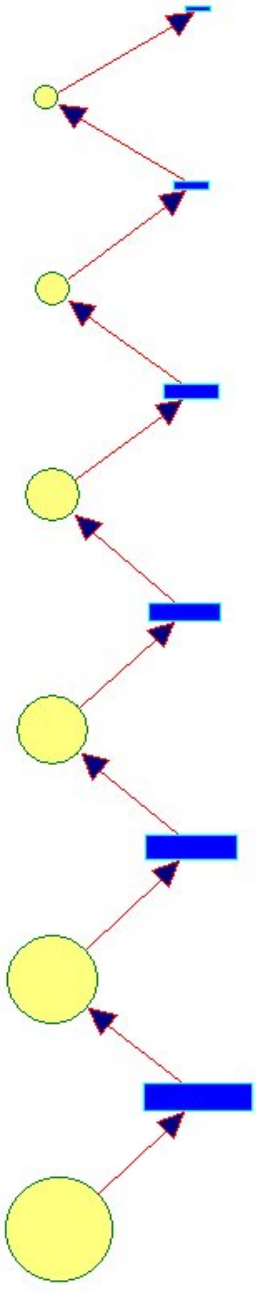


# Starten mit *PACE*

**Tutorial**

**IBE Simulation Engineering GmbH**

Verfasser: B. Eichenauer

**IBE** Simulation Engineering GmbH
Postfach 1142
D-85623 Glonn
Germany
Tel.: +49-(0)8093-5000
Fax: +49-(0)8093-902687
E-mail: info@ibepace.com
Home: http://www.ibepace.com

Diese Einführung wurde für PACE 2008 erstellt.

# *Inhaltsverzeichnis*

# 1. Vorbemerkungen

**Übung macht den Meister.**
Sprichwort

## 1.1 Allgemeines zum Starter

Wenn man ein neues Programm mit komplexer Funktionalität erwirbt, dann dauert es normalerweise mehrere Wochen, bis man mit ihm vertraut ist und man es effizient einsetzen kann. Die Vielzahl der Beschreibungsmittel und ihre adäquate Verwendung werden erst zum „Handwerkszeug“, wenn man ihre Wirkung an verschiedenen Beispielen gesehen und erprobt hat, die Verwendung der Sprachmittel sozusagen „in Fleisch und Blut“ übergegangen ist.

Die folgende Einführung hat sich zum Ziel gesetzt, diesen anfänglich mühsamen Weg durch eine praxisnahe Darstellung zu erleichtern und zu beschleunigen. In den Beispielen werden zahlreiche Eigenschaften von PACE beschrieben und verwendet, sodass der Leser nach der Durcharbeitung des Starters ein Grundwissen über PACE besitzt und eigene Modelle erstellen kann. Die anschließende Vervollkommnung sollte anhand der PACE-Handbücher, die während der Modellierung fallweise zu Rate zu ziehen sind, problemlos möglich sein.

Damit die bei der Modellierung durchzuführenden Einzelschritte verständlich werden, ist eine kurze Darstellung der Netz-Modellierung unumgänglich. Dabei wird hier versucht, die Angelegenheit möglichst praxisnah darzustellen und auf solche theoretischen Erörterungen zu verzichten, die in der praktischen Anwendung nicht oder nur selten hilfreich sind.

Bei der Einführung in die Netzmodellierung und in die Modellierung und Simulation mit PACE wird ein einfaches Beispiel, ein Autowaschplatz, verwendet. Der Leser wird dabei Schritt für Schritt, Mausklick für Mausklick durch die Modellierung und Simulation der Prozesse geführt. Danach wird gezeigt, wie mit PACE konfigurierbare, wiederverwendbare Bausteine erstellt und verwendet werden. Anschließend werden noch einige nützliche Eigenschaften von PACE beschrieben.

Nach dieser Einführung in die Handhabung von PACE werden die Verfahren beschrieben, die in PACE für die Optimierung von Modellen vorgesehen sind. Auch dabei wird zunächst ein einfaches Beispiel verwendet, das so gewählt wurde, dass die zu beschreibenden Verfahren nicht durch das Beispiel verdeckt werden.

Die PACE-Optimierungsverfahren werden danach an einem etwas umfangreicheren Beispiel demonstriert. Damit der Aufwand bei der Modellerstellung in vernünftigen Grenzen bleibt, musste die Aufgabenstellungen etwas vereinfacht werden. Es bleibt dem Leser überlassen, ob er diese etwas komplexeren Optimierungsmodelle nachmodellieren oder nur studieren will. Alle Beispiele finden sich auf der PACE-CD im Verzeichnis "samples\new starter" und lassen sich nach der Installation von PACE durch Doppelklick auf das jeweilige Modell mit Extension **.imm** starten.

## 1.2 Warum braucht man Simulationsmodelle?

Diskussionen zum Thema Geschäftsprozess-Modellierung beschäftigen sich häufig weniger mit den durch moderne Entwicklungstools erreichbaren Prozessverbesserungen, als mit der angeblichen Komplexität der erstellten Modelle. So wurde z.B. im Mai 2001 auf einer Tagung über Geschäftsprozesse in Stuttgart ein Round-Table-Gespräch mit dem Thema: „Wie komplex darf Geschäftsmodellierung sein?“ geführt.

Es ist eine weit verbreitete Annahme, dass man sich, insbesondere bei der Planung von Geschäftsmodellen, auf eine grobe Modellierung der in Frage stehender Sachverhalte beschränken könne und dass für eine detaillierte Simulation der Sachverhalte keine Notwendigkeit bestehe. Dafür werden regelmäßig folgende Argumente angeführt:

- Für die anstehende Aufgabe reicht eine einfache Modellierungsmethode, die es ermöglicht, Modelle einfach und flexibel zu ändern.
- Modelle für die Simulation erfordern zahlreiche zusätzliche Angaben und sind deshalb erheblich komplexer als nötig.
- Simulationstools sind zu IT-lastig und deshalb keine Modellierungstools für Geschäftsprozesse.

Dabei versucht man bei der heute üblichen Modellierung von Geschäftsprozessen die Vorgänge so zu vergröbern, dass die Modelle einfach werden. Daran ist wenig auszusetzen, solange die Modelle lediglich zur Dokumentation der Prozesse eingesetzt werden. Leider ist es aber gängige Praxis, sie auch, häufig unter Verwendung von Spreadsheets, für die Planung von Prozessen zu verwenden.

Nun sollen hier die großen Möglichkeiten von Spreadsheets für viele Anwendungen, wie z.B. für die Daten-Ein/Ausgabe, für die Kostenplanung und die Kostenanalyse, keineswegs in Abrede gestellt werden. Doch ist es kaum möglich, mit solchen statischen Hilfsmitteln die vielfältigen organisatorischen Abläufe und Zusammenhänge, wie sie beispielsweise in Fertigungs- und Produktionsanlagen oder in logistischen Aufgabenstellungen auftreten, transparent zu beschreiben, zu modellieren und ganzheitlich zu simulieren. Letzteres ist unter anderem notwendig, um die verschiedenen sequentiellen Abläufe darzustellen, aufeinander abzustimmen (zu synchronisieren) und um die jeweils erforderlichen Ressourcen zu finden und zu verteilen.

So zeigen viele Beispiele, die man gelegentlich sogar der Tagespresse entnehmen kann, dass die Planung von Vorhaben nicht mit der erforderlichen Genauigkeit vorgenommen wird. Ein großer Teil der Management-Entscheidungen in Industrie und Verwaltung werden bis heute wegen der fehlenden objektiven Entscheidungskriterien aus dem „hohlen Bauch“ heraus gefällt. Der durch Fehlentscheidungen verursachte Schaden lässt sich kaum abschätzen und wird nur in seltenen Fällen (z.B. im öffentlichen Bereich durch die Rechnungshöfe) registriert und evaluiert.

Das Standard-Beispiel für eine mangelhafte Planung ist natürlich das häufig beschriebene Desaster am Flughafen Denver, das Unsummen an Nachfolgekosten verschlun-

gen hat.[1] Aber auch in jüngerer Zeit konnte man etwa ein halbes Jahr nach einer größeren Umorganisation einer deutschen Großfirma in der Presse lesen, die Umorganisation habe nicht den gewünschten Erfolg gezeigt und man plane, sie teilweise wieder rückgängig zu machen. Durch Einsatz moderner Simulationsverfahren im Vorfeld der Umorganisation für vergleichsweise vernachlässigbare Kosten hätte man hier wenigstens die enormen Nachfolgekosten vermieden. Man hätte die geplante Zielorganisation schon vor ihrer Realisierung studieren können und damit schon vorher gewusst, ob die Änderung der Organisation in der vorgesehenen Weise erfolgversprechend sein kann oder ob es vielleicht bessere Alternativen gibt.

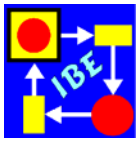
Für eine erfolgreiche Planung von Geschäftsprozessen benötigt man exakte Simulationsmodelle, die gute Vorgaben für schwierige Entscheidungen und Planungen machen. Diese Modelle sind dann eben so komplex wie sie sein müssen, damit sie diese Aufgabe zufriedenstellend lösen.

Das bedeutet nicht, dass die Modelle notwendiger Weise unangemessen aufwendig und teuer sind, sondern nur, dass für ihre Erstellung ein geeignetes Tool und ein entsprechendes Fachwissen erforderlich ist. Zumeist amortisieren sich die Kosten für die Modellerstellung und Simulation sehr schnell durch den reibungslosen und hinsichtlich der Ressourcen optimalen Ablauf der Prozesse, der andernfalls, wenn überhaupt, nur iterativ erreicht wird.

Diese Iteration kann sehr teuer sein. So hat eine größere deutsche Firma für die Einführung eines gängigen IT-Produkts 3 Jahre benötigt, bis die Bereitstellung und der Service einigermaßen reibungslos vonstatten ging. Während dieser Zeit gab es zahlreiche Probleme mit der Zuteilung von Ressourcen und viele Crash-Actions zur Behebung von Engpässen, ganz abgesehen von der Unzufriedenheit der Kunden und ihrer Abwanderung. Ein großer Teil der aufgetretenen Probleme wären sicher vermieden worden, wenn man die geplante Organisation vorab mit einem Simulationsmodell untersucht hätte.

Zum Schluss ein kleines überschlägiges Beispiel. Für die Erstellung der in Kapitel 7 dieses Tutorials beschriebenen einfachen Fertigungsoptimierung vom Scratch wird ein mit PACE vertrauter Bearbeiter etwa zwei Arbeitstage zu 8 Stunden benötigen. Bei einem Stundenpreis von 125 € für einen IT-Fachmann betragen die Kosten dafür 2000 €. Betrachtet man den einfachsten Fall, dass nur ein Facharbeiter mehr als nötig vorgesehen wird, so kostet das pro Woche, wenn man einen Stundenpreis von 50 € und eine Wochenarbeitszeit von 40 Stunden zugrunde legt, ebenfalls 2000 €. Die Erstellung des Simulationsmodells hätte sich damit schon nach einer Woche amortisiert!

[1] Kurz nach der Erstellung des vorliegenden Tutorials ereignete sich ein weiteres großes ziviles Desaster der angegebenen Art: das neue Terminal am Londoner Flughafen Heathrow. Nach den Berichten über die Zustände im Terminal muß man annehmen, daß auch hier bei der Planung keine hinreichende Modellierung und Simulation durchgeführt wurde. Wie anders könnte man sonst die unzureichende Verfügbarkeit von Ressourcen (z.B. für den Gepäcktransport oder beim Check-In von Fluggästen) erklären?




## 1.3 Die PACE-Vorgehensweise

Die in PACE verwendete Vorgehensweise basiert auf der Vorstellung, das man zu realistischen Aussagen über das Verhalten von Realzeitsystemen gelangen kann, wenn man diese Systeme so in Programme abbildet, dass sich die Programme genau wie die Realität verhalten, d.h. sowohl hinsichtlich der Abläufe als auch der dabei auftretenden Prozessdaten ein identisches Verhalten aufzeigen. Ist das gelungen, so kann man mit den Programmen kostengünstige Experimente durchführen, dabei erfahren, wie die Realität auf bestimmte äußere Einflüsse bzw. Umgebungsszenarien reagiert und wie die optimale Parametrierung hinsichtlich bestimmter Kriterien (Einsatzzeiten und -kosten, Produktionskosten, usw.) aussieht. In diesem Starter wird ein ingenieurmäßig handhabbarer Weg beschrieben, wie man Realzeitsysteme in derartige Simulationsprogramme umsetzen kann.

Bevor auf die Realisierung der vorgeschlagenen Methodik eingegangen wird, soll zum besseren Verständnis kurz der Unterschied zu anderen Verfahren skizziert werden. Bei älteren Verfahren wird das Verhalten des zu modellierenden Systems zunächst auf mathematische Rechenvorschriften abgebildet. Diese werden dann durch Programmierung in einer Simulationssprache (z.B. GPSS, Simula) oder unter Verwendung von Rechentools (z.B. Spreadsheets) computerisiert. Andere Verfahren zielen mehr in die hier vorgeschlagene Richtung und modellieren die Abläufe, ohne indessen alle für die realitätsidentische Ausführung der Abläufe erforderlich Eigenschaften vorsehen zu können (z.B. Flussdiagramm-Tools, CASE-Tools, Bausteinsysteme). Während mit Flussdiagramm-Tools und den meisten CASE-Tools überhaupt nur die statischen Aspekte von Prozesssystemen darstellbar sind, die dynamischen Aspekte höchstens in Form von Kommentaren beigefügt werden, berücksichtigen Bausteinsysteme auch teilweise die dynamischen Aspekte. Das Modell eines Prozesssystems wird mit vorgefertigten Bausteinen aufgebaut, die jeweils mit Parametern mehr oder minder gut an die realen Bedingungen angepasst werden. Jedem Baustein ist eine Rechenvorschrift zugeordnet, die ausgeführt wird, wenn der Baustein aufgerufen wird. Eine echte Simulation wie in PACE, bei der Objekte realitätsgerecht und animierbar durch eine virtuelle Anlage laufen, findet nicht statt. Auf die Problematik der Verwendung von Bausteinsystemen wird zu Beginn von Kapitel 4 kurz eingegangen.

Wesentlich an der PACE-Methodik ist, dass sowohl die parallelen Aktivitäten (Prozesse) als auch das Verhalten der Objekte, welche sie bearbeiten, sowohl statisch als auch dynamisch detailgenau nachgebildet werden. Der erforderliche Detaillierungsgrad richtet sich dabei nach der Genauigkeit, mit der die Ergebnisse benötigt werden.

Die Untersuchung der Beschreibungsmittel zur Darstellung von Realzeitsystemen führte zu drei Arten von Sprachelementen und zur Entwicklung einer halbgrafischen Modellierungssprache MSL, die Bestandteil von PACE ist. Die strukturelle Übereinstimmung zwischen Realität und Modell wird in PACE dadurch erreicht, dass die visuelle Struktur einer Anwendung mit einem Grafikeditor eins zu eins in ein Modell übertragen wird. Damit sich ein Modell auch funktionell identisch verhält, muss es ausführbar sein, d.h. als Simulationsmodell ausgelegt werden. Bei den drei Arten von Sprachelementen handelt es sich um

- Sprachelemente zur Beschreibung der Struktur einer Anwendung samt der Prozessbahnen in dieser Anwendung.

- Sprachelemente zur Darstellung der Objekte, die auf den Prozessbahnen bewegt werden.

- Sprachelemente für die Verarbeitung der Objekt- und sonstigen Prozessdaten.

Sprachelemente zur Darstellung von Prozessbahnen kann man von den Petri-Netzen übernehmen. In PACE werden die Verarbeitungsschritte mit verallgemeinerten Petri-Netzelementen beschrieben. Die Verallgemeinerung besteht darin, dass die Petri-Netzelemente mit zahlreichen Attributen versehen werden, welche die genaue Verarbeitung der Objekte an den Netzknoten festlegen. Durch Attributierung können beispielsweise bestimmte Netzelemente (sog. Transitionen) mit Programmcode versehen werden, der die Bearbeitung eines realen Objekts nachbildet.

Außer in den Prozessschritten muss sich auch die Systemstruktur des realen Systems im Modell widerspiegeln. Deshalb wurden in PACE zwei weitere Netzelemente, Module (Bausteine) und Kanäle, für die hierarchische Strukturierung von Netzen eingeführt.

Nachdem die Struktur und die Prozessbahnen dargestellt wurden, müssen die dynamischen Objekte eingeführt werden, die sich auf diesen Prozessbahnen bewegen. Sie werden im Modell durch Marken oder Behälter dargestellt, die auf den Prozessbahnen durch das Netz laufen und die im Modell benötigten charakteristischen Eigenschaften der realen Objekte als Attribute (Argumente oder Parameter) tragen. Durch den oben schon erwähnten Programmcode werden die Objektdaten beim Durchlaufen des Netzes verändert und damit die Bearbeitung bzw. die Veränderung der Objekte gemäß der Realität nachgebildet.

Schließlich muss noch die Script- bzw. Programmiersprache festgelegt werden, in der alle Netz-Inskriptionen (Programmcodes, Namen, Kommentare, usw.) niedergelegt werden. Da PACE selbst weitgehend in der objektorientierten Programmiersprache Smalltalk geschrieben ist, lag es zur Vermeidung von Schnittstellenproblemen nahe, als Script-Sprache die gleiche Sprache bzw. einen Subset davon vorzusehen. Damit steht für die Bearbeitung der Modelldaten eine einfach zu handhabende Sprache mit einer sehr umfangreichen Methodenbibliothek zu Verfügung.

Die verwendete Vorgehensweise erfüllt zwei weitere wichtige Forderungen, welche die Modellerstellung und Modelländerung betreffen, nämlich die nach einfachen Testmöglichkeiten und die nach Flexibilität der Modellierung. Beide Forderungen erleichtern und beschleunigen die Modellerstellung und Modellnutzung erheblich. Durch die schrittweise Abbildung der Abläufe kann in einem Debugging-Modus sehr schnell, gegebenenfalls mit einem Fachmann, der die modellierten Prozesse genau kennt, festgestellt werden, ob sich das Modell realitätsidentisch verhält, indem die parallel ablaufenden Prozessschritte und die dabei anfallenden Daten schrittweise analysiert werden.

# 2. Grundlagen

## 2.1 Netzbeschreibung von diskreten Prozessen

Die Computer-gestützte Simulation unternehmensweiter Prozesse steht heute erst am Anfang ihrer Entwicklung und nur wenige Programme sind in der Lage, die dabei zu betrachtenden komplexen Vorgänge umfassend und nachvollziehbar darzustellen. Das Programm PACE mit seiner seit mehr als 15 Jahren ständig fortentwickelten halbgrafischen **M**odeling and **S**imulation **L**anguage **MSL** hat sich bei der Modellierung, Simulation und Optimierung von Prozessen in Industrie und Forschung gleichermaßen bewährt. Es basiert auf der 1962 von Carl Adam Petri erfundenen Netzbeschreibung von Informationsflüssen und wurde für die detailgenaue Modellierung von Geschäftsprozessen um zahlreiche Beschreibungselemente erweitert. Hierzu gehören die Fähigkeit, Netzknoten durch Einfügen von Code in beliebige Verarbeitungsobjekte zu wandeln, gehören graphische und statistische Bibliotheken, Methoden der Fuzzy-Logik, Methoden für die visuelle und automatische Netzoptimierung, Bausteine für wiederkehrende Aufgabenstellungen und zahlreiche Möglichkeiten für die einfache Dateneingabe und Ergebnisvisualisierung.


Abb. 2.1: Gesamtbild der Autowaschanlage

Petri-Netze lassen sich aus der einfachen Vorstellung herleiten, dass sich die meisten Vorgänge in diskrete Einzelvorgänge zerlegen lassen, die sich in Form von Netzen (gerichteten Graphen) darstellen lassen.

Um die Elemente eines klassischen Petri-Netzes abzuleiten, wird im Folgenden ein einfacher Vorgang, nämlich der Waschvorgang in einer Auto-Waschanlage betrachtet, der in diskrete Einzelvorgänge zerlegt werden soll. Uns interessiert hier aber nicht der Waschvorgang selbst, was also beim Waschen mit einem Auto passiert, sondern nur das Verhalten der Waschanlage unter verschiedenen Belastungen. Deshalb wird das Auto selbst als unveränderliches Objekt angenommen, welches durch die Waschanlage bewegt wird. Dabei ändert sich der Zustand der Waschanlage.

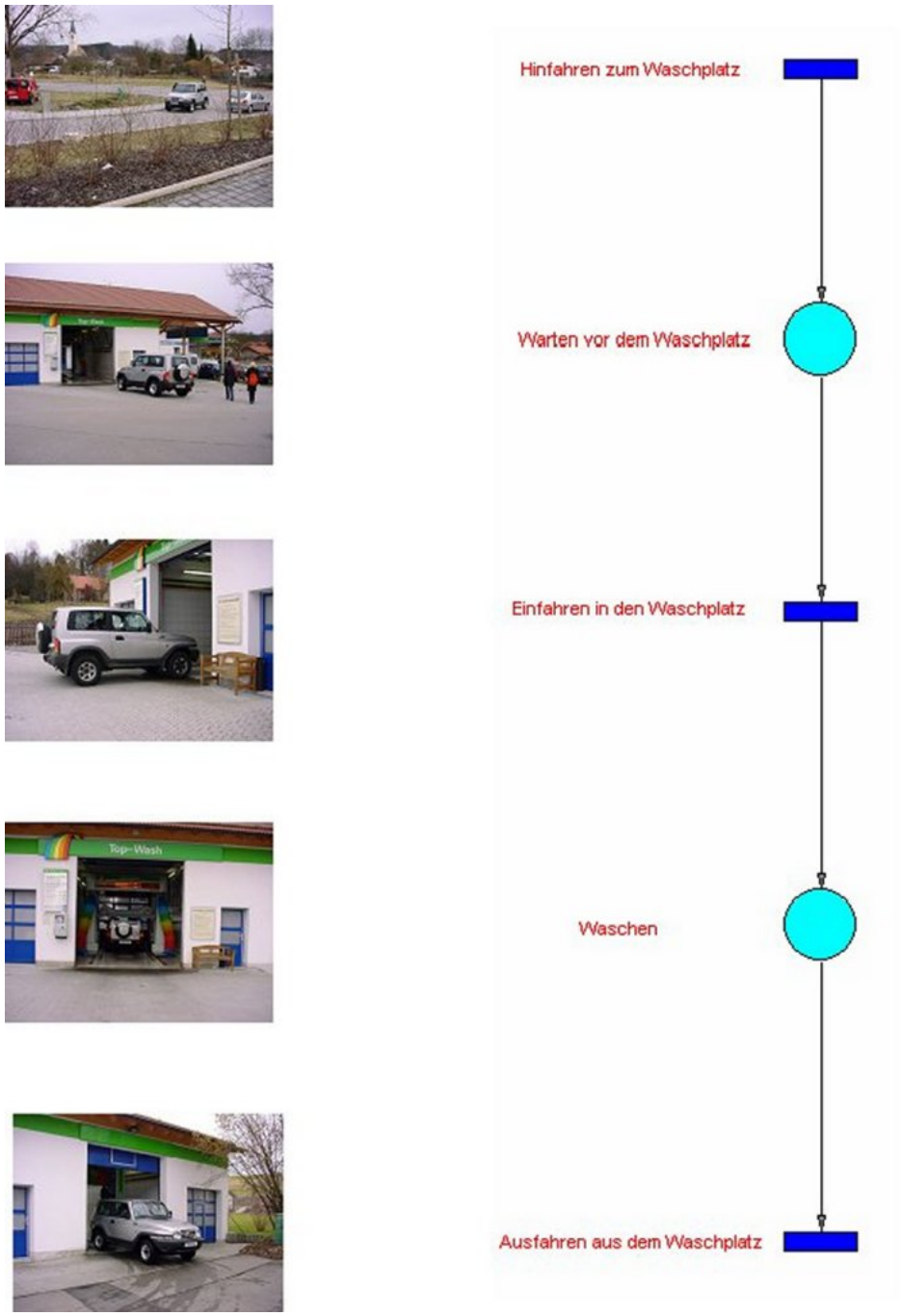



Abb. 2.2: Zerlegung des Waschvorgangs in Einzelvorgänge

Abb. 2.1 zeigt das Gesamtbild einer Autowaschanlage. Von rechts fahren Autos auf

die Waschanlage zu und müssen eventuell, wenn ein Fahrzeug gerade gewaschen wird oder weitere Fahrzeuge auf das Waschen warten, vor dieser warten. Nach dem Waschvorgang verlässt das Fahrzeug die Waschanlage auf der Rückseite des gezeigten Gebäudes.

Abb. 2.2 zeigt die Zerlegung des Waschvorgangs in fünf diskrete Einzelabschnitte.

Während die Vorgänge „Hinfahren zum Waschplatz“, „Einfahren in den Waschplatz“ und „Ausfahren aus dem Waschplatz  den Zustand der Waschanlage verändern, gilt das nicht für die beiden Vorgänge „Warten vor dem Waschplatz“ und „Waschen“. Man kann den Waschvorgang daher in drei Aktionen und in zwei Wartepositionen aufteilen. Ordnet man einer Aktion als Symbol ein kleines Rechteck zu (**Transition**), stellt eine Wartepositionen durch einen kleinen Kreis dar (Platz oder **Stelle**) und gibt die Bewegungsrichtung des Fahrzeugs durch verbindende Pfeile an (Kanten oder **Konnektoren**), so gelangt man zu dem auf der rechten Seite von Abb. 2.2 dargestellten einfachen Petri-Netz. Es ist noch unvollständig; zum Beispiel wird noch nicht zum Ausdruck gebracht, dass die zu waschenden Fahrzeuge statistisch verteilt eintreffen, dass sich nur ein Fahrzeug zu einem Zeitpunkt in der Waschanlage befinden kann und wie lange ein Waschvorgang dauert. Diese zusätzlichen Angaben  werden in PACE durch Attributierung der Netzelemente festgelegt (siehe Abschnitt 2.3).

Ein Petri-Netz wird schrittweise ausgeführt, indem ein zu bearbeitende Objekt, das durch einen kleinen gefüllten Kreis (Marke) dargestellt wird, von einem Eingangs-Lagerplatz (Eingangsstelle) geholt, in einer Transition bearbeitet und danach wieder auf einem Ausgangs-Lagerplatz (Ausgangsstelle) abgelegt wird. Stellen und Transitionen müssen deshalb alternierend auftreten.


Abb. 2.3: Autowaschplatz mit drei wartenden Fahrzeugen und einem Fahrzeug, das gerade gewaschen wird.

Abb. 2.3 zeigt das Petri-Netz aus Abb. 2.2 mit 3 wartenden Fahrzeugen vor dem Waschplatz und einem Fahrzeug, das gerade gewaschen wird. Die Stelle ‚Hinfahren zum Waschplatz’, die keinen Eingangskonnektor besitzt, erzeugt ständig Marken (Fahrzeuge), während die Stelle ‚Ausfahren aus dem Waschplatz’, die keinen Ausgangskonnektor besitzt, diese wieder aus dem Netz entfernt. Nehmen wir an, dass die Stelle ‚Warten vor dem Waschplatz’ beliebig viele Marken aufnehmen und in der Stelle ‚Waschen’ nur eine Marke gespeichert werden darf, so würde der nächste Schritt darin bestehen, dass die Transition ‚Ausfahren aus dem Waschplatz’ schaltet oder, wie man auch sagt "feuert", sobald die Zeit für den Waschvorgang abgelaufen ist. Dann kann die Transition ‚Einfahren in den Waschplatz’ feuern und das nächste Fahrzeug von der Stelle ‚Warten vor dem Waschplatz’ zur Stelle ‚Waschen’ transportieren, usw.

## 2.2 Einfache Netze

In diesem Abschnitt werden als Basis für die späteren Beschreibungen einige Eigenschaften klassischer Petri-Netze dargestellt.

**Markierung**
Als Markierung eines Petri-Netzes wird die Markenbelegung der Stellen eines Netzes bezeichnet. Sie ändert sich, ausgehend von einer Anfangsmarkierung während des Ablaufs eines Netzes durch Markenbewegung von Schritt zu Schritt. Durch die Markenbelegung wird der aktuelle Zustand eines Netzes definiert.

Die Anfangsmarkierung des in Abb. 2.5 dargestellten Netzes besteht in einer Anfangsmarke auf der Stelle ‚Waschplatz frei'.

**Schaltregel**
Die vorangegangene Beschreibung des Autowaschplatzes setzte voraus, dass die Stelle ‚Waschen' nur eine Marke, die Stelle ‚Warten vor dem Waschplatz' dagegen beliebig viele Marke aufnehmen darf. In klassischen Petri-Netzen wird diese Voraussetzung durch Einsatz der sog. Schaltregel realisiert. Eine weitere Möglichkeit besteht in der Festlegung der Kapazität einer Stelle, auf die später eingegangen wird.

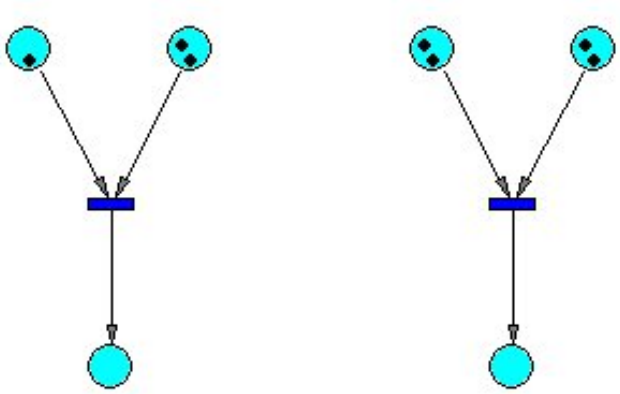


Abb. 2.4: Beispiele zur Schaltregel

Die Schaltregel behandelt den Fall, in dem eine Transition zwei oder mehr Eingangsstellen besitzt und besagt, dass in diesem Fall die Transition erst feuern darf, wenn in allen Eingangsstellen mindestens eine Marke vorliegt. Beim Feuern wird von allen Eingangsstellen jeweils eine Marke abgezogen und von der Transition verbraucht. Auf allen Ausgangsstellen wird danach jeweils eine Marke erzeugt. Eine Transition, in deren Eingangsstellen jeweils mindestens eine Marke liegt, wird als "aktiviert" bezeichnet.

Abb. 2.4 zeigt zwei Netze mit mehreren Eingangsstellen. Die Transition im linken Netz kann genau einmal feuern, weil eine der Eingangsstellen nur eine Marke trägt, die Transition im rechten Netz kann zweimal feuern, weil in beiden Eingangsstellen jeweils zwei Marken vorliegen.

Unter Verwendung der Schaltregel lässt sich nun erreichen, dass in der Stelle ‚Waschen' aus Abb. 2.3 nur jeweils ein Fahrzeug gespeichert wird. Dazu wird das Netz

aus Abb. 2.3, wie in Abb. 2.5 gezeigt, erweitert. Zu den Netzelementen aus Abb. 2.3 wurde eine weitere Stelle ‚Waschplatz frei' hinzugefügt. Der Waschplatz ‚Waschen' und die Warteschlange ‚Warten vor dem Waschplatz' werden zunächst als leer angenommen.

Da die Transition ‚Hinfahren zum Waschplatz' keinen Eingangskonnektor besitzt, ist ihr Feuern nicht von einer Bedingung abhängig. Sie kann deshalb zuerst feuern und legt damit eine Marke in die Stelle ‚Warten vor dem Waschplatz'. Die Transition ‚Einfahren in der Waschplatz' kann nun feuern, weil in ihren beiden Eingangsstellen jeweils eine Marke liegt. Durch das Feuern wird von jeder der beiden Eingangsstellen jeweils eine Marke abgezogen und eine Marke in die Stelle ‚Waschen' gelegt. Danach kann die Transition ‚Einfahren in den Waschplatz' nicht erneut feuern, weil nicht in beiden Eingangsstellen eine Marke vorliegt. In der Stelle ‚Waschen' kann zu einem Zeitpunkt also höchstens eine Marke vorliegen. Die Transition ‚Hinfahren zum Waschplatz' feuert erneut und legt erneut eine Marke in die Stelle ‚Warten vor dem Waschplatz'. Nach dem Waschen schaltet die Transition ‚Ausfahren aus dem Waschplatz' und legt dabei in die Stelle ‚Waschplatz frei' erneut eine Marke. Dadurch wird die Transition ‚Einfahren in den Waschplatz' wieder feuerbereit, usw.

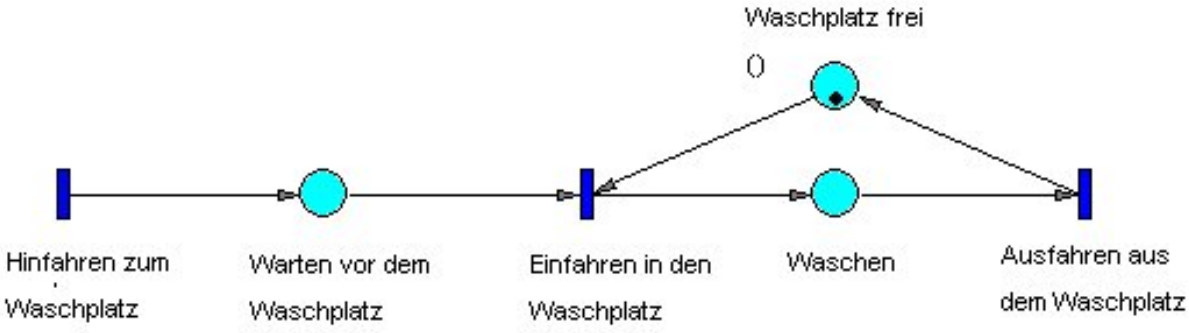



Abb. 2.5: Autowaschplatz mit Kontrolle des Waschplatzes.

**Doppelkonnektoren**

In den meisten Fällen werden zwei Netzelemente nur in einer Richtung, d.h. mit einem einfachen Konnektor verbunden. Solche Netzelemente werden als „reine" Netzelemente bezeichnet. Besteht ein Netz nur aus reinen Netzelementen, so wird es selbst als „rein" bezeichnet. Beispielsweise ist das in Abb. 2.5 dargestellte Netz rein.

Ein einfacher Fall, in dem zwei Netzelemente doppelt mit einem sog. Doppelkonnektor verbunden werden, liegt z.B. vor, wenn eine beliebige Anzahl von Marken erzeugt werden sollen. In dem Netz von Abb. 2.5 wurde das mit der Transition ‚Hinfahren zum Waschplatz' erreicht, die keinen Eingangskonnektor, d.h. keine Feuerbedingung besitzt und deshalb ständig feuert. Mit dem in Abb. 2.6 dargestellten Netz, in dem ein Doppelkonnektor verwendet wird, kann das gleiche erreicht werden.

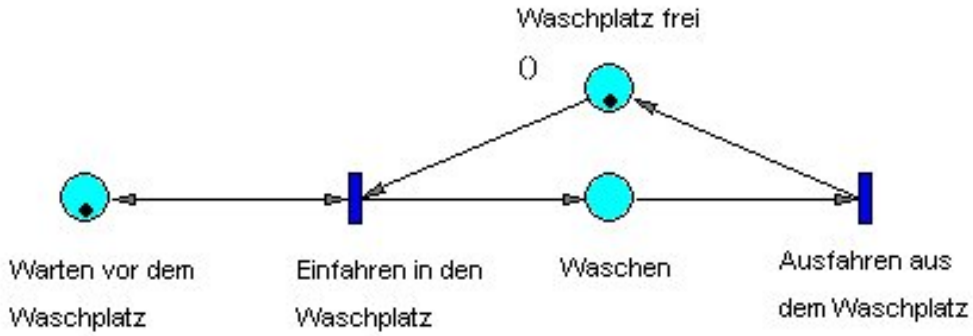



Abb. 2.6: Waschstraße unter Verwendung eines Doppelkonnektors

Beim Schalten der Transition ‚Einfahren in den Waschplatz', wird auch in die Stelle ‚Warten vor dem Waschplatz' eine Marke zurückgelegt, die das nächste zu waschende Fahrzeug darstellt.

**Reversibilität**
Ein Netz heißt reversibel, wenn die Anfangsmarkierung des Netzes ausgehend von einer beliebigen Folgemarkierung wiederhergestellt werden kann.

Sämtliche bisher betrachteten Beispielnetze sind reversibel. Die meisten der später betrachteten attributierten Petri-Netze sind aber irreversibel.

**Inhibitoren**
Neben den normalen Konnektoren gibt es invertierende Konnektoren (Inhibitoren), bei denen die Transition gerade dann schalten darf, wenn auf der Stelle, die über den Inhibitor mit der Transition verbunden ist, keine Marke liegt. Inhibitoren werden durch einen kleinen ausgefüllten Kreis auf der Pfeilspitze eines Konnektors dargestellt. Während über normale Konnektoren Marken fließen, unterbleibt dies bei einem Inhibitor. Er steuert nur das Feuern der Transition.

Auch mit einem Inhibitor lässt sich in einfacher Weise dafür sorgen, dass in der Stelle ‚Waschen' nur maximal eine Marke gespeichert wird. In Abb. 2.7 ist die in Abb. 2.5 dargestellte Waschstraße nun unter Verwendung eines Inhibitors modelliert.


Abb. 2.7: Die Waschstraße unter Verwendung eines Inhibitors

## *2.3 Attributierung von Netzen*

Um die bisher betrachteten Netze als realitätsnahe Modelle einer Waschstraße akzeptieren zu können, müssen sie mindestens noch folgende zwei weiteren Eigenschaften aufweisen:

1. Das statistisch verteilte Eintreffen der zu waschenden Fahrzeuge muss berücksichtigt sein.
2. Die Dauer eines Waschvorgangs muss im Modell angegeben werden.

