VIPtool - Ein halbordnungsbasiertes Simulations- und Validationswerkzeug für Petrinetze

Thomas Freytag
Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB)
Universität Karlsruhe, 76128 Karlsruhe
Telefon: ++49-721-608-6590 - Fax: ++49-721-693717 - E-Mail: freytag@aifb.uni-karlsruhe.de

Zusammenfassung

Dieser Beitrag beschreibt das auf kausaler Semantik basierende Petrinetz-Simulationswerkzeug ***VIPtool***, das im Rahmen des DFG-Projekts ***VIP*** [DO95] entwickelt wird. Nach einem Überblick über den theoretischen und algorithmischen Hintergrund werden die funktionalen Hauptkomponenten von ***VIPtool*** beschrieben: Editor, Simulator, Prozeßnetz-Browser, graphische Anfrageschnittstelle und Validator.

1 Theoretischer Hintergrund

Traditionell gibt es zwei Sichtweisen zur Beschreibung des dynamischen Verhaltens von Petrinetzen: Die erste - genannt *sequentielle Semantik* - verwendet Mengen von total geordneten Systemereignissen (*Schaltfolgen*), die zweite - genannt *kausale Semantik* - Mengen von kausal halbgeordneten Abläufen (*Prozessen*) des Netzes. Die kausale Semantik wird im Bereich der Petrinetz-Theorie favorisiert, während die sequentielle Semantik im Bereich der Anwendungen und Werkzeuge dominiert.

Ebenfalls zwei Möglichkeiten gibt es, um gewünschte oder unerwünschte Systemeigenschaften des Netzmodells zu überprüfen. Zum einen kann man versuchen, die formulierte Eigenschaft durch Verwendung von *formalen Analyseverfahren* zu beweisen. Dies scheitert jedoch ab einer gewissen Netzgröße an der praktisch nicht mehr handhabbaren Komplexität. Zum anderen kann man versuchen, die gewünschte Eigenschaft durch Auswertung eines durch *Simulationsverfahren* gewonnenen, geeignet ausgewählten Ausschnitts des Systemverhaltens empirisch nachzuweisen - vergleichbar mit dem systematischen Testen von Software.

Das ***VIP***-Projekt beschreitet einen Weg, der die theoretisch unbestrittenen Vorteile der kausalen Semantik (z.B. die effiziente und elegante Darstellung mit expliziter Repräsentation von Kausalität und Nebenläufigkeit) auch für den Anwendungsbereich - und hier insbesondere für die Simulation, Validation und Visualisierung komplexer Systeme - nutzbar machen soll. Es werden u.a. folgende Ziele verfolgt:

- Simulation von mit höheren Petrinetzen modellierten Systemen durch Generierung halbgeordneter Abläufe (Prozesse)
- Visualisierung und Validation des dynamischen Systemverhaltens durch Prozesse
- Graphische Formulierung von Systemeigenschaften
- Überprüfung dieser Eigenschaften durch Auswertung der Prozesse

Eine detailliertere Beschreibung dieser Projektziele ist in [DFO97a, DFO97b, DFOZ97] zu finden.

Ein Hauptproblem bei der Simulation von Petrinetzen ist der Umgang mit *unendlichem Verhalten* des Systems, das sich durch die Existenz unendlich langer oder unendlich vieler Prozesse auswirkt. In mehreren jüngeren Publikationen aus dem Bereich der Netztheorie [McM92, Esp94, ERV96] wurden Verfahren vorgestellt, die - basierend auf dem Konzept der Verzweigungsprozesse - *endliche Präfixe* für unendliche Ausführungen bestimmen, ohne relevante Zustände zu verlieren. Die Grundidee hierbei ist die Identifikation von Ereignissen, die zu einer Wiederholung bereits früher erreichter Zustände führen (sog. *cut-off events*). Im Rahmen von ***VIP*** wurden diese Ergebnisse auf Prozeßnetze übertragen, um Abbruchkriterien für die Prozeßgenerierung zu erhalten.

