UNIVERSITÄT DORTMUND

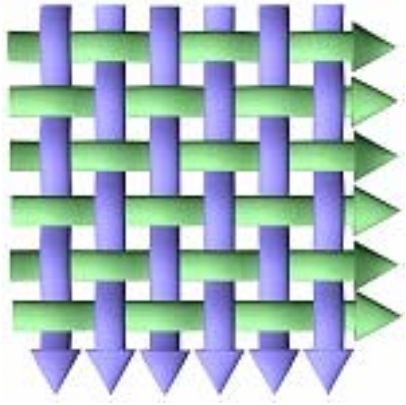
Technical Report 06002

ISSN 1612-1376

ProC/B-Editor

(Handbuch)

Andreas van Almsick, Jens Finzel, Michael Hierweck,
Jan Kriege, Mathias Schwenke



Sonderforschungsbereich 559
Modellierung großer Netze in der Logistik

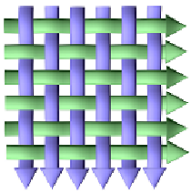
Universität Dortmund
44221 Dortmund

SFB 559 — Teilprojekt M1
LS Informatik IV
Universität Dortmund
28. September 2006
Version 1.1

Sonderforschungsbereich 559

Modellierung großer Netze in der Logistik

Bericht Nr. 06002



ProC/B-Editor

Handbuch

SFB 559 — Teilprojekt M1

Michael Hierweck, Jens Finzel, Andreas van Almsick,
Jan Kriege, Mathias Schwenke

Inhaltsverzeichnis

Kapitel 1

Einführung

Mit der Version 1.1 steht Ihnen das sechste offizielle Release des ProC/B-Editors zur Verfügung. Dieses Tool unterstützt Sie bei der Modellierung logistischer Prozesse anhand des ProC/B-Modellformalismus.

Im Folgenden setzen wir voraus, dass Ihnen zum einen der ProC/B-Modellformalismus ([2, 3]) bekannt ist und dass Sie dazu in der Lage sind, die grafische Oberfläche Ihres Computersystems mit Maus und Tastatur zielgerichtet zu bedienen. Erfahrungen im Umgang mit anderen Softwarelösungen sind sicherlich hilfreich, jedoch nicht erforderlich. Um es noch einmal zu hervorzuheben: Dieses Dokument kann und soll keine Einführung in die grundlegende Verwendung Ihrer Arbeitsumgebung darstellen. Wenn Sie aber mit deren Umgang vertraut sind, sollten Sie nach dem Studium dieses Dokuments und dem praktischen Nachvollziehen der Anwendungsbeispiele problemlos Prozessketten modellieren können.

In der Praxis führen häufig mehrere Wege zum Ziel. Wir werden nicht jede Funktion des ProC/B-Editors explizit beschreiben, sondern uns auf grundlegende Funktionen beschränken. Oftmals können Funktionen auf verschiedene Weise ausgelöst werden, z.B. durch Verwendung von Buttons, Tastenkombinationen oder (kontextsensitiven) Menüs. Wir werden zugunsten der Lesbarkeit und, um die Dokumentation nicht unnötig umfangreich werden zu lassen, nicht immer alle Möglichkeiten aufführen. Anhand der Anwendungsbeispiele werden wir jedoch verschiedene Wege und Möglichkeiten der Handhabung demonstrieren, die uns zweckmäßig erscheinen. Oftmals hilft auch die Online-Hilfe, die sich hinter den zahlreichen Hilfe-Schaltflächen verbirgt, weiter. Wichtige Hinweise zu einzelnen Prozesskettenelementen sind jeweils kursiv hervorgehoben.

An dieser Stelle möchten wir Ihnen herzlich für den Einsatz des ProC/B-Editors danken, und wünschen Ihnen viel Freude bei seiner Anwendung. Dank gilt auch den Projektgruppen 279 und 299, ohne deren Tool DOGGE PAX der Entwicklungsprozess erheblich langwieriger und aufwändiger gewesen wäre.

Dortmund, 28. September 2006

Kapitel 2

Installation

Dieses Kapitel beschreibt die Installation des ProC/B-Editors. Diese unterscheidet sich im Gegensatz zur Anwendung des Programms je nach verwendetem Betriebssystem erheblich. Jeder unterstützten Plattform ist im folgenden ein eigenständiger Abschnitt gewidmet.

2.1 Windows 95, 98, ME, NT 4.0 und XP

Der ProC/B-Editor für Windows-Betriebssysteme wird als Zip-Archiv ausgeliefert. Der ProC/B-Editor solte dann auf der Festplattenpartition c: im Verzeichnis
c:\Programme\procb-editor
installiert werden. Nach der Installation kann das Programm procb-editor.exe ohne vorherigen Neustart aus dem Verzeichnis heraus aufgerufen werden.

2.2 SUN OS

Das Programm wird noch nicht als Binary für SunOS ausgeliefert.

2.3 Linux

Die Dateien sind in einem eigenen Unterverzeichnis (z.B. in /usr/local/procb-editor) abzulegen. Die mitgelieferten dynamischen Libraries sind in einem der Standardverzeichnisse für Libs zu installieren.

Um das Handbuch über das Hilfemenü aufrufen zu können, muss sich der Adobe Acrobat Reader mit “acroread” aufrufen lassen.

2.4 Benutzerspezifische Verzeichnisse / Umgebungsvariablen

Beim ersten Programmstart legt der Editor im HOME-Verzeichnis die Datei procbprefs_*versionsnummer*.txt (also z.B. procbprefs_1.1.txt) an, sofern sie noch nicht existieren sollte. In dieser Datei werden die vom Editor verwendeten benutzerspezifischen Pfade verwaltet. Im einzelnen handelt es sich hierbei um das Arbeitsverzeichnis, das Verzeichnis der Aggregatdatenbank und das Verzeichnis für temporäre Dateien.

Die Pfade lassen sich entweder per Texteditor ändern oder, sofern vorhanden, mit dem ProC/B-GUI.

Beim Erstellen der Datei wertet der Editor folgende Umgebungsvariablen aus

PAXARBEITSVERZEICHNIS (UNIX-Notation!)

HOME

HOMEDRIVE,

HOMEPATH

PWD werden mit in der Reihenfolge sinkender Priorität ausgewertet und legen das Arbeitsverzeichnis (für Modelle und Druckdateien) fest.

AGGTOOLVERZEICHNIS legt den Pfad für die Aggregatdatenbank fest.

TEMP

TMP werden mit in der Reihenfolge sinkender Priorität ausgewertet und legen das Verzeichnis für temporäre Dateien fest.

Kapitel 3

ProC/B-Editor

Dieses Kapitel beschreibt die Handhabung des ProC/B-Editors. Sie erlernen das Anlegen, Speichern, Öffnen und Drucken der Modelldateien sowie die Verwendung der Zeichenfläche. Einige Modelle, auf die im Weiteren Bezug genommen wird, befinden sich im Ordner Beispielmodelle im Editorverzeichnis.

3.1 Das Hauptfenster

Nach dem Start des ProC/B-Editors erscheint das Hauptfenster des ProC/B-Editors. Nach dem Laden eines Modells bietet das Hauptfenster eine hierarchische Übersicht über die aktuell geöffnete Modelldatei (s. Abbildung 3.1) sowie Zugriff auf Funktionen zur Handhabung von Modelldateien.

3.1.1 Hierarchische Übersicht

In der hierarchischen Übersicht werden die Strukturen der Modelle übersichtlich dargestellt. In der Abbildung 3.1 existieren drei eigenständige Modelle. Das erste Modell besitzt mit “Produktionsstaette” und “Spedition” zwei hierarchisch untergeordnete Modellteile. Diese werden, wie später genauer erläutert, durch Funktionseinheiten repräsentiert.

Hinter jedem Quadrat verbirgt sich eine Funktionseinheit. Die Zeichenflächen dieser Funktionseinheiten kann direkt über ein kontextsensitives Menü (PopUp-Menü, linke Maustaste) zum Betrachten und Editieren geöffnet werden. Die Zahlen innerhalb der Symbole stehen für die Nummer des geöffneten Arbeitsfensters. Standard-Funktionseinheiten wie Server, Storages oder Counter werden durch Kreise symbolisiert.
Die hierarchische Struktur ist nicht, wie hier dargestellt, auf zwei Ebenen beschränkt, sondern kann durchaus tiefer geschachtelt werden.

3.1.2 Der Menüpunkt “Datei”

Der Menüpunkt “Datei” stellt Funktionen zum Anlegen, Öffnen, Speichern und Drucken von Modelldateien zur Verfügung.

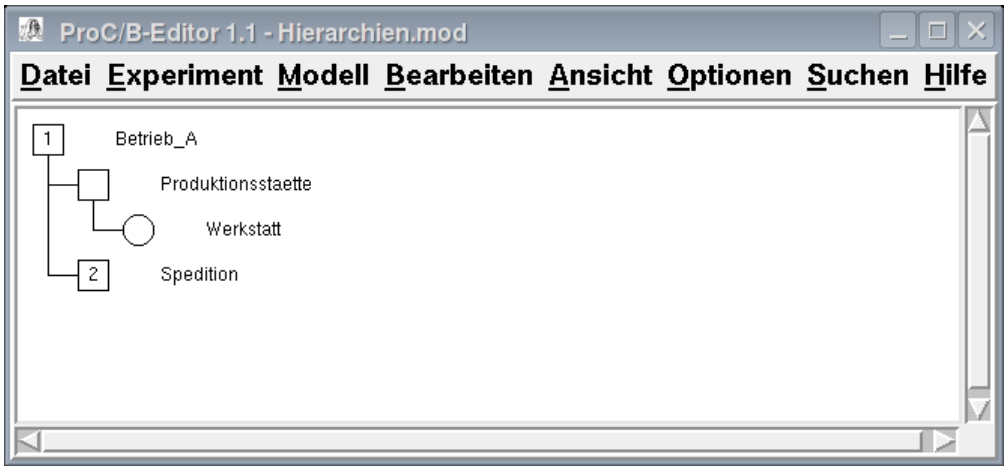


Abbildung 3.1: Das ProC/B-Editor-Hauptfenster

Neu Durch Auswahl dieses Menüpunktes wird eine neue leere Modelldatei erzeugt und geladen. Wurde die bisher geladene Modelldatei verändert, aber noch nicht gespeichert, so erscheint eine Sicherheitsabfrage, ob das aktuell geladene Modell durch das neue Modell ersetzt werden soll.

Öffnen... Dieser Menüpunkt ermöglicht das Öffnen einer zuvor gespeicherten Modelldatei. Es wird ein Dialogfenster zur Auswahl von Pfad und Dateinamen angezeigt. Modelldateien besitzen standardmäßig die Endung ".mod". Die Abbildung 3.1 zeigt das Hauptfenster nach dem Öffnen der Modelldatei "Hierarchien.mod". Falls die Version der ausgewählten Modelldatei nicht zu der Version des Editors passt, erscheint ein Dialog, über den die Datei vor dem Öffnen konvertiert werden kann.

Speichern Durch Auswahl dieses Menüpunktes wird die aktuelle Modelldatei gespeichert. Falls noch kein Dateiname für die Modelldatei angegeben wurde, so ist die Funktionalität mit der des Menüpunktes "Speichern unter..." identisch.

