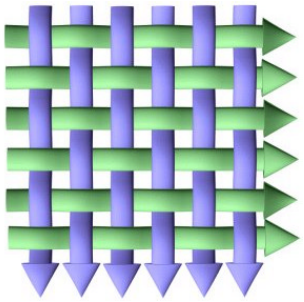
Sonderforschungsbereich 559
Modellierung großer Netze in der Logistik



Technical Report 04003

ISSN 1612-1376

Prozessablauf-Visualisierung von ProC/B-Modellen

Teilprojekt M1:
Carsten Tepper
Universität Dortmund
Informatik IV
August-Schmidt-Str. 12
44221 Dortmund

Dortmund, 7. Oktober 2004

1. Einführung

Für den Entwurf und das Design von grossen Netzen der Logistik ist eine automatisierter model-gestützter Ansatz sinnvoll (wünschenswert). Aus diesem Grund entstand in den ersten beiden Phasen das ProC/B-Paradigma, welches auf dem Prozesskettenparadigma nach Kuhn [Kuhn95] beruht. Um dieses ProC/B-Paradigma wurden DV-Werkzeuge [AFT01,AEF03,BBF+02] entwickelt, die die Modellierung und Analyse von prozessorientierten Systemen (ProC/B-Modellen) ermöglichen. Diese DV-Werkzeuge beinhalten einen Editor zur Erstellung von ProC/B-Modellen und Transformatoren in die Eingabesprache HiSlang des Simulators HIT, sowie in die Modellwelten Petri Netze und Warteschlangennetze.

Aussagen über Zeit und Kosten lassen sich durch quantitative Analysen, z.B. Simulation, des ProC/B-Modells gewinnen. Beim Experimentieren in der zweiten Phase des SFB hat sich gezeigt, dass die funktionale Korrektheit des Modells für die quantitative Analyse von zentraler Bedeutung ist, da ansonsten gerade bei der Simulation unerwünschte, fehlerhafte Ergebnisse geliefert werden. Ein häufiges funktionales Problem ist die Verklemmung von nebenläufigen Prozessen bei prozessorientierten Systemen, welches sich nur schwer durch Simulation entdecken lässt [AFK03, DiV03]. In [BaB99] wird ein Modell eines logistischen Systems angegeben, welches keine stationäre Phase besitzt. Auch diese Probleme kann eine Simulation nur schwer erkennen.

Solche Probleme im Modell kann der Modellierer nur schwer durch Betrachten und Nachvollziehen erkennen, deshalb wurde ein DV-Werkzeug entwickelt, dass Prozessabläufe visualisiert. Durch die Visualisierung des Prozessablaufes können Modellierungsfehler leichter entdeckt werden. Die Prozessabläufe werden durch sogenannte ***ProC/B-Traces*** protokolliert und können vorwärts und rückwärts betrachtet werden. Somit ist eine Art Debugging des Modells möglich, wie es viele Programmentwicklungssysteme beim Debuggen von Programmen anbieten.

In Anlehnung an das *Tokengame* bei Petri Netzen, wurde die Prozessablauf-Visualisierung *Processgame* getauft. Bei Petri Netzen werden Marken (engl. Token) auf Stellen zerstört und erzeugt und bei ProC/B-Modellen wandern Prozesse (engl. Process) von einer Aktivität zur nächsten Aktivität.

Dieser Bericht hat folgenden Aufbau:

Kapitel 2 erläutert das ProC/B-Paradigma und beschreibt das ProC/B-Modell eines Lagers für ein Güterverkehrszentrums. In Kapitel 3 wird das Instrumentarium zur Prozessablauf-Visualisierung vorgestellt. Im folgenden Kapitel 4 werden die möglichen Einträge/Schritte in den ProC/B-Trace anhand der ProC/B-Elemente erläutert. Kapitel 5 zeigt andere Möglichkeiten zur Gewinnung eines Prozessablaufes (ProC/B-Traces) durch Simulation mittels HIT und durch funktionale Analyse von Petri Netzen auf. Eine Zusammenfassung erfolgt in Kapitel 6 und im Anhang sind die Grammatiken für ProC/B-Traces und PN-Traces aufgelistet.

2. ProC/B-Paradigma

In diesem Kapitel wird kurz das ProC/B-Paradigma erläutert. Anschliessend wird ein ProC/B-Modell eines Lagers für ein Güterverkehrszentrums (GVZ) vorgestellt. Dieses Modell wird im nächsten Kapitel zur Vorstellung des Instrumentariums zur Prozessablauf-Visualisierung verwendet.

Das ProC/B-Paradigma ist aus dem mehr informellen Prozesskettenparadigma nach Kuhn entstanden. Verwendet wurde das Prozesskettenparadigma zur deskriptiven Beschreibung von Prozessabläufen, aber für eine Analyse (Simulation) musste noch ein weiteres Simulationsmodell entworfen werden. Ziel der Präzisierung des Prozesskettenparadigmas in das ProC/B-Paradigmas war, dass nur noch ein Modell für eine automatisierte Analyse entwickelt werden muss.

Die Beschreibung des ProC/B-Paradigmas umfasst dabei sowohl die Struktur als auch das Verhalten des Systems. Entsprechend der Gliederung realer Systeme bietet das ProC/B-Paradigma Funktionseinheiten (FEs) an, die nach aussen an definierten Schnittstellen Dienste anbieten. Diese Funktionseinheiten können wiederum Dienste benutzen, die von SubFEs angeboten werden. Dienste werden durch Prozessketten beschrieben, die durch Prozesskettenelemente (PKEs) eine Aktivitätenfolge modellieren. PKEs können Verzögerungen, Dienstaufrufe oder HiSlang-Code [HIT93] enthalten. Nach unten abgeschlossen wird die auf diese Weise entstehende FE-Hierarchie durch standardisierte FEs (Server, Counter, Storage). Nach oben abgeschlossen wird die FE-Hierarchie durch Quellen und Senken, die eine Schnittstelle zur Umwelt darstellen. Die wichtigsten ProC/B-Elemente sind in Tabelle 1 aufgelistet. Eine detailliertere Beschreibung des ProC/B-Paradigmas kann [BBF+02] entnommen werden.

ProC/B-Element	**Darstellung**	**Semantik**
Funktionseinheit (FE)	Rechteck	Submodell
Server	Rechteck abgerundet	Bedienstation
Counter	Trapez	Semaphor, Zähler
Quelle	Kreis mit Punkt in der Mitte	(Prozess/Last)-generierung
Senke	Kreis mit Kreuz	Prozessterminierung
Prozesskette (PK)	Horizontaler „Flow-Chart“	Prozess-Spezifikation
PK-Interface	Kringel	Schnittstelle
Delay-PKE	Pfeilartiges Hexagon	Zeitverzögerung im Prozess
Code-PKE	Pfeilartiges Hexagon	Programm-Code (HiSlang)
Loop-PKE	Pfeilartiges Hexagon	Kapselt Prozesse in Schleife
Und-Konnektor	Vertikaler Balken und gestrichelte Linien parallel	Prozess-Aufsplittung (Fork)
Oder-Konnektor	Vertikaler Balken	Prozess-Alternative (Branch)
PK-Konnektor	Vertikaler Balken und gestrichelte Linien parallel	Prozess-Synchronistaion

Tabelle 1: Auflistung ProC/B-Elemente

Das verwendete Beispiel (ProC/B-Modell) stammt aus der zweiten Phase des SFBs, in der ein GVZ modelliert worden ist [DiV03]. Ein Bestandteil dieses GVZs ist ein Lager, welches in diesem Bericht separat betrachtet wird. Abb. 1 zeigt das ProC/B-Modell des Lagers.

Dieses Modell besteht aus zwei Prozessketten. Prozesse in der oberen Prozesskette *Put* lagern Güter ins Lager (*Store*) ein und Prozesse in der unteren Prozesskette *Get* entnehmen Güter aus dem Lager. Beide Prozesse benötigen die Ressourcen *ForkLift* und *Staff* zum Einlagern bzw. Auslagern. Die Ressource *ForkLift* wird von beiden Prozessen in 90% der Fälle benötigt (Fallunterscheidung – Oder-Konnektor). Nach erfolgreichem Einlagern bzw. Auslagern werden die beiden Ressourcen, falls belegt, wieder freigeben. Das Lager (*Store*) kann maximal 10 Güter aufnehmen.