Ein weiterer vielversprechender Ansatz in diesem Zusammenhang ist das Konzept der *externen Transitionen*, das erstmals in [Rei95] vorgeschlagen wurde. Externe Transitionen sind eine Teilmenge der Netztransitionen, die semantisch eine besondere Rolle als Schnittstellen zur Außenwelt spielen. Der Grundgedanke ist der, daß die meisten zyklischen Systeme kein wirklich unendliches Verhalten haben, sondern i.d.R. nur eine endliche Dienstleistung erfüllen, die sie als *Server* ihrer Umgebung anbieten. Das unendliche Verhalten entsteht dann erst dadurch, daß diese Umgebung als *Client* agiert und diese Dienstleistung beliebig oft anfordern kann. Das Ziel ist, diese Schnittstellen zu identifizieren und durch externe Transitionen zu modellieren. Bei der Simulation werden externe Transitionen dann gesondert behandelt: Ab einem - von Benutzer oder automatisch - festgelegten Zeitpunkt werden sie bei der Simulationsfortsetzung nicht mehr berücksichtigt, so daß ein an sich nichtterminierendes System auf realistische Weise terminieren kann und ein möglicherweise unendlicher Prozeß beendet werden.

2 Komponenten von *VIPtool*

2.1 Netzeditor

Ausgangspunkt bei der Bedienung von ***VIPtool*** ist ein Netzeditor (Abbildung 1). Höhere Petrinetze (Prädikat/Transitons-Netze) können erzeugt, gespeichert, geladen, geändert und simuliert werden. Das verwendete Netzdateiformat basiert auf einer PROLOG-Syntax. Import- und Exportschnittstellen zu anderen Netzformaten sind leicht implementierbar und auch geplant.

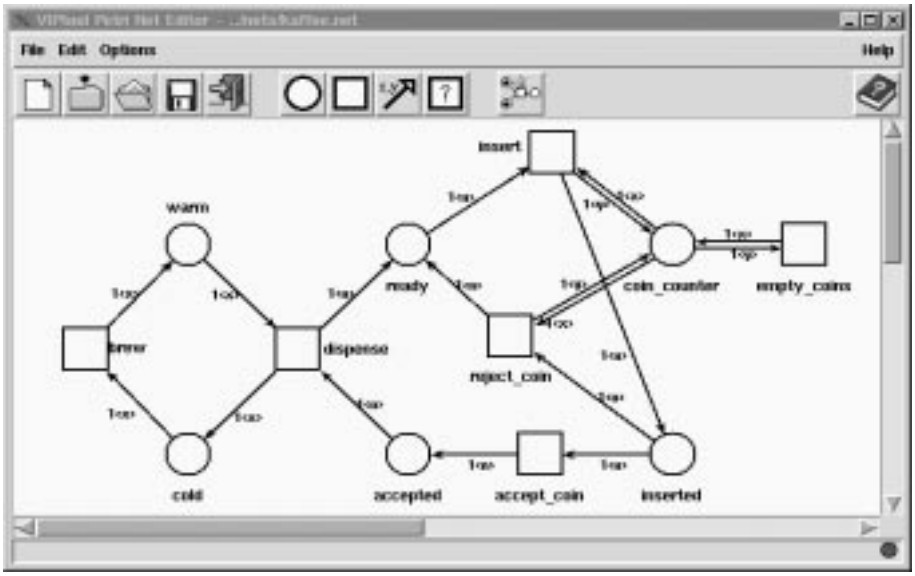

Abbildung 1: Der Netzeditor

2.2 Simulator

Aus dem Editor heraus kann der Simulator aufgerufen werden, woraufhin ein Dialogfenster erscheint (Abbildung 2). Hier kann der Benutzer Simulationsparameter spezifieren, wie z.B. die numerische Begrenzung der Prozeßlänge, die Verwendung von externen Transitionen oder *cut-off-events*. Danach beginnt der eigentliche Simulationsvorgang.