Speichern unter... Dieser Menüpunkt bietet ebenfalls die Möglichkeit, das aktuelle Modell zu speichern. Allerdings wird zuvor ein Dialogfenster zur Auswahl eines Pfades sowie eines Dateinamens geöffnet. Modelldateien besitzen standardmäßig die Endung ".mod". So besteht beispielsweise die Möglichkeit, versuchsweise Änderungen an einem Modell unter einem anderen Dateinamen als Kopie zu speichern, so dass das Original unverändert bleibt.

Aggregattyp Öffnen... Dieser Menüpunkt erlaubt das Laden von Aggregattypen. Es wird ein Dialogfenster zur Auswahl einer Datei aus der Aggregatdatenbank angezeigt.

Aggregattyp speichern unter... Nachdem ein geeigneter Aggregattyp für die Aggregierung definiert wurde, ermöglicht dieser Menüpunkt das Speichern des Aggregattyps in die Datenbank.

Drucken... Dieser Menüpunkt bietet Zugriff auf die Möglichkeit, Modelle als PostScript-Datei zu exportieren und auszudrucken. Zunächst wird nach einem Dateinamen gefragt, unter dem die verschiedenen Ansichten des Modells als Grafik (encapsulated Postscript) gespeichert werden sollen. Dabei wird für jede Zeichenflächen durch Anhängen einer hierarchischen Nummerierung an den Dateinamen eine eigene Datei erzeugt. Außerdem besteht die Möglichkeit, festzulegen, in welcher Größe die einzelnen Zeichenflächen gedruckt werden sollen: Alle Zeichenflächen können auf DIN-A4-Größe skaliert oder in einer einheitlichen Größe mit Angabe eines Zoom-Faktors ausgedruckt werden. Anschließend besteht die Möglichkeit, einen Drucker auswählen, auf dem diese Dateien direkt ausgedruckt werden können. (Die Auswahl des Druckers funktioniert nur auf UNIX-Betriebssystemen!)

Konsistenzprüfung durchführen... Über diesen Menüpunkt können Konsistenzprüfungen gestartet werden. Diese Prüfungen unterstützen den Nutzer beim Auffinden von Modellierungsfehlern. Außerdem kann das Modell auf Hinweise für nichtstationäres Verhalten untersucht werden. Eine ausführlichere Beschreibung der Konsistenzprüfungen findet sich in Anhang A.

Log-Datei laden... Während der Arbeit mit dem Editor werden alle durchgeführten Aktionen in einer Log-Datei gespeichert (procb-editor.log im Tempverzeichnis). Diese Log-Datei kann bei aufgetretenen Fehlern für Debugging-Zwecke genutzt werden. Über diesen Menüpunkt lässt sich eine ältere Log-Datei laden.

Beenden Die Auswahl dieses Menüpunkts beendet den ProC/B-Editor. Wurde die aktuell Modelldatei nicht gespeichert, so fragt der Editor, ob das Modell gespeichert werden soll.

3.1.3 Der Menüpunkt “Experiment”

Über den Menüpunkt “Experiment” können Experimentserien verwaltet werden.

Experiment laden... Experimente, die in vorangegangenen Sitzungen spezifiziert wurden, können wieder eingeladen werden.

Experiment speichern Durch diesen Menüpunkt wird Ihnen die Möglichkeit geboten, Spezifikationen von Experimenten zu speichern.

Experiment speichern unter... Dieser Menüpunkt ist analog zu dem vorherigen Menüpunkt “Experiment speichern”, erlaubt jedoch die Eingabe eines neuen Namens, unter dem das Experiment gespeichert werden kann.

Exportieren nach ProC/B Über diesen Menüpunkt wird DoggeToProC/B gestartet und aus Modell- und Experimentbeschreibung wird eine procb-Datei erzeugt.

Experimentoptionen... In dem Dialog, der nach Auswahl dieses Menüpunkts erscheint, können Modellzeit, Seed und Kalkulationszinssatz für das Experiment eingestellt werden.

Experiment schließen Hiermit wird die Experimentdarstellung beendet und der Editor wechselt zurück in die Modellierungsansicht.

3.1.4 Der Menüpunkt “Modell”

Der Menüpunkt “Modell” ermöglicht das Anlegen neuer Modelle innerhalb der geöffneten Modelldatei.

Neu Dieser Menüpunkt erzeugt eine Funktionseinheit auf oberster Ebene der Hierarchie. Anschließend wird die zugehörige Zeichenfläche angezeigt.

3.1.5 Der Menüpunkt “Bearbeiten”

Dieser Menüpunkt bietet unterstützende Funktionen bei der Bearbeitung von Modellen.

Rückgängig Nach der Auswahl dieses Eintrags wird ein Fenster geöffnet, welches die letzten Bearbeitungsschritte auflistet. Es besteht die Möglichkeit, den letzten oder mehrere Arbeitsschritte rückgängig zu machen. Dieser Menüpunkt ist im Hauptfenster angesiedelt, da sich die Rücknahmen der letzten Aktionen auf verschiedene Unterfenster auswirken kann.

Wiederherstellen Hinter dieser Funktion verbirgt sich das Gegenstück zur “Rückgängig”-Funktion. Versehentlich rückgängig gemachte Arbeitsschritte können hier wiederhergestellt werden.

3.1.6 Der Menüpunkt “Ansicht”

Über diesen Menüpunkt können Aktionen, die die Darstellung des Modells betreffen, ausgeführt werden.

Externe Funktionseinheiten anzeigen Bei Auswahl dieses Menüpunkts, werden die externen Funktionseinheiten im Hierarchiebaum dargestellt.

Entscheidungselemente expandieren Über diesen Menüpunkt kann festgelegt werden, ob die Entscheidungselemente in dem Modell expandiert oder komprimiert dargestellt werden sollen (siehe auch Anhang C).

3.1.7 Der Menüpunkt “Optionen”

Standardwerte ändern Über diesen Menüpunkt können für die einzelnen Elemente des ProC/B-Modellformalismus Standardwerte festgelegt werden.

3.1.8 Der Menüpunkt “Suchen”

Dieser Menüpunkt bietet eine Suchfunktion für Knoten. Nach Auswahl des Knotentyps wird ein Fenster, ähnlich den später beschriebenen Fenstern zur Festlegung von Attributen, geöffnet. Es besteht die Möglichkeit, die zu suchenden Objekte durch Festlegung relevanter Attribute und derer Werte zu beschreiben.

3.1.9 Der Menüpunkt “Hilfe”

Neben der kontextsensitiven Online-Hilfe bietet dieses Menü weitere Informationen zum ProC/B-Editor.

Hilfe... Diese Funktion öffnet ein Fenster, in dem Hilfetexte zu den einzelnen Menüpunkten angezeigt werden.

Handbuch Diese Funktion öffnet das Handbuch. Es bietet umfangreiche Information zum Umgang mit von DOGGE PAX erzeugten Editoren, wie dem ProC/B-Editor.

Hilfe zur Hilfe Es wird ein Fenster geöffnet, in welchem erklärt wird, wie Hilfe zu speziellen Themen gefunden werden kann.

Info zu DoggePax Es wird ein Fenster mit Informationen über das Tool DOGGE PAX auf dem der ProC/B-Editor basiert angezeigt.

Info zum ProC/B-Editor Es wird ein Fenster mit Informationen über den ProC/B-Editor angezeigt.

3.2 Die Zeichenfläche (Oberste Ebene)

Bei der Zeichenfläche handelt es sich um das zentrale Werkzeug, mit dem Modelle editiert werden können. Die Abbildung 3.2 zeigt eine Prozesskette auf oberster Hierarchieebene.

3.2.1 Verwendung der Zeichenfläche

Öffnen der Zeichenfläche Die Zeichenfläche wird beim Öffnen einer Funktionseinheit (s. z. B. Abschnitt 3.1.1) oder beim Anlegen einer neuen Prozesskette (s. Abschnitt 3.1.4) geöffnet.

Anlegen von Knoten und Kanten Die Verwendung der Zeichenfläche erfolgt sehr intuitiv. In der Regel wird man zunächst einen Knotentypen unter den in der Buttonleiste angezeigten Möglichkeiten auswählen. Dieser kann dann per Mausklick mit der linken Maustaste auf der Fläche frei platziert werden. Sagt einem die Position jetzt oder später einmal nicht mehr zu, so kann der Knoten per “Klicken und Ziehen” mit der linken Maustaste (Drag&Drop) nach belieben verschoben werden.

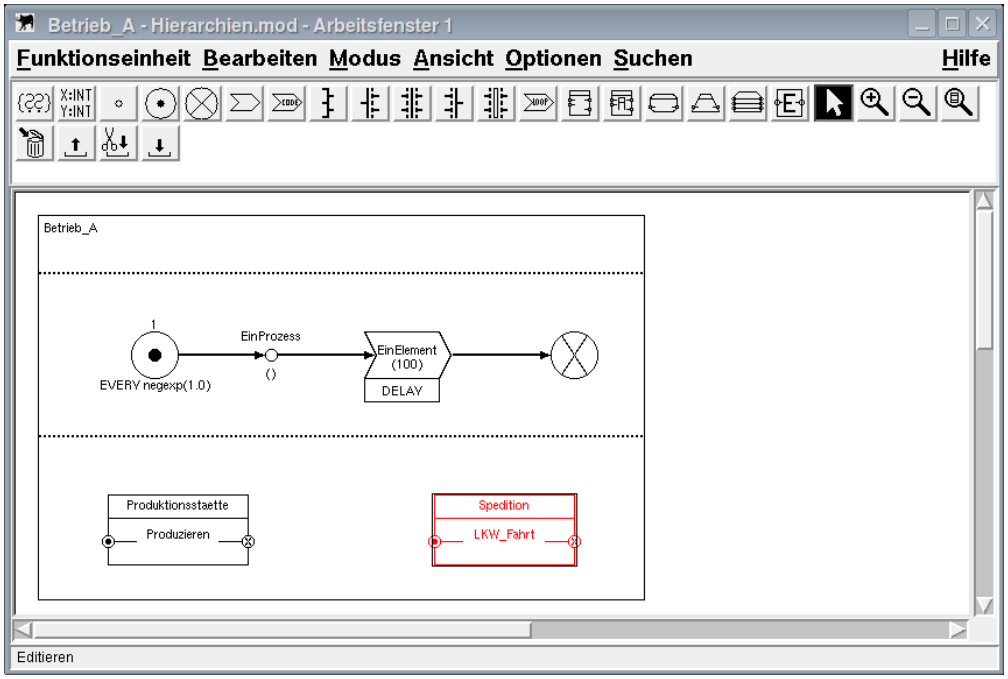


Abbildung 3.2: ProC/B-Editor-Arbeitsfenster

Falls die Rasterung (3.2.6) aktiviert ist (Standard), so werden alle Elemente auf einem Raster ausgerichtet, um so das Bemühen um ein ordentliches Erscheinungsbild zu unterstützen.

Nachdem man wenigstens zwei Knoten plaziert hat, ist es möglich diese, falls durch den Proc/B-Modellformalismus erlaubt, mit einer Kante zu verbinden. Dazu klickt man mit der rechten Maustaste auf den Startknoten der Kante. Die Kante wird dann sichtbar. Anschließend klickt man mit der rechten Maustaste auf den Zielknoten. Ist die Verbindung erlaubt, so wird die Kante gezogen. Ansonsten wird die Kante verworfen und der Fehler mit einer entsprechenden Meldung quittiert. Kanten verlaufen automatisch auf der kürzesten direkten Verbindung zwischen den Knoten. Allerdings besteht während des Ziehens der Kante die Möglichkeit, per rechtem Mausklick Stützpunkte für die Kante festzulegen.