Solche zusätzlichen Bedingungen werden in PACE durch Attributierung der Netze eingebracht.

Die zwei genannten Forderungen zusammen mit der Forderung, dass sich auf der Stelle **Waschen** maximal ein Fahrzeug befinden darf, bestimmen die Auslastung der betrachteten einfachen Waschanlage und lassen sich jeweils durch Anpassung einzelner Netzelemente erfüllen. So ist die 1. Forderung allein mit der Transition ‚Hinfahren zum Waschplatz' erfüllbar, denn diese erzeugt ja die Marken bzw. die zu waschenden Fahrzeuge. Die Forderung, dass sich auf der Stelle 'Waschen' nur eine Marke befinden darf, ist offenbar eine Eigenschaft dieser Stelle. Schließlich kann die 2. Forderung als Verzögerung des Ausfahrens aus dem Waschplatz aufgefasst werden und betrifft deshalb nur die Transition 'Ausfahren aus dem Waschplatz'.

Das Anheften von Attributen an Netzelemente erfolgt in PACE über sog. Netz-Inskriptionen oder kurz **Inskriptionen**. Das sind Programmtexte, die an die Netzelemente angeheftet und bei der späteren Ausführung des Netzes (Simulation) ausgewertet werden. Inskriptionen, die hier von Anfang an in allen Beispielen verwendet wurden, sind auch die Namen bzw. Bezeichner der Netzelemente.

Abb. 2.8 zeigt das Modell des Autowaschplatzes mit attributierten Netzelementen.


Abb. 2.8: Autowaschplatz mit Attributen

Das Eintreffen von Fahrzeugen vor der Waschanlage wurde durch eine Exponentialverteilung mit Mittelwert 10 min modelliert, welche die statistisch verteilten Zeitspannen zwischen der Ankunft zweier Fahrzeuge angibt. Die angegebene Botschaft wird bei jedem Feuern der Transition ‚Hinfahren zum Waschplatz' ausgeführt und liefert die Zeitverzögerung bis zum nächsten Feuern der Transition, d.h. bis zum Eintreffen des nächsten Fahrzeugs. Der Stelle ‚Waschen' wurde die Kapazität 1 zugeordnet, die in eckigen Klammern vor dem Bezeichner der Stelle angegeben ist. Das hat die gleiche Wirkung wie die im letzten Abschnitt beschriebenen Maßnahmen zur Beschränkung der maximalen Anzahl von Marken auf 1. Schließlich wird das Ausfahren aus dem Waschplatz um 6 min verzögert, d.h. ein Waschvorgang soll 6 min dauern.

Damit liegt ein einfaches Modell des Waschplatzes vor, in dem auch einige dynamischen Aspekte berücksichtigt sind. Das Modell kann deshalb dazu verwendet werden, um das Verhalten des realen Waschplatzes im Betrieb herauszufinden.

Interessant ist dabei die Stelle ‚Warten vor dem Waschplatz', in der die Warteschlange der Fahrzeuge vor der Waschstraße gespeichert ist. Um einen Eindruck von dem Verhalten des Waschplatzes bei dem vorausgesetzten Fahrzeugaufkommen zu erhalten, eignet sich besonders ein Zeit-Histogramm, das an die Stelle ‚Warten vor dem Waschplatz' angehängt wird.

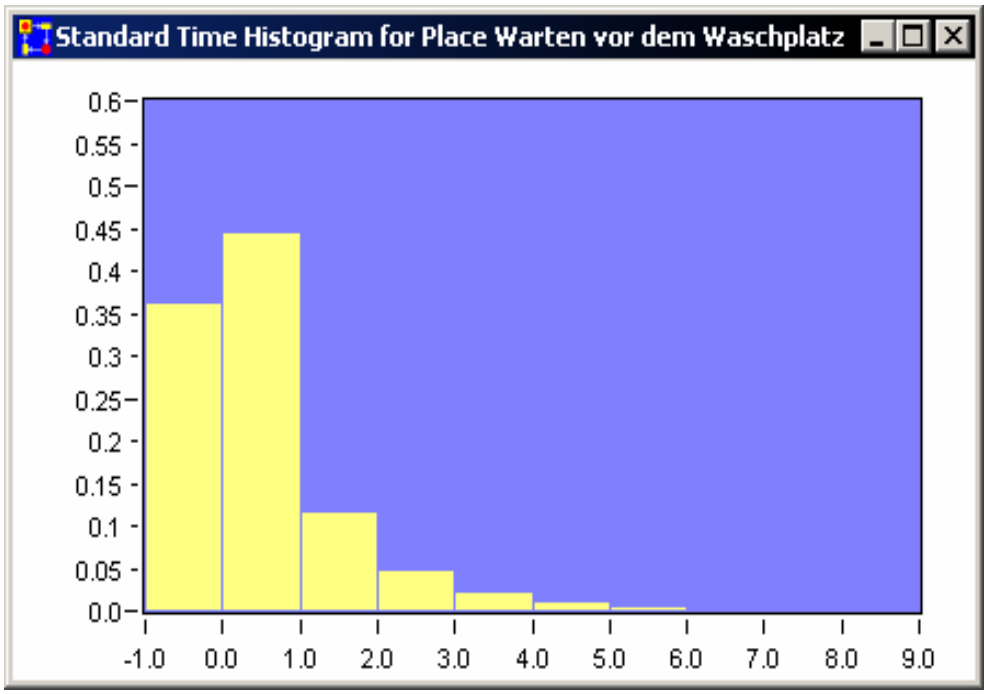



Abb. 2.9: Zeitanteile der verschiedenen Warteschlangenlängen

Führt man das Netz aus, so erhält man das in Abb. 2.9 dargestellte Ergebnis. Die einzelnen Balken stehen für die auf der Abszisse rechts neben dem Balken angegebene Anzahl von Fahrzeugen. Die Balkenordinate gibt zugeordnet den Prozentsatz der Betriebszeit an, in der die Fahrzeugwarteschlange diese Anzahl von Fahrzeugen enthält. Man erkennt aus der Abbildung, dass nur selten mehr als zwei Fahrzeuge in der Schlange stehen und dass die Warteschlange etwa 36% der Betriebszeit leer ist. Auch weitergehende Forderungen an das Modell, etwa das unterschiedliche Fahrzeugaufkommen zu verschiedenen Tageszeiten und Wetterbedingungen, verschiedene Waschprogramme mit jeweils unterschiedlichen Waschzeiten, das Verhalten mancher Autofahrer, welche an der Anlage vorbeifahren, wenn mehr als ein Fahrzeug wartet oder betriebswirtschaftliche Fragestellungen lassen sich in ähnlich einfacher Weise in das Modell integrieren.

Schon aus den gezeigten einfachen Netzen lässt sich schließen, dass die Zuordnung zwischen Netz und Realität oft schwierig ist. Vielen Netzen sieht man nicht ohne weiteres an, was sie beschreiben. Diese Schwierigkeit lässt sich durch geeignete Namen für die Netzelemente und durch Ersatzsymbole (Bilder) für einzelne Netzelemente sehr stark reduzieren. Abb. 2.10 zeigt eine Momentaufnahme des Netzfensters während der Simulation, in dem die Netzelemente durch Bilder (Ikonen) ersetzt wurden. Als Hintergrundbild dient das mattierte Bild der gesamten Waschanlage.


Abb. 2.10: Ein Bild sagt mehr als 1000 Worte.

Mit den drei elementaren statischen Netz-Elementen Transition, Stelle und Konnektor lässt sich eine Anwendung auf einzelne Prozessschritte abbilden. Werden z.B. Daten verarbeitet, so fließen diese von einer Eingangsstelle, die einen Speicherplatz modelliert, zur Transition, in der die eigentliche Bearbeitung stattfindet und werden nach der Verarbeitung in einer Ausgangsstelle gespeichert, die wiederum einen Speicherplatz repräsentiert. In ähnlicher Weise kann auch die Bearbeitung von Werkstücken durch Maschinen, der Produktionsablauf auf Förderbändern oder die Bearbeitung von administrativen Vorgängen in Behörden auf attributierte Netze abgebildet werden. Man kann deshalb ein attributiertes Netz als ein ablauffähiges virtuelles Abbild des jeweils abgebildeten realen Systems ansehen und für die Analyse und Weiterentwicklung des realen Systems verwenden.

# 3. Modellierung einer Auto-Waschanlage

## 3.1 Das Grundmodell

Die Bedienung von *PACE* erfolgt mit einer 3-Tasten-Maus. Bei den meisten heute gebräuchlichen PC-Mäusen ist die mittlere Maustaste als Drehrad ausgebildet und kann auch zum Klicken verwendet werden. Zur Vereinfachung der nachfolgenden Beschreibung werden die einzelnen Maustasten wie folgt bezeichnet:

| | | |
|---|---|---|
| **li.MT** | linke Maustaste | Selektieren oder Markieren eines Objekts |
| **mi.MT** | mittlere Maustaste | System-Menü anzeigen |
| **re.MT** | rechte Maustaste | Kontext-spezifisches Menü anzeigen |

Wenn Sie das Touchpad eines Notebooks mit zwei Tasten für die Cursor-Steuerung verwenden, so ist normalerweise die linke Taste der **li.MT** und die rechte Taste der **mi.MT** zugeordnet. Die rechte Taste zusammen mit der Strg-Taste der Tastatur simuliert die **re.MT**.

Starten Sie PACE nach der Installation über die Windows-Programmleiste oder indem Sie den Windows-Explorer aufrufen und in das PACE-Installationsverzeichnis gehen. Dort PACE durch Doppelklick mit **li.MT** auf das PACE-Image **pace2008.imm** starten. Nach dem Start von PACE erscheint auf dem Bildschirm die PACE-Menüleiste:


Abb. 3.1: PACE-Menüleiste


Abb. 3.2: File-Menü

Sie ist der Ausgangspunkt für die Modellerstellung mit PACE. Die Menüleiste ist während aller Arbeitsphasen verfügbar. Mit ihr können u.a. alle systemweiten Einstellungen vorgenommen werden.

Um ein neues Modell zu erstellen, muss die Funktion **new net** im File-Menü ausgeführt werden. Dabei öffnet sich ein Abfragefenster, in das der Name des neuen Netzes einzugeben ist. 'newNet' ist der Default-Name. Nach der Eingabe des Modellnamens **AutoWaschplatz** wird die Return-Taste gedrückt. Es öffnet sich der Rahmen für ein Fenster, die sog. Netzliste, in dem die Bausteine des Netzes später angezeigt werden. Schieben Sie durch Bewegen der Maus den Rahmen an die gewünschte Stelle auf dem Bildschirm, drücken die **li.MT** und halten Sie fest. Der Cursor springt von der linken oberen Ecke auf die rechte untere Ecke. Sie können nun durch Bewegung der Maus das Fenster zur Darstellung der Netzliste auf die

gewünschte Größe aufziehen. Dann die Maustaste loslassen. Das Ergebnis sehen Sie in Abb. 3.3. Im vorliegenden Fall enthält die Liste nur eine Zeile mit dem Namen **AutoWaschplatz**.

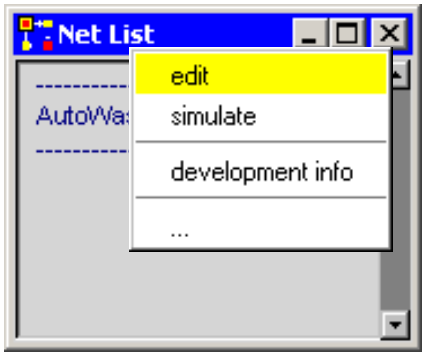



Abb. 3.3: Net-List und Net- List-Menu

Das File-Menü enthält die allgemeinen dateibezogenen Funktionen, wie Laden eines Netzes (load net), Sichern des aktuellen Netzes (store net), usw.

Die Zeile in der Netzliste mit **li.MT** anklicken (selektieren), mit der **re.MT** das Menü der Netzliste öffnen und **edit** auswählen.

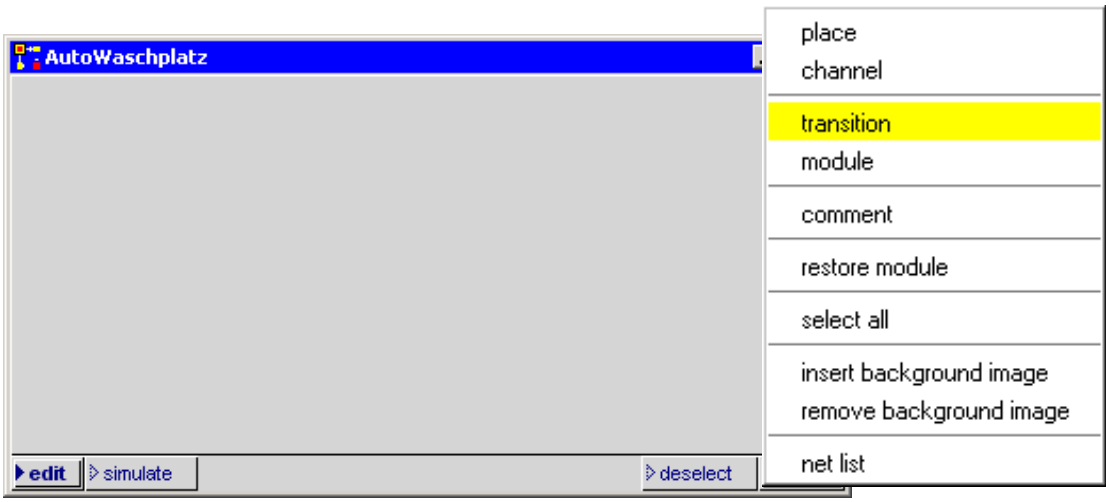



Abb. 3.4: Netzfenster und No-Selection-Menü


Abb. 3.5: Netzfenster nach Fixieren der Transition

Nun öffnet sich das Netz-Fenster von **AutoWaschplatz** im Editier-Modus (Abb. 3.4). Die Größe dieses Fensters kann in der bei Windows üblichen Weise verändert werden. In diesem Fenster kann nun das Netz für den Autowaschplatz modelliert werden. Die Modellierung erfolgt durch Einsetzen der für das zu erstellende Netz benötigten

Elemente. Diese werden mit dem graphischen *PACE*-Editor interaktiv eingesetzt. Dafür stehen Menüs zur Verfügung, aus denen die Netz-Elemente selektiert werden.

Im Netzfenster, d.h. mit dem Mauszeiger (Cursor) im Netzfenster, wird durch Drücken der **re.MT** das sog. **No-Selection-Menü** angezeigt. Sein Name besagt, dass es aufgerufen wird, wenn kein Netzelement im Fenster selektiert worden ist (im Augenblick ist ja noch keins vorhanden). Nach der Auswahl des Menüpunkts **transition** erscheint das Symbol für eine Transition an der Cursorposition. Es kann durch Verschieben der Maus an die vorgesehene Stelle gebracht werden. Dort wird es durch Drücken der **li.MT** fixiert (Abb.3.5).

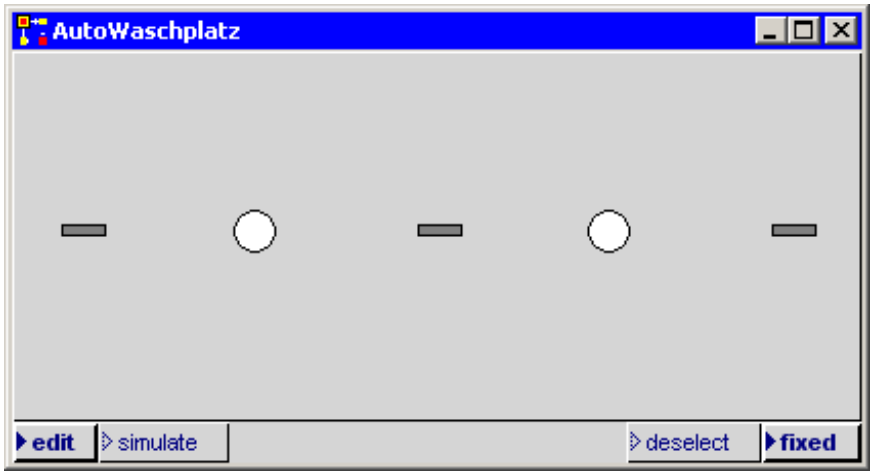



Abb. 3.6: Netzfenster ohne Konnektoren

In der gleichen Weise werden auch die übrigen Netzelemente in das Fenster eingebracht, wobei die Stellen durch Auswahl des Menüpunkt **place** ausgewählt werden. Falls durch Versehen ein anderes als das gewünschte Netzelement ausgewählt wird, so kann es vor dem Fixieren durch Drücken der **re.MT** wieder gelöscht werden. Nach dem Fixieren kann ein Netzelement gelöscht werden, indem es zunächst mit der **li.MT** selektiert wird und dann mit **re.MT** das Menü des Netzelements angezeigt wird. In diesem ist dann der Menüpunkt **delete** auszuwählen. Auf die Menüs der Netzelemente wird später noch genauer eingegangen. Das Ergebnis ist in Abb. 3.6 dargestellt.

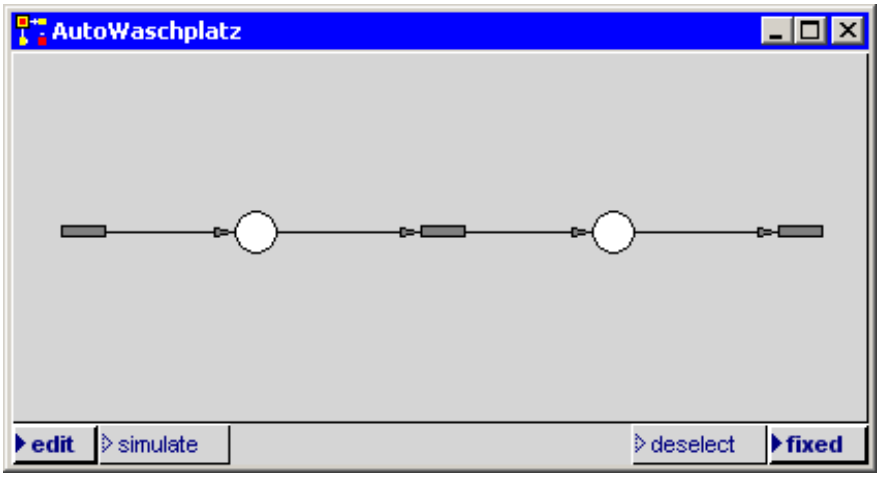



Abb. 3.7: Netzfenster mit Konnektoren

Falls Sie mit der Position des Netzelements nicht zufrieden sind und es an eine andere Stelle im Netzfenster verschoben werden soll, so wird es mit der **li.MT** markiert. Dann erneut mit der **li.MT** auf das Netzelement klicken und die Maustaste festhalten. Das Element mit festgehaltener Maustaste an die neue Position schieben und dort die Maustaste loslassen.

Als nächstes werden die Konnektoren eingesetzt. Dazu wird der Mauszeiger auf eines der Netzelemente positioniert, die **li.MT** gedrückt und festgehalten. Mit festgehaltener Maustaste wird der Mauszeiger auf das Netzelement geschoben, das verbunden werden soll. Dort wird die Maustaste losgelassen. Wie früher schon erwähnt müssen Stellen und Transitionen im Netz alternierend auftreten. Ein Konnektor wird nur eingerichtet und angezeigt, wenn diese Bedingung erfüllt ist. Das Ergebnis zeigt Abb. 3.7.

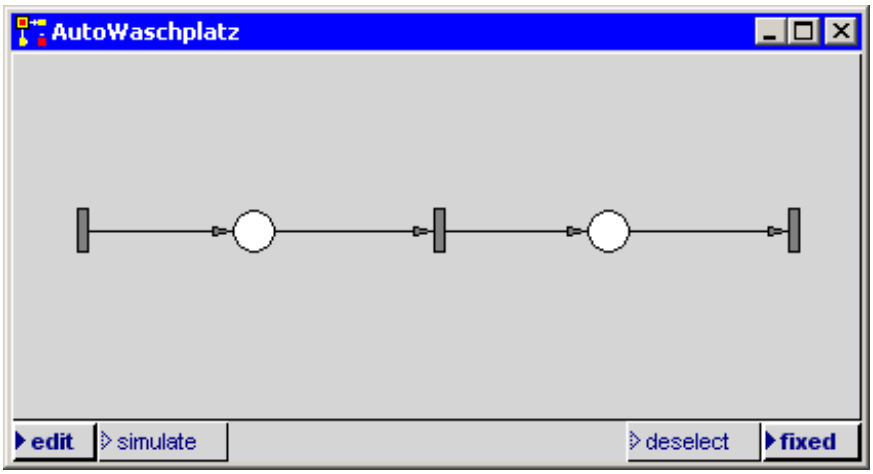



Abb. 3.8: Netzfenster mit alternativer Transitions-Darstellung

Wenn man will, kann man das Netz dadurch verschönern, dass man die Konnektoren nicht mit den schmalen Seiten, sondern mit den breiten Seiten der Transitionen verbindet. Dazu müssen die Standard-Ikonen der Transitionen um 90 Grad gedreht werden. Das geht wie folgt:

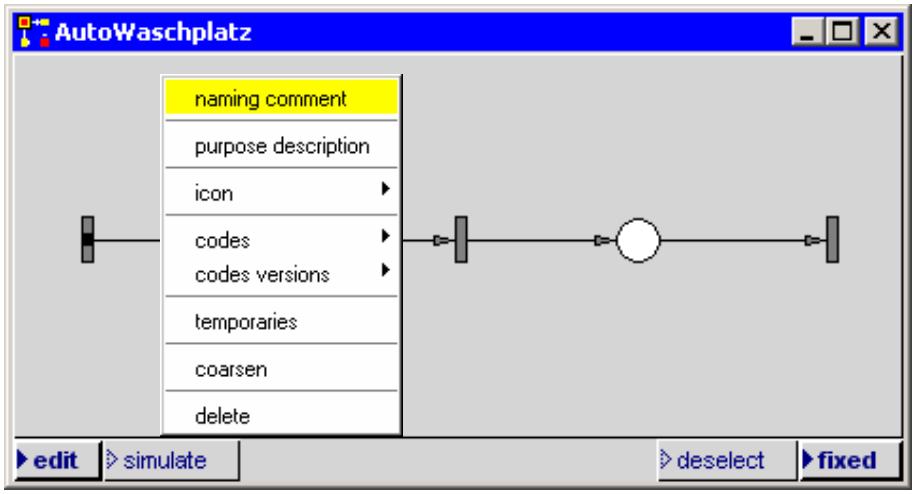



Abb. 3.9: Attributieren von Netzelementen

Selektiert wird eine der Transitionen mit der **li.MT**. Dann wird mit der **re.MT** das Menü der Transition angezeigt, das Submenü des Menüpunkt **icon** angezeigt und darin der Menüpunkt **alternate** ausgewählt.
Wird das für alle Transitionen durchgeführt, so sieht das Netzfenster jetzt wie in Abb. 3.8 dargestellt aus.

Als nächstes ist das Netz zu attributieren.

Die Bezeichner für die einzelnen Netzelemente werden wie folgt eingesetzt. Zunächst das ausgewählte Netzelement mit der **li.MT** markieren (auf dem Netzelement wird ein schwarzer Punkt angezeigt) und mit der **re.MT** das Menü des Netzelements anzeigen. Dann den Menüpunkt **naming comment** auswählen (Abb. 3.9). Es öffnet sich das in Abb. 3.10 gezeigte Eingabefenster für den Bezeichner des Netzelements.

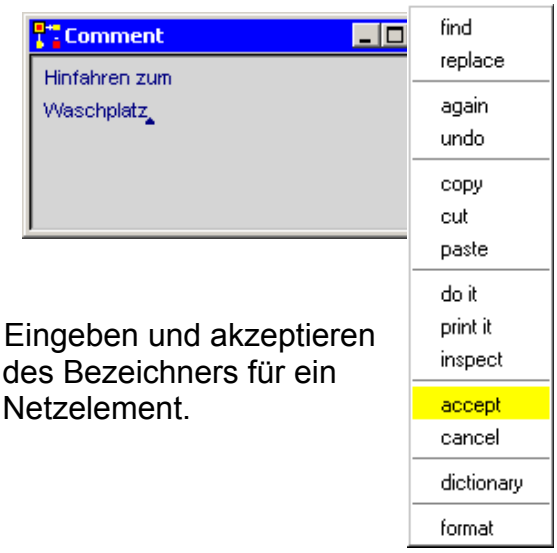



Abb. 3.10: Eingeben und akzeptieren des Bezeichners für ein Netzelement.

Nach der Eingabe des Bezeichners, der wie ein Kommentar über mehrere Zeilen gehen darf, mit der **re.MT** das Menü des Textfensters aufrufen und darin **accept** auswählen. Dadurch wird der Bezeichner dem selektierten Netzelement zugeordnet und im Netzfenster dargestellt (Abb. 3.10 und Abb. 3.11).

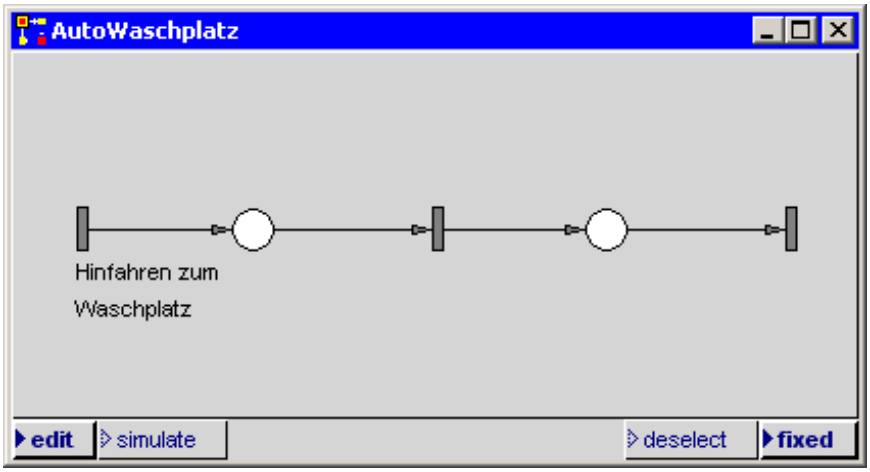



Abb. 3.11: Netzfenster mit Bezeichner für ein Netzelement

Falls der Bezeichner (der "naming comment") an eine andere Stelle im Netzfenster verschoben werden soll, so wird er mit der **li.MT** markiert. Dann erneut mit der **li.MT** auf den Bezeichner klicken und die Maustaste festhalten. Den Bezeichner mit festgehaltener Maustaste an die neue Position schieben und dort die Maustaste loslassen. Auf diese Weise alle Bezeichner aus Abb. 2.3 eintragen und positionieren. Dabei ggf. das Fenster etwas vergrößern, damit die Texte zugeordnet dargestellt werden können (Abb. 3.13).

Falls Sie später merken, dass Sie versehentlich einen Schreibfehler in einem der Bezeichner gemacht haben, können Sie das wie folgt korrigieren: Markieren Sie den Bezeichner, drücken Sie die **re.MT** und wählen Sie den Menüpunkt **inspect**. Dann öffnet sich wieder das Texteingabefenster mit dem fehlerhaften Text. Nach der Korrektur wieder wie früher den Menüpunkt **accept** ausführen. Im Übrigen: falls Sie statt **inspect** den Menüpunkt **owner** auswählen, wird der Mauszeiger auf dem Netzelement positioniert, zu dem der Bezeichner gehört. Diese Funktion ist bei größeren Netzen mit vielen Inskriptionen oft sehr nützlich!

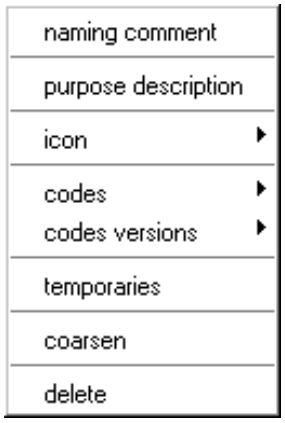



Abb. 3.12: Menü einer Transition

Als nächstes werden die in Abb. 2.8 gezeigten Attribute eingetragen. Dazu wird die Transition 'Hinfahren zum Waschplatz' mit der **li.MT** markiert und das Menü der Transition mit der **re.MT** angezeigt (Abb. 3.12).

Das Menu enthält einen Menüpunkt **codes** mit drei Untermenüpunkten, mit denen eine Transition an die jeweilige Aufgabe angepasst wird.

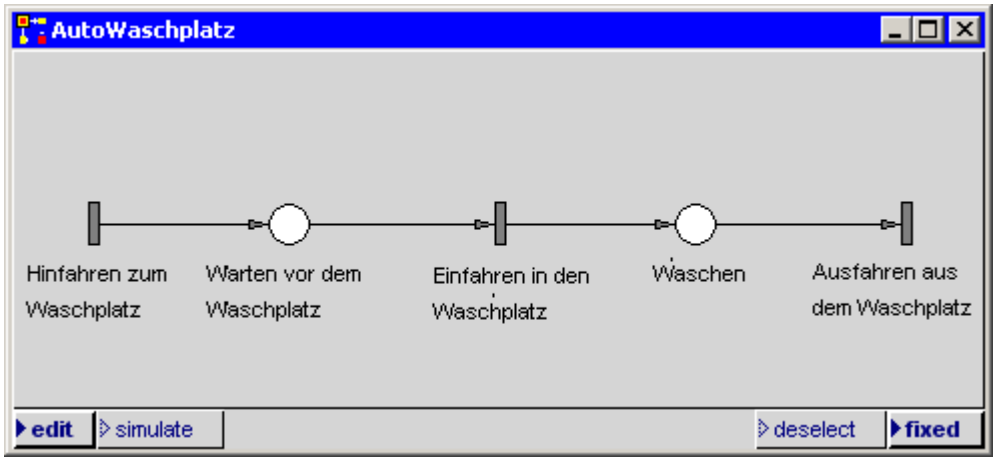



Abb. 3.13: Netzfenster mit Bezeichnern für alle Netzelemente

Ein Transition kann mit den folgenden drei Codeeinschüben angepasst werden:

**condition code:** (grün) Hier kann eine Bedingung formuliert werden, unter der die Transition feuern darf. Der angegebene Codeeinschub muss einen der Wahrheitswerte **true** oder **false** liefern.

**delay code:** (rot) Der Codeeinschub berechnet die Anzahl von Zeiteinheiten, um die das Feuern der Transition verzögert werden soll.

**action code:** (blau) Zur Darstellung eines realen Objekts in einem PACE-Modell, werden die das Objekt repräsentierenden Daten an das virtuelle Objekt (die Marke) angehängt. Der Aktionscode bildet die Bearbeitung des realen Objekts ab, indem er aus den Daten der einlaufenden Marken die Daten der auslaufenden Marken berechnet.

Im vorliegenden Fall ist nur der Verzögerungscode anzugeben. Nach Auswahl des Untermenüpunkts **delay code** öffnet sich ein Eingabefenster für die Eingabe des Programmtexts (Abb 3.14).

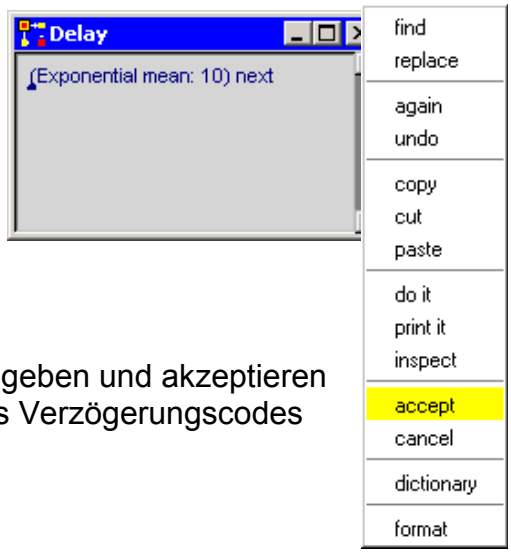



Abb. 3.14: Eingeben und akzeptieren des Verzögerungscodes

Der einzugebende Text ist eine Smalltalk-Botschaft. Der in Klammern stehende Ausdruck **(Exponential mean: 10)** definiert eine Exponentialverteilung mit Mittelwert 10 Zeiteinheiten. Durch den unären Operator **next** wird der nächste Wert der Verteilung berechnet. Durch Anwahl des Menüpunkt **accept** wird der Text wie früher bei den 'naming comments' an das Netzelement abgehängt. Falls der Text im Netzfenster an einer anderen Stelle positioniert werden soll, so wird das genau so wie bei einem 'naming comment' durchgeführt (siehe weiter oben). Entsprechend wird auch mit der Transition 'Ausfahren aus dem Waschplatz' verfahren, wobei hier nur die Zahl 6 einzugeben ist.

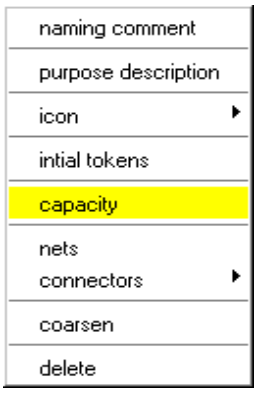



Abb. 3.15: Menu einer Stelle

Die Zuordnung der Simulator-Zeiteinheiten zu physikalischen Zeiteinheiten wird dadurch hergestellt, dass alle Zeitangaben im Modell bezüglich der gleichen Dimension, im vorliegenden Fall in Minuten gemacht werden. Dadurch wird implizite die physikalische Dimension der Simulator-Zeiteinheit festgelegt.

Um die Maximalzahl der Marken, die in der Stelle 'Waschen' gespeichert werden dürfen, auf 1 zu setzen, wird die Stelle mit der **li.MT** markiert und mit der **re.MT** das Menü der Stelle angezeigt. Wird der Menüpunkt capacity ausgewählt, so öffnet sich ein Eingabefenster zur Eingabe der zulässigen Maximalzahl von Marken.

Als Defaultwert wird nach dem Öffnen des Fensters der Wert 0 angezeigt. Er besagt, dass beliebig viele Marken gespeichert werden dürfen. Der Wert 0 wird durch den Wert 1 ersetzt und danach die **return**-Taste gedrückt. Vor dem Bezeichner 'Waschen' ist jetzt im Netzfenster in eckigen Klammern die maximale Anzahl von Marken der Stelle angegeben.

Das Netz für den Auto-Waschplatz ist damit bis auf die Ergebnisausgabe und eventuelle Verschönerungen durch Einsetzen von Ikonen fertiggestellt (siehe Abb. 3.16).

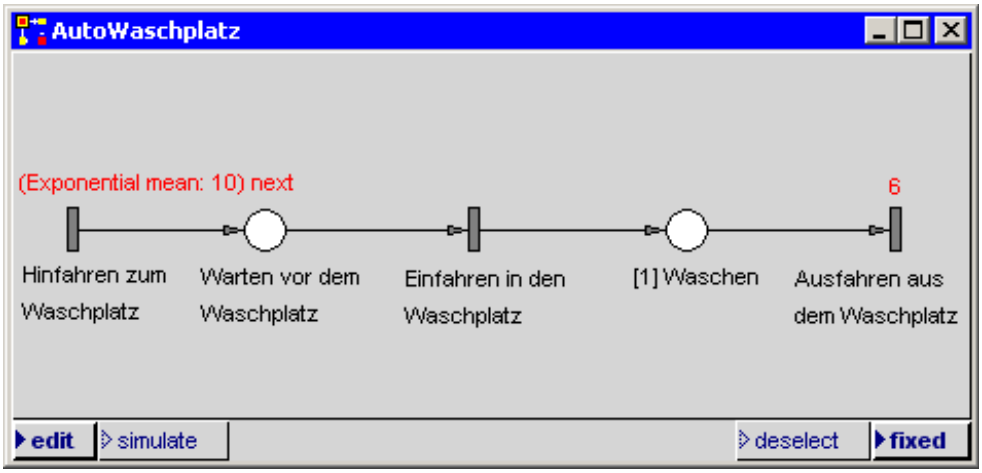



Abb. 3.16: Netzfenster des Auto-Waschplatz mit Inskriptionen

Um das Netz probehalber auszuführen, muss vom Editiermodus in den Simulationsmodus gewechselt werden. Das geschieht, indem man im Netzfenster links unten mit der **li.MT** den Knopf **simulate** drückt. Dann im Netzfenster mit der **re.MT** das in Abb. 3.17 dargestellte No-Selection-Menü des Simulationsmodus aufrufen.

Um zu testen, ob das Netz einwandfrei funktioniert, wird der Menüpunkt **initialize + run** ausgewählt. Sie können nun beobachten, wie von links Marken (Fahrzeuge) einlaufen, eventuell in der Stelle 'Warten vor dem Waschplatz' warten müssen und nach Durchlaufen des Waschplatzes 'Waschen' das Modell wieder verlassen. Die Simulation wird angehalten, wenn sie den Mauszeiger in das Netzfenster schieben und die **li.MT** drücken.

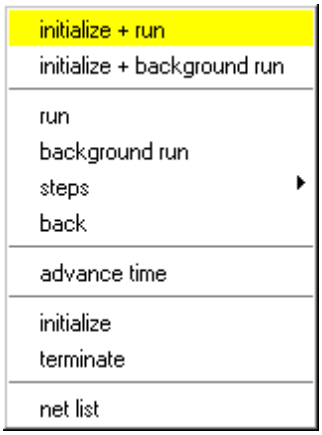



Abb. 3.17: Menü des Netzfensters im Simulationsmodus

Einen Überblick über die Eigenschaften der Warteschlange kann man sich mit einem Zeit-Histogramm verschaffen, das angibt, wie sich die Zeit prozentual auf die verschiedenen Warteschlangelängen verteilt.