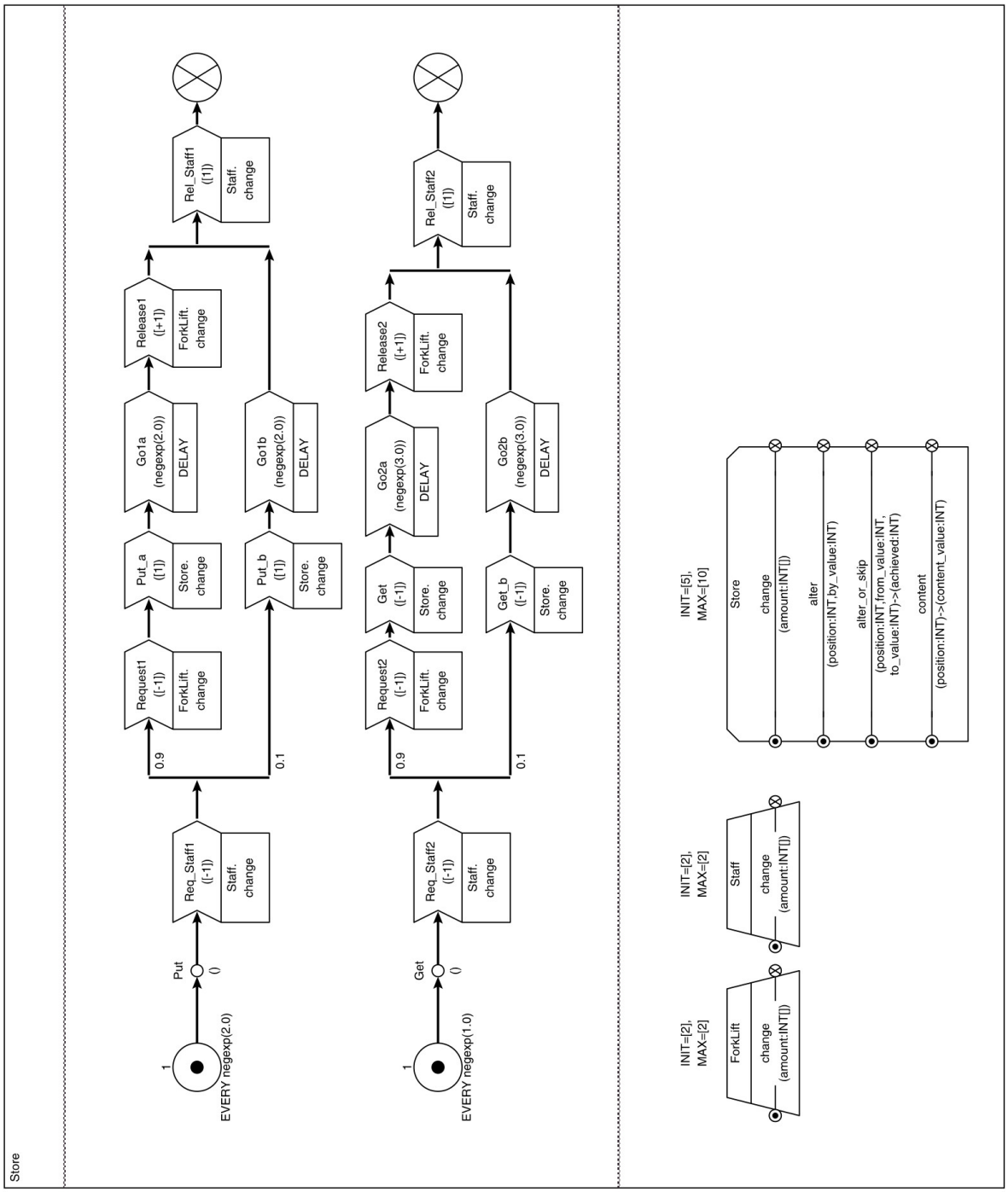


Abbildung 1: ProC/B-Modell Lager eines GVZs

3. Instrumentarium zur Prozessablauf-Visualisierung

In diesem Kapitel wird anhand des ProC/B-Modells aus Abb. 1 das Instrumentarium zur Prozessablauf-Visualisierung vorgestellt. Startet man das Instrumentarium (ProcessVis) erscheint zuerst das Hauptfenster. In Abb.2 ist dieses Fenster mit der Übersicht über die FE-Hierarchie zusehen. Die oberste Hierarchiestufe *Store* ist eine konstruierte FE. Abgeschlossen nach unten wird die FE-Hierarchie durch die drei Standardfunktionseinheiten *ForkLift*, *Store* und *Staff*.

Über dieses Hauptfenster können ProC/B-Modelle geladen werden (*File->Open*) und das Processgame (*Simulation->Processgame*) kann gestartet werden. Über den Menüpunkt *Liveness* kann eine Lebendigkeitsanalyse des ProC/B-Modells via Petri Netze durchgeführt werden. Dazu wird das ProC/B-Modell in ein PN-Modell transferiert und eine Lebendigkeitsanalyse des PN-Modells durchgeführt. Entdeckt die Lebendigkeitsanalyse eine Verklemmung (engl. Deadlock) wird ein PN-Trace generiert, der in einen ProC/B-Trace konvertiert und im Instrumentarium angezeigt wird.

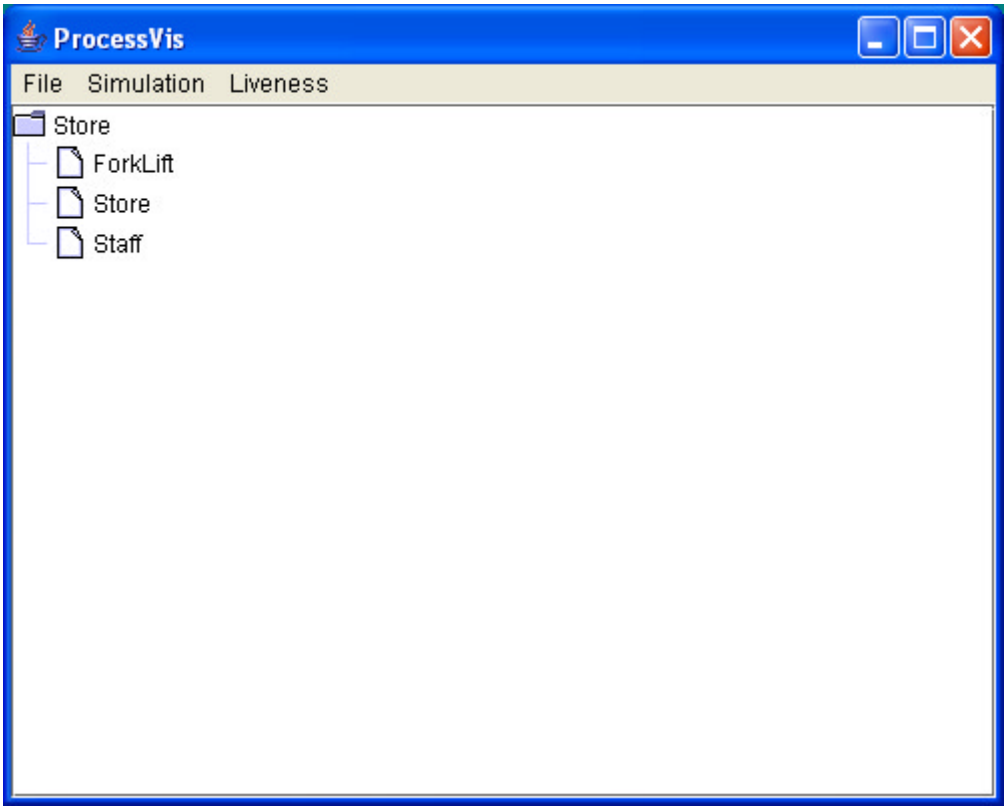


Abbildung 2: Hauptfenster des Instrumentariums

Mit dem Instrumentarium ist der Modellierer in der Lage automatisch und interaktiv einen Prozessablauf zu generieren und anschliessend vorwärts und rückwärts durchzuspielen. Es können Fehler, wie zum Beispiel Verklemmungen, entdeckt werden. Zur Entwurfsphase ist somit ein Debugging des Modells möglich, wodurch der Modellierer eine Unterstützung bei der Modellerstellung erhält.

Das ProC/B-Modell aus Abb. 1 (siehe auch Abb. 2) besitzt nur eine konstruierte Funktionseinheiten, sollten mehrere konstruierte Funktionseinheit in einem ProC/B-Modell existieren, so wird jede dieser Funktionseinheiten in einem separaten Fenster angezeigt (siehe Abb. 3).