Kanten sind grundsätzlich gerichtet. Wenn die Verbindung von Knotentyp A und Knotentyp B in dieser Richtung erlaubt ist, so gilt dies für die Gegenrichtung i.A. nicht. Manche Knoten können nur eine begrenzte Anzahl ein- und ausgehender Kanten anbinden.

Knoten können durch Klick mit der linken Maustaste markiert werden. Es besteht die Möglichkeit, durch Auswahl eines Bereichs der Zeichenfläche mit der linken Maustaste mehrere Knoten und Kanten gleichzeitig zu markieren.

Verborgene Kanten Verborgene Kanten dienen der Verdeutlichung und Modellierung der Anbindung von Ressourcen (Funktionseinheiten, Server, Counter u.ä.) an deren Nutzer. Hierbei handelt es sich in erster Linie um Prozesskettenelemente, aber auch um Quellen und Senken. Diese Kanten werden im ProC/B-Modellformalismus nicht erwähnt und gehören nicht zum eigentlichen Modell. Sie werden nur für interne Zwecke und zur Darstellung der Anbindungen verwendet.

Verborgene Kanten werden genau dann angezeigt, wenn sie auf einem aktuell markierten Knoten beginnen oder enden.

Löschen von Knoten und Kanten Markierte Knoten und Kanten können durch Drücken der Entfernen-Taste oder durch Klick auf den Papierkorb-Button gelöscht werden. Kanten, die an einem gelöschten Knoten angebunden waren, werden automatisch mitentfernt. Versehentlich gelöschte Objekt können über die “Rückgängig”-Funktion (s. 3.1.5) wiederhergestellt werden.

3.2.2 Der Menüpunkt “Funktionseinheit”

Bezeichner... An dieser Stelle kann ein Name für die im Arbeitsfenster angezeigte Funktionseinheit eingegeben werden. Dieser wird anschließend auch in der Titelleiste des Fensters, als Name in der umgebenden Funktionseinheit, sowie in der hierarchischen Darstellung des Hauptfensters angezeigt.

Arbeitsflächengröße... Hier kann der Umfang der logischen Zeichenfläche geändert werden. Die Größe der Zeichenfäche wird standardmäßig automatisch angepasst und ist i.d.R. ausreichend. Sollte ein Teilmodell so groß werden, dass die voreingestellte Zeichenfäche nicht mehr aussreicht, so kann sie über diesen Menüpunkt vergrößert werden.

Drucken... Der Menüpunkt bietet Zugriff auf eine Druckfunktion, ähnlich der des Hauptfensters. Nach Wunsch kann der Inhalt der aktuellen Zeichenflächen oder der aktuell sichtbare Bereich gedruckt werden.

Schließen... Hiermit kann das Zeichenfenster geschlossen werden.

Konsistenzprüfung durchführen... Über diesen Menüpunkt können Konsistenzprüfungen für einzelne Funktionseinheiten gestartet werden. Eine ausführlichere Beschreibung der Konsistenzprüfungen findet sich in Anhang A.

3.2.3 Der Menüpunkt “Bearbeiten”

Rückgängig Dieser Menüpunkt hat dieselbe Funktionalität wie die Rückgängig-Funktion, die aus dem Hauptfenster aufgerufen wird.

Wiederherstellen Dieser Menüpunkt hat dieselbe Funktionalität wie die Wiederherstellen-Funktion, die aus dem Hauptfenster aufgerufen wird.

Ausschneiden Dieser Menüpunkt kopiert die markierten Elemente in die Zwischenablage und löscht sie auf der Zeichenfläche. Diese Funktion steht auch über den Button “Ausschneiden” zur Verfügung.

Kopieren Dieser Menüpunkt kopiert die markierten Elemente in die Zwischenablage. Diese Funktion steht auch über den Button “Kopieren” zur Verfügung.

Einfügen Dieser Menüpunkt fügt die in der Zwischenablage befindlichen Elemente auf der Zeichenfläche ein. Diese Funktion steht auch über den Button “Einfügen” zur Verfügung.

Löschen Dieser Menüpunkt löscht die markierten Elemente auf der Zeichenfläche. Diese Funktion steht auch über den Button “Löschen” zur Verfügung.

Auswählen... Diese Funktion öffnet ein Fenster, in dem angegeben werden kann, welcher Knoten- bzw. Kantentyp markiert werden soll.

Auswahl exportieren... Dieser Menüpunkt kopiert die markierten Elemente in eine anzugebende Datei.

Importieren aus... Dieser Menüpunkt fügt die in einer anzugebenden Datei befindlichen Elemente auf der Zeichenfläche ein.

Aggregieren von... *Dieser Menüpunkt steht nur in der Linux- und Sun-Version des ProC/B-Editors zur Verfügung.*
An dieser Stelle erfolgt nur eine kurze Erläuterung der Eingabefelder des Aggregierungsdialogs. Weiterführende Literatur zur Aggregierung finden Sie in [4] und [8].
Um eine Aggregierung durchzuführen, muss zunächst eine Funktionseinheit oder ein Modellteil selektiert und danach die Funktion “Aggregieren von..” ausgewählt werden. Eine Eingabemaske erscheint (s. Abbildung 3.3) und bietet die Möglichkeit die Startparameter und eine Aggregierungstechnik auszuwählen.

- Aggregatname: Mit diesem Eingabefeld wird der Name des Aggregats festgelegt.
- Aggregattyp: Legt den Aggregattyp für die Aggregierung fest. Der Aggregattyp kann entweder direkt eingegeben werden oder über den Button “Setzen” ausgewählt werden. Wenn kein Aggregattyp angegeben wird, werden die Ergebnisse in einem Aggregatbaustein geladen.
- Alternative Aggregierungssoftware: Dieses Eingabefeld dient zur Auswahl der Aggregierungssoftware bei der alternativen Technik.
- Alternatives Auswertungsscript: Dieses Eingabefeld dient zur Auswahl des Auswertungsscripts bei der alternativen Technik.

- Parameter für Aggregierungssoftware: Sie können beim Aufruf des Aggregierungstools beliebig viele Parameter übergeben. Die Listbox ’Parameter für Aggregierungstools’ verwaltet die Parameter. Mit Hilfe der Buttons ’Einfügen’ und ’Löschen’ können Sie den eingegebenen Parameter in die Listbox aufnehmen bzw. entfernen.

- Pop. von ’dienstname’: Dieses Eingabefeld gibt die Populationsgrenze für den Dienst ’dienstname’ an.

- Trans. von ’dienstname’ (nur bei PN-Aggregierung): Mit diesem Eingabefeld können Sie die Mess-Transition im Petri-Netz zur Ermittlung des Durchsatzes von Dienst ’dienstname’ festlegen.

- Anzahl der Modi (nur bei PN-Aggregierung): Legt die Anzahl der Modi in der Approximation für die IS-Semantik beim Delay-PKE fest.

- Token-Belegung (nur bei PN-Aggregierung): Legt die initiale Token-Belegung von Stellen fest, die eine Kurzschlussstelle beschreiben.

- Analyse-Tool (nur bei PN-Aggregierung): Legt die Analysetechnik für die Aggregatberechnung fest. Mögliche Eingaben sind ’supgspn’ und ’simulator’.

- Ausgabeformat (nur bei PN-Aggregierung): Hiermit können Sie festlegen, ob die Aggregierungsergebnisse in einem Aggregatbaustein (’HIT’) oder in der unter ’Aggregattyp’ angebenen Ersatzdarstellung (’Normal’) gespeichert werden.

Mit Hilfe der Checkbuttons im unteren Bereich des Fensters können Sie die Aggregierungstechnik festlegen. Die Aggregierung wird durch Drücken des Buttons ’OK’ gestartet.
Nach der Berechnung des Aggregats und der Erzeugung der Ersatzdarstellung werden Sie gebeten, die Ersatzdarstellung durch einen Mausklick zu platzieren. Das aggregierte Modell sollte gespeichert werden, um eine spätere Aggregatuntersuchung zu ermöglichen. Die Verbindungen zwischen Ersatzdarstellung und dem Rest des Modells müssen zur Zeit noch manuell wiederhergestellt werden.

Aggregat importieren... Diese Funktion öffnet einen Auswahldialog, um ein zuvor bereits aggregiertes Modell zu laden.

Attribute... Hiermit wird ein Fenster zur Eingabe von Attributen für Elemente geöffnet. Dieses Fenster kann auch durch einen Klick mit der mittleren Maustaste auf das jeweilige Element erreicht werden.

Attribute drucken... Über diese Funktion können die Attribute aller ausgewählten Elemente gedruckt werden.

Öffnen Über diesen Menüeintrag kann die Innenansicht eines Knotens vom Typ Funktionseinheit geöffnet werden. Als Folge wird ein neues Zeichenfenster geöffnet, welches die Prozessketten der Funktionseinheit darstellt.