Dazu wird das Menü der Stelle 'Warten vor dem Waschplatz' im Simulationsmodus angezeigt und der Menüpunkt **diagram** ausgewählt. Im Submenü des Menüpunkt **diagram** wird der Menüpunkt **time histograms** ausgewählt, der wieder ein Submenü besitzt. In diesem wird schließlich der Menüpunkt **standard** ausgewählt. Es öffnet sich der Rahmen für ein Grafikfenster an der Cusor-Position, das wie früher beschrieben positioniert und geöffnet wird (Abb. 3.18).

Bevor man das Diagram verwenden kann, muss es noch skaliert werden. Die Ordinate wird skaliert, indem man den Mauszeiger auf die linke Umrahmung positioniert und dort die **re.MT** drückt. Es wird ein Menü aus drei Zeilen angezeigt, in dem der Menüpunkt **maximum value** ausgewählt wird. In das sich öffnende Fenster wird der Wert 0.6 eingegeben und dann die **return**-Taste gedrückt. Entsprechend wird mit dem Menüpunkt **inscriptions** die Anzahl der Skalenwerte festgelegt. Einzugeben ist hier die Zahl 12.

Analog wird die Abszisse skaliert: Mauszeiger im unteren Fensterrand positionieren und die rechte Maustaste drücken. Hier ist die Skalierung, der Maximalwert und die Anzahl der Balken festzulegen. Die Eingabe erfolgt wie eben bei der Skalierung der Ordinate geschildert. Gewählt werden die Angaben aus Abb. 2.9:

Minimalwert = -1
Maximalwert = 9
Anzahl der Balken = 10
Anzahl der Inskriptionen = 10

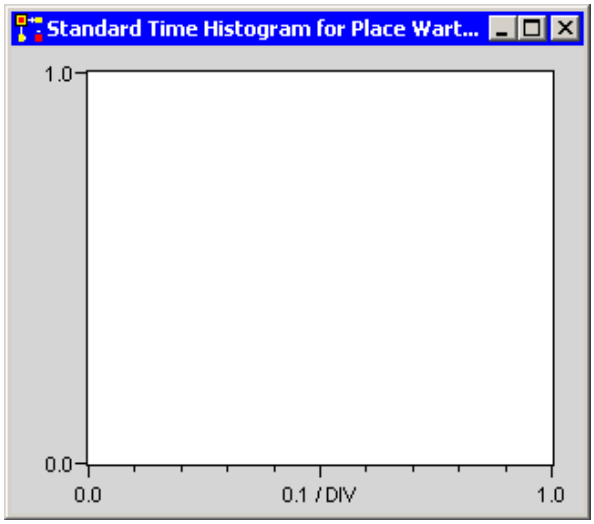



Abb. 3.18: Histogramm-Fenster vor der Skalierung

Das Ergebnis zeigt Abb. 3.19. Zu beachten ist, dass die jeweilige Balkenzahl durch die Skalenzahl am rechten Balkenrand angegeben wird.

Wenn man will, kann man auch noch die Farben wie in Abb. 2.9 einführen. Dazu den Mauszeiger in den mittleren Teil des Fensters bringen, die **re.MT** drücken und **colors** wählen.. Es öffnet sich ein Fenster, in dem die Vordergrund- (Farbe der Balken) und die Hintergrundfarbe (Farbe des Fensters) ausgewählt werden kann.

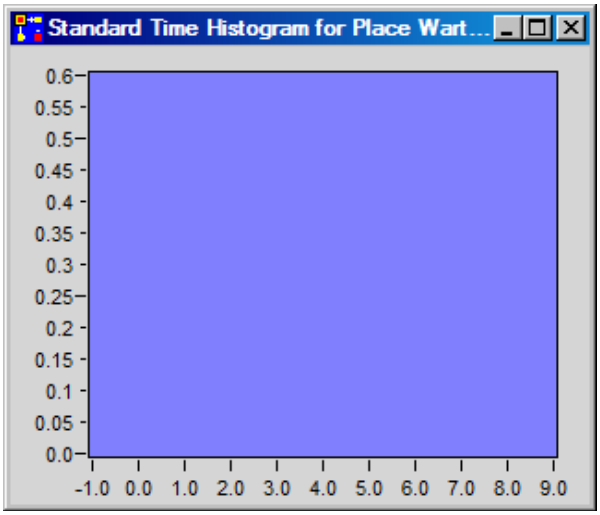



Abb. 3.19: Histogramm-Fenster nach der Skalierung

Das Histogramm für den Waschplatz wird erzeugt, indem das Menü des Netzfensters im Simulationsmode (Abb. 3.17) aufgerufen wird. Darin diesmal den Menüpunkt **initialize + background run**

wählen, damit das Histogramm (siehe Abb. 2.9) schnell erzeugt wird. Die Simulation wird angehalten, wenn sie den Mauszeiger in das Netzfenster schieben und die **li.MT** drücken.

Damit ist das Grundmodell eines Auto-Waschplatzes bis auf die Einsetzung von Bildern für die Netzelemente fertig gestellt. Das wird später nachgeholt.

Das Modell kann mit dem Menüpunkt **store image** im File-Menü der PACE-Leiste gesichert werden. Der Menüpunkt öffnet ein Windowsfenster zum Abspeichern einer Datei im Installationsverzeichnis. Es ist zweckmäßig, für das Modell einen neuen Namen, z.B. AutoWaschplatz zu verwenden. Wird das Modell danach mit dem Menüpunkt **leave PACE** beendet, so kann es später im Explorer durch Doppelklick auf den Namen AutoWaschplatz.imm wieder geladen werden und erscheint auf dem Bildschirm genau so, wie es vor dem Abspeichern vorlag.

Im nächsten Abschnitt werden einige wichtige Eigenschaften von PACE anhand einer Erweiterung des Modells vorgestellt.

## 3.2 Mehrere Waschprogramme

Bei der Beschreibung des Action-Codes wurde erwähnt, dass die realen Objekte auf Marken abgebildet werden, welche deren charakteristischen Daten tragen. Außer den Objektdaten können auch weitere für die Verarbeitung wichtige Daten an eine Marke angehängt und durch das Netz transportiert werden. Eine solche Date kann bei der Fahrzeugfertigung beispielsweise die Liste der Extras sein, die in das aktuelle Fahrzeug eingebaut werden sollen.

Im vorliegenden Fall sei angenommen, dass die Waschanlage zwei unterschiedliche Waschprogramme mit den unterschiedlichen Waschzeiten 6 und 9 Minuten bietet. Außerdem soll der Prozentsatz der Fahrzeuge, welche das kürzere Waschprogramm verwenden, mit einem Balkenschieber vor einem Simulationslauf einstellbar sein.

Der Zugriff auf die durch das Netz laufenden, an Marken angehängten Daten wird mit Hilfe von sog. **Konnektorvariablen** durchgeführt. Nehmen wir an, dass das Waschprogramm beim Einfahren in den Waschplatz ausgewählt wird, so muss durch die Marke die Information zur Transition 'Ausfahren aus dem Waschplatz' transportiert werden, damit das Waschprogramm und damit die Waschzeit richtig eingestellt wird.

Ausgegangen wird von dem in Abb. 3.16 dargestellten Netz. Falls sich das Netzfenster wegen der zwischenzeitlichen Ausführung von Simulationsläufen im Simulationsmodus befindet, den Knopfs **edit** im linken unteren Eck mit **li.MT** drücken. Der Mauszeiger wird auf den Konnektor zwischen den Elementen 'Einfahren in den Waschplatz' und 'Waschen' gesetzt und der Konnektor durch Drücken der **li.MT** markiert. Die Markierung wird durch einen schwarzen Punkt auf dem Konnektor angezeigt.

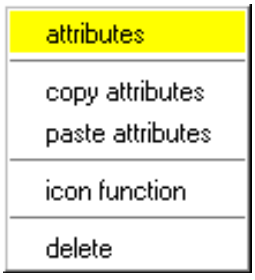



Abb. 3.20: Menü eines Ausgangs-Konnektors

Mit der **re.MT** wird nun das in Abb. 3.20 dargestellte Menü des Ausgangs-Konnektors einer Transition angezeigt. Wählt man den Menüpunkt **attributes**, so öffnet sich ein Fenster mit zwei Teilfenstern (Abb. 3.21).

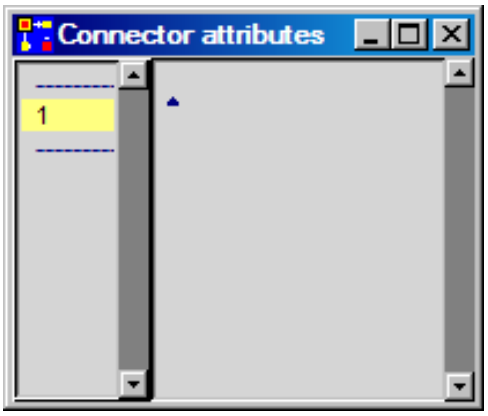



Abb. 3.21: Attributierung von Konnektoren

Das linke Teilfenster gibt die Anzahl der Attribute oder Parameter an, welche alle Marken aufzeigen müssen, die über den Konnektor laufen dürfen. In Abb. 3.21 ist ein Attribut vorgesehen, was für das vorliegende Beispiel ausreicht.

In dem rechten Fenster wird der Name der Konnektorvariablen angegeben. Dazu den Mauszeiger im rechten Fenster positionieren, die **li.MT** drücken und den Text **program** eingeben. Bitte beachten Sie, dass Konnektor-Variablen mit einem kleinen Buchstaben beginnen müssen.

Danach, wie früher, das Fenstermenü durch Drücken der **re.MT** im rechten Teilfenster aufrufen und den Menüpunkt **accept** auswählen. Die Konnektorvariable wird jetzt in Klammern eingeschlossen im Netzfenster zusammen mit dem Konnektor angezeigt. Sie kann, wie früher schon beschrieben, an eine andere Stelle geschoben werden.

Damit das ausgewählte Waschprogramm die Transition 'Ausfahren aus dem Waschplatz' erreichen kann, muss auch der Konnektor von der Stelle 'Waschen' nach der Transition 'Ausfahren aus dem Waschplatz' attributiert werden. Andernfalls könnte die Marke, welche das Waschprogramm als Attribut trägt, nicht weiterlaufen, weil der Konnektor bisher nur Marken ohne Attribut akzeptiert.

Man könnte hier wie oben verfahren, indem man ein Fenster für die Attribute des Konnektor aufmacht, usw. Das wäre aber sehr mühselig, wenn zahlreiche Konnektoren mit den gleichen Attributen auszustatten sind. Bequemer ist es, die Kopierfunktion für Attribute zu verwenden.

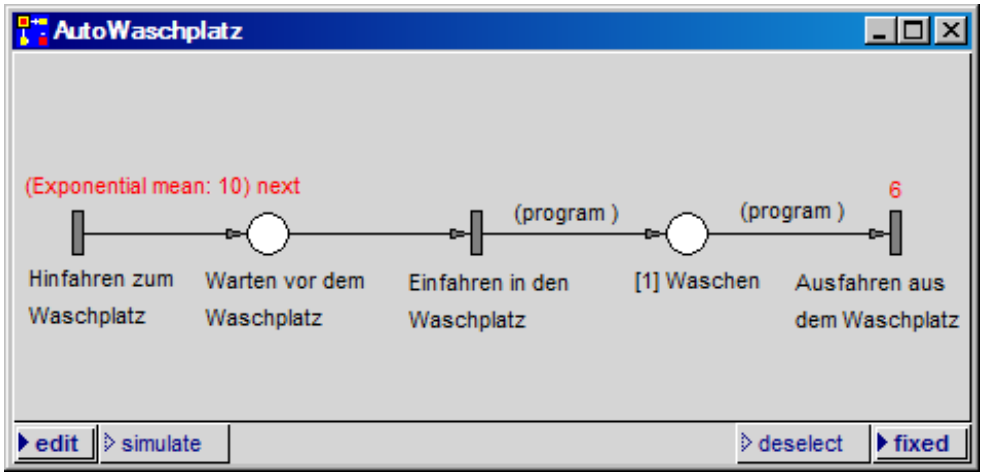



Abb. 3.22: Netzfenster mit Konnektorvariablen

Dazu den schon attributierten Konnektor markieren und mit der **re.MT** sein Menü aufrufen. Darin den Menüpunkt **copy attributes** auswählen. Nun den zu attributierenden Konnektor markieren, mit der **re.MT** sein Menü aufrufen und den Menüpunkt **paste attributes** aufrufen. Der gegenwärtige Stand des Netzfenster ist in Abb. 3.22 dargestellt.

Für die Eingabe des Prozentsatzes der Fahrzeuge, die Waschprogramm 1 mit 6 Minuten Waschzeit wählen, wird ein vertikaler Balkenschieberegler (Bar Gauge) verwendet. Um ihn darzustellen, in der PACE-Menüleiste das Views-Menü aufrufen, darin das Submenü **linear gauges** und darin den Menüpunkt **bar gauge** auswählen. Es öffnet sich wieder ein Rahmen für ein Grafikfenster, das wie oben schon beschrieben aufzuziehen ist.

Für das Fenster wird, in früher beschriebener Weise, der maximale Skalenwert 100 eingestellt. Außerdem wird die Skalierung verfeinert, indem die Anzahl der Inskriptionen auf 10 gesetzt wird.

Um den Fensternamen festzulegen, wird der Mauszeiger im Fenster positioniert und die **mi.MT** gedrückt. Dadurch wird das System-Menü des Fensters angezeigt. Nach Auswahl des Menüpunkt **relabel as…**, öffnet sich ein Eingabefenster, in das der Name **Prozentsatz** eingegeben wird. Danach mit der **li.MT** den ok-Knopf drücken.

Der Balkenschieberegler sollte jetzt wie in Abb. 3.23 aussehen. Um einen neuen Wert einzustellen, kann der Balken mit der **li.MT** verschoben werden.

Damit im Modell auf den Balkenschieberegler zugegriffen werden kann, muss er an das Modell angeschlossen werden. Das wird am zweckmäßigsten zu Beginn eines Simulationslaufs bei Initialisieren des Modells durchgeführt.

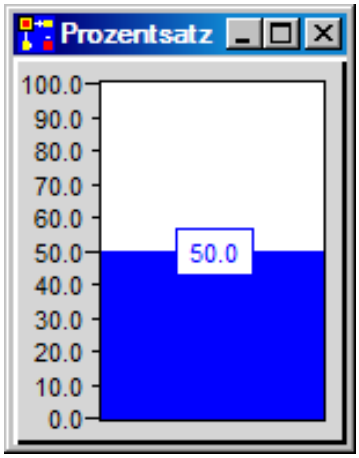



Abb. 3.23: Balkenschieberegler

Dazu wird das Menü **Net Editor** in der PACE-Hauptleiste angezeigt, in diesem das Submenü **extra codes** aufgerufen und darin **initialization code** ausgewählt. Es öffnet sich ein Textfenster zur Eingabe des Programmcodes, der zu Beginn eines Simulationslaufs ausgeführt wird. In das Textfenster wird folgender Text eingegeben:

```
ProzentsatzWaschprog1 := BarGaugeValue named: 'Prozentsatz'.
```

Wir früher beschrieben wird dieser Text akzeptiert, indem im Initialisierungsfenster (d.h. mit dem Mauszeiger im Initialisierungsfenster) die rechte Maustaste gedrückt und der Menüpunkt accept ausgewählt wird.

Die eingegebene Zeile wird nicht ohne weiteres akzeptiert, weil der Bezeichner **ProzentsatzWaschprog1** nicht bekannt ist. Es wird deshalb ein Abfragefenster angezeigt, in dem der Knopf **global** gedrückt wird. Damit wurde ProzentsatzWaschprog1

als globale Variable vereinbart, die im gesamten Modell bekannt ist und auf deren Inhalt, den BarGauge, im gesamten Modell zugegriffen werden kann. Globale Variable beginnen mit einem Großbuchstaben.

Das jeweilige Waschprogramm wird beim Einfahren in den Waschplatz ausgewählt. Deshalb, wie früher schon beschrieben, das Menü der Transition 'Einfahren in den Waschplatz' aufrufen und den Menüpunkt **action code** auswählen. Es öffnet sich wieder ein Textfenster, in das der Aktionscode, der beim Feuern einer Transition ausgeführt wird, eingegeben wird:

```
balkenwert := ProzentsatzWaschprog1 value.
program := (Bernoulli parameter: balkenwert / 100) next
```

Die erste Zeile weist einer lokalen Variablen **balkenwert** der Transition den eingestellten Wert des Balkens zu. In der zweiten Zeile wird der Konnektorvariablen **program** der "nächste" Wert einer Bernoulli-Verteilung mit der Wahrscheinlichkeit 'balkenwert / 100' zugewiesen. Sie liefert den Wert 1, wenn Waschprogramm 1 verwendet wird, andernfalls den Wert 0.

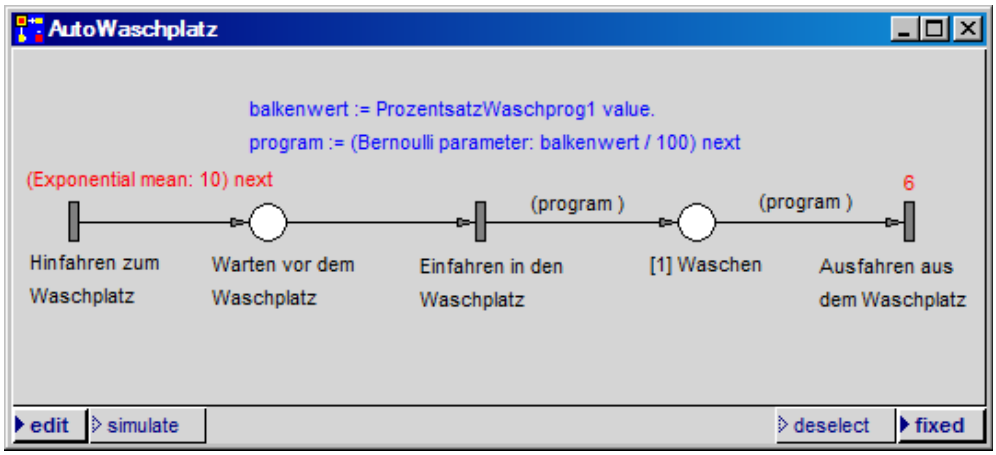



Abb. 3.24: Netzfenster mit Aktionscode

Der eingegebene Programmcode wird wieder mit der accept-Funktion in die Transition eingefügt. Dabei öffnet sich erneut ein Fenster, in dem darauf hingewiesen wird, dass die Variable 'balkenwert' nicht bekannt ist. Diesmal wird der Knopf **temp** gedrückt und damit die Variable 'balkenwert' als lokale Variable der Transition vereinbart. Auch den Aktionscode der Transition kann man wie früher beschrieben, an eine geeignete Stelle im Netzfenster schieben. Der derzeitige Stand des Netzfensters ist in Abb. 3.24 dargestellt.

Bei Ablauf des Netzes wird der aktuelle Wert der Bernoulli-Verteilung zugeordnet zur Marke (dem Fahrzeug) gespeichert. Auf ihn kann über die Konnektorvariable 'program' zugegriffen werden. Zur Einstellung der richtigen Verzögerungszeit (Waschzeit) wird der Verzögerungscode der Transition 'Ausfahren aus dem Waschplatz' wie folgt geändert:

```
program = 1 ifTrue: [6] ifFalse: [9]
```

Das ist ein sog. 'bedingter Ausdruck' (conditional expression). Er liefert das Ergebnis 6, wenn program = 1 ist, also das Waschprogramm 1 gewählt wurde, andernfalls den Wert 9.

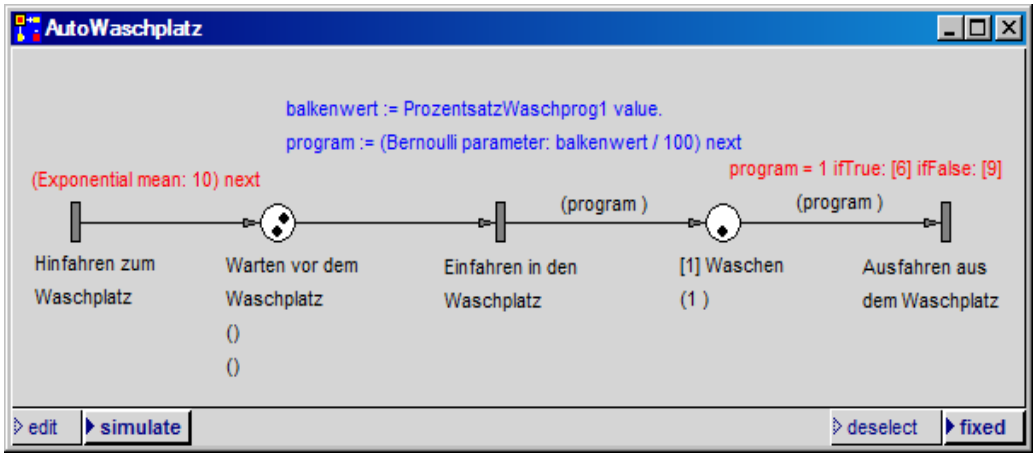



Abb. 3.25: Netzfenster mit Extra Codes

Bei der Änderung des Codes einer Transition braucht man nicht über das Menü der Transition zu gehen, sondern kann den Code direkt ändern. Markiert wird der aktuelle Verzögerungscode (Delay-Code) der Transition 'Ausfahren aus dem Waschplatz'. Ein kleiner schwarzer Punkt wird in der Mitte des Codes angezeigt. Dann mit der **re.MT** das Menü des Codes aufrufen und **inspect** wählen. Es öffnet sich ein Fenster mit dem aktuellen Delay-Code. In diesem wird der aktuelle Code 6 durch den eben angegebene bedingten Ausdruck ersetzt und schließlich wieder mit **accept** in die Transition eingebaut.

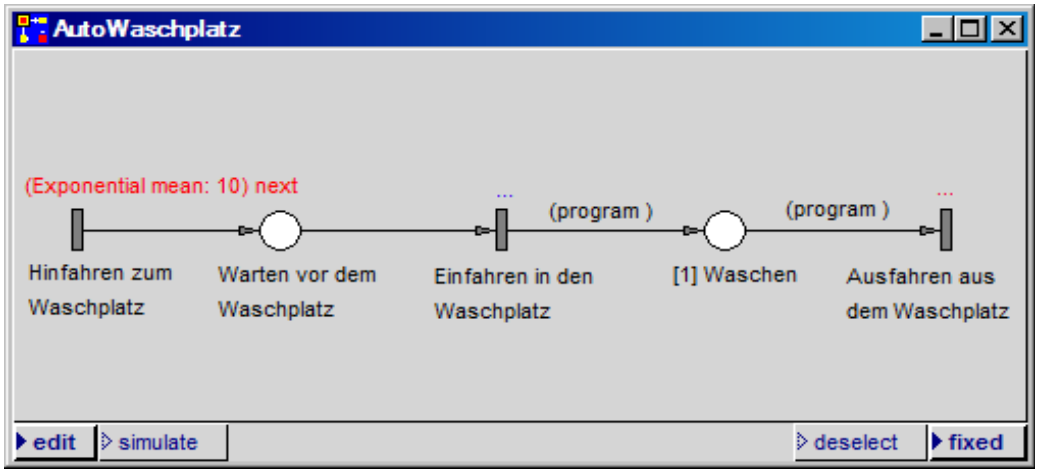



Abb. 3.26: Netzfenster mit Drei-Punkte-Codes

Das Netzfenster ist in Abb. 3.25 dargestellt. Die Betrachtung des Fensters lässt ahnen, dass bei vielen längeren Inskriptionen die Darstellung des Netzes unübersichtlich werden kann. Um das zu vermeiden, kann man die Transitionscodes verbergen und jeweils durch drei Punkte … ersetzen. Dazu wieder das Menü des jeweiligen Codes aufrufen, aber diesem nicht **accept**, sondern das Drei-Punkte-Menü **...** in der letzten Menüzeile auswählen. Die Inskription im Netzfenster wird durch drei Punkte in der

jeweiligen Textfarbe ersetzt. Die drei Punkte werden an der linken oberen Ecke der vormaligen Inskription erzeugt. Damit man sehen kann, zu welcher Transition der Code gehört, kann man das Menü der drei Punkte aufrufen. Dazu die Drei-Punkte mit der **li.MT** markieren, mit der **re.MT** das zugeordnete Menu aufrufen und **owner** auswählen. Der Mauszeiger wird auf der Transition platziert, zu denen die Drei-Punkte gehören.

Besser ist es, die drei Punkte im Netzfenster in die Nähe der Transition zu schieben, zu der sie gehören. Das geht genau so, wie früher bei den anderen Inskriptionen beschrieben. Abb. 3.26 zeigt das Netzfenster mit teilweise durch drei Punkte ersetzten Inskriptionen.

Will man statt der drei Punkte wieder den Text der Inskription anzeigen, so sind die drei Punkte zu markieren und im Menü der **re.MT** ist der Menüpunkt **inscript** zu wählen. Es öffnet sich ein Textfenster mit der Inskription. In diesem das Menü der **re.MT** aufrufen und wieder den Menüpunkt … wählen. Dieser Menüpunkt wirkt also wie ein Schalter. Um die hinter drei Punkten verborgene Inskription kann man natürlich auch über das Menü der Transition anzeigen.

Das Modell ist damit fertig gestellt und man kann wieder in den Simulationsmodus wechseln, um mit dem Modell zu experimentieren. Wenn man den Balkenschieberegel 'Prozentsatz' auf 100 % stellt, erhält man wieder das Ergebnis aus Abb. 2.9. Stellt man den Schieber auf 0 % ein, so ergibt sich das in Abb. 3.27 dargestellte Ergebnis. Es zeigt eine wesentlich bessere Auslastung der Anlage, die durch erheblich längere Warteschlangen erkauft wird.

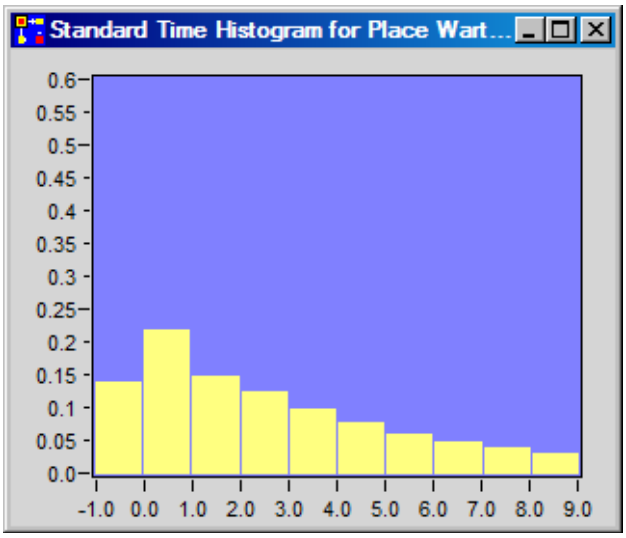



Abb. 3.27: Zeitanteile der verschiedenen Warteschlangenlängen bei einer Waschzeit von 9 Minuten

## 3.3 Durchlaufzeiten

In diesem Abschnitt soll der Umgang mit der Simulationszeit gezeigt werden. Die aktuelle Simulationszeit ist in der globalen Systemvariablen **CurrentTime** gespeichert und wird beim Initialisieren eines Modells auf Null gesetzt.

Bei der Änderung des Netzes kann man sich jetzt etwas kürzer fassen, da viele der Editiermaßnahmen früher zum Teil mehrfach beschrieben wurden. Die Änderungen gehen aus Abb. 3.28 hervor.

Für alle Konnektoren wird eine weitere Konnektorvariable 'time' eingeführt. Bei den beiden linken Konnektoren geht das wie früher beschrieben. Bei einem der beiden rechten Konnektoren wird wie folgt verfahren: im Fenster zur Attributierung der Kon-

nektoren (Abb. 3.21) wird mit der **li.MT** das Selection-Menü im linken Teilfenster aufgerufen und darin der Menüpunkt **insert** gewählt. Eine weitere Konnektorvariable wird vor der schon vorhandenen Variablen eingesetzt. Wollte man die Variable nach der schon vorhandenen Variablen einsetzen, so wäre zunächst die selektierte, gelb unterlegte Variable durch Klicken mit **li.MT** zu deselektieren, danach mit der **re.MT** das No-Selection-Menü des linken Teilfensters aufzurufen und schließlich der Menüpunkt **add** zu wählen. Im rechten Teilfenster den Bezeichner **time** eingeben und im Menü des rechten Teilfensters **accept** ausführen. Der zweite der beiden rechten Konnektoren wird wie früher durch Kopieren attributiert.

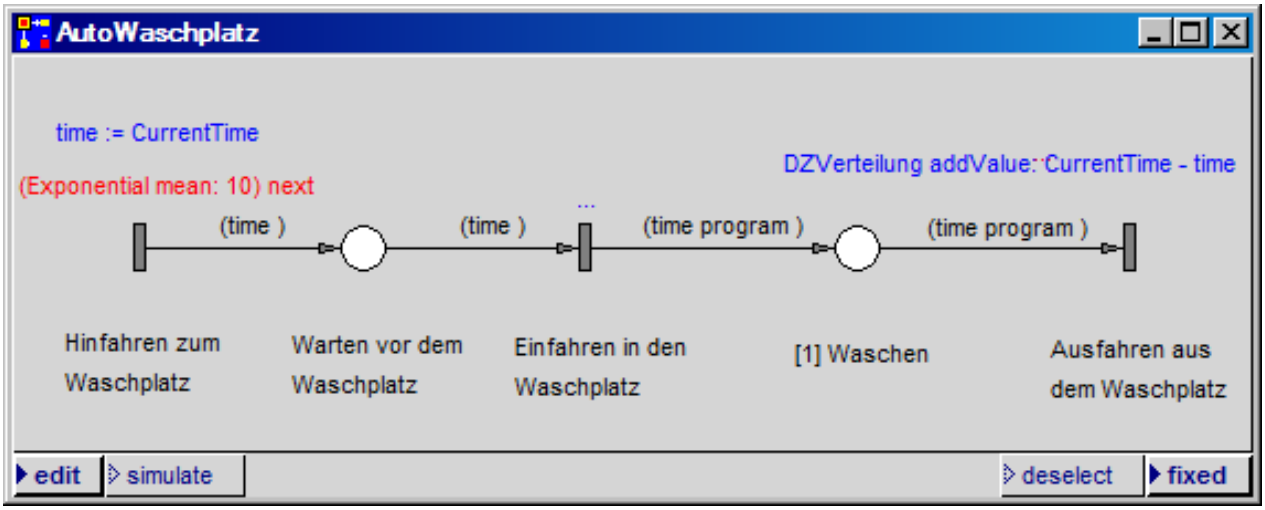



Abb. 3.28: Änderung im Netzfenster

In der Transition 'Hinfahren zum Waschplatz' wird der Konnektorvariablen **time** die aktuelle Zeit zugewiesen:

time := CurrentTime.

Nach Durchlaufen der Waschanlage beträgt die Durchlaufzeit:

CurrentTime – time.

Es ist interessant, die Verteilung der Durchlaufzeiten anzusehen. Dazu wird im Views-Menü der PACE-Menüleiste der Menüpunkt **histograms** und in seinem Submenü der Punkt **counts** ausgewählt. Das Count-Histogram zeigt im vorliegenden Fall die Anzahl der Marken bzw. Fahrzeuge über der Durchlaufzeit an. Das Fenster erhält den Namen 'Verteilung der Durchlaufzeiten' (**mi.MT**) und wird wie folgt skaliert (Abb. 3.29):

| | |
|---|---|
| Ordinate: | Maximalwert = 3200<br>Anzahl der Inskriptionen= 10 |
| Abszisse: | Maximalwert = 60<br>Inskriptionen = 10<br>Balken = 60 |

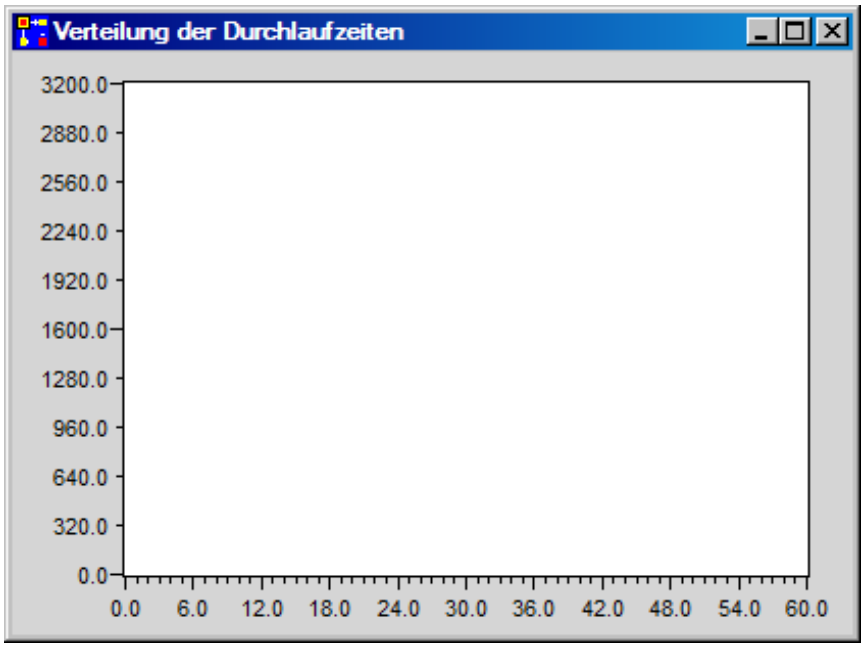



Abb. 3.29: Allgemeines Count-Histogramm

Das Count-Histogramm wird wie früher im Initialisierungscode an das Modell angeschlossen. Dazu wird dieser um die Zeilen:

```
DZVerteilung := CountHistogram named: 'Verteilung der Durchlaufzeiten'.
DZVerteilung clear.
```

erweitert. Die zweite Zeile löscht während der Initialisierung den Inhalt des Histogramms.

Der Aktionscode der Transition 'Ausfahren aus dem Waschplatz' wird um den Ausgabecode für das Standard-Histogramm erweitert. Die Erweiterung lautet:

```
DZVerteilung addValue: CurrentTime – time.
```

Führt man das Modell mit einem Prozentsatz von 50 % aus, so erhält man die in Abb. 3.30 dargestellte Verteilung der Durchlaufzeiten.

In diesem Ergebnis kommen sehr große unrealistische Durchlaufzeiten vor. Sie rühren von den im Modell auftretenden großen Warteschlangen vor der Anlage her, die aber in der Praxis nicht vorkommen, weil Fahrzeughalter ab einer Warteschlangenlänge von etwa 3 Fahrzeugen die Anlage nicht mehr anfahren. Diesem Umstand wird im erweiterten Modell (Kapitel 4) Rechnung getragen.

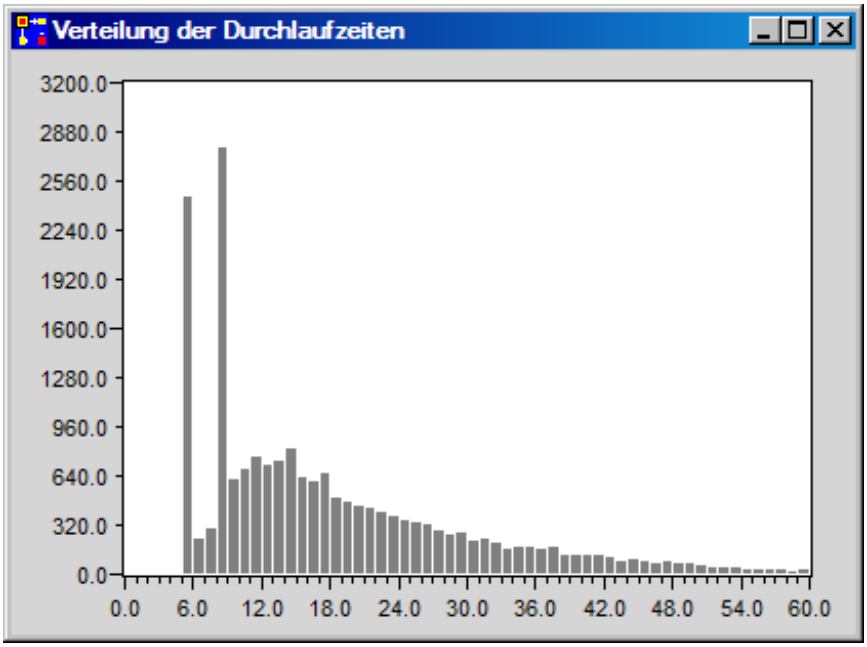



Abb. 3.30: Verteilung der Durchlaufzeiten

## 3.4 Ikonisierung

Im Allgemeinen kann man einem Netz nicht ohne weiteres ansehen, welche Prozesse dargestellt sind. Der Anwender, der später das Modell verwenden soll und den die Implementierung des Modells normalerweise nur am Rande interessiert, kann mit einem Modell besser umgehen, wenn die Default-Netzelemente durch anwendungsspezifische Bilder ersetzt werden (Abb. 2.10). Wie das geht, wird jetzt beschrieben.

Die hier benötigten Bilder sind in der Ikonendatei **Autowaschplatz.icn** im Verzeichnis **nets** des PACE-Installationsverzeichnisses gespeichert. Zum Laden wird die Zeile AutoWaschplatz in der Netzliste markiert. Dann wird das **Extras**-Menü der PACE-Hauptleiste und darin das **icons**-Submenü angewählt. Darin den Menüpunkt **load icon file** auswählen.

Es öffnet sich ein Windowsfenster zum Laden einer Icon-Datei. Darin werden die Einträge im **nets**-Verzeichnis angezeigt. Ausgewählt wird die Datei **Autowaschplatz.icn**, die entweder durch Doppelklick auf den Eintrag in der Explorer-Liste oder durch Drükken des Knopfs **Öffnen** in das Modell geladen wird.