Abbildung 2: Dialogfenster zur Festlegung der Simulationsparameter

Eine wichtige Rolle bei der Simulation spielen *Konflikt-Ereignisse*. Das sind Ereignisse, die alternative Ausführungen implizieren und so zur Entstehung eines neuen Prozesses führen. Die Konstruktion der Prozesse erfolgt nach folgendem Algorithmus: Zunächst wird intern ein Verzweigungsprozeß $\mathcal{B}$ mit den der Anfangsmarkierung entsprechenden Bedingungen als minimalen (und anfangs gleichzeitig maximalen) Elementen initialisiert. $\mathcal{B}$ ist anfangs identisch mit dem aktuell konstruierten Prozeß $\mathcal{P}_{curr}$. Dieser wird nun solange um Ereignisse erweitert, wie dies ohne Konflikt (d.h. unter ausschließlicher Verwendung der maximalen Elemente von $\mathcal{P}_{curr}$) möglich und die ggfs. vom Benutzer vorgegebene Prozeßlänge nicht erreicht ist. Zudem sind ab einem gewissen Zeitpunkt nur noch Ereignisse für interne Transitionen auswählbar. Für jedes neue Ereignis $\mathcal{E}$ wird überprüft, ob es sich um ein *cut-off-event* bzgl. anderer Ereignisse von $\mathcal{B}$ handelt. Falls ja, dann wird der Nachbereich von $\mathcal{B}$ als nicht mehr für Folgeereignisse verwendbar gekennzeichnet. Sobald keine konfliktfreie Erweiterung mehr möglich ist, wird ein Konflikt-Ereignis $\mathcal{C}$ ermittelt, dessen Vorbereich durch eine beliebige Menge von ungeordneten Bedingungen von $\mathcal{B}$ repräsentiert ist. Der aktuelle Prozeß $\mathcal{P}_{curr}$ wird somit beendet und ein neuer Prozeß begonnen, dessen maximale Elemente aus dem zu einem maximalen Schnitt erweiterten Nachbereich von $\mathcal{C}$ bestehen. Der neue Prozeß wird nun zum aktuellen Prozeß $\mathcal{P}$ und wieder sukzessiv konfliktfrei fortgesetzt, bis es nicht mehr geht. Anschließend wird ein neues Konflikt-Ereignis bestimmt usw. Das Verfahren terminiert, sobald kein Konflikt-Ereignis mehr gefunden oder die ggfs. vom Benutzer vorgegeben Prozeßanzahl erreicht wird.

Die Simulation basiert also auf einer prozeßweisen Konstruktion von Pfaden und Teilpfaden im Verzweigungsprozeß. Falls von Benutzer keine explizit gegenteiligen Simulationsparameter spezifiziert worden sind, dann gilt:

- Im Falle eines Netzes mit endlicher Prozeßmenge können alle Prozesse erzeugt werden
- Der entstandene Verzweigungsprozeß ist vollständig, aber nicht minimal

2.3 Prozeß-Browser

Nach dem Ende der Simulation können die Prozesse in einem Prozeß-Browser betrachtet werden (Abbildung 3). Die maschinell generierten Prozeßnetze werden mit Hilfe des aus dem Graphentheorie bekannten Sugiyama-Algorithmus [STT81] automatisch nach visuellen und ästhetischen Kriterien plaziert.