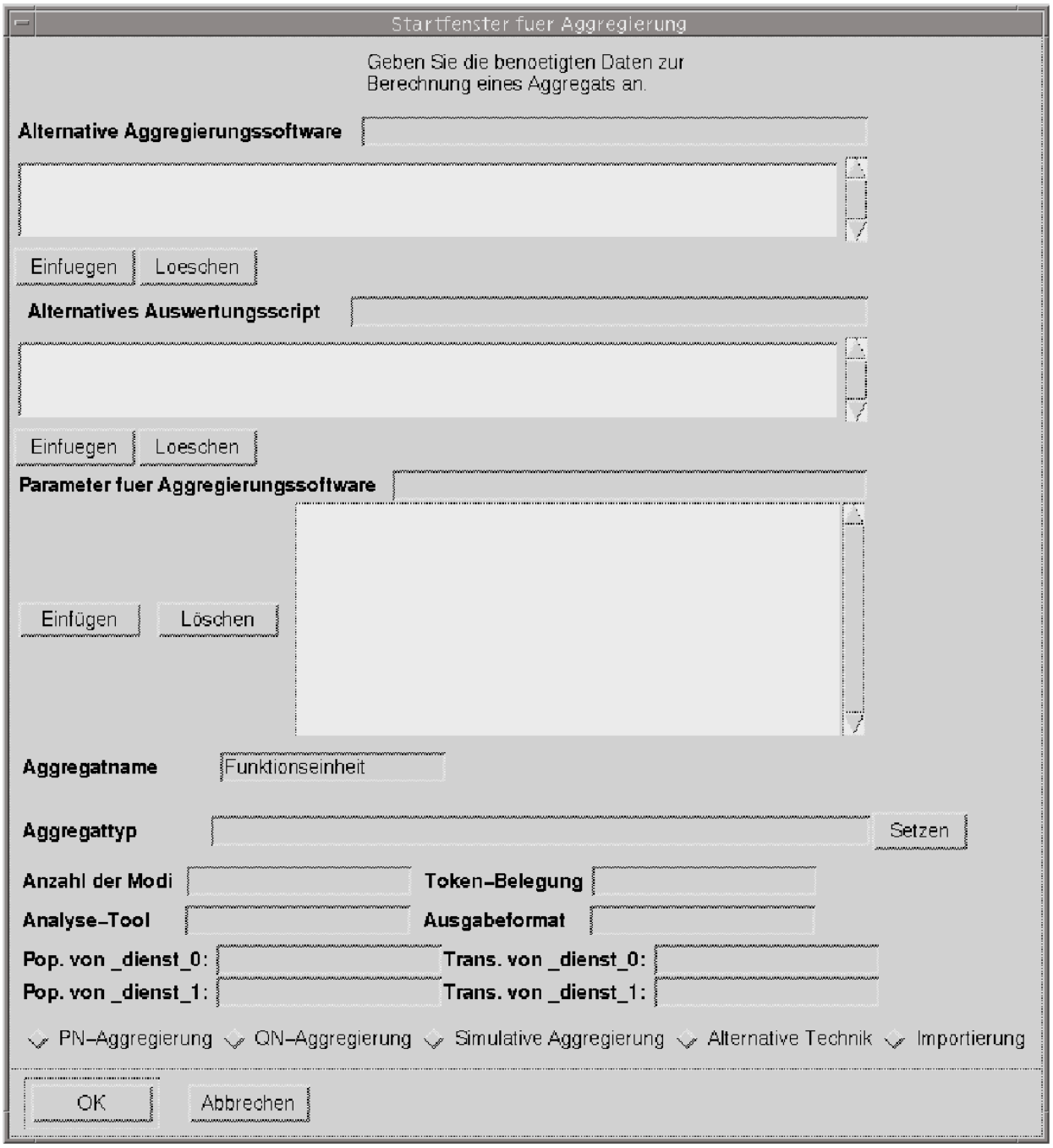


Abbildung 3.3: Dialogfenster zum Erzeugen von Aggregaten

Inhalt anbinden... Über diese Funktion kann der Inhalt der Zeichenfläche eines geöffneten Zeichenfensters in eine Funktionseinheit übertragen werden. In der Hierarchieübersicht (s. 3.1.1) ist der hierarchische Aufbau des Modells zu erkennen. Die Funktion 'Inhalt anbinden' dient dazu, zwei Bäume zu vereinigen, indem der Wurzelknoten eines Baumes über eine Funktionseinheit in einen anderen Baum eingehängt wird. Bevor der Inhalt eines geöffneten Fensters an eine Funktionseinheit angebunden werden kann, muss zunächst der Inhalt dieser Funkionseinheit gelöscht werden (s. 3.2.3). Existieren mehrere geöffnete Fenster, die als Quelle dienen können, so erscheint ein Fenster zur Auswahl der Quelle.

Inhalt löschen... Die Funktion, die sich hinter diesem Menüpunkt verbirgt, löscht den Inhalt einer Funktionseinheit. Der Knoten Funktionseinheit bleibt jedoch bestehen.

3.2.4 Der Menüpunkt "Modus"

Das Verhalten der Maus hängt davon ab, welcher Modus über die Buttonliste oder über dieses Menü eingestellt wurde. Im Editiermodus lassen sich Knoten markieren und verschieben, mit einem Modus für die Knotentypen ein Knoten des jeweiligen Typs anlegen.

Editieren Dieser Menüpunkt schaltet in den Editiermodus zurück. In diesem Modus ist es möglich, Knoten zu markieren, zu verschieben sowie mit der linken Maustaste einen Rahmen um eine Menge von Objekten zu ziehen. Diese Funktion steht auch über den Button "Editieren" zur Verfügung.

Kommentar, Variablen, ProzessID, ... Dieser Menüpunkt schaltet in den Knotenerzeugungsmodus. Ein Knoten des gewählten Typs wird beim nächsten Klick auf die Zeichenfläche an der entsprechenden Position erzeugt. Diese Funktion steht auch über die Buttons für die jeweiligen Knotentypen zur Verfügung.

3.2.5 Der Menüpunkt "Ansicht"

Zoom + Dieser Menüpunkt vergrößert die Darstellung auf der Zeichenfläche, um den als Zoomschrittweite spezifizierten Wert (s. 3.2.6). Die markierten Knoten werden in der Ansicht zentriert. Diese Funktion steht auch über den Button "Vergrößerungsglas +" zur Verfügung.

Zoom - Dieser Menüpunkt verkleinert die Darstellung auf der Zeichenfläche, um den als Zoomschrittweite spezifizierten Wert (s. 3.2.6). Die markierten Knoten werden in der Ansicht zentriert. Diese Funktion steht auch über den Button "Vergrößerungsglas -" zur Verfügung.

Zoom: Komplettansicht der FE Dieser Menüpunkt verkleinert oder vergrößert die Darstellung auf der Zeichenfläche so, dass die komplette Funktionseinheit im Fenster dargestellt wird. Diese Funktion steht auch über den Button "Vergrößerungsglas" zur Verfügung.

Zoomfaktor... In dem zugehörigen Fenster lässt sich der Zoomfaktor manuell einstellen. Einstellbar sind Prozentwerte zwischen 10 und 250.

3.2.6 Der Menüpunkt “Optionen”

Dieser Menüpunkt verbirgt Einstellungen für Funktionen, die das Editieren erleichtern sollen.

Rastern Hiermit wird die Rasterung ein- bzw. ausgeschaltet. Knoten, die neu angelegt oder verschoben werden, werden an dem unter 3.2.6 eingestellten Raster ausgerichtet.

Rastergröße... Hiermit kann die Granualität des Rasters eingestellt werden. Einstellbar sind Werte zwischen 2 und 100.

Zoom-Schrittweite Hier lässt sich der Prozentsatz einstellen, um den beim Zoomen vergrößert oder verkleinert wird. Einstellbar sind Prozentwerte zwischen 1 und 20.

Standardwerte ändern Über diesen Menüpunkt können für die einzelnen Elemente des ProC/B-Modellformalismus Standardwerte festgelegt werden.

3.2.7 Der Menüpunkt “Suchen”

Dieser Menüpunkt bietet eine Suchfunktion für Knoten, welche auch im Hauptfenster verfügbar ist. Nach Auswahl des Knotentyps wird ein Fenster, ähnlich den später beschriebenen Fenstern zur Festlegung von Attributen, geöffnet. Es besteht die Möglichkeit, die zu suchenden Objekte durch Festlegung relevanter Attribute und deren Werte zu beschreiben.

3.2.8 Der Menüpunkt “Hilfe”

Neben der kontextsensitiven Online-Hilfe bietet dieses Menü weitere Informationen zum ProC/B-Editor.

Hilfe... Diese Funktion öffnet ein Fenster, in dem Hilfetexte zu den einzelnen Menüpunkten angezeigt werden.

Handbuch Diese Funktion öffnet das Handbuch. Es bietet umfangreiche Information zum Umgang mit von DOGGE PAX erzeugten Editoren, wie dem ProC/B-Editor.

Hilfe zur Hilfe Es wird ein Fenster geöffnet, in welchem erklärt wird, wie Hilfe zu speziellen Themen gefunden werden kann.

Info zu DoggePax Es wird ein Fenster mit Informationen über das Tool DOGGE PAX auf dem der ProC/B-Editor basiert angezeigt.

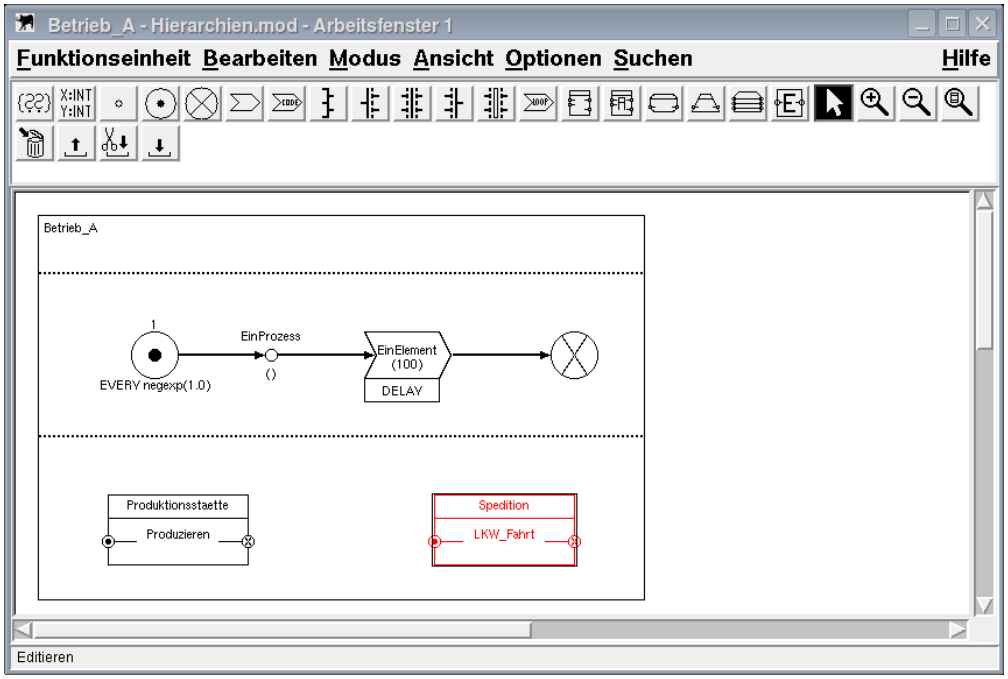


Abbildung 3.4: Das ProC/B-Editor-Arbeitsfenster

Info zum ProC/B-Editor Es wird ein Fenster mit Informationen über den ProC/B-Editor angezeigt.

3.3 Die Zeichenfläche (Untere Ebenen)

Bei der Zeichenfläche handelt es sich um das zentrale Werkzeug, mit dem Modelle editiert werden können. Abbildung 3.4 zeigt einen Prozess auf einer unteren Hierarchieebene.

Wie später erläutert wird, besitzen hierarchische Elemente (hier: Funktionseinheiten) Innendarstellungen. Diese können über entsprechende Fenster editiert und betrachtet werden. Die Handhabung dieser Fensters entspricht im Wesentlichen der Bedienung der Zeichenfläche auf oberster Ebene.

3.3.1 Virtuelle Quellen und Senken

Als zusätzliche Elemente stehen in der Innendarstellung von Funktionseinheiten die Ports virtuelle Quelle und virtuelle Senke (als Durchreiche) zur Verfügung, welche wie Knoten handhabbar sind.

Kapitel 4

ProC/B-Elemente

4.1 Quelle

Eine **Quelle** (siehe Abbildung 4.1) spezifiziert allgemein die Umstände der Entstehung von Prozessen. Im Folgenden werden die Attribute der **Quelle** vorgestellt, die sich über das Attributfenster (Abbildung 4.2) editieren lassen.

Reiter Quelle

Das Attribut „Bezeichner" erwartet einen Namen für die angelegte **Quelle**. Der „Bezeichner" der **Quelle** ist für die weitere Modellierung und Analyse nicht relevant und kann zur zusätzlichen Kommentierung verwendet werden.

Mit dem Attribut „Anzahl" geben Sie die Anzahl der individuellen Prozesse an, die zu einem Zeitpunkt generiert werden sollen.

Das Attribut „Typ" läßt die Auswahl `EVERY`, `AT` und `BEDINGT` zu. Die Einstellung `BEDINGT` zeigt, dass die Generierung der Prozess von außen gesteuert werden soll. Die Einstellung `EVERY` bedeutet, dass alle t Zeiteinheiten ein individueller Prozess generiert wird. Die Einstellung `AT` bedeutet, dass zum Zeitpunkt t ein individueller Prozess gestartet wird. Wie groß das t sein soll, bestimmen Sie mit dem Attribut „Zeitangabe".