Die geladenen Ikonen können betrachtet werden, indem der Menüpunkt **icons** im Extra-Menü und im dann angezeigten Submenü der Menüpunkt **individual icons** ausgewählt wird. Es öffnet sich ein Fenster, in dem die Namen aller im Modell gespeicherten Ikonen bzw. Bilder aufgelistet sind. Klickt man mit der **li.MT** auf einen Namen, so wird das zugeordnete Bild in der rechten oberen Ecke des Fensters solange angezeigt, wie die Maustaste gedrückt wird (Abb. 3.31).

Interessant ist auch das Selection-Menü des Fensters, das mit der **re.MT** aufgerufen wird. Es wird empfohlen, die verschiedenen Menüpunkte, insbesondere das Ersetzen einer Ikone (**from screen**, **from clipboard**, usw.), das Skalieren einer Ikone (**scale**)

und das Bleichen einer Ikone (**fade**) auszuprobieren. Die Bilder Gesamtanlage1 und Gesamtanlage2 sind durch einmaliges und zweimaliges Bleichen aus dem Bild Gesamtanlage hervorgegangen. Mit dem Icon-Editor kann ein Bild verändert werden. Das Non-Selection-Menü des Fenster enthält nur den Menüpunkt **add**, mit dem ein neuer Name zur Liste hinzugefügt wird. Diesem ist dann mit einem der from…-Menüpunkte ein Bild zuzuordnen.


Abb. 3.31: Ikonenliste

In dem in Abb. 3.28 dargestellten Netz wird die Transition 'Hinfahren zum Waschplatz' markiert und im Submenü **icon** der Menüpunkt **individual** angewählt. In dem sich öffenden Fenster wird die Zeile **Hinfahren** selektiert und dann der **ok**-Knopf gedrückt. Von dem Bild werden die Inskriptionen teilweise überdeckt, die entsprechend zu verschieben sind. In der gleichen Weise werden den weiteren Netzelementen die Bilder 'Warten, 'Einfahren, 'Waschen' und 'Ausfahren' zugeordnet. Alle Code-Inskriptionen werden durch Drei-Punkte-Inskriptionen ersetzt.

Um die Überschrift im Fenster darzustellen, wird zunächst ein Netzelement, z.B. eine Stelle, an der Position im Netzfenster erzeugt, an der später die Überschrift erscheinen soll. Dann wird das Netzelement durch das Bild 'Überschrift' ersetzt.

Den derzeitigen Stand des Netzfenster zeigt Abb. 3.32.

Will man auch das Standardsymbol für Marken (ein kleiner schwarz gefüllter Kreis) durch Abbilder der Objekte ersetzen, die sich durch das Netz bewegen, so ist wie folgt zu verfahren: der Ausgangs-Konnektor (einer Transition), über den die Ikone laufen soll, wird markiert und mit der **re.MT** sein Menü aufgerufen. Darin wird der Menüpunkt **icon function** angewählt. Im dann angezeigten Submenü wird der Punkt **edit** gewählt. Es öffnet sich das in Abb. 3.33 gezeigte Fenster.

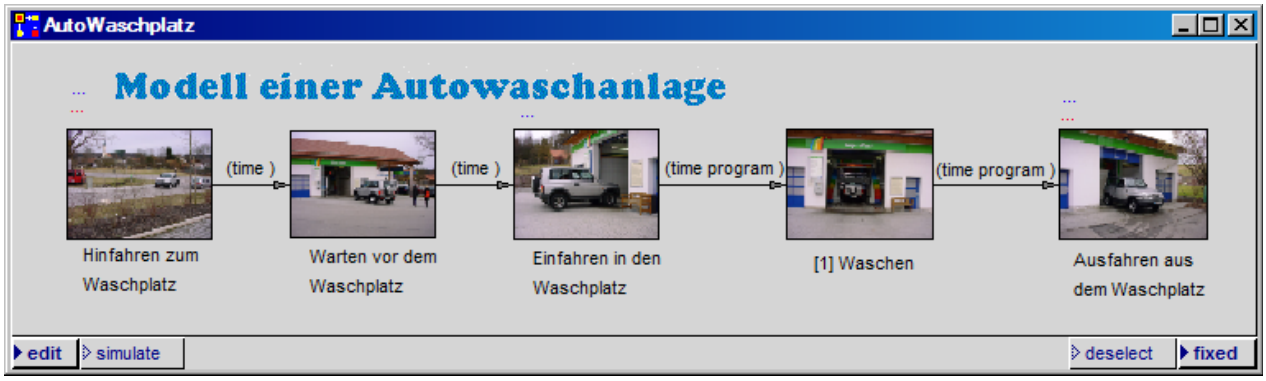



Abb. 3.32: Ikonisierung der Elemente im Netzfenster

In dem Fenster wird ein Smalltalk-Block angezeigt, der jedes Mal ausgewertet wird, wenn eine Marke über den Konnektor läuft. Der Block liefert als Ergebnis den Namen der Ikone, welche verwendet werden soll, in Form eines sog. Symbols. Ein Symbol geht aus dem Namen hervor, indem das Zeichen # vorangestellt wird, (Beispiel: #limousine).

Wird als Ergebnis des Blocks nur ein Name angegeben, so läuft immer das zugeordnete Ikon über den Konnektor. Der Block sieht dann wie folgt aus:

[:t| #limousine].

Wie früher wird der Code durch Ausführen von **accept** im **re.MT**-Menü des Fensters gültig. Danach ist das Ikon-Fenster zu schließen.

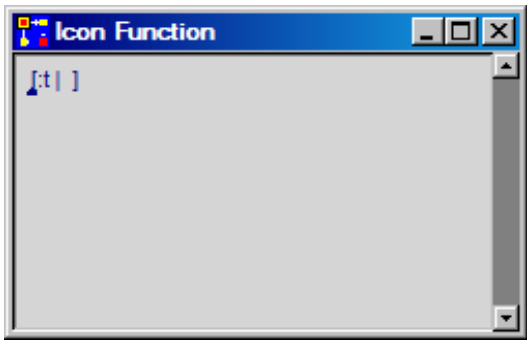



Abb. 3.33: Fenster für die Vorgabe einer Markenikone

Um auch den zweiten Ausgangskonnektor, der von der Transition "Einfahren in den Waschplatz' nach der Stelle 'Waschen' führt, für die Marken-Ikonisierung zu attributieren, kann man wieder eine copy-Funktion verwenden. Dazu den eben schon attributierten Konnektor markieren, im **re.MT**-Menü wieder den Menüpunkt **icon function** wählen und im Submenü **copy** ausführen. Jetzt den zweiten Ausgangskonnektor markieren, wieder **icon function** aufrufen und **paste** wählen.

Wenn man will, kann man auch ein Hintergrundbild einsetzen. Dazu im No-Selection-Menü des Fenster in Abb. 3.32 den Menüpunkt **insert background image** auswählen und in dem sich öffnenden Auswahlfenster die Zeile **Gesamtanlage2** auswählen. Um das störende Flackern beim Editieren zu vermeiden, sollten Hintergrundbilder erst nach der Fertigstellung eines Moduls oder Modells eingesetzt werden.

Damit ist das Netz fertiggestellt und PACE wird verlassen. Hierfür im File-Menü der PACE-Menüleiste den Menüpunkt **leave PACE** anwählen. Daraufhin wird ein Abfragefenster angezeigt, das zum Sichern des Modells auffordert. Sie sollten **yes** wählen, um die Sicherung auszuführen. Der weitere Ablauf wurde früher (am Ende von Abschnitt 3.1) schon beschrieben. Nach dem Abspeichern des Modells wird PACE automatisch beendet.

# 4. Bausteine (Module)

Ähnlich wie wir in unserer Sprache Begriffe für komplexe Zusammenhänge bilden, um uns effizient und verständlich ausdrücken zu können, benötigt man auch in Netzen eine Möglichkeit, in sich abgeschlossene Teilnetze, in denen bestimmte Teilaufgaben abgehandelt werden, in sog. Modulen oder Bausteinen zusammenzufassen. Erst durch diese Zusammenfassung kann man auch die in der Realität vorgezeichnete Hierarchisierung nachbilden und gewinnt eine größere Übersichtlichkeit der Netze: Netze werden verständlich.

Viele Simulator-Entwicklungssysteme kennen nur vorgefertigte Bausteine und setzen Simulatoren aus diesen Bausteinen zusammen. Da die Welt sich schwer über einen Leisten schlagen lässt, erfordert diese Vorgehensweise häufig eine Unzahl von teilweise schwer verständlichen Adaptionsparametern, mit denen vorgefertigten Bausteine an die aktuelle Situation angepasst werden und die das Auffinden von Fehlern zu einem "Abend-füllenden" Thema machen können. Trotzdem bilden die so aufgebauten Modelle die aktuelle Situation häufig nur näherungsweise ab und liefern dann nur beschränkt gültige Ergebnisse. Anpassungen durch Änderungen der Bausteine sind schwierig und können in der Regel nur von Spezialisten ausgeführt werden.

PACE geht hier einen anderen Weg, indem es Mittel für die exakte Abbildung der Realität in ein animierbares Modell bereitstellt, das sich Schritt für Schritt verifizieren lässt. Um die hierarchische Struktur eines Prozesssystems darzustellen, werden wiederverwendbare Module, auch als Bausteine oder Teilnetze bezeichnet, verwendet, die mit den früher beschriebenen Netzelementen erstellt werden. Auf diese Weise kann man mit den Mittel von PACE adaptierbare Bausteine und Baustein-Bibliotheken erstellen, die, falls sie im Einzelfall nicht passen bzw. die Realität zu ungenau abbilden, vom Anwender selbst angepasst und erweitert werden können.

## 4.1 Erstellen eines Bausteins (Moduls)

Ziel dieses Abschnitts ist es, einen adaptierbaren Baustein **Waschplatz** zu erstellen, den man in anderen Netzen verwenden kann. Er soll über die folgenden vier Parameter an einen aktuell vorliegenden Waschplatz anpassbar sein:

Waschzeit1
Waschzeit2
Prozentsatz
Warteschlangenlänge

und als Ergebnis, in Form eines Markenattributs, die Durchlaufzeit des gerade gewaschenen Fahrzeugs liefern. Unter der Durchlaufzeit wird dabei die Zeitspanne vom Eintreffen eines Fahrzeuges vor der Waschanlage bis zum Verlassen der Waschanlage verstanden (siehe auch Abschnitt 3.3). Die vier genannten Parameter wurden in früheren Abschnitten schon diskutiert.

Das jetzige Vorhaben unterscheidet sich von den in Kapitel 3 erstellten Modellen dadurch, dass bei der Adaption keine Netzänderungen vorgenommen werden sollen. Alle Anpassungen werden über die Vorbesetzung von Argumenten (Parametern) ausgeführt. Auf diese Weise braucht der spätere Anwender des Moduls keine detaillierten Kenntnisse über den Aufbau des Moduls zu besitzen; er muss nur wissen, welche Aufgabe der Modul löst, was die Adaptionsparameter bedeuten und wie sie eingestellt werden.

Grundsätzlich hat man zwei Möglichkeiten für die Erstellung eines Moduls:

- Man beginnt mit einem leeren Modul und baut das Netz neu auf.
- Falls ein verwendbares Ausgangsnetz schon vorliegt, wird es ggf. verändert und der gewünschte Modul wird durch Zusammenfassen (Menüpunkt: **coarsen**) von Netzelementen erzeugt.

Obwohl die an zweiter Stelle genannte Vorgehensweise unter Verwendung eines der schon erstellten Netze möglich wäre, wird hier die zuerst genannte Methode gewählt, weil die Beschreibung der durchzuführenden Änderungen den Blick vom Wesentlichen ablenken würde.

Gestartet wird mit einem neuen Modell, welches den Namen **Modulerstellung** erhält. Nach dem Öffnen des Netzfensters werden über das No-Selection-Menü des Fensters die drei in Abb. 4.1 dargestellten Netzelemente erzeugt und über Konnektoren verbunden. Das durch ein Quadrat dargestellte Netzelement ist ein Modul und wird mit dem Menüpunkt **module** erzeugt.

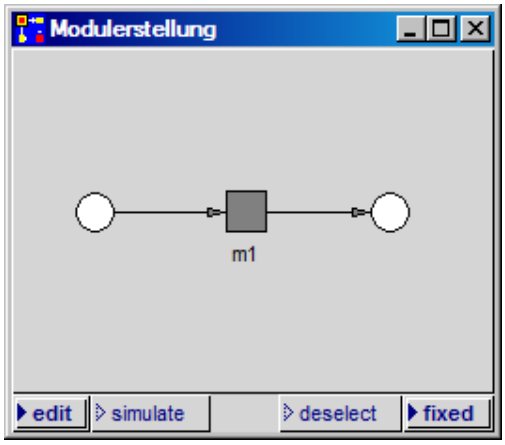



Abb. 4.1: Ausgangsnetz

Die Netzelemente werden wie in Abb. 4.2 gezeigt beschriftet.

Im Fall des Moduls wird der Default-Modul-Bezeichner m1 gegen den Bezeichner 'Waschplatz' ausgetauscht, indem m1 mit der **li.MT** markiert und dann in dem Menü der **re.MT** mit dem Menüpunkt **inspect** das Eingabefenster für den Modulnamen aufgemacht wird. Darin den Namen Waschplatz eingeben und entweder im Menü der **re.MT** den Menüpunkt **accept** wählen oder die Return-Taste betätigen.

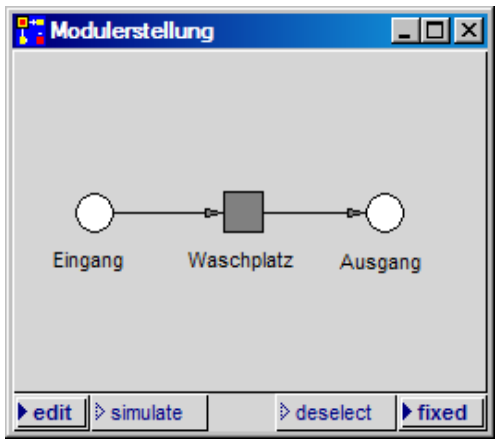



Abb. 4.2: Ausgangsnetz mit Beschriftung

Damit der Modul 'Waschplatz' unabhängig von seiner Umgebung ist, dürfen in ihm keine externen Bezüge auftreten. Deshalb ist es nicht möglich, die Parameter Waschzeit1,

Waschzeit2, usw. wie früher durch globale Variablen darzustellen. Es würde auch nicht helfen, wenn die globalen Variablen nachträglich in dem Modell deklariert würden, in das der Modul später eingesetzt wird! Wird nämlich der Modul mehrfach eingesetzt, so würden alle eingesetzten Module auf dieselben globalen Variablen zugreifen. Ohne Änderung der Netze des Moduls wäre es daher nicht möglich, Waschplätze mit unterschiedlichen Waschzeiten zu verwenden.

Um Module in sich abgeschlossen modellieren zu können, wurden sog. Netz- oder Modulvariablen eingeführt, die im gesamten Modul und falls vorhanden, seinen Untermodulen bekannt sind. Um die zu Beginn des Abschnitts angegebenen Parameter als Netzvariable zu deklarieren, wird der Modul 'Waschplatz' markiert und in Menü der **re.MT** der Menüpunkt **net variables** gewählt. Es öffnet sich das in Abb. 4.3 gezeigt Fenster zur Vereinbarung von Netzvariablen. Im linken Teilfenster werden die Namen von Netzvariablen in Form von Symbolen eingesetzt. Ihnen können im rechten Teilfenster, je nachdem, welcher der Knöpfe **initial value** oder **current value** gedrückt ist, Werte zugewiesen werden. Den als Initialwert angegebenen Wert wird der Netzvariablen beim Initialisieren eines Modells zugewiesen.

Im Menü der **re.MT** im linken Teilfenster wird der Menüpunkt **add** gewählt. Es öffnet sich ein Eingabefenster, in das der Name einer Netzvariablen in Form eines Symbols (d.h. mit vorangestelltem #-Zeichen) einzugeben ist. Eingegeben wird **#Waschzeit1**. Die Eingabe wird mit der **Return**-Taste abgeschlossen. Danach den Mauszeiger im rechten Teilfenster positionieren, die Waschzeit 6 eingeben und im Menü des rechten Teilfensters **accept** wählen. Danach den Bezeichner #Waschzeit1 im linken Teilfenster deselektieren (mit der **li.MT** auf den Bezeichner klicken; die gelbe Unterlegung verschwindet), die weiteren Netzvariablen wie in Abb. 4.4 aufgelistet eingeben und ihnen der Reihe nach die Anfangswerte 9, 50 und 0 zuweisen.

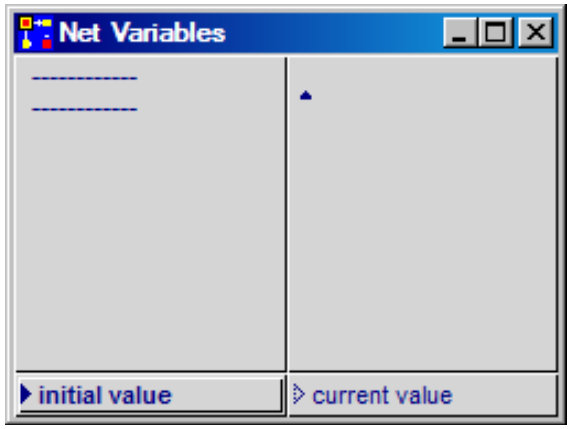



Abb. 4.3: Fenster zur Vereinbarung von Netzvariablen

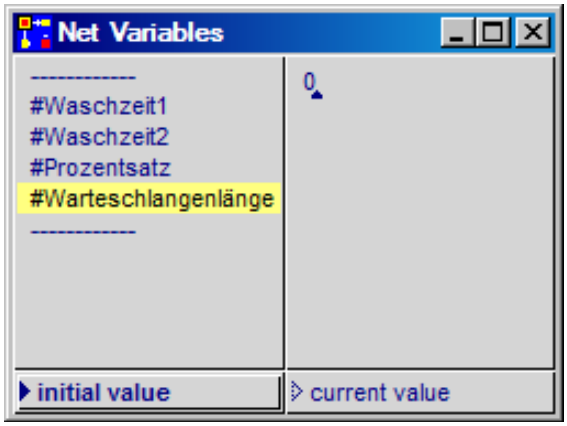



Abb. 4.4: Fenster mit vereinbarten Netzvariablen

Auf Netzvariable wird über Botschaften zugegriffen. Der aktuelle Wert der Netzvariablen **Warteschlangenlänge** ergibt der folgende Botschaft:

(self at: #Warteschlangenlänge) value

Der Wert 40 wird der Netzvariablen #Prozentsatz durch

(self at: #Prozentsatz) value: 40

zugewiesen.

Das Netzfenster des Moduls Waschplatz wird angezeigt, indem der Modul markiert und in seinem **re.MT**-Menü der Menüpunkt **subnet** gewählt wird (Abb. 4.5). Das sich öffenende Fenster des Moduls zeigt die beiden Stellen 'Eingang' und 'Ausgang', welche die Schnittstellen zur Umgebung des Moduls bilden. Um anzudeuten, dass diese außerhalb des Moduls vereinbart sind, im Modul also nur den Charakter von Platzhaltern besitzen, sind sie im Netz des Moduls schwächer (bleicher) gezeichnet.

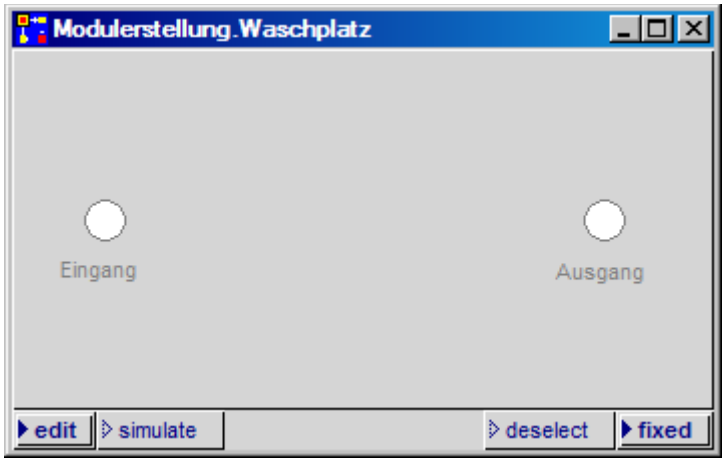



Abb. 4.5: Netzfenster nach dem Öffnen

Im Modul wird nun wie früher das in Abb. 4.6 gezeigte Netz zunächst ohne Code-Inskriptionen gezeichnet. Letztere werden nachfolgend besprochen.

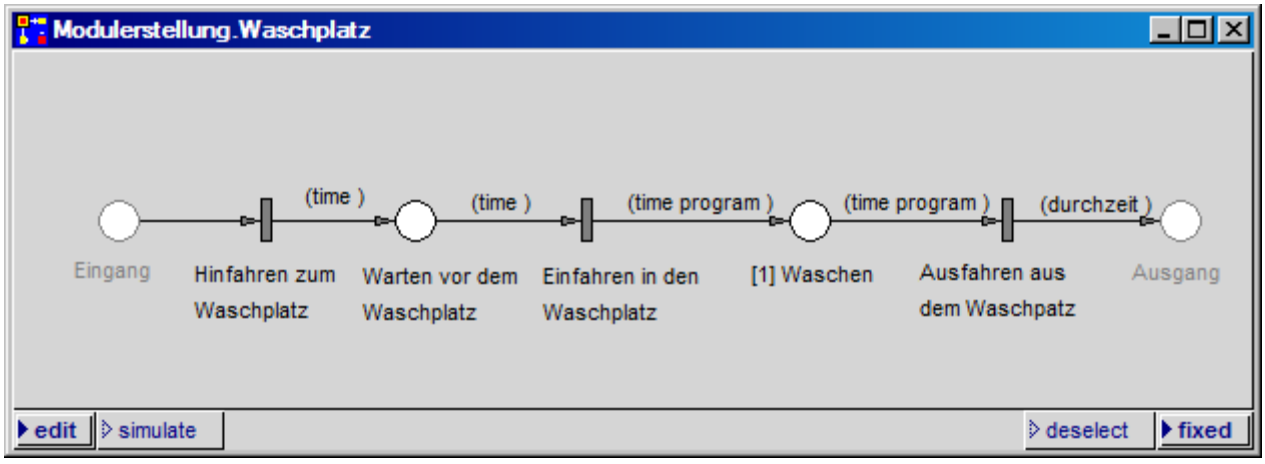



Abb. 4.6: Netz des Moduls Waschplatz ohne Code-Inskriptionen

Da die Beschickung des Waschplatzes von außen erfolgt, wird in der Transition 'Hinfahren zum Waschplatz' anders als in Kapitel 3 kein Delay-Code benötigt. Der Actioncode lautet.

```
time := CurrentTime.
temp := (self at: #Warteschlangenlänge).
temp value: (temp value + 1).
```

Die erste Zeile speichert die aktuelle Simulationszeit in der Konnektorvariablen time. Sie wird während der Simulation mit einer Marke durch das Netz transportiert und in der Transition 'Ausfahren aus dem Waschplatz' mit der dann vorliegenden Zeit verglichen. Aus der Differenz ergibt sich die Aufenthaltsdauer des Fahrzeugs in der Waschanlage.

In der nächsten Zeile wird eine temporäre Variable temp eingeführt, der das Objekt Netzvariable 'Warteschlangenlänge' (nicht deren Wert!) zugewiesen wird. An das Objekt werden in der dritten Zeile die früher angegebenen Botschaften an Netzvariable gesandt, um die Netzvariable 'Warteschlangenlänge' auf den neuesten Stand zu bringen.

Da die Transitions-lokale Variable temp nicht bekannt ist, wird bei der Ausführung des Menüpunkts **accept** ein Auswahlmenü zur Vereinbarung von temp angezeigt, in dem der Knopf **temp** gedrückt wird (Abb. 4.7).

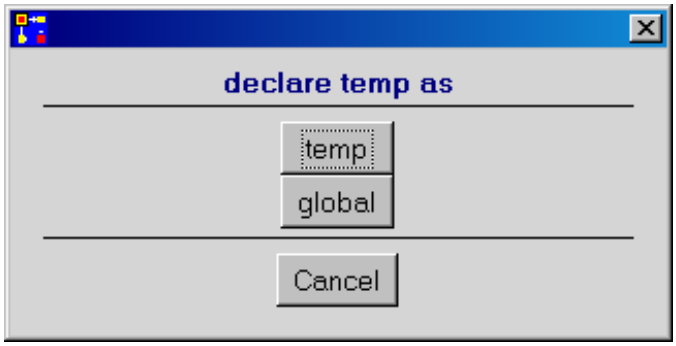



Abb. 4.7: Abfragefenster zur Vereinbarung der Variablen temp

Sie können die aktuell in einer Transition vorliegenden lokalen Variablen ansehen, wenn Sie im **re.MT**-Menü der Transition den Menüpunkt **temporaries** wählen. Dort können Sie weitere lokale Variablen im No-Selection-Menü mit add hinzufügen und selektierte temporäre Variablen mit dem **re-MT**-Menü entfernen oder umbenennen (Vorsicht !).

Der Action-Code der Transition 'Einfahren in den Waschplatz' muss die Warteschlangenlänge um 1 reduzieren und das Waschprogramm auswählen. Er lautet:

```
temp := (self at: #Warteschlangenlänge).
temp value: (temp value - 1).
balkenwert := (self at: #Prozentsatz) value.
program := (Bernoulli parameter: balkenwert / 100) next.
```

Seine Interpretation dürfte nach den früheren Beschreibungen kein Problem mehr bereiten.

Bleibt noch der Delay- und der Action-Code für die Transition 'Ausfahren aus dem Waschplatz'.

Den Delay-Code kann man aus dem früheren Code (siehe Abb. 3.25) ableiten. Er lautet:

```
program = 1 ifTrue: [(self at: #Waschzeit1) value]
            ifFalse: [(self at: #Waschzeit2) value].
```

Der Action-Code speichert die Durchlaufzeit in der Konnektorvariablen durchzeit:

durchzeit := CurrentTime – time.

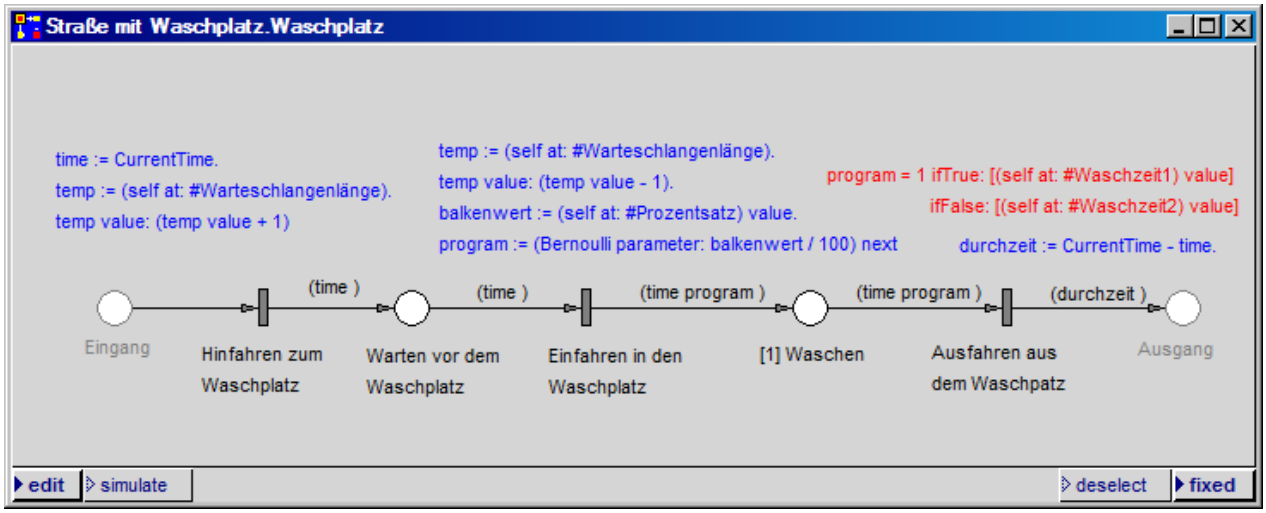



Abb 4.8: Netz des Moduls Waschplatz mit Code-Inskriptionen

Übrig bleibt noch die bei Entwicklern meist unbeliebte, aber für die Qualitätssicherung und für die Information späterer Nutzer des Moduls wichtige Restarbeit: **Dokumentation**. Hierfür sind in PACE mehrere Möglichkeiten vorgesehen:

1. Information über die Entwicklung des Bausteins (Moduls)
   Markiert wird der Modul Waschplatz in der Netzliste, danach mit **re-MT** das Selection-Menü des Fensters aufrufen und den Menüpunkt **development info** auswählen. Es öffnet sich ein Fenster, in das Angaben zur Entwicklung des Moduls eingegeben und mit **accept** gültig gemacht werden. Aufgrund dieser Information können bei Problemen mit dem Modul oder bei späteren Erweiterungen z.B. die durchgeführten Änderungen und die Entwickler des Moduls gefunden werden.

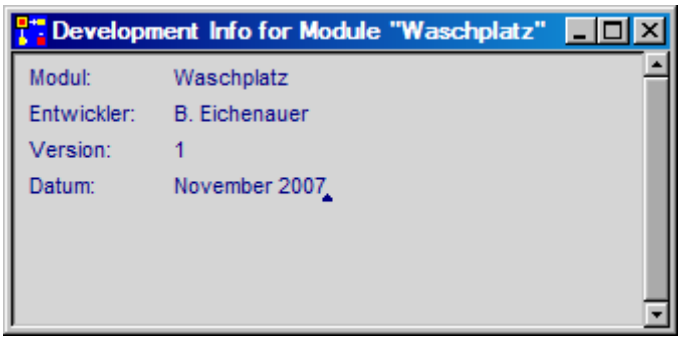



Abb. 4.9: Development Info für dem Modul 'Waschplatz'

2. Beschreibung des Bausteins (Moduls)
   Einzelne Netzelemente werden mit Einträgen dokumentiert, die an die Netzelemente angehängt werden. Im vorliegenden Fall den Modul im Netz Abb. 4.2 markieren, mit der **re.MT** das Menü des Moduls aufmachen und den Menüpunkt **purpose description** anwählen. Es öffnet sich ein Textfenster, in das Information zum Modul eingegeben wird.

Als Beispiel sind in Abb. 4.10 die wichtigsten Angaben zum Modul 'Waschplatz' für den späteren Nutzer des Moduls aufgeschrieben. Im Netz des Moduls können wahlweise weitere implementierungs-spezifische Angaben an die verwendeten Netzelemente angehängt werden.

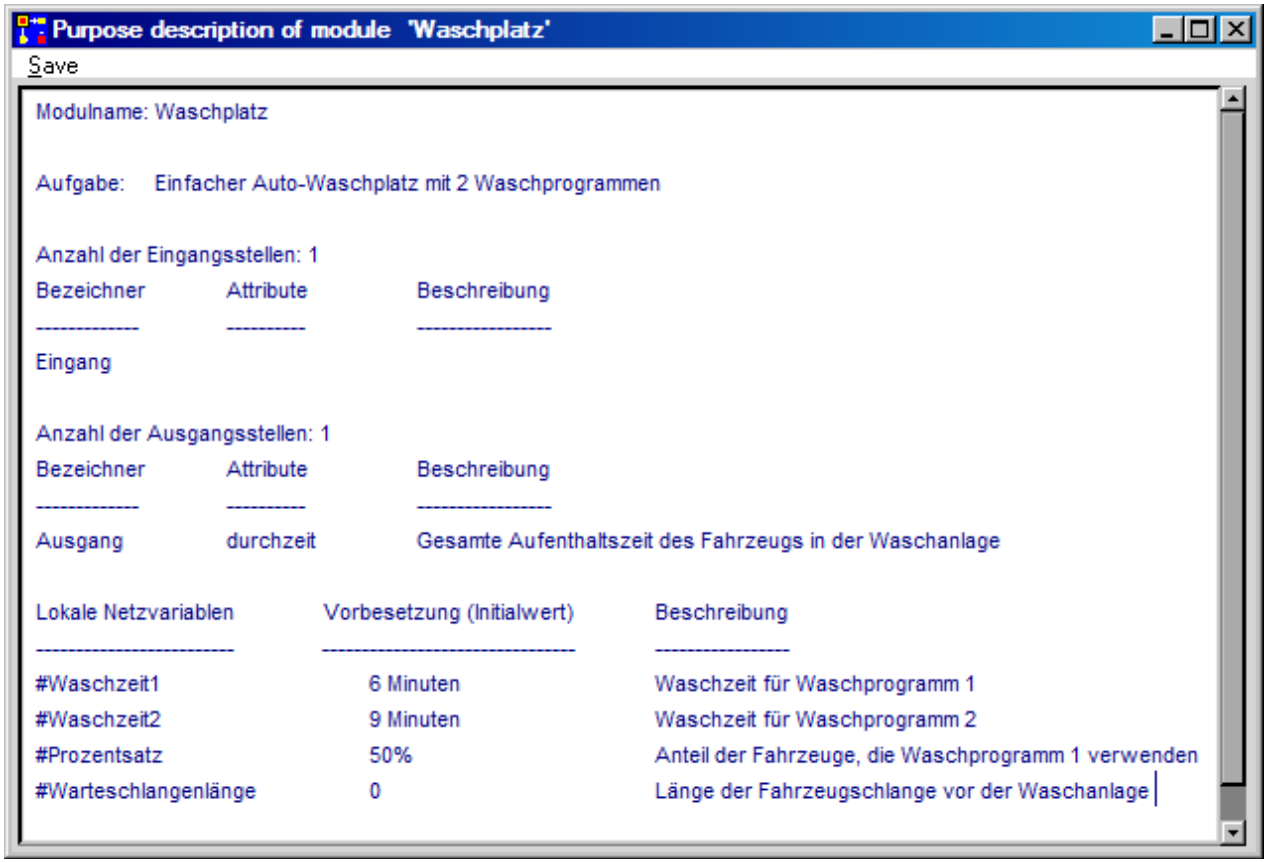



Abb. 4.10: Purpose Description des Moduls 'Waschplatz'

Für Modelle bietet PACE auch die Möglichkeit zur Erstellung einer Bedienungsanleitung, die fest mit dem Modell verbunden ist und wahlweise aufgerufen werden kann. Sie wird mit dem Menüpunkt **model user manual** im **Extras**-Menü der PACE-Programmleiste aufgerufen.

Damit ist der Modul 'Waschplatz' fertig gestellt (Abb. 4.8) und muss noch in einem Verzeichnis gespeichert werden. Standardmäßig ist dafür das Verzeichnis **modules** im PACE-Verzeichnis (das ist das Verzeichnis, aus dem heraus PACE gestartet wurde bzw. in dem das gestartete Image gespeichert ist) vorgesehen. Der Anwender kann aber auch ein eigenes Verzeichnis verwenden, um z.B. ein Paket von Bausteinen für einen bestimmten Anwendungsbereich bereitzustellen.

Im Fenster 'Net List' sind alle Module des Modells aufgelistet. Falls der Modul 'Waschplatz' noch nicht aufgeführt ist, mit der **li.MT** in das Fenster klicken. Dadurch wird das Fenster auf den neuesten Stand gebracht. Der Modul Waschplatz wird mit der **li.MT** markiert. Dann im File-Menü der PACE-Menüleiste den Menüpunkt **store module** aufrufen. Es erscheint das Standard-Fenster von Windows zum Abspeichern eines

Moduls mit dem voreingestellten Verzeichnis **modules**. Drückt man den Knopf **Speichern**, so wird der Modul unter dem Namen **Waschplatz.sub** im Verzeichnis **modules** des PACE-Verzeichnisses abgespeichert.

Danach kann PACE wie früher beschrieben beendet werden.

## 4.2 Verwenden von Bausteinen

Um zu zeigen wie ein Baustein in ein Modell eingesetzt wird, soll nachfolgend das Modell einer Straße erstellt werden, an der ein Auto-Waschplatz der modellierten Art liegt. Der Einfachheit halber sei angenommen, dass es sich bei der Straße um eine Einbahnstraße handelt, also der Fahrzeugfluss nur in einer Richtung betrachtet werden muss.

Begonnen wird wieder mit einem neuen Modell, das den Namen 'Straße mit Waschplatz' erhält. Die Straße mit Verzweigung zum Waschplatz ist in Abb. 4.11 dargestellt und kann leicht nachgezeichnet werden.