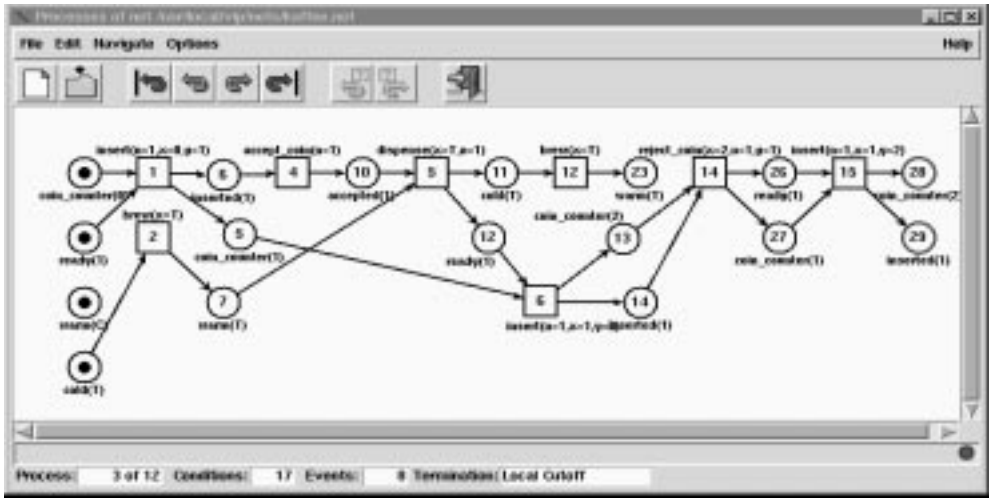

Abbildung 3: Der Prozeß-Browser

2.4 Graphische Anfrageschnittstelle

Eigenschaften des modellierten Systems können direkt im Netzeditor graphisch spezifiziert werden. Dies erfolgt durch sog. *Anfrage-Transitionen (query transitions)*, die nur deklarativen Charakter und keinen Einfluß auf das Schaltverhalten des Netzes haben. Derzeit sind in ***VIPtool*** drei Typen von Anfrage-Transitionen definiert (weitere Typen sowie die Definition einer adäquaten temporalen Logik sind geplant):

- *Fakten* (eingeschriebenes 'F'): Anfrage, ob ein bestimmter lokaler Zustand jemals erreicht wird
- *Kausalketten* (eingeschriebenes 'K'): Anfrage, ob eine Folge kausal geordneter Elemente einer vorgegebenen Länge (eingeschriebene Zahl) zwischen zwei Zuständen existiert
- *Ziele* (eingeschriebenenes 'T' für target): Anfrage, ob jede in einem bestimmten Zustand beginnende Systemfortsetzung nach endlich vielen kausal geordneten Ereignissen einen bestimmten Folgezustand erreicht

Abbildung 4 enthält für jeden dieser drei Anfragetypen ein Beispiel.

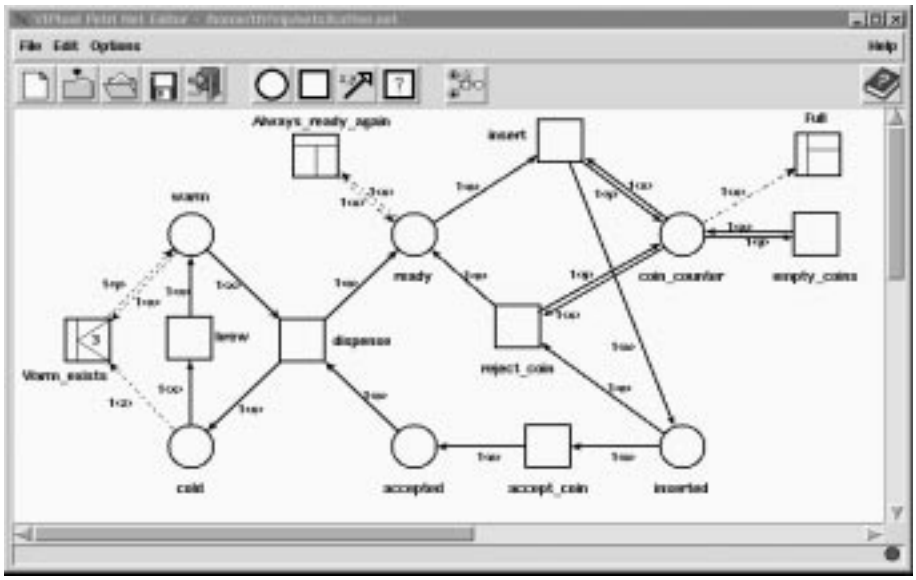

Abbildung 4: Die graphische Anfrageschnittstelle

2.5 Validator

Der Validator beantwortet die formulierten Anfragen durch Durchsuchen der generierten Prozesse nach entsprechenden Mustern aus Bedingungen und Ereignissen aus. Hierbei können die in den Prozessen explizit repräsentierten Kausalbeziehungen ausgenutzt werden. Die gefundenen Antworten werden auf Wunsch im Prozeß-Browser graphisch dargestellt (Abbildung 5).

