Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Bedirhan Seyda Sahin

Entwicklung eines grafischen SIPN-Editors mit AWL-Codegenerator

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Bedirhan Seyda Sahin

Entwicklung eines grafischen SIPN-Editors mit AWL-Codegenerator

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Ing. Ulfert Meiners
Zweitgutachter : Prof. Dr. rer. nat. Henning Hasemann

Abgegeben am 11. April 2011

Bedirhan Seyda Sahin

Thema der Bachelorthesis

Entwicklung eines grafischen SIPN-Editors mit AWL-Codegenerator

Stichworte

SINP, Grafischer Editor, SipnLab, Petrinetz, Schaltungstechnisch Interpretierte Petrinetz

Kurzzusammenfassung

Diese Arbeit befasst sich mit der automatischen Erstellung von Steuerungslösungen für SPS der SIMATIC S7 Serie von Siemens.
Dazu wird eine Softwarelösung entwickelt, welche in der Lage ist, die grafische Modellierung eines schaltungstechnisch interpretierten Petrinetzes zu ermöglichen. Die Softwarelösung stellt außerdem noch weitere Werkzeuge bereit, die das modellierte Netz mit der Hilfe einer Symboltabelle in AWL-Quellcode überführt.

Bedirhan Seyda Sahin

Title of the paper

Development of a graphical SIPN-Editor with AWL-code-generator

Keywords

SIPN, Graphical Editor, SipnLab, Petri Net, Signal Interpreted Petri Net

Abstract

This work deals with the automatic creation of control solutions for PLC SIMATIC S7 series of Siemens.
Therefor a software solution is developed, which is able to provide the graphical modelling of a signal interpreted Petri-net. The software solution also provides additional tools to convert the modelled network with the help of a symbol table in STL Source.

Inhaltsverzeichnis

Abbildungsverzeichnis ... V

Tabellenverzeichnis ... VI

Listingverzeichnis ... VI

1 Einführung ... 1

1.1 Motivation ... 1

1.2 Ziel dieser Arbeit ... 1

2 Theoretische Grundlagen ... 2

2.1 Anweisungsliste (AWL) ... 2

2.2 Petri-Netze ... 3

2.2.1 Grafische Modellierung von Petrinetzen ... 3

2.2.2 Mathematische Darstellungen von Petrinetzen ... 6

2.2.2.1 Darstellung von Petrinetzen durch Mengen ... 7

2.2.2.2 Darstellung von Petrinetzen im Zustandsraum ... 8

2.3 Steuerungstechnisch Interpretierte Petri-Netze ... 11

2.3.1 Ein- und Ausgänge ... 11

2.3.2 Anfangsinitialisierungen von SIPN ... 12

2.3.3 Zeittransitionen ... 12

2.3.4 Löschtransitionen ... 12

2.3.5 Inhibitor- und Testkanten ... 13

2.3.6 Codierung von SIPN in AWL ... 14

2.4 Model-View-Controller Konzept ... 16

2.5 Extensible Markup Language (XML) ... 17

3 Aufgabe & Anforderungsanalyse ... 18

3.1 Beschreibung – Vorplanung ... 18

3.2 Auseinandersetzung – Vorrangegangene Recherche ... 20

3.3 Annahmen und Entscheidungen ... 23

4 Aufbau und Implementierung von SipnLab ... 24

4.1 Die Programmstruktur ... 24

4.1.1 Grober Aufbaubau des Programms ... 24

4.1.2 Datenmodell ... 25

4.1.3 Speichern und Laden von SIPN ... 26

4.2 Hinzugefügte Funktionalität ... 28

4.2.1 Ein- und Ausgänge ... 28
4.2.1.1 Grafische Darstellung der Eingänge ... 30
4.2.1.2 Grafische Darstellung der Ausgänge ... 31
4.2.1.3 Zuweisen von Ein- und Ausgängen in SipnLab ... 32
4.2.2 Testkanten ... 35
4.2.3 Löschtransitionen ... 36
4.2.4 Übersetzungstabelle ... 38
4.2.5 AWL-Codegenerator ... 39
4.3 Diverse Modifikationen ... 42
4.3.1 Animationsmodus ... 42
4.3.2 Sortierung der SIPN-Elemente ... 44
4.3.3 Weitere Modifikationen ... 44
5 Experimentelle Verifikation von SipnLab ... 45
5.1 Erstellen eines beispielhaften SIPN ... 45
5.2 Verifikation des Datenmodells mit der Theorie ... 46
5.3 Verifikation der Funktion des Animationsmodi ... 48
5.4 Generieren des AWL-Codes mit dem Codegenerator ... 50
5.5 Verifikation des generierten AWL-Codes ... 51
5.6 Demonstration eines komplexeren Netzes in SipnLab ... 53
6 Schluss ... 54
6.1 Zusammenfassung ... 54
6.2 Fazit ... 54
6.3 Ausblick ... 55
6.4 Persönliche Bemerkungen und Danksagungen ... 56
Literaturverzeichnis ... 57
Abkürzungsverzeichnis ... 59
A Anhänge ... 60
A1 Kurzbeschreibung der wichtigsten Klassen ... 60
A2 Ausgabe des Debuggers bei der Verifikation ... 61
A3 Source Code und weitere Daten ... 61
B Eigenständigkeitserklärung ... 62

Abbildungsverzeichnis

Abbildung 2-1: Petri-Netz-Elemente (Stelle, Transition, Kante und Marke) 3
Abbildung 2-2: Grundstrukturen von Petrinetzen.. 4
Abbildung 2-3: Beispielhaftes Petri-Netz für den Wechsel der Aggregatszustände 5
Abbildung 2-4: Systematische Darstellung der Aggregatszustände durch ein Petrinetz 6
Abbildung 2-5: Blockschaltbild von Steuerungstechnisch Interpretierten Petrinetzen.................. 11
Abbildung 2-6: Ein- und Ausgängen von SIPN.. 11
Abbildung 2-7: Initialisierung von SIPN 12
Abbildung 2-8: Darstellung von Zeittransitionen 12
Abbildung 2-9: Darstellung von Löschtransitionen.. 12
Abbildung 2-10: Darstellung von Test- und Inhibitorkanten 13
Abbildung 2-11: Beispiel für Codierung von SIPN 14
Abbildung 2-12: Darstellung des MVC-Konzeptes... 16
Abbildung 3-1: Skizze des angestrebten grafischen Editors.. 18
Abbildung 3-2: Demonstration von JGraph.. 20
Abbildung 3-3: Demonstration von JPowerGraph 21
Abbildung 3-4: PIPE 2.5... 22
Abbildung 4-1: Strukturierung von SipnLab.. 24
Abbildung 4-2: Klassenhierarchie des Programmpacket „dataLayer“ 25
Abbildung 4-3: Speicherung des mathematisches Modell in SipnLab 26
Abbildung 4-4: Beispiel SIPN für XML-Transformation.. 26
Abbildung 4-5: UML-Diagramm der Anbindung der Ein- und Ausgänge 28
Abbildung 4-6: Beispielnetz für Ein- und Ausgänge 28
Abbildung 4-7: Rotieren eines Eingangs mit der Transition 30
Abbildung 4-8: Rotieren eines Ausgangs an einer Stelle 31
Abbildung 4-9: SipnLab Hauptfenster.. 32
Abbildung 4-10: Darstellung des Place Editors... 33
Abbildung 4-11: Darstellung des Transition Editors.. 34
Abbildung 4-12: Grafische Darstellung einer Testkante.. 35
Abbildung 4-13: Beispiel für Löschtransitionen.. 36
Abbildung 4-14: Darstellung und Rotation der Löschtransition 37
Abbildung 4-15: Beispielnetz für Übersetzungstabelle... 38
Abbildung 4-16: Beispiel für Übersetzungstabelle 38
Abbildung 4-17: Struktogramm des AWL-Codegenerators 39
Abbildung 4-18: SipnLab Animationsmodus 42
Abbildung 5-1: Beispiel für Verifikation 45
Abbildung 5-2: Zustand 1 im Animationsmodus.. 48
Abbildung 5-3: Zustand 2 im Animationsmodus.. 49
Abbildung 5-4: Zustand 3 im Animationsmodus.. 49
Abbildung 5-5: Zuweisungstabelle des Beispiels... 50
Abbildung 5-6: Torsteuerung... 53
Abbildung 5-7: Steuerungslösung der Torsteuerung 53
Abbildung A-1: Ausgabe des Debuggers.. 61

Tabellenverzeichnis

Tabelle 1: Vorwärtsinzidenzmatrix ... 9
Tabelle 2: Rückwärtsinzidenzmatrix ... 9
Tabelle 3: Beispiel für eine Ausgangsmatrix ... 29
Tabelle 4: Beispiel für eine Eingangsmatrix ... 29
Tabelle 5: Beispiel für eine Löschmatrize ... 36

Listingverzeichnis

Listing 2-1: Codierung der Anfangsinitialisierung in OB100 ... 14
Listing 2-2: Codierung der Anfangsinitialisierung in OB1 ... 14
Listing 2-3: Codierung von Transitionen in OB1 ... 15
Listing 2-4: Codierung der Ausgänge in OB1 ... 15
Listing 2-5: Beispielhafte XML-Datei ... 17
Listing 4-1: Vereinfachte Transformation eines SIPN in eine XML-Baumstruktur ... 27
Listing 4-2: Auszug aus ValidatorArc.java ... 35
Listing 4-3: Die Methode getInitializeAWL ... 39
Listing 4-4: Die Methode getTransitionAWL ... 40
Listing 4-5: Die Methode getOutputAWL ... 41
Listing 4-6: Modification an dataLayer.fireTransition() ... 43
Listing 4-7: Modification an dataLayer.getTransitionEnabledStatusArray() ... 43
Listing 4-8: Darstellung der Klasse PetriNetObjectComparatorName ... 44
Listing 5-1: Auszug des Datenmodells beim Debuggen von SipnLab ... 47
Listing 5-2: Initialisierungsteil der generierten AWL-Code ... 51
Listing 5-3: Transitionsteil des generierten AWL-Code ... 51
Listing 5-4: Ausgangsteil des generierten AWL-Code ... 52

1 Einführung

Die Entwicklung von Steuerungslösungen für SPS, mit den klassischen Werkzeugen, ist der Punkt schnell überschritten, an dem die Quelltexte noch übersichtlich und verständlich sind. Vor allem wenn man bedenkt, dass es heutzutage nicht das einzige Kriterium ist, ein Ergebnis zu liefern, sondern dass es in einem möglichst kurzen Zeitraum mit einer guten Dokumentation abgeliefert werden muss, ist die Verwendung der klassischen Werkzeuge nicht mehr zeitgemäß.

Ein Werkzeug, welches sich bei der Modellierung von steuerungstechnischen Lösungen immer mehr verbreitet, sind steuerungstechnisch interpretierte Petrinetze (SIPN). Sie werden verwendet, um zum Beispiel die Steuerung von Ampelanlagen, Autowaschanlagen, Produktionsstraßen, usw. zu realisieren.

Die großen Vorteile, die durch die Verwendung von SIPN im Gegensatz zu den klassischen Mittel eröffnet werden, sind, dass es bei der Erstellung von größeren Steuerungslösungen nicht so schnell unübersichtlich wird. Zusätzlich stellen die modellierten SIPN automatisch eine gute Dokumentation der Steuerungslösung dar, welche spätere Erweiterung und Modifikationen erleichtern.

1.1 Motivation

Die Intention dieser Bachelorarbeit hat sich bei der Entwicklung von Steuerungslösungen mit SIPN ergeben. Die Entwicklungsarbeiten sahen so aus, dass die SIPN manuell auf Papier gebracht wurden. Bei der Modellierung auf Papier sind Korrekturen und Modifikationen oft mit mehr Arbeit verbunden. Nachdem der Entwurf fertig war, wurde das modellierte Netz manuell in AWL[1] transformiert. Oft schlichen sich bei diesem Schritt Fehler ein und diese führten dazu, dass die entwickelte Steuerungslösung nicht funktionierte und eine aufwendige Fehlersuche gestartet wurde. Nachdem alle Fehler beseitigt wurden, kam es manchmal noch vor, dass die entwickelte Steuerungslösung nicht die geforderte Aufgabenstellung erfüllte und Änderungen am SIPN erforderlich waren. Somit musste der Anpassungsprozess erneut durchgeführt werden.

Die beschriebene Problematik könnte zum größten Teil vermieden werden, indem man die Vorgänge bei der Entwicklung optimiert und zum Teil automatisiert. Dieses könnte man effizient durch eine Softwarelösung realisieren, aber leider existiert eine solche Lösung bisher nicht.

1.2 Ziel dieser Arbeit

Ziel dieser Arbeit ist es nun, eine Softwarelösung zu entwickeln, mit der es möglich ist, ein steuerungstechnisch interpretiertes Petrinetz, ähnlich wie bei Simulink[2] oder LabVIEW[3], mit einem grafischen Editor zu modellieren und bei Bedarf zu modifizieren. Die Software sollte außerdem auf Knopfdruck die Transformation des modellierten SIPN in AWL-Code erzeugen können.

---

[1] Anweisungsliste, kurz AWL dient der Programmierung von SPS (Speicherprogrammierbaren Steuerungen)
[2] Simulink ist ein grafischer Editor zum Modellieren und Simulieren mathematischer Zusammenhänge
[3] LabVIEW ist eine grafische Programmierumgebung für technische Systeme

2 Theoretische Grundlagen

In Kapitel 2 werden die theoretischen Grundlagen, die zum Verständnis und zur Beschreibung dieses Projekts benötigt werden, näher beschrieben und erläutert. Dafür werden die Themen Petrinetze, Schaltungstechnisch Interpretierte Petrinetze, AWL, Model-View-Controller Konzept und Extensible Markup Language (XML) im nötigen Umfang näher beschrieben und erläutert.

2.1 Anweisungsliste (AWL)

Die Anweisungsliste (AWL) ist eine Programmiersprache zur Entwickelung von Steuerungslösungen für Speicherprogrammierbare Steuerungen. Da jeder Hersteller von SPS seinen eigenen Dialekt von AWL besitzt, wird im Rahmen dieser Arbeit, die hier in Europa verbreitete Form benutzt, welche von Siemens für die S7 Serie verwendet wird. AWL ähnelt der Maschinensprachen für Prozessoren, weswegen sie auch zur Klasse der Pseudoassembler[4] gezählt wird.

Die Grundidee hinter dieser Sprache ist, dass die SPS trotz unterschiedlicher Hardware, immer dieselbe CPU-Architektur mit zwei Registern emuliert, und das Steuerprogramm, welches in dieser Sprache umgesetzt wurde auch auf neuere Generationen von SPS angewendet werden kann.

Die Strukturierung von AWL ist gegenüber der von Hochsprachen sehr umständlich, da die Syntax nur einen Operanden und einen Operator pro Zeile vorsieht. Verschachtelungen und bedingte Anweisungen sind nur mit der Hilfe von Sprungbefehlen möglich, welches nicht der Übersichtlichkeit und der Programmpflege dient.

[4] Pseudoassembler ist eine nachgebildete maschinennahe Programmiersprache

2.2 Petri-Netze

Petri-Netze wurden erstmals von Carl Adam Petri 1962 in seiner Dissertation[5] eingeführt. Sie dienen meistens zur Beschreibung von nebenläufigen Prozessen und verteilten Systemen. Im Laufe der Zeit sind weitere Varianten von Petri-Netzen entstanden, wie z. B. gefärbte Netze, stochastische Netze usw. Im weiteren Verlauf dieser Arbeit wird sich auf die Klasse der B/E-Netze[6] (Bedienung-Ereignis-Netze) beschränkt.

2.2.1 Grafische Modellierung von Petrinetzen

Petri-Netze ähneln den gerichteten Graphen aus der Automatentheorie, z. B. Mealy und Moore Automaten. Der hauptsächliche Unterschied zu den Automaten ist, dass Petri-Netze zwei Knotenelemente besitzen und der Zustand des Netzes durch Marken beschrieben wird. Das erste Kontenelement ist die Stelle, auch Platz oder Zustand genannt, und wird durch einen Kreis wie in Abb.1 dargestellt. Ein weiteres Knotenelement ist die Transition, auch Übergang oder Schaltbedingung genannt und wird durch ein Rechteck wie in Abb.1 dargestellt. Verbindungselemente der Netze sind die Kante und werden durch einen Pfeil wie in Abb.1 dargestellt. Marken werden durch einen Punkt wie in Abb.1 dargestellt.

Abbildung 2-1: Petri-Netz-Elemente (Stelle, Transition, Kante und Marke)

Es müssen folgende Gesetzmäßigkeiten beim Erstellen eines Petri-Netzes beachtet werden:

- Die Kanten dienen als Verbindungsstück zwischen den Knotenelementen und besitzen immer eine Richtung in die sie zeigen.
- Die Knotenelemente müssen alternieren, d.h. das auf eine Stelle eine Transition folgen und umgekehrt.
- Die aktiven Stellen/Zustände werden durch die Marken gekennzeichnet.
- Ein Markenfluss kommt nur zustande, wenn alle Stellen vor einer Transition mit einer Marke belegt sind und alle Stellen nach der Transition mit keiner Marke besetzt sind.
- Das Schalten von Transitionen erfolgt unendlich schnell.

---

[5] [Petri1962] Carl Adam Petri, 1962, Kommunikation mit Automaten
[6] [Grude1988] Prof. Dr. Ulrich Grude, 1988, Petri-Netze eine informelle Einführung in die Grundideen

Unter Beachtung der Gesetzmäßigkeiten können nun alle möglichen Arten von Strukturen aufgebaut werden, um mit ihnen noch komplexere Zusammenhänge aus der Realität abzubilden. Zuvor werden aber die Grundstrukturen die durch die Kombination der Petrinetzelementen entstehen näher betrachtet und analysiert. Diese Grundstrukturen sind in der folgenden Abbildung 2-2 dargestellt.

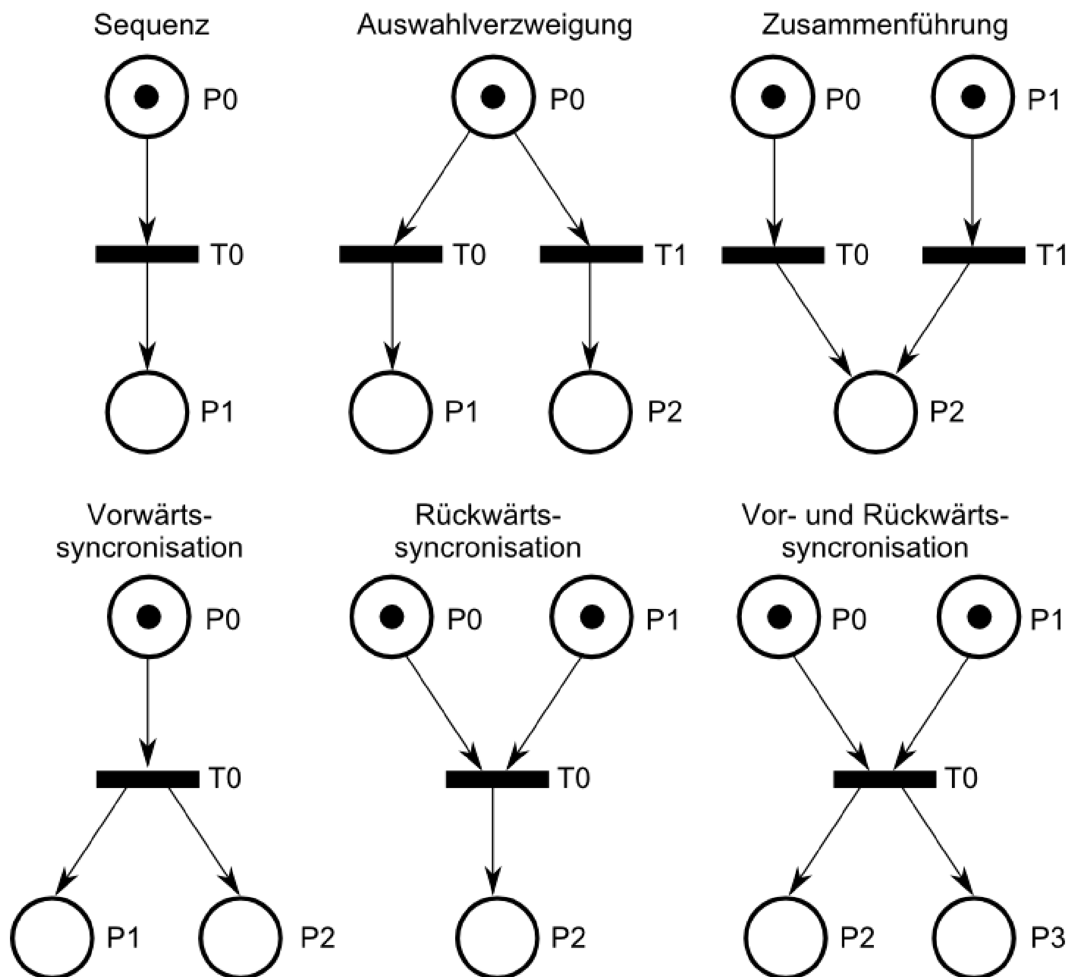


Abbildung 2-2: Grundstrukturen von Petrinetzen[7]

- Bei der „**Sequenz**" kommt beim Schalten von T0 ein Markenfluss von der Stelle P0 zu P1 zustande.
- Bei der „**Auswahlverzweigung**" hängt der Markenfluss davon ab, welche der Transitionen schaltet, wobei kein gleichzeitiges schalten beider Transitionen möglich ist. In der Praxis kommt das Problem des simultanen Schaltens nicht vor, weil das Schalten der Transitionen in der Regel einer Reinfolge und somit einer Priorisierung unterliegt. Somit fließt die Marke entweder von P0 zu P1 beim Schalten von T0 oder die Marke fließt von P0 zu P2 beim Schalten von T1.
- Bei der „**Zusammenführung**" kommt entweder ein Markenfluss von P0 zu P2 über T0 zustande oder von P1 zu P2 über T1. Auch hier ist ein gleichzeitiges Schalten der Transitionen T0 und T1 zur selben Zeit nicht zulässig. Deswegen verhält sich die Priorisierung bei der Zusammenführung genauso wie bei der Auswahlverzweigung.

[7] Vgl. [Meiners2009], Folie 138 und 140

- Die „**Vorwärtssynchronisation**“ wandelt einen sequenziellen Ablauf eines Markenflusses in einen parallelen Markenfluss um. Dabei wir die Marke von P0 beim Schalten von T0 auf P1 und P2 übertragen.
- Bei der „**Rückwärtssynchronisation**“ wird der Markenfluss aus zwei parallelen Zweigen auf einen sequenziellen Zweig geführt. Die Transition schaltet nur, wenn die Stellen vor der Transition mit Marken besetzt sind und die Stelle nach der Transition mit keiner Marke besetzt ist. Dabei werden die Marken aus den Stellen P0 und P1 abgezogen und eine Marke in P2 erzeugt.
- Die „**Vor- und Rückwärtssynchronisation**“ ist eine Kombinationen der vorigen beiden Strukturen und dient der Synchronisation des Markenflusses zweier parallelen Zweige. Dabei werden die Marken beim Schalten von der Transition T0 aus den Stellen P0 und P1 zur selben Zeit abgezogen und in den Stellen P2 und P3 erzeugt.