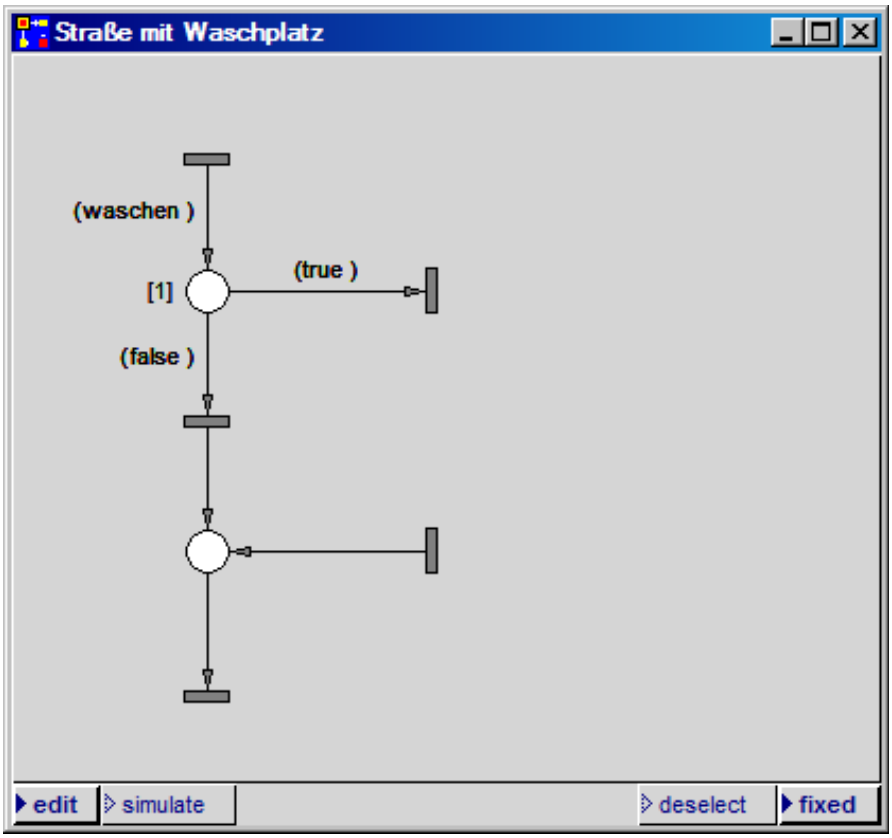



Abb. 4.11: Einbahnstraße mit Anschlusspunkten für einen Auto-Waschplatz

Der im letzten Abschnitt erzeugte Modul wird wie folgt eingesetzt: im No-Selection-Menü des Netzfensters wird der Menüpunkt **restore module** ausgeführt. Es öffnet sich ein Windows-Fenster zur Auswahl des Moduls. Eingestellt ist das Verzeichnis **modules** im PACE-Verzeichnis. Darin wird die Datei **Waschplatz.sub** selektiert und

danach der Öffnen-Knopf gedrückt. An der Mauszeiger-Position wird dann ein Rahmen angezeigt, der an die geplante Position des Moduls im Netzfenster geschoben

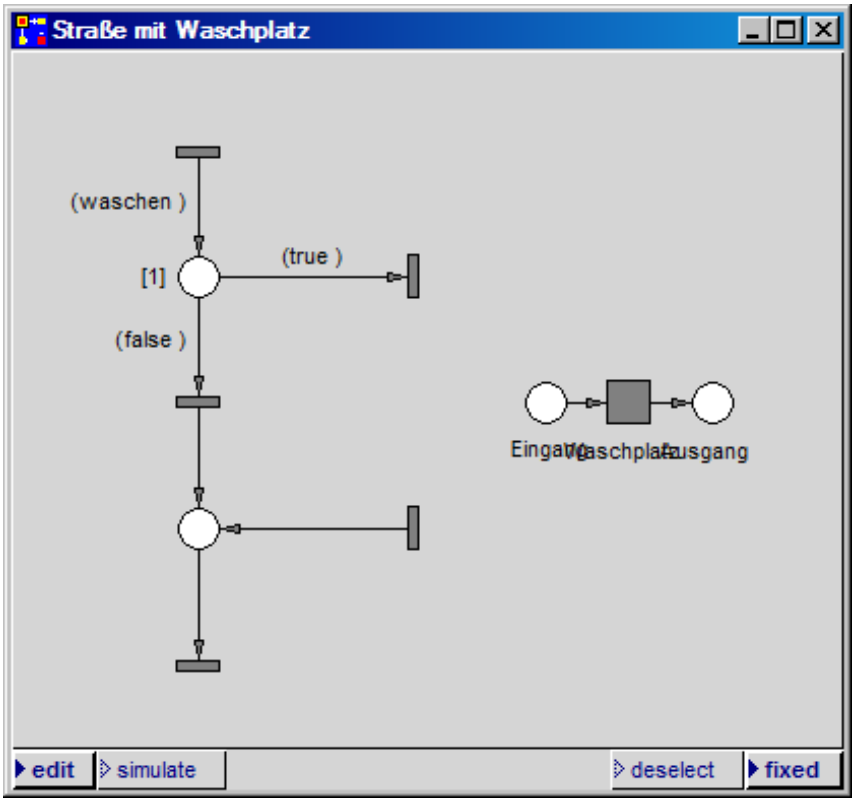



Abb. 4.12: Netzfenster nach Einsetzen des Moduls 'Waschplatz'

wird. Dort die **li.MT** drücken. Das Ergebnis zeigt Abb. 4.12.

Jetzt die einzelnen Netzelemente an die jeweils gewünschte Position schieben und die Schnittstellen (die Stellen 'Eingang' und 'Ausgang') mit dem Netz der Straße verbinden. Es entsteht des Netz in Abb. 4.13.

Um die Adaption des Moduls abzuschließen, sind die Modul- bzw. Netzvariablen wie folgt zu besetzen:

Waschzeit1 = 7
Waschzeit2 = 10
Prozentsatz = 60.

und der Konnektor von der Stelle 'Ausgang' zur anschließenden Transition zu inskribieren. Letzteres ist notwendig, damit Marken mit dem Attribut **durchzeit** über diesen Konnektor fließen können und nicht in der Stelle 'Ausgang' hängen bleiben. Die erforderlichen Maßnahmen dafür wurden früher schon beschrieben.

Abb. 4.14 zeigt das Netz mit allen Inskriptionen.

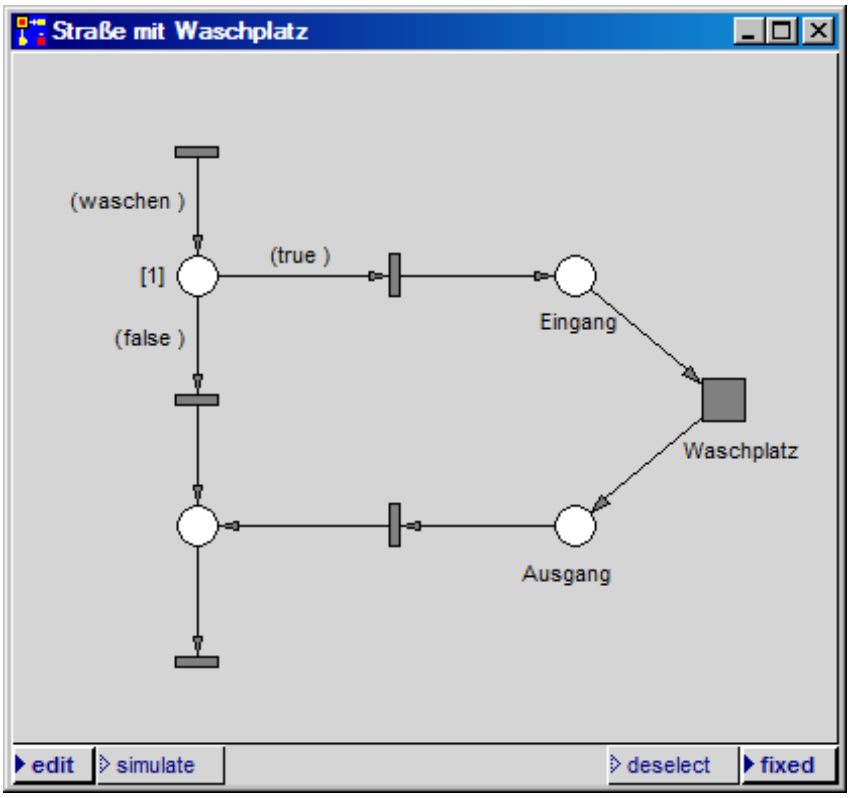



Abb. 4.13: Straße mit angekoppeltem Auto-Waschplatz

Der Delay-Code der obersten Transition besagt, dass im Mittel alle 12 Sekunden ein Fahrzeug in das modellierte Straßenstück einfährt. Im Action-Code wird festgelegt, welche Fahrzeuge gewaschen werden. Dazu wird eine Konnektorvariable 'waschen' eingeführt, welcher im Actioncode einer der Wahrheitswerte true oder false zugewiesen wird. Diese Werte werden in auslaufenden Marken als Argument gespeichert.

Abhängig vom Wert dieses Arguments, läuft eine Marke ausgehend von der angeschlossenen Stelle, entweder nach rechts (true) in die Waschanlage oder nach unten an der Waschanlage vorbei (false). Dabei wird von der Regel Gebrauch gemacht, dass eine Marke einen Konnektor nur passieren darf, wenn die Anzahl der Argumente der Marke und die Anzahl der Konnektorinskriptionen übereinstimmen. Falls Konnektorkonstanten (im vorliegenden Fall die Wahrheitswerte true und false) angegeben sind, müssen zugeordnete Argumente der Marke und der Konnektorinskription übereinstimmen. Wenn die Marke also den Wert true bzw. false trägt, kann sie nur den nach rechts bzw. nach unten führenden Konnektor passieren.

Der Actioncode besteht aus einer Zuweisung, deren rechte Seite aus zwei bedingten Ausdrücken besteht, die mit **and:** (logisches Und) verknüpft sind. Der erste Ausdruck:

RandomNumber next < 0.02

verwendet die System-Methode RandomNumber, mit der gleichverteilte Zufallszahlen zwischen 0 und 1 erzeugt werden können. Die Methode next liefert die nächste Zu-

fallszahl. Wenn diese kleiner als 0.02 ist, also im Mittel in 2 % aller Fälle, liefert der Ausdruck true, andernfalls false.

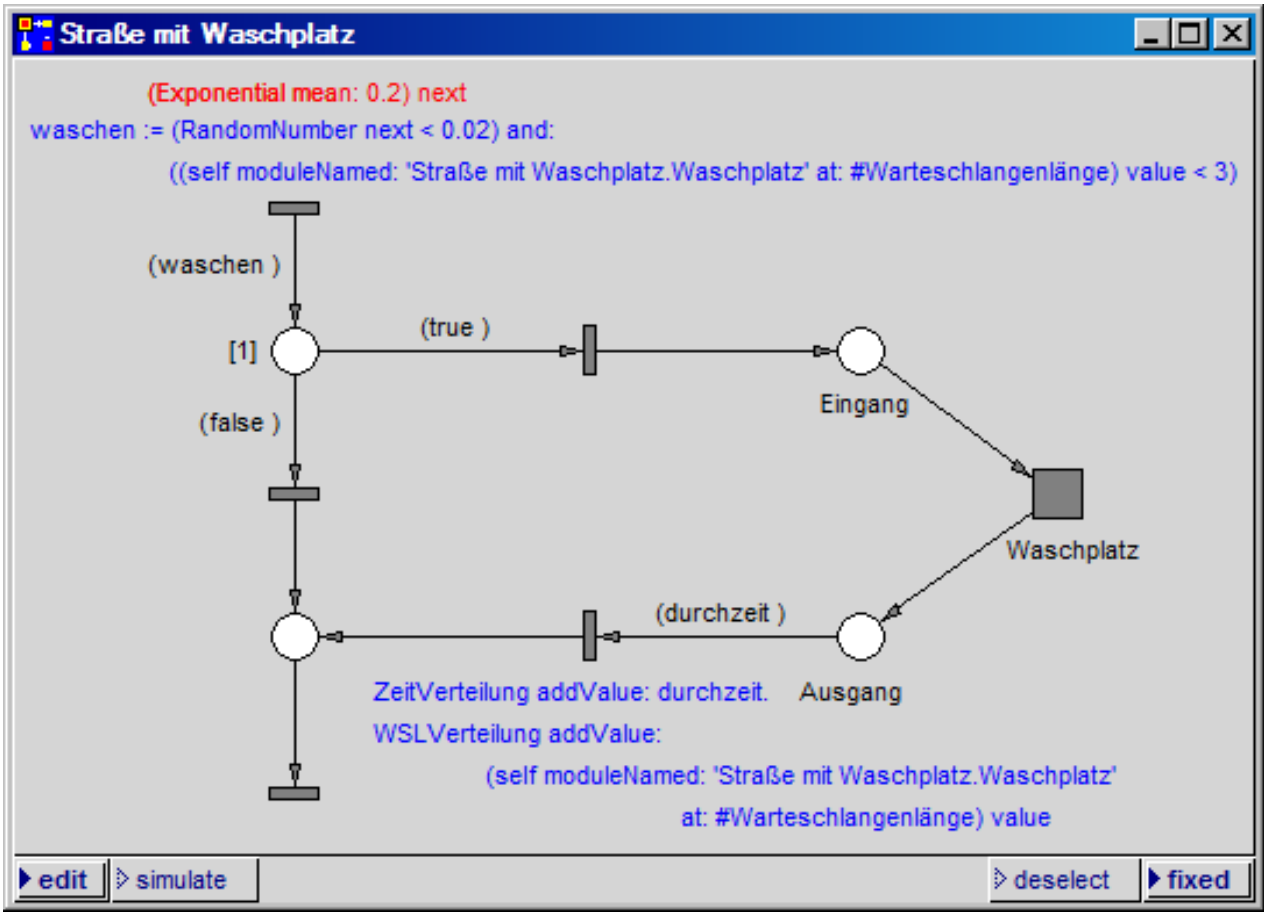



Abb. 4.14: Straße mit Waschplatz inskribiert

Mit dem zweiten Ausdruck wird auf die Netzvariable #Warteschlangenlänge im Modul 'Waschplatz' zugegriffen. Die Netzvariable (nicht deren Wert!) ergibt sich aus dem Ausdruck:

self moduleNamed: 'Straße mit Waschplatz.Waschplatz' at: #Warteschlangenlänge.

Der Modul wird dabei durch den String 'Straße mit Waschplatz.Waschplatz' identifiziert, in dem die Module, ausgehend von der Wurzel 'Straße mit Waschplatz' durch Punkte getrennt in der Reihenfolge angegeben sind, wie sie im Fenster 'Net List' hierarchisch angeordnet sind. Mit der Methode **value** wird der Wert der Netzvariablen ausgelesen. Ist er kleiner als 3, so liefert der Ausdruck true, andernfalls false. Fahrzeughalter fahren also den Waschplatz nur an, wenn weniger als 3 Fahrzeuge in der Warteschlange stehen.

Die Und-Verküpfung der Ausdrücke besagt dann: Wenn ein Fahrzeug zu den 2% gehört, die gewaschen werden, und wenn die Länge der Warteschlange vor dem Waschplatz kleiner als 3 ist, fährt das Fahrzeug zur Waschanlage hin.

Die Inskription der Transition, die mit der Stelle 'Ausgang' verbunden ist:

ZeitVerteilung addValue: durchzeit.

setzt voraus, dass ein Count-Histogramm erzeugt und im Initialisierungscode zugeordnet wurde:

```
ZeitVerteilung :=  CountHistogram named: 'Verteilung der Durchlaufzeiten'.
ZeitVerteilung clear
```

Wegen der geänderten Parameter wurde die Skalierung des Histogramms gegenüber früher geändert.

Wechselt man in den Simulationsmodus und führt das Netz als **background run** aus, so wird die in Abb. 4.15 dargestellte Verteilung der Durchlaufzeiten angezeigt.

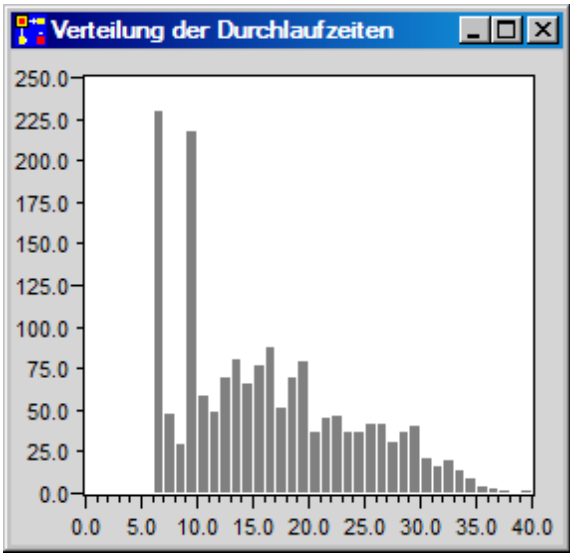



Abb. 4.15: Verteilung der Durchlaufzeiten

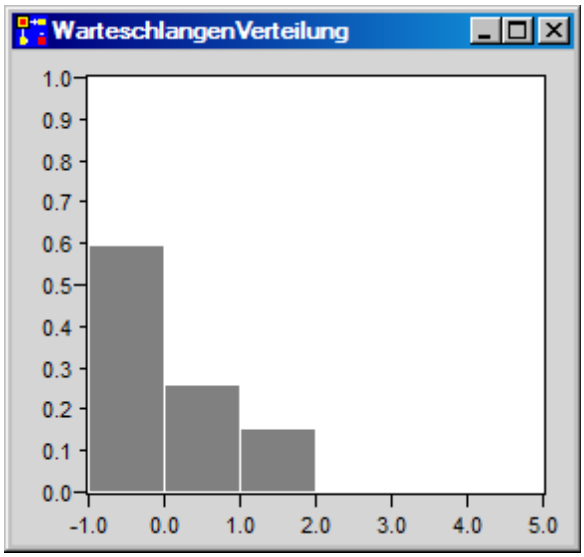



Abb. 4.16: Verteilung der Warteschlangen

Auch die Verteilung der Zeit auf die verschiedenen Warteschlangenlängen kann wie früher angezeigt werden. Erzeugt wird im View-Menü der PACE-Leiste ein Standard-Histogramm mit Namen: 'WarteschlangenVerteilung'. Der Initialisierungscode wird um die folgenden beiden Zeilen erweitert:

```
WSLVerteilung := StandardHistogram named: 'WarteschlangenVerteilung'.
WSLVerteilung clear.
```

Der Action-Code der Inskription, die mit der Stelle 'Ausgang' verbunden ist, wird um folgende Botschaft erweitert:

```
WSLVerteilung addValue: (self moduleNamed: 'Straße mit Waschplatz.Waschplatz'
                                at: #Warteschlangenlänge) value.
```

Führt man einen Simulationslauf aus, so erhält man das in Abb. 4.16 dargestellte Histogramm. Es zeigt, dass die Warteschlange etwa 60% der Zeit leer ist, dass während eines Viertel der Zeit ein Auto auf das Waschen wartet, usw.

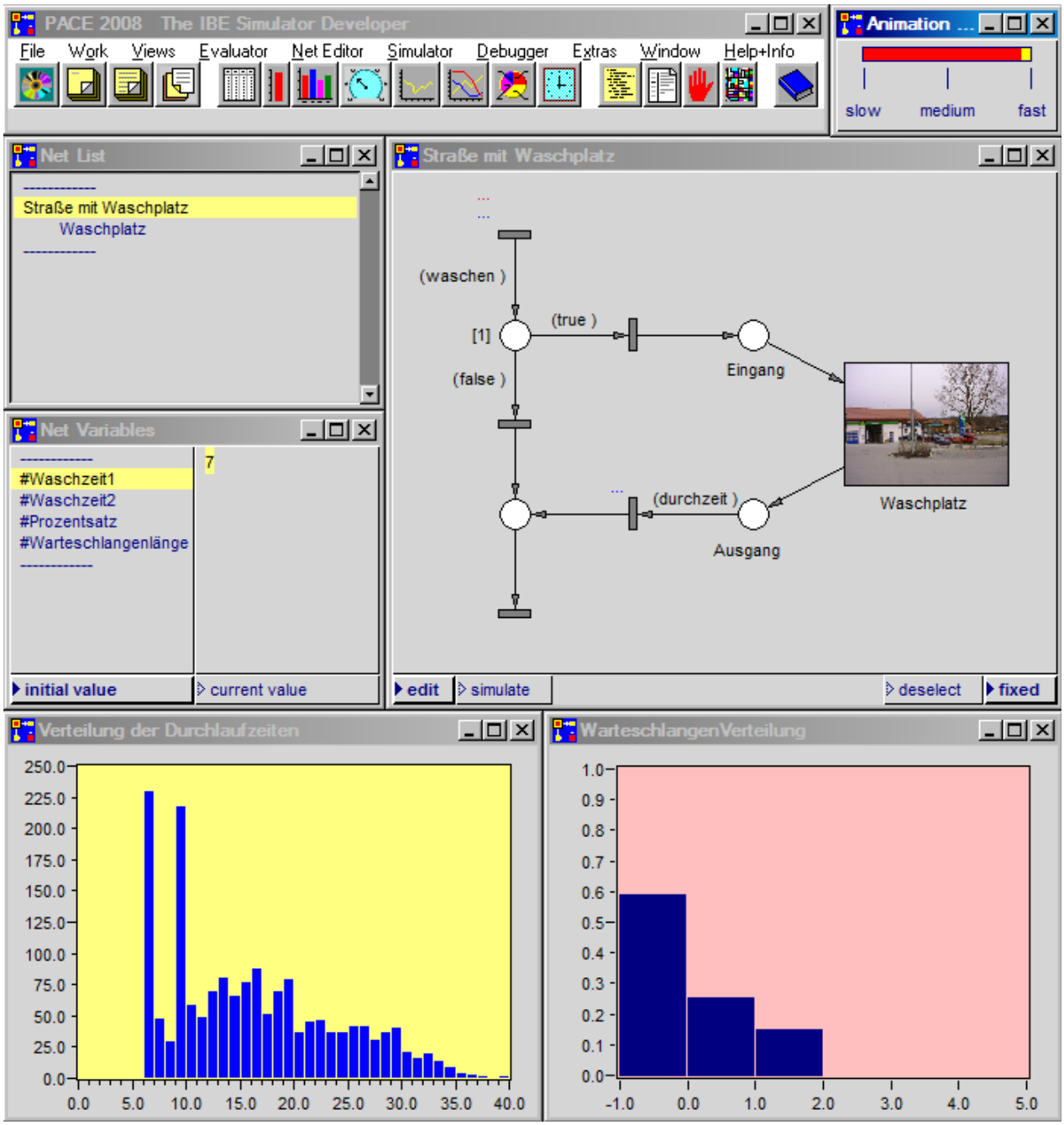



Abb. 4.17: Arbeits-Oberfläche für das Experimentieren

Abb. 4.17 zeigt eine Arbeits-Oberfläche des Simulationsmodells für die Durchführung von Experimenten. Man erhält sie durch Anordnung der PACE-Fenster. Das rechts oben dargestellte Fenster, mit dem die Animationsgeschwindigkeit eingestellt werden kann, wird mit dem Menüpunkt **animation speed** im Simulator_Menü der PACE-Menüleiste geöffnet.

Damit ist das Modell fertig gestellt und kann wie früher beschrieben gesichert werden.

Festzuhalten ist, dass für die Erstellung des Modells die Interface-Beschreibung des Moduls 'Waschplatz' ausreichend war. Das ist die Voraussetzung für die Bereitstellung von Baustein-Bibliotheken und für den Aufbau von Simulationsmodellen aus vorgefertigten Bausteinen.

# 5. Einige weitere nützliche Eigenschaften

Im Folgenden werden einige nützliche Eigenschaften von PACE kurz angesprochen, um sie bekannt zu machen. Sie sind im PACE-Handbuch 'Modellierung und Simulation' ausführlich beschrieben.

## 5.1 Definition und Veränderung der Standard Darstellung von Netzbestandteilen

Die Standard-Ikonen können für jedes Netz gesondert geändert werden. Die Änderungen erfolgen über **'Extras'-Menü, 'icons',** '**default icons**'. Mit '**change**' können die vordefinierten Ikonen durch beliebige andere vordefinierte Ikonen oder durch Ikonen ersetzt werden, die in der Liste der individuellen Ikonen gespeichert sind. '**scale**' verändert die Größe der vordefinierten Ikonen.

Es ist auch möglich, die Standard-Darstellung einzelner Netzelemente zu skalieren. Dazu im **re.MT**-Menü des Netzelements den Menüpunkt **scale** wählen und in das sich öffnende Fenster den Skalierungsfaktor eingeben. Dann die **return**-Taste drücken oder im **re.MT**-Menü des Eingabefensters **accept** wählen.

Weitere Änderungen der Bestandteile eines Netzes kann man mit dem Menü **options** im **Net Editor**-Menü der PACE-Hauptleiste durchführen.

So kann die Größe der Pfeilspitzen eines Konnektors verändert werden. Dies wird im **options**-Menü über die Menüpunkte **arrow length** und **arrow angle** ausgeführt. **element size** verändert die absolute Größe aller Elemente (mit Ausnahme der ‘individual icons’) im Netz. Schließlich kann mit **grid** die Auflösung der Netzfenster erhöht werden. Damit wird eine genauere Positionierung von Netzelementen möglich.

## 5.2 Farbige Netze

*PACE* bietet die Möglichkeit, die Fenster- und Netz-Komponenten (Ikonen, Texte, usw.) in verschiedenen Farben darzustellen. Die zur Verfügung stehenden Farben kann man mit der Funktion **'show default platform colors'** unter dem Menüpunkt **colors** im **'view'-Menü** der PACE-Menüleiste ansehen.

Die Funktion zum Verändern der Farben kann über das **view'-Menü , colors, net constituents colors** aufgerufen werden. Es öffnet sich ein Fenster, in dem die wichtigsten Bestandteile von PACE-Netzen aufgelistet sind (Abb. 5.1). Rechts von der Bezeichnung der Bestandteile ist jeweils die aktuelle Farbe des Bestandteils angezeigt. Durch ‘**change**’ können die Farben verändert werden. Es erscheint ein Auswahlmenü für die einzelnen Farben. Hier die entsprechende Farbe markieren und den ok-Knopf drücken. Über einen Scrollbar am rechten Fensterrand kann zu den nachfolgenden Farben gescrollt werden. Die neuen Farbeinstellungen werden im Modell bzw. dem sog. Image gespeichert, d.h. nach dem Verändern muss das Modell (Image) mit **store image** gesichert werden.

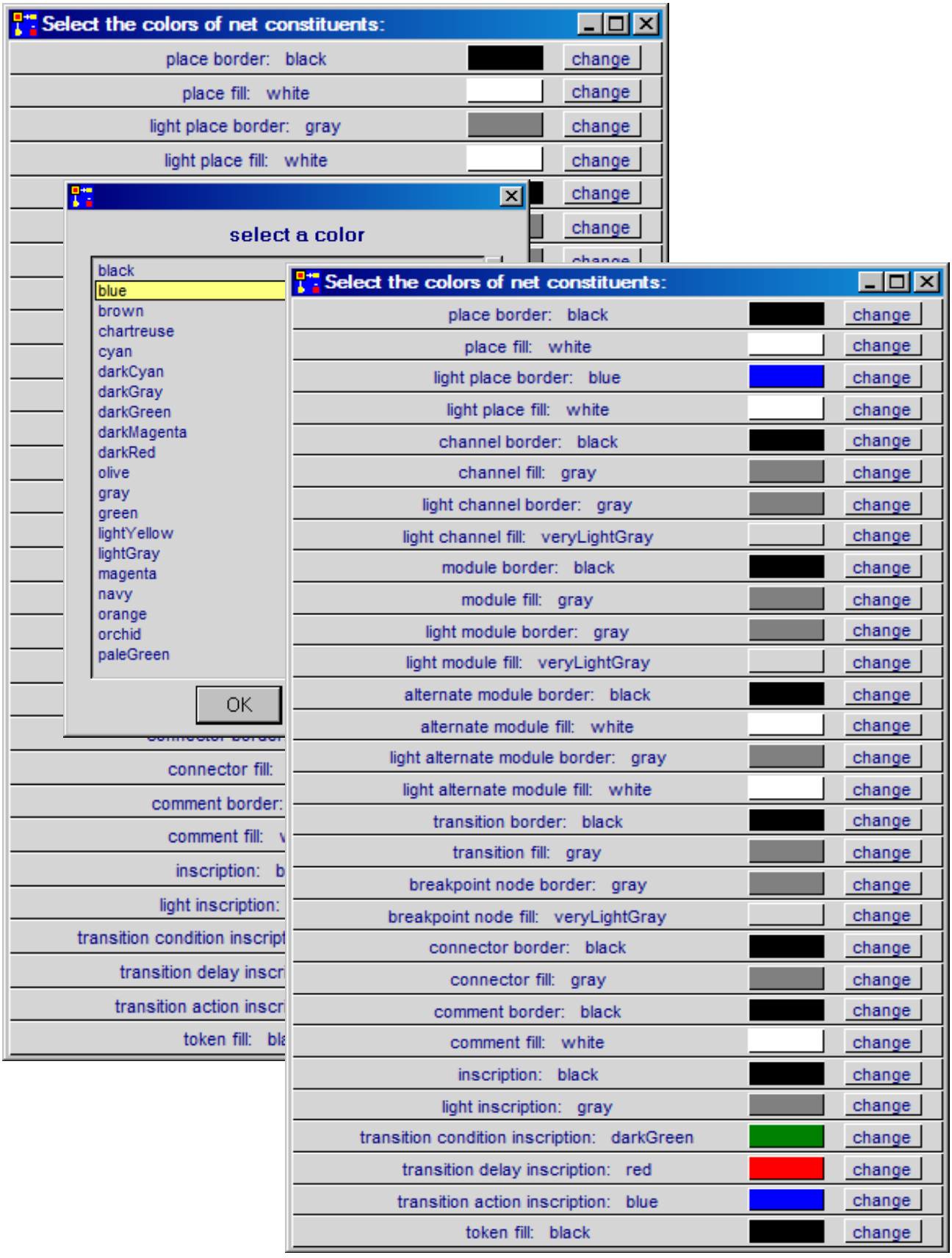



Abb. 5.1: Ändern der Farbe eines Netz-Bestandteils

## 5.3 Initialization code, break code, continue code und termination code

Diese Funktionen werden im **Net Editor**-Menü mit dem Menüpunkt **extra codes** aufgerufen. Hier kann Smalltalk-Code eingegeben werden, der bei der Initialsierung (**initialization code**), beim Unterbrechen (**break code**), beim Fortsetzen (**continuation code**) oder beim Beenden (**termination code**) eines Simulationslaufs ausgeführt werden soll.

## 5.4 Spezialleisten für die Simulation; Vergießen von Modellen

Um die Bedienung von fertigen Modellen zu erleichtern, die zum Beispiel von Anwendern ohne PACE-Kenntnisse im täglichen Betrieb verwendet werden, gibt es 8 Spezialleisten (Exekutiven), die man einem Modell zufügen kann (siehe z.B. 5.2). Die Leisten enthalten die Steuerfunktionen zur Durchführung von Simulationsläufen, sodass mit den Modellen ohne Menütechnik gearbeitet werden kann.

Für das Öffnen einer Exekutive im **Simulator**-Menü den Menüpunkt **install executive** wählen. Danach entweder eine horizontale oder vertikale Exekutive durch Auswahl eines der Menüpunkte **horizonal layout** oder **vertical layout** auswählen. Danach eine der vier möglichen Exekutiven wählen.


Abb. 5.2: Kurze horizontale Exekutive

Um die versehentliche Änderung von Modellen durch ungeübte Anwender zu verhindern, können Modelle mit einer hinzugefügten Exekutiven "vergossen" oder "eingefroren" werden. Beim Vergießen werden die PACE-Hauptleiste und die Netzliste entfernt, so dass keine Änderung oder Erweiterung des Modells mehr möglich ist. Weiter lassen sich die meisten Menüs der **re-MT** nicht mehr anzeigen und alle beim Vergießen schon vorhandenen Fenster können nicht geschlossen werden. Dagegen werden Daten weiterhin, z.B. über Dateneingabefenster, durch Selektion mit der **li.MT** oder Texteingaben bereitgestellt. Auf diese Weise wird ein einfach verwendbares Anwendungsmodell erzeugt, das nur über eine Exekutive und Eingabefenster steuerbar ist.

## 5.5 Szenen

Bei der Erstellung größerer Modelle ist die Oberfläche eines Bildschirms fast immer zu klein, um alle benötigten Fenster (Netzfenster, Exekutiven, grafische Ein/Ausgabefenster, usw.) zusammen anzeigen zu können. Normalerweise werden aber nicht alle Fenster gleichzeitig benötigt. Man kann in PACE deshalb die Fenster, die gleichzeitig angezeigt werden sollen, in sog. Szenen zusammenfassen und auf Knopfdruck (**li.MT**) gemeinsam anzeigen oder verbergen. Auf diese Weise lässt sich eine sachgerechte übersichtlich Benutzeroberfläche bereitstellen.

Die Zusammenstellung von Szenen ist denkbar einfach. Man öffnet im **Extra**-Menü der PACE-Hauptleiste mit dem Menüpunkt **define scenery** das Definitionsfenster für Szenen. Es besteht aus zwei Teilfenstern. Mit dem No-Selection-Menü des linken Teilfensters können Namen für Szenen eingegeben werden (Menüpunkt **add**). Wird ein Szenennamen im linken Teilfenster markiert, so kann man im rechten Teilfenster mit dem No-Selection-Menüpunkt **add** den Namen eines Fensters eingeben, das zur Szene gehören soll.

Dabei empfiehlt sich, insbesondere bei längeren Fensternamen, nicht das Abschreiben des Fensternamens, sondern folgende Vorgehensweise: man aktiviert das Fenster, das in der Szene angezeigt werden soll und ruft mit der **mi.MT** dessen Systemmenü auf. Darin den Menüpunkt **relable as…** aufrufen. Es öffnet sich ein Eingabefenster für den neuen Fensternamen, in dem der aktuelle Fenstername markiert ist. Jetzt das **re.MT**-Menü des Eingabefensters anzeigen, darin **copy** aufrufen und zum Schluss den **ok**-Knopf drücken. Danach das No-Selection-Menü des rechten Teilfensters von **define scenery** aufrufen und **add** wählen. In dem sich öffnenden Eingabefenster für den Fensternamen das Menü der **re.MT** aufrufen und **paste** ausführen. Schließlich das Eingabefenster mit der **Return**-Taste der Tastatur schließen.

Hat man alle Szenen zusammengestellt, so wird das **define scenery**-Fenster geschlossen und mit dem Menüpunkt **select scenery** des **Extra**-Menüs das Auswahlfenster für Szenen geöffnet. Darin sind die Namen aller Szenen aufgelistet. Selektiert bzw. deselektiert man einen Namen in diesem Fenster so wird die zugeordnete Szene ein- bzw. ausgeblendet.

Beim Design von Oberflächen geht man normalerweise so vor, dass man einen permanenten Basissatz von Fenstern anzeigt, der fallweise von Szenen überlagert wird. Zu bemerken ist, dass Szenen auch während der Simulation per Programm angezeigt und wieder verborgen werden können. Das ist insbesondere nützlich, wenn während der Simulation verschiedene grafische Auswertungsfenster angezeigt werden sollen oder Parameter einzugeben sind.

## 5.6 Modell-Lexikon

Bei der Weiterentwicklung oder dem Nachvollziehen von Modellen ist oft sehr hinderlich, dass die genaue Bedeutung von Programmgrößen erst mühsam aus dem Kontext entnommen werden muss. Das nachfolgend beschriebene Modell-Lexikon gibt dem Entwickler eines Modells eine bequeme Möglichkeit, diese Information während

der Entwicklung des Model aufzuzeichnen und nach der Fertigstellung des Modells zusammen mit dem Modell bereitzustellen.1

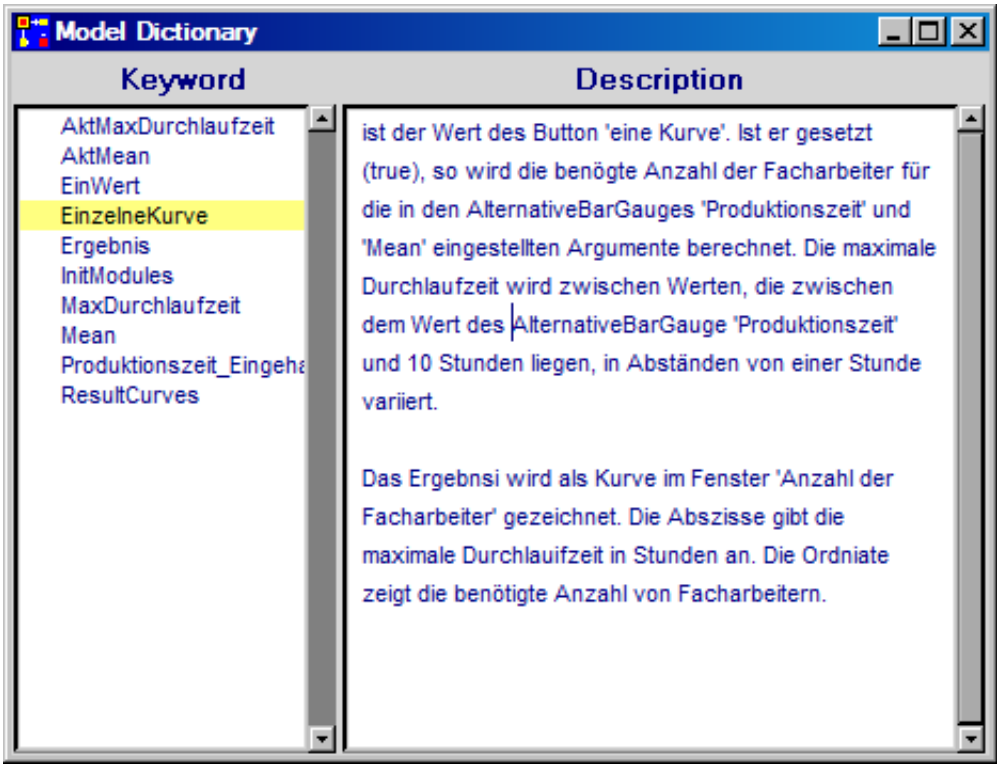



Abb. 5.3: Modell-Lexikon mit Einträgen

Das Modell-Lexikon kann über das Net-Editor-Menü, Menüpunkt **model dictionary,** der PACE-Hauptleiste oder durch Anklicken des 4. Ikon von rechts der unter der Hauptleiste angeordneten Ikonenliste geöffnet werden. Es wird auch geöffnet, wenn ein Text in einer Inskription markiert und dann im Menü der Inskription (**re.MT**) der Menüpunkt **dictionary** angewählt wird. In diesem Fall trägt PACE den markierten Text als Entry in das Lexikon ein und markiert den Entry. Der Anwender muss dann im rechten Teilfenster (siehe Abb. 5.3) den beschreibenden Text eingeben. Die Eingabe wird in das Lexikon übernommen, wenn die Markierung des Entry (durch Anklicken des Entry mit der **li.MT**) zurückgesetzt wird.

