12


-- reachability graph constructed in 0m0.0s--
used construction options:
firing rule: single
boundedness check: yes
maximal construction depth: none
reduction: none
```

```
states: 26
edges: 69
#scc: 1
#terminal scc: 1
formula: F hotel
negation: [false R hotel != 1 ]
the formula is: false
counter example:
( T_0 , T_1 , T_5 , T_6 , T_7 , T_8 )
0m0.0s
```

Der Entwurf einer Komponente für den Import von Gegenbeispielen in *Snoopy* muss verschiedene Ausgaben von Gegenbeispielen verabeiten. Durch Syntax-Analyse, auch Parsen genannt, ist die Import-Komponente in der Lage das Textformat in die Datenstruktur einer Liste von Transitionen umzuwandeln.
Die Liste wird an das Verwalter-Objekt übergeben. Anhand der in der Liste enthaltenen Transitionen und deren lineare Abfolge innerhalb der Liste wird die entsprechende Clip-Struktur vom Verwalter generiert. Es ist zu beachten, dass die Anfangsmarkierung des generierten Clips gleich der Markierung des analysierten Petrinetzes ist.

3.5 Exportieren

Ein wesentlicher Teil dieser Arbeit ist die Aufgabe des Exportierens der aufgenommenen Clips in ein geeignetes Webformat. Das Export-Format muss die plattformunabhängige Darstellung und Interaktion der Animation in einem Webbrowser unterstützen.

Das simple HTML-Format ist nicht in der Lage, die teilweise recht komplexen Vektorgrafiken eines Petrinetzes, geschweige denn die Animation eines Markenspiels darzustellen. Es muss ein anderes Format gefunden werden, in das ein Clip aus *Snoopy* heraus exportiert werden kann.

Es gibt derzeit zahlreiche Formate, die den Anforderungen der Animation und Interaktion genügen. Einige von ihnen werden nachfolgend vorgestellt und bewertet. Abschließend wird be-

urteilt, welches Format sich für den Export am besten eignet.

3.5.1 Shockwave Flash

Das erste in Frage kommende Format ist das sogenannte **S**hock**w**ave **F**lash (SWF), besser bekannt als *Flash*. Dieses auf Vektorgrafik basierende Grafik- und Animationsformat zur Erstellung von interaktiven Multimedia-Anwendungen wurde ursprünglich von der Firma *Macromedia* entwickelt, die durch die Firma *Adobe* im Jahre 2005 übernommen wurde. Meistens wird die Quelldatei (FLA-Format) dieses Formats mit der proprietären integrierten Entwicklungsumgebung *Adobe Flash* erstellt und anschließend zum Einsatz auf einem Webserver in das SWF-Format kompiliert. Dadurch ergibt sich auch ein großer Nachteil, da die Dateien in dieser Form nicht mehr einfach verändert werden können. Die Tatsache, dass die zugrundeliegende Technologie nicht in Form offener Standards spezifiziert wurde, macht es schwierig dieses Format mit anderer Software zu erstellen. Auch die vom Hersteller empfohlene Einbindung in Webseiten durch das HTML-Element <embed>, welches nie in den W3C-Standards zu HTML enthalten war, bereitet einige Probleme. Zur Darstellung im Browser ist außerdem ein Plug-in erforderlich, das aber laut *Adobe* einen weltweiten Verbreitungsgrad von über 97 Prozent besitzt [Adobe1]. Insbesondere die nach außen geschlossene Technologie des SWF-Formats führt dazu, dass sich dieses Format kaum in Snoopy integrieren lässt. Somit eignet sich das SWF-Format nicht für das Exportieren von Clips.

Das Format eignet sich zwar nicht zur Integration in *Snoopy*, aber es gibt bereits Implementierungen in Flash, die den Markenfluß visualisieren. Eine flash-basierte Lösung zur Darstellung wurde zum Beispiel an der Yamaguchi Universität in Japan umgesetzt. Die Website des Projekts *Petri Net Pathways* der wissenschaftlichen Fakultät bietet animierte Modelle von metabolischen Netzwerken an [PNP].

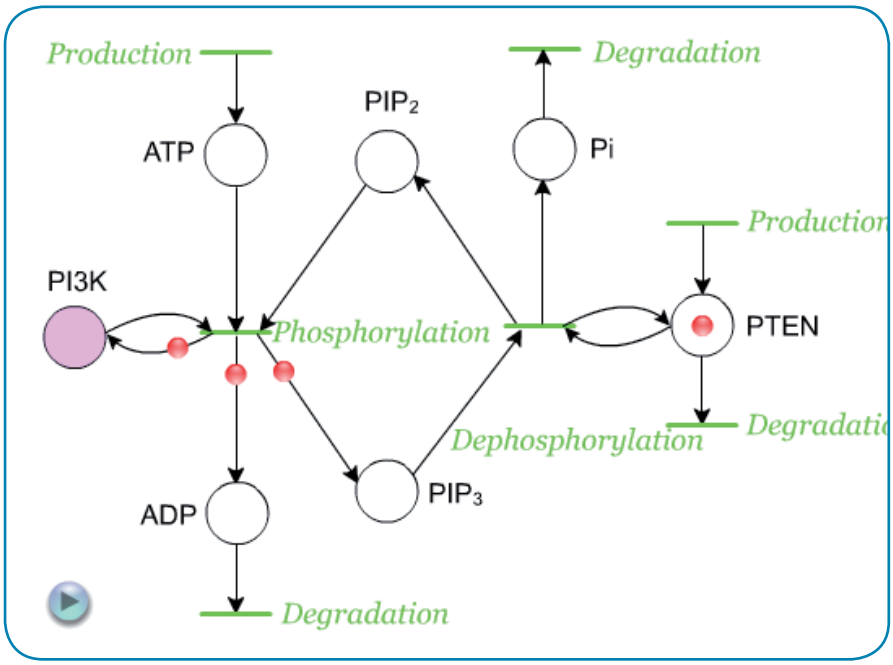


Abb. 13
Flash-basierte Petrinetz-Animation

Das einzige Element zur Steuerung der Animation ist ein Play-Button unten links, der die Animation zwar startet aber nicht wieder anhält.

3.5.2 Java Applet

Ein weiteres in Frage kommende Format wäre das *Java*-Applet, welches durch eine im Webbrowser integrierte *Java*-VM, also eine virtuelle Laufzeitumgebung für *Java*, ausgeführt wird.
Dieses in der Programmiersprache *Java* geschriebene Format wurde von der Firma *Sun Microsystems* entwickelt. Die Technologie der Applets bietet dem Entwickler den vollen Funktionsumfang der *J2SE*-API, womit sich komplexe interaktive Anwendungen erstellen lassen, die ohne großen Aufwand mit unterschiedlichen Browsern und auf verschiedenen Betriebssystemen laufen können. Nachteilig ist die teilweise lange Initialisierungszeit für die *Java*-VM und die Dauer des Herunterladens und das Initialisieren des Applets im Browser. Doch der größte Nachteil, warum sich auch dieses Format nicht für den Export eignet, ist die Tatsache, dass das Applet von Snoopy generiert und kompiliert werden müßte. Da Snoopy aber in der Programmiersprache C++ geschrieben ist, gestaltet sich

die Integration der Applet-Generierung sehr schwierig. Theoretisch könnte man das Applet auch außerhalb von *Snoopy* entwickeln und müßte dann eine Kommunikation für den Datenaustausch zwischen den beiden Programmen gewährleisten. Der Aufwand der Konzeptionierung und Implementierung dieser Neuentwicklung wäre zu groß und würde den Rahmen dieser Bachelor-Arbeit sprengen. Hinzu kommt, dass die kompakte Struktur der interagierenden Komponenten der Software *Snoopy* aufgebrochen werden müßte. Dies wäre sicher nicht im Sinne der weiteren Entwicklung von *Snoopy*.

3.5.3 Scalable Vector Graphics

Als Letztes soll das Format der **S**calable **V**ector **G**raphics (SVG) näher betrachtet werden. Es handelt sich um einen vom *W3C* veröffentlichten Standard [SVG1] zur Beschreibung zweidimensionaler Vektorgrafiken in der XML-Syntax. Er kann von vielen Webbrowsern dargestellt werden, da sie SVG bereits nativ unterstützen. Ein Negativbeispiel dafür ist der *Internet Explorer* der Firma *Microsoft*, der für die Darstellung des SVG-Formats ein Plugin, z.B. den *SVG Viewer* von *Adobe*, benötigt. Die Weiterentwicklung und der Support dieses Plugins wird aber ab dem 01.01.2009 nicht mehr vorangetrieben [Adobe2].

Die Möglichkeit der Animation von Vektorgrafiken wird über den XML-Standard **S**ynchronized **M**ultimedia **I**ntegration **L**anguage (SMIL) in SVG integriert [SMIL05]. Durch die für Animationen ausgezeichneten Elemente kann das Markenspiel eines Petrinetzes animiert werden:

- `<animate>`

  Mit diesem Element lassen sich einzelne Attribute des übergeordneten Elements, auch Elternelement genannt, animieren.
  Damit können Animationen für GUI-Komponenten realisieren werden.

- `<animateMotion>`

  Dieses Element animiert sein Elternelement entlang eines

vorher definierten Pfades. Dadurch können die Token des Petrinetzes entlang der Kanten animiert werden.

- `<animateColor>`

  Durch dieses Element lassen sich die Farbwerte des Elternelements interpolieren. Die an einer Animation beteiligten Netzstrukturen können eingefärbt werden.

Folgende Tabelle spiegelt die implementierte Funktionalität zur Darstellung von SVG in den gängigsten Browsern wieder. Die Daten der Tabelle wurden im Laufe dieser Bachelor-Arbeit von den Webseiten der Browseranbieter zusammengetragen [MOZ] [FOX] [OPERA] [WKit]. Es soll gezeigt werden, ob der vom *W3C* definierte SVG-Standard von möglichst vielen Browsern unterstützt wird. Durch die SVG-Testsuite des *W3C* [W3C1] wurden die implementierten Spezifikationen persönlich verifiziert. Da der *Internet Explorer* von *Microsoft* weder in der Version 6 noch in der aktuellsten Version 7 native Unterstützung für SVG bietet, wird er in dieser Tabelle nicht explizit erwähnt. Stattdessen wurde der *SVG Viewer* von *Adobe* als Plugin für die Versionen 6 und 7 des *Internetexplorers* getestet.

Tabelle 1
Implementierte Spezifikationen von verschiedenen Webbrowsern auf verschiedenen Betriebssystemen zur Darstellung von SVG

Betriebssystem	Browser	Grundformen	Scripting	Animationen
Windows XP	Safari	Ja	Ja	Nein
	Firefox 2.0	Ja	Ja	Nein
	Opera 9	Ja	Ja	Ja
	Adobe Plugin SVG Viewer 3	Ja	Ja	Ja
Mac OS X	Firefox 2.0	Ja	Ja	Nein
	Opera 9	Ja	Ja	Ja
	Safari	Ja	Ja	Nein
Linux	Firefox 2.0	Ja	Ja	Nein
	Opera 9	Ja	Ja	Ja

Anhand der Tabelle 3.1 ist abzulesen, dass einige Browser nicht in der Lage sind die Animations-Elemente richtig zu interpretieren. Nur Opera ab der Version 9 und das Plugin von Adobe untestützen die definierten Standards komplett.

3.5.4 Fazit