Mit Hilfe der zuvor betrachteten Wergzeugen ist es nun möglich, eine Vielzahl von Prozessen aus der Technik detailliert zu beschreiben. Als Beispiel kann dazu betrachtet werden, wie der Wechsel der Aggregatszustände von Stoffen mit der Hilfe von Petrinetzen dargestellt wird. Dieses Beispiel wurde gewählt, weil es sehr anschaulich ist. Genau genommen ist der Wechsel der Aggregatszustände kein diskreter Vorgang, sondern eine Verschiebung des Gleichgewichts zwischen den Phasenübergängen. Für dieses Beispiel wird aber ein diskretes Modell angenommen.

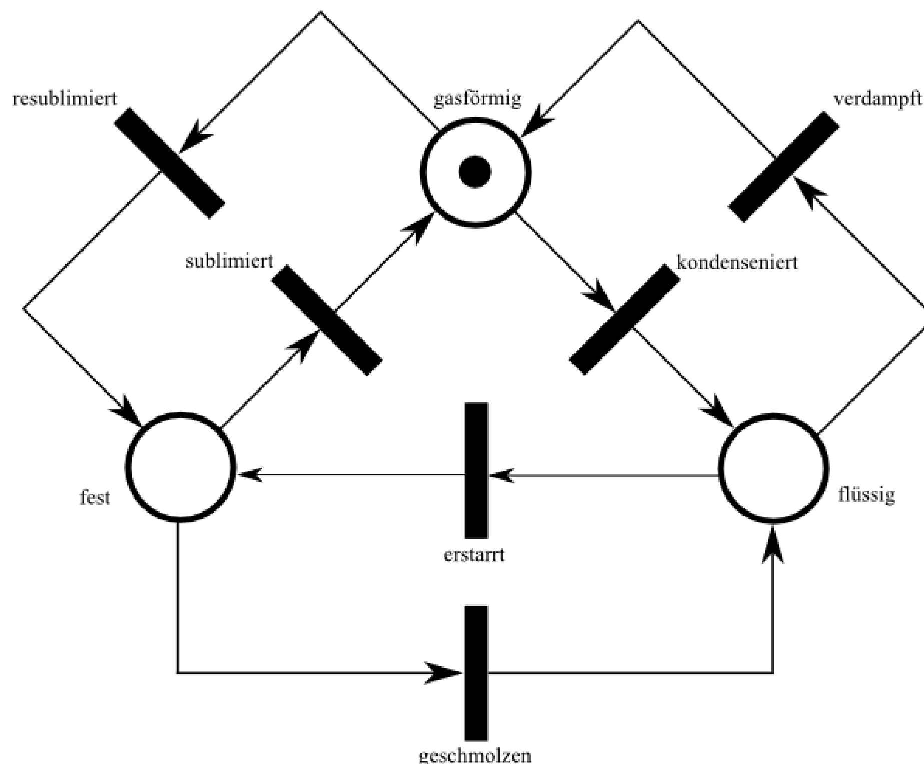


Abbildung 2-3: Beispielhaftes Petri-Netz für den Wechsel der Aggregatszustände

In Abbildung 2-3 ist das oben angesprochene Modell als fertiges Petrinetz zu sehen. Es ist zu erkennen, dass mit der Hilfe von Petrinetzen, ein gutes Modell von Prozessen erstellt werden kann, welches das Verhalten von der Realität gut wiedergibt.

2.2.2 Mathematische Darstellungen von Petrinetzen

Das Beschreiben von Petrinetzen ist nicht nur auf die Form der grafischen Darstellung beschränkt, sondern es gibt auch die Möglichkeit diese Netze mit der Hilfe der Mathematik zu beschreiben. Am Beispiel des dargestellten Netzes in Abbildung 2-3 werden im Folgenden die Darstellung von Petrinetzen durch Mengen und Matrizen erläutert. Um die systematische Beschreibung des Netzes zu ermöglichen, werden die Bezeichnungen der Stellen und Transitionen, wie in Abbildung 2-4, durch durchnummerierte Variablennamen ersetzt.

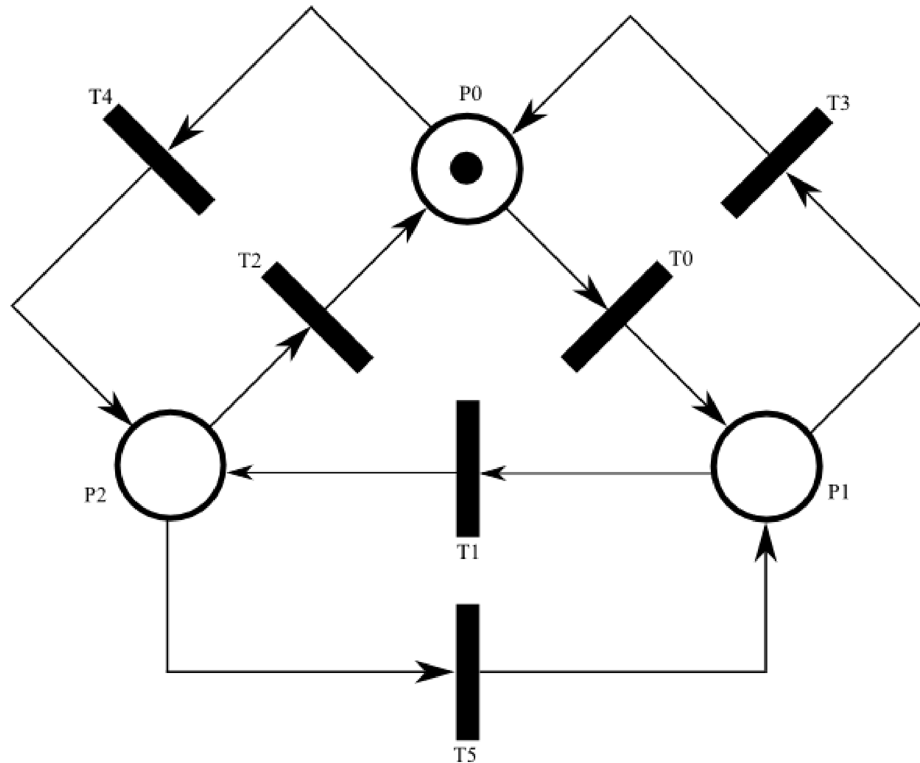


Abbildung 2-4: Systematische Darstellung der Aggregatszustände durch ein Petrinetz

2.2.2.1 Darstellung von Petrinetzen durch Mengen

Eine Methode Petrinetze formal zu beschreiben ist die Verwendung der Mengenlehre. Zu diesem Zweck werden sämtliche Elemente des Petrinetzes durch Mengen beschrieben.

- P → Menge der Stellen
- T → Menge der Transitionen
- Pre → Menge der Präkanten
- Pos → Menge der Postkanten
- M_0 → Mende der initialisierten Stellen

Die Verbindungsmengen, die die Kanten beschreiben, müssen auf Grund ihrer Unterschiede zwischen den Präkanten (Kanten die von Stellen auf Transitionen gerichtet sind) und Postkanten (Kanten die von Transitionen auf Stellen gerichtet sind) aufgeteilt werden. Die Verbindungsmengen

$$Pre \subseteq P \times T \quad \text{Formel 2-1}^{8}$$

$$Pos \subseteq T \times P \quad \text{Formel 2-2}^{9}$$

beschreiben somit alle möglichen Verbindungen in einem Netz.

Anhand der bisher definierten Mengen lässt sich nun ein Petrinetz komplett beschreiben. Für das in Abbildung 4 dargestellte Petrinetz können nun die Mengen

$$P = \{P_0, P_1, P_2\}$$
$$T = \{T_0, T_1, T_2, T_3, T_4, T_5\}$$
$$Pre = \{P_0 \to T_0, P_1 \to T_1, P_2 \to T_2, P_0 \to T_4, P_2 \to T_5, P_1 \to T_3\}$$
$$Pos = \{T_0 \to P_1, T_1 \to P_2, T_2 \to P_0, T_3 \to P_0, T_4 \to P_2, T_5 \to P_1\}$$
$$M_0 = \{P_0\}$$

beschrieben werden. Aus diesen Mengen kann nun das vollständige Petrinetz N durch den Ausdruck

$$N = (P, T, Pre, Pos, M_0) \quad \text{Formel 2-3}$$

definiert werden.

[8] [Lunze2006], S. 263
[9] [Lunze2006], S. 263

2.2.2.2 Darstellung von Petrinetzen im Zustandsraum

Eine weitere Form der Darstellung von Petrinetzen kann unter der Verwendung von Matrizen bewerkstelligt werden. Für diesen Zweck wird der Markierungsvektor

$$\boldsymbol{p}(k) = \begin{pmatrix} p_0(k) \\ \vdots \\ p_n(k) \end{pmatrix}$$ Formel 2-4

eingeführt, dessen Länge durch die Anzahl der Stellen vorgegeben ist. Den Elementen p_0 bis p_n wird eine 1 für jede markierte Stellen zum Schaltzeitpunkt k zugewiesen. Zusätzlich wird noch der Transitionsvektor

$$\boldsymbol{t}(k) = \begin{pmatrix} t_0(k) \\ \vdots \\ t_n(k) \end{pmatrix}$$ Formel 2-5

eingeführt, dessen Länge durch die Anzahl der Transitionen bestimmt wird. Den Elementen t_0 bis t_n wird eine 1 für jede aktive Transition zum Schaltzeitpunkt k zugewiesen. Als letztes Element wird noch die Netzmatrix, bzw. Inzidenzmatrix $\boldsymbol{I}$

$$\boldsymbol{I} = \boldsymbol{I}^+ - \boldsymbol{I}^-$$ Formel 2-6[10]

eingeführt, die aus der Differenz der Vorwärtsinzidenzmatrix $\boldsymbol{I}^+$ und Rückwärtsinzidenzmatrix $\boldsymbol{I}^-$ besteht. Die Vorwärtsinzidenzmatrix wird durch die Verbindungen der Postkanten gewonnen. Sie ist durch die Matrix

$$\boldsymbol{I}^+ = \begin{pmatrix} i_{00} & \cdots & i_{0s} \\ \vdots & \ddots & \vdots \\ i_{z0} & \cdots & i_{zs} \end{pmatrix} \quad \text{für } i_{zs} = \begin{cases} 1 & \text{wenn } t_s \to p_z \\ 0 & sonst \end{cases}$$ Formel 2-7

definiert. Die Rückwärtsinzidenzmatrix wird durch die Verbindungen der Präkanten gewonnen. Sie ist durch die Matrix

$$\boldsymbol{I}^- = \begin{pmatrix} i_{00} & \cdots & i_{0s} \\ \vdots & \ddots & \vdots \\ i_{z0} & \cdots & i_{zs} \end{pmatrix} \quad \text{für } i_{zs} = \begin{cases} 1 & \text{wenn } p_z \to t_s \\ 0 & sonst \end{cases}$$ Formel 2-8

definiert. Mit den eingeführten Vektoren und Matrizen kann nun das Petrinetz in der Zustandsraumdarstellung

$$\boldsymbol{p}(k+1) = \boldsymbol{p}(k) + \boldsymbol{I} \cdot \boldsymbol{t}(k)$$ Formel 2-9[11]

beschrieben werden. Die Zustandsraumdarstellung beschreibt die Änderungen der Zustände des Netzes zum Zeitpunkt k berechnen, wenn die Schaltbedingungen aus 2.2.1 für $\boldsymbol{t}(k)$ eingehalten werden.

[10] Vgl. [Lunze2006], S. 271
[11] Vgl. [Lunze2006], S. 271

Um auf die Matrixdarstellung für das in Abbildung 2-1 abgebildeten Beispielnetz zu kommen, werden der Markierungsvektor $\boldsymbol{p}$ und der Transitionsvektor $\boldsymbol{t}$ für den Zeitpunkt k=0 erstellt.

$$\boldsymbol{p}(0) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \qquad \boldsymbol{t}(k) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Dabei ist zu beachten, dass die Transitionen T_0 und T_4 theoretisch schalten könnten, da P_0 eine Auswahlverzweigung darstellt. Um dieses Problem zu umgehen, wird wie in Kapitel 2.2.1 beschrieben eine Priorisierung der Transitionen eingeführt, die bei zwei gleichzeitig aktiven Transitionen, die mit der geringeren Zahl in der Bezeichnung bevorzugt. Für das Beispiel heißt es, dass die Transition T_0 der Transition T_4 bevorzugt wird. Als nächsten Schritt werden noch die Vorwärts- und Rückwärtsinzidenzmatrizen für das Netz erstellt. Für eine bessere Veranschaulichung, werden die beiden Matrizen in Tabellen eingetragen. Somit sieht die Darstellung der Vorwärtsinzidenzmatrix $\boldsymbol{I}^+$

Tabelle 1: Vorwärtsinzidenzmatrix

	T0	T1	T2	T3	T4	T5
P0	0	0	1	1	0	0
P1	1	0	0	0	0	1
P2	0	1	0	0	1	0

und die Darstellung für die Rückwärtsinzidenzmatrix $\boldsymbol{I}^-$

Tabelle 2: Rückwärtsinzidenzmatrix

	T0	T1	T2	T3	T4	T5
P0	1	0	0	0	1	0
P1	0	1	0	1	0	0
P2	0	0	1	0	0	1

wie in Tabelle 1 und Tabelle 2 aus. Aus den beiden Matrizen $\boldsymbol{I}^+$und $\boldsymbol{I}^-$ lässt sich nach Formel 2-6

$$\mathbf{I} = \mathbf{I}^+ - \mathbf{I}^- = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} - \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} -1 & 0 & 1 & 1 & -1 & 0 \\ 1 & -1 & 0 & -1 & 0 & 1 \\ 0 & 1 & -1 & 0 & 1 & -1 \end{pmatrix}$$

die Inzidenzmatrix $\boldsymbol{I}$ berechnen.

Nachdem alle Elemente berechnet wurden, kann der Markierungsvektor $\boldsymbol{p}(1)$ nach Formel 2-9 mit

$$\boldsymbol{p}(1) = \boldsymbol{p}(0) + \boldsymbol{I} \cdot \boldsymbol{t}(0) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} -1 & 0 & 1 & 1 & -1 & 0 \\ 1 & -1 & 0 & -1 & 0 & 1 \\ 0 & 1 & -1 & 0 & 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

berechnet werden. Das berechnete Ergebnis ist wie erwartet, dass beim Schalten von T0 die Marke von P0 abgezogen und in P1 erzeugt wird.

Abschießend sei noch erwähnt, dass es bei Auswahlverzweigungen und Zusammenführungen, wie in Abbildung 2-2 oder auch diesem Beispiel, zu Problemen der Priorisierung von Transitionen kommen kann. Es gibt zwar einen analytischen Weg dieses Problem zu erkennen, wie auch Schleifen auf analytischem Weg durch die Vorwärts- und Rückwärtsinzidenzmatrix zu identifizieren sind, weil es im Rahmen dieser Arbeit jedoch um die Darstellung geht, wird hier nicht näher drauf eingegangen.

2.3 Steuerungstechnisch Interpretierte Petri-Netze

In der Steuerungstechnik finden die klassischen Petrinetze, wie sie in Kapitel 2.1 vorgestellt wurden, keine Verwendung, da für Petrinetze keine Ein- und Ausgänge definiert sind, die bei steuerungstechnischen Entwürfen üblicherweise gebraucht werden. Daher kommt in der Steuerungstechnik eine erweiterte Form von Petrinetzen zum Einsatz, welche um Ein- und Ausgänge erweitert werden.

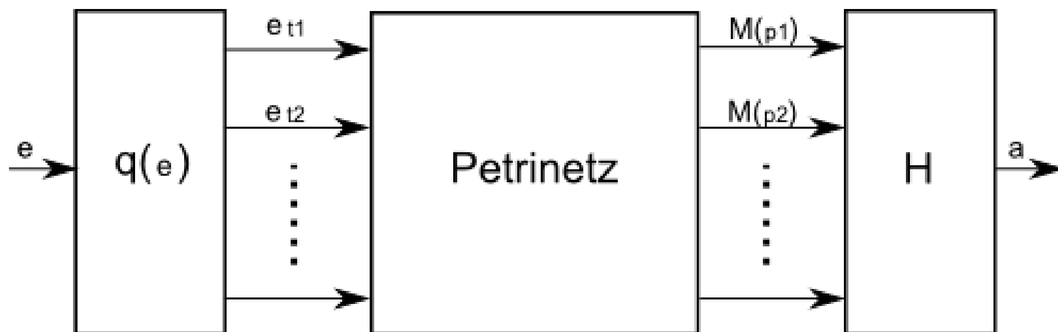


Abbildung 2-5: Blockschaltbild von Steuerungstechnisch Interpretierten Petrinetzen[12]

Die Erweiterung der Petrinetze zu den steuerungstechnisch interpretierten Petrinetzen ist in Abbildung 2-5 als Blockschaltbild dargestellt.

2.3.1 Ein- und Ausgänge

Eingänge verknüpfen die Signale der Außenwelt mit den SIPN. Im Prinzip kann jede Art von schaltalgebraischem Ausdruck als Eingangsbedingung dienen, welche als Ergebnis „1“ und „0“ hat. Transitionen die so beschaltet sind, feuern nur noch, wenn zu den bisherigen beschrieben Schaltbedingungen zusätzlich eine „1“ als Ergebnis des schaltalgebraischen Ausdruck vorliegt.

Ausgänge dienen den SIPN als Medium, um Signale für die Außenwelt zu erzeugen. Ausgänge die mit Stellen verknüpft werden, sind nur dann aktiv, wenn die Stellen mit jeweils einer Marke besetzt sind. Falls ein und derselbe Ausgang mit mehreren Stellen verknüpft ist, ist die aktive Verknüpfung immer dominierend. Bei den Ausgängen sind in dieser Arbeit schaltalgebraische Ausdrücke nicht vorgesehen.

Die Visualisierung von Ein- und Ausgängen werden durch Linien an den Transition für Eingänge bzw. Linien an den Stellen für Ausgänge dargestellt.

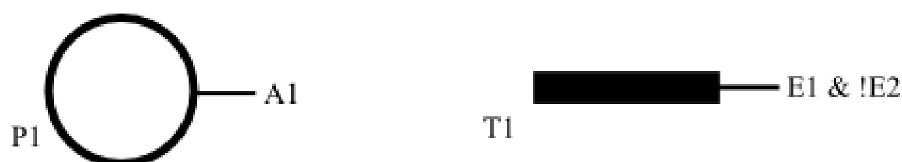


Abbildung 2-6: Ein- und Ausgängen von SIPN

Die Verknüpfung mit den Ein- und Ausgängen wird wie in Abbildung 2-6 visualisiert.

[12] Vgl. [Lunze2006], S. 302

2.3.2 Anfangsinitialisierungen von SIPN

Die Anfangsinitialisierungen[13] in SIPN werden mit einem inneren Kreis, wie in Abbildung 2-7 abgebildet, dargestellt.

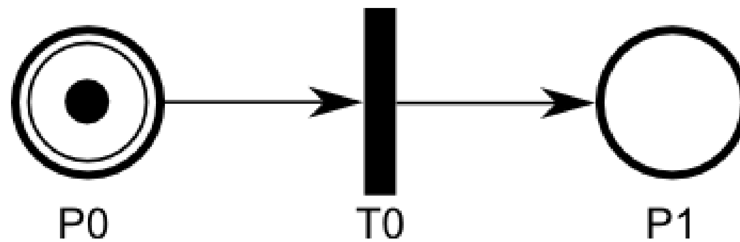


Abbildung 2-7: Initialisierung von SIPN

Diese Initialisierungen definieren den Anfangszustand des Netzes.

2.3.3 Zeittransitionen

In der Steuerungstechnik kommt es häufig vor, dass Steuerungsaufgaben von zeitlichen Faktoren abhängen. Um auf diese zeitlichen Abhängigkeiten reagieren zu können, sind SIPN um Zeittransitionen[14] erweitert worden. Bei dieser Art von Transitionen kommt ein Markenfluss nur zustande, wenn die Transition für den Zeitraum der Verzögerung aktiviert ist und somit alle Schaltbedingungen erfüllt sind. Ist beim Ablauf der Verzögerungszeit die Transition durch das Nichterfüllen einer Bedingung kurzzeitig nicht aktiv, so beginnt die Verzögerungszeit bei einer erneuten Aktivierung wieder von vorn.

Die Zeittransition werden durch ein Rechteck ohne Füllung und die Angabe der Verzögerungszeit und der Bezeichnung dargestellt. Wie in Abbildung 2-8 zu sehen ist, können Zeittransitionen auch mit Eingängen verknüpft werden.

Abbildung 2-8: Darstellung von Zeittransitionen

2.3.4 Löschtransitionen

Löschtransitionen[15] sind eine Erweiterung der Funktionalität der bisher bekannten Transitionen. Mit ihnen ist es möglich, Petrinetze beim Schalten der entsprechenden Transition in einen definierten Zustand zu bringen. Diese Funktionalität wird häufig dafür verwendet, um Teile von Netzen beim Eintreten von bestimmten Ereignissen zu deaktivieren.

Die grafische Darstellung dieser Funktionalität wird durch ein „X“ symbolisiert, was, wie in Abbildung 2-9 gezeigt, bei Transitionen und Zeittransitionen möglich ist.

Abbildung 2-9: Darstellung von Löschtransitionen

[13] Vgl. [Aspern2003], S. 85
[14] Vgl. [Aspern2003], S. 113
[15] Vgl. [Aspern2003], S. 105

2.3.5 Inhibitor- und Testkanten

Die Inhibitor[16]- und Testkanten[17] erweitern die Funktionalität von SIPN. Mit ihrer Hilfe ist es möglich die Aktivität von Stellen auszuwerten. Bei beiden Kantenarten findet kein Markenfluss zwischen den abgefragten Stellen und den Transitionen statt.

Die Matrizen der Inhibitor- und Testkanten werden nach demselben Bildungsgesetz gebildet, wie bei der Rückwärtsinzidenzmatrix nach Formel 2-8. Bei der Bildung der Matrizen werden nur entsprechende Test- oder Inhibitorkanten berücksichtigt.

Die Testkante wird durch eine quer liegende Linie am Ende der Kante dargestellt. Sie verhindert das Schalten einer Transition, falls die Stelle an die sie angekoppelt ist, nicht mit einer Marke besetzt ist. In Abbildung 2-10 ist eine Testkante dargestellt. Sie verbindet die Stelle P2 mit der Transition T1.