Das Attribut „Zeitangabe" erwartet einen arithmetischen Ausdruck (siehe auch Abschnitt 4.21 über Wahrscheinlichkeitsverteilungen oder den ProC/B-Modellformalismus).

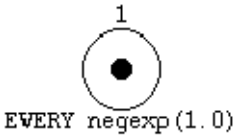


Abbildung 4.1: Quelle

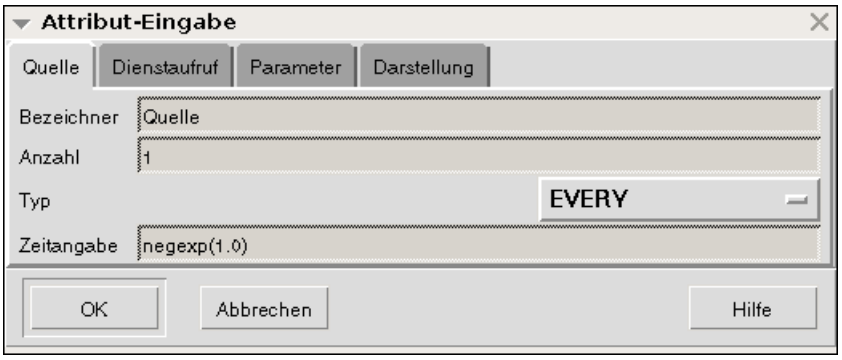


Abbildung 4.2: Quellenattribute

Reiter Dienstaufruf

Wird ein Dienst einer Funktionseinheit benutzt, um die **Quelle** zu steuern, können Sie den Namen der Funktionseinheit und des Dienstes in den Attributen „Funktionseinheit" und „Dienst" einsehen. Die Attribute des Dienstaufrufes sind schreibgeschützt, können folglich nicht verändert werden.

Um eine Dienstanbindung durchzuführen, sollten Sie die Anbindung der Dienste bequem mit der Maus erledigen. Die benötigten Attribut-Einstellungen werden automatisch vollzogen.

Reiter Parameter

Die in dieser Liste angezeigten Parameter sind die Parameter aus der verbundenen **ProzessID**. Das einzelne Parameterrecord besteht aus dem „Parameternamen", dem „Typ" des Parameter z.B. `REAL` oder `INT`, der „Dimension" des Parameters und dem „Wert". Die Attribute „Parametername", „Typ" und „Dimension" können in der **Quelle** nicht verändert werden. Das Attribut „Wert" kann hingegen dem gesetzten „Typ" und der gewählten „Dimension" entsprechend beliebig gewählt werden.

Reiter Darstellung

Mit Hilfe des Attributes „Hintergrundfarbe" kann die farbliche Darstellung eigenen Bedürfnissen (z.B. Markierung, Hervorhebung usw.) angepasst werden. Die restlichen Attribute können zum Ein- und Ausblenden der Quelle genutzt werden (siehe auch Anhang C).

4.2 Senke

Eine **Senke** (siehe Abbildung 4.3) spezifiziert allgemein die Umstände der Beendigung von Prozessen. Im Folgenden werden die Attribute der **Senke** vorgestellt, die sich über das Attributfenster (Abbildung 4.4) editieren lassen.

Abbildung 4.3: Senke

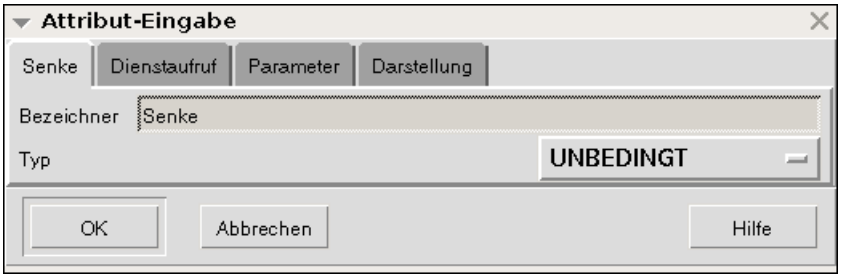


Abbildung 4.4: Senkenattribute

Reiter Senke

Das Attribut „Bezeichner" erwartet einen Namen für die angelegte **Senke**. Der „Bezeichner" der **Senke** ist für die weitere Modellierung und Analyse nicht relevant und kann zur zusätzlichen Kommentierung verwendet werden. Im Folgenden werden die schreibgeschützten und die veränderbaren Attribute der **Senke** vorgestellt.

Das Attribut „Typ" läßt nur die Auswahl zwischen `UNDEBINGT` und `BEDINGT` zu. `BEDINGT` bedeutet, dass der Prozess in der **Senke** beendet wird und einen weiteren Prozess innerhalb einer im Reiter Dienstaufruf angegebenen Funktionseinheit anstößt. Ist das Attribut „Typ" auf `UNBEDINGT` gestellt, dann wird der Prozess beendet ohne eine weitere Prozesserzeugung.

Reiter Dienstaufruf

Wird ein Dienst einer Funktionseinheit in der **Senke** benutzt, dann sehen Sie den Namen der Funktionseinheit und des Dienstes in den Attributen „Funktionseinheit" und „Dienst". Die Attribute des Dienstaufrufes sind schreibgeschützt, können folglich nicht verändert werden.

Um eine Dienstanbindung durchzuführen, sollten Sie die Anbindung der Dienste bequem mit der Maus erledigen. Die benötigten Attribut-Einstellungen werden automatisch vollzogen.

Reiter Parameter

Falls Sie einen Dienst einer Funktionseinheit an der **Senke** angebundenen haben, so sind die Parameter für den Dienstaufruf in dieser Parameterliste zu sehen. Das einzelne Parameterrecord besteht aus dem „Parameternamen", dem „Typ" des Parameters z.B. `REAL` oder `INT`, der „Dimension" des Parameter und dem „Wert".

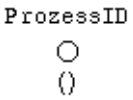


Abbildung 4.5: ProzessID

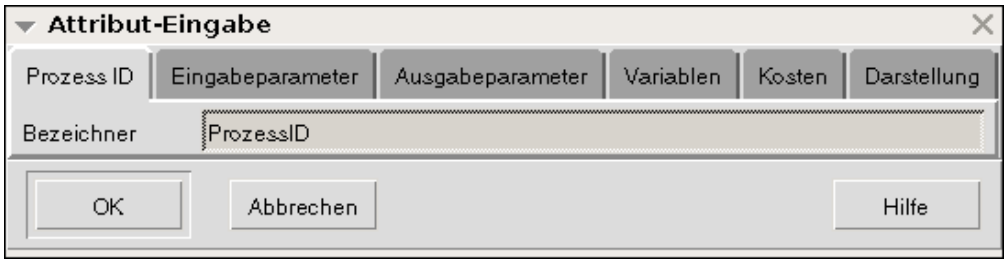


Abbildung 4.6: Attribut der ProzessID

Reiter Darstellung

Mit Hilfe des Attributes „Hintergrundfarbe" kann die farbliche Darstellung eigenen Bedürfnissen (z.B. Markierung, Hervorhebung usw.) angepasst werden. Die restlichen Attribute können zum Ein- und Ausblenden der Senke genutzt werden (siehe auch Anhang C).

4.3 ProzessID

Die **ProzessID** (siehe Abbildung 4.5) ist streng genommen nicht im ProC/B-Modellformalismus vorgesehen. Die **ProzessID** spezifiziert jeden generierten Prozess. D.h. wollen Sie ein Element benutzen, dass einen Prozess generiert (**Quelle**, **Prozessskettenkonnektor**), dann müssen Sie dieses Element mit einer **ProzessID** durch Kanten verbinden. Die **ProzessID** spezifiziert genau den generierten Prozess. Im Folgenden werden die Attribute der **ProzessID** vorgestellt, die sich über das Attributfenster (Abbildung 4.6) editieren lassen.

Reiter ProzessID

Das Attribut „Bezeichner" erwartet einen eindeutigen Namen für die **ProzessID**. Auf diesen „Bezeichner" wird unter Umständen in anderen Elementen weiter Bezug genommen, weshalb er eindeutig gewählt werden muss. Eine Überprüfung der Eindeutigkeit des „Bezeichners" übernimmt der ProC/B-Editor.

Reiter Eingabeparameter

„Eingabeparameter" dienen der Übergabe von Informationen aus der aufrufenden Umgebung der Prozesskette. Da es sich unter Umständen um mehrere Parameter handeln

kann, wird eine Liste von Parametern verwaltet. Das einzelne Parameterrecord besteht aus dem „Parameternamen", dem „Typ" des Parameters, der „Dimension" des Parameter und dem „Wert". Als Attribut „Typ" der Parameterliste werden folgende Typen unterstützt:

`INT (INTEGER)`, `REAL`, `STRING`, `BOOL (BOOLEAN)`

Mit der geeigneten Wahl einer „Dimension" für einen Parameter ist es möglich, Arrays zu verwenden (siehe Abschnitt 4.20, "Verwendung von Arrays").

„Eingabeparameter" sind lokale Variablen, auf sie kann nur innerhalb der Prozesskette zugegriffen werden, in der sie über die **ProzessID** definiert sind. Zur Unterscheidung zu **globalen Variablen** (s. 4.18) muss beim Zugriff auf die „Eingabeparameter" dem Parameternamen das Präfix `data.` vorrangestellt werden.

Reiter Ausgabeparameter

„Ausgabeparameter" dienen der Übergabe von Informationen aus einem abgeschlossenen Prozess an die Umgebung der Prozesskette. Da es sich unter Umständen um mehrere Parameter handeln kann, wird eine Liste von Parametern verwaltet. Das einzelne Parameterrecord besteht aus dem „Parameternamen", dem „Typ" des Parameter, der „Dimension" des Parameter und dem „Wert". Als „Typ" der Parameterliste werden die folgenden Typen unterstützt:

`INT (INTEGER)`, `REAL`, `STRING`, `BOOL (BOOLEAN)`

Mit der geeigneten Wahl einer „Dimension" für einen Parameter ist es möglich, Arrays zu verwenden (siehe Abschnitt 4.20, "Verwendung von Arrays").

Auf die lokalen Variablen „Ausgabeparameter" kann nur innerhalb der Prozesskette zugegriffen werden, in der sie über die **ProzessID** definiert sind. Zur Unterscheidung zu **globalen Variablen** (s. 4.18) muss beim Zugriff auf die „Ausgabeparameter" dem Parameternamen das Präfix `data.` vorrangestellt werden.

Reiter Variablen

Der ProC/B-Modellformalismus sieht vor, dass die Prozessketten eigene Datenräume/ Zustandsräume besitzen können. Dieser Forderung wird durch die Verwaltung einer Liste von „Variablen" in der **ProzessID** Rechnung getragen. Ein Variablenrecord besteht aus „Variablenname", „Variablentyp", „Variablendimension" und einem „Initialwert".

Auf die lokalen Variablen der **ProzessID** kann nur innerhalb der Prozesskette zugegriffen werden, in der sie über definiert sind. Zur Unterscheidung zu **globalen Variablen** (s. 4.18) muss beim Zugriff auf die lokalen „Variablen" dem Parameternamen das Präfix `data.` vorrangestellt werden.