Da es sich bei SVG um ein reines Textformat handelt, sind die Vorteile gegenüber den anderen Formaten offensichtlich: Die Daten lassen sich mit jedem beliebigen Texteditor ansehen und bearbeiten, Das Format ist lizenzfrei und plattformunabhängig und kann durch die XSL-Technologie [XSL1] in jedes andere XML-Format umgewandelt werden.

Die mangelnde Umsetzung der Animationselemente in den meisten Browsern kann durch die Script-Unterstützung ausgeglichen werden. Damit ist die Wiedergabe der in *Snoopy* aufgenommenen Clips in SVG möglich. Diese Möglichkeit der Animation wurde mit Javascript prototypisch umgesetzt und getestet. Dieser Prototyp wird im nachfolgenden Kapitel der Implementierung ausführlich besprochen.

Die Vorteile überwiegen und dadurch eignet sich das SVG-Format am besten für den Export von Clips.

Implementierung

Kapitel 4

Zu Beginn der Implementierung wurde die Beschreibung einer Transformation in XSL-Syntax (XSL – e**X**tensible **S**tylesheet **L**anguage) erstellt. Mit Hilfe dieser Datei [Anhang B] ist es möglich die XML-Struktur einer beliebigen SPPED-Datei in die Struktur einer SVG-Datei zu transformieren. Die Transformation entspricht dabei der Funktionalität der Export-Komponente aus Kapitel 3 Abschnitt 5. Das Stylesheet kann somit als eine prototypische Umsetzung der Export-Komponente aufgefasst werden.

Noch bevor eine entsprechende Klasse der Export-Komponente in C++ programmiert wird, kann mit dem Prototyp getestet werden, welche SVG-Elemente die komplexen Strukturen eines Petrinetzes abbilden. Ebenso läßt sich der Markenfluß mit Javascript umsetzen und das Ergebnis kann kurzerhand in einem Webbrowser betrachtet werden. Ohne diesen Prototypen müßte zum Testen der Export-Komponente das Programm *Snoopy* immer wieder neu kompiliert werden, was auf Dauer sehr zeitintensiv ist.

Beim Testen der Ergebnisse der XSL-Transformation hat sich gezeigt, dass eine möglichst genaue Darstellung von Petrinetzen in SVG, insbesondere hierarchischer Strukturen, nach einer Neukonzeption dieser Arbeit verlangt. Es sollen folgende Aufgaben vom Export-Format erfüllt werden:

- Ein Petrinetz soll im Webbrowser beliebig skaliert werden können. Daraus folgt, dass

- es eine Funktion zum Verschieben eines Petrinetzes geben muss, damit man bei einem hohen Skalierungsfaktor zu allen Bereichen eines Petrinetzes gelangen kann.

- Die Möglichkeit einer Navigation durch ein hierarchisches Petrinetz soll gegeben sein. Dies schließt auch die Erstellung entsprechender GUI-Elemente mit ein.

- Für die Visualisierung des Markenspiels und das Abspielen von Clips muss die Schaltregel für Petrinetze in Javascript nachgebildet werden. Auch das manuelle Schalten durch Klicken auf eine Transition soll möglich sein.

- Es sollen GUI-Elemente geschaffen werden, womit der Markenfluß gesteuert wird. Dies sind im Einzelnen Buttons zum Abspielen, Stoppen und Aktivieren einer Single-Step-Animation.

Die Erfüllung dieser Anforderungen nahm mehr Zeit in Anspruch als vorgesehen war. Vor allem die Implementierung für die Steuerung der Animation und für das Navigieren durch hierarchische Petrinetze gestaltet sich schwierig. Die Ursache dafür liefert die Tatsache, dass es keine vorgefertigten GUI-Elemente in SVG gibt. Das Aussehen und Verhalten der grafischen Benutzerschnittstelle muss selbst entworfen und implementiert werden. Aus diesem Grund wurde nur ein Teil der ursprünglichen Aufgabenstellung umgesetzt, da es nicht genug Ressourcen gab, alle Anforderungen zu erfüllen.

Nachfolgend werden die Komponenten vorgestellt, die innerhalb des Rahmens dieser Arbeit implementiert wurden.

4.1 Prototyp

Wie in der Einleitung dieses Kapitels erwähnt, soll der Prototyp eine Unterstützung bei der Implementierung der Export-Komponente sein. Der Prototyp beinhaltet eine Reihe von XSL-Technologien, die vom *W3C* als Standard herausgegeben wurden:

- XML Path Language (XPath)
- XSL Transformation Language (XSLT)

Eine XSL-Datei besteht aus Anweisungen in XSLT-Syntax [XSLT]. Mit ihnen wird festgelegt, welche Elemente einer XML-Datei wie transformiert werden sollen. Die zu verarbeitenden Elemente wählt man mit einer XPath-Anfrage aus [XPath]. Eine Anfrage beschreibt die Menge von Elementen, die den XPath-Ausdruck erfüllen.

Zu jeder Struktur in einer SPPED-Datei gibt es eine entsprechende Darstellung in SVG. Um dies zu erreichen, werden verschiedene Templates definiert. Die wichtigsten Templates und deren Beitrag zur Transformation sind nachfolgend erläutert:

- `<xsl:template match="Snoopy">`

  Das Template reagiert auf das Root-Element der SPPED-Datei (`Snoopy`) und gibt die grundlegende Struktur eines SVG-Dokuments aus. Alle weiteren Templates werden innerhalb dieses Templates aufgerufen.

- `<xsl:template match="nodeclass[@name='Place']/node | nodeclass[@name='Coarse Place']/node">`

  Das Template wird ausgeführt wenn in der SPPED-Datei normale Plätze (`Place`) oder auch hierarchische Plätze (`Coarse Place`) vorhanden sind. Die Plätze werden als Kreise (`circle`) in SVG-Syntax ausgegeben. Innerhalb eines Kreises, der für die Visualisierung eines hierarchischen Platzes steht, wird ein kleinerer Kreis definiert.

- `<xsl:template match="nodeclass[@name='Transition']/node | nodeclass[@name='Coarse Transition']/node">`

  In diesem Template werden die Transitionen – egal ob normale oder hierarchische – durch Rechtecke (`rect`) dargestellt. Eine hierarchische Transition wird zusätzlich innen mit einem kleineren Rechteck versehen.

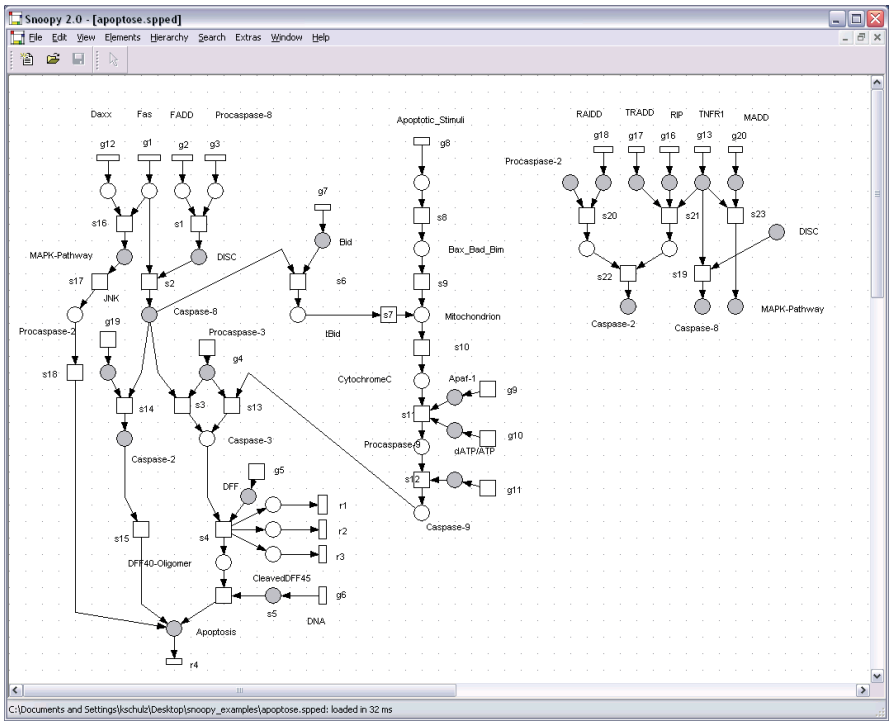
- `<xsl:template match="edge">`

  Bei diesem Template werden die Kanten durch Linien (`polyline`) bezeichnet. Da es sich um gerichtete Kanten handelt, wird das `polyline`-Element mit dem Attribut `marker-end` belegt. Der Wert des Attributs entspricht einer vorher definierten Pfeilspitze (`marker-end="url(#Triangle)"`).

Es wurde darauf Wert gelegt, das die Darstellung eines Petrinetzes in Snoopy und in einem Webbrowser möglichst genau übereinstimmt.

Der untere Screenshot zeigt ein Petrinetz im Werkzeug Snoopy an.

Abb. 14
Screenshot eines Petrinetzes in Snoopy



Es handelt sich um ein Modell der Apoptose. Die Darstellung auf der gegenüberliegenden Seite ist der Screenshot eines Safari-Browsers. Es wird die SVG-Datei angezeigt, welche durch die XSL-Transformation erzeugt wurde.

Durch die Anweisungen in der XSL-Datei wurden die Strukturen in entsprechende SVG-Elemente übertragen.

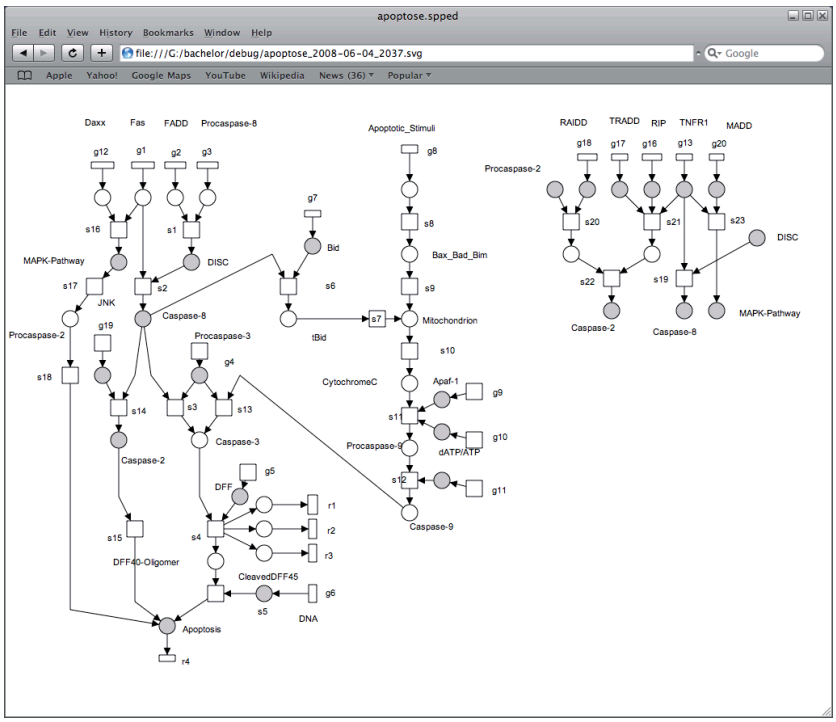


Abb. 15
Darstellung der erzeugten SVG-Datei im Webbrowser Safari

Die Struktur des Netzes wird im Browser richtig wiedergegeben. Es ist zu Testen, ob andere Strukturen von Petrinetzen auch fehlerfrei angezeigt werden.

Die eigentliche Transformation von einem Format in ein anderes wird durch einen XSLT-Prozessor vollzogen. Der Prozessor ist eine Implementierung, welche die definierten Sprachen XSLT und XPath umzusetzen weiß.

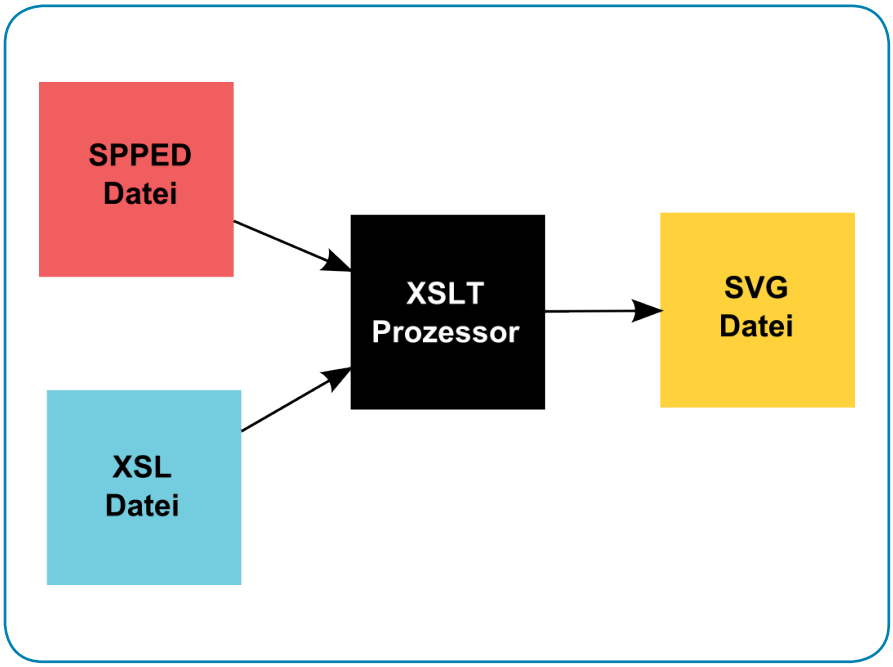


Abb. 16
Schematische Darstellung der Transformation einer SPPED-Datei in eine SVG-Datei mit Hilfe einer XSL-Datei und einem XSLT-Prozessor

Die Liste der verfügbaren Werkzeuge, die einen solchen XSLT-Prozessor implementieren ist lang. Das ist darauf zurückzuführen, dass bereits viele Texteditoren und Webbrowser einen Prozessor integriert haben.

Während der Implementierungsphase wurde das Werkzeug *XML Copy Editor* von Gerald Schmidt [XCE] zum Bearbeiten des Stylesheets genutzt. Mit dieser freien Software (GNU General Public License) lassen sich auch XSL-Transformationen ausführen.

4.2 Steuerung

Ein Petrinetz soll in der Anzeige eines Webbrowsers skaliert – genauer vergrößert oder verkleinert – und verschoben werden können. Der SVG-Standard bietet für diese Aufgaben nur unzureichende Möglichkeiten. Deswegen wurden dafür Steu-

erungs-Routinen in Form von Javascript-Funktionen geschrieben.

Die Skalierung bezieht sich immer auf den momentanen Skalierungsfaktor des Netzes. Dieser Faktor wird über ein Attribut (`snp:scale`) festgelegt. Da dieses Attribut nicht im Standard von SVG enthalten ist, wird es über einen neuen Namensraum im Root-Element des Stylesheets definiert (`xmlns:snp="http://www-dssz.informatik.tu-cottbus.de/"`). Dadurch ist es möglich neue Attribute in die SVG-Syntax zu integrieren. Diese Attribute wirken sich nicht auf die Darstellung aus, kommen aber bei der Problemlösung der Aufgaben zum Einsatz.

4.2.1 Skalieren

Folgende Funktion wurde implementiert:

```
    function zoomIn(value){
      var scale = Number(net.getAttribute(,snp:scale‘)) + value;
      var translate = net.getAttribute(,snp:translate‘);
      net.setAttribute(,transform‘, ,scale(,+scale+‘) translate(,+translate+‘)
‘);
      net.setAttribute(,snp:scale‘, scale);
    }
```

Es wird der Skalierungsfaktor als Funktionsparameter (`value`) übergeben. Dieser Parameter wird benutzt, um den neuen Skalierungsfaktor zu ermitteln. Dazu wird die momentane Skalierung des Netzes über dessen Attribut `snp:scale` ausgelesen und um den Wert des Parameters inkrementiert. Die globale Variable `net` repräsentiert das SVG-Element, dass die gesamte Struktur des Netzes beinhaltet.

Die aktuelle Verschiebung des Netzes, das im Attribut `snp:translate` steht, muss dann beim Setzen des Attributs `transform` berücksichtigt werden. Die Neubelegung des Attributs `transform` wirkt sich sofort auf die Anzeige des Netzes im Browser aus. Am Ende der Funktion wird der neu berechnete Skalierungsfaktor (`scale`) als momentaner Skalierungsfaktor im Attribut `snp:scale` verankert.

Entsprechend verhält es sich mit der Funktion zur Verkleinerung eines Petrinetzes:

```
    function zoomOut(value){
      var scale = Number(net.getAttribute(,snp:scale‘)) - value;
      var translate = net.getAttribute(,snp:translate‘);
      net.setAttribute(,transform‘, ,scale(,+scale+‘) translate(,+translate+‘)
‘);
      net.setAttribute(,snp:scale‘, scale);
    }
```

Der Unterschied zur vorhergehenden Funktion liegt in der Dekrementierung des neuen Skalierungsfaktors um den Wert des Funktionsparameters `value`.

Um die ursprüngliche Skalierung eines Netzes wieder herzustellen, wird folgende Funktion aufgerufen:

```
    function zoomReset(){
      var scale = 1;
      var translate = net.getAttribute(,snp:translate‘);
      net.setAttribute(,transform‘, ,scale(,+scale+‘) translate(,+translate+‘)
‘);
      net.setAttribute(,snp:scale‘, scale);
    }
```

Die momentane Skalierung wird auf den Wert 1 gesetzt.

4.2.2 Verschieben

Das Verschieben wird über die Interaktion des Anwenders mit der Maus realisiert. Bevor der Anwender das Netz bewegen kann, muss er einen Button zum Aktivieren des Verschiebens drücken. Folgende Funktion wird dabei aufgerufen:

```
    function setNetMovability(activated){
      if(activated){
        net.addEventListener( „mousedown“, netMouseDown, false );
        net.addEventListener( „mouseup“, netMouseUp, false );
        net.addEventListener( „mousemove“, netMove, false );
        document.getElementById(,translatePanel‘).setAttribute(,opacity‘, ,1‘);
      } else {
        net.removeEventListener( „mousedown“, netMouseDown, false );
        net.removeEventListener( „mouseup“, netMouseUp, false );
        net.removeEventListener( „mousemove“, netMove, false );
        document.getElementById(,translatePanel‘).setAttribute(,opacity‘,
,0.3‘);
      }
      movable = activated;
    }
```

Im Falle, dass der Funktionsparameter gleich dem booleschen Wert `true` ist, werden Event-Listener zur globalen Variable `net` hinzugefügt. Jeder Event-Listener bekommt ein Ereignis und

den Namen einer Event-Handler-Funktion zugeordnet. Sollte der Parameter aber gleich `false` sein, dann werden die Event-Listener wieder entfernt.

Die global gültige Variable `movable` nimmt am Ende der Funktion immer den Wert von `activated` an und bestimmt, ob das Netz verschiebbar ist oder nicht.

4.2.3 GUI

Die GUI-Komponenten zur Steuerung der Skalierung und der Verschiebung des Netzes werden durch Icons repräsentiert:

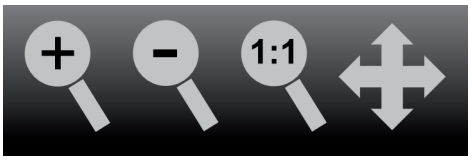


Abb. 17
Icons als GUI-Komponenten zur Steuerung des Skalierens und des Verschiebens eines Petrinetzes im Browser

Die Funktionen des Skalierens werden über die ersten drei Icons von links angesprochen:

1. Das Einzoomen
2. Das Auszoomen
3. Das Wiederherstellen der Originalgröße

Das rechte Icon wird bei Aktivierung durch einen Mausklick rot eingefärbt. Dies symbolisiert dem Anwender, dass das Netz verschoben werden kann. Dazu muss er mit der linken Maustaste auf das Netz klicken. Durch Ziehen der Maus bei gedrückter linker Maustaste verschiebt er das Netz in Richtung der Mausbewegung. Läßt er die Maus los, bestimmt die aktuelle Mausposition die neuen Koordinaten des Netzes. Das Schalten von Transitionen ist dabei solange deaktiviert, bis der Anwender abermals auf das Verschieben-Icon klickt.

4.3 Hierarchie

Um hierarchische Strukturen in Netzen abzubilden, werden diese Strukturen bei der Transformation in Container-Elemente gepackt. Das Anzeigen und Verbergen einzelner Subnetze der Hierarchie gestaltet sich einfacher, da sich die Sichtbarkeit des Container-Elements (`opacity`) auf alle untergeordneten Elemente auswirkt. Die untergeordneten Elemente bekommen zusätzlich noch das Attribut `snp:net`, um sie eindeutig zu einem Subnetz zuzuordnen.

4.3.1 GUI

Die Auswahlliste zur Navigation hierarchischer Petrinetze ist ähnlich zu der Navigation in *Snoopy* gestaltet:

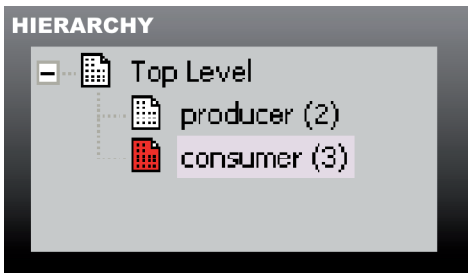


Abb. 18
Auswahlliste zur Navigation und Anzeige eines hierarchischen Petrinetzes

Durch die Abbildung der Hierarchien als Baum-Struktur kann der Anwender durch Auf- und Zuklappen der Subnetze (Äste) zu den untersten Subnetzen (Blättern) gelangen. Durch Doppelklick auf den Namen eines Subnetzes in der Liste wird dieses Subnetz ausgewählt und sichtbar gemacht. Gleichzeitig wird das vorher angezeigte Netz verborgen.

4.4 Animation

Visualisierung von Bewegungen in Javascript ist mit viel Aufwand zu implementieren, da es keine vorgefertigten Komponenten für Animationen gibt. Es bestehen aber bereits Script-Libraries, die Bewegungsabläufe von Elementen unterstützen. Sie können mit einem `script`-Element in SVG-Dokumente eingebunden werden.

Für die Abläufe der Animation in einem Petrinetz wird die frei verfügbare Javascript-Bibliothek *Animator.js* verwendet [Sum1]. Da die Bibliothek ursprünglich für HTML-Dokumente geschrieben wurde, war eine geringe Modifizierung ihres Quelltextes für die Anwendung in SVG notwendig.

Noch bevor die Funktionen der Bibliothek zum Tragen kommen, überprüft nachfolgender Algorithmus, ob eine Transition feuern kann:

```
function fire(transitionId) {
  var transition = $(transitionId);
  if(transition.getAttribute('class')=='Coarse Transition') {
    var netId = transition.getAttribute('snp:coarse');
    var transitions = getElementsByAttribute($('net'+netId), 'g', 'class', 'Transition');
    for(var i=0;i<transitions.length;i++){
      fire(transitions[i].getAttribute('id'));
    }
  } else {
    var items = transition.getElementsByTagName('rect');
    if(transition.getAttribute('snp:firing')!='0') {
      return false;
    } else {
      for(var i=0;i<items.length;i++){
        var outputs = getEdges(items.item(i).getAttribute('id'), 'snp:source');
        var inputs = getEdges(items.item(i).getAttribute('id'), 'snp:target');
        if(inputs.length>0){
          for(var j=0;j<inputs.length;j++){
            var place = document.getElementById(inputs[j].getAttribute('snp:source'));
            var multiplicity = inputs[j].parentNode.getAttribute('snp:multiplicity');
            var marking = place.parentNode.getAttribute('snp:marking');
            if(marking<multiplicity || marking==0) {
              return false;
            }
          }
          animatePreStep(items.item(i).getAttribute('id'), inputs, outputs);
        } else {
          animatePostStep(items.item(i).getAttribute('id'), inputs, outputs);
        }
      }
    }
  }
}
```

Der Algorithmus der Funktion prüft, ob eine normale oder eine hierarchische Transition feuern soll. Bei einer hierarchischen Struktur der Transition wird versucht, deren untergeordneten Transitionen durch einen rekursiven Aufruf zum Schalten zu bringen.

Bei einer normalen Transition wird zuerst über ein für diesen Zweck definiertes Attribut (`snp:firing`) abgefragt, ob eine Transition bereits feuert. Ist dies der Fall so wird die Funktion mit dem Rückgabewert `false` verlassen. Dadurch wird verhindert, dass der Anwender durch wiederholtes Klicken auf eine Transition mehrere Animationsabläufe in einem Schritt initialisiert.

In einem Petrinetz können auch logische Transitionen vorkommen. In der Syntax des SPPED-Formats werden diese Transitionen durch grafische Kopien definiert. Diese Kopien werden im SVG-Dokument durch ein Container-Element (`<g>`) gruppiert. Für die korrekte Darstellung und das richtige Schaltverhalten müssen alle Kopien einer logischen Transition berücksichtigt werden.

Sollte eine Transition gerade nicht feuern, werden ihre Kanten in die Variablen `inputs` und `outputs` aufgeteilt. Die Unterteilung bezieht sich auf die Unterscheidung zwischen den gerichteten Kanten. Die Kanten, die von einer Vorbedingung auf die Transition zeigen sind Eingänge (`inputs`) und die Kanten, die von der Transition zu einer Nachbedingung führen sind die Ausgänge (`outputs`).

Dementsprechend gibt es zwei Funktionen die das Schalten einer spezifischen Transition animieren:

- `animatePreStep`

  Wird ausgeführt, wenn die Anzahl der Eingänge größer Null und die Kantengewichtung (`snp:multiplicity`) größer oder gleich der Markierung (`snp:marking`) der Vorbedingung ist.

- `animatePostStep`

  Wenn es keine Vorbedingungen gibt ist die Transition schaltfähig. Die Animation der Token von der Transition zu eventuellen Nachbedingungen kann sofort aufgerufen werden.

Die eigentliche Animation wird in beiden Funktionen durch jeweils eine Instanz der Animator-Klasse ausgeführt.

Die Animator-Instanz in `animatePreStep` bewegt die Marken von den Vorbedingungen in Richtung der Transition. Nach dem Beenden dieser Animation wird die Funktion `animatePostStep` aufgerufen. Für jeden nachfolgenden Platz der Transition wird ein Token generiert und dann entlang der jeweiligen Kante in Richtung Nachbedingung animiert.

4.4.1 Optionen

Die in *Snoopy* einstellbaren Eigenschaften *Duration* und *Stepping* sollen auch für den Ablauf der Animation in SVG gelten.

Der Wert für die Schrittdauer (*Duration*) wird dem Animator-Objekt als Parameter übergeben. Beim Laden des SVG-Dokuments im Browser wird die Schrittdauer standardmäßig auf `2000` gesetzt. Dadurch ist selbst bei größeren Netzen eine flüssige Animation gewährleistet.

Die verschiedenen Regeln für das Schalten der Transitionen beim Abspielen – *Maximum, Intermediate* und *Single* – sind in Abschnitt 3.1 dargelegt. Diese Schaltregeln (*Stepping*) werden mit Javascript umgesetzt. Die benötigten Zufallswerte liefert die bestehende Funktion `random()` des Javascript-Objekts `Math`.

Das Einfärben der an einer Animation beteiligten Netzstrukturen findet während der Ausführung der Animation der Token statt. Dieses Verhalten kann vom Anwender über eine Check-Box (*Colorize*) deaktiviert werden.

Das Ändern dieser Optionen und das Steuern der Animation erfolgen über die Interaktion des Anwenders mit entsprechenden GUI-Komponenten.

4.4.3 GUI

Die Steuerung für die Animation in einem Petrinetz erschließt sich dem Anwender durch 3 Icons:

Abb. 19
Icons als GUI-Komponenten zur Steuerung der Animation eines Petrinetzes im Browser

Der grafische Entwurf und die Anordnung dieser Icons lehnt sich stark an die Steuerelemente in *Snoopy* an:

- Stop-Button zum Beenden der Animation
- Play-Forward-Button zum Starten der Animation
- Single-Step-Forward-Button für schrittweise Animation

Die Steuerelemente für die Optionen:

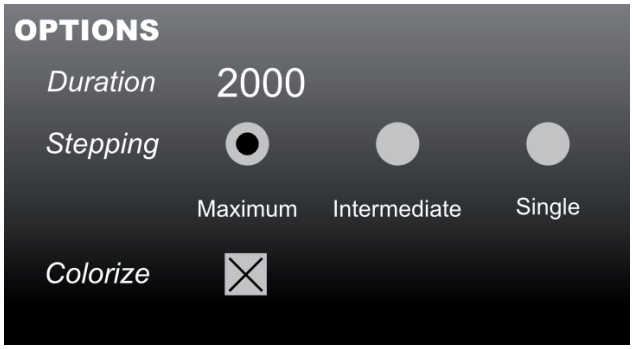


Abb. 20
Fenster zum Einstellen der Optionen einer Petrinetz-Animation im Browser

Die Schaltregeln werden über Radio-Buttons ausgewählt. Dadurch kann nur eine Schaltregel aktiv sein.

Um die Schrittdauer einzugeben, muss der Anwender auf den angezeigten Wert klicken und ein Javascript-Eingabe-Dialog erscheint. Der Anwender hat die Möglichkeit den Dialog abzubrechen, dann bleibt der alte Wert bestehen. Oder er gibt einen neuen Wert für die Schrittdauer ein und bestätigt mit dem OK-Button. Dieser Wert wird dem Anwender dann in der GUI-Komponente der Optionen angezeigt.

Zusammenfassung

Kapitel 5

Mit dieser Arbeit ist eine Möglichkeit zur Transformation von Petrinetzen im SPPED-Format in die entsprechende Vektorgrafik im SVG-Format geschaffen worden. Das Ergebnis der Transformation, die SVG-Datei, ist plattformübergreifend. Da das Format und weitere XML-Techniken auf offenen Standards des W3C beruhen, gibt es viele freie Werkzeuge zur Darstellung und Bearbeitung von SVG.

Die generierte SVG-Datei soll vor allem in einem Webbrowser zum Einsatz kommen. Aus diesem Grund sind Steuerelemente für das dynamische Skalieren und Verschieben mit Javascript umgesetzt worden. Damit sich das Navigieren durch hierarchische Petrinetze einfach gestaltet, wird dem Anwender eine Auswahl-Liste bereitgestellt.

Der Markenfluß eines Petrinetzes kann im Browser animiert werden. Dabei können alle an einer Animation beteiligten Netzstrukturen – also Kanten, Transitionen und Plätze – eingefärbt werden. Zum Steuern der Animation sind entsprechende GUI-Komponenten implementiert worden.

Leider konnte das Aufnehmen und Abspielen von Clips in *Snoopy* nicht umgesetzt werden. Dies führt dazu, dass auf die Forderungen nach dem Verwalten, Speichern und Laden von Clips in dieser Arbeit nicht mehr eingegangen werden konnte.

5.1 Ausblick

Diese Arbeit erfüllt zwar nicht alle gestellten Aufgaben, sie bildet aber die Grundlage für zukünftige Erweiterungen in *Snoopy*. Mit etwas Zeitaufwand und den erarbeiteten Entwürfen aus Kapitel 3 lassen sich die fehlenden Aufgaben implementieren.

Der Export von Petrinetzen aus Snoopy heraus in das geforderte Webformat würde sich am schnellsten umsetzen lassen. Dazu müßte beim Speichern der SPPED-Dateien nur eine Zeile vor dem Root-Element des Dokuments eingefügt werden [W3C2]:

```
<?xml-stylesheet href="spped2svg.xsl" type="text/xsl"?>
```

Die neueren Webbrowser können diese Zeile interpretieren und startet automatisch, den integrierten XSLT-Prozessor. Das `href`-Attribut bezeichnet die XSL-Datei, die die Beschreibung für die Transformation beinhaltet. Diese Datei kann auch frei erreichbar auf einem Webserver hinterlegt werden. Durch Angabe der URL zu der XSL-Datei kann jeder das Stylesheet in eine lokale SPPED-Datei einbinden. Durch eine zentrale Verwaltung des Stylesheets, zum Beispiel auf den Webseiten des Lehrstuhls für Datenstrukturen und Softwarezuverlässigkeit sind die verknüpften SPPED-Dateien immer auf dem neuesten Entwicklungsstand. Die Datei-Erweiterung `spped` muss noch in `xml` geändert werden, damit ein Browser die Transformation ausführt. Diese Änderung des SPPED-Formats würde das manuelle Transformieren mit einem Werkzeug ersparen.

Auf Grundlage des Prototyps zur Umwandlung von SPPED-Dateien ist es möglich für jeden Graphen, der in Snoopy modellierbar ist, eine gleichwerige Darstellung im SVG-Format zu generieren. Die Erstellungen von adäquaten Stylesheets könnte Thema für neue Arbeiten sein.

Anhang

Anhang A – spclip.xsd

Dieser Anhang beinhaltet die XSD-Datei, die das XML-Format für das externe Speichern der Clips definiert.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Snoopy">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="reference" type="xs:string">
          <xs:annotation>
            <xs:documentation>the referenced file</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="properties">
          <xs:annotation>
            <xs:documentation>the animation properties</xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="refresh" type="xs:unsignedInt" />
              <xs:element name="duration" type="xs:unsignedInt" />
              <xs:element name="stepping">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:enumeration value="Minimum" />
                    <xs:enumeration value="Intermediate" />
                    <xs:enumeration value="Maximum" />
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="clips">
          <xs:complexType>
            <xs:sequence>
              <xs:element minOccurs="0" maxOccurs="unbounded" name="clip">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="name" type="xs:string" />
                    <xs:element name="initialmarking">
                      <xs:annotation>
                        <xs:documentation>initial marking of the places</xs:documentation>
                      </xs:annotation>
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element minOccurs="0" maxOccurs="unbounded" name="place">
                            <xs:complexType>
                              <xs:simpleContent>
                                <xs:extension base="xs:unsignedInt">
                                  <xs:attribute name="id" type="xs:unsignedInt" use="required" />
                                </xs:extension>
                              </xs:simpleContent>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="steps">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element minOccurs="0" maxOccurs="unbounded" name="step">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element maxOccurs="unbounded" name="transition">
                                  <xs:complexType>
                                    <xs:simpleContent>
                                      <xs:extension base="xs:anySimpleType">
                                        <xs:attribute name="id" type="xs:unsignedInt" use="required" />
                                      </xs:extension>
                                    </xs:simpleContent>
                                  </xs:complexType>
                                </xs:element>
                              </xs:sequence>
                            </xs:complexType>
                          </xs:element>
```

```
74:                          </xs:sequence>
75:                        </xs:complexType>
76:                      </xs:element>
77:                    </xs:sequence>
78:                  </xs:complexType>
79:                </xs:element>
80:              </xs:sequence>
81:            </xs:complexType>
82:          </xs:element>
83:        </xs:sequence>
84:        <xs:attribute name="version" type="xs:string" use="required" />
85:        <xs:attribute name="revision" type="xs:string" use="required" />
86:      </xs:complexType>
87:    </xs:element>
88:  </xs:schema>
```