Die Inhibitorkante wird durch einen Kreis am Ende der Kante dargestellt. Bei ihr verhält es sich so, dass sie ein Schalten der Transition verhindert, falls die Stelle, der sie entstammt, mit einer Marke besetzt ist. In Abbildung 2-10 ist eine Inhibitorkante dargestellt. Sie verbindet die Stelle P3 mit der Transition T1.

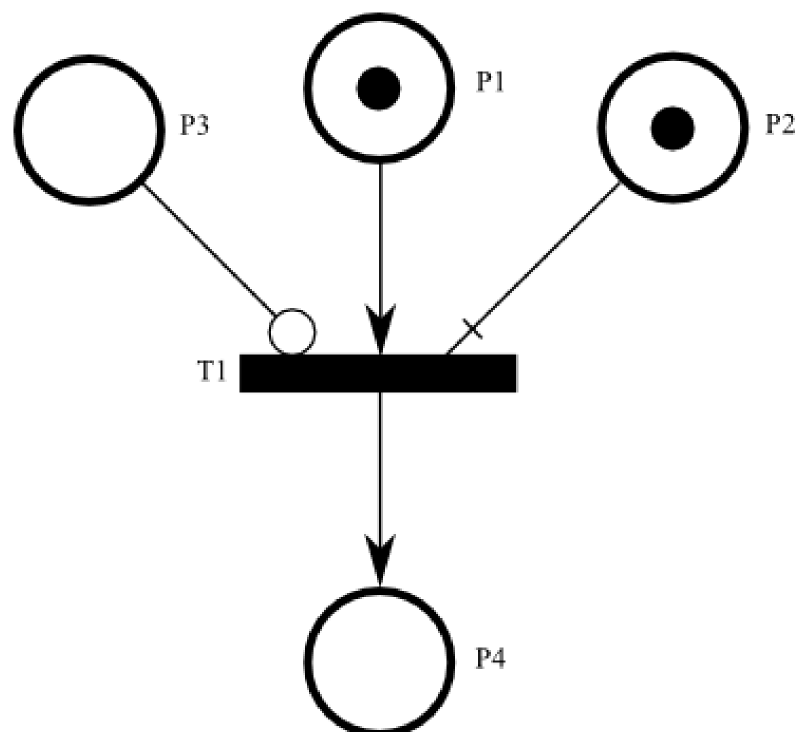


Abbildung 2-10: Darstellung von Test- und Inhibitorkanten

In Abbildung 2-10 wird eine beispielhafte Situation gezeigt, in der die Transition gefeuert werden kann und die Marke von P1 abgezogen und in P4 erzeugt. Die Stellen P2 und P3 werden vom Schalten der Transition nicht weiter beeinflusst.

---

[16] Vgl. [Aspern2003], S. 135
[17] Vgl. [Aspern2003], S. 138

2.3.6 Codierung von SIPN in AWL

Die Transformation von SIPN in AWL ist erforderlich, damit sie auf SPS ausgeführt werden können. Zu diesem Zweck werden die SIPN-Elemente den Elementen der SPS zugewiesen. Dafür werden die Stellen des SIPN den Merkern der SPS, die Eingänge des SIPN den Eingängen der SPS und die Ausgänge des SIPN den Ausgängen der SPS zugewiesen.

Die Codierung wird anhand des Beispielnetzes demonstriert, das in Abbildung 2-11 dargestellt ist.

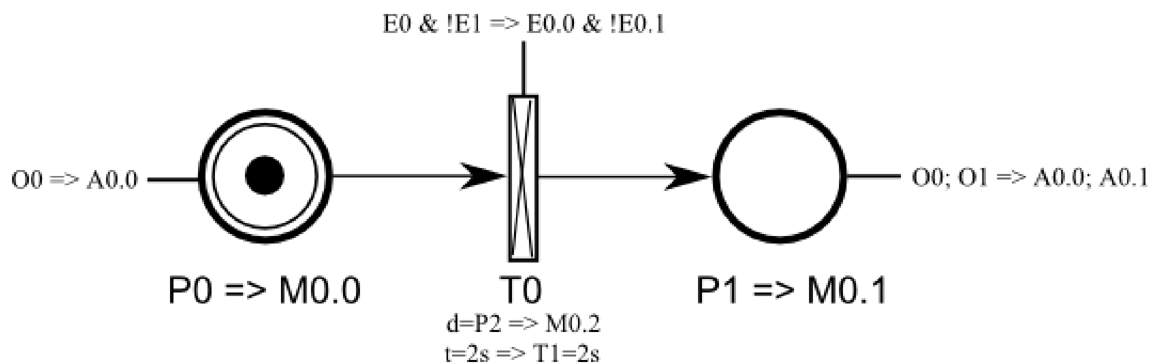


Abbildung 2-11: Beispiel für Codierung von SIPN

Anfangsinitialisierungen

Für die Codierung der Anfangsinitialisierungen gibt es 2 Varianten. Beide Varianten werden am Beispiel von Abbildung 2-11 demonstriert. Hierfür wird der Stelle P0 die Marke M0.0 zugewiesen.

1. Die Anfangsinitialisierung wird einmalig im OB100 (Organisationsbaustein 100) ausgeführt. Hierfür werden die Stellen, die als Anfangsinitialisierungen gekennzeichnet sind, wie in Listing 2-1 dargestellt, untereinander aufgelistet.

   Listing 2-1: Codierung der Anfangsinitialisierung in OB100[18]

   ```
   1  S M0.0       // Stelle P0
   2  ...          // weitere Inisitalisierungsstellen
   ```

2. Die alternative Variante der Anfangsinitialisierung wird im OB1 (Organisationsbaustein 1) realisiert. OB1 wird zyklisch ausgeführt und zu diesem Zweck muss ein Hilfsmerker verwendet werden, um abzufragen, ob die Anfangsinitialisierung schon gesetzt wurde. Hierfür werden die Stellen, die als Anfangsinitialisierungen gekennzeichnet sind, wie in Listing 2-1 dargestellt, untereinander aufgelistet.

   Listing 2-2: Codierung der Anfangsinitialisierung in OB1[19]

   ```
   1  UN M36.0     // Hilfsmerker
   2  S M0.0       // Stelle P0
   3  ...          // weitere Inisitalisierungsstellen
   ```

[18] Vgl. [Meiners2009], Folie 146
[19] Vgl. [Meiners2009], Folie 146

Transitionen

Die Reihenfolge der Codierung der Transitionen wird durch die Nummerierung bestimmt, mit der sie bezeichnet sind. Durch diese Reihenfolge wird auch die Priorisierung der Transitionen realisiert. Transitionen können durch weitere Funktionalitäten ergänzt werden, wie in den vorherigen Abschnitten erläutert. Dafür werden, wie in Listing 2-1 in den Zeilen 1 bis 8 dargestellt, die Schaltbedingung der Transition abgefragt. Dieser Abschnitt besteht aus konjunktiven Verknüpfungen aller Kanten sowie Eingängen. Im zweiten Teil, welcher in den Zeilen 9 bis 13 dargestellt sind, werden die Funktionalitäten der Transition, wie das Löschen und das Verzögern, implementiert. Im letzten Abschnitt, welcher in den Zeilen 14 bis 15 zu finden ist, wird der Markenfluss implementiert, der beim Schalten der Transition ausgelöst wird.

Listing 2-3: Codierung von Transitionen in OB1[20]

```
1   U    M0.0     // Präkante P0
2                 // gegeben falls weitere Präkanten
3   UN   M0.1     // Postkante P1
4                 // gegeben falls weitere Postkanten
5                 // eventuelle Test- und Inhibitorkanten
6   U    E0.0     // Eingang E0
7   UN   E0.0     // Eingang !E1
8                 // gegeben falls weitere Eingänge
9   L    S5T#2s   // Zeitfunktionalität – Zeit in Register Laden
10  SE   T1       // Zeitfunktionalität – setzen der Einschaltverzögerung mit Zeitregister T1
11  U    T1       // Zeitfunktionalität – Abfrage des Zeitregisters
12  R    M0.2     // Löschfunktionalität – löschen der Marke P2 beim Schalten von T0
13                // Löschfunktionalität – gegeben falls weitere zu löschende Marken
14  R    M0.2     // Markenfluss beim Schalten – löschen von P0
15  S    M0.2     // Markenfluss beim Schalten – setzen von P1
```

Ausgänge

Die Umsetzung der Ausgänge wird am Ende von OB1 durch disjunktive Verknüpfungen der Stellen realisiert. Die disjunktiven Verknüpfungen sorgen dafür, dass die Ausgänge 1 dominierend sind. Für das Beispiel in Abbildung 2-11 würde die Codierung wie im Listing 2-4 aussehen.

Listing 2-4: Codierung der Ausgänge in OB1

```
1   O   M0.0    // Stelle P0
2   O   M0.1    // Stelle P1
3   =   A0.0    // setzen des Ausgangs A0
4
5   O   M0.1    // Stelle P0
6   =   A0.1    // setzen des Ausgangs A1
```

[20] Vgl. [Meiners2009], Folie 146 und Folie 163

2.4 Model-View-Controller Konzept

Das Model-View-Controller Konzept, kurz MVC, ist die Weiterentwicklung bzw. Verfeinerung der objektorientierten Programmierung. Das Ziel dieses Konzeptes ist es, die Strukturen einer Softwareentwicklung in 3 Teile zu unterteilen: Modell (Model), Darstellung (View) und Bedienung (Controller). Durch die Aufspaltung in drei Teile ist es nun möglich, Erweiterungen und Änderungen leichter vorzunehmen. Es steigert außerdem den Grad der Wiederverwendbarkeit des produzierten Codes.

Model: Das „Model“ enthält alle relevanten Daten der Softwarelösung. Das beinhaltet auch die logischen Zusammenhänge sowie die Daten, die für die grafische Darstellung benötigt werden.

View: Die „View“ erzeugt die visuelle Darstellung eines Programms, indem es auf die vom Modell bereitgestellten Daten zugreift.

Controller: Der „Controller“ wertet die Interaktionen des Nutzers aus und ändert gezielt durch Nutzung vom Modell vorgegebener Methoden, die relevanten Daten entsprechend der Benutzervorgaben. Die Darstellung wird entsprechend des Modells aktualisiert.

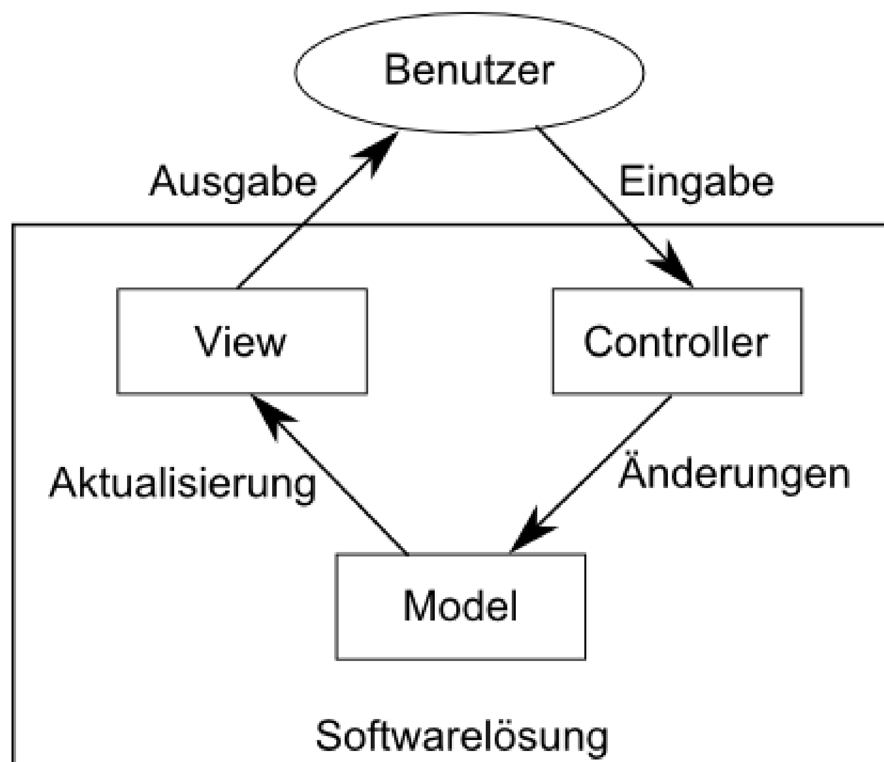


Abbildung 2-12: Darstellung des MVC-Konzeptes[21]

Das Zusammenspiel der oben beschrieben Elemente wird in Abbildung 2-12 dargestellt. Es ist zu erkennen, wie die Softwarelösung mit dem Nutzer interagiert und welche Vorgänge die Interaktionen auslösen.

[21] Vgl. [Baray2010]

2.5 Extensible Markup Language (XML)

Die Extensible Markup Language, kurz XML genannt, wurde vom „World Wide Web Consortium“ (W3C[22]) 1998[23] spezifiziert. XML ist eine Auszeichnungssprache und ähnlich wie bei HTM dient es dazu, Daten in einer Textdatei strukturiert zu speichern. Das Einsatzgebiet von XML unterscheidet sich aber von HTML. Es wird eher dazu verwendet um plattformunabhängig Daten von Programmen zu speichern. Durch diese Art der Speicherung ist es möglich die Daten, im Gegensatz zur binären Speicherung von Daten, noch lesen zu können.

Der Aufbau von XML Dokumenten ähnelt dem des objektorientierten Denkmodells, welchem viele moderne Programmiersprachen wie Java, C# usw. zugrunde liegen, weswegen es kaum verwunderlich ist, dass XML als Datenaustauschformat immer beliebter wird. Um ein populäres Beispiel auf die Umstellung auf XML zu nennen, kann Microsoft Word als Beispiel genannt werden. Sie verwenden inzwischen als Standardformat DOCX, welches auf einer XML-Struktur basiert.

Die Struktur von XML-Dateien besteht aus Elementen, welche baumförmig angeordnet sind, wobei jedes Element andere Elemente enthalten kann, die ihrerseits auch wieder Elemente enthalten können. Zusätzlich kann jedes Element noch eine beliebige Anzahl von Attributen enthalten, womit durch den Aufbau des XML-Baums ermöglicht wird, beliebig komplexe Datenstrukturen übersichtlich abgebildet werden. Ein Beispiel für eine XML-Datei könnte wie folgt aussehen:

Listing 2-5: Beispielhafte XML-Datei

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <Aquarium>
3               <Fische art =“Rote Schwertträger“>
4                           <Fisch name=“Manfred“ geschlecht="m" laenge="7cm"/>
5                           <Fisch name=“ Sandra“ geschlecht="f" laenge="6cm"/>
6                           <Fisch name=“Sabrina“ geschlecht="f" laenge="6cm"/>
7           </ Fische >
8           < Fische Art =“Black Molly“>
9                           <Fisch name=“Mylo“ geschlecht="m" laenge="5cm"/>
10                          <Fisch name=“Christina“ geschlecht="f" laenge="4cm"/>
11                          <Fisch name=“Carina“ geschlecht="f" laenge="4cm"/>
12              </ Fische >
13  </ Aquarium >
```

Im Beispiel ist die Bestückung eines Aquariums beschrieben. Die erste Zeile muss immer die oben abgebildete sein, damit Programme, die diese Datei lesen, wissen, um welches Format es sich handelt. Elemente, die Elemente enthalten, wie z.B. Aquarium, sind immer von zwei Tags (Auszeichnungen) eingeschlossen, welcher einer am Anfang und ein anderer am Ende des Elements zu finden sind. Attribute sind in den Anfangstags und haben immer einen Wert zugewiesen, wie im Beispiel beim Element Fische.

[22] [W3C2011]
[23] Nach [WikiXML]

3 Aufgabe & Anforderungsanalyse

In diesem Kapitel werden die Rahmenbedingungen und Anforderungen, die an das Projekt gestellt werden, näher betrachtet.

3.1 Beschreibung – Vorplanung

Am Anfang der Entwicklung steht die Frage, wie eine solche Softwarelösung aussehen kann und welchen Funktionsumfang sie mit sich bringen muss, um ein Werkzeug zu entwickeln, welches der Zielsetzung entspricht. Auf diese Überlegungen hin, wurde ein grober Entwurf skizziert, wie ein solches Programm aussehen könnte.

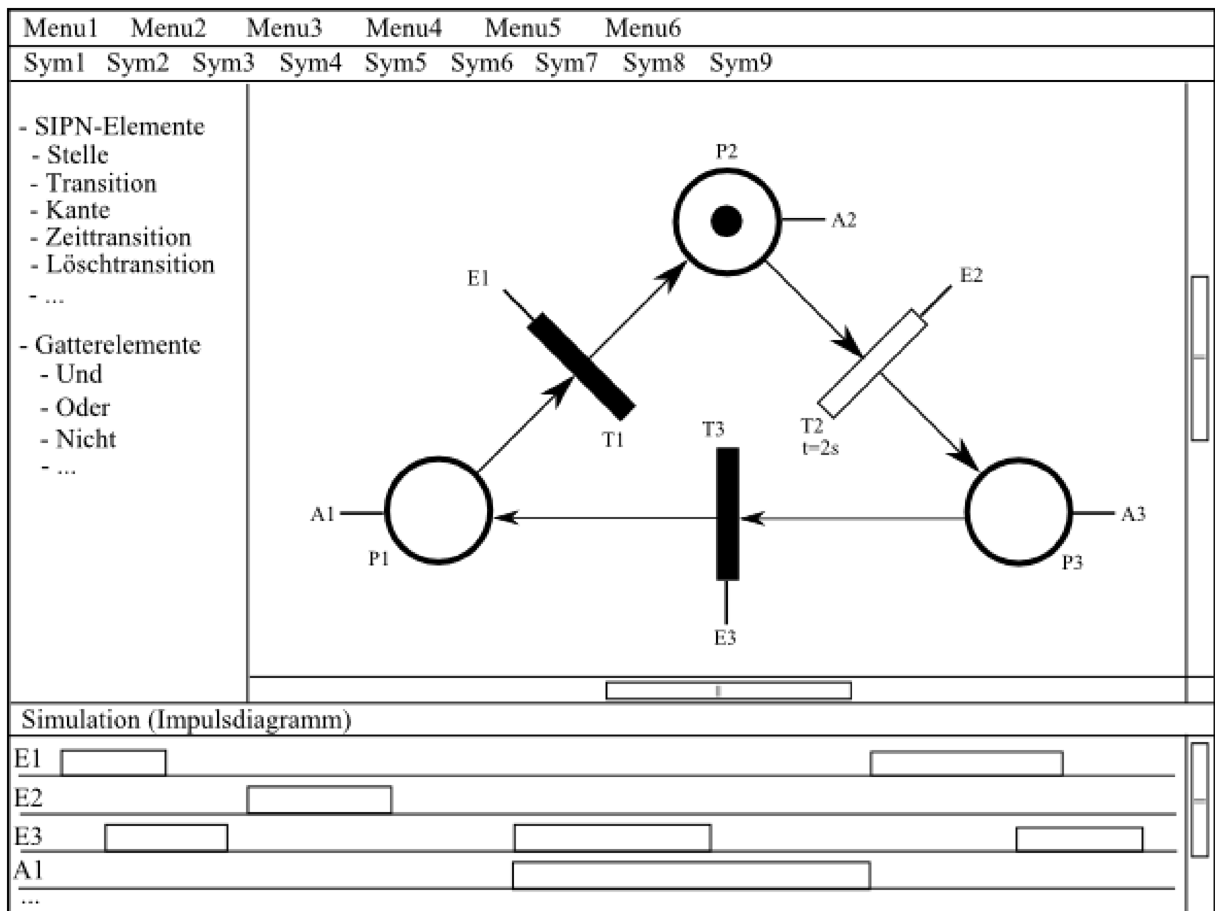


Abbildung 3-1: Skizze des angestrebten grafischen Editors

Bei der möglichen und erforderlichen Funktionalität wurde folgende Liste ausgearbeitet und anschließend die Punkte nach Dringlichkeit priorisiert. Die Priorität steht im Anschluss der Funktionalität in Klammern.

- das grafisches Erstellen von Stellen, Transitionen, Ein- und Ausgängen (hoch)
- das Erstellen von Test- und Inhibitorkanten (mittel)
- das Erstellen von Zeittransitionen (hoch)
- das Erstellen von Löschtransitionen (mittel)
- das Erstellen eines Funktionsblocks/Subnetze (niedrig)
- die Interpretation von Schaltbedingungen für die Eingänge von Transitionen (mittel)
- das Generieren von AWL-Code (hoch)
- das Generieren von SCL-Code (niedrig)
- die Simulation von erstellten Netzen (niedrig)
  - Schrittweise Simulation
  - Zyklische Simulation
  - Transiente Simulation
- die automatische Prüfung des erstellten Netzes auf Plausibilität (niedrig)
- das Erstellen einer Übersetzungs-/Zuweisungstabelle (hoch)

Durch die Priorisierung der Funktionalitäten in der Planungsphase wurde gewährleistet, dass das Ergebnis der Entwicklung brauchbare Resultate liefert. Des Weiteren wurde dadurch ein Stellglied geschaffen, mit dem der Umfang der Arbeit beeinflusst werden konnte.

3.2 Auseinandersetzung – Vorrangegangene Recherche

An diesem Punkt wirft sich die Frage auf, wie das Vorhaben umzusetzen ist. Um ein optimales Ergebnis zu erzielen, wurde eine Vielzahl an Möglichkeiten in Betracht gezogen und auf ihre Umsetzbarkeit analysiert.

Als Entwicklungswerkzeuge kommen Eclipse[24] mit der Kombination von Java zum Einsatz, weil sie nicht nur plattformunabhängig sind, sondern auch eine hohe Verbreitung und Beliebtheit in der Open Source Community genießen.

Die in Java integrierte Grafikbibliothek Swing beinhaltet nur rudimentäre Elemente zum Erstellen von grafischen Oberflächen, die meisten von ihnen entsprechen Formularelementen wie man sie von HTML-Webseiten her kennt, wie z. B. Buttons, Tabellen, Eingabefelder usw. Mit diesen Mitteln lässt sich in einer angemessen Zeit kein grafischer Editor von nur einer Person entwickeln.

Alternativ gibt es die Möglichkeit schon existierende Grafikbibliothek für Graphen, wie z. B. JGraph[25], zu verwenden. JGraph ist ein sehr mächtiges Programmiergerüst zur Erstellung von Graphen. Ein Beispiel für die Funktionalität des Frameworks ist in Abbildung 3-2 abgebildet.

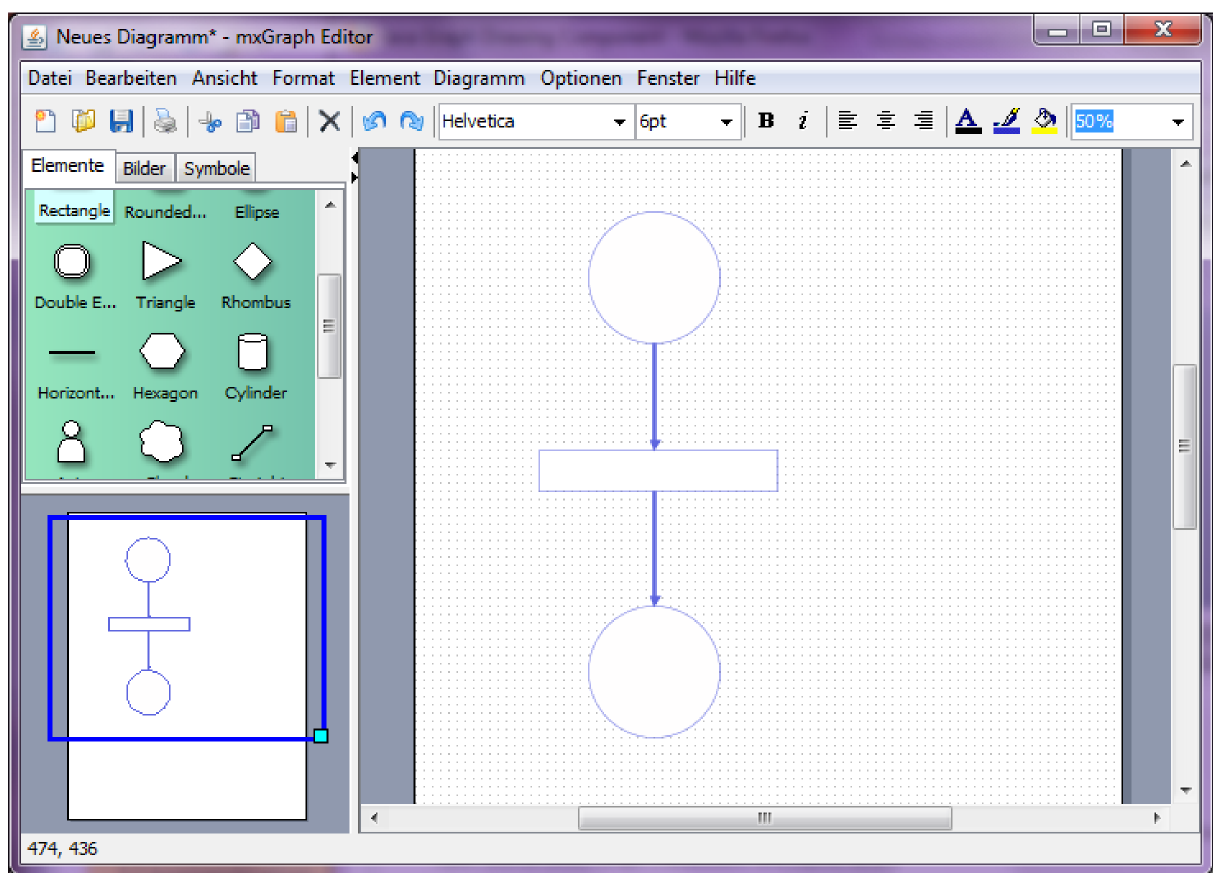


Abbildung 3-2: Demonstration von JGraph[26]

[24] [Ecl2010]
[25] [JGraph2010], JGraph, Framework für Java Komponenten, URL: www.jgraph.com
[26] [JGraph2010], Beispiel für JGraph, http://www.jgraph.com/demo/jgraphx/jgraphx.jnlp

Ein weiteres Framework zur Erstellung von Graphen ist JPowerGraph[27]. Die Möglichkeiten, die sich mit JPowerGraph eröffnen, sind Abbildung 3-3 dargestellt.

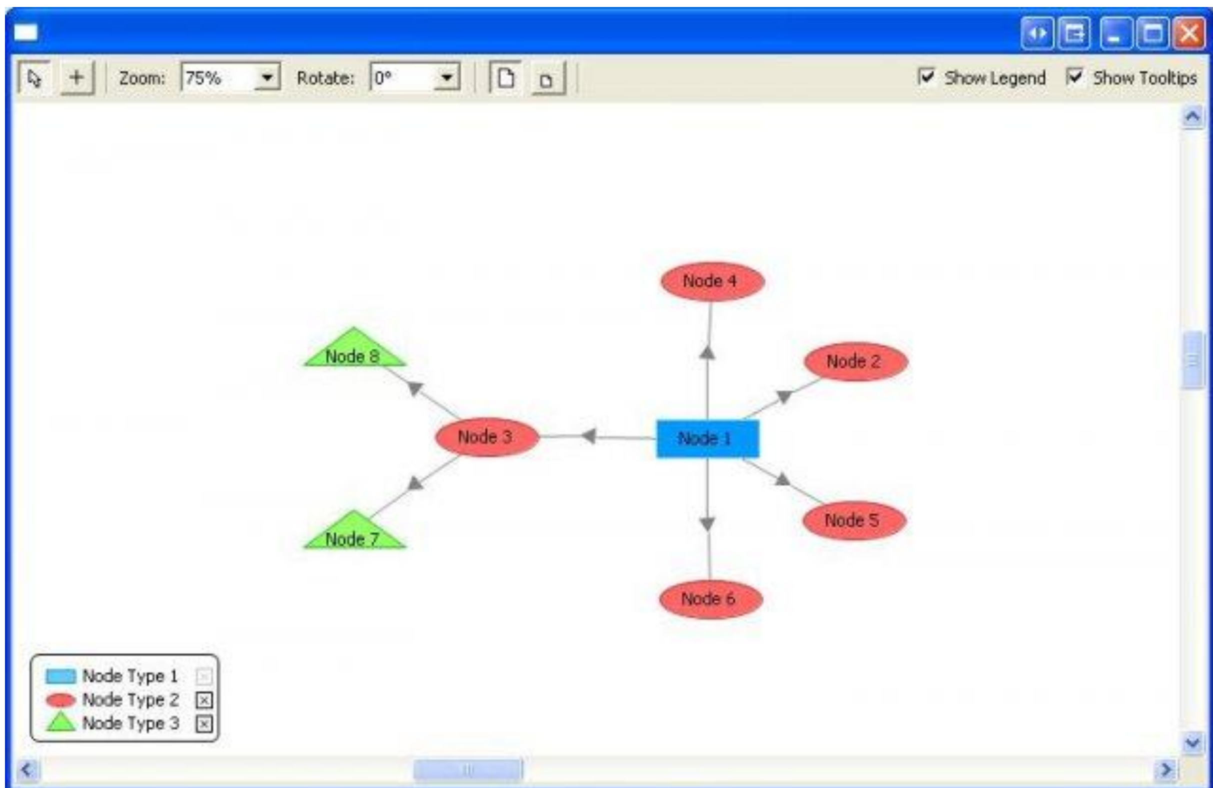


Abbildung 3-3: Demonstration von JPowerGraph[28]

Beide Frameworks ermöglichen es, durch vorgefertigte Komponenten, beliebige Graphen zu erstellen. Zu diesem Zweck gibt es Klassen mit denen verschiedene Knotenelemente und Verbindungselemente dargestellt werden können. All diese Elemente können durch integrierte Funktionalitäten in den beiden Bibliotheken beliebig verbunden, verschoben, formatiert und transformiert werden.

[27] [JPowerGraph2011], Grafikbibliothek für Graphen, URL: http://sourceforge.net/projects/jpowergraph/
[28] [JPowerGraph2011], URL: http://sourceforge.net/projects/jpowergraph/

Als weitere Alternative bietet sich die Möglichkeit ähnliche Projekte zu suchen und diese bei einer gewissen Tauglichkeit den eigenen Bedürfnissen anzupassen. Die Webseite[29] „Petri Nets World" und der auf ihre enthaltene Datenbank, hat geeignete potenzielle Open Source Kandidaten, die als Grundgerüst dienen konnten. Der Vorteil an Open Source Lösungen ist, dass der Quellcode der Programme frei verfügbar ist und es ausdrücklich erlaubt ist, diese nach den eigenen Bedürfnissen zu modifizieren und anzupassen.

Eines der Open Source Programmen aus der Datenbank ist PIPE (Plattform Independent Petri-net Editor)[30]. Die Software ist 2003[31] als Gruppenprojekt von Masterstudenten des Imperial College of Science, Technology and Medicine in London entwickelt worden. In den Jahren 2004[32], 2005[33] und 2007[34] wurde PIPE in weiteren Masterprojekten derselben Universität um weitere Module ergänzt.

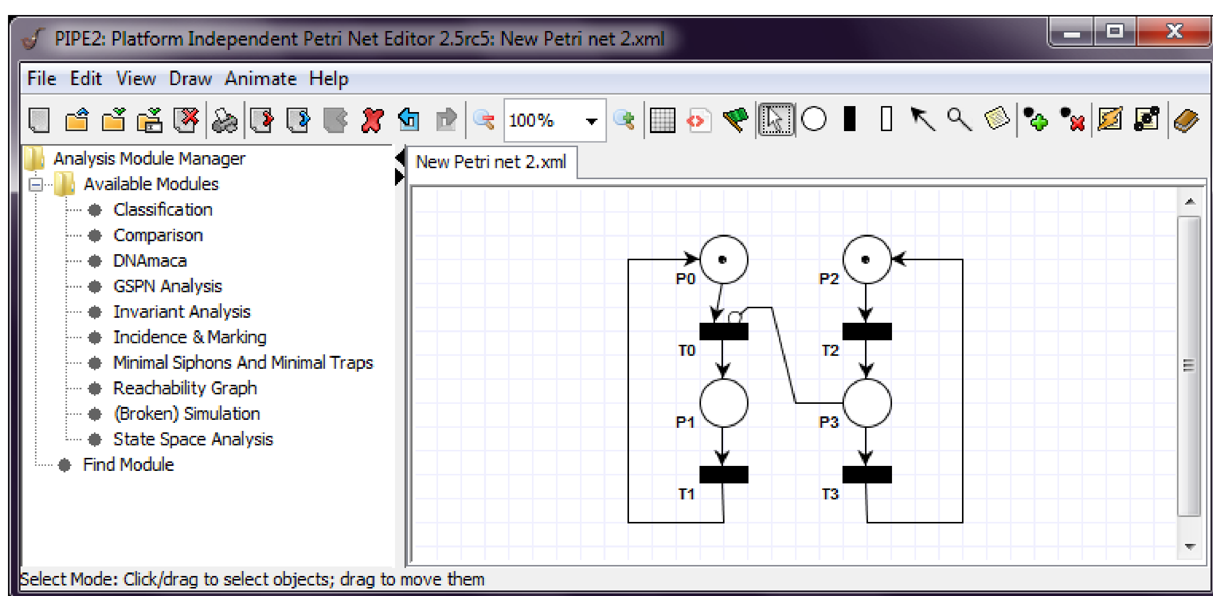


Abbildung 3-4: PIPE 2.5

PIPE, wie in Abbildung 3-4 zu sehen, erfüllt schon einige Punkte, die in Kapitel 3.1 aufgelistet sind. Es ist möglich Petrinetze grafisch zu modellieren, der grafische Editor besitzt schon Funktionalitäten wie das Zoomen, Kopieren oder Verschieben von Netzen. Was das Programm besonders attraktiv macht ist die gute Dokumentation. Nachteil der Software ist, dass es nicht für Schaltungstechnisch Interpretierte Petrinetze ausgelegt ist, sondern nur für reine Petrinetze.

[29] [PNW2010]
[30] [PIPE]
[31] [Bloom2003]
[32] [Tom2011]
[33] [Nad2005]
[34] [Edw2007]

3.3 Annahmen und Entscheidungen

Durch die vorangegangen Recherchen hat sich ein immer konkreteres Bild über den Aufwand und die Umsetzung des Projektes ergeben, welches zu Anfang des Projektes überhaupt nicht einzuschätzen war. Um dem zeitlichen und inhaltlichen Rahmen der Bachelorarbeit zu entsprechen, musste bei den ursprünglichen Zielen eine Neubewertung der Wichtigkeit vorgenommen werden.

Auf die Verwendung der grafischen Bibliotheken JGraph und JPowerGraph wird verzichtet, weil JGraph für die geplante Softwarelösung zu komplex ist und es dadurch zu langen Einarbeitungszeiten kommen würde. Auf die Verwendung von JPowerGraph wird auch verzichtet, da es für die Verwendung der Bibliothek keine ausreichende Dokumentation gibt.

Indem auf das Open Source Projekt PIPE zurückgegriffen wurde, konnte zeitökonomisch gearbeitet werden und somit das Hauptziel erreicht werden, einen grafischen Editor zu erstellen. PIPE ist ein langjährig betriebenes Projekt, welches schon durch mehrere Entwicklergruppen gelaufen ist. Um das Projekt noch überschaubar zu halten, wurden alle unnötigen Komponenten entfernt, die nicht für die Umsetzung des Projekts benötigt wurden. Zusätzlich ist im Rahmen dieser Arbeit darauf verzichtet worden, überflüssige Funktionalität umzusetzen, welcher in der Planungsphase nur eine geringe Priorität zugesprochen wurde.

Auf die Entwicklung eines Interpreter für beliebige boolesche Ausdrücke wird verzichtet, da es nicht zu den Primärzielen dieser Arbeit gehört und die Fertigstellung dieses Programms gefährden könnte. Stattdessen wird auf eine pragmatische Lösung hingearbeitet, die es erlaubt einfache konjunktive Verknüpfungen der Eingänge zu ermöglichen, welche gegebenenfalls auch negiert werden können. Die disjunktiven Verknüpfungen der Eingänge können auch durch eine Auswahlverzeigung des Netzes ermöglicht werden. Diese Lösung erlaubt es, trotz der eingeschränkten Möglichkeiten der Interpretation der Eingänge, trotzdem alles mit den gegebenen Mitteln zu ermöglichen.

Alle weiteren Ziele aus Kapitel 3.1, die mit niedriger Priorität gekennzeichnet wurden, werden aus den gleichen Gründen wie beim Interpreter, nicht berücksichtigt.

4 Aufbau und Implementierung von SipnLab

In diesem Kapitel wird auf die Entwicklungen eingegangen, die nötig waren, um die Softwarelösung SipnLab (Signal Interpreted Petri Net Laboratory) umzusetzen. Dabei werden der Aufbau des Programms und die zusätzlich entwickelte Funktionalität näher betrachtet, sowie der Aufbau der grafischen Oberfläche und dessen Gebrauch erläutert.

4.1 Die Programmstruktur

In diesem Unterabschnitt werden die Programmstruktur und Modifikationen der Entwicklung näher beschrieben. Dies ist von Nöten, da das Programm einen sehr komplexen Aufbau hat und es dem Verständnis der Entwicklung dient.

4.1.1 Grober Aufbaubau des Programms

Das Programm ist hauptsächlich in 2 Programmpakete unterteilt. Das erste Programmpaket „dataLayer" enthält alle Klassen, die mit der Programmlogik und der Datenspeicherung in Verbindung stehen. Das zweite große Programmpaket „gui" enthält alle Klassen, die für die grafische Darstellung und Bedienung des Programms verantwortlich sind.

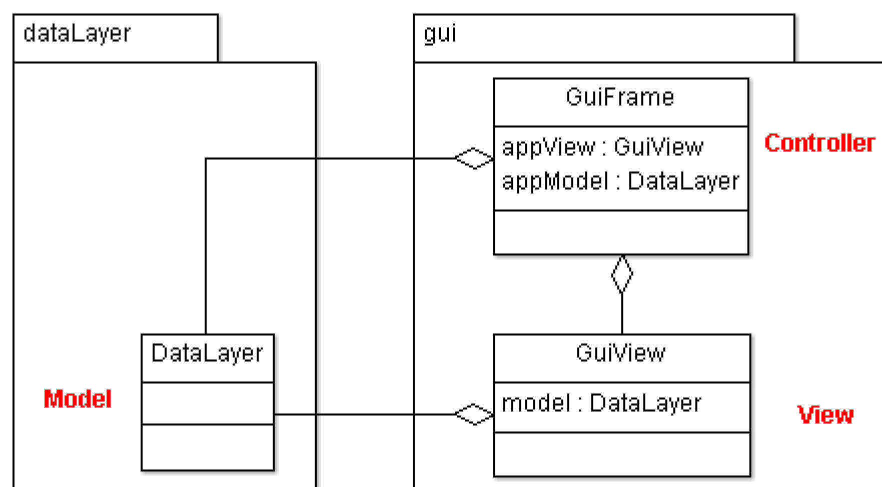


Abbildung 4-1: Strukturierung von SipnLab[35]

Betrachtet man die Struktur des Programms in Abbildung 4-1 näher, ist der Aufbau von SipnLab an das in Abschnitt 2.4 beschrieben MVC-Konzept angelehnt.

Das Programmpaket „gui" enthält weitere Subpakete, die die Struktur weiter aufgliedert. Die Programmpakete lauten:

- Im Programmpaket **action** sind Klassen enthalten, die Aktionen im grafischen Editor abfangen und weiter leiten.
- Im Programmpaket **handler** sind Klassen enthalten, die auf die Eingaben mit der Maus im grafischen Editor reagieren.

[35] Vgl. [Bloom2003], Seite 15

- Im Programmpaket **undo** sind Klassen enthalten, die die Änderungen an grafischen Elementen im Editor speichern. Diese Klassen werden benötigt, um Veränderungen rückgängig machen zu können durch betätigen der „zurück"-Funktion.
- Im Programmpaket **widgets** sind Klassen enthalten, welche die Fenster beschreiben, die zusätzlich zum Hauptfenster benötigt werden.

4.1.2 Datenmodell

Beim Datenmodell von SipnLab wird die Strukturierung der Daten in 2 Arten unterschieden. Für den internen Aufbau des Programms ist es wichtig die Petrinetze in einer Klassenhierarchie unterzubringen.

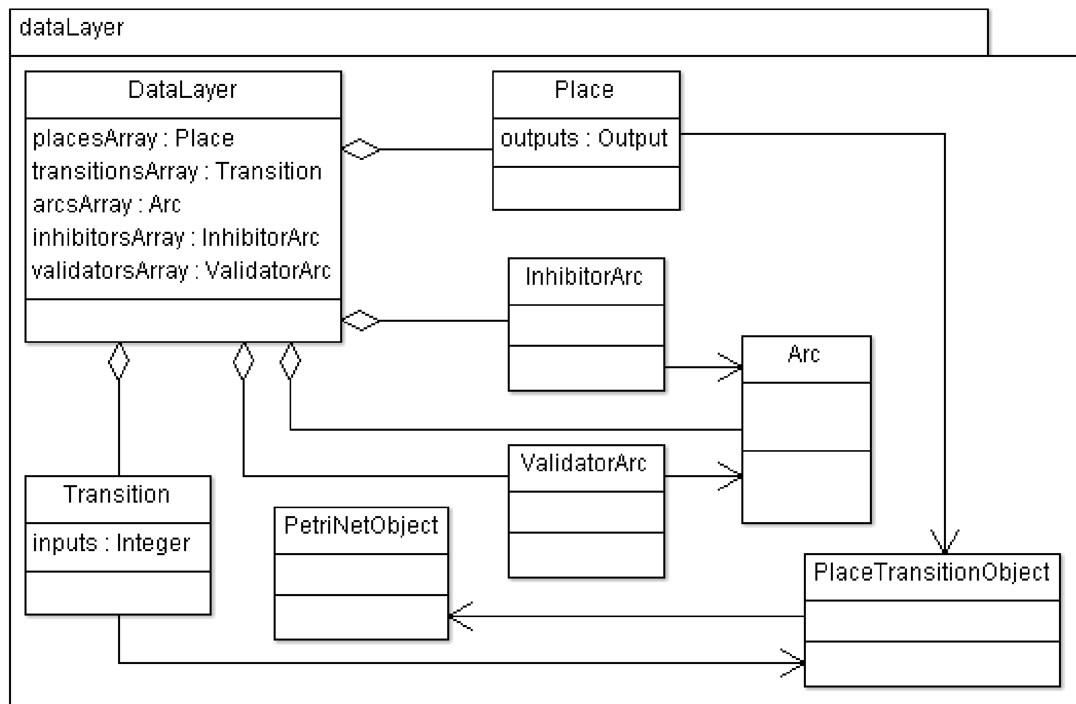


Abbildung 4-2: Klassenhierarchie des Programmpacket „dataLayer"

Durch diese Struktur, welche in Abbildung 4-2 abgebildet ist, werden alle Daten, die während der Laufzeit des Programms erzeugt werden, in einer Instanz von der Klasse „DataLayer" gespeichert und können durch die implementierten Methoden weiter verarbeitet werden.

Als Schnittstelle für die Verwendung von anderen verarbeitenden Teilen des Programms werden die im „DataLayer“ gesammelten Daten in ein mathematisches Modell, wie in Kapitel 2.1.2.2 beschrieben, umgewandelt.

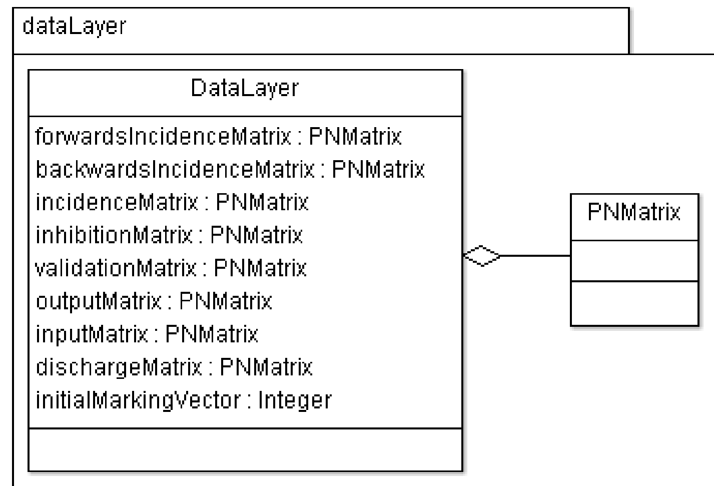


Abbildung 4-3: Speicherung des mathematisches Modell in SipnLab

4.1.3 Speichern und Laden von SIPN

Die Daten der Netze werden in SipnLab in XML-Dateien gespeichert. Zu diesem Zweck werden die SIPN in eine Baumstruktur überführt.

Für die Verarbeitung von XML wird in diesem Projekt die „Java API for XML Processing“ (JAXP[36]) verwendet. Diese Bibliothek ist seit der Version 1.4 von Java enthalten.

Diese Transformation zum Speichern wird von der Klasse „dataLayerWriter“ und zum Laden von der Methode „createFromPNML()“, welche sich in der Klasse „dataLayer“ befindet, implementiert.

Der Aufbau der XML-Baumstruktur wird anhand vom Beispielnetz von Abbildung 4-4 demonstriert.

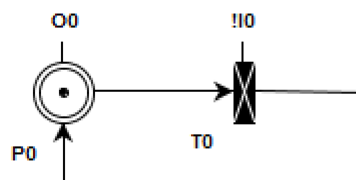


Abbildung 4-4: Beispiel SIPN für XML-Transformation

Zu beachten ist, dass die Darstellung im Listing 4-1, zur Steigerung der Übersichtlichkeit, leicht vereinfacht wurde. Die Vereinfachung wurde bewerkstelligt, indem z. B. einige Elemente, wie die Koordinaten der SIPN-Objekte, vernachlässigt wurden. Die Struktur des XML-Baums ist an die Petri Net Markup Language[37] (PNML) angelehnt, welche nach ISO/IEC 15909 standarisiert ist. Der Aufbau der XML Datei sieht so aus, dass zuerst ein PNML-Element erzeugt wird, dieses

---

36 [SUN2011]
37 [PNML2011]

enthält das Net-Element mit den Attributen Id und Type. Das Net-Element enthält alle Elemente des SIPN.

Die Reihenfolge der Elementtypen ist wichtig für die Rekonstruktion des Netzes. So ist es erforderlich, dass die Ein- und Ausgänge vor allen anderen Elementen eingelesen werden, damit die Abhängigkeiten die im weiteren Verlauf durch die Verknüpfung der Ein- und Ausgänge mit den Stellen und Transitionen vorhanden sind, erfasst werden können. Die Abhängigkeiten der Stellen und Transitionen mit den Kanten verhalten sich genauso. Die Stellen und Transitionen müssen vor den Kanten eingelesen werden, damit beim generieren des Modells aus der XML-Datei, die Abhängigkeiten erfasst werden können.

Listing 4-1: Vereinfachte Transformation eines SIPN in eine XML-Baumstruktur

