

Verhaltensmodellierung von Petrinetz-Agenten*

Daniel Moldt und Heiko Rölke
Universität Hamburg – Fachbereich Informatik
Arbeitsbereich „Theoretische Grundlagen der Informatik“
Vogt-Kölln-Straße 30 – 22527 Hamburg
{moldt,roelke}@informatik.uni-hamburg.de

3. April 2001

Zusammenfassung

Agenten werden zur Zeit hauptsächlich direkt in Hochsprachen wie Java programmiert (zum Beispiel in Frameworks wie Jackal [CFL⁺98]) oder in Form einfacher Skripte definiert. Eine graphische Modellierung, wie sie sich in der Objekt-Orientierung mit UML durchgesetzt hat, findet nicht statt.

Diese Arbeit zeigt auf, wie das Verhalten von Agenten in Form ausführbarer Protokolle mit gefärbten Petrinetzen festgelegt werden kann.

Dazu findet eine Spezialform der gefärbten Netze, die Referenznetze, Anwendung. Das komplexe Verhalten der Agenten wird durch dynamische Komposition einfacher Teile modelliert; besonders die bei Petrinetzen inhärente Berücksichtigung von Nebenläufigkeit macht diese zu einer geeigneten Modellierungstechnik.

1 Motivation

Agentenorientierung wird durch Intelligenz, Autonomie und Mobilität gekennzeichnet. Die Behandlung der Kapselung, Strukturierung und Flexibilität im Rahmen der Modellierung ist dabei eine große softwaretechnische wie auch theoretische Herausforderung.

An unserem Arbeitsbereich wird untersucht inwiefern Petrinetze sich zur Bewältigung dieser Fragestellungen eignen. Ausgehend von den fundamentalen Petrinetzkonzepten werden insbesondere Petrinetze als aktive Marken (siehe [Val98]) und der grundlegende Aufbau der objektorientierten Petrinetze (siehe [Mol96]) eingesetzt. Von den agentenorientierten Petrinetzen (siehe [MW97]) ausgehend wird den Überlegungen aus [Röl99] folgend in diesem Beitrag die Modellierung von Agenten auf der Basis von Petrinetzen vorgestellt. Zur Modellierung findet der Formalismus der Referenznetze nach Kummer (siehe [Kum98]) Anwendung.

Der Schwerpunkt liegt in diesem Beitrag auf der Modellierung des Verhaltens eines Agenten durch Protokolle, wobei dieser Ansatz als grundlegend auch für weitere Sichten angesehen werden kann, ohne daß dies hier vertieft wird. Die Struktur eines Agenten und auch das ihn umgebende Multiagentensystem werden in unserem Ansatz ebenfalls durch Referenznetze modelliert, hierauf kann jedoch in diesem Rahmen nicht weiter eingegangen werden.

*©Moldt / Rölke. Beitrag auf dem Workshop „Visuelle Verhaltensmodellierung verteilter und nebenläufiger Software Systeme“, 13.-14.11.2000, Universität Münster

Der Rest des Dokuments ist wie folgt gegliedert: Im nachfolgenden Kapitel 2 wird der Formalismus der Referenznetze kurz eingeführt. Kapitel 3 beschreibt die grundsätzliche Architektur der Agenten, deren Verhalten durch Petrinetzprotokolle festgelegt wird. Diese Art von Netzen wird in Kapitel 4 anhand eines Beispiels vorgestellt. Der Ausblick im abschließenden Kapitel 5 nennt weitere Arbeitspunkte, die noch in der Diskussion stehen oder über den Rahmen dieses Papiers hinausgehen.

2 Referenznetze

Es wird in diesem Text davon ausgegangen, daß der Leser mit Petrinetzen im allgemeinen sowie gefärbten Petrinetzen im speziellen vertraut ist. Eine allgemeine Einführung gibt beispielsweise Reisig [Rei85], gefärbte Petrinetze beschreibt Jensen [Jen92]. Allgemein gesprochen erlauben gefärbte Petrinetze bei gleicher Mächtigkeit eine kompaktere Darstellung als beispielsweise einfache Stellen/Transitions-Netze.