Reiter Kosten

Mit dem Attribut „Leistungskosten" können Sie die Kosten angeben, die bei jedem Start eines Prozesses entstehen.

Mit dem Attribut „Umsatz" können Sie den betriebswirtschaftlichen Umsatz dieser Prozesskette für die Kostenrechnung angeben.

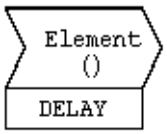


Abbildung 4.7: Prozesskettenelement

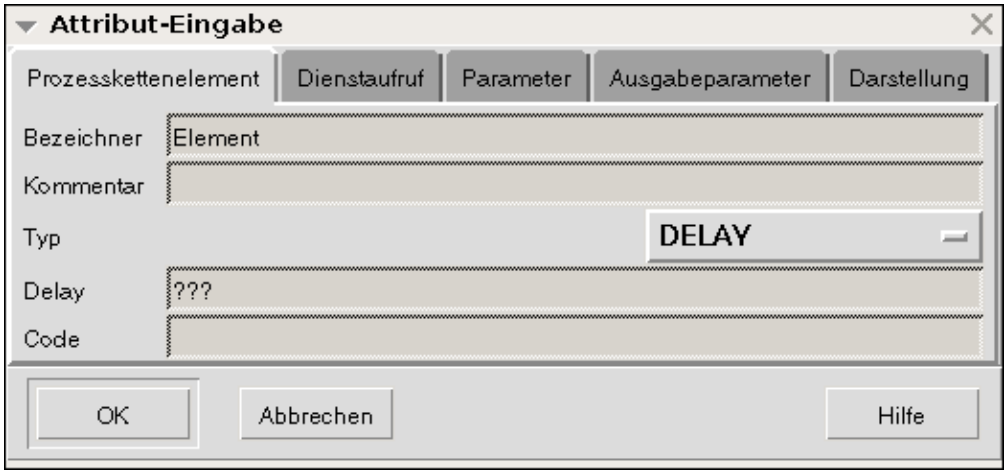


Abbildung 4.8: Attribute des Prozesskettenelementes

Reiter Darstellung

Mit Hilfe des Attributes „Hintergrundfarbe” kann die farbliche Darstellung eigenen Bedürfnissen (z.B. Markierung, Hervorhebung usw.) angepasst werden. Die restlichen Attribute können zum Ein- und Ausblenden der ProzessID genutzt werden (siehe auch Anhang C).

4.4 Prozesskettenelement

Für das **Prozesskettenelement** (siehe Abbildung 4.7) sind laut ProC/B-Modellformalismus zwei Grundformen vorgesehen. Das zeitbehaftete und das zeitlose **Prozesskettenelement**. Beide Grundformen sind in einem **Prozesskettenelement** vereinigt worden. Im Folgenden werden die Attribute des **Prozesskettenelementes** vorgestellt, die sich über das Attributfenster (Abbildung 4.8) editieren lassen.

Reiter Prozesskettenelement

Das Attribut „Bezeichner” erwartet einen Namen für das **Prozesskettenelement**. Der „Bezeichner” des **Prozesskettenelementes** ist für die weitere Modellierung und Analyse nicht relevant und kann zur zusätzlichen Kommentierung verwendet werden.

Das Attribut „Kommentar” ist für die eigentliche Kommentierung des **Prozesskett-**

tenelements gedacht. Dieses Attribut ist ebenfalls für die weitere Analyse nicht relevant.

Um eine Grundform des **Prozesskettenelementes** einzustellen, müssen Sie das Attribut „Typ" entsprechend wählen. Die Einstellung `DELAY` entspricht dem zeitbehafteten PKE des ProC/B-Modellformalismus, bei dem die Ausführung der Aktivität immer eine gewisse Zeit benötigt. Die Verzögerungszeit wird mit dem Attribut „Delay" eingegeben. Einzugeben ist ein arithmetischer Ausdruck. (Siehe auch den Abschnitt 4.21 über Wahrscheinlichkeitsverteilungen oder den ProC/B-Modellformalismus)

Bei der Einstellung `EXTERN` ergibt sich der Zeitverbrauch aus dem Zeitverbrauch des angebundenen Dienstes. Wird im angebundenen Dienst also keine Zeit verbraucht, ist auch das PKE zeitlos.

Die Einstellung `CODE` entspricht dem zeitlosen PKE des ProC/B-Modellformalismus. Im Attribut „Code" können Sie Ihre Anweisungen hinterlassen. Mit Hilfe des Befehls `writeln` können Kommentare in den Code eingefügt werden, zum Beispiel: `writeln(''Kommentar''); writeln(variable);` `writeln(data.variable)`.
Mit dem Ausdruck `time` kann auf die aktuelle Modellzeit zugegriffen werden.

Reiter Dienstaufruf

Wird ein Dienst einer Funktionseinheit benutzt, dann können Sie den Namen der Funktionseinheit und des Dienstes in den Attributen „Funktionseinheit" und „Dienst" einsehen. Die Attribute des Dienstaufrufes sind schreibgeschützt, können folglich nicht verändert werden.

Reiter Parameter

Eingabeparameter dienen der Übergabe von Informationen aus der aufrufenden Umgebung des Prozesskettenelementes an die aufgerufene Umgebung eines angebundenen Dienstes. Da es sich unter Umständen um mehrere Parameter handeln kann, wird eine Liste von Parametern verwaltet. Das einzelne Parameterrecord besteht aus dem „Parameternamen", dem „Typ" des Parameters, der „Dimension" des Parameters und dem „Wert". Das Attribut „Wert" läßt sich als einziges Attribut in der Eingabeparameterliste ändern, d.h. das Format für die Aufrufparameter wird innerhalb des Dienstes der Funktionseinheit definiert.

Reiter Ausgabeparameter

„Ausgabeparameter" dienen der Übergabe von Informationen aus der aufgerufenen Umgebung eines angebundenen Dienstes an die aufrufende Umgebung. Da es sich unter Umständen um mehrere Parameter handeln kann, wird eine Liste von Parametern verwaltet. Das einzelne Parameterrecord besteht aus dem „Parameternamen", dem „Typ" des Parameters, der „Dimension" des Parameters und dem „Wert". Das Attribut „Wert" läßt sich als einziges Attribut in der Ausgabeparameterliste ändern, d.h. das Format für die Rückgabeparameter wird innerhalb des Dienstes der Funktionseinheit definiert.

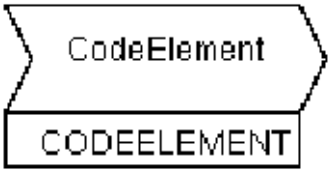


Abbildung 4.9: Codeelement

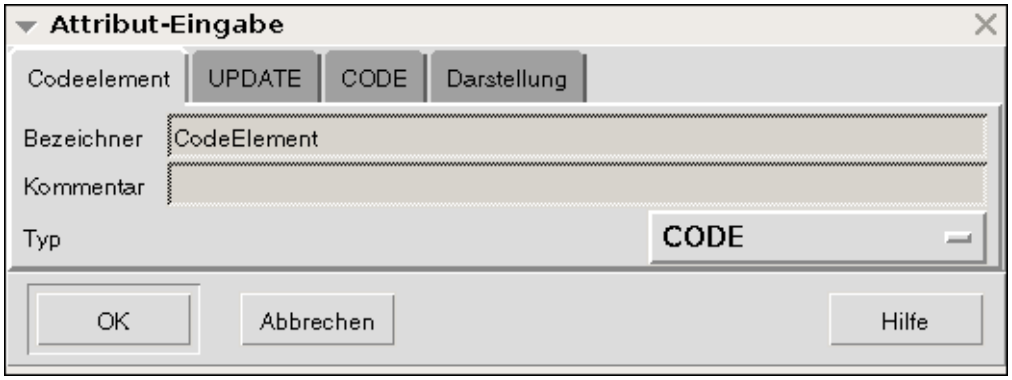


Abbildung 4.10: Attribute des Codeelements

Reiter Darstellung

Mit Hilfe des Attributes „Hintergrundfarbe" kann die farbliche Darstellung eigenen Bedürfnissen (z.B. Markierung, Hervorhebung usw.) angepasst werden. Mit Hilfe des Attributes „Breite" können Sie die Breite der Darstellung festlegen. Über „Anpassung" wird festgelegt, ob die Größe der Darstellung automatisch oder manuell festgelegt werden soll. Die restlichen Attribute können zum Ein- und Ausblenden des Prozesskettenelements genutzt werden (siehe auch Anhang C).

4.5 Codeelement

Das **Codeelement** (siehe Abbildung 4.9) kann zur Eingabe von Hi-Slang-Code und für Update-Operationen von Rewards eingesetzt werden. Mit Hilfe des **Codeelements** können mehrere Zeilen Code oder mehrere Update-Anweisungen eingegeben werden, die beim Ausdruck des Gesamtmodells auf eine separate Seite ausgedruckt werden. Eine weitere Funktionalität ist die Schnellanzeige aller eingegebenen Zeilen durch die Öffnen-Funktion des **Codeelements**. Dazu klicken Sie zunächst auf das **Codeelement** und drücken nach Selektion die mittlere Maustaste. Wählen Sie anschließend die Funktion Öffnen, und ein weiteres Fenster wird geöffnet, das die eingegebenen Codezeilen oder Update-Anweisungen auf einen Blick darstellt. Im Folgenden werden die Attribute des **Codeelementes** vorgestellt, die sich über das Attributfenster (Abbildung 4.10) editieren lassen.

Reiter Codeelement

Das Attribut „Bezeichner" erwartet einen Namen für das **Codeelement**. Der „Bezeichner" des **Codeelements** ist für die weitere Modellierung und Analyse nicht relevant und kann zur zusätzlichen Kommentierung verwendet werden.

Das Attribut „Kommentar" ist für die eigentliche Kommentierung des **Codeelements** gedacht. Dieses Attribut ist für die weitere Analyse nicht relevant.

Das Attribut „Typ" legt fest, ob Sie das **Codeelement** für die Eingabe von Code für HIT oder Update-Anweisungen nutzen wollen.

Reiter Update

Hier können die Update-Anweisungen für Rewards eingegeben werden. Jede Anweisung besteht aus dem „Namen" eines Rewards und einem „Wert" des Typs `INT` oder `REAL`. Die Bedeutung des Wertes ist abhängig vom Typ des Rewards. Für Rewards vom Typ `STATE` beschreibt „Wert" die Differenz zwischen dem alten und dem neuen Zustand. Bei Rewards vom Typ `EVENT` beschreibt „Wert" den tatsächlich beobachteten Wert. Beim Typ `COUNT` wird der Wert ignoriert und der Zähler des Rewards um 1 erhöht (siehe auch die Abschnitte 4.18 für die Deklaration von Rewards und 5.3 über Messpunkte und Verursacherpfade).

Reiter Code

Hier kann eine Menge von Codezeilen eingegeben werden.

Reiter Darstellung

Mit Hilfe des Attributes „Hintergrundfarbe" kann die farbliche Darstellung eigenen Bedürfnissen (z.B. Markierung, Hervorhebung usw.) angepasst werden.

Mit Hilfe des Attributes „Breite" können Sie die Breite der Darstellung festlegen.

Über „Anpassung" wird festgelegt, ob die Größe der Darstellung automatisch oder manuell festgelegt werden soll. Die restlichen Attribute können zum Ein- und Ausblenden des Code-Elements genutzt werden (siehe auch Anhang C).