Anhang B – spped2svg.xsl

Das Stylesheet, dass die Transformation von SPPED zu SVG beschreibt.

```
1:    <?xml version=“1.0“ encoding=“UTF-8“?>
2:    <xsl:stylesheet xmlns:xsl=“http://www.w3.org/1999/XSL/Transform“ xmlns=“http://www.w3.org/2000/svg“ xmlns:xlink=“http://www.w3.org/1999/xlink“ xmlns:ev=“http://www.w3.org/2001/xml-events“ xmlns:snp=“http://www-dssz.informatik.tu-cottbus.de/“ version=“1.0“>
3:      <xsl:output doctype-system=“http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd“ doctype-public=“-//W3C//DTD SVG 1.1//EN“ indent=“no“ standalone=“no“ cdata-section-elements=“script text“/>
4:      <xsl:template match=“Snoopy“>
5:        <svg xmlns=“http://www.w3.org/2000/svg“ xmlns:xlink=“http://www.w3.org/1999/xlink“ width=“100%“ height=“100%“ version=“1.1“ onload=“init()“>
6:        <title>test</title>
7:          <defs>
8:            <linearGradient id=“titleBarGradient“ x1=“0%“ y1=“0%“ x2=“0%“ y2=“100%“>
9:          <stop offset=“40%“ style=“stop-color:rgb(128,128,128);stop-opacity:1“/>
10:         <stop offset=“60%“ style=“stop-color:rgb(50,50,50);stop-opacity:1“/>
11:       </linearGradient>
12:       <marker id=“Triangle“ viewBox=“0 0 10 10“ refX=“10“ refY=“5“ markerUnits=“strokeWidth“ markerWidth=“10“ markerHeight=“9“ orient=“auto“>
13:         <path d=“M 0 0 L 10 5 L 0 10 z“/>
14:       </marker>
15:       <symbol id=“play“>
16:         <path d=“M 0,0 L 10,5 L 0,10 z“/>
17:       </symbol>
18:       <symbol id=“stop“>
19:         <path d=“M 0,0 L 10,0 L 10,10 L 0,10 z“/>
20:       </symbol>
21:       <symbol id=“pause“>
22:         <path d=“M 0,0 L 4,0 L 4,10 L 0,10 z M 6,0 L 10,0 L 10,10 L 6,10 z“/>
23:       </symbol>
24:       <script type=“text/javascript“ xlink:href=“animator.js“/>
25:       <script type=“text/javascript“><![CDATA[
26:
27:
28:         var duration, delta, starttime, aktiv;
29:         var main, net, control, netAlias, controlAlias;
30:         var clipList, controlBox;
31:
32:         var activeTransitions = new Array();
33:
34:         var movable = false;
35:
36:         var net_down = false;
37:         var control_down = false;
38:         var net_lastX, net_lastY;
39:         var net_tx = 0, net_ty = 0;
40:         var control_tx = 500, control_ty = 50;
41:
42:         function $(id) {
43:                   return document.getElementById(id);
44:         }
45:
46:         function setNetMovability(activated){
47:           if(activated){
48:             net.addEventListener( „mousedown“, netMouseDown, false );
49:             net.addEventListener( „mouseup“, netMouseUp, false );
50:             net.addEventListener( „mousemove“, netMove, false );
51:             document.getElementById(,translatePanel‘).setAttribute(,opacity‘, ,1‘);
52:           } else {
53:             net.removeEventListener( „mousedown“, netMouseDown, false );
54:             net.removeEventListener( „mouseup“, netMouseUp, false );
55:             net.removeEventListener( „mousemove“, netMove, false );
56:             document.getElementById(,translatePanel‘).setAttribute(,opacity‘, ,0.3‘);
57:           }
58:           movable = activated;
59:         }
60:
61:         function netMouseDown(evt){
62:           net_lastX = evt.clientX;
63:           net_lastY = evt.clientY;
64:           net_down = true;
65:           netAlias = document.createElementNS(„http://www.w3.org/2000/svg“,“rect“);
66:           netAlias.setAttribute(,id‘, ,netAlias‘);
67:           netAlias.setAttribute(,x‘, 0);
68:           netAlias.setAttribute(,y‘, 0);
69:           netAlias.setAttribute(,width‘, ,100%‘);
```

```
70:           netAlias.setAttribute(‚height‘, ‚100%‘);
71:           netAlias.setAttribute(‚stroke‘, ‚rgb(0,0,0)‘);
72:           netAlias.setAttribute(‚fill‘, ‚rgb(255,255,255)‘);
73:           netAlias.setAttribute(‚opacity‘, ‚0‘);
74:           netAlias.setAttribute(‚style‘, ‚cursor:move;‘);
75:
76:           net.appendChild(netAlias);
77:           net.setAttribute(‚opacity‘, ‚0.5‘);
78:         }
79:
80:         function netMouseUp(evt){
81:           net_down = false;
82:           net.setAttribute(‚opacity‘, ‚1‘);
83:           net.removeChild(netAlias);
84:           net.setAttribute(‚snp:translate‘, net_tx+‘, ‚+net_ty);
85:         }
86:
87:         function netMove(evt){
88:           if (!net_down) return;
89:           var x = evt.clientX;
90:           var y = evt.clientY;
91:           var dx = x - net_lastX;
92:           var dy = y - net_lastY;
93:           net_lastX = x;
94:           net_lastY = y;
95:           net_tx += dx;
96:           net_ty += dy;
97:           var scale = net.getAttribute(‚snp:scale‘);
98:           net.setAttribute(„transform“, „translate(„+net_tx+“,“+net_ty+“) scale(„+scale+“)“);
99:         }
100:
101:        function trim(string){
102:          return string.replace(/^\s+|\s+$/g, ‚‘);
103:        }
104:
105:        function Point(x, y){
106:          this.x = x;
107:          this.y = y;
108:          this.setPoint = function(x, y){
109:            this.x = x;
110:            this.y = y;
111:          }
112:          this.getX = function(){
113:            return this.x;
114:          }
115:          this.getY = function(){
116:            return this.y;
117:          }
118:          this.getPoint = function(){
119:            return this;
120:          }
121:        }
122:
123:        function Box(id, title, x, y, width, height){
124:          this.id = id;
125:          this.title = title;
126:          this.x = x;
127:          this.y = y;
128:          this.width = width;
129:          this.height = height;
130:          this.dragged = false;
131:          this.element = document.createElementNS(„http://www.w3.org/2000/svg“,“g“);
132:          this.content = document.createElementNS(„http://www.w3.org/2000/svg“,“g“);
133:
134:          this.mouseMove = function(evt){
135:            if (!control_down) return;
136:            var x = evt.clientX;
137:            var y = evt.clientY;
138:            var dx = x - control_lastX;
139:            var dy = y - control_lastY;
140:            control_lastX = x;
141:            control_lastY = y;
142:            control_tx += dx;
143:            control_ty += dy;
144:            control.setAttribute(„transform“, „translate(„+control_tx+“,“+control_ty+“)“);
145:          }
146:
147:          this.mouseUp = function(evt){
148:            control_down = false;
149:            control.setAttribute(‚opacity‘, ‚1‘);
150:            control.setAttribute(‚snp:translate‘, control_tx+‘, ‚+control_ty);
151:            //control.removeChild(controlAlias);
152:          }
153:
```

```
154:          this.mouseDown = function(evt){
155:            control_lastX = evt.clientX;
156:            control_lastY = evt.clientY;
157:            control_down = true;
158:
159:            control.setAttribute(,opacity‘, ,0.5‘);
160:            controlAlias = document.createElementNS(„http://www.w3.org/2000/svg“,“rect“);
161:            controlAlias.setAttribute(,id‘, ,netAlias‘);
162:            controlAlias.setAttribute(,x‘, 0);
163:            controlAlias.setAttribute(,y‘, 0);
164:            controlAlias.setAttribute(,width‘, ,100%‘);
165:            controlAlias.setAttribute(,height‘, ,100%‘);
166:            controlAlias.setAttribute(,stroke‘, ,rgb(0,0,0)‘);
167:            controlAlias.setAttribute(,fill‘, ,rgb(255,255,255)‘);
168:            controlAlias.setAttribute(,opacity‘, ,0‘);
169:            controlAlias.setAttribute(,style‘, ,cursor:move;‘);
170:
171:            //control.appendChild(controlAlias);
172:          }
173:
174:          this.add = function(element){
175:            this.content.appendChild(element);
176:          }
177:
178:          this.createElement = function(){
179:            this.element.setAttribute(,id‘, this.id);
180:            this.element.setAttribute(,transform‘, ,translate(,+this.x+‘, ,+this.y+‘)‘);
181:            this.element.setAttribute(,snp:translate‘, this.x+‘,‘+this.y);
182:
183:            var frame = document.createElementNS(„http://www.w3.org/2000/svg“,“rect“);
184:            frame.setAttribute(,width‘, this.width);
185:            frame.setAttribute(,height‘, this.height);
186:            frame.setAttribute(,stroke‘, ,rgb(0,0,0)‘);
187:            frame.setAttribute(,fill‘, ,rgb(255,255,255)‘);
188:            frame.setAttribute(,fill-opacity‘, ,0.5‘);
189:
190:            var titleBar = document.createElementNS(„http://www.w3.org/2000/svg“,“rect“);
191:            titleBar.setAttribute(,width‘, this.width);
192:            titleBar.setAttribute(,height‘, 20);
193:            titleBar.setAttribute(,stroke‘, ,rgb(0,0,0)‘);
194:            titleBar.setAttribute(,fill‘, ,rgb(0,0,0)‘);
195:            titleBar.setAttribute(,style‘, ,fill:url(#titleBarGradient);cursor:move;‘);
196:            titleBar.addEventListener( „mousedown“, this.mouseDown, false );
197:            titleBar.addEventListener( „mouseup“, this.mouseUp, false );
198:            titleBar.addEventListener( „mousemove“, this.mouseMove, false );
199:
200:            var title = document.createElementNS(„http://www.w3.org/2000/svg“,“text“);
201:            title.setAttribute(,x‘, 5);
202:            title.setAttribute(,y‘, 15);
203:            title.setAttribute(,fill‘, ,rgb(255,255,255)‘);
204:            title.appendChild(document.createTextNode(this.title));
205:
206:            this.element.appendChild(frame);
207:            this.element.appendChild(titleBar);
208:            this.element.appendChild(title);
209:
210:            this.content.setAttribute(,transform‘, ,translate(0,20)‘);
211:
212:            this.element.appendChild(this.content);
213:
214:            return this.element;
215:          }
216:
217:        }
218:
219:        function Timer(code, duration) {
220:            this.duration = duration;
221:            this.stopped = true;
222:            this.started = false;
223:            this.time = 0;
224:            this.interval = null;
225:            this.code = code;
226:            this.start = function(){
227:              this.stopped = false;
228:              this.started = true;
229:              starttime = new Date().getTime();
230:              this.interval = window.setInterval(„“+this.code+““, duration);
231:            };
232:            this.stop = function(){
233:              window.clearInterval(this.interval);
234:              this.stopped = true;
235:              this.started = false;
236:            };
237:            this.reset = function(){
238:              this.time = 0;
239:              this.stop();
```

Anhang

```
240:            }
241:        }
242:
243:        function dropDown(evt){
244:          if(clipList.expanded){
245:            clipList.box.setAttribute(,height‘, ,20‘);
246:            clipList.expanded = false;
247:          } else {
248:            clipList.box.setAttribute(,height‘, ,100‘)
249:            clipList.expanded = true;
250:          }
251:        }
252:
253:        function itemOver(index){
254:              clipList.ul.childNodes.item(index).setAttribute(,fill‘, ,rgb(255,255,255)‘);
255:              clipList.ul.childNodes.item(index).childNodes.item(0).setAttribute(,opacity‘, ,1‘);
256:        }
257:
258:        function itemOut(index){
259:              clipList.ul.childNodes.item(index).setAttribute(,fill‘, ,rgb(0,0,0)‘);
260:              clipList.ul.childNodes.item(index).childNodes.item(0).setAttribute(,opacity‘, ,0‘);
261:        }
262:
263:        function ListBox(x, y, items){
264:          this.x = x;
265:          this.y = y;
266:          this.items = items;
267:          this.currentItem = items[0];
268:          this.expanded = false;
269:
270:          this.box = document.createElementNS(„http://www.w3.org/2000/svg“,“rect“);
271:          this.button = document.createElementNS(„http://www.w3.org/2000/svg“,“rect“);
272:
273:          this.ul = document.createElementNS(„http://www.w3.org/2000/svg“,“g“);
274:
275:          this.createListItem = function(index, content){
276:
277:            var rect = document.createElementNS(„http://www.w3.org/2000/svg“,“rect“);
278:            rect.setAttribute(,x‘, 0);
279:            rect.setAttribute(,y‘, 20 * index);
280:            rect.setAttribute(,width‘, 280);
281:            rect.setAttribute(,height‘, 20);
282:            rect.setAttribute(,fill‘, ,rgb(0,0,0)‘);
283:            rect.setAttribute(,opacity‘, ,0‘);
284:            rect.setAttribute(,onmouseover‘, ,itemOver(,+index+‘)‘);
285:            rect.setAttribute(,onmouseout‘, ,itemOut(,+index+‘)‘);
286:
287:            var text = document.createElementNS(„http://www.w3.org/2000/svg“,“text“);
288:            text.setAttribute(,x‘, 5);
289:            text.setAttribute(,y‘, 20 * (index + 1) - 5);
290:
291:            var textNode = document.createTextNode(content);
292:            text.appendChild(textNode);
293:
294:            var li = document.createElementNS(„http://www.w3.org/2000/svg“,“g“);
295:            li.appendChild(rect);
296:            li.appendChild(text);
297:
298:            return li;
299:          }
300:
301:          this.getElement = function(){
302:            var element = document.createElementNS(„http://www.w3.org/2000/svg“,“g“);
303:            element.setAttribute(,id‘, ,clipList‘);
304:            element.setAttribute(,transform‘, ,translate(,+this.x+‘,‘+this.y+‘)‘);
305:
306:            this.box.setAttribute(,fill‘, ,rgb(255,255,255)‘);
307:            this.box.setAttribute(,stroke‘, ,rgb(0,0,0)‘);
308:            this.box.setAttribute(,width‘, 280);
309:            this.box.setAttribute(,height‘, this.items.length * 20);
310:
311:            this.button.setAttribute(,fill‘, ,rgb(0,0,0)‘);
312:            this.button.setAttribute(,stroke‘, ,rgb(0,0,0)‘);
313:            this.button.setAttribute(,width‘, ,20‘);
314:            this.button.setAttribute(,height‘, ,20‘);
315:            this.button.setAttribute(,x‘, ,180‘);
316:            this.button.setAttribute(,y‘, ,0‘);
317:            this.button.addEventListener(„click“, dropDown, false );
318:
319:            element.appendChild(this.box);
320:
321:            // if(this.expanded){
322:              for(var i = 0; i < this.items.length; i++){
323:                var li = this.createListItem(i, this.items[i].name);
```

```
324:                this.ul.appendChild(li);
325:                element.appendChild(this.ul);
326:              }
327:              /*
328:            } else {
 329:
330:            }
331:            */
332:
333:            //element.appendChild(this.button);
 334:
335:            return element;
336:          }
337:        }
338:
339:        function Clip(name) {
340:          this.name = name;
341:        }
342:
343:        function Token(x ,y, r, fill, stroke) {
344:          var date = new Date();
345:          this.id = date.getTime();
346:          this.element = document.createElementNS(„http://www.w3.org/2000/svg“,“circle“);
347:          this.element.setAttribute(,id‘, this.id);
348:          this.element.setAttribute(,class‘, ,Token‘);
349:          this.from = new Point(x, y);
350:          this.now = this.from;
351:          this.to = this.from;
352:          this.x = x;
353:          this.y = y;
354:          this.element.setAttribute(,cx‘, this.x);
355:          this.element.setAttribute(,cy‘, this.y);
356:          this.element.setAttribute(,fill‘, fill);
357:          this.element.setAttribute(,stroke‘, stroke);
358:          this.element.setAttribute(,r‘, r);
359:        }
360:
361:        function getEdges(transitionId, attribute){
362:          var edges = [];
 363:
364:          var items = document.getElementsByTagName(,polyline‘);
365:          for(var i=0;i<items.length;i++){
366:            if(items.item(i).getAttribute(attribute)==transitionId){
367:              edges.push(items.item(i));
368:            }
369:          }
370:          //alert(transitionId+‘ : ,+attribute+‘ = ,+edges.length);
371:          return edges;
372:        }
373:
374:        function getElementsByAttribute(parentNode, tagName, attributeName, attributeValue){
375:          var resultElements = [];
376:          var items = null;
377:          if(parentNode!=null) items = parentNode.getElementsByTagName(tagName);
378:          if(items!=null) {
379:            for(var i=0;i<items.length;i++){
380:              if(items.item(i).getAttribute(attributeName)==attributeValue){
381:                resultElements.push(items.item(i));
382:              }
383:            }
384:          }
385:          return resultElements;
386:        }
387:
388:        function getTokenText(placeId){
389:          var place = document.getElementById(placeId);
390:          var textElements = place.getElementsByTagName(,text‘);
391:          for(var i=0;i<textElements.length;i++){
392:            if(textElements.item(i).getAttribute(,class‘)==‘Token‘) return textElements.item(i);
393:          }
394:          return null;
395:        }
396:
397:        function getPlaceCircle(placeId){
398:          var place = document.getElementById(placeId);
399:          var circleElements = place.getElementsByTagName(,circle‘);
400:          for(var i=0;i<circleElements.length;i++){
401:            if(circleElements.item(i).getAttribute(,class‘)!=‘Token‘) return circleElements.item(i);
402:          }
403:          return null;
404:        }
405:
406:        function getPlaceCircles(placeId){
407:          var place = document.getElementById(placeId);
408:          var circles = [];
409:          var circleElements = place.getElementsByTagName(,circle‘);
```