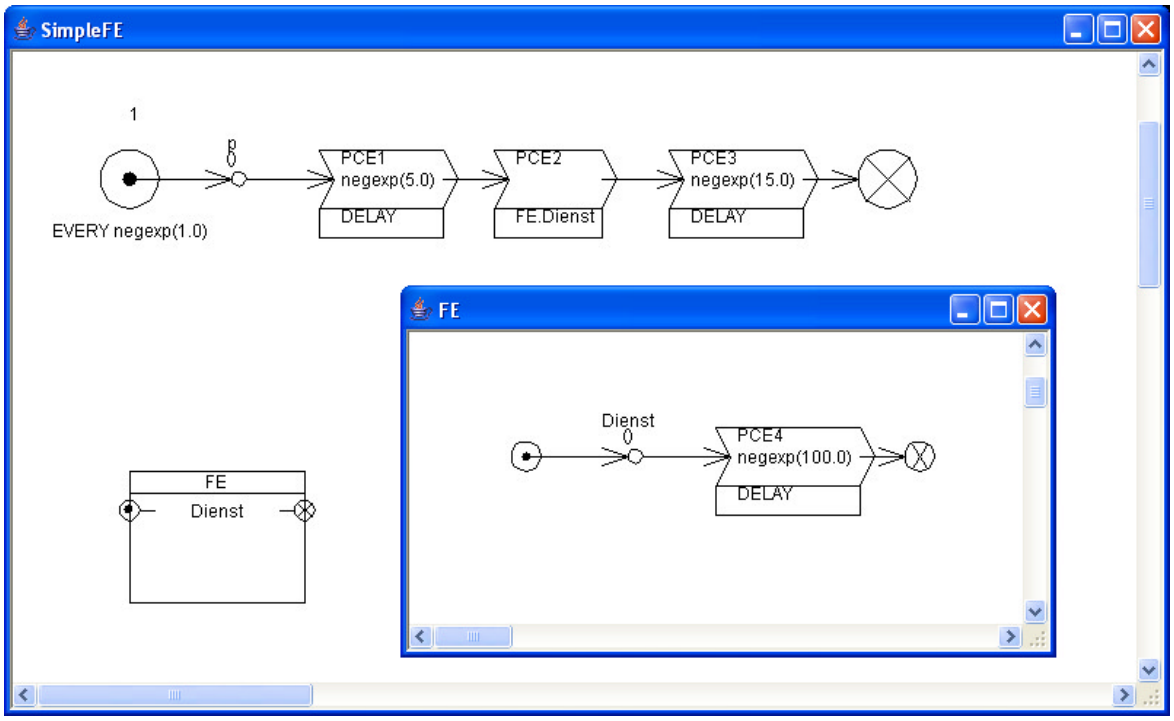


Abbildung 3: ProC/B Modell mit zwei konstruierten Funktionseinheiten

Wird das Processgame gestartet, kann der Prozessablauf über den Kontrolldialog aus Abb.4 gesteuert werden. Über diesen Dialog kann ein ProC/B-Trace geladen (*Load*) und gespeichert (*Save*) werden. Manuelles und automatisches Vorwärts- und Rückwärtsgehen durch den ProC/B-Trace ist über die beiden Buttons *Previous* und *Forward* möglich. Über den Button *Reset* kann das Modell in den Anfangszustand zurückversetzt werden. Der Button *Export2XML* speichert den ProC/B-Trace als Sequenzdiagramm im XML-Format ab und über *Exit* kann das Processgame beendet werden.

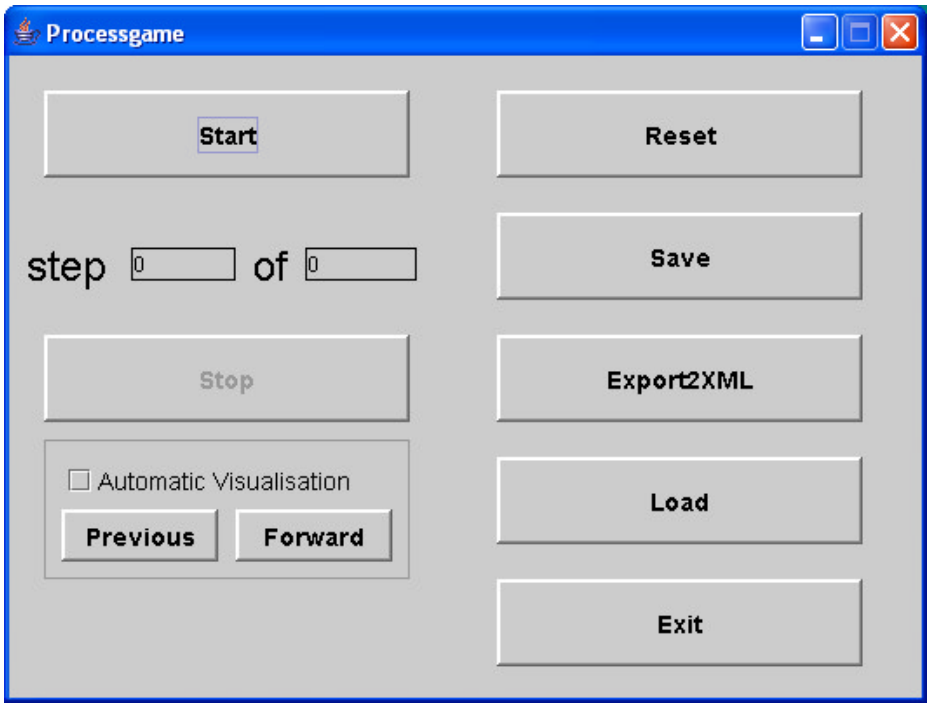


Abbildung 4: Kontrolldialog des Processgames

Über *Start* und *Stop* kann der interaktive Modus des Processgames gestartet werden. In diesem Modus ist der Modellierer in der Lage selbst Prozessabläufe zu generieren. Es kann entschieden werden, welcher Prozess als nächstes von einem PKE zum nächsten PKE wandert und dabei müssen keine Zeitrestriktionen beachtet werden. Auf diese Art entstehen natürlich Prozessabläufe, die nicht sinnvoll sind, aber durch das Spielen mit dem Modell kann der Modellierer sein Verständnis über das Modell erweitern. Des Weiteren können die so entstandenen Traces (Prozessabläufe) in einer Trace-gesteuerten Simulation verwendet werden.

Abb 5. zeigt das ProC/B-Modell aus Abb. 1 im Instrumentarium. Das Processgame ist gestartet und der interaktive Modus ist aktiviert.

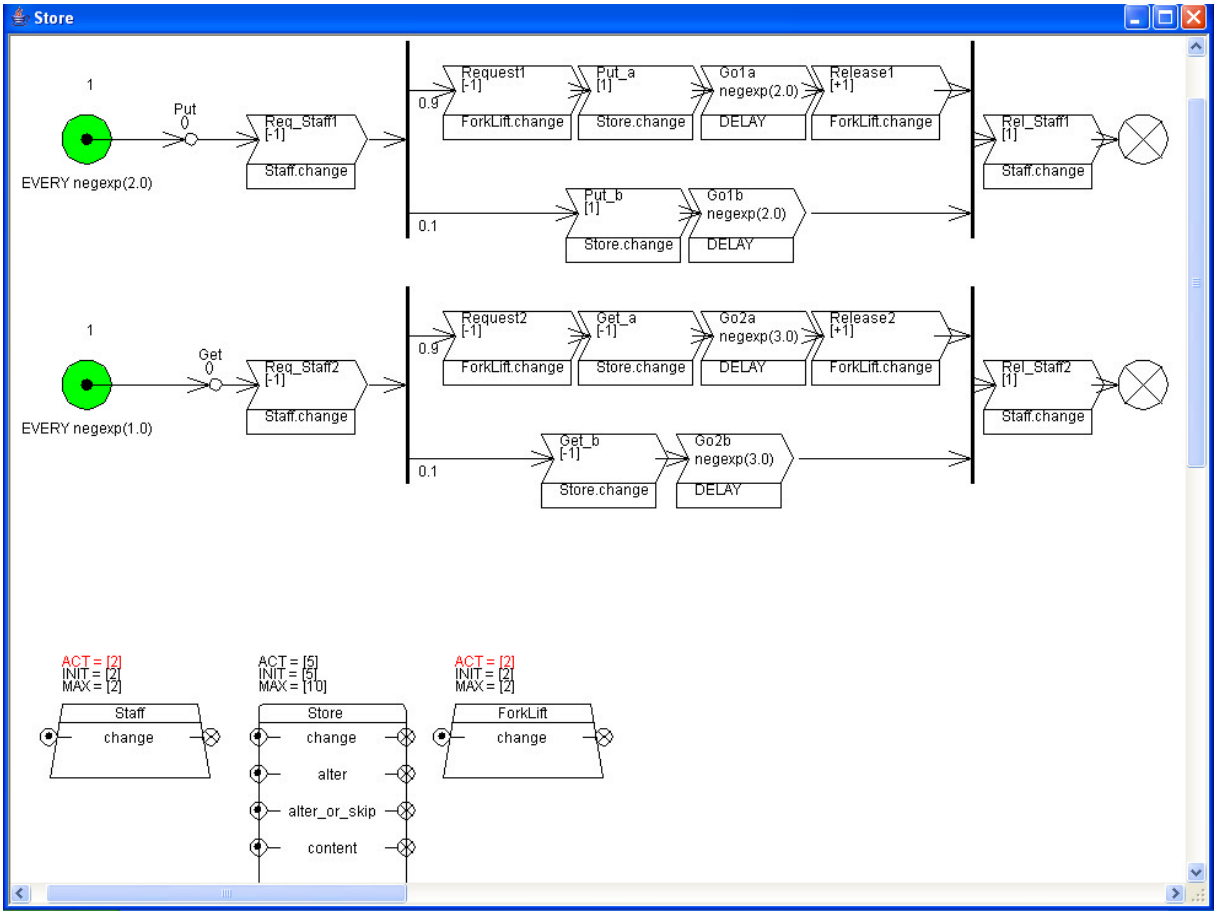


Abbildung 5: ProC/B-Modell aus Abb.1 im Instrumentarium

Zu Beginn (siehe Abb. 5) kann der Modellierer nur entscheiden, welche von den beiden Quellen einen Prozess erzeugen soll. Beide Quellen sind aktiv, da sie grün ausgefüllt gezeichnet werden. Sind PKEs aktiv, d.h. ist mindestens ein Prozess an diesem PKE und dieser Prozess kann zum nächsten PKE wandern, dann wird auch das PKE grün ausgefüllt gezeichnet. Eine Zahl über dem PKE zeigt die Anzahl der Prozesse an diesem PKE an. Können Prozesse nicht zum nächsten PKE wandern, dann werden die PKEs nicht in grün ausgefüllt, aber die Zahl über den PKEs bleibt vorhanden.