4.6 Loop-Element

Das **Loop/Endloop-Element** (siehe Abbildung 4.11) kann zur Realisierung von Schleifen verwendet werden. Eine Schleife beginnt immer mit einem Element vom Typ `LOOP` und endet mit einem Element von Typ `ENDLOOP`. Der zwischen den beiden Elementen liegende Teil der Prozesskette wird so lange ausgeführt, bis die im **Endloop-Element** angegebene „Abbruchbedingung" zutrifft. Im Folgenden werden die Attribute des **Loop-Elements** vorgestellt, die sich über das Attributfenster (Abbildung 4.12) editieren lassen.

Reiter Loop/Endloop

Das Attribut „Bezeichner" erwartet einen Namen für das **Loop-Element**. Der „Bezeichner" des **Loop-Elements** ist für die weitere Modellierung nicht relevant und kann

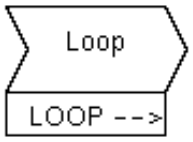


Abbildung 4.11: Loop-Element

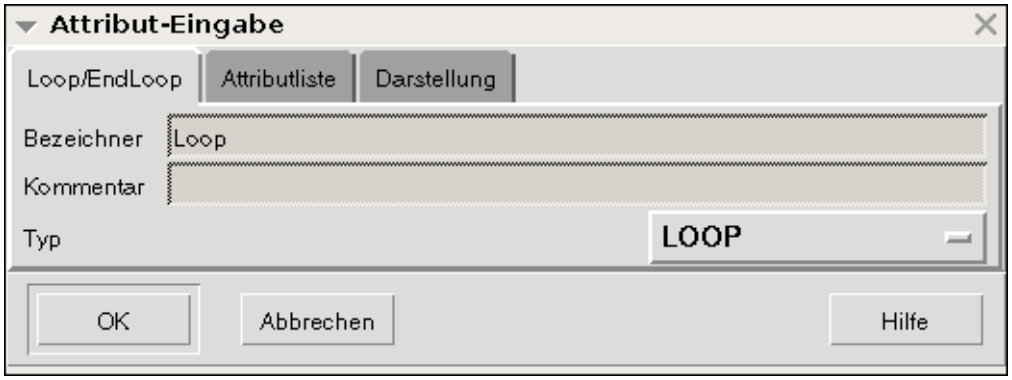


Abbildung 4.12: Attribute des Loop-Elements

zur zusätzlichen Kommentierung verwendet werden. Das Attribut „Kommentar" ist für die eigentliche Kommentierung des **Loop-Elements** gedacht. Dieses Attribut ist ebenfalls für die weitere Analyse nicht relevant.

Mit dem Attribut „Typ" wird festgelegt, ob das Element den Beginn einer Schleife (`LOOP`) oder das Ende einer Schleife (`ENDLOOP`) markiert.

Reiter Attributliste

Die „Abbruchbedingung" muss nur bei einem Element des Typs `ENDLOOP` angegeben werden. Die Schleife wird so lange ausgeführt, bis die Bedingung zutrifft. Wenn keine Abbruchbedingung angegeben ist, wird die Schleife unendlich oft durchlaufen.

Reiter Darstellung

Mit Hilfe des Attributes „Hintergrundfarbe" kann die farbliche Darstellung eigenen Bedürfnissen (z.B. Markierung, Hervorhebung usw.) angepasst werden. Mit Hilfe des Attributes „Breite" können Sie die Breite der Darstellung festlegen. Über „Anpassung" wird festgelegt, ob die Größe der Darstellung automatisch oder manuell festgelegt werden soll. Die restlichen Attribute können zum Ein- und Ausblenden des Loop-Elements genutzt werden (siehe auch Anhang C).

Abbildung 4.13: Oder-Konnektor

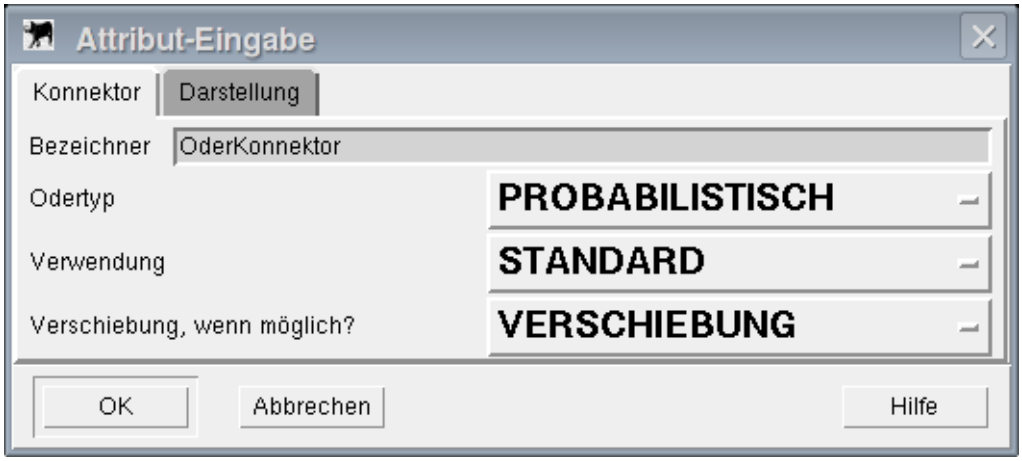


Abbildung 4.14: Attribute des Oder-Konnektor

4.7 Oder-Konnektor

Öffnende Oder-Konnektoren (siehe Abbildung 4.13) ermöglichen eine alternative Fortsetzung von Prozessen, **Schließende Oder-Konnektoren** ermöglichen die Zusammenführung alternativer Zweige. Öffnende und schließende Oder-Konnektoren sind im ProC/B-Editor mit dem **Oder-Konnektor** verwirklicht worden (ODER/ODER-Konnektor im ProC/B-Formalimus). **Oder-Konnektoren** können als öffnender, als schließender sowie als sowohl schließender als auch öffnender Oder-Konnektor benutzt werden. Oder-Konnektoren müssen immer paarweise, also ein öffnender und ein schließender Konnektor, auftreten.

Um Wahrscheinlichkeiten (probabilistischer Oder-Konnektor) oder Bedingungen (boolescher Oder-Konnektor) für die verschiedenen Zweige angeben zu können, müssen Sie die Attribute der ausgehenden Kanten des Oder-Konnektors verändern. Die Attribute der ausgehenden Kanten sind „Bezeichner” und „Wert”. Der „Bezeichner” gibt der Kante einen Namen, der nicht weiter relevant ist. Mit Hilfe des Attributes „Wert” können Sie eine Wahrscheinlichkeit oder eine Bedingung für diesen alternativen Zweig angeben. Zur Vervollständigung der Beschreibung folgen die Attribute des **Oder-Konnektors**, die sich über das Attributfenster (Abbildung 4.14) editieren lassen.

Reiter Konnektor

Das Attribut „Bezeichner" erwartet einen Namen für den **Oder-Konnektor**. Der „Bezeichner" des **Oder-Konnektors** ist für die weitere Modellierung und Analyse nicht relevant und kann zur zusätzlichen Kommentierung verwendet werden.

Das Attribut „Odertyp" entscheidet, ob es sich um einen probabilistischen oder einen booleschen **Oder-Konnektor** handelt. Über die Attribute „Verwendung" und „Verschiebung" kann der Oder-Konnektor zusätzlich als Alternativenbaustein verwendet werden, um Zweige des Konnektors auszublenden (siehe auch Anhang C).

Reiter Darstellung

Mit Hilfe des Attributes „Höhe" können Sie die Höhe der Darstellung festlegen.

Über „Anpassung" wird festgelegt, ob die Größe der Darstellung automatisch oder manuell festgelegt werden soll. Bei der Einstellung `AUTO` richtet sich die Darstellung nach der eingestellten Höhe und den ein- und ausgehenden Kanten. Die restlichen Attribute können zum Ein- und Ausblenden des Oder-Konnektors genutzt werden (siehe auch Anhang C).

4.8 Öffnender Parallelkonnektor

Öffnende Parallelkonnektoren (siehe Abbildung 4.15) sind eine Realisierung der ODER/UND-Konnektoren. Die ausgehenden Kanten entsprechen wie im ProC/B-Modellformalismus einer parallelen Fortsetzung von Prozessen. Gleichzeitig kann der **Öffnende Parallelkonnektor** auch als schliessender Oder-Konnektor genutzt werden, der die eingehenden Kanten als alternative Zweige zusammenführt. Parallel-Konnektoren müssen immer paarweise, also ein öffnender und ein schließender Konnektor, auftreten. Im ProC/B-Formalismus wird eine Kurzform verwendet, mit der Sie gleichartige parallele Nachfolgeabschnitte vereinfacht darstellen können. (Attribut Anzahl der ausgehenden Kanten). Diese verkürzende Schreibweise wird auch vom ProC/B-Editor unterstützt. Jede ausgehende Kante des **Öffnenden Parallelkonnektors** hat ein Attribut mit dem Namen „Wert". Dieses Attribut „Wert" ist gleichbedeutend mit dem Attribut „Anzahl" aus dem ProC/B-Formalismus. Zur Vervollständigung der Beschreibung folgen die Attribute des öffenden Parallelkonnektors, die sich über das Attributfenster (Abbildung 4.16) editieren lassen.

Sie sind auch in der Lage eine Kostengewichtung an den ausgehenden Kanten anzugeben. Die Initialbelegung ist eine Gleichverteilung, d.h. an jedem ausgehenden Zweig ist der Wert für das „Kostengewicht" gleich 1. Wollen Sie eine andere Verteilung der Kosten, dann brauchen Sie nur die Kostengewichtsattribute an den ausgehenden Kanten verändern.

Reiter Konnektor

Das Attribut „Bezeichner" erwartet einen Namen für den **Öffnenden Parallelkonnektor**. Der „Bezeichner" des **Öffenden Parallelkonnektors** ist für die weitere Modellierung und Analyse nicht relevant und kann zur zusätzlichen Kommentierung verwendet werden.

Abbildung 4.15: Öffnender Parallelkonnektor

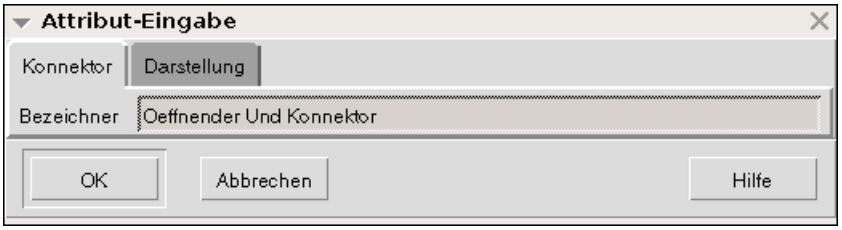


Abbildung 4.16: Attribute des Öffnenden Parallelkonnektor

Reiter Darstellung

Mit Hilfe des Attributes „Höhe" können Sie die Höhe der Darstellung festlegen.

Über „Anpassung" wird festgelegt, ob die Größe der Darstellung automatisch oder manuell festgelegt werden soll. Bei der Einstellung `AUTO` richtet sich die Darstellung nach der eingestellten Höhe und den ein- und ausgehenden Kanten. Die restlichen Attribute können zum Ein- und Ausblenden des Konnektors genutzt werden (siehe auch Anhang C).