Referenznetze [Kum98] sind sogenannte höhere (gefärbte) Petrinetze, also eine graphische Notation, die sich speziell zur Beschreibung und Ausführung komplexer, nebenläufiger Prozesse eignet. Für Referenznetze existieren (wie auch für andere Netzformalismen) Werkzeuge zum Simulieren solcher Beschreibungen [KW98].

Referenznetze weisen gegenüber „herkömmlichen“ gefärbten Netzen einige Erweiterungen auf: zusätzliche Kantenarten, Netzexemplare und Kommunikation über synchrone Kanäle. Ansonsten sind sie den gefärbten Petrinetzen sehr ähnlich. Auf die Unterschiede wird nun kurz eingegangen.

Kantenarten Referenznetze bieten neben den normalen Kanten Reservierungskanten, die an beiden Enden eine Pfeilspitze tragen und eine Marke lediglich für einen Schaltvorgang *reservieren*, Testkanten und Inhibitorkanten. Testkanten ziehen eine Marke nicht von einer Stelle ab, weshalb eine Marke beliebig häufig gleichzeitig (auf Existenz) *getestet* werden kann. Inhibitorkanten verhindern Schaltvorgänge, solange die verbundenen Stellen markiert sind.

Netzexemplare Netzexemplare lassen sich mit den Objekten einer objektorientierten Programmiersprache vergleichen. Sie stellen Ausprägungen eines Netzes dar, genau wie Objekte Instanzen einer Klasse sind. Verschiedene Exemplare desselben Netzes können zur selben Zeit unterschiedliche Zustände einnehmen und sind auch ansonsten nicht direkt voneinander abhängig.

Synchrone Kanäle Synchrone Kanäle [CH94] erlauben eine Verschmelzung von (jeweils zwei) Transitionen für die Dauer eines Schaltvorganges. Bei Referenznetzen (siehe [Kum98]) wird ein Kanal über seinen Namen und die übergebenen Argumente identifiziert. Kanäle sind gerichtet, d.h. genau eine der beiden Kanalanschriften der Transitionen gibt das Netzexemplar an, in dem sich das Gegenstück des Kanals befindet. Das Gegenüber kann entsprechend von beliebigen Netzexemplaren aus angesprochen werden. Die Kommunikation über einen synchronen Kanal kann allerdings bidirektional stattfinden und ist auch innerhalb eines Netzexemplars möglich.

3 Agentenaufbau

Ein Agent wird als nachrichtenverarbeitende Einheit aufgefaßt, das heißt, er muß in der Lage sein, Nachrichten entgegenzunehmen, diese eventuell zu be- oder verarbeiten und eigene Nachrichten zu generieren. Hierbei ist zu beachten, daß ein vollständig synchroner Nachrichtenmechanismus, wie

er aus der Welt der Objektorientierung bekannt ist, häufig dem Agentengedanken zuwiderläuft und deshalb nicht vorgeschrieben ist.¹

Als Petrinetz modelliert stellt Abbildung 1 die am weitesten vergrößerte noch sinnvolle Sicht auf einen solchen Agenten dar. Als Spezialität der Referenznetze fallen hier die Eingangs- und Ausgangstransitionen auf. Diese können mit anderen Transitionen in anderen Netzexemplaren über synchrone Kanäle kommunizieren beziehungsweise Nachrichten austauschen. Die Transition **Verarbeitung** macht sich die grundlegende Eigenschaft von Petrinetzen zu Nutze, daß Transitionen mit unterschiedlichen Bindungen beliebig häufig nebenläufig zu sich selbst schalten können. Die Verarbeitungstransition kann für konkrete Agenten beliebig verfeinert werden. Bei Abbildung 1 und allen nachfolgenden Netzabbildungen sind zum Verständnis nicht unbedingt notwendige Beschriftungen weggelassen worden.