```
1  <?xml version="1.0" encoding="iso-8859-1"?>
2  <pnml>
3      <net id="Net-One" type="SIPN">
4          <input id="I0" name="I0">
5          </input>
6          <output id="O0" name="O0">
7          </output>
8          <place id="P0">
9              <name>
10                 <value>P0</value>
11             </name>
12             <initialMarking>
13                 <value>1</value>
14             </initialMarking>
15             <io id="O0"/>
16         </place>
17         <transition id="T0">
18             <name>
19                 <value>T0</value>
20             </name>
21             <timed>
22                 <value>false</value>
23             </timed>
24             <io id="!I0"/>
25             <discharge id="P0"/>
26         </transition>
27         <arc id="P0 to T0" source="P0" target="T0">
28             <type value="normal"/>
29         </arc>
30         <arc id="T0 to P0" source="T0" target="P0">
31             <type value="normal"/>
32         </arc>
33     </net>
34 </pnml>
```

4.2 Hinzugefügte Funktionalität

Im diesem Unterabschnitt wird die zusätzliche Funktionalität und Modifikationen beschrieben, die im Verlauf der Entwicklung dieses Programms umgesetzt wurde.

4.2.1 Ein- und Ausgänge

Die Entwicklung der Funktionalität der Ein- und Ausgänge war eines der wichtigsten Elemente für das Programm. Es wurden die Klassen „Input“ und „Output“ dem Strukturkonzept entsprechend im Paket „dataLayer“ erstellt. Die Klassen „Input“ und „Output“ erben von „PetriNetObject“ in gleicher Weise die Klasse „PlaceTransitionObject“. Die Klassen „Place“ und „Transition“ erben von „PlaceTransitionObject“. Diese Vererbungen sind notwendig, um die Ein- und Ausgänge, genauso wie alle anderen Elemente, mit dem Programm zu verweben und das bestehende Verarbeitungsmethoden auch auf die neuen Elemente angewendet werden können. Die Vererbungen sind im UML-Diagramm mit Pfeilen dargestellt. Die Pfeile, mit rautenförmigen Enden, stellen Assoziationen dar. Die Objekte, auf die sie zeigen, können Objekte enthalten, von denen sie ausgehen. Die Assoziationen werden benötigt um den Zusammenhang zwischen den Objekten, wie in Abbildung 4-5 dargestellt, zu erfassen.

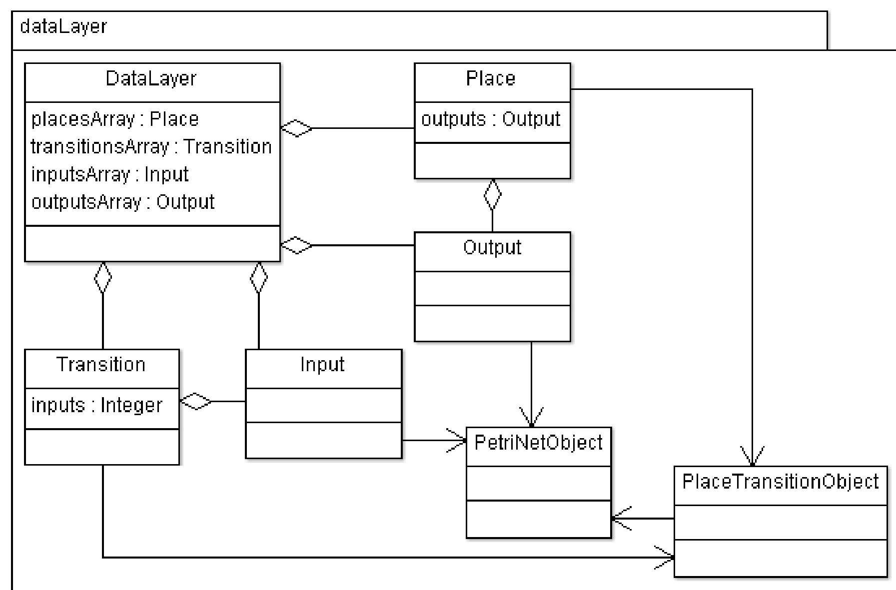


Abbildung 4-5: UML-Diagramm der Anbindung der Ein- und Ausgänge

Mithilfe der Verknüpfungen, die bereits erläutert wurden, ist es nun möglich ein Modell zu erstellen. Bei der Erstellung des Modells wurde auf dasselbe Prinzip wie bei den übrigen Komponenten zurückgegriffen, um eine nahtlos Integration zu ermöglichen. Die Integration der Ein- und Ausgänge in das Modell wird an einem Beispielnetzes betrachtet, welches in Abbildung 4-6 dargestellt wird.

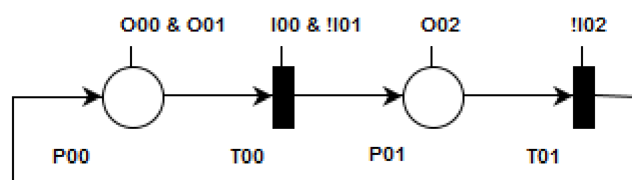


Abbildung 4-6: Beispielnetz für Ein- und Ausgänge

Die Ausgänge, die mit den Stellen verknüpft sind, werden in einer Tabelle hinterlegt. Die Tabelle ist so aufgebaut, dass die Zeilen die Stellen mit P für „Place“ und die Spalten die Ausgänge mit O für „Output“ repräsentieren. Bei der Verknüpfung von Stellen mit Ausgängen wird an der entsprechenden Stelle der Tabelle/Matrix eine „1“ vermerkt.

Tabelle 3: Beispiel für eine Ausgangsmatrix

	O00	O01	O02
P00	1	1	0
P01	0	0	1

Bei den Eingängen und Transitionen verhält sich die Tabelle analog zu den Ausgängen, allerdings können die Eingänge invertiert sein. Ein invertierter Eingang wird dadurch gekennzeichnet, dass in die betreffende Zelle eine „-1“ eingetragen wird.

Tabelle 4: Beispiel für eine Eingangsmatrix

	I00	I01	I02
T00	1	-1	0
T01	0	0	-1

Dadurch ist eine kompatible Form der Datenaufbereitung für das Programm geschaffen worden, mit der auch eingeschränkte schaltalgebraische Ausdrücke verarbeitet werden können.

4.2.1.1 Grafische Darstellung der Eingänge

Die Eingänge werden, wie oben beschrieben, durch eine Linie an den Transitionen dargestellt. Das Problem dabei ist, dass die Transitionen rotiert werden können und somit die Linie im selben Winkel um den Mittelpunkt mit der Transition gedreht werden müsste.

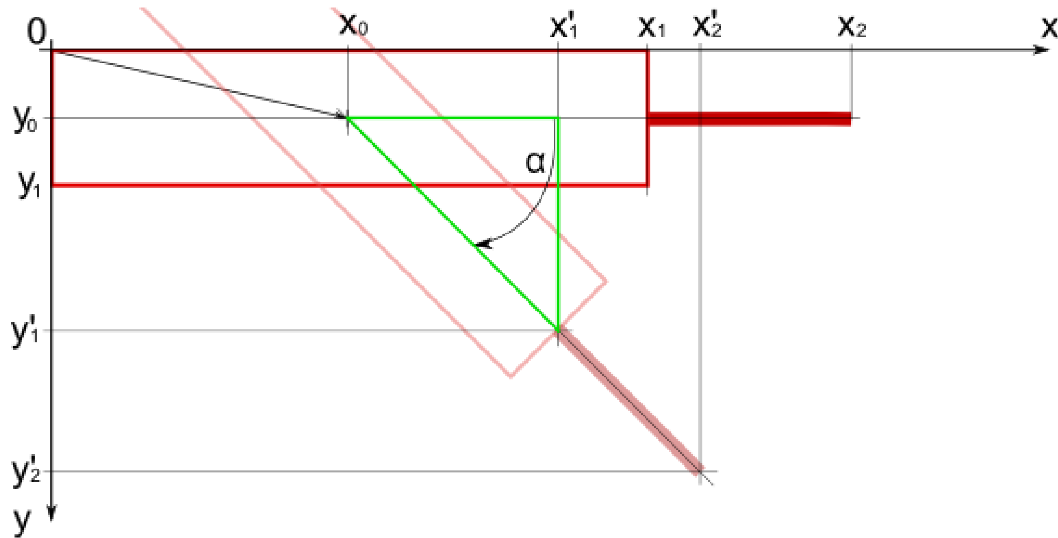


Abbildung 4-7: Rotieren eines Eingangs mit der Transition

In Java werden grafische Objekte so beschrieben, dass die linke obere Ecke den Ursprung eines relativen Koordinatensystems und damit die Position eines Objektes beschreibt. Dreht man nun die Transition im Mittelpunkte um den Winkel α, muss die Linie auch um diesen Winkel gedreht werden. Dieses Problem wird gelöst, indem eine Koordinatentransformation in Abhängigkeit des Objektwinkels durchgeführt wird, wie in Abbildung 18 gezeigt.

Um eine Transformation durchführen zu können, wird zuerst der Mittelpunkt des Objekts bestimmt, um den die Rotation stattfindet. Dafür werden die Koordinaten des Mittelpunktes mit

$$x_0 = 0{,}5 * x_1 = 0{,}5 * Objektbreite \qquad \text{Formel 4-1}$$
$$y_0 = 0{,}5 * y_1 = 0{,}5 * Objekthöhe \qquad \text{Formel 4-2}$$

bestimmt. Nun werden mit der Hilfe der Trigonometrie die rotierten Punkte der Linie bestimmt. Dafür wird ein rechtwinkliges Dreieck (Abbildung 18 in grün) konstruiert, welches durch die Koordinaten x_0, y_0, x_1'und y_1' beschrieben werden kann. Somit kann man durch Anwenden der Trigonometrie die transformierten Koordinaten

$$x_1' = x_0 + x_0 * \cos \propto \qquad \text{Formel 4-3}$$
$$y_1' = y_0 + y_0 * \sin \propto \qquad \text{Formel 4-4}$$

bestimmen. Die Koordinaten für den Endpunkt werden analog wie für den Anfangspunkt bestimmt, dabei ist jedoch der Punkt um die Linienlänge weiter verschoben. Damit erhält man die Gleichungen

$$x_2' = x_0 + [x_0 + (Linienlänge)] * \cos \propto \qquad \text{Formel 4-5}$$
$$y_2' = y_0 + [y_0 + (Linienlänge)] * \sin \propto \qquad \text{Formel 4-6}$$

für die Transformation der Endpunkte.

4.2.1.2 Grafische Darstellung der Ausgänge

Die Ausgänge werden, wie bereits beschrieben, auch durch eine Linie an den Stellen dargestellt. Das Problem bei den Stellen ist, dass um eine möglichst flexible Anbindung der Kanten zu gestatten, die Ausgänge in eine beliebige Position an der Stelle rotiert werden müssen.

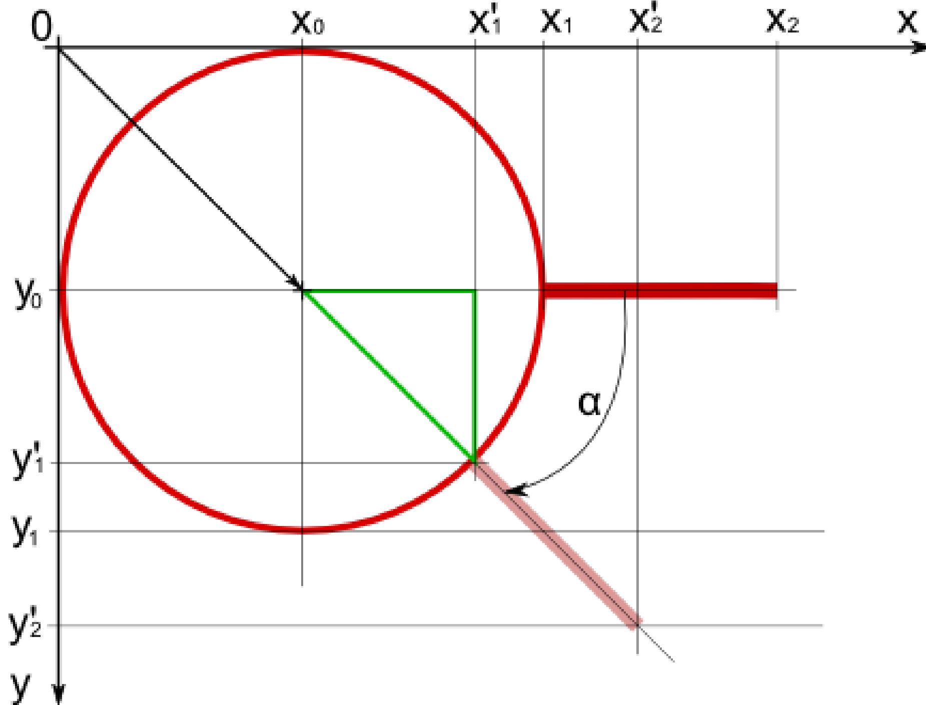


Abbildung 4-8: Rotieren eines Ausgangs an einer Stelle

Genau wie bei der Transition weiter oben beschrieben, ist der Ursprung in der oberen linken Ecke des Koordinatensystems, indem sich das Objekt befindet. Das Problem wird genauso gelöst, wie in Kapitel 4.2.1.1 zuvor, indem die Koordinaten des Mittelpunkts des Objekts, welches in Abbildung 4-8abgebildet ist, durch die Formeln

$$x_0 = 0{,}5 * x_1 = 0{,}5 * Objektdurchmesser \quad \text{Formel 4-7}$$

$$y_0 = 0{,}5 * y_1 = 0{,}5 * Objektdurchmesser \quad \text{Formel 4-8}$$

bestimmt werden. Anschließend werden wieder die transformierten Koordinaten

$$x'_1 = x_0 + x_0 * \cos \propto \quad \text{Formel 4-9}$$

$$y'_1 = y_0 + y_0 * \sin \propto \quad \text{Formel 4-10}$$

$$x'_2 = x_0 + [x_0 + (Linienlänge)] * \cos \propto \quad \text{Formel 4-11}$$

$$y'_2 = y_0 + [y_0 + (Linienlänge)] * \sin \propto \quad \text{Formel 4-12}$$

bestimmt.

4.2.1.3 Zuweisen von Ein- und Ausgängen in SipnLab

In SipnLab werden die Ein- und Ausgänge global definiert. Dies geschieht im Hauptfenster des Programms, wie in Abbildung 4-9 abgebildet, indem einer der „add"-Buttons betätigt wird. Der obere Button fügt Eingänge und der untere Button für Ausgänge hinzu.

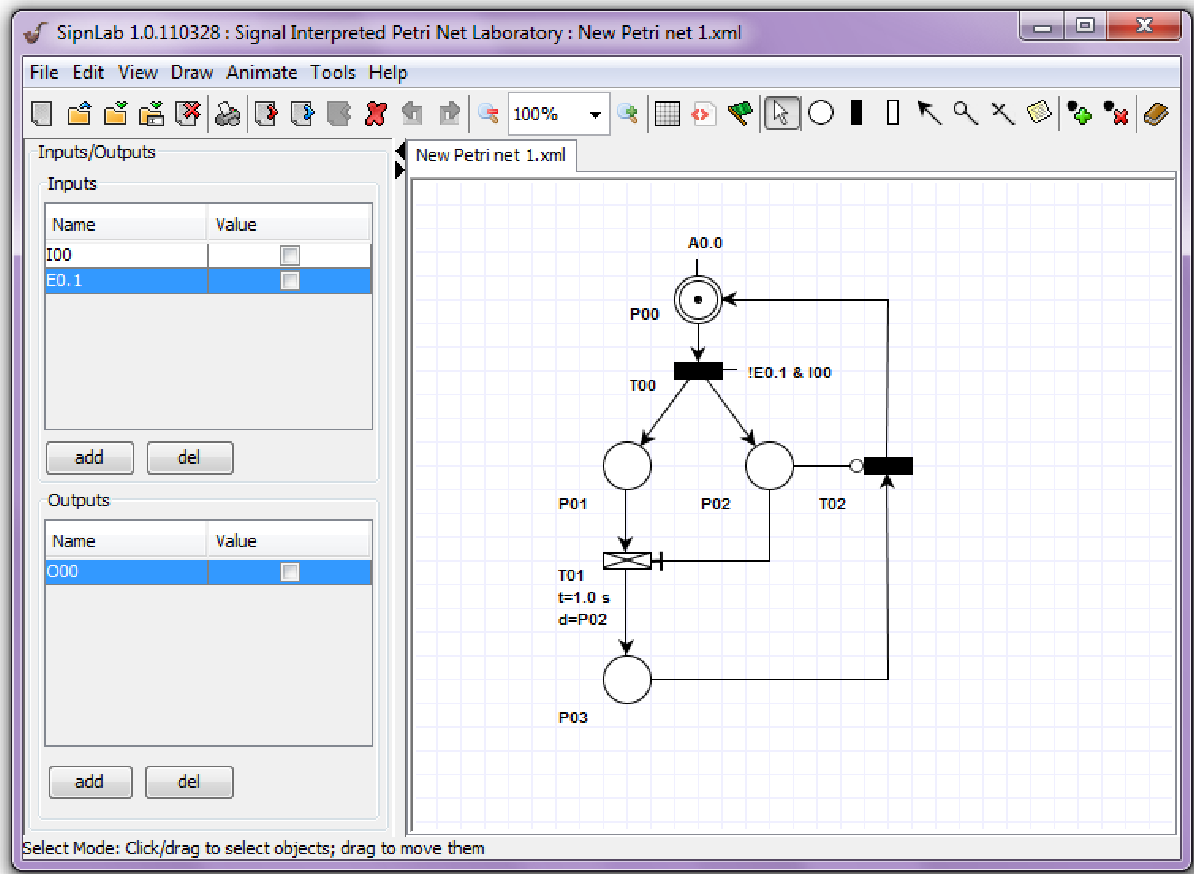


Abbildung 4-9: SipnLab Hauptfenster

Die so erzeugten Ein- und Ausgänge werden mit einer eineindeutigen ID (Ident) im „dataLayer" hinterlegt. Die Bezeichnung der Elemente ist bei der Erzeugung dieselbe wie die ID, kann aber nach Belieben verändert werden.

Die Zuweisung der Ein- und Ausgänge zu den entsprechenden Stellen und Transitionen, wird durch die Platz- und Transitionseditoren getätigt. Diese Editoren werden aufgerufen indem auf die betreffende Knotenelement ein Doppelklick ausgeführt wird. Im „Place Editor“, wie in Abbildung 4-10, können die Ausgänge der ausgewählten Stelle verknüpft werden.

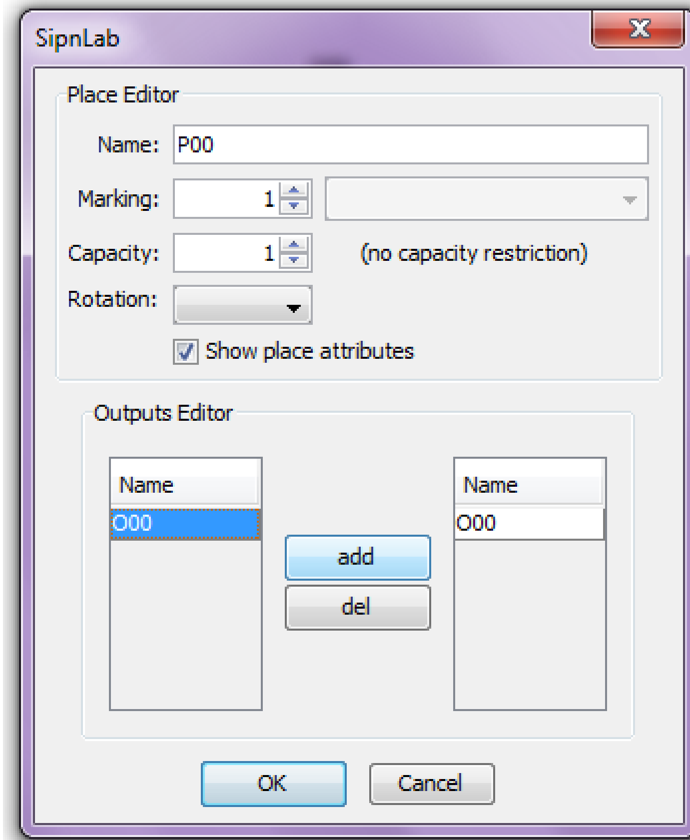


Abbildung 4-10: Darstellung des Place Editors

Dazu wird im Feld „Outputs Editor“ der Ausgang mit der Maus ausgewählt und mit dem „add-Button“ hinzugefügt. Mit dem ersten Hinzufügen eines Ausganges wird automatisch die grafische Darstellung der Stelle angepasst.

Der „Transition Editor“, wie in Abbildung 4-11 abgebildet, wird dazu verwendet, um die Eingänge mit der Transition zu verknüpfen. Dazu werden im Feld „Inputs Editor“ die gewünschten Eingänge ausgewählt und mit dem Button „add“ der Transition zugeordnet.

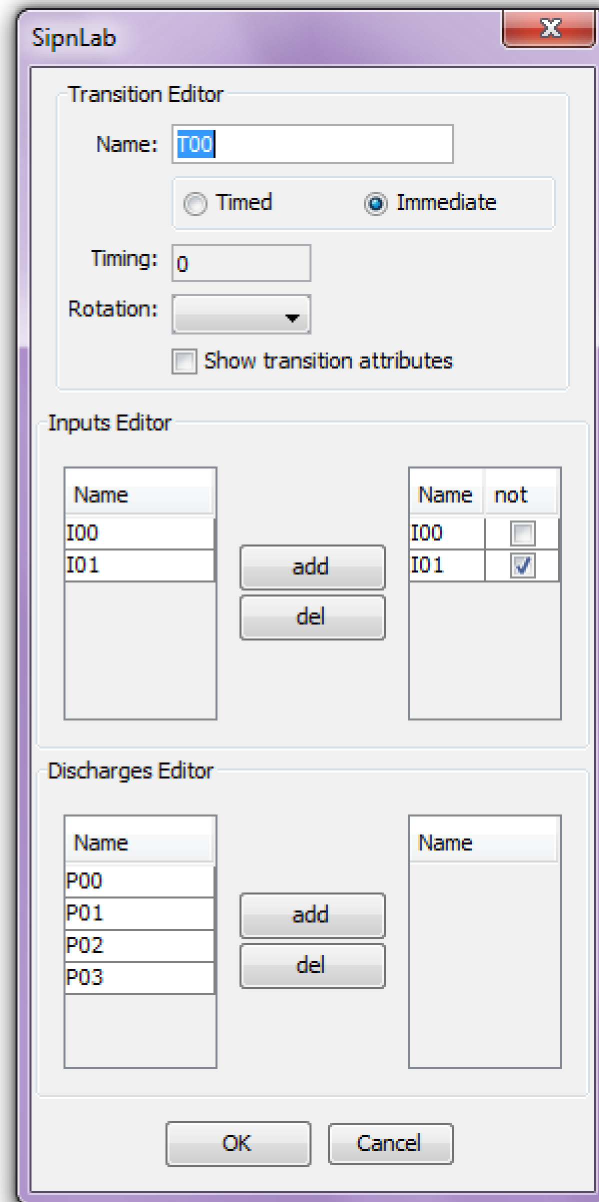


Abbildung 4-11: Darstellung des Transition Editors

Die so zugeordneten Eingänge können über das Kästchen neben der Bezeichnung in der Zuweisungstabelle invertiert werden. Die Eingänge, die in die Zuweisungstabelle mit der Transition verknüpft werden, sind untereinander konjunktiv verbunden. Die Schaltbedingung $q_{T_{00}}$, die sich für die Transition T_{00} dadurch ergibt, lautet

$$q_{T_{00}} = I_{00} \wedge \overline{I_{01}} \; .$$

An der Transition wird die Schaltbedingung des Eingangs automatisch in Form einer Zeichenkette „I00 & !I01“ dargestellt.

4.2.2 Testkanten

Die Umsetzung der Testkanten wurde verwirklicht, indem das schon existierende Element der Inhibitorkante kopiert und entsprechend modifiziert wurde. Die Modifikationen entsprechen zum größten Teil der veränderten Funktionalität und der grafischen Darstellung. In der Klasse „ValidatorArc“ sind folgende Modifikationen gemacht worden.

Listing 4-2: Auszug aus ValidatorArc.java