Anhang

```
410:          for(var i=0;i<circleElements.length;i++){
411:            if(circleElements.item(i).getAttribute(,class‘)!=‘Token‘){
412:              circles.push(circleElements.item(i));
413:            }
414:          }
415:          return circles;
416:        }
417:
418:        function increaseMarking(placeId, value){
419:          var place = document.getElementById(placeId);
420:          setMarking(placeId, Number(place.getAttribute(,snp:marking‘))+Number(value));
421:        }
422:
423:        function decreaseMarking(placeId, value){
424:          var place = $(placeId);
425:          setMarking(placeId, Number(place.getAttribute(,snp:marking‘))-Number(value));
426:        }
427:
428:        function setMarking(placeId, value){
429:          var place = $(placeId);
430:          var marking = Number(place.getAttribute(,snp:marking‘));
431:
432:          var tokenCircles = getElementsByAttribute(place, ,circle‘, ,class‘, ,Token‘);
433:          var placeCircles = getPlaceCircles(placeId);
434:
435:          var tokenTexts = getElementsByAttribute(place, ,text‘, ,class‘, ,Token‘);
436:          var token = null;
437:          var cx = 0;
438:          var cy = 0;
439:          if(value>3){
440:            if(tokenCircles.length!=0){
441:              for(var i=0;i<tokenCircles.length;i++){
442:                place.removeChild(tokenCircles[i]);
443:              }
444:            }
445:            if(tokenTexts.length==0) {
446:              for(var i=0;i<placeCircles.length;i++){
447:                cx = Number(placeCircles[i].getAttribute(,cx‘));
448:                cy = Number(placeCircles[i].getAttribute(,cy‘));
449:                var textNode = document.createTextNode(value);
450:                var text = document.createElementNS(„http://www.w3.org/2000/svg“,“text“);
451:                text.setAttribute(,class‘, ,Token‘);
452:                text.setAttribute(,x‘, cx-4);
453:                text.setAttribute(,y‘, cy+4);
454:                text.setAttribute(,style‘, ,font-family:sans-serif;font-size:11px;‘);
455:                text.appendChild(textNode);
456:                place.appendChild(text);
457:              }
458:            } else {
459:              for(var i=0;i<tokenTexts.length;i++){
460:                var text = tokenTexts[i];
461:                if(Number(value)==10) text.setAttribute(,x‘, Number(text.getAttribute(,x‘))-3);
462:                if(Number(value)==100) text.setAttribute(,x‘, Number(text.getAttribute(,x‘))-3);
463:                text.firstChild.nodeValue = value;
464:              }
465:            }
466:          }
467:          if(value<4){
468:            if(tokenTexts.length!=0){
469:              for(var i=0;i<tokenTexts.length;i++){
470:               place.removeChild(tokenTexts[i]);
471:              }
472:            }
473:            if(tokenCircles.length!=0){
474:              for(var i=0;i<tokenCircles.length;i++){
475:                place.removeChild(tokenCircles[i]);
476:              }
477:            }
478:          }
479:          if(value==3){
480:            for(var i=0;i<placeCircles.length;i++){
481:              cx = Number(placeCircles[i].getAttribute(,cx‘));
482:              cy = Number(placeCircles[i].getAttribute(,cy‘));
483:              token = new Token(cx, cy-4, 2, ,rgb(0,0,0)‘, ,rgb(0,0,0)‘);
484:              place.appendChild(token.element);
485:              token = new Token(cx+3.5, cy+2, 2, ,rgb(0,0,0)‘, ,rgb(0,0,0)‘);
486:              place.appendChild(token.element);
487:              token = new Token(cx-3.5, cy+2, 2, ,rgb(0,0,0)‘, ,rgb(0,0,0)‘);
488:              place.appendChild(token.element);
489:            }
490:          }
491:          if(value==2){
492:            for(var i=0;i<placeCircles.length;i++){
493:              cx = Number(placeCircles[i].getAttribute(,cx‘));
```

```
494:              cy = Number(placeCircles[i].getAttribute(,cy‘));
495:              token = new Token(cx-3.5, cy, 2, ,rgb(0,0,0)‘, ,rgb(0,0,0)‘);
496:              place.appendChild(token.element);
497:              token = new Token(cx+3.5, cy, 2, ,rgb(0,0,0)‘, ,rgb(0,0,0)‘);
498:              place.appendChild(token.element);
499:            }
500:          }
501:          if(value==1){
502:            for(var i=0;i<placeCircles.length;i++){
503:              cx = Number(placeCircles[i].getAttribute(,cx‘));
504:              cy = Number(placeCircles[i].getAttribute(,cy‘));
505:              token = new Token(cx, cy, 2, ,rgb(0,0,0)‘, ,rgb(0,0,0)‘);
506:              place.appendChild(token.element);
507:            }
508:          }
509:          place.setAttribute(,snp:marking‘, value);
510:        }
511:
512:        function fire(transitionId) {
513:          var transition = $(transitionId);
514:
515:          if(transition.getAttribute(,class‘)==‘Coarse Transition‘) {
516:            var netId = transition.getAttribute(,snp:coarse‘);
517:            var transitions = getElementsByAttribute($(,net‘+netId), ,g‘, ,class‘, ,Transition‘);
518:            for(var i=0;i<transitions.length;i++){
519:              fire(transitions[i].getAttribute(,id‘));
520:            }
521:          } else {
522:            var items = transition.getElementsByTagName(,rect‘);
523:            if(transition.getAttribute(,snp:firing‘)!=‘0‘) {
524:              return false;
525:            } else {
526:              for(var i=0;i<items.length;i++){
527:                var outputs = getEdges(items.item(i).getAttribute(,id‘), ,snp:source‘);
528:                var inputs = getEdges(items.item(i).getAttribute(,id‘), ,snp:target‘);
529:                if(inputs.length>0){
530:                  for(var j=0;j<inputs.length;j++){
531:                    var place = document.getElementById(inputs[j].getAttribute(,snp:source‘));
532:                    var multiplicity = inputs[j].parentNode.getAttribute(,snp:multiplicity‘);
533:                    var marking = place.parentNode.getAttribute(,snp:marking‘);
534:                    if(marking<multiplicity || marking==0) {
535:                      return false;
536:                    }
537:                  }
538:                  animatePreStep(items.item(i).getAttribute(,id‘), inputs, outputs);
539:                } else {
540:                  animatePostStep(items.item(i).getAttribute(,id‘), inputs, outputs);
541:                }
542:              }
543:            }
544:          }
545:        }
546:
547:        function animatePreStep(transitionId, inputs, outputs){
548:          var preAnimator;
549:          var preToken = new Array();
550:          var completeToken = new Array();
551:          var transition = $(transitionId);
552:
553:          transition.parentNode.setAttribute(,snp:firing‘, ,1‘);
554:
555:          for(var i=0;i<inputs.length;i++){
556:            var edgeIn = inputs[i];
557:            var items = edgeIn.parentNode.getElementsByTagName(,polyline‘);
558:
559:            for(var j=0;j<items.length;j++){
560:              var points = trim(String(items.item(j).getAttribute(„points“))).split(„ „);
561:              var startpoint = points[0];
562:              var endpoint = points[1];
563:              var x1 = Number(startpoint.split(„,“)[0]);
564:              var y1 = Number(startpoint.split(„,“)[1]);
565:              var x2 = Number(endpoint.split(„,“)[0]);
566:              var y2 = Number(endpoint.split(„,“)[1]);
567:              var netId = items.item(j).getAttribute(,snp:net‘);
568:
569:              var token = new Token(x1, y1, 4, ,rgb(255,0,0)‘, ,rgb(0,0,0)‘);
570:              token.element.setAttribute(,snp:net‘, netId);
571:
572:              $(,net‘+netId).appendChild(token.element);
573:
574:              var subjectX = new NumericalStyleSubject(token.element, ,cx‘, x1, x2);
575:              var subjectY = new NumericalStyleSubject(token.element, ,cy‘, y1, y2);
576:
577:              preAnimator = new Animator({
578:                duration: duration,
579:                onComplete: function() {
```