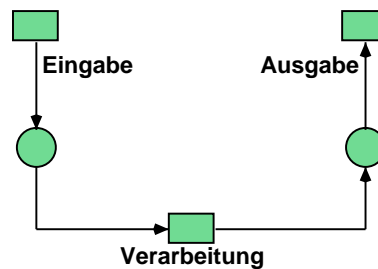


Abbildung 1: Größte noch sinnvolle Sicht auf einen Agenten

Das vorgestellte Grundmodell beinhaltet eine Kapselung der Agenten, die ungeachtet ihrer inneren Struktur nur über eine klar definierte Kommunikationsschnittstelle ansprechbar sind. In Abbildung 1 sind diese Schnittstellen durch die Transitionen **Eingabe** und **Ausgabe** dargestellt. In der Abbildung ist die Realisierung der Schnittstellen (durch Verbindung der beiden Transitionen mit einem Nachrichtenübertragungsnetz über synchrone Kanäle) nicht dargestellt. Es können selbstverständlich mehrere (dann virtuelle) Kommunikationskanäle auf diese beiden Transitionen abgebildet werden.

Eine solche Modellierung entspricht dem Agentengedanken; da Agenten Autonomie aufweisen sollen, müssen sie eine eigenständige Kontrolle über ihre Handlungen ausüben können. Dazu gehört auch die Möglichkeit, Nachrichten eines bestimmten Typs oder Absenders frühzeitig ausblenden zu können. Ungeachtet der prinzipiellen Autonomie kann ein Agent selbstverständlich so entworfen werden, beziehungsweise für sich die Verpflichtung übernehmen, nach außen wie ein Objekt zu wirken, also vollkommen kooperativ zu sein.

Die zur Modellierung von interessanten Fragestellungen erforderlichen Agenten müssen einerseits einen hohen Grad an Flexibilität, auch und gerade zur Laufzeit, aufweisen und andererseits leicht zu modellieren und anzupassen sein. Darüber hinaus ist für eine breitere Akzeptanz intuitive Verständlichkeit der Vorgänge innerhalb der Agenten notwendig. Diese Überlegungen spielten eine wichtige Rolle bei der Entwicklung der hier skizzierten Protokoll-gesteuerten Agenten.

Dazu wird das abstrakte Agentennetz aus Abbildung 1 in folgender Weise verfeinert: Das Agentennetz aus der Abbildung bildet (weiterhin) die Schnittstelle des Agenten zur Außenwelt. Die Transition **Verarbeitung** wird zu einer Auswahl entsprechender Subnetze verfeinert, die die Funktionalität des Agenten erbringen, also (neben dem Auswahlprozeß) sein Verhalten steuern. Diese Subnetze werden im folgenden *Protokollnetze* oder kurz *Protokolle* genannt.

¹Unserem Verständnis nach sind Agenten nicht ausschließlich (künstlich) intelligente Agenten, sondern ein allgemeines, auf dem Objektgedanken aufbauendes Strukturierungsparadigma für Software.

Jeder Agent kann über eine beliebige Menge solcher Protokolle verfügen, besitzt jedoch nur ein Netz (in Referenznetz-Nomenklatur: eine Netzseite), das seine Schnittstelle zum Agentensystem und damit seine Identität darstellt. Diese Hauptseite ist damit von außen (im Agentensystem) soweit sichtbar, daß mit ihr Nachrichten ausgetauscht werden können.

Die Hauptseite ist für die hier vorgestellten (protokollgesteuerten) Agenten in Abbildung 2 wiedergegeben. Es handelt sich um eine Verfeinerung des in Abbildung 1 gegebenen allgemeinen Agentennetzes.