1 Weitere in PACE vorgesehene Möglichkeiten zur Dokumentation von Modellen wurden an Ende von Abschnitt 4.1 beschrieben.

# 6. Optimierung

## 6.1 Optimierungsverfahren in PACE

Bei der Optimierung von Modellen ist zwischen Netzoptimierung und Parameteroptimierung zu unterscheiden.

Die Netzoptimierung vergleicht die mit verschieden gestalteten Netzen erzielten Ergebnisse miteinander. Daraus werden Vorschlage für die Planung oder Umgestaltung von Prozessen abgeleitet. Im Allgemeinen können keine automatischen Verfahren für die Netzoptimierung eingesetzt werden, weil bei jeder Modellvarianten die Machbarkeit in der Realität berücksichtigt werden muss, die nur von einem Prozessingenieur beurteilt werden kann. Ob es sich bei einer gefundenen "optimalen" Lösung wirklich um das Optimum handelt, ist nicht bekannt. Deshalb wird die "Netzoptimierung" oft auch als "Netzverbesserung" bezeichnet.

Im Gegensatz dazu können für die Parameteroptimierung von Netzen automatische Verfahren bereitgestellt werden. Sie werden auch für den Vergleich von verschiedenen Alternativen bei der oben genannten Netzoptimierung benötigt. Bei der Parameteroptimierung (im Folgenden kurz: Optimierung) wird ein Netz mit fest vorgegebenen Prozessbahnen manuell oder automatisch mit unterschiedlichen Parametern (z.B. Anzahl von Ressourcen) ausgeführt, um die Parameterkombination zu finden, für die bestimmte Ergebnisse (z.B. Kosten, Durchlaufzeit, usw.) optimal werden.

In PACE stehen für die Optimierung von Netzen grundsätzlich drei Verfahren zur Auswahl:

1. Wiederholte Ausführung des gesamten Modells
   Wiederholte Ausführung eines Modells mit jeweils geänderten Parametern

2. Automatische Wiederholung von Teilmodellen
   Wiederholung von Teilmodellen mit per Programm weitergeschalteten Parametern und Auflisten der Ergebnisse und/oder grafischer Darstellung der Ergebnisse. Aus der grafischen Darstellung kann das Optimum visuell abgelesen werden (sog. grafische oder visuelle Optimierung).

3. Mathematische Optimierung von Modellen
   Einsatz mathematischer Optimierungsverfahren (z.B. Aufstiegsverfahren, Simplex, genetische Verfahren, Schwellenwert-Akzeptanz) zur automatischen Bestimmung des Optimums eines Modells.

Welches der drei Verfahren im Einzelfall zu wählen ist, kann erst bei der Betrachtung der jeweils anliegenden Aufgabenstellung entschieden werden. Im Allgemeinen wird man bei mehrdimensionalen Aufgabenstellungen zu mathematischen Optimierungsverfahren greifen, um die Anzahl der Modellausführungen möglichst klein zu halten. Bevor auf die Optimierung von Netzen eingegangen wird, zeigt der nächste Abschnitt die direkte Verwendung eines PACE-Optimierungsverfahren im Programmcode.

## 6.2 Optimierung von mathematischen Funktionen

In den nachfolgenden Abschnitten werden alle drei in PACE verfügbaren Optimierungsverfahren für Netze an einem Beispiel erläutert, das wieder so einfach gewählt wurde, dass bei der Beschreibung die vorzustellenden Verfahren und nicht das Beispiel im Vordergrund stehen. Als Beispiel wird die Bestimmung des Maximum des Sinus im Intervall [0, π] gewählt.

In diesem Fall kann das Optimum auch ohne Netzmodellierung z.B. mit einem Smalltalk-Programmstück gefunden werden. Da es gelegentlich zweckmäßig ist, Smalltalk-Code vor seiner Verwendung in Netz-Inskriptionen in einem sog. Workspace auszuprobieren, wird in diesem Abschnitt übungshalber erläutert, wie man dabei vorzugehen hat.

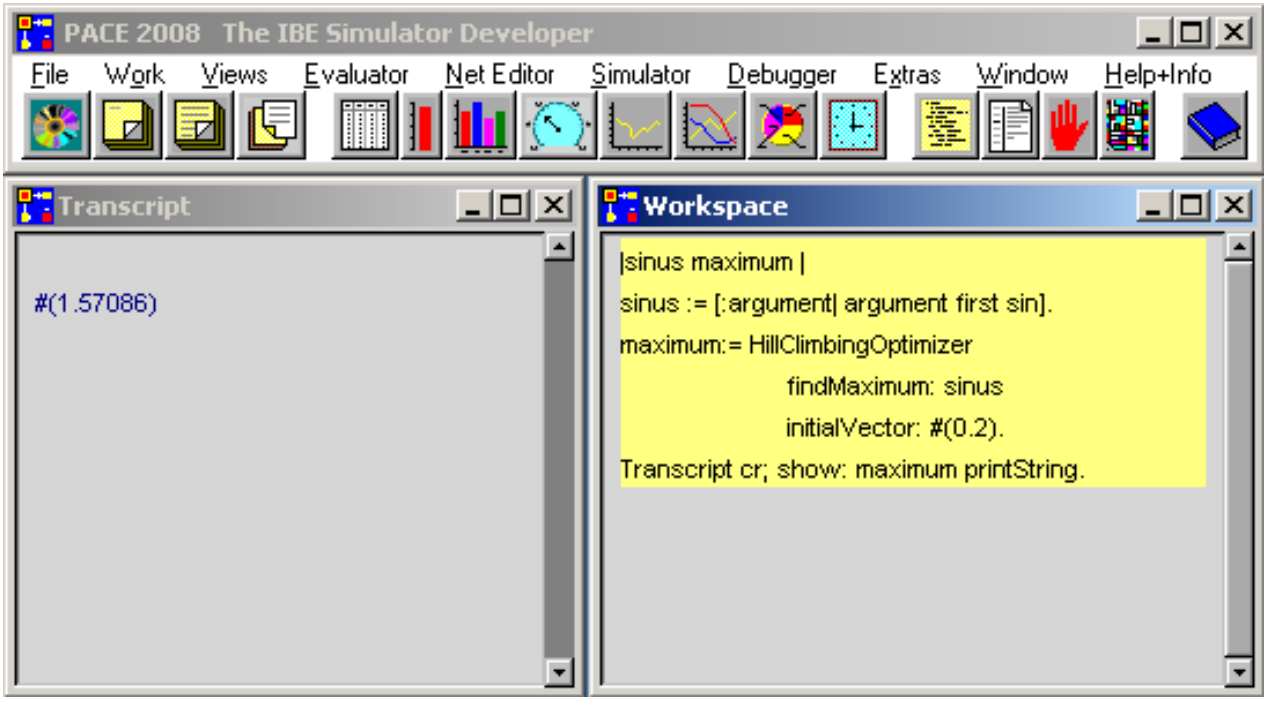



Abb. 6.1: Mathematische Optimierung in PACE

Im Work-Menü der PACE-Hauptleiste werden hintereinander die Menüpunkte **transcript** und **workspace** selektiert und dadurch zwei Fenster, ein Transcript- und ein Workspace-Fenster (kurz ein "Workspace") aufgemacht. Die Fenster kann man z.B. so wie in Abb. 6.1 anordnen.

In das Workspace-Fenster wird der in Abb. 6.1 markierte Code eingetragen. In der ersten Zeile werden zwischen den beiden senkrechten Strichen die Variablen **sinus** und **maximum** vereinbart. Die zweite Zeile enthält die Definition eines sog. Smalltalk-Blocks:

[:argument| argument first sin ].

Er wird mit einer öffnenden eckigen Klammer "[" begonnen und mit einer schließenden eckigen Klammer "]" beendet. Nach dem Doppelpunkt wird das Argument des Blocks

angegeben (es wird in anderen Programmiersprachen als "formaler Parameter" bezeichnet). Gibt es mehrere Argumente, so werden deren Namen jeweils nach einem Zwischenraum und einem Doppelpunkt eingegeben. Das Ende der Argumentliste wird durch einen senkrechten Strich festgelegt.

Danach kommt der Code des Blocks, im vorliegenden Fall nur eine Zeile. Da die Optimierungsverfahren die Daten in Form eines Arrays bereitstellen, ist das Argument ein Array. Deshalb muss zu Beginn des Blockcodes das erste Element des Arguments berechnet werden. Das geschieht mit dem Ausdruck **argument first**; man hätte auch weniger elegant **(argument at: 1)** schreiben können. Auf das Element wird dann der Sinus **sin** angewendet. Der Block wird danach der lokalen Variablen **sinus** zugewiesen.

In den nächsten drei Zeilen wird eines der in PACE vorgesehenen mathematischen Optimierungsverfahren, der sog. HillClimbingOptimizer, aufgerufen. Das im HillCllimbingOptimizer verwendete Verfahren wird auch als Aufstiegsverfahren bezeichnet. Als Berechnungsvorschrift für die Optimierung wird der oben beschriebene Block **sinus** angegeben. Danach wird in der fünften Zeile ein Anfangswert 0.2, bei dem die Berechnung des Optimums starten soll, in Form eines initialisierten Arrays **#(0.2)** angegeben. Sind mehrere Argumente anzugeben, so werden diese innerhalb der runden Klammern durch Zwischenräume getrennt aufgelistet. Man hätte stattdessen auch weniger elegant **Array with: 0.2** schreiben können.

Der HillClimbingOptimizer berechnet den Maximalwert des Blocks **sinus** und weist das zugehörige Argument in Form eines Arrays der Variablen **maximum** zu. In der letzten Zeile des Codestücks wird dieser Wert dann in das Transcript-Fenster geschrieben.

Um das Codestück auszuführen wird es durch Überstreichen mit gedrückter **li.MT** selektiert. Eleganter kann ein Codestück selektiert/deselektiert werden, indem man hinter der letzten Zeile in den freien Raum eines Fensters klickt. Der markierte Code wird durch das Markieren gelb unterlegt. Danach mit der **re.MT** das Menü des Fensters aufrufen und den Menüpunkt **do it** auswählen.

## *6.3 Wiederholte Ausführung eines gesamten Modells*

Bei der Ausführung eines PACE-Modells wird vor dem Netz zunächst der sog. Initialisierungscode ausgeführt. Das hier zu schildernde Verfahren beruht darauf, dass im Initialisierungscode unterschieden werden kann, ob es sich während eines Simulationslaufs um den ersten Durchlauf des Modells nach der manuellen Initialisierung oder um eine programmgesteuerte Wiederholung handelt. Diese Unterscheidung ermöglicht die Initialisierung von Programmgrößen beim ersten Durchlauf und deren Fortschreibung bei den nachfolgenden Durchläufen.

Die Abfrage, ob es sich um die erste oder um eine weitere Ausführung des Modells handelt, wird mit der Botschaft:

self isRestarted

durchgeführt. Bei der ersten Ausführung des Modells direkt nach seiner Initialisierung durch den Anwender liefert diese Abfrage den Wahrheitswert **false**, bei den nachfolgenden Ausführungen den Wert **true**.

Die Ausführung eines Modell bzw. Simulationslaufs wird automatisch beendet, wenn keine Transition mehr feuern kann. Sie kann auch in einer Netz-Inskription mit der Botschaft

self terminate.

erzwungen werden. Wird anstelle der terminate-Botschaft die Botschaft:

self restart.

verwendet, so wird ein erneuter Durchlauf des Modells gestartet.

Begonnen wird wieder mit einem neuen Modell, welches "Beispiel 1" heißen soll. Zum Anzeigen der Ergebnisse werden ein Message-Fenster und ein MultipleCurve-Fenster geöffnet und skaliert.

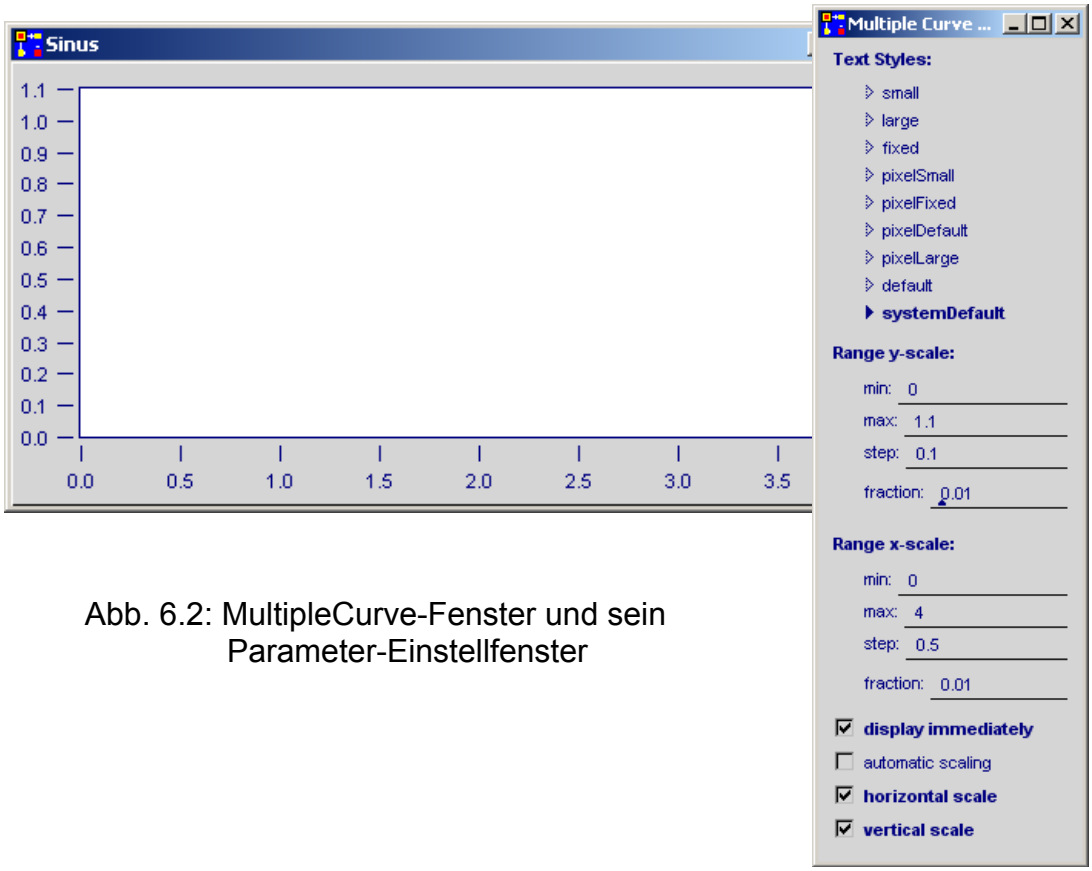



Abb. 6.2: MultipleCurve-Fenster und sein Parameter-Einstellfenster

Das Message-Fenster wird mit dem Menüpunkt **message window** im View_Menü der PACE-Hauptleiste geöffnet. Es öffnet sich ein Eingabefenster für den Namen des Message-Windows, in das der Name **Maximalwert** eingetragen wird. Danach die **Return**-Taste betätigen. Das Message-Fenster wird jetzt geöffnet und ist, wie früher be-

schrieben, aufzuziehen.

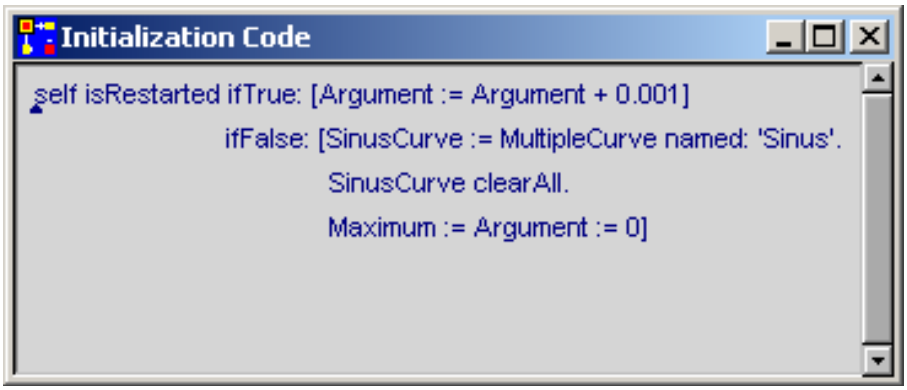



Abb. 6.3: Initialisierungscode von Beispiel 1

Das MulitpleCurve-Fenster kann über das View-Menü oder über den Ikon-Button mit mehreren Kurven der PACE-Hauptleiste (8. Ikon von rechts) geöffnet werden. Nach der Öffnung wird mit dem Menü der **mi.MT**, wie früher beschrieben, der Fenstername **Sinus** eingetragen und mit der **re.MT** das Menü des Fensters aufgerufen. Darin wird der Menüpunkt **parameter** ausgewählt. Es öffnet sich ein Parameter-Fenster, mit dem unter anderem die Skalierung des Fensters festgelegt werden kann. Die Parameter werden so wie in Abb. 6.2 dargestellt festgelegt. Nach der Eingabe wird die Return-Taste gedrückt, wodurch die Skalierung des Fensters neu eingestellt wird.

Als nächstes wird der Initialisierungscode eingetragen. Dazu das zugeordnete Ikon der PACE-Hauptleiste anklicken (5. Ikon von rechts) und den Code wie in Abb. 6.3 eintragen.

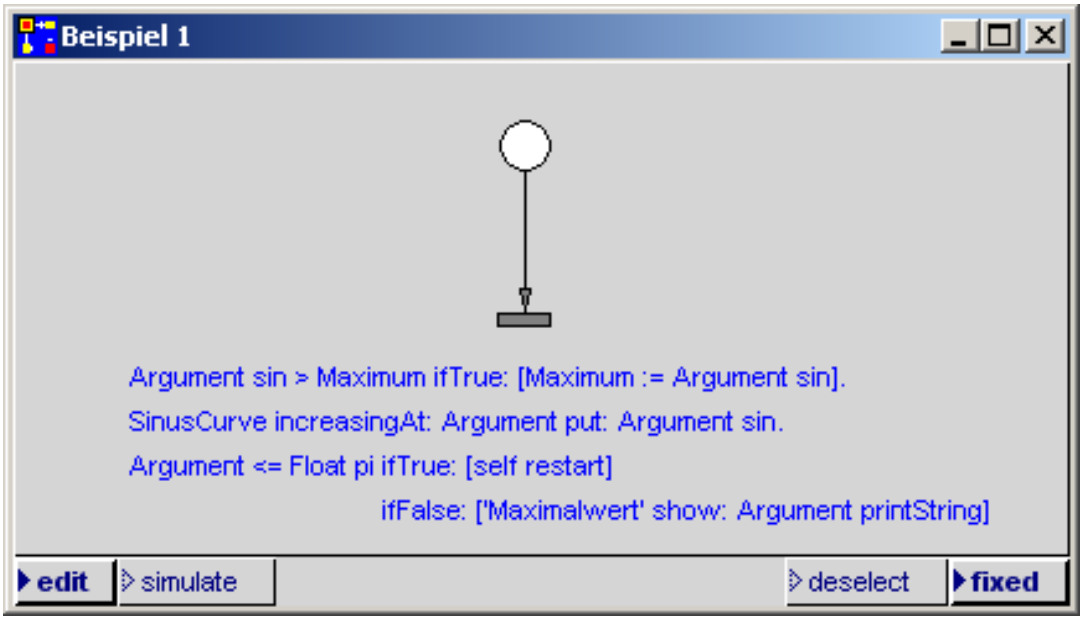



Abb. 6.4: Netz von Beispiel 1

Der Initialisierungscode besteht aus einer "bedingten Botschaft". Je nachdem, ob die isRestarted-Botschaft true oder false liefert, wird der in eckige Klammern eingeschlos-

sene zugeordnete Block ausgeführt. Im True-Zweig der Botschaft wird das Argument für den nächsten Durchlauf des Modells erhöht. Der false-Zweig führt die Initialisierung der globalen Variablen **Argument**, **Maximum** und **SinusCurve** durch. Mit den ersten beiden Zeilen des false-Zweigs wird das eben erzeugte MulitpleCurve-Fenster an das Modell angeschlossen und ein ggf. von früheren Simulationen herrührender Inhalt gelöscht.

Nach dem Eintragen des Programmcodes wird mit der **re.MT** das Fenstermenü des Textfensters aufgerufen und **accept** gewählt. Es erscheinen nacheinander drei Abfragefenster, in denen festzulegen ist, dass es sich bei den oben genannten Variablen um globale Variable handelt.

Das Netz des Beispiels ist denkbar einfach und in Abb. 6.4 abgebildet. In die Stelle ist eine sog. Initialmarke einzubringen, die bei der Initialisierung erzeugt wird. Das geht wie folgt:

Mit der **re.MT** das Menü der Stelle aufrufen und **initial tokens** auswählen. Angezeigt wird daraufhin das in Abb. 6.5 dargestellte Fenster für die Vereinbarung von Anfangsmarken.

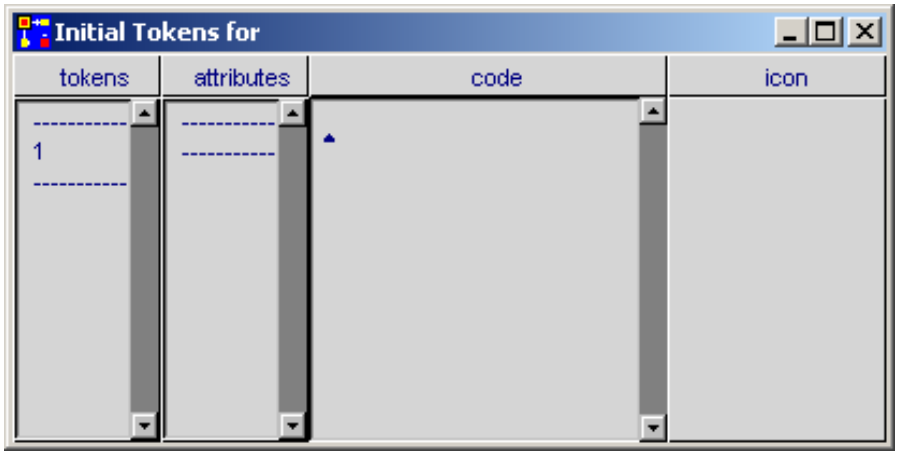



Abb. 6.5: Fenster zur Vereinbarung von Anfangsmarken

Mit ihm kann man beliebig viele Anfangsmarken mit Attributen und Anfangs-Ikonen festlegen. Im vorliegenden Fall wird nur eine Marke ohne Attribute benötigt. Deshalb im linken Teilfenster (tokens) im Menü der **re.MT** den Menüpunkt **add** wählen. Das Fenster sieht jetzt so wie in Abb. 6.5 dargestellt aus und kann geschlossen werden.

Das Modell ist jetzt fertiggestellt und kann, wie früher beschrieben, ausgeführt werden.

Es ist immer zweckmäßig, die bei der Simulation anzuzeigenden Fenster zu einer Arbeits-Oberfläche zusammenzufassen. Speichert und verlässt man das Modell (**leave PACE** im File-Menü), so wird es beim erneuten Laden genau wieder so angezeigt, wie es vor dem Verlassen vorlag. Im vorliegenden Fall kann man z.B. die in Abb. 6.6 dargestellte Arbeitsoberfläche zusammenstellen.

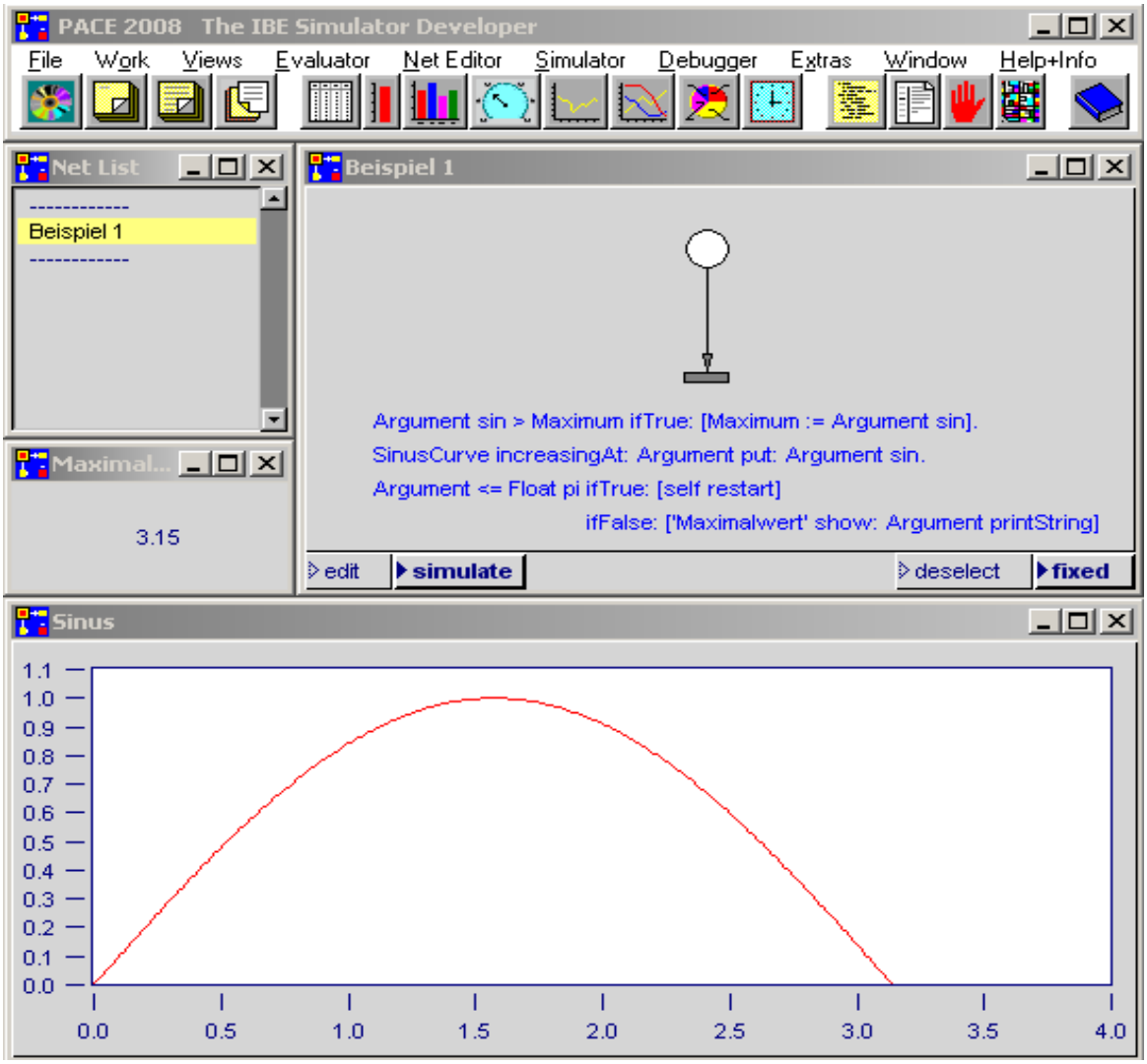



Abb. 6.6: Arbeitsoberfläche für Beispiel 1

## 6.4 Verwendung von PACE-Netzfunktionen

Immer wenn ein funktionaler Zusammenhang zwischen Rechengrößen, einem Satz von Argumenten und einem Resultat vorliegt, kann nach den Argumenten gefragt werden, bei denen das Resultat seine Extremwerte annimmt. Die Optimierung von Teilnetzen basiert auf den sog. PACE-Netzfunktionen. Werden Teile eines Modells in Form einer PACE-Netzfunktion dargestellt, so kann diese von anderen Orten des Modells her aufgerufen und dabei mit Eingangsargumenten versorgt werden.

Abb. 6.7 zeigt eine Arbeitsoberfläche, mit der das Maximum der Netzfunktion Sinus berechnet und in ein Transcript-Fenster ausgegeben wird. Da die meisten Modellierungsschritte in früheren Abschnitten schon erläutert wurden, werden weiterhin nur die Schritte ausführlicher beschrieben, die bisher noch nicht vorkamen.

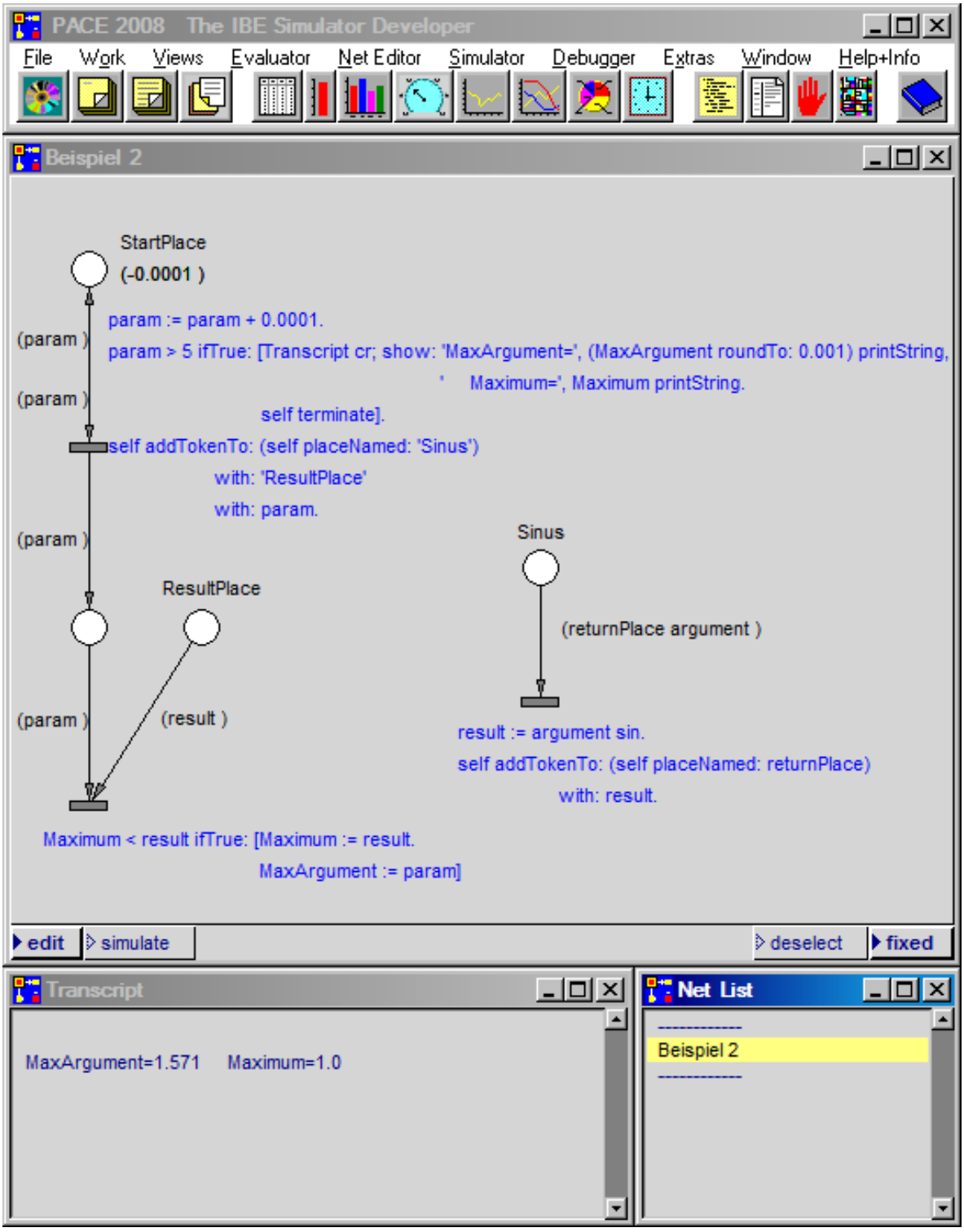



Abb. 6.7: Arbeitsoberfläche für Beispiel 2

Die Netzfunktion besteht in dem vorliegenden einfachen Beispiel nur aus einer Stelle und einer Transition auf der rechten Seite des Netzfensters. Aufgerufen wird sie, indem das aufrufende Teilnetz auf der linken Seite eine Marke mit Parametern in die Eingangs-Stelle **Sinus** legt. Da keine feste Rückkehr-Stelle vereinbart ist, muss außerdem noch die Rückkehr-Stelle **ResultPlace** übergegeben werden, in die eine Marke mit dem Ergebnis, dem Funktionswert, gelegt werden soll.

Der Aufruf der Netzfunktion wird mit der Botschaft:

```
self addTokenTo: eineStelle with: argument1 with: argument2.
```

ausgeführt. Fehlen die with:-Angaben, so wird eine Marke ohne Argumente erzeugt. Es können bis zu vier with:-Argumente angegeben werden. Als erstes Argument ist immer die Stelle zu übergeben, in welche die Marke gelegt werden soll. Eine Stelle ergibt sich aus deren Namen mit der Botschaft: self placeNamed:. Als Argument ist der Name der Stelle in Hochkommas, d.h. in Form eines Smalltalk-Strings, anzugeben. Auch die Rückkehr aus der Netzfunktion wird mit der addTokenTo:-Botschaft durchgeführt.

Im linken Teilnetz sind zwei weitere neue Modellierungsschritte zu erläutern, nämlich die Installation einer Initialmarke mit dem Anfangsargument -0.0001 und die Transcript-Botschaft, mit das Ergebnis in das Transcript-Fenster geschrieben wird.

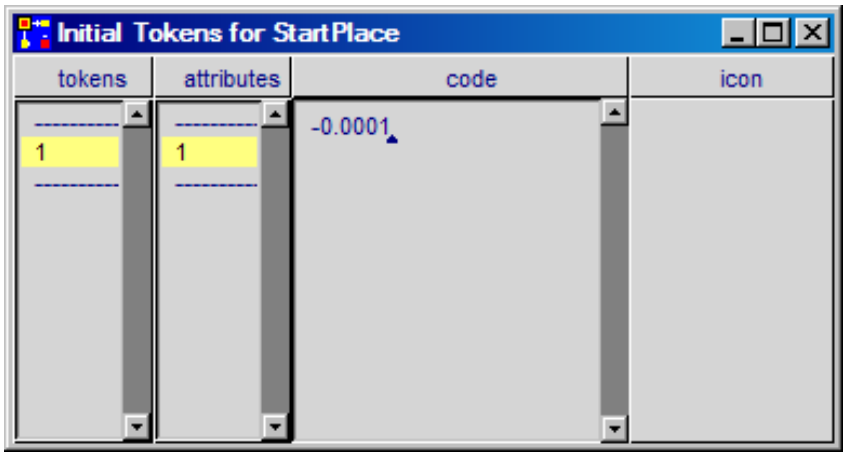



Abb. 6.8: Vereinbarung von Anfangsmarken mit Werten

Zur Installation der Initialmarke wird die Stelle **StartPlace** markiert und in ihrem **re.MT**-Menü der Menupunkt **initial tokens** ausgewählt. In dem sich öffnenden Fenster wird, wie im vorangegangenen Abschnitt, im linken Teilfenster mit der **add**-Funktion eine Marke installiert. Diese Marke soll ein Argument tragen. Deshalb die Markenbezeichnung 1 im Teilfenster **tokens** selektieren und im Teilfenster **attributes** das **re.MT**-Menü aufrufen. Es besteht nur aus dem Menüpunkt **add**, der ausgewählt wird. Jetzt im Teilfenster **code** den Wert -0.0001 eingeben und mit **re.MT**, **accept** gültig machen. Danach sieht das Fenster wie in Abb. 6.8 aus.

Die Transcript-Botschaft, mit der die Ausgabe durchgeführt wird, besteht aus dem Schlüsselwort Transcript, dem normalerweise eine sog. Kaskade folgt. Eine Kaskade ist eine Folge von Smalltalk-Botschaften, die durch Semikolon getrennt sind und die sich alle auf dasselbe Objekt, in diesem Fall auf das durch Transcript angegebene Text-Ausgabefenster, beziehen. Statt der im Fenster angegebenen Transcript-Botschaft hätte man auch die folgenden beiden Botschaften schreiben können:

```
Transcript cr.
Transcript show: 'MaxArgument=', (MaxArgument roundTo: 0.001) printString,
```

' Maximum=', Maximum printString.

Die erste der beiden Botschaften gibt einen Wagenrücklauf (carriage return) aus. Nach der show:-Botschaft in der zweiten Zeile wird immer ein String-Objekt erwartet. Mit dem Operator Komma **,** können verschiedene Strings aneinandergereiht (konkateniert) werden. Zahlen können mit **printString** in Strings gewandelt werden. Schließlich rundet die **roundTo:**-Botschaft die links von ihr stehende Zahl (den sog. Empfänger) auf ein Vielfaches der rechts stehenden Zahl. MaxArgument wird also bis auf vier Stellen hinter dem Komma berechnet und vor der Ausgabe auf drei Stellen nach dem Komma gerundet. Das Ergebnis der Transcript-Botschaft zeigt das Transcript-Fenster in Abb. 6.7.

## *6.5 Mathematische Optimierung von Modellen*

Die Vorgehensweise in den vorangegangenen Abschnitten hat den Nachteil, dass sie bei komplexen Modellen, insbesondere wenn mehrere Argumente vorliegen, eine große Anzahl von Modellausführungen erfordert und daher sehr rechenaufwendig ist. Es liegt deshalb nahe, die Anzahl der Modellausführungen durch Verwendung eines Optimierers zu verringern.