```
1  public final static String type = "validator";
2  private final static Polygon head = new Polygon(new int[]{0, 1, 1, 6, 6, -6, -6, -1, -1},
3                                                   new int[]{0, 0, -6, -6, -8, -8, -6, -6, 0},
4                                                9);
```

Die erste Zeile setze den Typ des Objekts auf „validator“ fest. In der zweiten Zeile sind die Daten der grafischen Darstellung enthalten, die durch ein Polygon definiert werden. Die Klasse „Polygon“ bekommt die X-Punkte, Y-Punkte und die Anzahl der Koordinaten übergeben.

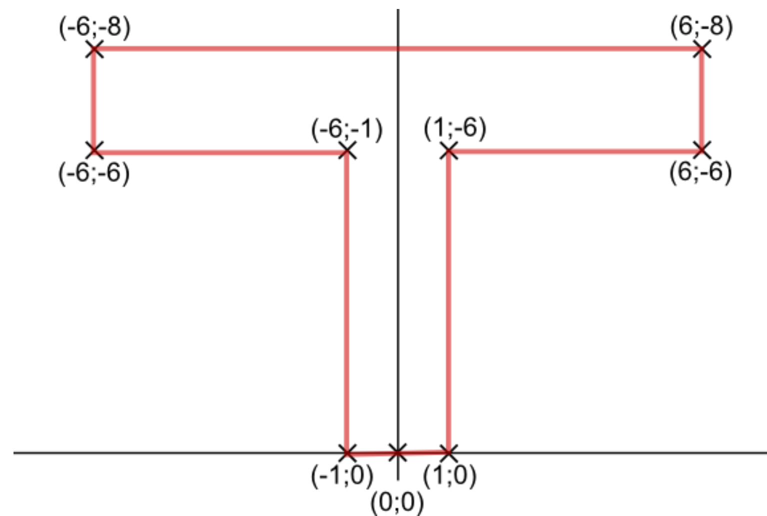


Abbildung 4-12: Grafische Darstellung einer Testkante

Aus den der Klasse „Polygon“ übergeben Daten wird die in Abbildung 4-12 dargestellte Grafik erzeugt, welches die Spitze einer Testkante repräsentiert.

4.2.3 Löschtransitionen

Die Funktion des Löschens steht nicht im Konflikt mit der Zeittransition, weil eine Transition, die eine zeitliche Verzögerung zum Feuern braucht, immer noch zum Löschen der Marken verwendet werden kann. Aus diesem Grund wurde die Klasse der Transition mit der Option der Löschung ergänzt. Das hat zur Folge, dass alle Transitionen das Löschen von Marken beherrschen, indem in einem Array in den Transitionen referenziert wird, welche Marken im Falle eines Schaltens gelöscht werden.

Durch diese Modifikation ergeben sich zwei unterschiedliche Formen der Darstellung, welche in Abbildung 4-13 gezeigt werden.

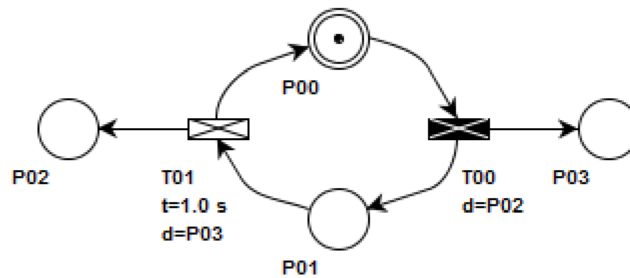


Abbildung 4-13: Beispiel für Löschtransitionen

Für das Modell wird bei den Löschtransitionen, wie bei den vorherigen Elementen, auf eine Matrixdarstellung zurückgegriffen. Die Spalten der Matrix stehen für die Stellen P und die Zeilen für die Transitionen T. Für das Beispiel in Abbildung 4-13 wurde nun in Tabelle 5 eine Matrix aufgestellt.

Tabelle 5: Beispiel für eine Löschmatrize

	P00	P01	P02	P03
T00	0	0	1	0
T01	0	0	0	1

Durch die Löschmatrix wird beschrieben, welche Stellen beim Schalten der Transitionen zurückgesetzt werden.

Grafische Darstellung der Löschtransitionen

Die Löschtransitionen werden, wie in Abbildung 4-14 gezeigt, durch ein X gekennzeichnet. Die Kennzeichnung wird bei der Rotation der Transition mit gedreht. Um dieses zu ermöglichen, wird auch hier eine Transformation der Koordinaten durchgeführt.

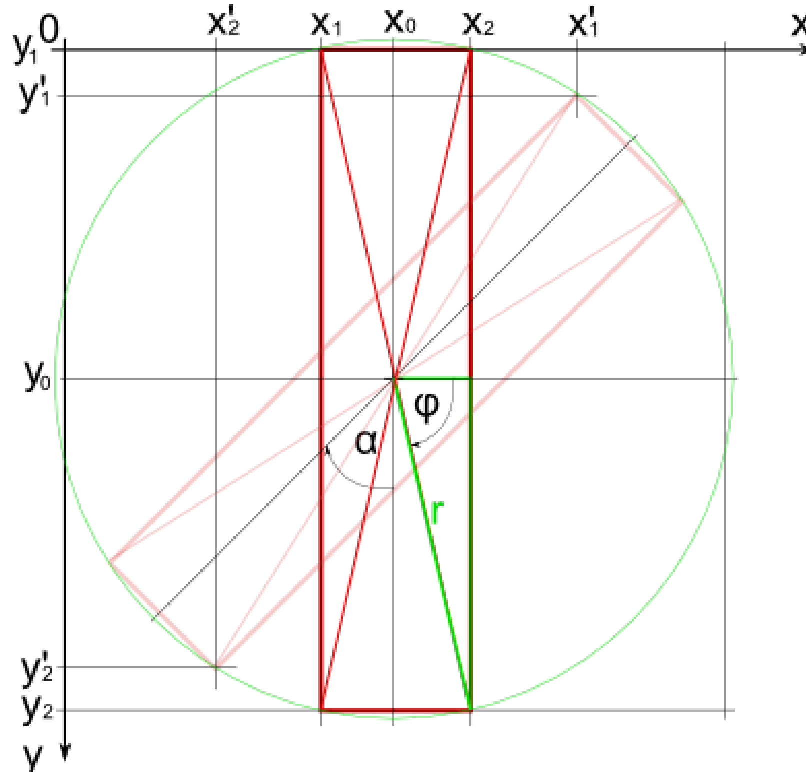


Abbildung 4-14: Darstellung und Rotation der Löschtransition

Als Ausgangspunkt dient der Mittelpunkt des Objekts, welcher durch x_0 und y_0 beschrieben werden.

$$x_0 = 0{,}5 * Objektbereite \qquad \text{Formel 4-13}$$

$$y_0 = 0{,}5 * Objektdhöhe \qquad \text{Formel 4-14}$$

Die Rotation der Kennzeichnung wird beschrieben, indem die Eckpunkte der Transition auf einem gedachten Kreis verschoben werden. Für die Berechnung der Punkte wird mit der Trigonometrie der Winkel φ und der Radius r eines rechtwinkligen Dreiecks bestimmt.

$$\varphi = \tan^{-1}\left(\frac{\cancel{0{,}5} * Transitionshöhe}{\cancel{0{,}5} * Transitionsbreite}\right) \qquad \text{Formel 4-15}$$

$$r = 0{,}5 * \frac{Transitionshöhe}{\sin\varphi} \qquad \text{Formel 4-16}$$

Mit diesen beiden Parametern können die Ecken der Transition um einen beliebigen Winkel α auf dem gedachten Kreises beschrieben werden.

$$x'_1 = x_0 + r * \cos(\propto + \varphi + \pi) \qquad \text{Formel 4-17}$$

$$y'_1 = y_0 + r * \sin(\propto + \varphi + \pi) \qquad \text{Formel 4-18}$$

$$x'_2 = x_0 + r * \cos(\propto - \varphi) \qquad \text{Formel 4-19}$$

$$y'_2 = y_0 + r * \sin(\propto - \varphi) \qquad \text{Formel 4-20}$$

4.2.4 Übersetzungstabelle

Die Übersetzungstabelle bzw. Zuweisungstabelle wird benötigt, um die SIPN-Elemente in SipnLab den Elementen der SPS, wie in Kapitel 2.3.6 beschrieben wurde, zuzuweisen. Die Zuweisungstabelle wurde in der Klasse „SPSTranslationPanel" realisiert. Die Funktion der Zuweisungstabelle wird anhand des Netzes demonstriert, welches in Abbildung 4-15 dargestellt ist.

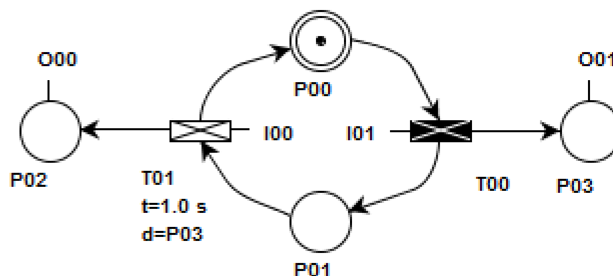


Abbildung 4-15: Beispielnetz für Übersetzungstabelle

In Abbildung 4-16 ist die Übersetzungstabelle für das Beispielnetz in Abbildung 4-15 dargestellt. Die linke Spalte der Tabelle enthält die Bezeichnungen der SipnLab-Elemente. Der mittleren Spalte werden die Elemente der SPS zugewiesen, wobei nur den Zeittransitionen ein Timer zugeordnet werden kann. In der rechten Spalte können die Elemente mit Kommentaren versehen werden.

SipnLab

SipnLab	SPS	comment
I00	E0.0	
I01	E0.1	
O00	A0.0	
O01	A0.1	
P00	M0.0	
P01	M0.1	
P02	M0.2	
P03	M0.3	
T00	---	
T01	T1	

OK Cancel

Abbildung 4-16: Beispiel für Übersetzungstabelle

Das Bestätigen der Tabelle mit dem „OK"-Button erzeugt mit der Hilfe des Codegenerators, welcher im nächsten Abschnitt beschrieben wird, eine Text-Datei mit dem AWL-Code, die im selben Verzeichnis wie die XML-Datei des Netzes gespeichert wird.

4.2.5 AWL-Codegenerator

Ein weiterer zentraler Punkt bei der Umsetzung von SipnLab ist die Entwicklung des AWL-Codegenerators. Dieser ist im Groben, wie in Abbildung 21 gezeigt, in 3 Abschnitte unterteilt.

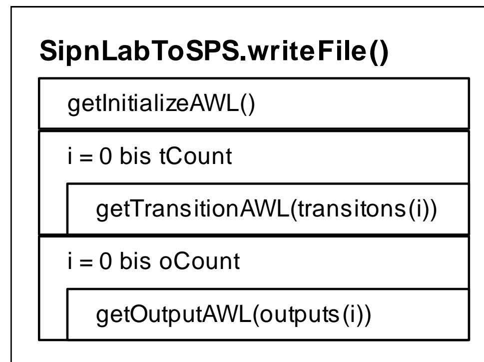


Abbildung 4-17: Struktogramm des AWL-Codegenerators

Im ersten Abschnitt wird aus dem Datenmodell die Initialisierung des SIPN in AWL-Code umgesetzt. Dabei wird für OB1 und OB100 die AWL-Syntax erzeugt. Im zweiten Teil wird der AWL-Code für die Transitionen erzeugt und im dritten Teil der Quellcode für die Ausgänge.

Initialisierung

Der Initialisierungsteil ist nur eine reine Abfrage, ob die Stelle einen Initialisierungsflag hat. Dieser Teil erzeugt den Initialisierungscode für OB100 und alternativ für OB1 gleichzeitig. Die SPS führt OB1 zyklisch aus und deswegen muss ein Merker definiert werden, der für die Initialisierung zuständig ist. AWL-Code den SipnLab erzeugt, wird der Merker M36.0 vorgegeben.

Listing 4-3: Die Methode getInitializeAWL

```
1   public String getInitializeAWL() {
2           String awl = "";
3           awl += "//===> Choose one of initialize blocks\r\n";
4           awl += "//===> Please choose this block for OB100\r\n";
5           for (int i = 0; i < pCount; i++) {
6                   if (places.get(i).getInitialMarking()>0
7                           awl += "S " + places.get(i).getComment1() + "\r\n";
8           }
9           awl += "//===> OB100 block end\r\n";
10          awl += "//===> Please choose this block for OB1\r\n";
11          awl += "UN " + "M36.0" + "//choose a unused flag(e.g. M36.0)\r\n";
12          awl += "S " + "M36.0" + "//choose same flag as above(e.g. M36.0)\r\n";
13          for (int i = 0; i < pCount; i++) {
14                  if (places.get(i).getInitialMarking()>0)
15                          awl += "S " + places.get(i).getComment1() + "\r\n";
16          }
17          awl += "//===> OB1 block end\r\n";
18          return awl;
19  }
```

Transitionen

Die Methode „getTransitionAWL“, wie im Listing 4-4 gezeigt, erzeugt den AWL-Code der Transition, die ihr zuvor übergeben wurde.

Die Zeilen 6 bis 15 erzeugen, durch Auswertung des Modells, den Code, der durch die Kanten die Stellen abfragt.

Die Zeilen 18 bis 23 erzeugen aus der Eingangsmatrix die Abfragen der Schaltbedingungen.

Die Zeilen 26 bis 30 fragen die übergebene Transition ab, ob sie eine Zeittransition ist. Bei einer positiven Bestätigung wird der dazu gehörige Code erzeugt.

Die Zeilen 33 bis 37 prüfen, ob die Transition eine Löschtransition ist. Bei erfolgreicher Prüfung wird der Code erzeugt, der die Stellen beim Schalten der Transition deaktiviert.

Die Zeilen 40 bis 45 erzeugen aus dem Modell den Code, welcher die Stellen vor und hinter der schaltenden Transition beeinflusst.

Listing 4-4: Die Methode getTransitionAWL

```
1   public String getTransitionAWL(Transition transition) {
2           int trNo = transitions.indexOf(transition);
3           String awl = "";
4   
5           // places
6           for (int i = 0; i < pCount; i++) {
7                   if (backwardsIncidenceMatrix.get(i, trNo) != 0)
8                           awl += "U " + places.get(i).getComment1() + "\r\n";
9                   if (forwardsIncidenceMatrix.get(i, trNo) != 0)
10                          awl += "UN " + places.get(i).getComment1() + "\r\n";
11                  if (validationMatrix.get(i, trNo) != 0)
12                          awl += "U " + places.get(i).getComment1() + "\r\n";
13                  if (inhibitionMatrix.get(i, trNo) != 0)
14                          awl += "UN " + places.get(i).getComment1() + "\r\n";
15          }
16  
17          // inputs
18          for (int i = 0; i < iCount; i++) {
19                  if (inputMatrix.get(trNo, i) != 0)
20                          awl += "U " + inputs.get(i).getComment1() + "\r\n";
21                  else
22                          awl += "UN " + inputs.get(i).getComment1() + "\r\n";
23          }
24  
25          // timer
26          if (transition.isTimed()) {
27                  awl += "L S5T#" + transition.getRate() + "s" + "\r\n";
28                  awl += "SE " + transition.getComment1() + "\r\n";
29                  awl += "U " + transition.getComment1() + "\r\n";
30          }
31  
```

```
32          // discharge
33          if (transition.getDischargeargeSize()>0) {
34                  for (int i = 0; i < transition.getDischargeargeSize(); i++) {
35                          awl += "R " + transition.getDischarge(i).getComment1() + "\r\n";
36                  }
37          }
38
39          // token
40          for (int i = 0; i < pCount; i++) {
41                  if (backwardsIncidenceMatrix.get(i, trNo) != 0)
42                          awl += "R " + places.get(i).getComment1() + "\r\n";
43                  if (forwardsIncidenceMatrix.get(i, trNo) != 0)
44                          awl += "S " + places.get(i).getComment1() + "\r\n";
45          }
46          return awl;
47  }
```

Ausgänge

Die Ausgänge werden am Ende des AWL-Codes erzeugt. Dafür wird die „outputMatrix“ untersucht, ob der übergebene Ausgang mit Stellen verknüpft ist. Für die verknüpften Stellen werden disjunktive Abfragen generiert, welche im Ergebnis den nicht speichernden Ausgang setzen.

Listing 4-5: Die Methode getOutputAWL

```
1   public String getOutputAWL(Output output) {
2           int outNo = outputs.indexOf(output);
3           int temp = 0;
4
5           String awl = "";
6           for (int i = 0; i < pCount; i++) {
7                   if (outputMatrix.get(i, outNo) != 0) {
8                           awl += "O " + places.get(i).getComment1() + "\r\n";
9                           temp++;
10                  }
11          }
12          if (temp > 0)
13                  awl += "= " + outputs.get(outNo).getComment1() + "\r\n";
14          return awl;
15  }
```

4.3 Diverse Modifikationen

Das Hinzufügen von Funktionalitäten erfordert diverse Modifikationen am Grundgerüst der Softwarelösung.

4.3.1 Animationsmodus

Der Animationsmodus ermöglicht es, den Markenfluss eines Netzes Stritt für Stritt zu durchlaufen. Für die Erweiterung des Ursprünglichen Programms durch weitere Elemente, ist es notwendig, den Animationsmodus an die Gegebenheiten anzupassen. Der Animator wird gestartet, indem der Animationsbutton aktiviert wird, wie in Abbildung 4-18. Bei Aktivierung werden Transitionen, die schalten können, durch eine Einfärbung in Rot gekennzeichnet. Die Transitionen die so gekennzeichnet sind, können durch einen Klick mit der linken Maustaste gefeuert werden. Der Verlauf der gefeuerten Transitionen wird im Fenster „Animation history" protokolliert.

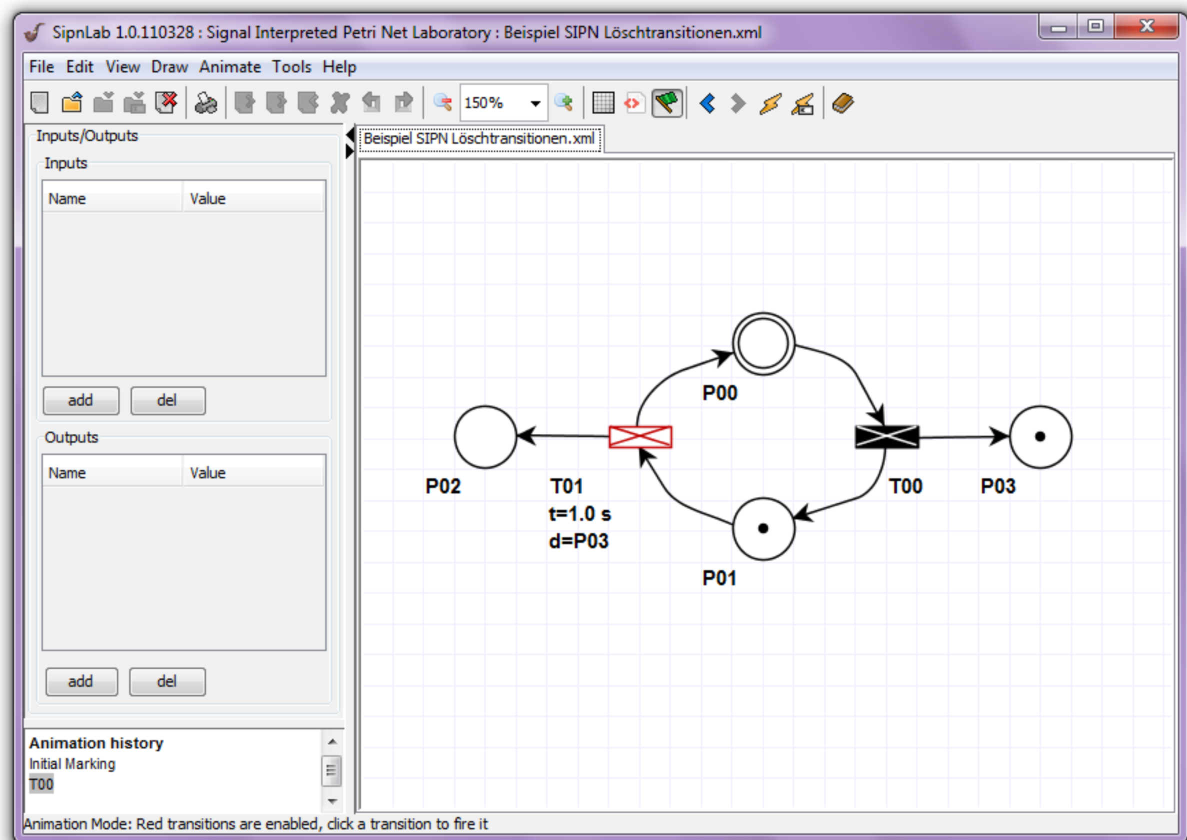


Abbildung 4-18: SipnLab Animationsmodus

Der Animationsmodus wurde dahingehend modifiziert, dass beim Schalten von Transitionen mit Löschfunktionalität, die entsprechenden Marken gelöscht werden und dass Testkanten das Schalten von Transitionen verhindern, falls die Stelle mit der sie dadurch verknüpft ist, nicht mit einer Marke besetzt ist.

Die für die Löschtransition benötigte Modifikation, wurden in der Methode „fireTransition“ in der Klasse „dataLayer“ ergänzt. Die Modifikation ist im Listing 4-6 dargestellt. Der Code prüft beim Feuern einer Transition, ob sie mit zu löschenden Stellen verknüpft ist. Im Falle einer positiven Prüfung werden die entsprechenden Stellen zurückgesetzt.

Listing 4-6: Modification an dataLayer.fireTransition()