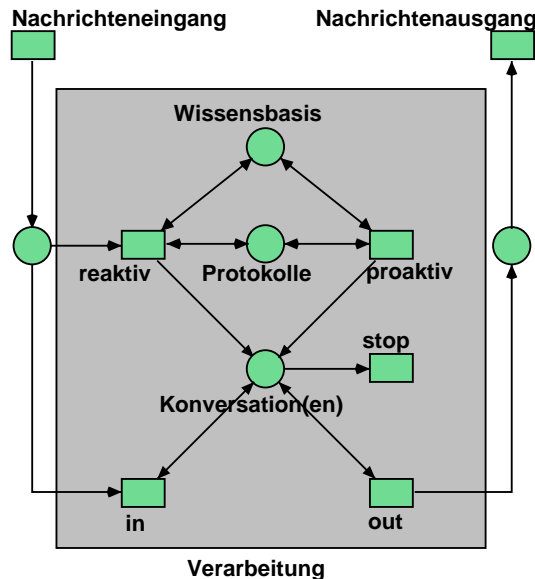


Abbildung 2: Hauptseite eines protokollgesteuerten Agenten

Im Mittelpunkt der Aktivität eines solchen Agenten steht die Protokollauswahl und damit die Aufnahme von Konversationen [CCF⁺99, Rö100]. Die Protokollauswahl kann grundsätzlich proaktiv (der Agent nimmt von sich aus eine Konversation auf) oder reaktiv (Protokollauswahl aufgrund einer von einem anderen Agenten angestoßenen Konversation) erfolgen. Diese Unterscheidung entspricht in der Abbildung den beiden Transitionen *reaktiv* und *proaktiv*. Zu beachten ist die Kante von der Nachrichteneingangsstelle zur Transition mit der reaktiven Protokollauswahl. Diese Transition kann also nur nach dem Eintreffen einer Nachricht schalten. Die Auswahl einer Konversation wird vom Wissen eines Agenten beeinflusst. Im Fall der proaktiven Protokollauswahl ist die Wissensbasis sogar die einzige Schaltbedingung, die Protokolle werden als Nebenbedingung angesehen. Die Wissensbasis kann in einfachen Fällen beispielsweise als Subnetz ausgeführt werden, aufwendigere Implementationen wie die Anbindung einer Inferenzeinheit sind ebenfalls möglich, sollen hier aber nicht weiter vertieft werden.

Ein solcherart aktiviertes Protokoll² wird im weiteren als *Konversation* bezeichnet, da es im Normalfall den Nachrichtenaustausch mit anderen Agenten impliziert. Eine aufgenommene Konversation erhält eine eindeutige Identifikation, die in der Abbildung nicht sichtbar ist. Unter Bezugnahme auf diese Identifikation können zur Konversation gehörende Nachrichten verschickt oder angenommen

²In Anlehnung an objekt-orientierte Sprechweisen spricht man von einem instantiierten Netz beziehungsweise Protokoll (das durch ein Netz dargestellt wird).

werden. Findet ein Nachrichteneingang statt, der sich auf eine bestehende Konversation bezieht, so schaltet statt der Transition *reaktiv* die Transition *in*. Die Netzbeschriftungen, die diese Bindungen sicherstellen, sind in Abbildung 2 aus Gründen der Übersichtlichkeit nicht dargestellt. Die Transition *in* reicht eingehende Nachrichten an das entsprechende in Ausführung befindliche Protokoll weiter. Beispiele für diesen Vorgang folgen in Abschnitt 4.

Ist im Ablauf einer Konversation das Versenden von Nachrichten an andere Agenten erforderlich, so werden diese über die Transition *out* aus dem (Protokoll-)Netz an die dargestellte Agentenhauptseite übergeben und von dieser über den hier nicht behandelten Nachrichtenübertragungsmechanismus an den oder die an der Konversation beteiligten Agenten weitergereicht. Die Kommunikation zwischen Protokollnetz (Konversation) und dem dargestellten Agentennetz erfolgt über synchrone Kanäle.