In PACE wurden für die Optimierung von mathematischen und von Netzfunktionen die folgenden Optimierer mit zahlreichen Optionen implementiert, die teilweise auch in Kombination verwendbar sind:

Hill-Climbing-Optimizer
GeneticOptimizer
ThresholdAcceptingOptimizer
Simplex-Optimizer.

Davon können die ersten drei skaliert, d.h. mit reduzierter Genauigkeit eingesetzt werden. Im folgenden wird der Einsatz eines HillClimbingOptimizer für das ausgewählte einfache Beispiel beschrieben. In Kapitel 7 werden die drei skalierbaren Optimierer für die Optimierung einer Fertigung eingesetzt.

Begonnen wird wieder mit einem neuen Modell, das **Beispiel 3** genannt wird. Für den Einsatz eines Optimierers stellt PACE den Bibliotheks-Modul **NetOptimizer** zur Verfügung, der in das leere Netzfenster eingesetzt wird (**re.MT**, **restore module**). Nach dem Einsetzen den Modul **NetOptimizer** markieren und im **re.MT**-Menü den Menüpunkt **refine** wählen. In dem sich öffnenden Abfragefenster den Button **yes** drücken.

Dadurch wird der Modul Netoptimizer aufgelöst und sein Netz im Netzfenster angezeigt. Normalerweise müssen die Netzelemente und Inskriptionen neu angeordnet werden, damit das Netz übersichtlich aussieht. Das Ergebnis ist in Abb. 6.9 dargestellt.

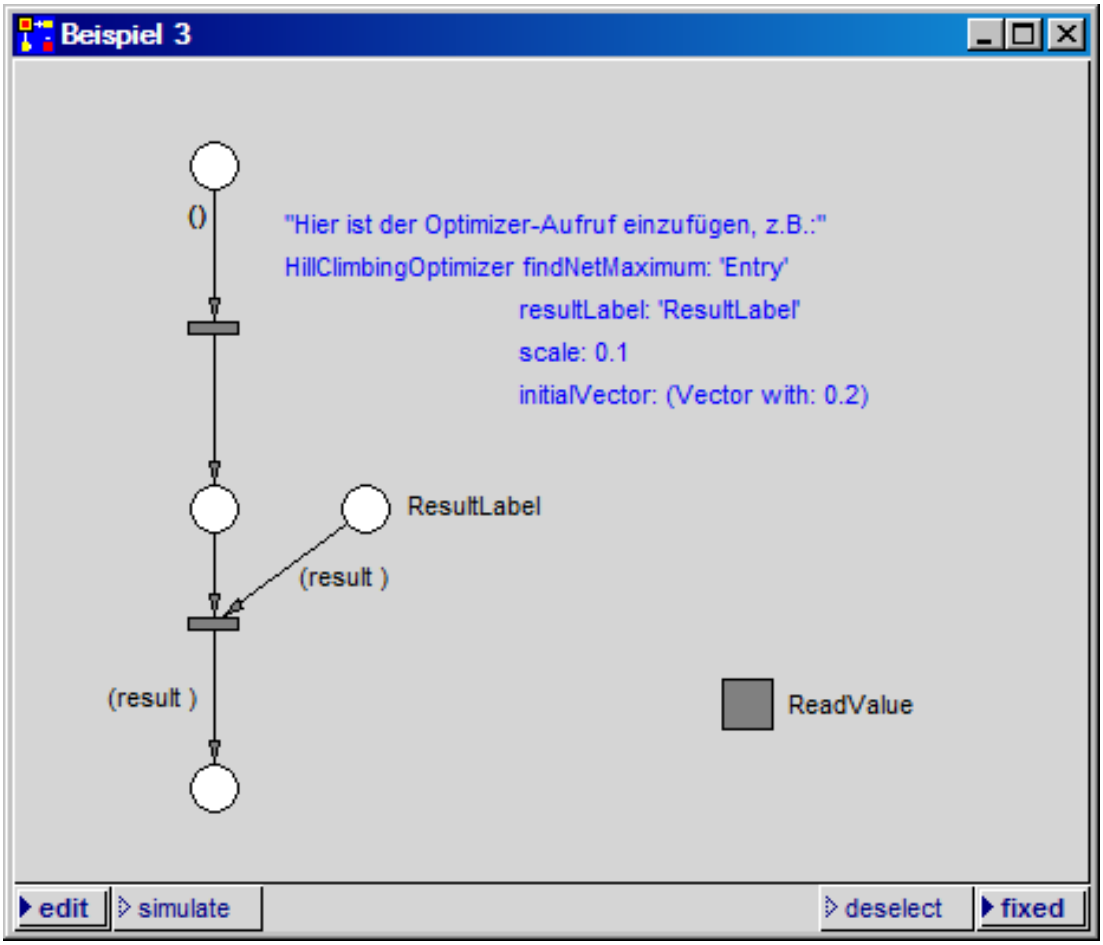



Abb. 6.9: Netzfenster nach Ausführen von refine

Im vorliegenden Fall wird bei der Optimierung kein Skalenfaktor gebraucht. Er wird normalerweise zur Reduzierung der Rechenzeit eingesetzt, wenn Modellparameter nur grob bekannt sind und/oder die Ausführung des Modells große Zeit beansprucht. Deshalb wird die Zeile

scale: 0.1

entweder eliminiert oder auskommentiert (vor dem ersten Zeichen der Zeile und nach dem letzten Zeichen jeweils ein Anführungszeichen " einfügen). Dazu die Action-Inskription mit der **li.MT** markieren und im Menü der **re.MT** den Menüpunkt **inspect** auswählen. Es öffnet sich ein Textfenster, in dem die Änderung durchgeführt wird. Danach das Menü der **re.MT** aufrufen und **accept** wählen.

Die aufzurufende PACE-Netzfunktion ist in dem Modul **ReadValue** enthalten. Der Modul wird markiert und im Menü der **re.MT** der Menüpunkt **subnet** ausgewählt. Es öffnet sich das in Abb. 6.10 gezeigte Netzfenster des Moduls. Es muss noch um die Netze zur Berechnung des Ergebnisses erweitert werden.

Für das hier betrachtete einfache Beispiel ist das Netz nur geringfügig zu erweitern (Abb. 6.11). Die dafür erforderlichen Modellierungsschritte sind nach den früheren Beschreibungen bekannt. Zu erläutern sind aber die beiden Action-Inskriptionen.

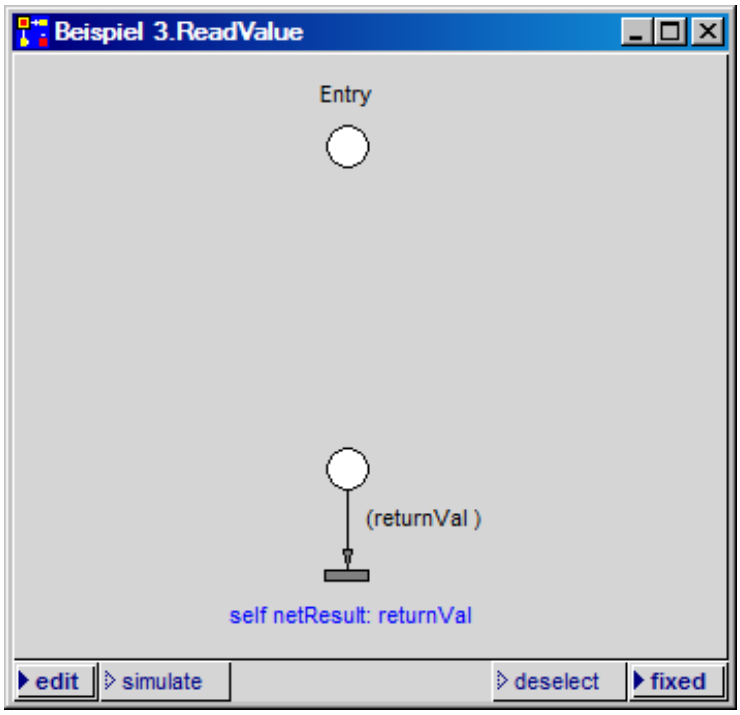



Abb. 6.10: Netz des Moduls ReadValue

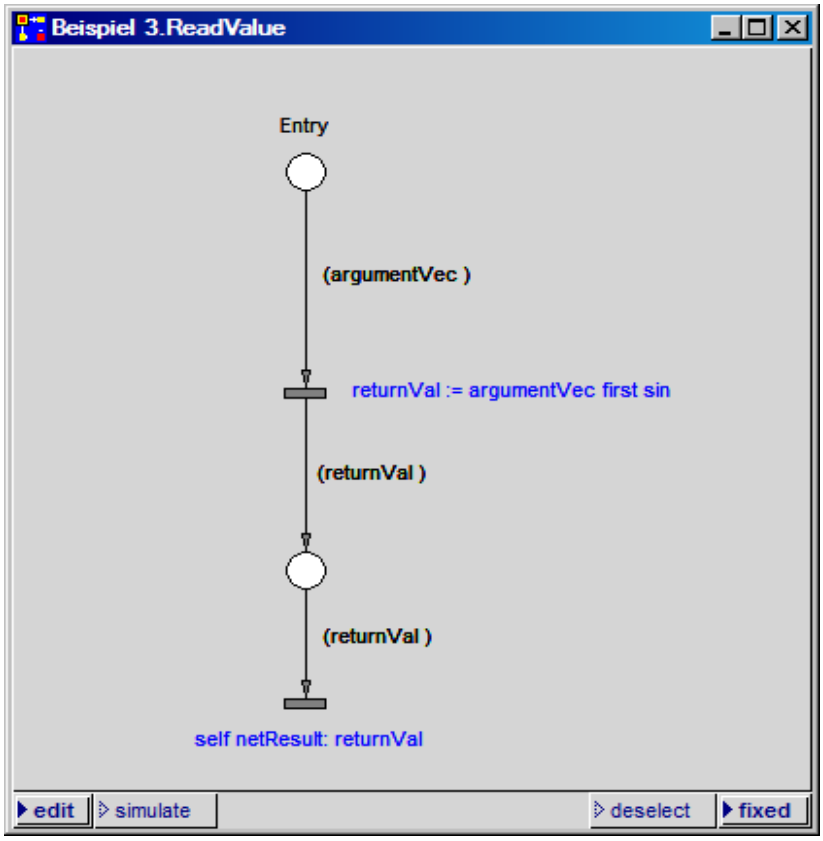



Abb. 6.11: Vervollständigtes Netz des Moduls ReadValue

Die Inskription zur Berechnung des Sinus muss nach den Ausführungen in Abschnitt 6.2 nicht mehr kommentiert werden. Auch bei den Netzoptimierern werden die Argu-

mente in Form eines Vectors übergeben (Vector ist eine Unterklasse von Array und kann deshalb mit den gleichen Botschaften bearbeitet werden).

Die Rückgabe des Ergebnisses erfolgt mit der Botschaft:

self netResult: returnVal.

returnVal wird an den Optimierer zurückgegeben, der die Netzfunktion solange mit jeweils geänderten Argumenten aufruft, bis das Ergebnis die gewünschte Genauigkeit zeigt. Das Ergebnis wird dann als Attribut an eine Marke angehängt, die in die beim Aufruf des Optimierers angegebene Rückkehr-Stelle ResultLabel gelegt wird.

Abb. 6.12 zeigt wieder die endgültige Arbeitsoberfläche mit einigen Vereinfachungen im Netzfenster **Beispiel 3** und einer Ergebnismarke.

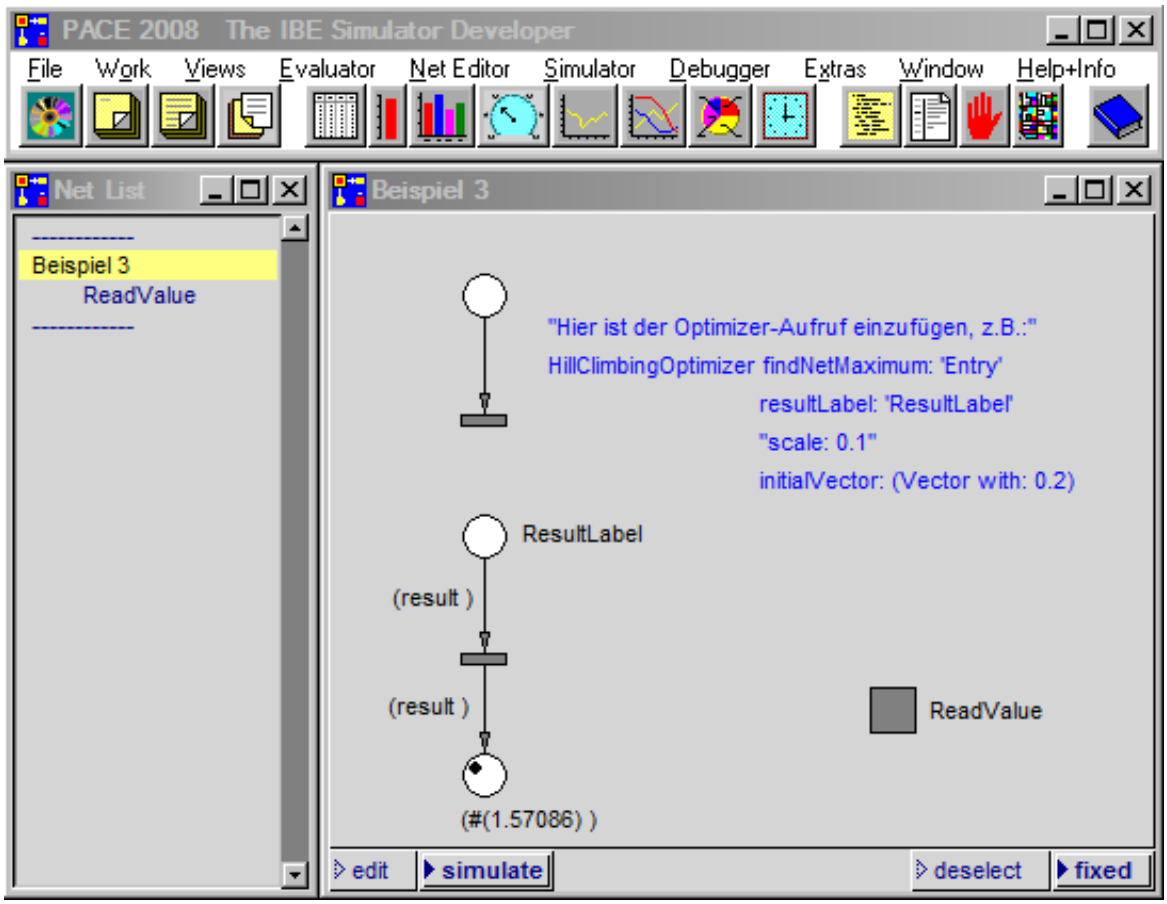



Abb. 6.12: Arbeitsoberfläche für die mathematische Modelloptimierung

# 7. Beispiel: Optimierung einer Fertigung

## 7.1 Aufgabenstellung

Es soll ein Modell erstellt werden, mit dem die Anzahl von Facharbeitern zur Herstellung oder Bearbeitung eines bestimmten Produkts berechnet werden kann. Dabei sind folgende Forderungen zu berücksichtigen:

- **Aufträge**
  Die Aufträge für jeweils ein Produkt treffen exponentiell verteilt ein. Der Mittelwert der Verteilung soll im (linksseitig offenen) Intervall (0, 10] Stunden liegen und wird vor der Simulation vom Anwender eingestellt.

- **Charakteristiken der Fertigung**
  Für die Bearbeitung eines Auftrags wird eine feste Zeit benötigt, die vor einem Simulationslauf im Intervall (0, 10] Stunden eingestellt wird. Ein Auftrag wird jeweils von genau einem Facharbeiter durchgeführt.

- **Durchlaufzeit**
  Die Zeit für die Auftragsdurchführung ist beschränkt und wird vor einem Simulationslauf im Intervall (0, 10] Stunden vorgegeben. Die vorgegebene Durchführungszeit darf nur bei einem sehr geringen Prozentsatz der Aufträge überschritten werden.

## 7.2 Erstellen des Modells

Während in den früheren Abschnitten, die (syntaktische) Konstruktion der Modelle Schritt für Schritt beschrieben wurde, wird im folgenden eine verkürzte Darstellung verwendet, die davon ausgeht, dass die dargestellten Grafiken und Texte vom Leser nachmodelliert werden. Nur die Einzelschritte, die in den vorangegangenen Abschnitten noch nicht erläutert wurden, oder Ergänzungen zu früheren Beschreibungen werden genauer dargestellt. Das gilt natürlich nicht für die inhaltliche (semantische) Beschreibung des Modells. Letztere wird anhand der abgebildeten Grafiken und Texte vorgenommen.

Begonnen wird wieder mit einem neuen Modell, das den Namen FertigungsOptimierung erhält. In das Netzfenster wird, wie in Abschnitt 6.5 beschrieben, der Aufruf eines Optimierers eingesetzt. Man gelangt damit wieder zu der in Abb. 6.9 dargestellten Grafik mit geändertem Fensternamen.

In diesem Netzfenster werden, wie in Abb. 7.1 dargestellt, die Actioncodes geändert. Das Optimum soll ausgehend von 30 Facharbeitern gefunden werden. Die Skalierung ist ganzzahlig, weil jeder Auftrag von genau einem Bearbeiter durchgeführt wird.

Die Ausgabe des Ergebnisses gibt der zweite Actioncode an. Zunächst wird mit einer Transcript-Botschaft mit Kaskade das Ergebnis in ein Transcript-Fenster geschrieben. Danach wird das Ergebnis an einen noch zu definierenden AlternativeBarGauge ausgegeben, der in der globalen Variablen **Ergebnis** gespeichert ist. Das Fenster des

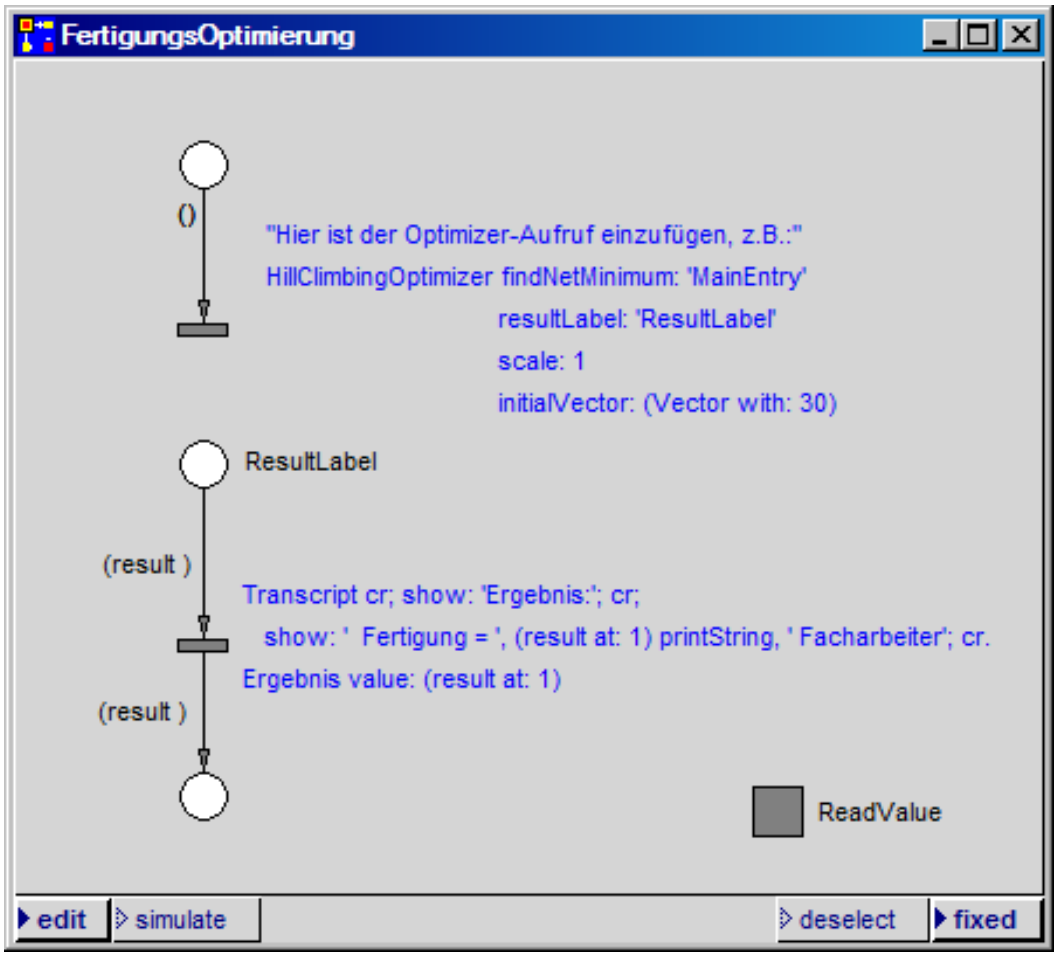



Abb. 7.1: Aufruf des Optimierers für die Fertigung

AlternativeBarGauge (sein sog. View) wird weiter unten geöffnet. Bei der Übersetzung des Actionscode (Menüpunkt **accept** im **re-MT**-Menü des Actionscode) wird **Ergebnis** als globale Variable vereinbart.

Statt eines HillClimbingOptimizers kann man auch einen der beiden anderen anderen skalierbaren Optimierer von PACE verwenden. Im Fall eines genetischen Optimierers könnte der Actioncode wie folgt aussehen:

```
GeneticOptimizer findNetMinimum: 'MainEntry'
                 resultLabel: 'ResultLabel'
                 scale: 1
                 origin: (Vector with: 1)
                 range: (Vector with: 30)
```

Das Minimum, d.h. die kleinste Zahl von Facharbeitern, mit denen sich die vorgegebenen Bedingungen erfüllen lassen, soll zwischen den Grenzen 1 (origin) und 30 (range) gesucht werden.

Im Fall der Schwellenwert-Akzeptanz sieht der Aufruf wie folgt aus:

```
ThresholdAcceptionOptimizer findNetMinimum: 'MainEntry'
                            resultLabel: 'ResultLabel'
                            scale: 1
                            initialVector: (Vector with: 10)
                            initialThreshold: 3
```

Es ist möglich, alle drei Aufrufe zugeordnet zur Transition parallel zu speichern und wahlweise zu verwenden. Dazu wird im **re.MT**-Menü der Transition der Menüpunkt **code versions** aufgerufen und im Submenü **action code** ausgewählt. Es öffnet sich ein Fenster, mit dem die Versionen der Actioncodes der Transition verwaltet werden. Im linken Teilfenster wird das **re.MT**-Menü aufgerufen und der Menüpunkt **add** gewählt. Es öffnet sich ein Eingabefenster für den Versionsnamen, unter dem der aktuelle Actioncode gespeichert werden soll (Abb. 7.2).

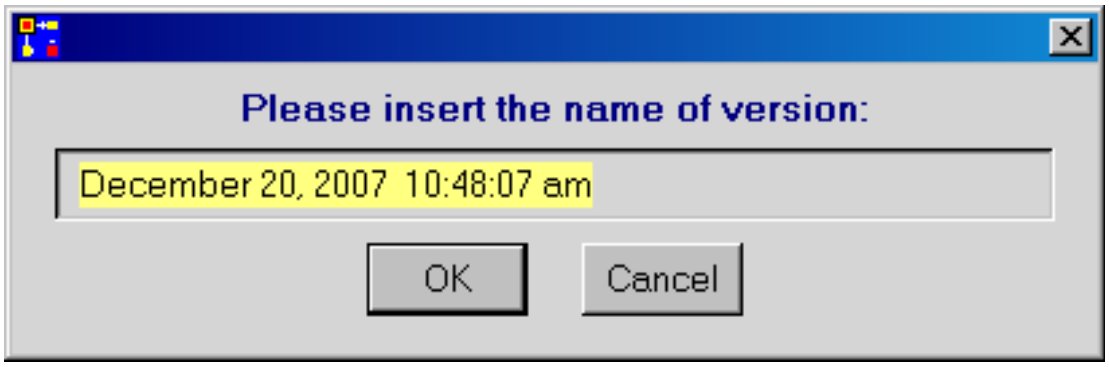



Abb.7.2: Eingabe des Versionsnamens

Als Default-Name ist die aktuelle Zeit eingetragen. Im vorliegenden Fall handelt es sich nicht um die Handhabung von Versionen, sondern um das Abspeichern der drei alternativen Optimierer-Aufrufe. Der Default-Name wird deshalb durch den Namen: HillClimbing ersetzt. Auf die gleiche Weise kann man auch die beiden weiteren Optimiereraufrufe speichern und gelangt damit zu dem in Abb. 7.3 dargestellten Fenster.

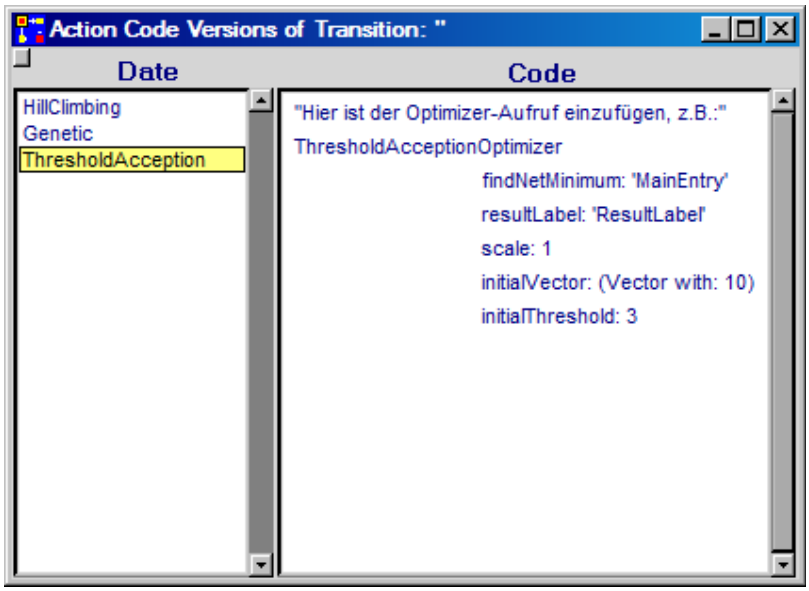



Abb. 7.3: Verwaltung von Code-Versionen

Will man den aktuellen Actioncode der Transition durch einen der gespeicherten Codes ersetzen, so wird der Name des Codes im linken Fenster markiert und im Menü des linken Fensters (**re.MT)** der Menüpunkt **restore** ausgewählt.

Um das Fenster Abb. 7.1 in der Arbeitsoberfläche klein machen zu können, werden beide Transitionscodes in Drei-Punkte-Menüs … verborgen.

Als nächstes kann jetzt die Arbeitsoberfläche des Optimierungsmodells erstellt werden (Abb. 7.4).

Zur Steuerung der Simulation wird eine horizontale kurze Exekutive verwendet. Außer dem AlternativeBarGauge **Ergebnis** sind drei weitere AlternativeBarGauges **Produktionszeit**, **Max. Durchlaufzeit** und **Mean** vorgesehen, mit denen die bei der Fertigung zu berücksichtigenden Vorgaben eingestellt werden.

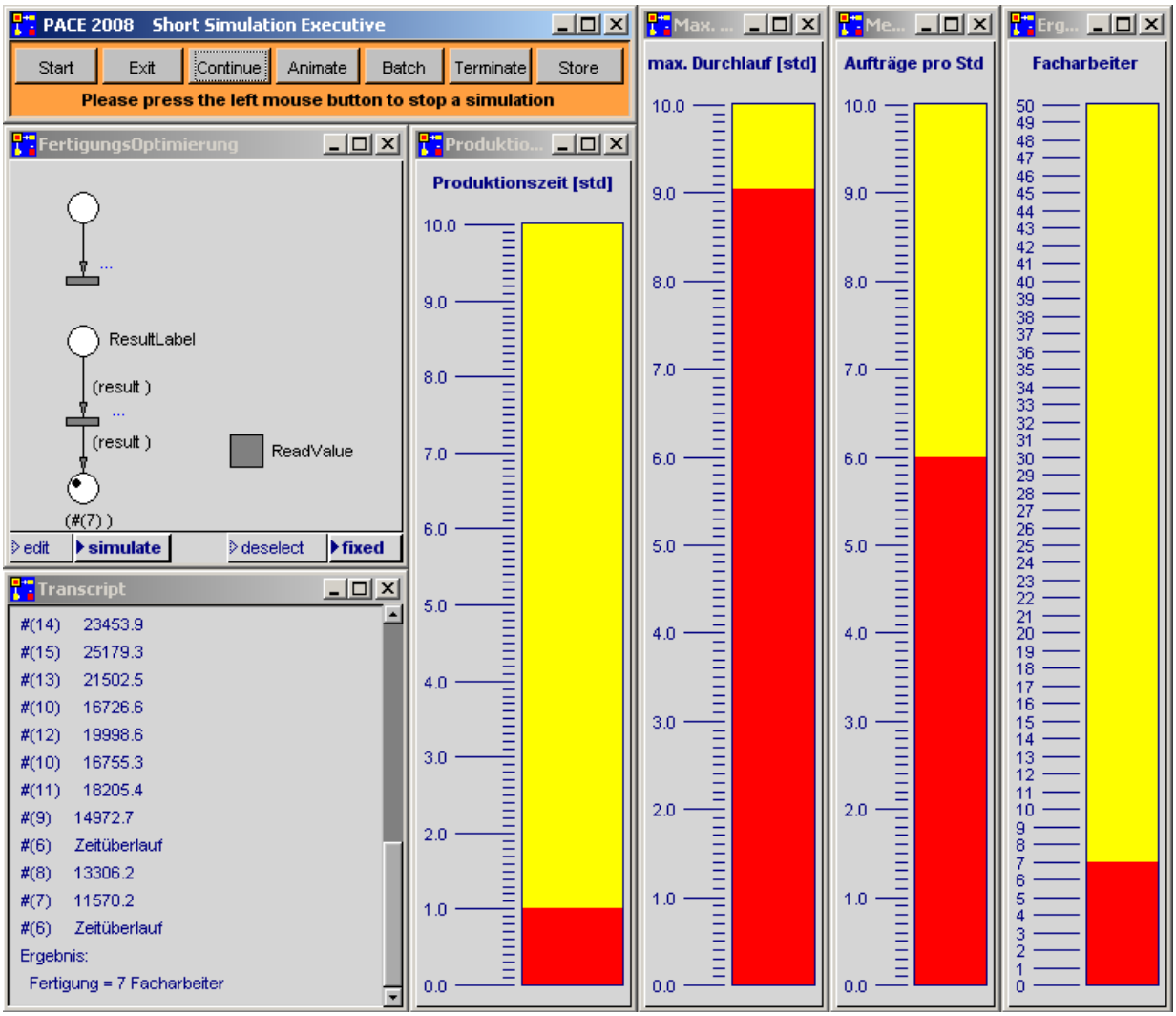



Abb. 7.4: Arbeitsoberfläche für die FertigungsOptimierung

Die Skaleneinteilung und weitere Daten eines AlternativeBarGauges kann man über sein Parameterfenster einstellen. Dieses wird über sein Menü (**re.MT**) durch Auswahl des Menüpunkt **parameter** angezeigt. In Abb. 7.5 ist das Parameterfenster von **Pro-**

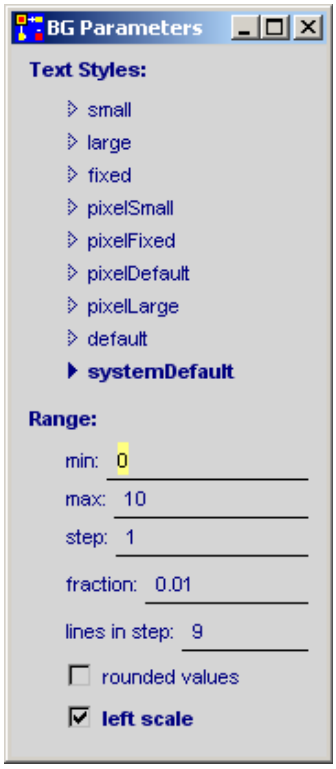



Abb.7.5:Parameterfenster eines AlternativeBarGauge

**duktionszeit** dargestellt.

Mit **Text Styles** kann die Zeichen-Darstellung der Skalierung ausgewählt werden.

Die **Range**-Angaben setzen die Werte der Skalierung. **min** und **max** geben den Minimal- und Maximalwert der Skala an. **step** legt fest, in welchen Werte-Abständen die Beschriftung erfolgen soll. **fraction** gibt die Genauigkeit vor, mit der Werte dargestellt werden sollen. Schließlich kann man mit **lines in step** vorgeben, wie viele Linien zwischen den "steps" zur weiteren Untergliederung der Skalenbeschriftung eingefügt werden sollen.

Durch Ankreuzen von **rounded values** wird festgelegt, dass nur Vielfache des "step"-Wertes eingestellt werden können. Mit **left scale** wird bestimmt, ob die Skala rechts oder links vom Balken angeordnet werden soll.

Es empfiehlt sich, die Parameterfenster der verschiedenen Grafiken, die über das View-Menü der PACE-Leiste angezeigt werden können, experimentell zu erkunden, um die Wirkungsweise der verschiedenen Einstellungen herauszufinden.

Wie Abb. 7.4 zeigt, kann man in AlternativeBarGauges über dem Balken einen einzeiligen Kommentar einfügen. Hierzu wieder das Menü des BarGauge aufrufen und **label** wählen. In das sich öffnende Eingabefenster denKommentartext eingeben und mit der Return-Taste ins Fenster einsetzen.

Wird die Skalierung später in einem anderen Modell wieder benötigt, so kann sie abgespeichert und später wieder in einen anderen AlternativeBarGauge geladen werden. Das geschieht mit den Menüpunkten **store** und **restore** im Menü des Balkenfensters. Durch **store** wird die Skalierung in eine Datei im Unterverzeichnis **ioutils** des Verzeichnisses, aus dem heraus PACE gestartet wurde, geschrieben. **restore** öffnet ein Auswahlfenster mit den verfügbaren Skalierungen. Die meisten Ein/Ausgabe-Fenster von PACE verfügen über diese Funktionen, um die Skalierung der Grafiken zu erleichtern.

Die in Abb. 7.4 angegebenen AlternativeBarGauges werden wie früher über den Initialisierungscode an das Modell angeschlossen. Abb. 7.6 zeigt den Initialisierungscode.

In den ersten fünf Zeilen werden die vier AlternativeBarGauges an vier globale Variablen zugewiesen und die maximal zulässige Durchlaufzeit in der globalen Variablen **AktMaxDurchlaufzeit** gespeichert. Über die globalen Variablen wird später auf die BarGauges zugegriffen. Die 6. Zeile leert das Transcript-Fenster, sodaß die Ausgabe der Ergebnisse (siehe Abb. 7.4 ) in ein leeres Fenster erfolgt.

In den nächsten beiden Zeilen wird untersucht, ob die Produktionszeit größer als die

geforderte maximale Durchlaufzeit ist. Wenn dies der Fall ist, wird eine Fehlermeldung ausgegeben und der Simulationslauf wird beendet.

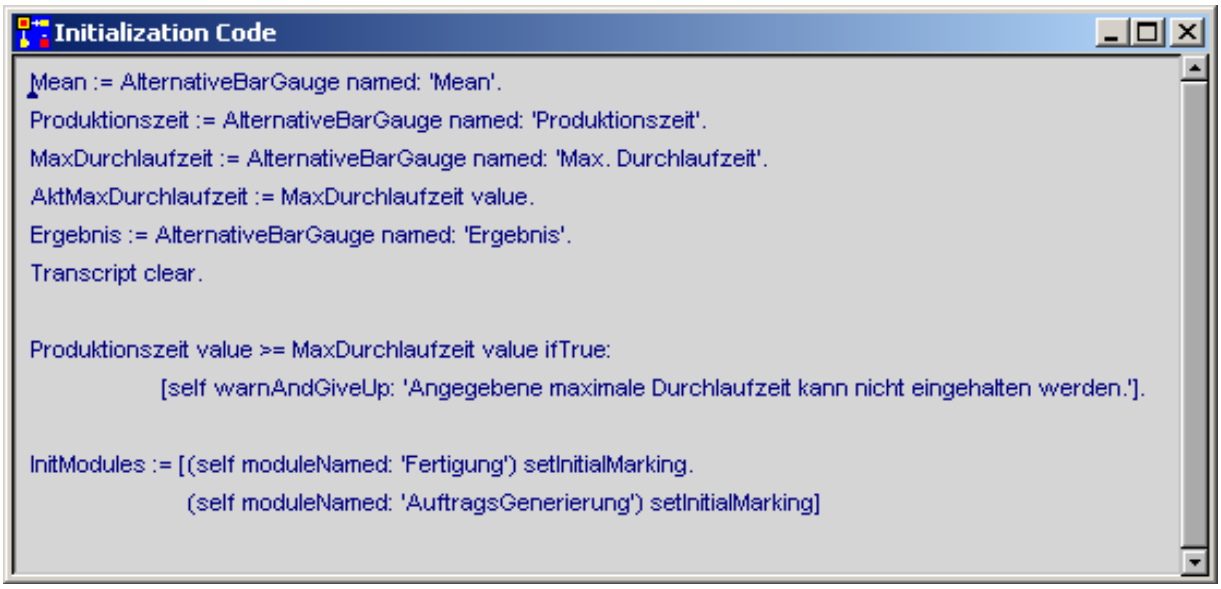



Abb.7.6: Initialisierungscode der FertigungsOptimierung

Die letzten beiden Zeilen wurden während der Modellierung des Moduls **ReadValue** hinzugefügt. Da die im Block angegebenen Zeilen, der sog. Blockcode, im Modell an verschiedenen Stellen auszuführen sind, wird der Block einer globalen Variablen **InitModules** zugewiesen. Damit kann der Anfangszustand der noch zu erstellenden Module **AuftragsGenerierung** und **Fertigung** durch Aufruf des Blocks mit der Botschaft **InitModules value** wiederhergestellt werden.