Befinden sich mehr als ein Prozess bei einem PKE, dann muss der Modellierer entscheiden, welcher Prozess zum nächsten PKE wandern soll. In diesem Fall erscheint der Dialog aus Abb. 6.

Abbildung 6: Auswahldialog

Eine weitere Auswahl muss getroffen werden, wenn der Prozess als nächstes PKE einen Oder-Konnektor (Fallunterscheidung) vorfindet. Hierbei muss entschieden werden, in welchen Prozesskettenstrang der Prozess wandern soll. Der Dialog ist dem Dialog aus Abb.6 sehr ähnlich. Nur stehen jetzt nicht Prozesse zur Auswahl sondern PKEs.

Besitzt das ProC/B-Modell mehrere konstruierte FEs, so muss entschieden werden, welchen Dienst des Prozess benutzen soll. Die Prozesse müssen sich merken, aus welcher Prozesskette der höheren Hierarchie sie den Dienst aufgerufen haben, so dass ein Rücksprung später möglich ist. Gibt es mehr als eine Hierarchieebene, so muss der Prozess sich für jede Hierarchieebene die Rücksprungstelle merken. Die Datenstruktur die hierfür verwendet wird, ist ein sogenannter Keller, der das LIFO-Prinzip (Last in, First out) realisiert.

Noch einige technische Angaben zum Instrumentarium. Das Instrumentarium ist in Java programmiert worden. Die automatische Visualisierung (siehe Abbildung 4 – Checkbox: *Automatic Visualisation*) und die automatische Erzeugung eines Prozessablaufes benutzt jeweils einen Thread. Über den Dialog aus Abb.5 kann die Zeit, die zwischen den einzelnen Schritten gewartet werden soll, eingestellt werden, sowie der Abbruch der automatischen Visualisierung bzw. automatische Erzeugung von Prozessabläufen beendet werden (Button *Stop*).

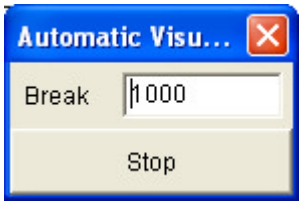


Abbildung 5: Kontrolldialog für die automatische Visualisierung und Erzeugung

4. Einträge (Schritte) im ProC/B-Trace

Im nun folgenden werden anhand der ProC/B-Elemente die jeweiligen Einträge in den ProC/B-Trace erläutert. Dabei werden die Regeln für Prozessabläufe bei ProC/B-Modellen erklärt, die zum Beispiel bei Fork/Join-Abläufen auftreten. Die Einträge/Schritte in den ProC/B-Traces entsprechen der Grammatik aus Anhang A.

Der folgende zweispaltige Auszug stammt aus einer B1-Datei. B1-Dateien beinhalten eine textuelle Beschreibung eines ProC/B-Modells. Diese textuelle Beschreibung dient als Austauschformat für verschiedene Analyseverfahren und wird ebenfalls vom Instrumentarium zur Prozessablauf-Visualisierung als Eingabeformat für ein ProC/B-Modell verwendet.

```
EXPERIMENT {

...

UNBEDQUELLE {
ID { "2" }
ANZAHL { "1" }
AKTUELLEPARAMETER CONTAINER {
} //END CONTAINER
TYP { every }
ZEITANGABE { "negexp(1.0)" }
KOMMENTAR { "" }
QUELLE {
PROZESS { "1" "p1" }
} //END QUELLE
} //END UNBEDQUELLE

...

DELAYPKE {
LINEARELEMENT {
BEZEICHNER { "pk1" }
ID { "9" }
} //END LINEARELEMENT
VORGABEZEIT { "negexp(1.0)" }
KOMMENTAR { "" }
} //END DELAYPKE

...

PKKONNEKTOR CONTAINER {
PKOEFFNENDSCHLIESSEND {
PKKONNEKTOR {
ID { "50" }
KOMMENTAR { "" }
} //END PKKONNEKTOR
} //END PKOEFFNENDSCHLIESSEND
} //END CONTAINER

...

DELAYPKE {
LINEARELEMENT {
BEZEICHNER { "pk16" }
ID { "51" }
} //END LINEARELEMENT
VORGABEZEIT { "negexp(1.0)" }
KOMMENTAR { "" }
} //END DELAYPKE
PKKONVERTRETER {
LINEARELEMENT {
BEZEICHNER { "PK-Konnektor" }
ID { "50" }
} //END LINEARELEMENT
PKKONNEKTOR { "50" }
ANZAHL { "1" }
} //END PKKONVERTRETER
DELAYPKE {
LINEARELEMENT {
BEZEICHNER { "pk17" }
ID { "52" }
} //END LINEARELEMENT
VORGABEZEIT { "negexp(1.0)" }
KOMMENTAR { "" }
} //END DELAYPKE

...

PROZESSKETTE {
ID { "53" }

...

DELAYPKE {
LINEARELEMENT {
BEZEICHNER { "pk18" }
ID { "54" }
} //END LINEARELEMENT
VORGABEZEIT { "negexp(1.0)" }
KOMMENTAR { "" }
} //END DELAYPKE
} //END LINEARELEMENT

...

} //END PROZESSKETTE
} //END EXPERIMENT
```

Anhand des Auszuges der B1-Datei können die Einträge in den ProC/B-Trace für die unbedingte Quelle aus Abb. 6 und für den PK-Konnektor aus Abb. 12 nachvollzogen werden, da die IDs im Auszug zu finden sind. Ein vollständige B1-Datei kann an dieser

Stelle nicht angegeben werden, da dieses den Rahmen dieses Dokumentes sprengen würde.

Unbedingte Quelle:

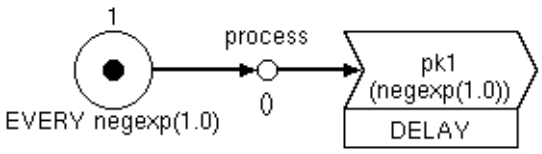


Abbildung 6: Unbedingte Quelle mit exponentiell verteilter Prozesserzeugung

Unbedingte Quellen sind selbst aktiv und erzeugen zu festen Zeitpunkten oder mit einer gewissen Verteilung Prozesse. Die unbedingte Quelle aus Abbildung 3 erzeugt Prozesse unter Verwendung einer exponentiellen Verteilung mit Rate 1.0.

Wird von einer unbedingten Quelle ein Prozess erzeugt, so entsteht folgender Eintrag im ProC/B-Trace:

\create{2 process1 9}

Es wird an der Quelle mit ID *2* ein Prozess mit ID *process1* erzeugt und dieser Prozess wandert zum PKE *pk1* mit ID *9*. Im Auszug der B1-Datei kann man die IDs der Quelle und des PKEs *pk1* finden.

Delay- und Code-PKE:

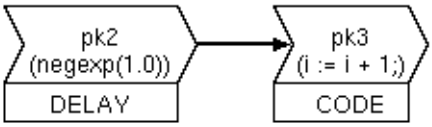


Abbildung 7: ProC/B Darstellung eines Delay- und eines Code-PKEs

Bei diesen ProC/B Elemente wandert ein Prozess einfach zum nächsten PKE weiter. Beim Code-PKE gilt es zu beachten, dass Variablen verändert werden müssen. Im Falle des Code-PKE aus Abb. 7 wird die Variable i um eins erhöht. Hinweis: Dieses ist noch nicht im Instrumentarium realisiert!

Eine Delay- und ein Code-PKE erzeugen den folgenden Eintrag im ProC/B-Trace:

\move{13 process1 25}

bzw.

\move{25 process1 33}

Der obere Eintrag wird erzeugt, wenn der Prozess mit ID *process1* vom Delay-PKE *pk2* mit ID *13* zum Code-PKE *pk3* mit ID *25* wandert. Der untere Eintrag wird erzeugt, wenn der Prozess von Code-PKE *pk3* mit ID *25* zu einem PKE mit ID *33* wandert. Das PKE mit ID *33* ist in Abb. 7 nicht dargestellt.