```
1  if(transition.getDischargeargeSize()>0){
2          for(int i = 0; i < transition.getDischargeargeSize(); i++){
3                  int placeNo = placesArray.indexOf(transition.getDischarge(i));
4                  ((Place)placesArray.get(placeNo)).setCurrentMarking(0);
5          }
6  }
```

Eine weitere Modifikation für die Funktionalität der Testkanten ist in der Mehtode „getTransitionEnabledStatusArray“ erstellt worden, welche sich in der Klasse „dataLayer“ befindet. Im Listing 4-7 ist der dafür erforderliche Code dargestellt. Dieser setzt den Schaltstatus der Transition auf „false“, wenn die Stelle, die mit der Transition durch die Testkante verknüpft ist, nicht mit einer Marke besetzt ist.

Listing 4-7: Modification an dataLayer.getTransitionEnabledStatusArray()

```
1  if (validation[j][i] > 0 && marking[j] < validation[j][i]) {
2          result[i] = false;
3          break;
4  }
```

4.3.2 Sortierung der SIPN-Elemente

In SipnLab werden alle Elemente nach ihrer Bezeichnung sortiert. Einerseits werden die Elemente sortiert, damit die Tabellen die im Programm existieren übersichtlich bleiben, anderseits wird damit das Problem der Prioritäten der Transitionen gelöst. Die Sortierung hat auch zur Folge, dass der AWL-Codegenerator die Transitionen in der richtigen Reihenfolge generieren kann.

Die Sortierung wurde umgesetzt, indem dafür eine spezielle Komparator-Klasse erzeugt wurde, welche im Listing 4-8 gezeigt wird.

Listing 4-8: Darstellung der Klasse PetriNetObjectComparatorName

```
1  public class PetriNetObjectComparatorName  implements Comparator<PetriNetObject>
2  {
3          public int compare( PetriNetObject o1, PetriNetObject o2 )
4          {
5             return o1.getName().compareTo(o2.getName());
6          {
7  }
```

Mit dieser Klasse lassen sich alle Arrays sortieren, die Objekte enthalten, dessen Klasse von PetriNetObject abgeleitet ist. Das Sortieren übernimmt die von der Java Application Programming Interface (Java API) zur Verfügung gestellten Sortieralgorithmen.

4.3.3 Weitere Modifikationen

Weitere Modifikationen waren nötig, um die Funktionalität von SipnLab zu gewährleisten. Einige von ihnen dienten dazu Fehler auszumerzen, die schon in PIPE vorhanden waren, andere hingegen dazu, um die Funktionalität des Programms zu erweitern. Diese Modifikationen sind im Rahmen dieser Arbeit beiläufig gemacht worden und werden daher nicht im Detail analysiert und erläutert. Der Vollständigkeit halber werden an dieser Stelle erwähnt.

- Das Öffnen von gespeicherten Netzen hatte eine Deformation der Darstellung zufolge, weil die Netzelemente zum Teil dabei verschoben wurden.
- Beim Zoomen des Netzes wurde das Raster nicht an die veränderten Größenverhältnisse mit angepasst.
- Der Eingangs- und Ausgangsteil am linken Bildschirmrand musste dem Datenmodell des aktiven Netzes zugewiesen werden.

5 Experimentelle Verifikation von SipnLab

In diesem Kapitel wird die entwickelte Softwarelösung vorgestellt und anhand eines Beispiels die Funktionalität verifiziert.

5.1 Erstellen eines beispielhaften SIPN

Die Funktionalität von SipnLab wird anhand des Beispiels analysiert, welches in Abbildung 5-1 dargestellt ist. Dieses SIPN wurde so modelliert, dass das erstellte Programm auf all seine Funktionalitäten untersucht werden kann.

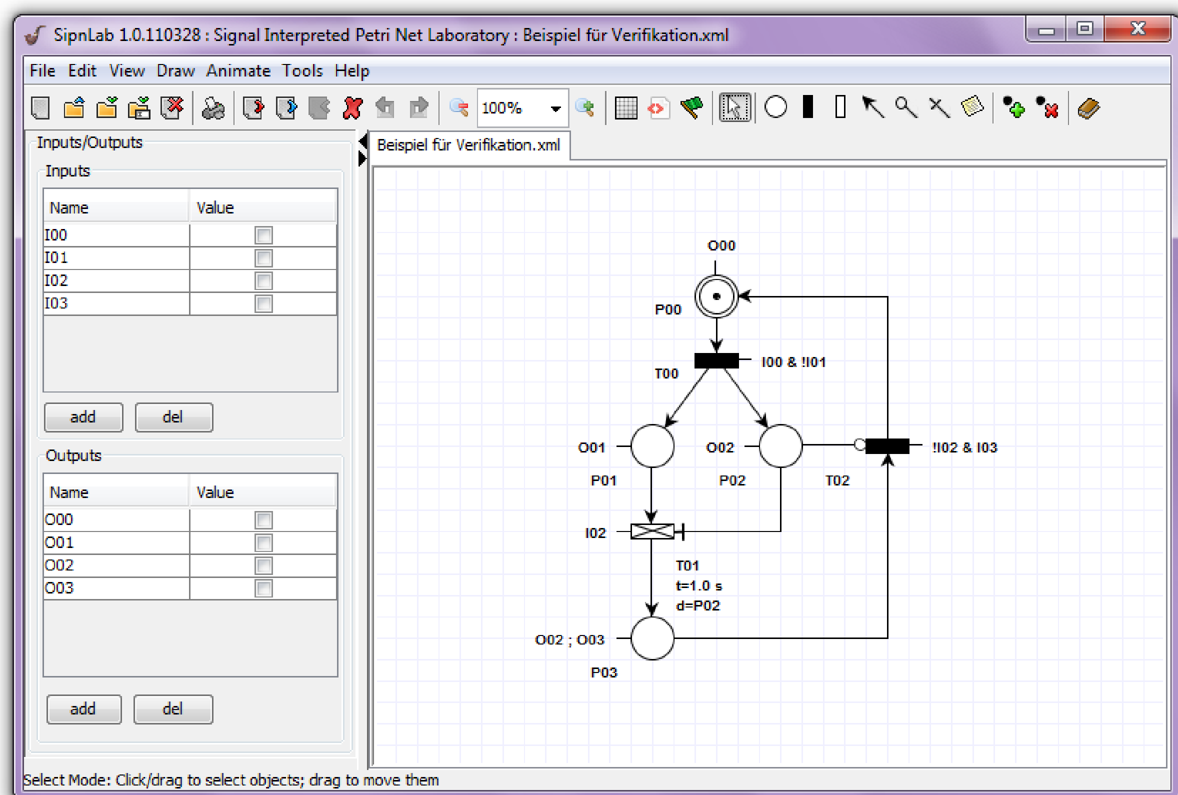


Abbildung 5-1: Beispiel für Verifikation

5.2 Verifikation des Datenmodells mit der Theorie

Die Verifikation des Datenmodells wird überprüft, indem die Matrizen des Netzes, wie in Abschnitt 2.2.2.2 und die Erweiterungen in Abschnitt 4.2 beschrieben, mit den Ergebnissen im Debugger der Entwicklungsumgebung Eclipse verglichen wird. Am Beispiel, des in Abbildung 5-1 dargestellten Netzes, werden die theoretischen Matrizen erstellt. Hierfür werden die Vorwärtsinzidenzmatrix $\boldsymbol{I}^{+}$ nach Formel 2-7 und Rückwärtsinzidenzmatrix $\boldsymbol{I}^{-}$ nach Formel 2-8 gebildet.

$$\boldsymbol{I}^{+} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}; \qquad \boldsymbol{I}^{-} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Aus der Differenz der beiden Matrizen, wird die Inzidenzmatrix $\boldsymbol{I}$ nach Formel 2-9 berechnet.

$$\boldsymbol{I} = \boldsymbol{I}^{+} - \boldsymbol{I}^{-} = \begin{pmatrix} -1 & 0 & 1 \\ 1 & -1 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{pmatrix}$$

Der Markierungsvektor $\boldsymbol{p}(0)$ lautet nach Formel 2-4:

$$\boldsymbol{p}(0) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Die Matrizen der Testkanten $\boldsymbol{V}$ und der Inhibitorkanten $\boldsymbol{N}$ lauten nach Abschnitt 2.3.5:

$$\boldsymbol{V} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad \boldsymbol{N} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

Die Matrizen der Eingänge $\boldsymbol{E}$ und der Ausgänge $\boldsymbol{A}$ lauten nach Abschnitt 4.2.1:

$$\boldsymbol{E} = \begin{pmatrix} 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \qquad \boldsymbol{A} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

Die Matrix der Löschtransitionen $\boldsymbol{D}$ lautet nach Abschnitt 4.2.3:

$$\boldsymbol{D} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Die im Datenmodell von SipnLab erzeugten Matrizen werden mit dem Debugger von Eclipse beim Ausführen von SipnLab ausgelesen. Die zusammengestellten Ausgaben des Debuggers sind im Listing 5-1 dargestellt.

Listing 5-1: Auszug des Datenmodells beim Debuggen von SipnLab

```
1  backwardsIncidenceMatrix
2  [[1, 0, 0], [0, 1, 0], [0, 0, 0], [0, 0, 1]]
3  
4  discargeMatrix
5  [[0, 0, 0, 0], [0, 0, 1, 0], [0, 0, 0, 0]]
6  
7  forwardsIncidenceMatrix
8  [[0, 0, 1], [1, 0, 0], [1, 0, 0], [0, 1, 0]]
9  
10 incidenceMatrix
11 [[-1, 0, 1], [1, -1, 0], [1, 0, 0], [0, 1, -1]]
12 
13 inhibitionMatrix
14 [[0, 0, 0], [0, 0, 0], [0, 0, 1], [0, 0, 0]]
15 
16 initialMarkingVector
17 [1, 0, 0, 0]
18 
19 inputMatrix
20 [[1, -1, 0, 0], [0, 0, 1, 0], [0, 0, -1, 1]]
21 
22 outputMatrix
23 [1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 1, 1]]
24 
25 validationMatrix
26 [[0, 0, 0], [0, 0, 0], [0, 1, 0], [0, 0, 0]]
```

Zur Verifikation werden die Ergebnisse des Debuggers mit den theoretisch errechneten Werten verglichen. Dabei ist festzustellen, dass keine Unterschiede der beiden Ergebnisse vorliegen und somit das Programm in dieser Hinsicht richtig arbeitet.

5.3 Verifikation der Funktion des Animationsmodi

Der Animationsmodus wird überprüft, indem das Verhalten des SIPN, welches in Abbildung 5-1 abgebildet ist, durch die in Abschnitt 2.2 und 2.3 beschrieben Schaltbedingungen der Transitionen, überprüft wird. Zu diesem Zweck werden die drei möglichen Schaltschritte, die mit diesem Netz möglich sind, mit den Schaltverhalten des Animators verglichen. Die demnach zu erwarteten Schritte wären:

Schritt 1: Die Transition T_{00} schaltet, wenn die Stelle P00 mit einer Marke besetzt ist und die Stellen P01 und P02 nicht mit Marken besetzt sind. Die Marke wird von P00 abgezogen und in P01 und P02 erzeugt.

Schritt 2: Die Transition T01 schaltet, wenn die Stellen P01 und P02 mit Marken besetzt sind und die Stelle P03 nicht mit einer Marke besetzt ist. Beim Feuern der Transition wird die Marke von P01 abgezogen und eine Marke in P03 erzeugt. Zusätzlich wird durch die Löschfunktionalität der Transition die Marke von der Stelle P02 gelöscht.

Schritt 3: Die Transition T02 schaltet, wenn die Stelle P3 mit einer Marke besetzt ist und die Stellen P02 und P00 nicht mit Marken besetzt sind. Beim Schalten wird die Marke von P3 nach P00 transferiert.

Der Übergang vom Zustand 1 (Abbildung 5-2) zum Zustand 2 (Abbildung 5-3) deckt sich mit dem Verhalten des Netzes, welches im Schaltschritt 1 beschrieben wurde.

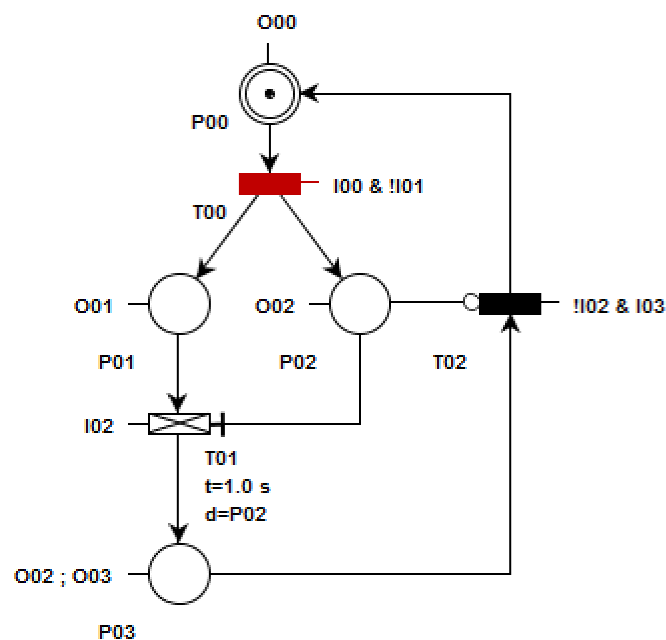


Abbildung 5-2: Zustand 1 im Animationsmodus

Der Übergang vom Zustand 2 (Abbildung 5-3) zum Zustand 3 (Abbildung 5-3) deckt sich mit dem Verhalten des Netzes, welches im Schaltschritt 2 beschrieben wurde.

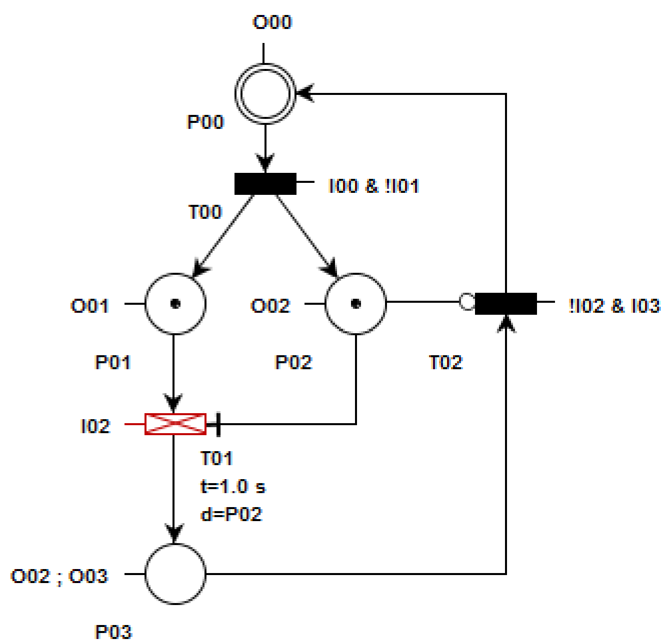


Abbildung 5-3: Zustand 2 im Animationsmodus

Der Übergang vom Zustand 3 (Abbildung 5-4) zum Zustand 1 (Abbildung 5-2) deckt sich mit dem Verhalten des Netzes, welches im Schaltschritt 3 beschrieben wurde.

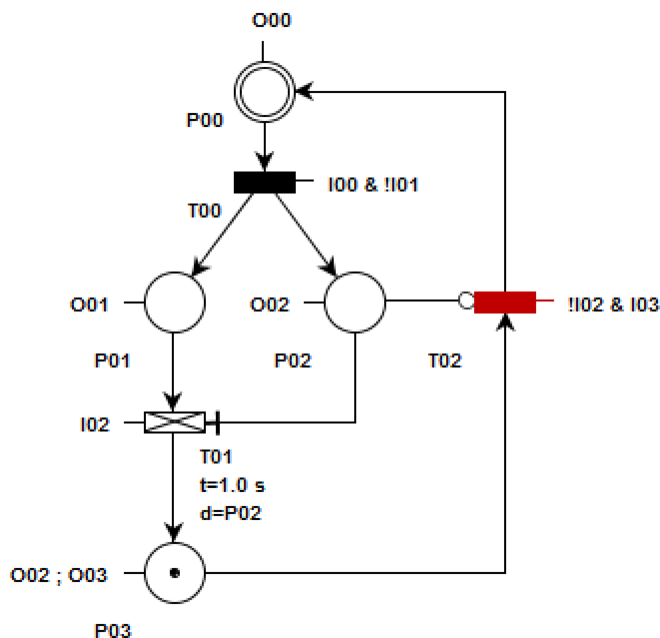
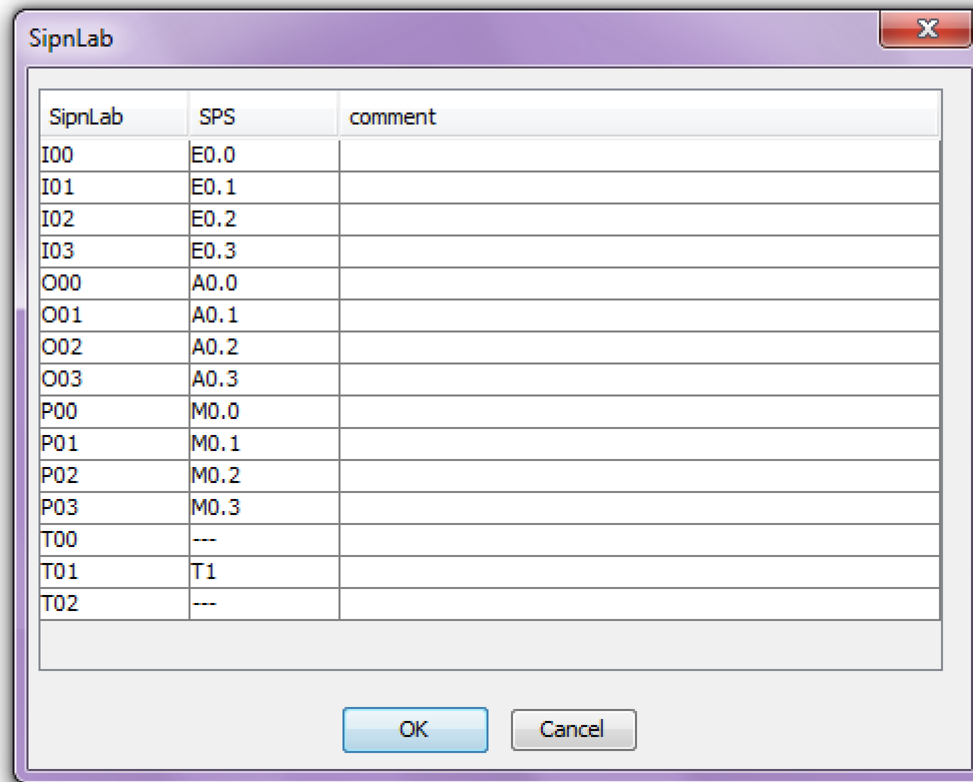


Abbildung 5-4: Zustand 3 im Animationsmodus

Der Vergleich zeigt, dass alle 3 Schaltschritte des theoretisch erwarteten Verhaltens, mit dem 3 Schaltschritten von SipnLab übereinstimmen.

5.4 Generieren des AWL-Codes mit dem Codegenerator

Damit der Code geriert werden kann, müssen den Elementen von SipnLab die Elemente der SPS zugewiesen werden. Die Zuweisungstabelle, welche in Abbildung 5-5 dargestellt ist, wird aufgerufen, indem man im Hauptfenster auf „Tools“ klickt und dort den „AWL Translator“ ausführt.



SipnLab	SPS	comment
I00	E0.0	
I01	E0.1	
I02	E0.2	
I03	E0.3	
O00	A0.0	
O01	A0.1	
O02	A0.2	
O03	A0.3	
P00	M0.0	
P01	M0.1	
P02	M0.2	
P03	M0.3	
T00	---	
T01	T1	
T02	---	

Abbildung 5-5: Zuweisungstabelle des Beispiels

In der Zuweisungstabelle werden die für die automatische Übersetzung nötigen Zuweisungen gemacht.

Durch Betätigung des „OK“-Button wird der AWL-Code automatisch erzeugt.

5.5 Verifikation des generierten AWL-Codes

Der vom Codegenerator automatisch erzeugte AWL-Code, welcher in Kapitel 5.4 erzeugt wurde, wird überprüft, indem der AWL-Code Schritt für Schritt mit dem in Kapitel 2.3.6 beschrieben Codierungsvorschriften verglichen wird. Zur besseren Übersicht wird der Code in drei Teile geteilt.

Der erste Teil enthält die Initialisierung des generierten AWL-Quellcodes. Die Analyse des im Listing 5-2 dargestellten Codes ergibt, dass die Initialisierungen des SIPNs korrekt transformiert wurden.

Listing 5-2: Initialisierungsteil der generierten AWL-Code