Anhang

```
580:                   completeToken.push(„ready“);
581:                   if(completeToken.length==inputs.length){
582:                     for(var k=0;k<preToken.length;k++){
583:                       $(,net‘+preToken[k].element.getAttribute(,snp:net‘)).removeChild(preToken[k].element);
584:                     }
585:                     animatePostStep(transitionId, inputs, outputs);
586:                   }
587:                 }
588:               }).addSubject(subjectX).addSubject(subjectY);
589:               preToken.push(token);
590:               preAnimator.play();
591:             }
592:             var source = edgeIn.parentNode.getAttribute(,snp:source‘);
593:             var multiplicity = edgeIn.parentNode.getAttribute(,snp:multiplicity‘);
594:             decreaseMarking(source, multiplicity);
595:           }
596:         }
597:
598:         function animatePostStep(transitionId, inputs, outputs){
599:           var transition = $(transitionId);
600:           transition.parentNode.setAttribute(,snp:firing‘, ,1‘);
601:           var postAnimator;
602:           var postToken = new Array();
603:           var completeToken = new Array();
604:           var edges = new Array();
605:
606:           if(outputs.length>0){
607:             for(var i=0;i<outputs.length;i++){
608:               var edgeOut = outputs[i];
609:               var items = edgeOut.parentNode.getElementsByTagName(,polyline‘);
610:
611:               for(var j=0;j<items.length;j++){
612:                 var points = trim(String(items.item(j).getAttribute(„points“))).split(„ „);
613:                 var startpoint = points[0];
614:                 var endpoint = points[1];
615:                 var x1 = Number(startpoint.split(„,“)[0]);
616:                 var y1 = Number(startpoint.split(„,“)[1]);
617:                 var x2 = Number(endpoint.split(„,“)[0]);
618:                 var y2 = Number(endpoint.split(„,“)[1]);
619:                 var netId = items.item(j).getAttribute(,snp:net‘);
620:
621:                 var token = new Token(x1, y1, 4, ,rgb(255,0,0)‘, ,rgb(0,0,0)‘);
622:                 token.element.setAttribute(,snp:net‘, netId);
623:
624:                 $(,net‘+netId).appendChild(token.element);
625:
626:                 postAnimator = new Animator({
627:                   duration: duration,
628:                   onComplete: function() {
629:                       transition.parentNode.setAttribute(,snp:firing‘, ,0‘);
630:                       completeToken.push(„ready“);
631:                       if(completeToken.length==postToken.length) {
632:                         for(var l=0;l<postToken.length;l++){
633:                           var multiplicity = edges[l].parentNode.getAttribute(,snp:multiplicity‘);
634:                           $(,net‘+postToken[l].element.getAttribute(,snp:net‘)).removeChild(postToken[l].element);
635:                         }
636:                         for(var k=0;k<outputs.length;k++) {
637:                           increaseMarking(outputs[k].parentNode.getAttribute(,snp:target‘), multiplicity);
638:                         }
639:                       }
640:                   }
641:                 }).addSubject(new NumericalStyleSubject(token.element, ,cx‘, x1, x2)).addSubject(new NumericalStyleSubject(
token.element, ,cy‘, y1, y2));
642:                 postToken.push(token);
643:                 edges.push(items.item(j));
644:                 postAnimator.play();
645:               }
646:             }
647:           }
648:         }
649:
650:         function init(){
651:           duration = 1500;
652:           main = $(,main‘);
653:           net = $(,net‘);
654:           var items = [new Clip(,Clip A‘), new Clip(,Clip B‘), new Clip(,Clip C‘), new Clip(,Clip D‘), new Clip(,Clip E‘),
new Clip(,Clip F‘)];
655:           clipList = new ListBox(10, 70, items);
656:
657:           controlBox = new Box(,control‘, ,Control‘, 500, 50, 300, 400);
658:           controlBox.add($(,translatePanel‘));
659:           controlBox.add($(,zoomPanel‘));
660:           controlBox.add($(,player‘));
661:           controlBox.add(clipList.getElement());
```

```
662:
663:          main.appendChild(controlBox.createElement());
664:          control = document.getElementById(,control‘);
665:
666:        }
667:
668:        function getPosition(id)
669:        {
670:          var x = Number(document.getElementById(id).getAttribute(,x‘));
671:          var y = Number(document.getElementById(id).getAttribute(,y‘));
672:          return [x, y];
673:        }
674:
675:
676:      function zoomIn(value){
677:        var scale = Number(net.getAttribute(,snp:scale‘)) + value;
678:        var translate = net.getAttribute(,snp:translate‘);
679:        net.setAttribute(,transform‘, ,scale(,+scale+‘) translate(,+translate+‘)‘);
680:        net.setAttribute(,snp:scale‘, scale);
681:      }
682:
683:      function zoomReset(){
684:        var scale = 1;
685:        var translate = net.getAttribute(,snp:translate‘);
686:        net.setAttribute(,transform‘, ,scale(,+scale+‘) translate(,+translate+‘)‘);
687:        net.setAttribute(,snp:scale‘, scale);
688:      }
689:
690:      function zoomOut(value){
691:        var scale = Number(net.getAttribute(,snp:scale‘)) - value;
692:        var translate = net.getAttribute(,snp:translate‘);
693:        net.setAttribute(,transform‘, ,scale(,+scale+‘) translate(,+translate+‘)‘);
694:        net.setAttribute(,snp:scale‘, scale);
695:      }
696:
697:
698:      ]]></script>
699:        </defs>
700:        <text id=“0“ x=“10“ y=“800“>test</text>
701:        <g id=“main“>
702:          <g id=“net“ snp:scale=“1“ snp:translate=“0,0“>
703:            <g id=“net0“></g>
704:            <g id=“net1“></g>
705:            <xsl:for-each select=“nodeclasses/nodeclass[@name=‘Coarse Transition‘]/node“>
706:              <xsl:variable name=“coarseTransition“ select=“.“/>
707:              <g id=“net{$coarseTransition/@coarse}“ opacity=“0.3“>
708:                <xsl:apply-templates select=“//node[@net=$coarseTransition/@coarse]“/>
709:                <xsl:apply-templates select=“//edge[@net=$coarseTransition/@coarse]“/>
710:              </g>
711:            </xsl:for-each>
712:            <xsl:for-each select=“nodeclasses/nodeclass[@name=‘Coarse Place‘]/node“>
713:              <xsl:variable name=“coarsePlace“ select=“.“/>
714:              <g id=“net{$coarsePlace/@coarse}“ opacity=“0.3“>
715:                <xsl:apply-templates select=“//node[@net=$coarsePlace/@coarse]“/>
716:                <xsl:apply-templates select=“//edge[@net=$coarsePlace/@coarse]“/>
717:              </g>
718:            </xsl:for-each>
719:            <xsl:apply-templates select=“edgeclasses/edgeclass/edge[@net&lt;2]“/>
720:            <xsl:apply-templates select=“nodeclasses/nodeclass/node[@net&lt;2]“/>
721:          </g>
722:          <g>
723:            <g id=“zoomPanel“ transform=“scale(2) translate(35, 15)“ fill=“rgb(0,0,0)“ style=“cursor:pointer;“>
724:              <rect x=“0“ y=“0“ width=“8“ height=“5“ onclick=“zoomOut()“/>
725:              <rect x=“10“ y=“-5“ width=“8“ height=“10“ onclick=“zoomReset()“/>
726:              <rect x=“20“ y=“-10“ width=“8“ height=“15“ onclick=“zoomIn(0.25)“/>
727:            </g>
728:            <g id=“player“ transform=“scale(2) translate(80, 10)“ style=“cursor:pointer;“>
729:              <use xlink:href=“#play“ transform=“translate(0, 0)“ onclick=““/>
730:              <use xlink:href=“#stop“ transform=“translate(15, 0)“ onclick=““/>
731:              <use xlink:href=“#pause“ transform=“translate(30, 0)“ onclick=““/>
732:            </g>
733:            <g id=“translatePanel“ fill=“rgb(0,0,0)“ opacity=“0.3“ transform=“scale(2) translate(2.5,2.5)“
onclick=“(movable) ? setNetMovability(false) : setNetMovability(true);“ style=“cursor:pointer;“>
734:              <path d=“M 10,5 L 7.5,5 L 12.5,0 L 17.5,5 L 15,5 L 15,20 L 17.5,20 L 12.5,25 L 7.5,20 L 10,20 z“/>
735:              <path d=“M 5,10 L 5,7.5 L 0,12.5 L 5,17.5 L 5,15 L 20,15 L 20,17.5 L 25,12.5 L 20,7.5 L 20,10 z“/>
736:            </g>
737:          </g>
738:        </g>
739:      </svg>
740:    </xsl:template>
741:    <xsl:template match=“nodeclass[@name=‘Place‘]/node | nodeclass[@name=‘Coarse Place‘]/node“>
742:      <xsl:variable name=“marking“>
743:        <xsl:if test=“not(attribute[@name=‘Marking‘])“>
744:          <xsl:value-of select=“0“/>
745:        </xsl:if>
746:        <xsl:if test=“attribute[@name=‘Marking‘]“>
```

```
747:          <xsl:value-of select="normalize-space(attribute[@name='Marking'])"/>
748:        </xsl:if>
749:      </xsl:variable>
750:      <g id="{@id}" class="{../@name}" onclick="increaseMarking({@id}, 1);" snp:marking="{$marking}" snp:net="{@net}">
751:        <xsl:if test="./../@name='Coarse Place'">
752:          <xsl:attribute name="snp:coarse"><xsl:value-of select="@coarse"/></xsl:attribute>
753:        </xsl:if>
754:        <xsl:for-each select="graphics/graphic">
755:          <xsl:variable name="stroke">
756:            <xsl:if test="@pen and not(@net &gt; ../../@net)">
757:              <xsl:value-of select="@pen"/>
758:            </xsl:if>
759:            <xsl:if test="not(@pen) and not(@net &gt; ../../@net)">
760:              <xsl:text>0,0,0</xsl:text>
761:            </xsl:if>
762:            <xsl:if test="@net &gt; ../../@net">
763:              <xsl:text>0,0,255</xsl:text>
764:            </xsl:if>
765:          </xsl:variable>
766:          <xsl:variable name="fill">
767:            <xsl:if test="@brush">
768:              <xsl:value-of select="@brush"/>
769:            </xsl:if>
770:            <xsl:if test="not(@brush) and not(../../attribute[@name='Logic']=1)">
771:              <xsl:text>255,255,255</xsl:text>
772:            </xsl:if>
773:            <xsl:if test="../../attribute[@name='Logic']=1">
774:              <xsl:text>200,200,200</xsl:text>
775:            </xsl:if>
776:          </xsl:variable>
777:          <circle id="{@id}" cx="{@x}" cy="{@y}" r="{@w div 2}" fill="rgb({$fill})" stroke="rgb({$stroke})" snp:net="{@
net}"/>
778:        </xsl:for-each>
779:        <xsl:apply-templates select="attribute[@name='Name']"/>
780:        <script type="text/javascript">
781:              <xsl:text>setMarking(,</xsl:text>
782:              <xsl:value-of select="@id"/>
783:              <xsl:text>', </xsl:text>
784:              <xsl:value-of select="$marking"/>
785:              <xsl:text>);</xsl:text>
786:          </script>
787:      </g>
788:    </xsl:template>
789:    <xsl:template match="nodeclass[@name='Transition']/node | nodeclass[@name='Coarse Transition']/node">
790:    <!--
791:      <script type="text/javascript">
792:        <xsl:text>alert(,</xsl:text>
793:        <xsl:value-of select="./../@name"/>
794:        <xsl:text>');</xsl:text>
795:      </script>
796:      -->
797:      <g id="{@id}" class="{../@name}" snp:firing="0" snp:net="{@net}" onclick="fire(,{@id}');">
798:        <xsl:if test="./../@name='Coarse Transition'">
799:          <xsl:attribute name="snp:coarse"><xsl:value-of select="@coarse"/></xsl:attribute>
800:        </xsl:if>
801:        <xsl:for-each select="graphics/graphic">
802:          <xsl:variable name="stroke">
803:            <xsl:if test="@pen and not(@net &gt; ../../@net) and not(../../../@name='Coarse Transition')">
804:              <xsl:value-of select="@pen"/>
805:            </xsl:if>
806:            <xsl:if test="not(@pen) and not(@net &gt; ../../@net) and not(../../../@name='Coarse Transition')">
807:              <xsl:text>0,0,0</xsl:text>
808:            </xsl:if>
809:            <xsl:if test="../../../@name='Coarse Transition'">
810:              <xsl:text>0,0,0</xsl:text>
811:            </xsl:if>
812:            <xsl:if test="@net &gt; ../../@net and not(../../../@name='Coarse Transition')">
813:              <xsl:text>0,0,255</xsl:text>
814:            </xsl:if>
815:          </xsl:variable>
816:          <xsl:variable name="fill">
817:            <xsl:if test="@brush">
818:              <xsl:value-of select="@brush"/>
819:            </xsl:if>
820:            <xsl:if test="not(@brush)">
821:              <xsl:text>255,255,255</xsl:text>
822:            </xsl:if>
823:          </xsl:variable>
824:          <xsl:variable name="width">
825:            <xsl:if test="@w">
826:              <xsl:value-of select="@w"/>
827:            </xsl:if>
828:            <xsl:if test="not(@w)">
829:              <xsl:value-of select="20"/>
```

```
830:            </xsl:if>
831:          </xsl:variable>
832:          <xsl:variable name="height">
833:            <xsl:if test="@h">
834:              <xsl:value-of select="@h"/>
835:            </xsl:if>
836:            <xsl:if test="not(@h)">
837:              <xsl:value-of select="20"/>
838:            </xsl:if>
839:          </xsl:variable>
840:          <rect id="{@id}" x="{@x - ($width div 2)}" y="{@y - ($height div 2)}" width="{$width}" height="{$height}"
fill="rgb({$fill})" stroke="rgb({$stroke})"/>
841:          <xsl:if test="../../../@name='Coarse Transition'">
842:            <rect x="{@x - ($width div 2) + ($width div 4)}" y="{@y - ($height div 2) + ($height div 4)}" width="{$width
div 2}" height="{$height div 2}" fill="none" stroke="rgb(0,0,0)" stroke-width="1px"/>
843:          </xsl:if>
844:        </xsl:for-each>
845:        <xsl:apply-templates select="attribute[@name='Name']"/>
846:      </g>
847:    </xsl:template>
848:    <xsl:template match="nodeclass[@name='Comment']/node">
849:      <g id="{@id}" class="{../@name}" snp:net="{@net}">
850:        <text x="{graphics/graphic/@x}" y="{graphics/graphic/@y}" style="font-family:sans-serif; font-size:11px;">
851:          <xsl:value-of select="attribute[@name='Comment']"/>
852:        </text>
853:      </g>
854:    </xsl:template>
855:    <xsl:template match="attribute[@name='Name']">
856:      <xsl:for-each select="graphics/graphic">
857:        <xsl:variable name="x" select="number(@x) - ((string-length(normalize-space(../../.)) div 2)*8)"/>
858:        <text x="{$x}" y="{number(@y)+4}" style="font-family:sans-serif; font-size:11px;"><xsl:value-of select="normalize-
space(../../.)"/></text>
859:      </xsl:for-each>
860:    </xsl:template>
861:    <xsl:template match="attribute[@name='Comment']">
862:      <text x="{graphics/graphic/@x}" y="{graphics/graphic/@y}" style="font-family:verdana; font-size:11px;">
863:        <xsl:value-of select="."/>
864:      </text>
865:    </xsl:template>
866:    <xsl:template match="edge">
867:      <xsl:variable name="multiplicity" select="normalize-space(attribute[@name='Multiplicity'])"/>
868:      <xsl:if test="$multiplicity&gt;1">
869:            <xsl:variable name="xoff">
870:              <xsl:if test="attribute[@name='Multiplicity']/graphics/graphic/@xoff">
871:                <xsl:value-of select="attribute[@name='Multiplicity']/graphics/graphic/@xoff"/>
872:              </xsl:if>
873:              <xsl:if test="not(attribute[@name='Multiplicity']/graphics/graphic/@xoff)">
874:                <xsl:value-of select="0"/>
875:              </xsl:if>
876:            </xsl:variable>
877:            <xsl:variable name="yoff">
878:              <xsl:if test="attribute[@name='Multiplicity']/graphics/graphic/@yoff">
879:                <xsl:value-of select="attribute[@name='Multiplicity']/graphics/graphic/@yoff"/>
880:              </xsl:if>
881:              <xsl:if test="not(attribute[@name='Multiplicity']/graphics/graphic/@yoff)">
882:                <xsl:value-of select="0"/>
883:              </xsl:if>
884:            </xsl:variable>
885:            <text x="{attribute[@name='Multiplicity']/graphics/graphic/@x}" y="{attribute[@name='Multiplicity']/graphics/
graphic/@y}" style="font-family:verdana; font-size:11px;">
886:              <xsl:value-of select="$multiplicity"/>
887:            </text>
888:      </xsl:if>
889:      <g id="{@id}" class="{../@name}" snp:source="{@source}" snp:target="{@target}" snp:net="{@net}" snp:multiplicity="{$m
ultiplicity}">
890:        <xsl:for-each select="graphics/graphic">
891:          <xsl:variable name="list_of_points">
892:            <xsl:for-each select="points/point">
893:              <xsl:value-of select="@x"/>
894:              <xsl:text>,</xsl:text>
895:              <xsl:value-of select="@y"/>
896:              <xsl:if test="position() != last()"><xsl:text> </xsl:text></xsl:if>
897:            </xsl:for-each>
898:          </xsl:variable>
899:          <xsl:variable name="color">
900:            <xsl:if test="@pen and not(@net &gt; ../../@net)">
901:              <xsl:value-of select="@pen"/>
902:            </xsl:if>
903:            <xsl:if test="not(@pen) and not(@net &gt; ../../@net)">
904:              <xsl:text>0,0,0</xsl:text>
905:            </xsl:if>
906:            <xsl:if test="@net &gt; ../../@net">
907:              <xsl:text>0,0,255</xsl:text>
908:            </xsl:if>
909:          </xsl:variable>
910:          <polyline id="{@id}" fill="none" stroke="rgb({$color})" stroke-width="1px" points="{$list_of_points}" marker-
```

```
end="url(#Triangle)" snp:source="{@source}" snp:target="{@target}" snp:net="{@net}"
onclick="alert(,{@id} : {@source} -> {@target}');"/>
911:           </xsl:for-each>
912:       </g>
913:     </xsl:template>
914:   </xsl:stylesheet>
```