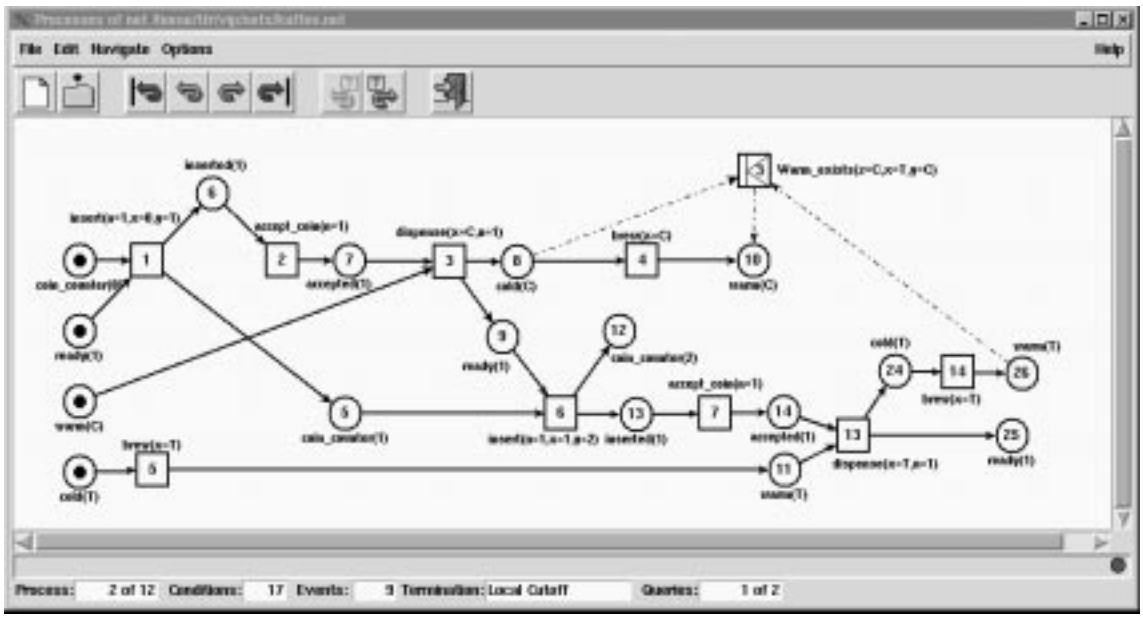

Abbildung 5: Validator-Antwort im Prozeß-Browser

Im Falle von Fakt- und Kausalketten-Anfragen werden alle gefundenen Ereignisse angezeigt. Im Falle von Ziel-Anfragen werden nur die Widerlegungen angezeigt (d.h. diejenigen Ereignisse, deren Vorbedingungen vorliegen und deren Nachbedingungen nicht erfüllt wurde).

3 Implementation

3.1 Architektur

Editor und Prozeß-Browser sind in *Python* programmiert, einer frei verfügbaren, objektorientierten Interpretersprache. Die graphische Oberfläche basiert auf *Tkinter,* einer *Python*-Schnittstelle zu den Funktionen des ebenfalls frei verfügbaren und weitverbreiteten *Tcl/Tk*-Systems. Die Editor-Software greift auf Teile des ebenfalls am AIFB der Universität Karlsruhe entwickelten Werkzeug *GAPS++* [Wei96] zurück.

Simulator und Validator sind in *ANSI-C* geschrieben und als *Python*-Erweiterungsmodul realisiert, einer sehr eleganten und flexiblen Methode, den *Python*-Standardinterpreter um benutzerdefinierte Funktionen zu ergänzen. Die hierdurch erreichte Trennung von graphischer Oberfläche und algorithmischer Implementation begünstigt die Wiederverwendung der Simulations- und Validationsalgorithmen in anderen Umgebungen, wie z.B. bei der geplanten Anbindung von ***VIPtool*** an den *Petrinetz-Kern* [DK96]. Abbildung 6 gibt einen Überblick über den Aufbau und die Funktionalität von ***VIPtool***.