Beispiele für konkrete Konversationsprotokolle sind im folgenden Kapitel 4 zu finden, wobei ein Erzeuger-Verbraucher-Prozeß beispielhaft modelliert wird.

4 Agentenprotokolle

Ein wichtiges Einsatzgebiet von Petrinetzen ist die Spezifikation von (Ablauf-)Protokollen wie dem in Abbildung 3, das einen einfachen Erzeuger-Verbraucher-Prozeß darstellt. Um keine begriffliche Verwirrung aufkommen zu lassen, werden solche mehrere Agenten bzw. getrennte Funktionseinheiten übergreifenden Netze im weiteren „Überblicksnetz“ genannt.

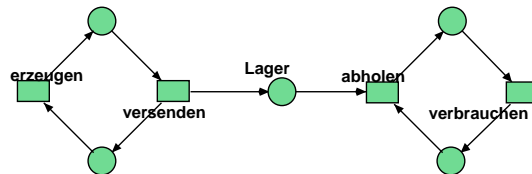


Abbildung 3: Erzeuger-Verbraucher Überblicksnetz

Die Stelle *Lager* in der Mitte der Abbildung stellt eine asynchrone Kopplung zwischen dem Prozeß des Erzeugens und dem des Verbrauchens dar. Diese Kopplung ist jedoch in dem Sinne eigenständig, als das sie beispielsweise in leerem Zustand den Verbraucher blockiert oder mit einer Kapazität versehen werden kann und dann bei maximaler Füllung den Erzeuger blockiert. Im nachfolgenden werden Erzeuger und Verbraucher als autonome Agenten angenommen und in Anlehnung an Abbildung 2 mittels eines Petrinetze modelliert. Das Lager wird hier nicht als eigenständiger Agent betrachtet.

Ein interessanter Punkt ist die Wiederverwendbarkeit der Protokolle: Soll in einer verfeinerten Modellierung das Lager eine aktive Rolle spielen, also als Agent modelliert werden, so müssen die Protokolle von Erzeuger und Verbraucher strukturell nicht verändert werden, lediglich die Adressaten der verschickten Nachrichten verändern sich. Diese sollten allerdings in jedem Fall als zur Laufzeit veränderbar modelliert werden. Eine entsprechende Darstellung des Lagers als eigenständiger Agent entfällt wiederum aus Platzgründen.

Im nachfolgenden Beispiel wird davon ausgegangen, daß das Lager die Kapazität eins besitzt. Diese Einschränkung dient wiederum der Vereinfachung und ist leicht aufzuheben. Die in den Abbildungen 4 und 5 gegebenen Netze weisen vom Überblicksnetz abweichende Transitionsnamen auf.

Diese sind aus der Überführung des Überblicksnetzes in die einzelnen Protokollnetze begründet und können hier nicht erläutert werden.

Erzeuger Das Protokoll des Erzeuger-Agenten ist in Abbildung 4 wiedergegeben. Typisch für alle Arten von Protokollnetzen sind die oberen Transitionen mit den Kanälen `:start`, `:out`, `:in` und `:stop`. Über den `start` Kanal können dem Protokoll nach seiner Erzeugung eventuell benötigte Parameter übergeben werden. Der Kanal wird auf der Agentenhauptseite (siehe Abb. 2) von den Transitionen reaktiv bzw. proaktiv aufgerufen. Die Kanäle `:in` und `:out` sind für die Kommunikation eines in Ausführung befindlichen Protokolls mit der Umwelt zuständig. Sie verbinden sich mit den gleichnamigen Transitionen der Hauptseite. Über den Kanal `stop` kann ein ausgeführtes Protokoll (korrekter: ein Protokoll exemplar) gelöscht werden.