Öffnender Und-Konnektor:

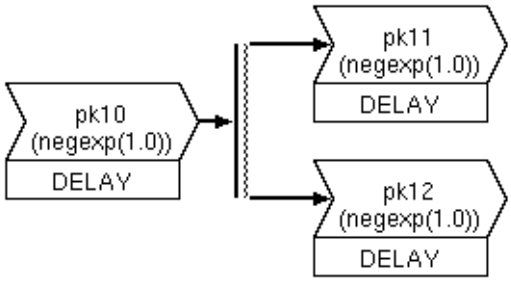


Abbildung 8: ProC/B Darstellung eines öffnenden Und-Konnektors

Dieser Konnektor ermöglicht die gleichzeitige/parallele Fortsetzung von Prozessen. In Abb. 8 wird ein Prozess durch den öffnenden Und-Konnektor aufgesplittet in die zwei PKEs *pk11* und *pk12* (Fork-Operation).

Die Aufsplittung eines Prozesses durch einen öffnenden Und-Konnektor erzeugt den folgenden Eintrag im ProC/B-Trace:

\split{\move{18 process1 34} \move{18 process1 48}}

Der Prozess mit ID *process1* wird aufgesplitet und die beiden PKEs mit ID *34* und *48* erhalten je eine Instanz des Prozesses.

Schliessender Und-Konnektor:

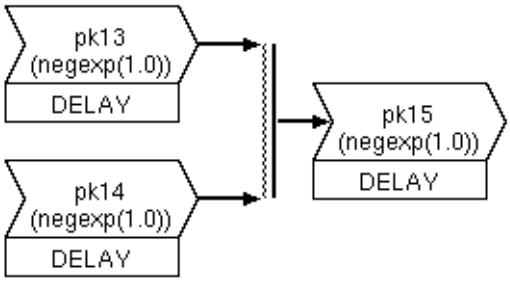


Abbildung 9: ProC/B Darstellung eines schliessenden Und-Konnektors

Mit Hilfe des schliessenden Und-Konnektors werden parallele Abläufe eines Prozesses zusammengeführt. Der Prozess kann nur dann wieder zusammengeführt werden, wenn an beiden PKEs *pk13* und *pk14* die Instanz des Prozesses angekommen ist.

Die Zusammenführung eines Prozesses durch einen schliessenden Und-Konnektor erzeugt den folgenden Eintrag in den ProC/B-Trace:

\merge{\move{56 process1 67} \move{59 process1 67}}

Der Prozess mit ID *process1* wird zusammengeführt und das PKE mit ID *67* erhält eine Instanz des Prozesses. Voraussetzung ist, dass vor dem Zusammenfügen je eine Instanz des Prozesses in den beiden PKEs mit ID *56* und *59* existiert hat.

Öffnender Oder-Konnektor:

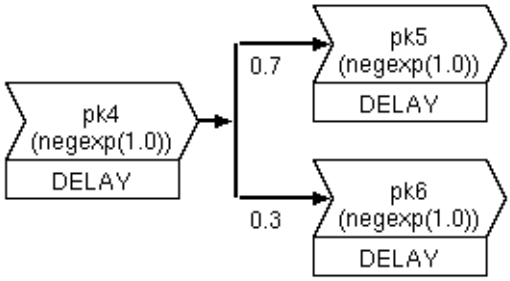


Abbildung 10: ProC/B Darstellung eines öffnenden Oder-Konnektors

Öffnende Oder-Konnektoren ermöglichen eine alternative Fortsetzung von Prozessen. Die Verzweigung kann nach Wahrscheinlichkeiten oder nach boolschen Bedingungen erfolgen. Die Verzweigungen nach Wahrscheinlichkeiten wird nicht von der automatischen Prozessablauferzeugung im Instrumentarium berücksichtigt, sondern es wird zufällig eines von beiden PKE ausgewählt.

Ein Öffnender Oder-Konnektor erzeugt lediglich einen Move-Eintrag der folgenden Form im ProC/B-Trace:

\move{69 process1 75}

oder

\move{69 process1 78}

Der obere Eintrag wird im ProC/B-Trace erzeugt, wenn der Prozess mit ID *process1* in das PKE *pk5* mit ID *75* wandert. Beim unteren Eintrag ist der Prozess in das untere PKE *pk6* mit ID *78* gewandert.

Schliessender Oder-Konnektor:

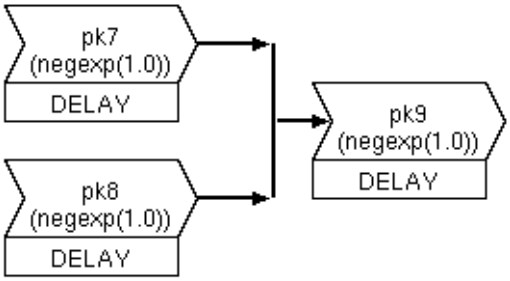


Abbildung 11: ProC/B Darstellung eines schliessenden Oder-Konnektors

Ein schliessender Oder-Konnektor führt Prozesse aus alternativen Zweigen wieder zusammen. Auch hier reicht ein einfacher Move-Eintrag im ProC/B-Trace aus.

\move{81 process1 96}

oder

\move{85 process1 96}

Der obere Eintrag im ProC/B-Trace wird erzeugt, wenn der Prozess mit ID *process1* vom oberen PKE *pk7* mit ID *81* in das PKE *pk9* mit ID *96* wandert. Beim unteren Eintrag ist der Prozess vom unteren PKE *pk8* mit ID *85* in das PKE *pk9* mit ID *96* gewandert.

PK-Konnektor:

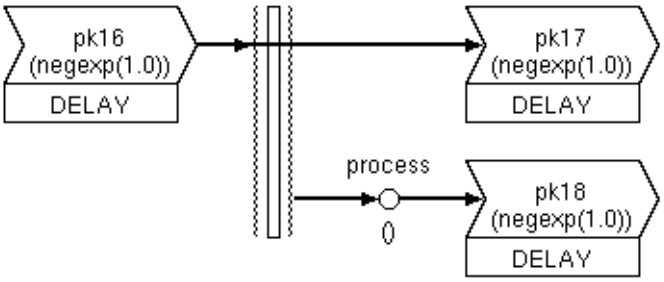


Abbildung 12: ProC/B Darstellung eines PK-Konnektors

Prozesskettenkonnektoren synchronisieren Anfänge und Enden von Prozessketten. Prozesse können am PK-Konnektor fortgeführt werden (graphisch durchgezogen), d.h. sie synchronisieren sich mit den anderen Prozessen. Nach Synchronisation wird der Prozess weiter fortgesetzt. Prozesse können auch am Konnektor enden und andere Prozesse starten.

Unterstützt wird im Moment nur die in Abb. 12 gezeigte Variante. Hier wird ein Prozess fortgeführt und ein Prozess erzeugt.

Der PK-Konnektor aus Abb. 12 erzeugt den folgenden Eintrag im ProC/B-Trace:

\split{\move{51 process1 52} \create{53 process2 54}}

Der Prozess mit ID *process1* wandert vom PKE *pk16* mit ID *51* zum PKE *pk17* mit ID *52.* Die PK-Quelle ID *53* erzeugt einen Prozess mit ID *process2*, der zum PKE *pk18* mit ID *54* wandert (IDs siehe Auszug B1-Datei).

5. Erzeugung von Prozessabläufen durch Simulation und funktionaler Analyse von Petri Netzen

Neben der Möglichkeit automatisch und interaktiv durch den Modellierer einen Prozessablauf über das Instrumentarium zu generieren, können Prozessabläufe durch Simulation und funktionaler Analyse von Petri Netzen gewonnen werden.

Als Beispiel wird in diesem Kapitel weiter das ProC/B-Modell des GVZ-Lagers aus Abb. 2 verwendet. In den nachfolgenden beiden Abschnitten wird die Erzeugung von Prozessabläufen durch den Simulator HIT und funktionaler Analyse von PN beschrieben.

5.1 Simulation

Simulation ist eine weitverbreitete Analysetechnik zur Bestimmung von quantitativen Resultaten. Im ProC/B Toolset existiert eine Abbildung von ProC/B-Modellen auf die Eingabesprache HiSlang des Simulators HIT [HIT93].

HiSlang ist eine hierarchische Modellbeschreibungssprache, die auf Simula beruht. Einen genaueren Überblick über die Transformation der ProC/B Elemente nach HiSlang liefert [BBS03].

Der ProC/B-Trace entsteht durch die Simulation, indem jedes ProC/B Element beim Verlassen eines Prozesses einen Schritt in den ProC/B-Trace schreibt. Aufgezeichnet wird der ProC/B-Trace in einem Ausgabefile. HiSlang bietet für das Schreiben in ein Augabefile den „WRITELN“-Befehl an.

Die folgenden Tabelle listet die ProC/B Elemente und die nötigen „WRITELN“-Befehl auf.