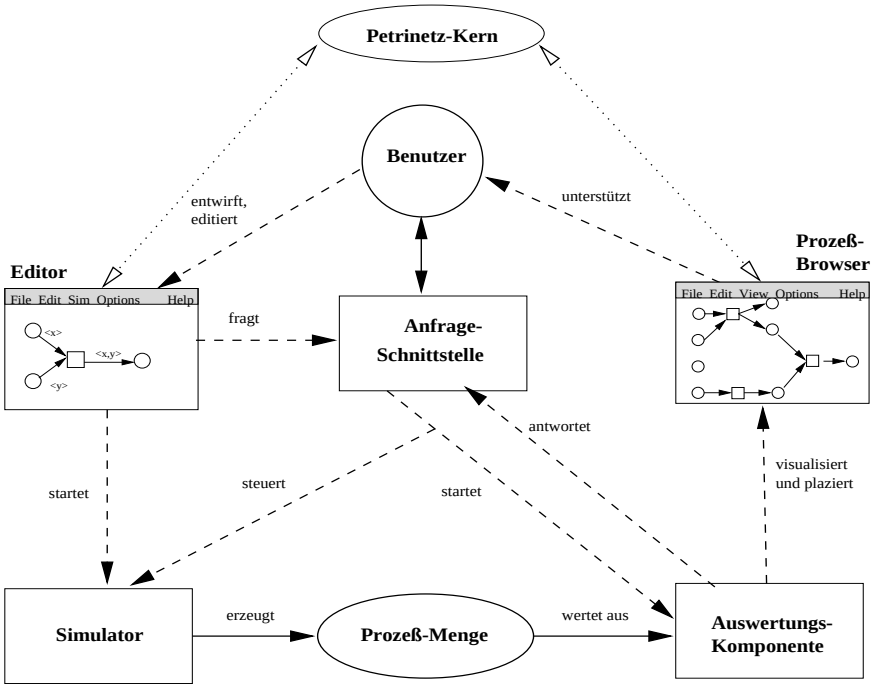

Abbildung 6: Softwarestruktur von ***VIPtool***

3.2 Verfügbarkeit und Ausblick

VIPtool ist kostenlos verfügbar für PCs unter den Betriebssystemen *Linux* und *Windows 95/NT,* sowie für SUN Workstations unter *SunOS* und *Solaris.* ***VIPtool*** erfordert die Installation von *Python 1.4* mit *Tkinter, Tcl 7.6* oder später, *Tk 4.2* oder später und *Tix 4.1.* Es ist jedoch auch geplant, reine Binärinstallationen anzubieten, die keine weitere Softwareinstallation voraussetzen. Die Software sowie weitere Informationen über ***VIPtool*** sind auch im WWW verfügbar [DFOZ97].

Literatur

[DFO97a] J. Desel, T. Freytag und A. Oberweis. *Prozesse, Simulation und Eigenschaften netzmodellierter Systeme.* in: Entwurf komplexer Automatisierungssysteme (EKA 97), S. 141–162. Braunschweig, 1997.

[DFO97b] J. Desel, T. Freytag und A. Oberweis. *Causal-semantic-based simulation and validation of high-level Petri nets.* in: Proceedings of the European Simulation Multiconference (ESM 97), S. 826–831. Istanbul, 1997.

[DFOZ97] J. Desel, T. Freytag, A. Oberweis und T. Zimmer. *The VIP project homepage.* World Wide Web. http://www.aifb.uni-karlsruhe.de/InfoSys/VIP/overview/vip.html.

[DK96] J. Desel und E. Kindler. *Der Traum von einem universellen Petrinetz-Werkzeug - der Petrinetz-Kern.* in: AIFB-Forschungsbericht 341, S. 27–32. Karlsruhe, 1996.

[DO95] J. Desel und A. Oberweis. *Verifikation von Informationssystemen durch Auswertung halbgeordneter Petrinetz-Abläufe.* AIFB-Forschungsbericht 324. Karlsruhe, 1995.

[Esp94] J. Esparza. *Model checking using net unfoldings.* Science of Computer Programming. (23):151–195, 1994.

[ERV96] J. Esparza, S. Römer und W. Vogler. *An improvement of McMillan's Unfolding Algorithm.* in: Proceedings of TACAS '96, S. 87–106. Passau, 1996.

[McM92] K. McMillan. *Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits.* in: Proceedings of 4th Workshop on Computer Aided Verification, S. 164–174. Montreal, 1992.

[Rei95] W. Reisig. *Petri Net Models of Distributed Algorithms.* Computer Science Today, LNCS 1000, S. 441–454. Springer-Verlag, 1995.

[STT81] K. Sugiyama, S. Tagawa und M. Toda. *Methods for visual understanding of hierarchical system structures.* IEEE Transaction on Systems, Man and Cybernetics, SMC-11(2):109–125, 1981.

[Wei96] W. Weitz. *GAPS++ - ein verteiltes Petrinetz-Werkzeug.* Workshop Algorithmen und Werkzeuge für Petrinetze. AIFB-Forschungsbericht 341, S. 77–82. Karlsruhe, 1996.