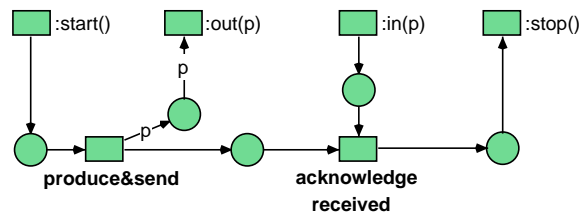


Abbildung 4: Erzeuger

Nach dem Start des Protokolls wird von der Transition `produce&send` ein Performativ³ (hier `p`) erzeugt, das an den Verbraucher gerichtet ist. Das Performativ wird über den `:out` Kanal versandt; anschließend wird auf Antwort gewartet. Das Warten ist notwendig, um eine synchrone Kommunikation zwischen Erzeuger und Lager zu simulieren. Ohne den Wartevorgang könnte der Erzeuger das Lager „überschwemmen“. Bei ankommender Bestätigung kann die Transition `acknowledge received` schalten, das Protokoll ist nicht weiter blockiert und beendet sich. Der Erzeuger-Agent kann dann erneut ein solches Protokoll auswählen.

Verbraucher Das Protokollnetz, das das Verhalten des Verbraucher-Agenten modelliert, wird von der Agentenhauptseite reaktiv zur Bearbeitung einer eintreffenden Nachricht vom Erzeuger ausgewählt und aktiviert.

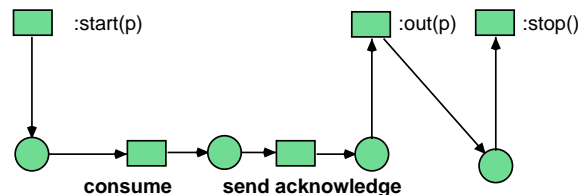


Abbildung 5: Verbraucher

³Die Wurzeln der hier vorgestellten Agenten liegen teilweise im Bereich der KQML- ([FL97]) oder FIPA-Agenten ([FIP98]). Ein Performativ ist vereinfacht gesprochen eine Nachricht. KQML steht für „Knowledge Query and Manipulation Language“, FIPA ist die Abkürzung von „Foundation for Intelligent Physical Agents“.

Der Verbraucher kann eine ankommende Nachricht solange blockieren, bis er sie „verbrauchen“ kann. In Abbildung 5 ist dies durch die Transition `consume` dargestellt. Das eigentliche Annehmen der Nachricht geschieht durch die Hauptseite des Agenten. Nach dem Verbrauchen wird dem Erzeuger eine Bestätigungsnachricht geschickt, damit dessen Protokoll terminieren kann. Damit ist das Protokoll beendet und kann gelöscht werden.

Die beiden Abbildungen 4 und 5 zeigen die Konversationen, die innerhalb der jeweiligen Agenten ablaufen. Den Rahmen bieten Agenten in der Gestalt wie sie in Abbildung 2 zu finden sind. Für das Erzeuger-Verbraucher-Modell konnte somit exemplarisch illustriert werden, wie Systemverhalten mittels agentenorientierter Petrinetze modelliert werden kann.

5 Ausblick

Bedingt durch den geringen zur Verfügung stehenden Raum konnten viele interessante Aspekte der hier vorgestellten Agenten nicht aufgeführt werden: Die Protokollnetze sind beispielsweise zur Laufzeit austauschbar (so der Agent dieses zuläßt). Ein mobiler Agent (Mobilität ist im hier nicht vorgestellten Rahmenwerk möglich) kann beispielsweise beim Eintreffen in eine Plattform die dort gültigen Protokolle übergeben bekommen.

Ebenfalls nicht behandelt werden konnte der eigentliche Modellierungsprozeß, der sich u.a. mit der Fragestellung beschäftigt, wie man „Übersichtsnetze“ (siehe Abbildung 3 erstellt und wie diese formal in die im letzten Kapitel vorgestellten Agentenprotokolle zu überführen sind.