ProC/B Element	WRITELN-Befehl
Unbedingte Quelle	WRITELN FILE t,“\create{<id_source> <id_process> <id_pke2>}”;
Delay-PKE	WRITELN FILE t,”\move{<id_pke1> <id_process> <id_pke2>}”;
Code-PKE	WRITELN FILE t,”\move{<id_pke1> <id_process> <id_pke2>}”;
Update-PKE	WRITELN FILE t,”\move{<id_pke1> <id_process> <id_pke2>}”;
Aufruf-PKE	WRITELN FILE t,”\move{<id_pke1> <id_process> <id_pke2>}”;
Öffnender Oder-Konnektor	WRITELN FILE t, “\move{<id_pke1> <id_process> <id_pke2>}“;
Schliessender Oder-Konnektor	WRITELN FILE t, “\move{<id_pke1> <id_process> <id_pke2>}“;
Öffnender Und-Konnektor	WRITELN FILE t, “\split{\move{<id_pke1> <id_process> <id_pke2>} \move{<id_pke1> <id_process> <id_pke3>}}“;
Schliessender Und-Konnektor	WRITELN FILE t, “\merge{\move{<id_pke1> <id_process> \move{<id_pke2> <id_process> <id_pke3>}}“;
PK-Konnektor	WRITELN FILE t, “\split{\move{<id_pke1> <id_process> <id_pke2>} \create{<id_pksource> <id_new_process> <id_pke3>}}“;
FE Aufruf	WRITELN FILE t , “\move{<id_pke_beforeCall> <id_process> <id_first_pke_subFE>} “;
FE Terminierung	WRITELN FILE t, “\move{<id_last_pke_subFE> <id_process> <id_pke_afterCall>}”;

Tabelle1: Übersicht über die WRITELN-Befehle der ProC/B Elemente

Dabei sind:

<id_source> → ID einer Quelle

<id_pke1>, <id_pke2>, <id_pke3> → ID eines PKEs

<id_pksource> → ID einer PKQuelle

<id_process>, <id_new_process> → ID eines Prozesses

<id_pke_beforeCall> → ID des PKEs, von dem aus der Dienst der FE aufgerufen wird

<id_pke_afterCall> → ID des PKEs, nach Beendigung des Dienstes der FE

<id_first_pke_subFE> → ID des ersten PKEs des Dienstes der FE

<id_last_pke_subFE> → ID des letzten PKEs, bevor der Dienst der FE verlassen wird

Das Beispiel des GVZ-Lagers wurde 1000 Zeiteinheiten simuliert. Bei dieser Modellzeit entstand ein ProC/B-Trace aus 3320 Schritten. An dieser Stelle erkennt man, dass schon geringe Modellzeiten zu vielen Schritten und somit zu einem grossen Ausgabefile führen.

5.2 Petri-Netze

Bei der Modellierung ist die funktionale Korrektheit des Modells von zentraler Wichtigkeit. Damit bestehende Techniken anderer Modellierungsformalismen verwendet werden können, sind Transformatoren von ProC/B in andere Modellierungsformalismen notwendig. Aus diesem Grunde ist ein Transformator von ProC/B nach Petri Netzen entwickelt worden [Fit03, FKTW03].

Petri Netze sind geeignet für die funktionale Analyse und es können Verklemmungen (engl. Deadlocks) ermittelt werden. Falls eine Verklemmung existiert, wird der Ablauf dorthin vom Startzustand aus bis zur Verklemmung in einem PN-Trace mitprotokolliert.

Unter Verwendung des PN-Trace kann auf Petri Netz Ebene die Ursache für den Deadlock erkannt werden. Da aber ein Modellieren von ProC/B-Modellen in der Regel nicht mit dem Modellierungsformalismus Petri Netzen vertraut ist, ist eine Konvertierung des PN-Traces in einen ProC/B-Trace nötig.

Im nun folgenden wird anhand des Beispiels des GVZ-Lagers die Erzeugung eines Prozessablaufes für das ProC/B-Modell durch Petri Netze verdeutlicht.

Zuerst wird das ProC/B-Modell in ein GSPN-Modell transformiert. Abb. 13 zeigt das GSPN-Modell des GVZ-Lagers. Der Zeitaspekt spielt bei der funktionalen Analyse keine Rolle und somit können alle zeitbehafteten in zeitlose Transitionen transferiert werden. Hierdurch wird aus dem GSPN [ABC+95] ein Stellen-Transition-Netz (STN).

Eine der Restriktionen bei der Transformation eines ProC/B-Modells in ein Petri Netz ist, dass nur endlich viele Prozesse an der Quelle erzeugt werden können (Stellen Env_Put, Env_Get). Des Weiteren ist nötig, dass das PN-Modell kurzgeschlossen wird. Hierdurch wird der Zustandsraum des PN-Modells endlich und somit sind zustandsraum-basierte Analyseverfahren möglich.

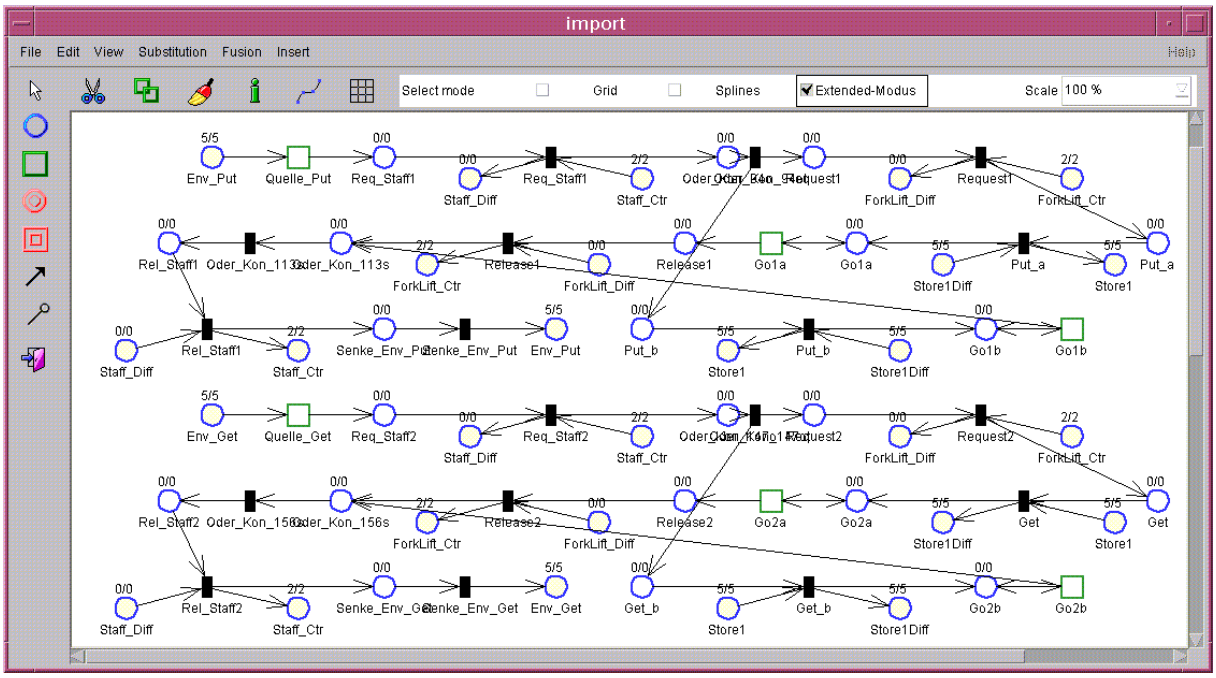


Abbildung 13: Petri Netz Modell des Lagers

Die funktionale Analyse des Petri Netzes aus Abb. 13 entdeckt einen Deadlock und erzeugt einen PN-Trace. Dieser PN-Trace kann auf Ebene des PN visualisiert werden, aber da die meisten Modellierer von logistischen Systemen nicht mit PN vertraut sind, wird dieser PN-Trace durch Algorithmus 1 in einen ProC/B-Trace konvertiert.

Algorithmus 1: Transformation eines PN-Traces in einen ProC/B-Trace

1. PN-Trace Kopf wird 1-zu-1 in ProC/B-Trace übernommen;
2. Überlese \current-Block;
3. WHILE (nächste Feuerung existiert im PN-Trace)
4. {
5. Hole Feuerung aus PN-Trace;
6. Ermittle PKE durch <ID PKE>;
7. Ermittle nächstes PKE durch <ID des nächsten PKEs>;
8. IF (nächstes PKE == Oder-Konnektor)
9. {
10. Hole nächste Feuerung aus PN-Trace;
11. Ermittle nächstes PKE durch <ID des nächsten PKE>;
12. }
13. IF (PKE == QUELLE)
14. Erzeuge CREATE-Schritt im ProC/B-Trace;
15. ELSE // restlichen PKEs liefern MOVE-Schritte
16. Erzeuge MOVE-Schritt im ProC/B-Trace;
17. }
18. PN-Trace Abspann wird 1-zu-1 in den ProC/B-Trace übernommen;

Die folgende Bedingungen müssen/werden bei der Transformation von ProC/B-Modellen nach Petri Netzen eingehalten, damit ein PN-Trace in einen ProC/B-Trace transformiert werden kann.