```
1   //This Code is generated with SipnLab 1.0.110328 Codegenerator
2
3   //Initialize
4   //===> Choose one of initialize blocks
5   //===> Please choose this block for OB100
6   S M0.0
7   //===> OB100 block end
8   //===> Please choose this block for OB1
9   UN M36.0//choose a unused flag(e.g. M36.0)
10  S M36.0//choose same flag as above(e.g. M36.0)
11  S M0.0
12  //===> OB1 block end
```

Der mittlere Teil der Codeausgabe enthält die Codierung der Transitionen. Der Vergleich der drei Transitionen und ihrer Funktionalitäten mit dem im Listing 5-3 dargestellten Code zeigt, dass der Teil korrekt übersetzt wurde.

Listing 5-3: Transitionsteil des generierten AWL-Code

```
1   //Insert all below in OB1
2   // T00
3   U M0.0
4   UN M0.1
5   UN M0.2
6   U E0.0
7   UN E0.1
8   R M0.0
9   S M0.1
10  S M0.2
11
12  // T01
13  U M0.1
14  U M0.2
15  UN M0.3
16  U E0.2
17  L S5T#1.0s
18  SE T1
```

```
19  U T1
20  R M0.2
21  R M0.1
22  S M0.3
23
24  // T02
25  UN M0.0
26  UN M0.2
27  U M0.3
28  UN E0.2
29  U E0.3
30  S M0.0
31  R M0.3
```

Der dritte Teil der Codeausgabe, welche im Listing 5-4 dargestellt ist, zeigt beim Vergleich der Verknüpfungen der Stellen mit den Ausgängen, dass der Code auch hier korrekt übersetzt wurde.

Listing 5-4: Ausgangsteil des generierten AWL-Code

```
1   //Outputs
2   // O00:
3   O M0.0
4   = A0.0
5
6   // O01:
7   O M0.1
8   = A0.1
9
10  // O02:
11  O M0.2
12  O M0.3
13  = A0.2
14
15  // O03:
16  O M0.3
17  = A0.3
```

Alle drei Teile sind korrekt übersetzt worden und somit ist die korrekte Funktion des Codegenerators von SipnLab bewiesen.

5.6 Demonstration eines komplexeren Netzes in SipnLab

Die Funktionalität von SipnLab ist soweit gegeben, dass auch komplexere Netze modelliert werden können. Dies wird anhand einer Torsteuerung demonstriert, welche in Abbildung 5-6 dargestellt ist.

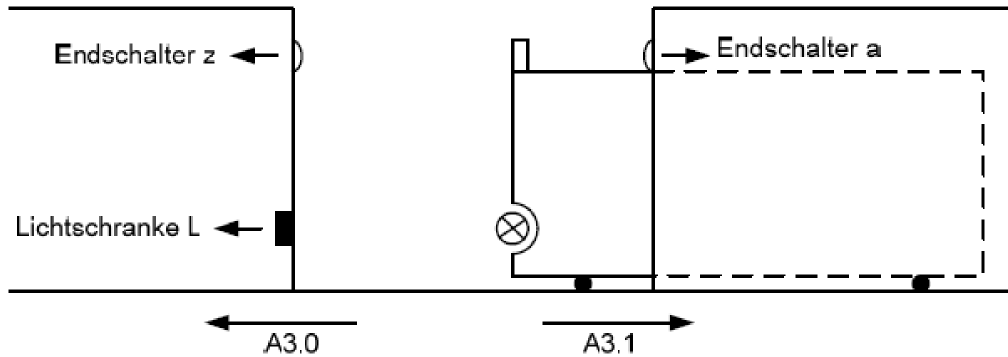


Abbildung 5-6: Torsteuerung[38]

Die Steuerungslösung für das Beispiel ist in Abbildung 5-7 gezeigt.

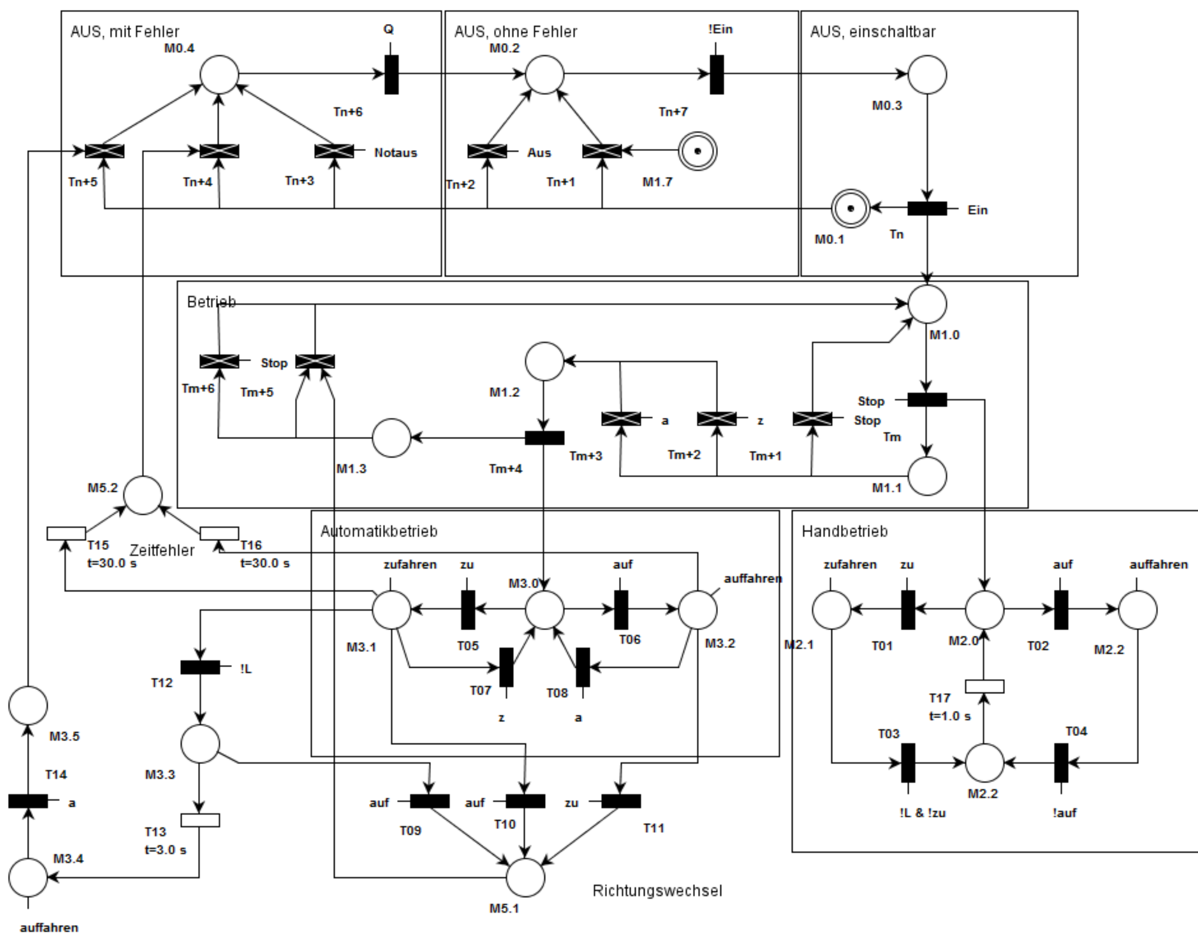


Abbildung 5-7: Steuerungslösung der Torsteuerung[39]

[38] [Meiners2009], Folie 179

[39] Vgl. [Meiners2009], Ergänzung zur Folie 179

6 Schluss

6.1 Zusammenfassung

Ziel dieser Arbeit war es eine Softwarelösung zur grafischen Modellierung von SIPN zu erstellen, welches automatisiert AWL- Quellcode für eine SPS der Siemens S7-Serie generiert.

Zu diesem Zweck wurde die Aufgabenstellung weiter verfeinert, indem die Hauptaufgabe weiter unterteilt wurde. Die Intention dieser Aufteilung war es, ein ingenieurmäßiges Herangehen der Aufgabe zu ermöglichen. Durch die Analyse und Priorisierung der Teilaufgaben wurde eine Herangehensweise zum Lösen der Aufgabenstellung gebildet.

Für die Entwicklung des geplanten grafischen Editors wurde nach einer Recherche und Analyse der Umsetzungsvarianten dafür entschieden, als Basis für den Editor, auf das schon existierende Projekt PIPE zurückzugreifen. Nachdem PIPE von den nicht weiter benötigten Teilen und zum Teil vorhanden Altlasten befreit wurde, begann die Modifikation und Erweiterung der Basis. Das Programm wurde um die Funktionalitäten ergänzt, deren Priorität in der Planungsphase mit „hoch“ und zum Teil mit „mittel“ bewertet wurden.

Ein Highlight der Entwicklungen war die Umsetzung des AWL-Codegenerators. Hierfür wurde das mathematische Modell, welches durch die Daten des modellierten Netzes vom Programm erzeugt wird, in AWL-Code transformiert.

Als Letztes wurde die Funktionalität der Softwarelösung an einem Beispiel verifiziert. Dies geschah, indem die theoretisch zu erwartenden Ergebnisse mit den praktisch im Programm erzeugten Werten verglichen wurden.

6.2 Fazit

Mit dieser Arbeit konnte eine Softwarelösung erzeugt werden, die es ermöglicht grafisch SIPN zu modellieren und automatisch in AWL-Code zu transformieren. Zusätzlich ist es möglich, von modellierten Netzen, Schritt für Schritt den Markenfluss des Netzes im Animationsmodus zu verfolgen.

Abstriche mussten bei der Interpretation der booleschen Ausdrücke für die Schaltbedingungen der Eingänge gemacht werden, welche für disjunktive Verknüpfungen durch Erweitern des Netzes um eine Auswahlverzweigung realisiert werden muss.

Trotz der Einschränkungen sind viele Funktionalitäten von SipnLab soweit ausgereift, dass es sich auch in der Praxis einsetzen lässt.

Damit wurden die Primärziele und die meisten Sekundär- und Tertiärziele erreicht, die zu Beginn dieser Arbeit gesteckt wurden.

6.3 Ausblick

In Zukunft sollte dieses Projekt fortgesetzt werden, weil es sich als gut und ausbaufähig herausgestellt hat. Um dieses sicher zu stellen habe ich mich entschieden nach Abschluss dieser Arbeit das Projekt Interessierten als Open Source unter http://sipnlab.sourceforge.net/ zur Verfügung zu stellen, in der Hoffnung, dass es dadurch zur Verbesserung und Erweiterung des Programmes kommt.

Folgende Erweiterungen für SipnLab sind in Zukunft denkbar:

- Die Erweiterung des Programms mit der Fähigkeit erstellte SIPN zu simulieren.
  - Die transiente Simulation des Netzes durch die Vorgabe der Eingangssignale durch ein Impulsdiagramm.
- Das Erstellen weiterer Codegeneratoren, welche zusätzliche Sprachen für SPS und andere Plattformen wie Mikrocontroller und FPGA (Field Programmable Gate Array) integrieren.
- Das Hinzufügen weiterer SIPN Elemente und die Möglichkeit vom Gebrauch von Sub-Netzen bzw. Hierarchien.
- Das Erweitern von SipnLab um einen vollwertigen Interpreter für booleschen Ausdrücke, wodurch die möglichen Verknüpfungen der Eingangsbedingungen von Transitionen erhöht werden.
- Das Fertigstellen der Hilfefunktion von SipnLab, welches Neueinsteigern den Umgang mit SipnLab vereinfacht.
- Anbindung an MatLab oder LabVIEW mit der Integration von OPC (OLE[40] for Process Control) Protokoll. Durch die Erweiterung können Prozessmodelle gesteuert werden, welche in den beiden Programmen erstellt sind.

Die meisten aufgezählten Erweiterungen wären möglich, ohne in die Struktur des Programms eingreifen zu müssen, da alle erforderlichen Daten, welche die Module benötigen, durch das Datenmodell zur Verfügung gestellt werden.

[40] Object Linking and Embedding

6.4 Persönliche Bemerkungen und Danksagungen

Die Arbeit an diesem Projekt hat sich als sehr langwierig und komplex herausgestellt und ich möchte hier an dieser Stelle zuerst meiner Familie und Freundin danken, dass sie mich in dieser schwierigen Zeit seelisch und physisch unterstützt haben.

Außerdem möchte ich noch Prof. Dr. Ulfert Meiners dafür danken, dass er mich mit Rat und Motivation bei diesem Projekt begleitend betreut hat.

Zu guter Letzt danke ich noch Sabrina Brüse und Sandra Eilers dafür, dass sie geholfen haben diese Arbeit auf Rechtschreibfehler zu kontrollieren.

Literaturverzeichnis

[Aspern2003] Jens von Aspern: SPS-Softwareentwicklung mit Petrinetzen (Berlin und Offenbach, 2003).

[Lunze2006] Jan Lunze: Ereignisdiskrete Systeme (München, 2006).

[Meiners2009] Ulfert Meiners: Skript Steuerungstechnik (Hamburg, 2009).

[Petri1962] Carl Adam Petri: Kommunikation mit Automaten (Institut für instrumentelle Mathematik der Universität Bonn, 1962).

Onlinequellen

[Baray2010] Cristobal Baray: the model-view-controller (MVC) design pattern ()
[http://cristobal.baray.com/indiana/projects/mvc.html] (Zugriff: 24. 08 2010).

[Bloom2003] James Bloom, Clare Clark, Camilla Clifford, Alex Duncan, Haroun Khan, Manos Papantoniou: Final Report: Platform Independent Petri Net Editor (2003)
[http://pipe2.sourceforge.net/documents/PIPE-Report.pdf] (Zugriff: 03. 02 2011).

[Ecl2010] Eclipse Foundation: Eclipse ()
[http://www.eclipse.org] (Zugriff: 10. 10 2010).

[Edw2007] Edwin Chung, Tim Kimber, Ben Kirby, Thomas Master, Matthew Worthington: Final Report: PETRI NETS GROUP PROJECT (2007)
[http://pipe2.sourceforge.net/documents/PIPE2-Report-20070327.pdf] (Zugriff: 07. 03 2011).

[Grude1988] Ulrich Grude: Petri-Netze eine informelle Einführung in die Grundideen (1988)
[http://public.beuth-hochschule.de/~grude/Petrinetze.pdf] (Zugriff: 09. 03 2011).

[JGraph2010] JGraph Ltd: JGraph ()
[http://www.jgraph.com] (Zugriff: 03. 06 2010).

[JPowerGraph2011] Mick Kerrigan: JPowerGraph ()
[http://sourceforge.net/projects/jpowergraph/] (Zugriff: 17. 02 2011).

[Nad2005] Nadeem Akharware: PIPE2: Platform Independent Petri Net Editor (2005)
[] (Zugriff: 07. 03 2011).

[PIPE] Imperial College London MSc. Group Project at the Department of Computing: The Platform Independent Petri net Editor PIPE (2003)
[http://pipe2.sourceforge.net] (Zugriff: 09. 03 2011).

[PNML2011] Jonathan Billington, Jun Ginbayashi, Lom Hillah, Ekkart Kindler, Fabrice Kordon, Laure Petrucci, Yann Thierry-Mieg, Nicolas Trèves: PNML.org ()
[http://www.pnml.org/] (Zugriff: 18. 01 2011).

[PNW2010] TGI group University of Hamburg: Petri Nets World ()
[http://www.informatik.uni-hamburg.de/TGI/PetriNets/] (Zugriff: 17. 06 2010).

[SUN2011] SUN: Project JAXP ()
[http://jaxp.java.net/] (Zugriff: 14. 01 2011).

[Tom2011] Tom Barnwell, Michael Camacho, Matthew Cook, Maxim Gready, Peter Kyme, Michail Tsouchlaris: Final Report: PETRI NET ANALYSER (2004)
[http://pipe2.sourceforge.net/documents/PIPE2%20Final%20Report.pdf] (Zugriff: 06. 02 2011).

[W3C2011] World Wide Web Consortium (W3C): Extensible Markup Language (XML) ()
[http://www.w3.org/XML/] (Zugriff: 03. 03 2011).

[WikiXML] Wikipedia: Extensible Markup Language ()
[http://de.wikipedia.org/wiki/Extensible_Markup_Language] (Zugriff: 06. 02 2011).

Abkürzungsverzeichnis

AWL : Anweisungsliste
API : Application Programming Interface (Programmierschnittstelle)
CPU : Central Processing Unit (Hauptprozessor)
FPGA : Field Programmable Gate Array
HTML : Hyper Text Markup Language
MVC : Model View Controller
OB1 : Organisationsbaustein 1 (zyklische Abarbeitung)
OB100 : Organisationsbaustein 100
OLE : Object Linking and Embedding
OPC : OLE for Process Control
PNML : Petri Net Markup Language
SIPN : Schaltungstechnisch interpretierte Petri-Netze
SipnLab : Signal Interpreted Petri Net Laboratory
SPS : Speicher programmierbare Steuerung
Tag : Auszeichner
XML : Extensible Markup Language

A Anhänge

Anhang A1 : Kurzbeschreibung der wichtigsten Klassen
Anhang A2 : Ausgabe des Debuggers bei der Verifikation
Anhang A3 : Source Code und ausführbare Version von SipnLab

Der Anhang A3 ist in elektronischer Form auf einer CD abgelegt und beim Prüfer Prof. Dr. Ing. Ulfert Meiners einzusehen.

A1 Kurzbeschreibung der wichtigsten Klassen

RunGUI	enthält die main-Methode, welche das Programm startet.

dataLayer

Arc	beschreibt die Kanten**,** ihre grafische Darstellung und die Funktionalitäten
DataLayer	enthält alle Daten zur Laufzeit des Programms, durch enthalten Methoden werden die Daten in ein mathematisches Modell umgewandelt.
DataLayerWriter	wandelt das Modell in einen XML-Baumstruktur um
InhibitorArc	beschreibt die Inhibitorkante und ihre grafische Darstellung
Input	beschreibt die Eingänge
Output	beschreibt die Ausgänge
PetriNetObject	beschreibt die Elernklasse aller Petrinetzelemente
Place	beschreibt die Stellen und ihre grafische Darstellung
PlaceTransitionObject	beschreibt die Elternklasse der Stellen und Transitionen, enthält gemeinsame Funktionalitäten
PNMatrix	stellt Matrizen und Rechenoberationen für diese bereit
SipnLabToSPS	transformiert das Datenmodell in AWL-Code
Transiton	beschreibt die Transitionen und die grafische Darstellung
ValidatorArc	beschreibt die Testkanten und ihre grafische Darstellung

gui

CreateGui	beschreibt den Aufbau des Hauptfensters
GuiFrame	beschreibt und verarbeitet alle die Eingaben des Benutzers
GuiView	erzeugt die grafische Darstellung des Editors und aller enthaltenen Elemente

A2 Ausgabe des Debuggers bei der Verifikation

Die Abbildung A-1 zeigt die Ausgabe von Eclipse im Debugmodus. Mit den Ergebnissen des Debuggers wurde die Verifikation des Datenmodells in Kapitel 5.2 bewerkstelligt.

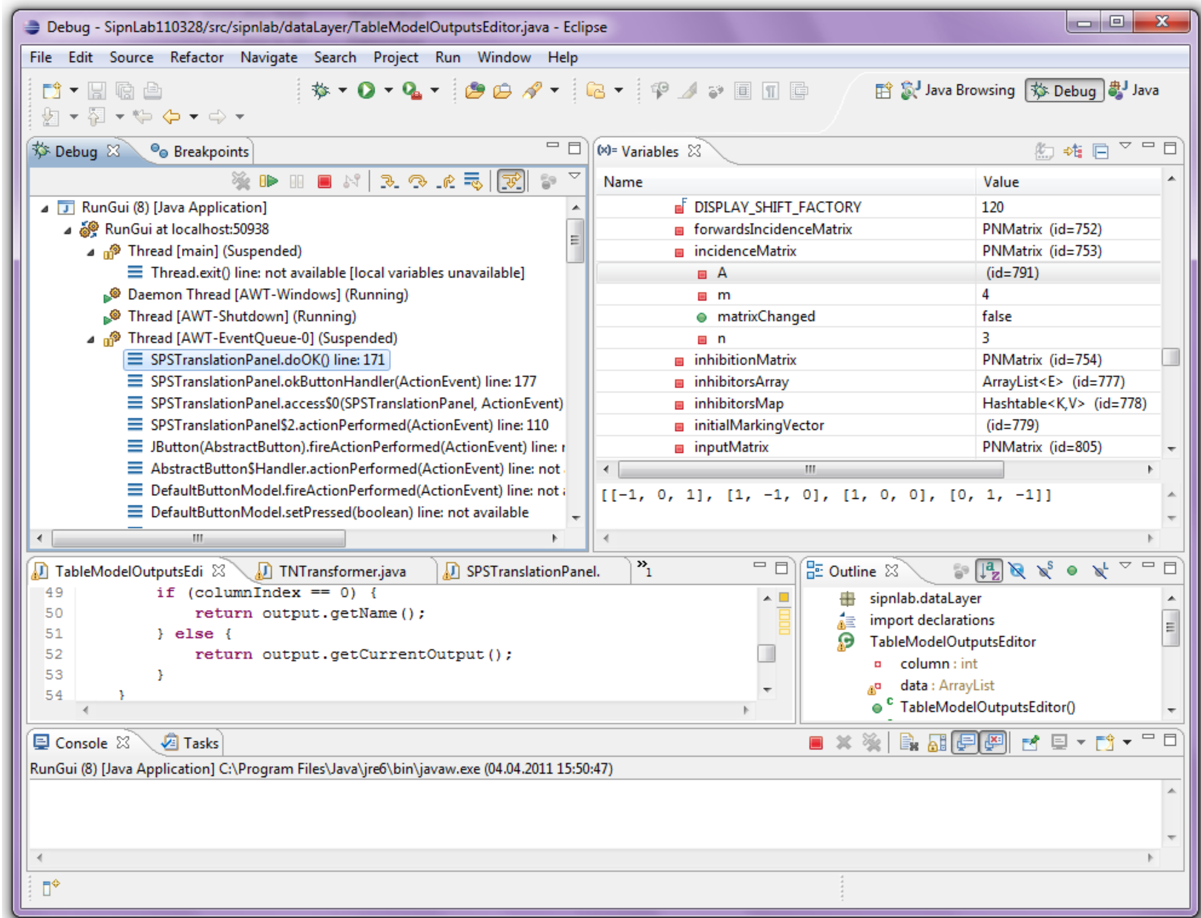


Abbildung A-1: Ausgabe des Debuggers

A3 Source Code und weitere Daten

Der Datenträger enthält den Source Code als Eclipseprojekt, sowie eine ausführbare Version von SipnLab. Des Weiteren sind alle Beispielnetze enthalten, die mit SipnLab im Rahmen dieser Arbeit erstellt worden sind.

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5)APSO-TI-BM ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 11.04.11
Ort, Datum

____
Unterschrift