Der Modul **ReadValue** ist in Abb. 7.9 dargestellt. Die PACE-Netzfunktion MainEntry wird vom Hill Climbing Optimizer mit einem Argument, welches das einzige Element des Vectors **paramVector** ist, aufgerufen. Es gibt die Anzahl von Facharbeitern an, mit der die Produktion bzw. die Netzprozedur ausgeführt werden soll.

Um negative Werte, die beim HillClimbingOptimizer auftreten können, abzufangen, werden diese sofort nach rechts abgeleitet (Bedingungscode: paramVector first <= 0). Dabei wird ein sehr großer Wert an den Optimierer zurückgegeben, der weit oberhalb des erwarteten Minimums liegt (siehe weiter unten).

Bei positiven Werten wird in der Transition mit Bedingungscode paramVector first > 0 der Ausgangszustand für die Simulation hergestellt. Zunächst wird in das Transcript-Fenster zur Dokumentation des Optimiererablaufs die Anzahl der Facharbeiter ausgegeben. Danach wird die Simulationszeit zurückgesetzt und es werden die Module **AuftragsGenerierung** und **Fertigung** wie oben beschrieben initialisiert. Schließlich wird die Kapazität der Stelle 'Facharbeiter', die sich im Modul Fertigung befindet, mit der Anzahl von Facharbeitern besetzt.

Da die Aufträge statistisch verteilt ankommen, d.h. sich in gewissen Grenzen häufen können, genügt es nicht, nur einen Auftrag zu betrachten. Man muss vielmehr die Auftragsmengen in der Stelle **Auftragswarteschlange**, die sich im Zuge der Bearbeitung

einstellen können, berücksichtigen. Deshalb erzeugt der Modul AuftragsGenerierung eine große Anzahl von Aufträgen, deren Ankunftszeiten exponentiell verteitt sind. Der Mittelwert der Exponentialverteilung wird über den AlternativeBarGauge **Mean** eingestellt.

Abb.7.7 zeigt den Modul **Auftragsgenerierung**. In der ersten Transition von oben wird die Exponentialverteilung erzeugt, über welche der zeitliche Abstand zwischen zwei aufeinander folgenden Aufträgen festgelegt wird. Danach wird eine Konnektorvariable **auftragsZähler** mit der Anzahl von Aufträgen initialisiert, die insgesamt bearbeitet werden sollen. Im Modell ist die Netzvariable **#Auftragszahl** des Moduls **AuftragsGenerierung** mit 10000 vorbesetzt. In der globalen Variablen **Produktionszeit_Eingehalten** wird notiert, ob während der Abarbeitung der Aufträge irgendwann die vorgegebene maximale Durchlaufzeit überschritten wurde. Dann kann zur Verkürzung der Simulationszeit der Simulationslauf mit der gerade verwendeten Anzahl von Facharbeitern abgebrochen und zum Optimizer zurückgekehrt werden.

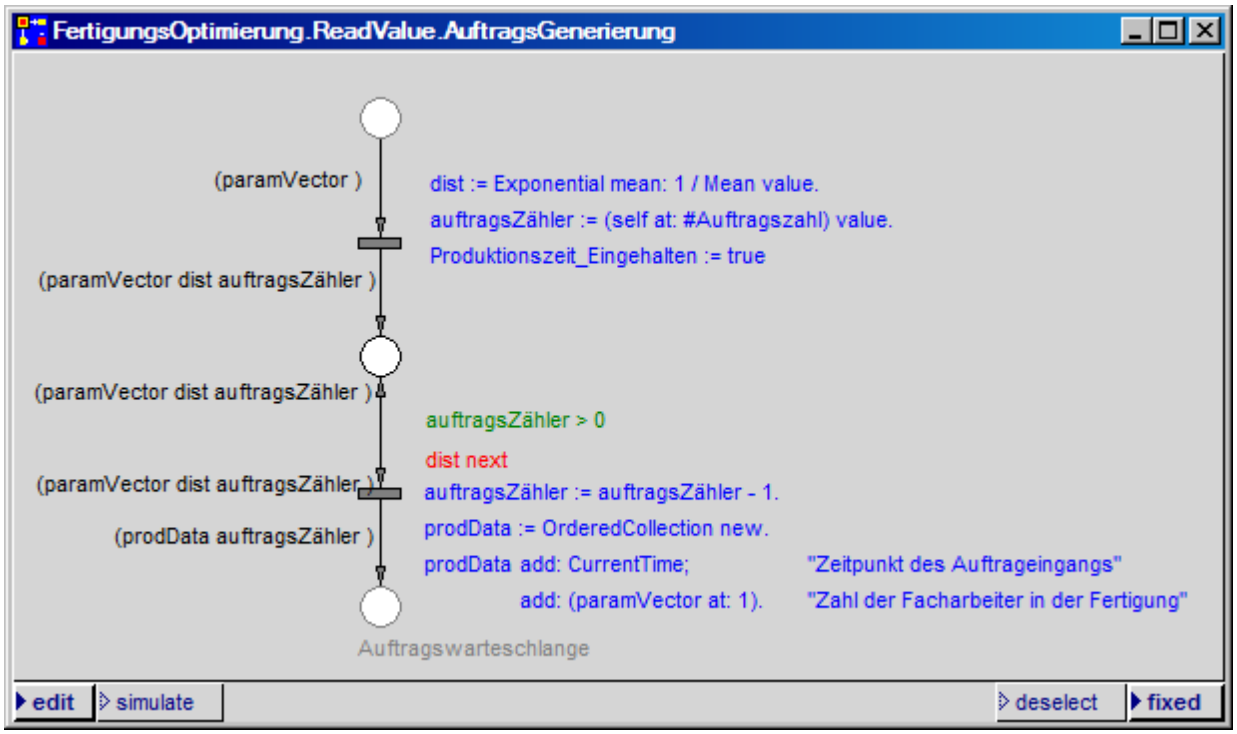



Abb. 7.7: Der Modul Auftragsgenerierung

Die zweite Transition des Moduls ist mit der darüber liegenden Stelle durch einen Doppelkonnektor verbunden. Schaltet die Transition, so wird, solange die Bedingung **auftragsZähler > 0** erfüllt ist, von der oben liegenden Stelle eine Marke abgezogen und für den nächsten Auftrag wieder zurückgelegt. Die Transition schaltet erneut nach der Zeitspanne, die durch den nächsten Wert der Exponentialverteilung **dist next** bestimmt ist. Außerdem wird eine Marke, der nächste Auftrag, in die Stelle **Auftragswarteschlange** gelegt. Dabei wird der Actioncode ausgeführt. In ihm wird der Auftragszähler um 1 erniedrigt und in einer sog. OrderedCollection **prodData**, die als Attribut einer den Auftrag repräsentierenden Marke durch das Netz läuft, die aktuelle Zeit und die Anzahl der aktuell eingestellten Facharbeiter gespeichert. Eine OrderedCollection ist ein spezieller Array mit dynamisch änderbarer Länge. Der aktuelle Wert der Kon-

nektorvariablen **auftragsZähler** gibt die Zahl der noch zu erzeugenden Aufträge an. In Abb. 7.8 ist der Modul **Fertigung** dargestellt, in dem wie bei der früher beschriebenen Waschstraße eine Warteschlange modelliert ist. Im Unterschied zu dieser dürfen hier allerdings mehrere Objekte gleichzeitig bearbeitet werden.

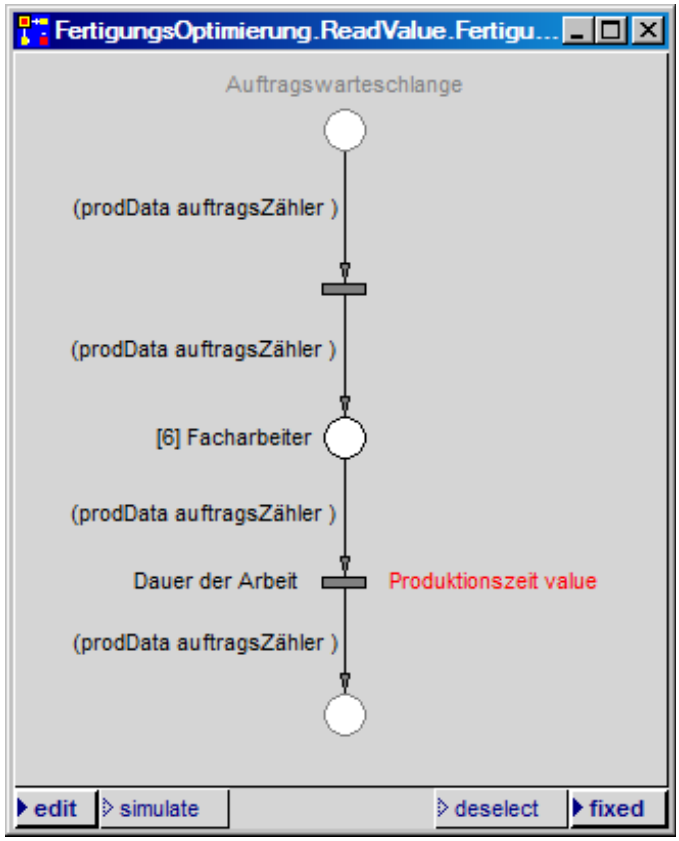



Abb. 7.8: Der Modul Fertigung

Im Anschluss an den Modul **Fertigung** wird im Modul **ReadValue** überprüft, ob die maximale Produktionszeit überschritten wurde. Die Differenz **CurrentTime - prodData first** liefert die Durchlaufzeit des Auftrags. Ist sie größer als die im AlternativeBarGauge eingestellte maximale Durchlaufzeit, so ist die Anzahl der Facharbeiter zu klein gewählt und die globale Variable **Produktionszeit_Eingehalten** wird mit **false** besetzt.

In der darauf folgenden Stelle läuft die Marke nach rechts und wird vernichtet, wenn noch weitere Aufträge vorliegen und die Produktionszeit eingehalten wurde. Andernfalls läuft die Marke nach unten. Hier wird zuerst wieder die Erzeugung weiterer Marken bzw. Aufträge abgestellt, indem der Block InitModules aufgerufen wird. Danach wird der Funktionswert berechnet.

Die gesamte Zeit für die Abwicklung der 10000 Aufträge ist CurrentTime. Multipliziert man diese Zeit mit der Anzahl der Facharbeiter, die in (prodData at: 2) steht, so ergibt sich eine Größe, die proportional zu den Bearbeitungskosten ist und deshalb als Funktionswert für die Kostenoptimierung verwendet werden kann. Falls die Produktionszeit nicht eingehalten wurde, wird ein großer Ersatzwert 1.5e6 als Funktionswert zurückgegeben.

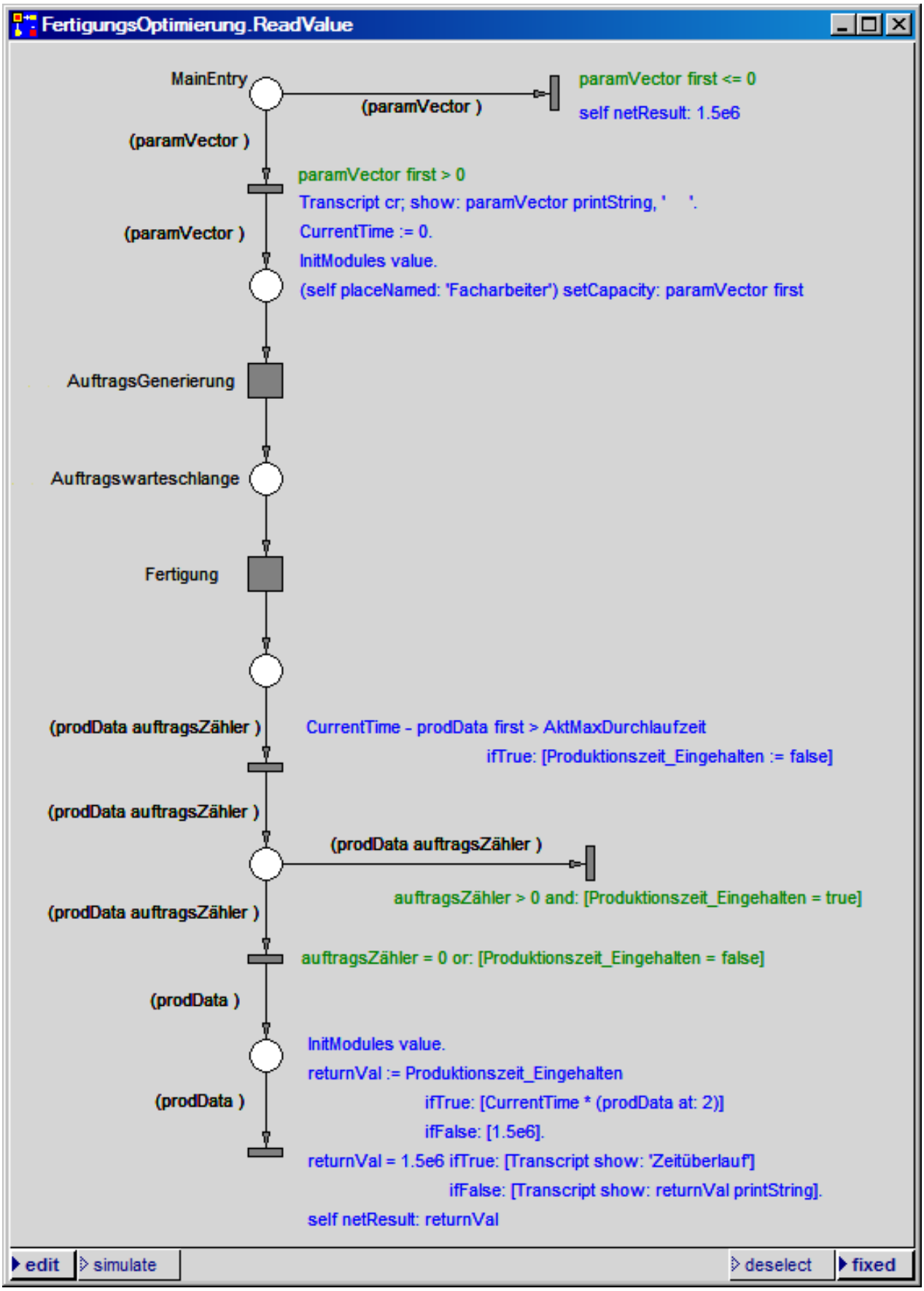



Abb. 7.9: Der Modul ReadValue für die FertigungsOptimierung

Als letzte Maßnahme in einem Simulationslauf ist die am Anfang des Moduls begonnene Ergebnisausgabe zu vervollständigen. Je nachdem, ob die Produktionszeit eingehalten wurde oder nicht, wird in das Transcript-Fenster der berechnete Funktionswert oder die Zeichenkette **Zeitüberlauf** ausgegeben.

## *7.3 Experimentieren mit dem Modell*

Sehr häufig basieren Planungen auf der Betrachtung von Mittelwerten. Das ist nur dann zulässig, wenn die Durchlaufzeiten und Auftragswarteschlangen nicht beschränkt sind. Setzt man z.B. die Produktionszeit und den Mittelwert der Aufträge pro Stunde beide auf 1 und die maximale Durchlaufzeit auf 10, so reichen im vorliegenden Modell 2 Facharbeiter für die Durchführung der Aufträge aus. Diesen Wert würde man auch bei Betrachtung der Mittelwerte erhalten.

Das ändert sich dramatisch, wenn die maximal zulässige Durchlaufzeit beschränkt wird. Liegt die maximal erlaubte Durchlaufzeit bei einem Wert nahe bei 1, so werden 7 Facharbeiter für die Auftragsdurchführung benötigt. Die Forderung nach schneller Lieferung eines Produkts bzw. schneller Bearbeitung eines Auftrags führt erwartungsgemäß zu erheblichen zusätzlichen Kosten.

Eine Übersicht über die Situation verschafft man sich am besten durch Tabellen oder geeignete Grafiken. Um sich einen Überblick über die Abhängigkeit der Facharbeiteranzahl von der maximalen Durchlaufzeit für Auftragsverteilungen mit verschiedenen Mittelwerten zu verschaffen, wird das Modell aus Abschnitt 7.2 für die Durchführung von Experimenten geringfügig erweitert. Dabei wird von der in Abschnitt 6.3 beschriebenen Möglichkeit Gebrauch gemacht, das Modell insgesamt wiederholt auszuführen.

Abb. 7.14 zeigt die erweiterte Modelloberfläche. Zu der früheren Oberfläche (Abb. 7.4) ist ein sog. ButtonBoard (Knopfleiste) und ein MultipleCurve-Fenster hinzugekommen. Mit der Knopfleiste wird bestimmt,

- Button: **ein Wert**
  dass für eine Kombination Produktionszeit, max. Durchlaufzeit und Aufträge pro Stunde die erforderliche Anzahl von Facharbeitern berechnet werden soll.
- Button. **eine Kurve**
  dass eine Kurve berechnet und gezeichnet werden soll, welche die Anzahl von Facharbeitern bei festgehaltener Produktionszeit und festgehaltenem Auftragsmittelwert (mittlere Anzahl von Aufträgen pro Stunde) darstellt. Die maximale Produktionszeit durchläuft dabei den Bereich von der eingestellten Produktionszeit bis zum Wert von 10 Stunden in Abständen von einer Stunde.
- Button: **alle Kurven**
  dass die im vorangegangenen Punkt genannte Kurve für alle ganzzahligen Auftragsmittelwerte im Bereich von 1 bis 10 gezeichnet werden soll.

Den erweiterten Initialisierungscode zeigt Abb. 7.10. Der größte Teil des früheren Codes (siehe Abb. 7.6) ist im false-Zweig der mit **self isRestarted** beginnenden bedingten Botschaft enthalten. Hinzugekommen ist eine globale Variablen **ResultCurves**, welche die mehrfache Kurve 'Anzahl der Facharbeiter' speichert. Nach ihrer Zuord-

nung werden mit **clearAll** zunächst alle eventuell vorhandenen Kurven im Kurvenfenster gelöscht. Danach wird das ButtonBoard **Options** in einer weiteren globalen Variablen **Options** gespeichert. In den weiteren globalen Variablen **EinzelneKurve** und **EinWert** werden die Zustände der Knöpfe '**eine Kurve**' und '**ein Wert**' des ButtonBoard gespeichert. Ist ein Button ausgewählt, so liefert die Abfrage des Button den Wert **true**, andernfalls der Wert **false**.

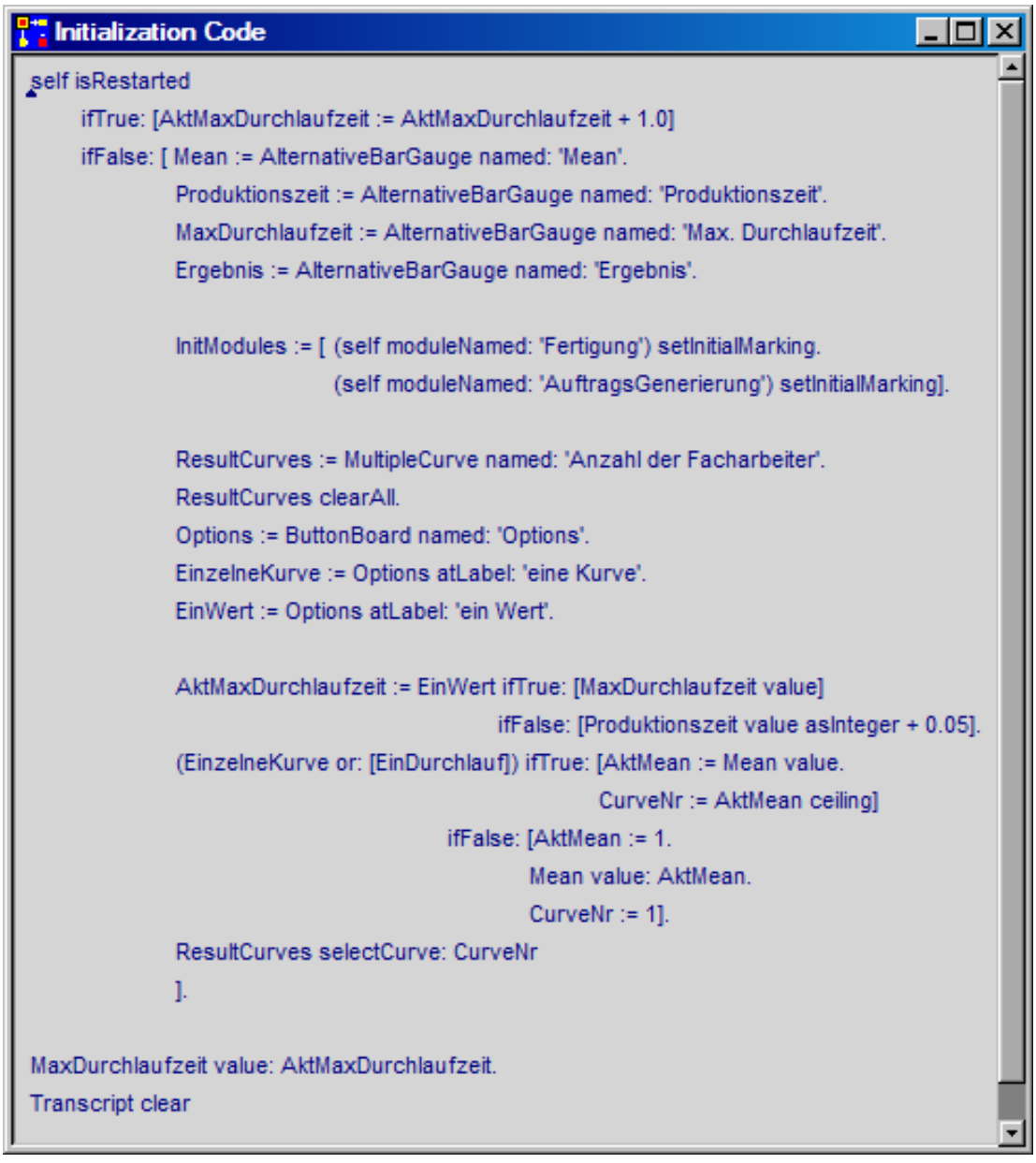



Abb. 7.10: Erweiterter Initialisierungscode

Danach wird die aktuelle Durchlaufzeit **AktMaxDurchlauf** besetzt. Ist der Button 'ein Wert' gesetzt, so wird der Wert des Balken **Max. Durchlaufzeit** zugewiesen. Andernfalls wird ein Wert verwendet der geringfügig größer als der Wert des Balken **Produktionszeit** ist. De Produktionszeit ist die untere Grenze für die Durchlaufzeit.

In den weiteren Zeilen des false-Block wird der Mittelwert der Exponentialverteilung

**AktMean** und die zugeordnete Kurvenfarbe eingestellt, die durch die Nummer der Kurve **CurveNr** definiert ist. Soll nur ein Wert berechnet oder eine Kurve gezeichnet werden, so gibt der Balken **Mean** diesen Mittelwert vor und die Kurvennummer ist durch die nächst größere ganze Zahl definiert (AktMean ceiling). Andernfalls sollen alle Kurven gezeichnet werden und an AktMean und CurveNr wird der Anfangswert 1 zugewiesen.

Die letzte Zeile des Blocks stellt für die weiteren Ausgaben an ResultCurves die eingestellte Kurve ein.

Im true-Block wird für den nächsten Durchlauf des Modells der nächste Wert der Abszisse durch Erhöhen der aktuellen maximalen Durchlaufzeit **AktMaxDurchlaufzeit** um 1 berechnet. Nach der bedingten Botschaft wird schließlich der Balken **Max. Durchlaufzeit** auf den neuen Wert eingestellt und der Inhalt des Transcript-Fensters gelöscht.

Außer dem Initialisierungscode ist auch der Transitionscode zu ändern, der ausgeführt wird, wenn eine Marke, die der Optimierer in die Stelle **ResultLabel** legt, verarbeitet wird (Abb. 7.11).

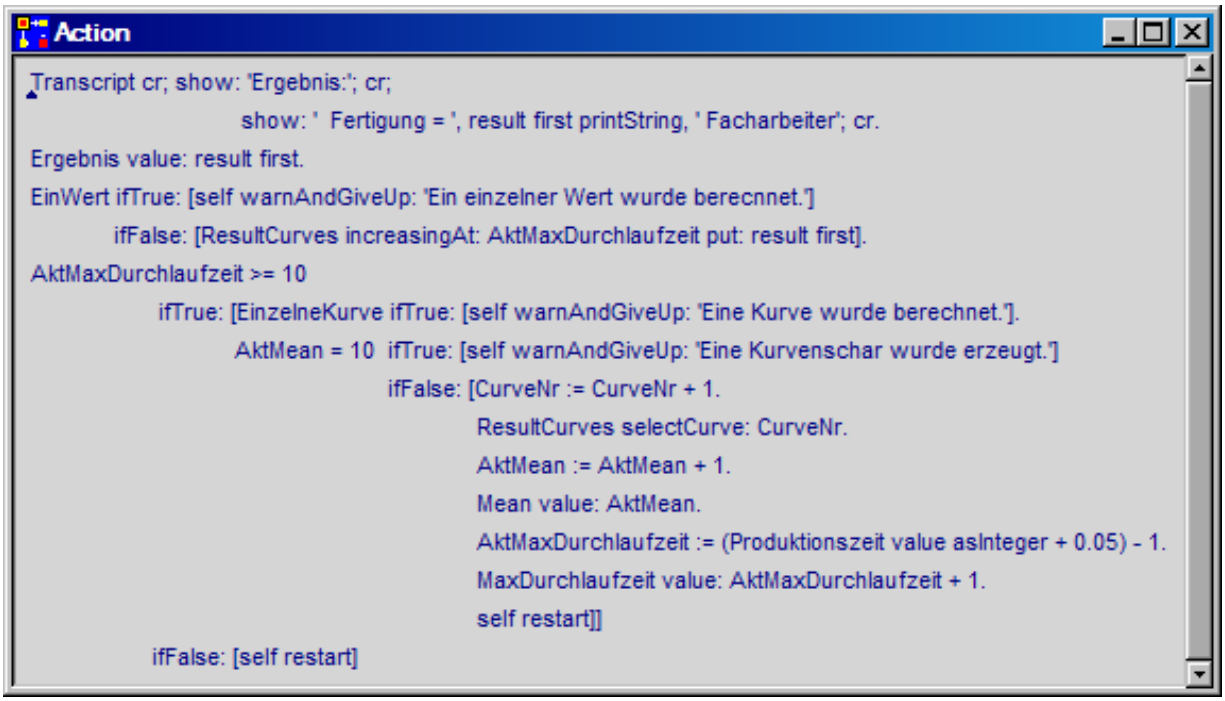



Abb. 7.11: Erweiterter Transitionscode

Die ersten drei Zeilen sind schon aus Abschnitt 7.2, Abb. 7.1 bekannt.

In der bedingten Botschaft, die **EinWert** ausgewertet, wird im Fall true der Simulationslauf beendet, andernfalls der nächste Wert an die Kurve **ResultCurves** ausgegeben.

Ist **AktMaxDurchlaufzeit** >= 10, so ist eine Kurve fertiggestellt, andernfalls wird mit

der Botschaft **restart** der nächste Durchlauf des Modells gestartet und der nächste Wert der aktuellen Kurve berechnet. Im true-Zweig wird in Abhängigkeit davon, ob Aktmean = 10 ist, entweder die Simulation beendet oder zur nächsten Kurve weitergeschaltet (AktMean := AktMean + 1).

Es bleibt noch die Beschreibung, wie das in Abb. 7.12 enthaltene ButtonBoard erstellt wird. Dazu im **Views**-Menü der PACE-Hauptleiste den Menüpunkt **button board** auswählen. Es öffnet sich ein Fenster mit zwei Eingabezeilen, mit dem die Form des Button-Board festgelegt werden kann. Die bei Öffnen erscheinenden Vorgabewerte sind gerade die, welche hier gebraucht werden (3 Knöpfe in einer Zeile). Zur Erzeugung des ButtonBoard (Abb. 7.12) wird deshalb die Return-Taste gedrückt.


Abb. 7.12: Button Board direkt nach seiner Erzeugung

Das Fenster erhält den Namen **Options** (**mi.MT**, **relable as...**). Um die in Abb. 7.12 angegebenen Default-Namen der Knöpfe **label** durch die Namen **ein Wert**, **eine Kurve**, **alle Kurven** (Abb. 7.14) zu ersetzen, wird der Cursor auf einen der Knöpfe positioniert und das Menü des Knopf aufgerufen (**re.MT**). In dem angezeigten Menü den Menüpunkt **label** wählen und in das sich öffnende Eingabefenster den entsprechenden Namen eingeben. Dann die **Return**-Taste drücken.

Damit jeweils nur ein Knopf gedrückt ist, müssen beim Drücken eines Knopfs alle anderen gedrückten Knöpfe losgelassen werden. Dazu wird jedem der Knöpfe Code zugeordnet, der beim Drücken des Knopfs ausgeführt wird. Für den Knopf ganz links geht das z.B. wie folgt:

Den Cursor auf den Knopf stellen und erneut das Knopfmenü aufrufen (**re.MT**). Diesmal den Menüpunkt **block** auswählen. Es öffnet sich ein Block, in den der Code zum Löschen eingegeben wird. Nach der Eingabe sieht der Block wie folgt aus (Abb. 7.13):

**flag** enthält den aktuellen Knopfzustand. Ist **flag = true**, so ist der Knopf gedrückt. In diesem Fall wird den anderen beiden Knöpfen der Wert false zugewiesen, d.h. deren

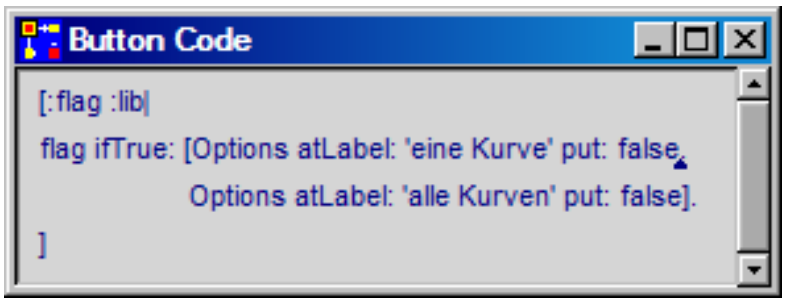



Abb. 7.13: Code zum Löschen von Buttons

Knöpfe werden gelöscht.

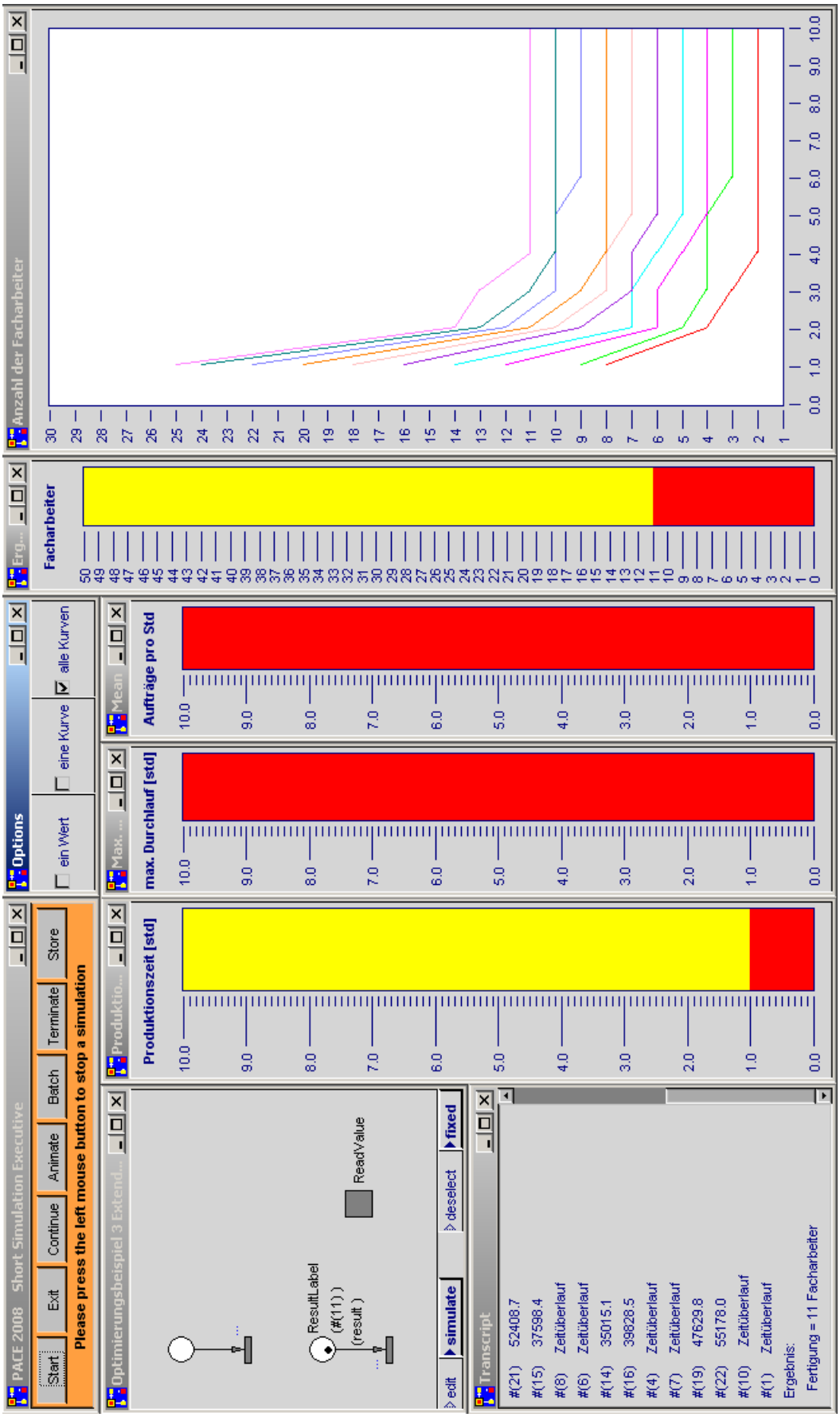



Abb. 7.14: Erweiterte Modelloberfläche

Um zu verhindern, dass durch erneutes Drücken eines schon gedrückten Knopfs mit der Maus der Knopf losgelassen wird, also überhaupt kein Knopf mehr gedrückt ist, wird die Control-Funktion des Knopfs geändert. Dazu wieder für jeden der Knöpfe das Knopfmenü aufrufen und diesmal **parameters** auswählen. Daraufhin wird das in Abb. 7.15 gezeigte Auswahlfenster angezeigt. Geändert wird die control function, indem **trigger** selektiert wird.

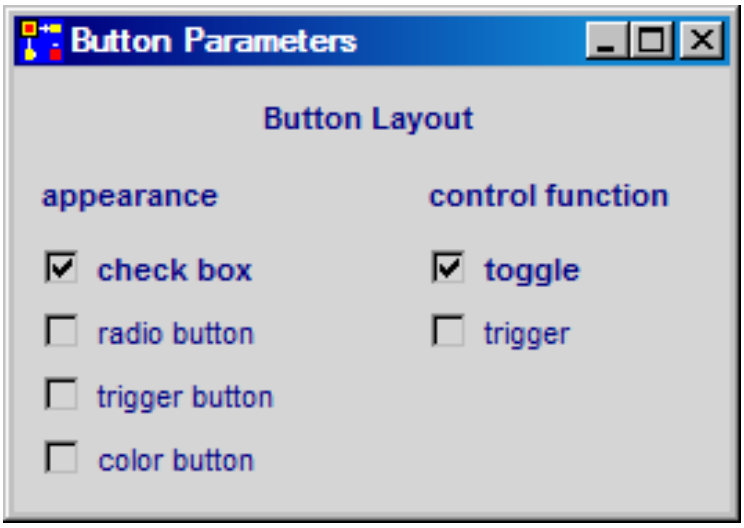
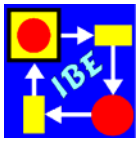



Abb. 7.15: Default-Layout eines Button

Hat man die weiteren beiden Knöpfe entsprechend bearbeitet, so ist das ButtonBoard fertiggestellt.

Abb. 7.14 zeigt "alle Kurven" für eine Auftragsbearbeitungszeit von einer Stunde. Man erkennt, dass die erforderliche Anzahl von Facharbeitern für kurze Antwortzeiten (maximale Bearbeitungszeiten - Bearbeitungszeit) unterhalb von einer Stunde stark ansteigt und erst langsam für Antwortzeiten von mehreren Stunden gegen den Wert konvergiert, den die übliche Rechnung mit den Mittelwerten der Exponentialverteilung liefert.




# 8. Was bleibt zu tun?

Wie das Beispiel zeigt, ist die Modellierung in PACE eine Kombination von Netzbeschreibung und Programmcode zur Steuerung der Abläufe in den Netzen. Für Inskriptionen und Extra-Codes werden normalerweise nur wenige Konstrukte von Smalltalk wie Zuweisung, bedingte Botschaft, usw. und einige spezielle Botschaften wie show:, value, value:, restart, usw. benötigt.

Während das Erstellen von Netzen meist relativ schnell erlernt wird, bereitet das Erstellen von Inskriptionen in Smalltalk häufig Probleme, insbesondere dann, wenn zuvor noch keine andere Programmiersprache bekannt ist. Deshalb wurde PACE eine Smalltalk-Fibel beigefügt, in der die erforderlichen Sprachkenntnisse beschrieben und mit zahlreichen Beispielen erläutert wurden. Es wird dringend empfohlen, Kap. 4 bis 6 der Smalltalk-Fibel durchzulesen, damit die Erstellung von Modellen zügig vonstatten gehen kann und dann auch Spaß macht.