Ein weiterer Punkt betrifft die Protokolle selbst: Es wurden nur sehr einfache Abläufe aufgezeigt. Kompliziertere Protokolle beinhalten eine hierarchische Schachtelung, die erst zur Laufzeit durch gegenseitige Aufrufe von Protokollen (auch und gerade innerhalb eines Agenten) aufgebaut wird. Dadurch wird eine dynamische Anpassung der Agenten durch Selbstmodifikation möglich.

Stark in der (internen) Diskussion ist zur Zeit die Wissensbasis. Es existiert ein in Java implementierter und sich in Renew einfügender Prologinterpreter, der bei komplizierteren Beispielen eingesetzt wird. Wünschenswert ist allerdings auch in diesem Bereich eine graphische (Wissens-) Modellierung.

Literatur

- [CCF⁺99] R. Scott Cost, Ye Chen, T. Finin, Y. Labrou, and Y. Peng. Modeling agent conversation with colored Petri nets. In *Working notes on the workshop on specifying and implementing concersation policies (Autonomous agents '99)*, 1999.
- [CFL⁺98] R.S. Cost, T Finin, Y. Labrou, X. Luan, Y. Peng, L. Soboroff, J. Mayfield, and A. Voughannanm. Jackal: A Java-based Tool for Agent Development. In *Working Notes of the Workshop on Tools for Developing Agents, AAAI'98*, pages 73–82. AAAI Press, 1998.
- [CH94] S. Christensen and N.D. Hansen. Coloured Petri nets extended with channels for synchronous communication. In Rober Valette, editor, *Application and Theory of Petri Nets 1994, Proc. of 15th Intern. Conf. Zaragoza, Spain, June 1994*, LNCS, pages 159–178, June 1994.
- [FIP98] FIPA. FIPA 97 Specification, Part 2 - Agent Communication Language. Technical report, Foundation for Intelligent Physical Agents, <http://www.fipa.org>, Oktober 1998.
- [FL97] Tim Finin and Yannis Labrou. A Proposal for a new KQML Specification. Technical report, University of Maryland, Februar 1997.

- [Jen92] K. Jensen. *Coloured Petri nets, Basic Methods, Analysis Methods and Practical Use*, volume 1 of *EATCS monographs on theoretical computer science*. Springer-Verlag, 1992.
- [Kum98] Olaf Kummer. Simulating synchronous channels and net instances. In J. Desel, P. Kemper, E. Kindler, and A. Oberweis, editors, *Forschungsbericht Nr. 694: 5. Workshop Algorithmen und Werkzeuge für Petrinetze*, pages 73–78. Universität Dortmund, Fachbereich Informatik, 1998.
- [KW98] Olaf Kummer and Frank Wienberg. *Reference net workshop (Renew)*. Universität Hamburg, <http://www.renew.de>, 1998.
- [Mol96] Daniel Moldt. *Höhere Petrinetze als Grundlage der Systemspezifikation*. Dissertation, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, 22527 Hamburg, Deutschland, 1996.
- [MW97] Daniel Moldt and Frank Wienberg. Multi-agent-systems based on coloured Petri nets. In P. Azéma and G. Balbo, editors, *Lecture Notes in Computer Science: 18th International Conference on Application and Theory of Petri Nets, Toulouse, France*, volume 1248, pages 82–101, Berlin, Germany, June 1997. Springer-Verlag.
- [Rei85] Wolfgang Reisig. *Petri nets: an introduction*. Springer, 1985.
- [Röl99] Heiko Rölke. Modellierung und Implementation eines Multi-Agenten-Systems auf der Basis von Referenznetzen. Diplomarbeit, Universität Hamburg, 1999.
- [Röl00] Heiko Rölke. Die Mulan Architektur. Technical report, Universität Hamburg, 2000.
- [Val98] Rüdiger Valk. Petri nets as token objects: An introduction to elementary object nets. In Jörg Desel and Manuel Silva, editors, *Application and Theory of Petri Nets*, volume 1420 of *Lecture Notes in Computer Science*, pages 1–25, June 1998.