1. Bei der Transformation eines ProC/B-Modells in ein PN-Modell müssen die IDs der Transitionen folgendes Format besitzen:

   <beliebiger Text>_<ID PKE>.<beliebiger Text>_<ID des nächsten PKEs>

2. Die Feuerungen von Transitionen von Oder-Konnektoren, Und-Konnektoren und Senken werden nicht transformiert.
3. Der Current-Block des PN-Trace wird nicht transformiert, da bei ProC/B-Modellen der Startzustand explizit bekannt ist.
4. Können mehrere Prozesse an einem PKE ausgewählt werden, so geschieht die Auswahl, welcher Prozess zum nächsten PKE wandert, nach der Bedienstrategie FCFS. D.h. der Prozess der am längsten am PKE wartet, wandert zum nächsten PKE.

Es folgt nun der resultierende ProC/B-Trace, der aus Konvertierung des PN-Trace entstanden ist. Als Kommentare stehen die PN-Trace Einträge, damit leichter nachvollzogen werden kann, welche ProC/B-Trace Schritte aus welchen PN-Trace Einträgen entstanden sind.

```
\begin{trace}{n1} % 1-zu-1 uebernehmen
% Current-Block nicht transformieren
%\current{
% 5'pe46.schwarz +
% 5'pe49.schwarz +
% 2'pe187.schwarz +
% 5'pe173.schwarz +
% 5'pa173.schwarz +
% 2'pe59.schwarz }
\create{11 process1 192} %\fire{t49_11.Get_192}
\move{192 process1 18}   %\fire{t192_192.modus_147}
                         %\fire{tt147o_147.modus1_18} weglassen
\move{18 process1 34}    %\fire{t18_18.modus_34}
\move{34 process1 26}    %\fire{t34_34.modus_26}
\create{11 process2 192} %\fire{t49_11.Get_192}
\create{11 process3 192} %\fire{t49_11.Get_192}
\create{11 process4 192} %\fire{t49_11.Get_192}
\create{11 process5 192} %\fire{t49_11.Get_192}
\create{8 process6 188}  %\fire{t46_8.Put_188}
\create{8 process7 188}  %\fire{t46_8.Put_188}
\create{8 process8 188}  %\fire{t46_8.Put_188}
\create{8 process9 188}  %\fire{t46_8.Put_188}
\move{192 process2 139}  %\fire{t192_192.modus_147}
                         %\fire{tt147o_147.modus2_139} weglassen
\move{139 process2 143}  %\fire{t139_139.modus_143}
\move{26 process1 42}    %\fire{t126_26.modus1_42}
\create{8 process10 188} %\fire{t46_8.Put_188}
\move{42 process1 207}   %\fire{t42_42.modus_156}
                         %\fire{t156s_156.modus_207} weglassen
\move{207 process1 5}    %\fire{t207_207.modus_5}
                         %\fire{t5.modus} weglassen
\move{192 process3 139}  %\fire{t192_192.modus_147}
                         %\fire{tt147o_147.modus2_139} weglassen
\move{139 process3 143}  %\fire{t139_139.modus_143}
```

```
\move{143 process2 207}  %\fire{t1143_143.modus2_156}
                         %\fire{t156s_156.modus_207} weglassen
\create{11 process11 192} %\fire{t49_11.Get_192}
\move{207 process2 5}    %\fire{t207_207.modus_5}
                         %\fire{t5.modus} weglassen
\move{192 process4 139}  %\fire{t192_192.modus_147}
                         %\fire{tt147o_147.modus2_139} weglassen
\move{139 process4 143}  %\fire{t139_139.modus_143}
\move{143 process3 207}  %\fire{t1143_143.modus2_156}
                         %\fire{t156s_156.modus_207} weglassen
\create{11 process12 192} %\fire{t49_11.Get_192}
\move{207 process3 5}    %\fire{t207_207.modus_5}
                         %\fire{t5.modus} weglassen
\move{192 process5 139}  %\fire{t192_192.modus_147}
                         %\fire{tt147o_147.modus2_139} weglassen
\move{139 process5 143}  %\fire{t139_139.modus_143}
\move{143 process4 207}  %\fire{t1143_143.modus2_156}
                         %\fire{t156s_156.modus_207} weglassen
\create{11 process13 192} %\fire{t49_11.Get_192}
\move{207 process4 5}    %\fire{t207_207.modus_5}
                         %\fire{t5.modus} weglassen
\move{192 process11 139} %\fire{t192_192.modus_147}
                         %\fire{tt147o_147.modus2_139} weglassen
\create{11 process14 192} %\fire{t49_11.Get_192}
\move{143 process5 207}   %\fire{t1143_143.modus1_156}
                         %\fire{t156s_156.modus_207} weglassen
\move{207 process5 5}     %\fire{t207_207.modus_5}
                         %\fire{t5.modus} weglassen
\move{192 process12 139}   %\fire{t192_192.modus_147}
                         %\fire{tt147o_147.modus2_139} weglassen
\create{11 process15 192} %\fire{t49_11.Get_192}
\end{trace} % 1-zu-1 uebernehmen
```

Abb. 14 das ProC/B-Modell des Lagers im Instrumentatrium zur Prozessablauf-Visualisierung. Das ProC/B-Modell befindet sich in der Verklemmungssituation. Das Lager ist voll (Store Act = [10]) und somit werden die Prozesse der oberen Prozesskette *Put* beim PKE *Put_a* blockiert (Position 2 in Abb. 14). Diese zwei Prozesse können nur Güter ins Lager einlagern, wenn vorher Güter entnommen werden. Eine Auslagerung von Gütern ist aber nicht möglich, da die zwei Einlagerungsprozesse die Ressourcen *ForkLift* und *Staff* besitzen und auch nicht für Auslagerungsprozesse freigeben können. Somit können die beiden Quellen weiterhin Prozesse erzeugen, aber diese werden an Position 1 in Abb. 14 blockiert, da sie dort die Ressource *Staff* anfordern. Diese Art von Verklemmung tritt häufig bei der Modellierung auf und ist teilweise nur schwer durch Simulation des Modells erkennbar.