Anhang C – liveness_1.ltl

Quelle:

http://www.fmi.uni-stuttgart.de/szs/tools/mckit/liveness_1.ltl

letzter Zugriff: 02.06.2008

```
G ((„trying1“ => (F „incs1“)) & („trying2“ => (F „incs2“)))
```

Anhang D – dekker.cfa

Dieser Anhang beschreibt den Dekker-Mutex-Algorithmus [Ray86] in CFA-Syntax (C*ommunicating Finite Automata*).
Quelle:

`http://www.fmi.uni-stuttgart.de/szs/tools/mckit/dekker.cfa`

letzter Zugriff: 02.06.2008

```
CFA
@global
var flag1, flag2:{0..1} init 0;
var turn:{1..2} init 1;

@automaton dek1
A1 {@start @label „idle1“}.
A2 {@label „trying1“}.
A3 {@label „incs1“}.
A5 {@label „trying1“}.
A6 {@label „trying1“}.
A7 {@label „trying1“}.
A8 {@label „trying1“}.
A1 [flag1‘=1] A2
A2 [flag2=0] A3
A3 [turn‘=2] A4
A4 [flag1‘=0] A1
A2 [flag2=1] A5
A5 [turn=1] A2
A5 [turn=2] A6
A6 [flag1‘=0] A7
A7 [turn=2] A7
A7 [turn=1] A8
A8 [flag1‘=1] A2

@automaton dek2
B1 {@start @label „idle2“}.
B2 {@label „trying2“}.
B3 {@label „incs2“}.
B5 {@label „trying2“}.
B6 {@label „trying2“}.
B7 {@label „trying2“}.
B8 {@label „trying2“}.
B1 [flag2‘=1] B2
B2 [flag1=0] B3
B3 [turn‘=1] B4
B4 [flag2‘=0] B1
B2 [flag1=1] B5
B5 [turn=2] B2
B5 [turn=1] B6
B6 [flag2‘=0] B7
B7 [turn=1] B7
B7 [turn=2] B8
B8 [flag2‘=1] B2
```

Quellen

Literaturverzeichnis

[Abel90]
Dirk Abel:
Petri-Netze für Ingenieure.
Berlin, Heidelberg: Springer Verlag 1990.

[Adobe1]
http://www.adobe.com/products/player_census/flashplayer/version_penetration.html
letzter Zugriff am 02.04.2008

[Adobe2]
http://www.adobe.com/svg/viewer/install/
letzter Zugriff am 15.05.2008

[CHARLIE]
http://www-dssz.informatik.tu-cottbus.de/index.html?/software/charlie.html
letzter Zugriff: 06.06.2008

[DöPe88]
Dörfler, Willibald / Peschek, Werner:
Einführung in die Mathematik für Informatiker.
München, Wien: Carl Hanser Verlag 1988.

[Dube07]
Dube, Matthias:
Entwicklung und Realisierung eines allgemeinen Konzepts zur
Interaktion zweier Netzklassen in Snoopy.
Diplomarbeit, Brandenburgische Technische Universität Cottbus,
Lehrstuhl für Datenstrukturen und Softwarezuverlässigkeit 2007.

[Freeman06]
Freeman, Eric / Freeman, Elisabeth /Sierra, Kathy / Bates, Bert:
Entwurfsmuster von Kopf bis Fuß.
Köln: O'Reilly Verlag 2006.

[FOX]
http://developer.mozilla.org/en/docs/SVG_in_Firefox
letzter Zugriff am 23.04.2008

Quellen

[Gamma96]
Gamma, Erich / Helm, Richard / Johnson, Ralph / Vlissides, John:
Entwurfsmuster.
München: Addison-Wesley Verlag 1996.

[Heiner08]
Heiner, Monika / Richter, Ronny / Schwarick, Martin / Rohr, Christian:
Snoopy - A Tool to Design and Execute Graph-Based Formalisms.
Technical Paper, Brandenburgische Technische Universität Cottbus, Lehrstuhl für Datenstrukturen und Softwarezuverlässigkeit 2008.

[INK]
Inkscape
http://www.inkscape.org/
letzter Zugriff: 06.06.2008

[MCKit]
Model-Checking Kit
http://www.fmi.uni-stuttgart.de/szs/tools/mckit/
letzter Zugriff: 02.06.2008

[MOZ]
http://www.mozilla.org/projects/svg/status.html
letzter Zugriff am 02.04.2008

[OPERA]
http://www.opera.com/docs/specs/svg/
letzter Zugriff: 14.05.2008

[PEP]
Programming Environment based on Petri Nets
http://theoretica.informatik.uni-oldenburg.de/~pep/
letzter Zugriff: 06.06.2008

[PNP]
Petri Net Pathways
http://genome.ib.sci.yamaguchi-u.ac.jp/~pnp/
letzter Zugriff: 06.06.2008

Quellen

[PROD]
An Advanced Tool for Efficient Reachability Analysis
http://www.tcs.hut.fi/Software/prod/
letzter Zugriff: 06.06.2008

[Ray86]
Raynal, Michel:
Algorithms for Mutual Exclusion.
North Oxford: Academic Publishers Ltd. 1986.

[Schwa06]
Schwarick, Martin:
Ein Werkzeug zur Analyse von Petrinetzmodellen.
Masterarbeit, Brandenburgische Technische Universität Cottbus,
Lehrstuhl für Datenstrukturen und Softwarezuverlässigkeit 2006.

[SMIL05]
W3C Recommendation 13 December 2005:
Synchronized Multimedia Integration Language (SMIL 2.1)
http://www.w3.org/TR/SMIL/
letzter Zugriff am 02.04.2008

[SPIN]
http://netlib.sandia.gov/spin/
letzter Zugriff: 06.06.2008

[Star90]
Starke, Peter H.
Analyse von Petri-Netz-Modellen
Stuttgart: Teubner 1990

[Sum1]
Sumption, Bernhard:
Animator.js – a Javascript animation library.
http://www.berniecode.com/writing/animator.html
letzter Zugriff: 03.06.2008

[SVG1]
http://www.w3.org/Graphics/SVG/
letzter Zugriff: 06.06.2008

Quellen

[W3C1]
http://www.w3.org/Graphics/SVG/Test/
letzter Zugriff: 05.06.2008

[W3C2]
http://www.w3.org/TR/xml-stylesheet/
letzter Zugriff: 05.06.2008

[WKit]
WebKit SVG Status
http://webkit.org/projects/svg/status.xml
letzter Zugriff: 06.06.2008

[wxW]
wxWidgets - Cross-Plattform GUI Library
http://www.wxwidgets.org/
letzter Zugriff: 02.06.2008

[XCE]
http://xml-copy-editor.sourceforge.net/
letzter Zugriff: 06.06.2008

[XPath]
http://www.w3.org/TR/xpath
letzter Zugriff: 06.06.2008

[XSD]
http://www.w3.org/XML/Schema
letzter Zugriff: 06.06.2008

[XSL1]
http://www.w3.org/Style/XSL/
letzter Zugriff: 06.06.2008

[XSLT]
http://www.w3.org/TR/xslt
letzter Zugriff: 06.06.2008

Abbildungsverzeichnis

Abb. 1 ..... **1**

Beispiele für verschiedene Netzwerke (v.l.n.r.): Eisenbahnnetz in Deutschland, Routen durch Teile des Internets, Blutkreislauf des Menschen

Quelle:
`http://de.wikipedia.org/w/index.php?title=Bild:Grafik_blutkreislauf.jpg`
letzter Zugriff: 02.05.2008

Abb. 2 ..... **3**

Visualisierung eines einfachen ungerichteten Graphen mit sechs Knoten und sechs unbenannten Kanten.

Quelle:
`http://de.wikipedia.org/w/index.php?title=Bild:Nicht_topologisch_sortierbarer_Graph.svg`
letzter Zugriff: 02.05.2008

Abb. 3 ..... **6**

Visualisierung eines gerichteten Graphen mit sechs Knoten und sechs Kanten mit Anfangs- und Endpunkt

Quelle:
`http://de.wikipedia.org/w/index.php?title=Bild:Nicht_topologisch_sortierbarer_Graph.svg`
letzter Zugriff: 06.06.2008

Abb. 4 ..... **7**

Transition mit Vor- und Nachbedingungen

Quelle:
Ergebnis des EPS-Exports in Snoopy mit der Datei fire1.spped. Die SPPED-Datei ist in den Snoopy Beispielen unter
`http://www-dssz.informatik.tu-cottbus.de/software/snoopy/snoopy_examples.zip`
zu finden.

Quellen

Abb. 5 8

Erreichbarkeitsgraph eines Petrinetzes mit 9 verschiedenen System-Zuständen

Quelle:
`http://en.wikipedia.org/wiki/Image:Reachability_graph_for_petri_net.png`
letzter Zugriff: 06.06.2008

Abb. 6 10

Das Werkzeug Snoopy in Anwendung auf dem Windows-Betriebssystem

Quelle:
Selbst erstellter Screenshot von Snoopy am 06.06.2008

Abb. 7 12

Animationsfenster in Snoopy

Quelle:
Selbst erstellter Screenshot von Snoopy am 04.05.2008

Abb. 8 13

Fenster-Dialog für das Ändern der Eigenschaften einer Animation in *Snoopy*

Quelle:
Selbst erstellter Screenshot von Snoopy am 07.05.2008

Abb. 9 15

schematische Darstellung der Struktur eines Clips

Quelle:
Selbst erstellte Grafik mit Indesign am 10.05.2008

Abb. 10 16

Klassendiagramm des Beobachter-Musters

Quelle:
[Freeman06] S.52

Abb. 11 17

Buttons zur Steuerung der Animation und Record-Button zur Steuerung der Aufnahme

Quelle:
Selbst erstellter Screenshot von Snoopy (Ausschnitt) am 06.06.2008

Quellen

Abb. 12 20

Erweitertes Animationsfenster mit zusätzlichen GUI-Elementen:Auswahlliste, Texteingabefeld und Button zum Löschen von Clips

Quelle:
Selbst erstellter Screenshot von Snoopy (Ausschnitt) am 06.06.2008

Abb. 13 28

Flash-basierte Petrinetz-Animation

Quelle:
`http://genome.ib.sci.yamaguchi-u.ac.jp/~pnp/token_EGF/c_8.html`
letzter Zugriff: 05.06.2008

Abb. 14 36

Screenshot eines Petrinetzes in Snoopy

Quelle:
Selbst erstellter Screenshot von Snoopy am 06.06.2008

Abb. 15 37

Darstellung der erzeugten SVG-Datei im Webbrowser Safari

Quelle:
Selbst erstellter Screenshot von Safari am 06.06.2008

Abb. 16 38

Schematische Darstellung der Transformation einer SPPED-Datei in eine SVG-Datei mit Hilfe einer XSL-Datei und einem XSLT-Prozessor

Quelle:
Selbst erstellte Grafik in Inkscape am 06.06.2008

Abb. 17 41

Icons als GUI-Komponenten zur Steuerung des Skalierens und des Verschiebens eines Petrinetzes im Browser

Quelle:
Selbst erstellter Screenshot von Safari (Ausschnitt) am 06.06.2008

Quellen

Abb. 18 42

Auswahlliste zur Navigation und Anzeige eines hierarchischen Petrinetzes

Quelle:
Selbst erstellter Screenshot von Safari (Ausschnitt) am 06.06.2008

Abb. 19 46

Icons als GUI-Komponenten zur Steuerung der Animation eines Petrinetzes im Browser

Quelle:
Selbst erstellter Screenshot von Safari (Ausschnitt) am 06.06.2008

Abb. 20 46

Fenster zum Einstellen der Optionen einer Petrinetz-Animation im Browser

Quelle:
Selbst erstellter Screenshot von Safari (Ausschnitt) am 06.06.2008

Tabellenverzeichnis

Tabelle 1 ..... **30**

Implementierte Spezifikationen von verschiedenen Webbrowsern auf verschiedenen Betriebssystemen zur Darstellung von SVG

Quelle:[MOZ] [FOX] [OPERA] [WKit]