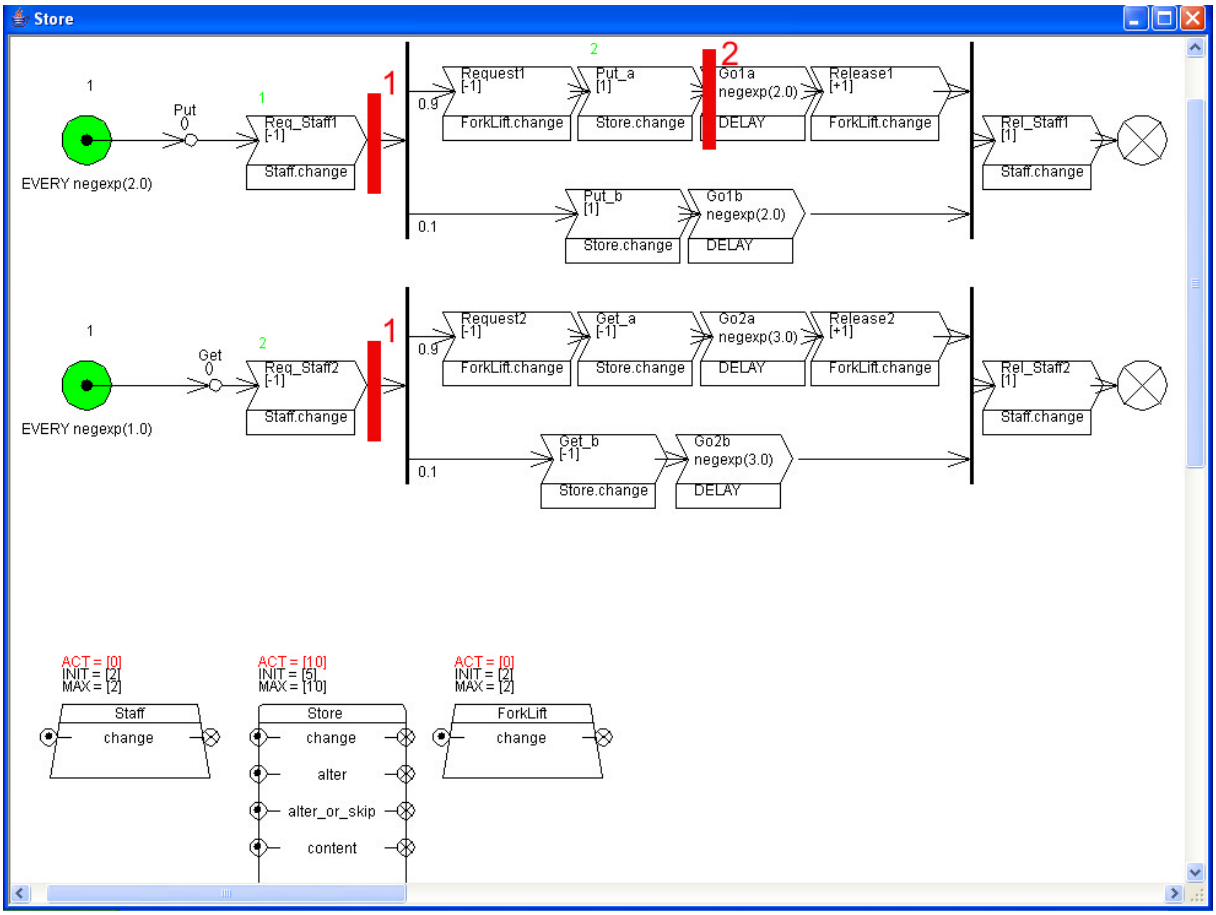


Abbildung 14: ProC/B-Modell des Lagers in der Verklemmungssituation

6. Zusammenfassung

Es wurde ein Instrumentarium zur Visualisierung von Prozessabläufen vorgestellt. Dieses Instrumentarium unterstützt den Modellierer von ProC/B-Modellen bei der Erstellung von ‚korrekten' Modellen. Es ist eine Art Debugging möglich, indem Prozessabläufe vorwärts und rückwärts betrachtet werden können. Die Prozessabläufe lassen sich durch Simulation, funktionaler Analyse von Petri Netzen und durch inaktive Benutzung des Modellierers gewinnen. Die von der Simulation erzeugten ProC/B-Traces, können ohne Transformation sofort im Instrumentarium verwendet werden. Hingegen bei der funktionalen Analyse von Petri Netzen entstandenen PN-Traces müssen in ProC/B-Traces transformiert werden.

Anhand eines Beispiels des Lagers eines Güterverkehrszentrums wurde die Funktionsweise des Instrumentariums vorgestellt. Es konnte ein Deadlock im ProC/B-Modell, mittels der durch Simulation und funktionaler Analyse von Petri Netzen generierten ProC/B-Traces, lokalisiert werden.

Weiterer Entwicklungsbedarf des Instrumentariums besteht zum einen darin, dass das Instrumentarium alle ProC/B Elemente unterstützt, so dass der ProC/B-Editor und dieses Instrumentarium auf dem gleichen Stand sind. Für das volle ProC/B-Paradigma müssen Prozessabläufe generiert und angezeigt werden können. Dieses beinhaltet auch Variablen und Veränderungen auf diesen Variablen.

Anhang A

Nicht-Terminale werden in Grossbuchstaben und Terminale in Kleinbuchstaben geschrieben. ‚empty' steht für eine leere Zeichenkette.

Grammatik ProC/B Traces:

```
TRACE ::= \begin{trace} {EXPID} CURR_PROCESSES TRACE_ITEM \end{trace}

EXPID ::= STRING

CURR_PROCESSES ::= empty
                        | \current{PROCESS} CURR_PROCESSES

PROCESS ::= \process{PROCESS_ID} \at{PKE_ID}

TRACE_ITEM ::= empty
                    | CURR_PROCESSES TRACE_ITEM
                    | MOVE TRACE_ITEM
                    | CREATE TRACE_ITEM
                    | SPLIT TRACE_ITEM
                    | MERGE TRACE_ITEM

MOVE ::= \move{PKE_ID PROCESS_ID PKE_ID}

CREATE ::= \create{SOURCE_ID PROCESS_ID PKE_ID}

SPLIT ::= \split{ SPLIT_CONTAINER }
SPLIT_CONTAINER ::= empty
                        | MOVE SPLIT_CONTAINER
                        | CREATE SPLIT_CONTAINER

MERGE ::= \merge{ MERGE_CONTAINER }


MERGE_CONTAINER ::= empty
                                      | MOVE MERGE_CONTAINER

PKE_ID ::= STRING

PROCESS_ID ::= STRING

SOURCE_ID ::= STRING
```

Anmerkung: CURR_PROCESSES ist meistens nicht notwendig, da bei ProC/B-Modellen der Startzustand der Anfangszustand ist.

Anhang B

Nicht-Terminale werden in Grossbuchstaben und Terminale in Kleinbuchstaben geschrieben. ‚empty' steht für eine leere Zeichenkette.

Grammatik Petri Netz Trace:

```
TRACE           ::= \begin{trace}{NETID} CURR_MARKING TRACE_ITEM
\end{trace}

NETID           ::= STRING

TRACE_ITEM      ::= empty
                    | TRANS_FIRE      TRACE_ITEM
                    | CURR_MARKING    TRACE_ITEM
                    | SUCC_MARKING    TRACE_ITEM
                    | CHANGED_MARKING TRACE_ITEM
                    | SERVICE         TRACE_ITEM
                    | FIRING_TIME     TRACE_ITEM


TRANS_FIRE      ::= \fire{ID}{NAME COLOR}
FIRING_TIME     ::= \time{ REAL }

SERVICE         ::= \serve{ID}{NAME COLOR}

CURR_MARKING    ::= \current{ MARKING }
SUCC_MARKING    ::= \succ{ MARKING }
CHANGED_MARKING ::= \add{ MARKING }


ID              ::= STRING
NAME            ::= empty | \name{ STRING }
COLOR           ::= empty | \color{ STRING }


MARKING         ::= empty
  | \place{ ID }{ NAME MULTISET } MARKING

MULTISET        ::= INTEGER | INTEGER`STRING MSLIST
MSLIST          ::= empty | + MULTISET
```

Literaturverzeichnis

[ABC+95] Ajmone Marsan, M.; Balbo, G.;Conte, G.;Donatelli, S.;Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. Wiley, 1995.

[AEF03] Arns, M.; Eickhoff, M; Fischer, M; Tepper, C; Völker, M.: New Features in the ProC/B toolset. In: [Bau03c], S. 26-29, 2003.

[AFK03] Arns, M.; Fischer, M.; Kemper, P.: Anwendung nicht-simulativer Techniken zur Analyse eines dezentralen Güterverkehrszentrums. SFB 559-Bericht 03017, ISSN 1612-1376, 2003.

[AFT01] Arns, M.; Fischer, M.; Tatlitürk, H.; Tepper, C.; Völker, M.: Modeling and Analysis Framework of Logistic Process Chains. In. Proc. Of Joint Tool Session at PNPM/MMB/PAPM Conferences, pp. 56-61, Aachen, Germany, Sept. 2001.

[BaB99] Bause, F.; Beilner, H.: Intrinsic Problems in Simulation of Logistic Networks. Simulation in Industry, 11th European Simulation Symposium and Exhibition (ESS99). SCS Publishing House, 1999.

[Bau03c] Bause, F.: Tools of the 2003 Illinois International Multiconference on Measurement, Modelling, and Evaluation of Computer-Communication Systems. Forschungsbericht Nr. 781 des Fachbereichs Informatik der Universität Dortmund, 2003.

[BBF+02] Bause, F.; Beilner, H.; Fischer, M.; Kemper, P.; Völker, M.: The ProC/B Toolset for the Modelling and Analysis of Process Chains. In T. Field, P.G. Harrison, J. Bradley, U. Harder (eds.) Computer Performance Evaluation Modelling Techniques and Tools (Proc. Performance TOOLS 2002), Springer, LNCS 2324, pp. 51-70, 2002.

[BBS03] Bause, F.; Beilner, H.;Schwenke, M.: Semantik des ProC/B-Paradigmas. SFB 559-Bericht 03001, ISSN 1612-1376, 2003.

[BMW94] Beilner, H.; Mäter, J.;Wysocki, C.: The Hierarchical Evaluation Tool HIT. 7th Int. Conference on Modelling, Techniques and Tools for Computer Performance Evaluation, 1994.

[DiV03] Dilling, C.; Völker, M.: Beispielmodellierung eines Güterverkehrszentrums im ProC/B-Paradigma. SFB 559-Bericht 03016, ISSN 1612-1376, 2003.

[FiT03] Fischer, M.; Tepper, C.: GSPNs to support aggregation in the ProC/B Toolset. In P. Kemper (ed.): Workshop on Stochastic Petri nets and related formalisms. Technical Report 780, Uni Dortmund, Fachbereich Informatik, 2003.

[FKTW03] Fischer, M.; Kemper, P.; Tepper, C.; Wu, Z.: Abbildung von ProC/B nach Petri-Netzen – Version 2. SFB 559-Bericht 03011, ISSN 1612-1376, 2003.

[HIT93] Büttner, M. (ed.);Fricke, B.;Klaßen, O.;Nolte, S.;Stahl, H.;Weißenberg,N.: Hi-Slang Reference Manual for the Hierarchical Evaluation Tool HIT. Universität Dortmund, Informatik IV, 1993.

[Kuhn95] Kuhn, A.; Prozessketten in der Logistik, Entwicklungstrends und Umsetzungsstrategien. Verlag Praxiswissen, Dortmund, 1995.