4.9 Schließender Parallelkonnektor

Schließende Parallelkonnektoren (siehe Abbildung 4.17) sind eine Realisierung der UND/ODER-Konnektoren. Die eingehenden Kanten entsprechen wie im ProC/B-Modellformalismus einer Zusammenführung von parallelen Prozessen. Gleichzeitig kann der **Schließende Parallelkonnektor** als öffnender Oder-Konnektor genutzt werden, der eine alternative Fortführung von Prozessen ermöglicht. Parallel-Konnektoren müssen immer paarweise, also ein öffnender und ein schließender Konnektor, auftreten. Im ProC/B-Formalismus wird eine Kurzform verwendet, mit der Sie gleichartige parallele Nachfolgeabschnitte vereinfacht darstellen können (Attribut Anzahl). Diese verkürzende Schreibweise wird auch vom ProC/B-Editor unterstützt. Jede ein- bzw. ausgehende Kante des Schließenden Parallelkonnektors hat ein Attribut mit dem Namen „Wert". Dieses Attribut „Wert" ist gleichbedeutend mit dem Attribut „Anzahl" aus dem ProC/B-Formalismus. Um Wahrscheinlichkeiten oder Bedingungen für die ausgehenden Zweige angeben zu können, müssen Sie die Attribute der ausgehenden Kanten des Schließenden Parallelkonnektors verändern.

Die Attribute der ausgehenden Kanten sind „Bezeichner" und „Wert". Der „Bezeichner" gibt der Kante einen Namen, der nicht weiter relevant ist. Mit Hilfe des

Abbildung 4.17: Schließender Parallelkonnektor

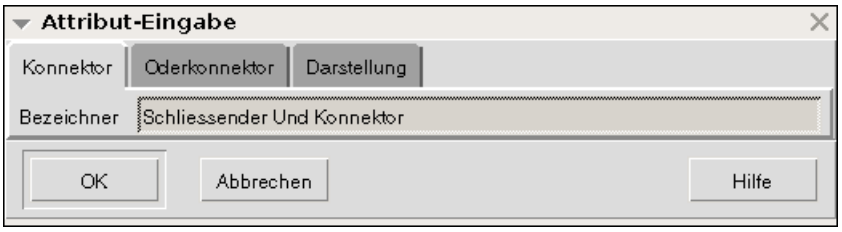


Abbildung 4.18: Attribute des Schließenden Parallelkonnektor

Attributes „Wert” können Sie eine Wahrscheinlichkeit oder eine Bedingung für diesen alternativen Zweig angeben. Zur Vervollständigung der Beschreibung folgen die Attribute des **Schließenden Parallelkonnektors**, die sich über das Attributfenster (Abbildung 4.18) editieren lassen.

Reiter Konnektor

Das Attribut „Bezeichner” erwartet einen Namen für den **Schließenden Parallelkonnektor**. Der „Bezeichner” des **Schließenden Parallelkonnektors** ist für die weitere Modellierung und Analyse nicht relevant und kann zur zusätzlichen Kommentierung verwendet werden.

Reiter Oderkonnektor

Das Attribut „Typ” entscheidet, ob es sich um einen probabilistischen oder einen booleschen Oder-Konnektor handelt.

Reiter Darstellung

Mit Hilfe des Attributes „Höhe” können Sie die Höhe der Darstellung festlegen.

Über „Anpassung” wird festgelegt, ob die Größe der Darstellung automatisch oder manuell festgelegt werden soll. Bei der Einstellung `AUTO` richtet sich die Darstellung nach der eingestellten Höhe und den ein- und ausgehenden Kanten. Die restlichen Attribute können zum Ein- und Ausblenden des Konnektors genutzt werden (siehe auch Anhang C).

Abbildung 4.19: Schließender und Öffnender Parallelkonnektor

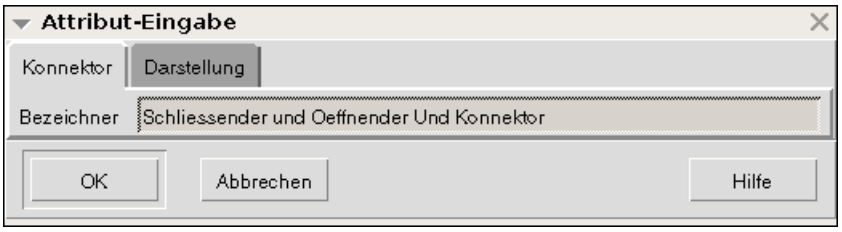


Abbildung 4.20: Attribute des Schließenden und Öffnenden Parallelkonnektors

4.10 Schließender und Öffnender Parallelkonnektor

Schließende und Öffnende-Parallelkonnektoren (siehe Abbildung 4.19) sind eine Realisierung der UND/UND-Konnektoren. Die eingehenden Kanten entsprechen wie im ProC/B-Modellformalismus einer Zusammenführung von parallelen Prozessen. Die ausgehenden Kanten entsprechen einer parallelen Fortsetzung von Prozessen. Im ProC/B-Formalismus wird eine Kurzform verwendet, mit der Sie gleichartige parallele Nachfolgeabschnitte vereinfacht darstellen können (Attribut Anzahl der ausgehenden Kanten). Diese verkürzende Schreibweise wird auch vom ProC/B-Editor unterstützt. Jede ein- bzw. ausgehende Kante des **Schließenden und Öffnenden Parallelkonnektors** hat ein Attribut mit dem Namen „Wert". Dieses Attribut „Wert" ist gleichbedeutend mit dem Attribut „Anzahl" aus dem ProC/B-Formalismus. Zur Vervollständigung der Beschreibung folgen die Attribute des **Schließenden und Öffnenden Parallelkonnektors**, die sich über das Attributfenster (Abbildung 4.20) editieren lassen.

Reiter Konnektor

Das Attribut „Bezeichner" erwartet einen Namen für den **Schließenden und Öffnenden Parallelkonnektor**. Der „Bezeichner" ist für die weitere Modellierung und Analyse nicht relevant und kann zur zusätzlichen Kommentierung verwendet werden.

Reiter Darstellung

Mit Hilfe des Attributes „Höhe" können Sie die Höhe der Darstellung festlegen.

Über „Anpassung" wird festgelegt, ob die Größe der Darstellung automatisch oder manuell festgelegt werden soll. Bei der Einstellung `AUTO` richtet sich die Darstellung

nach der eingestellten Höhe und den ein- und ausgehenden Kanten. Die restlichen Attribute können zum Ein- und Ausblenden des Konnektors genutzt werden (siehe auch Anhang C).

4.11 Prozesskettenkonnektor

Ein **Prozesskettenkonnektor** (siehe Abbildung 4.21) dient zur Synchronisation unterschiedlicher Prozessketten. Der **Prozesskettenkonnektor** erlaubt sowohl die Beendigung als auch die Fortführung ankommender Prozesse. Zusätzlich können neue Prozesse gestartet werden. Alle eingehenden Kanten symbolisieren ankommende Prozesse. Sind alle Prozesse linksseitig endender Prozessketten bzw. die vor dem **Prozesskettenkonnektor** liegenden Aktivitäten fortgeführter Prozessketten beendet, werden alle Prozesse der rechtsseitig startenden Prozessketten bzw. die hinter dem **Prozesskettenkonnektor** liegenden Aktivitäten fortgeführter Prozessketten gestartet.

Falls Sie neue Prozesse starten möchten, dann muss das erste Element Ihrer rechten Prozesskette eine **ProzessID** sein. Mit Hilfe der **ProzessID** können Sie den Prozess spezifizieren.

Falls Sie einen Prozess nach Erreichen eines **Prozesskettenkonnektors** weiterführen möchten, dann legen sie an dem **Prozesskettenkonnektor** einen Durchgangsport an. Abbildung 4.23 zeigt einen **Prozesskettenkonnektor** mit drei Prozessketten. Die untere Prozesskette wird mit Hilfe eines Durchgangsports fortgesetzt.

Im ProC/B-Modellformalismus wird ein Attribut „Anzahl" benutzt, mit dem Sie notieren können, wieviel parallele Prozesse einer den **Prozesskettenkonnektor** erreichenden Prozesskette zur Synchronisation benötigt werden bzw. wieviel parallele Prozesse einer den **Prozesskettenkonnektor** verlassenden Prozesskette erzeugt bzw. fortgeführt werden sollen. Für jede fortgeführte Prozesskette gilt, dass die Anzahl der fortgeführten Prozesse jener Anzahl der den Konnektor auf dieser Prozesskette erreichenden Prozesse entsprechen muss, also auf fortgeführten Prozessketten am entsprechenden Konnektor weder Prozesse erzeugt noch beendet werden können. Diese Notation wird auch vom ProC/B-Editor unterstützt. Jede Kante des **Prozesskettenkonnektors** hat ein Attribut mit dem Namen „Wert". Dieses Attribut „Wert" ist gleichbedeutend mit dem Attribut Anzahl aus dem ProC/B-Formalismus. An allen Kanten des **Prozesskettenkonnektors** kann ausserdem Hi-Slang-Code angegeben werden, der sich zur Variablenübergabe nutzen lässt. Der Code an den eingehenden Kanten wird vor der Synchronisation am Konnektor ausgeführt. Hier lassen sich die Werte von lokalen Variablen in globale Variablen schreiben. Die hierfür benötigten globalen Variablen müssen vom Nutzer selbst angelegt werden. Der Code an den ausgehenden Kanten wird direkt nach der Synchronisation ausgeführt. Hier können die Werte aus den globalen Variablen wieder in lokale Variablen übernommen werden. Durch den **Prozesskettenkonnektor** wird dabei sichergestellt, dass immer nur diejenigen Prozesse den Code an ihrer Kante ausführen dürfen, die für die Synchronisation am Konnektor benötigt werden.

Zur Vervollständigung der Beschreibung folgen die Attribute des **Prozesskettenkonnektors**, die sich über das Attributfenster (Abbildung 4.22) editieren lassen.

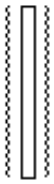


Abbildung 4.21: Prozesskettenkonnektor

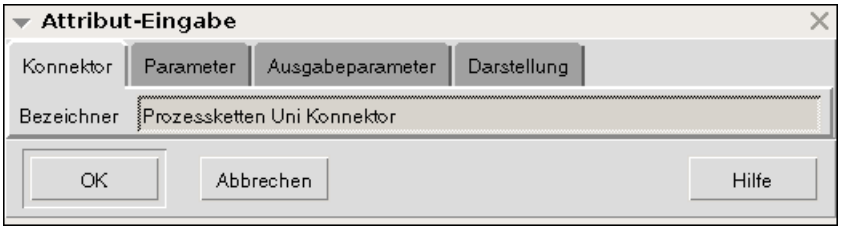


Abbildung 4.22: Attribute des Prozesskettenkonnektors

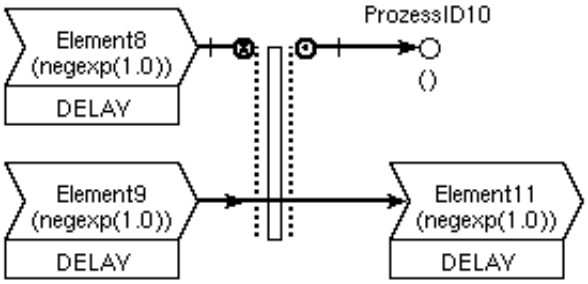


Abbildung 4.23: Prozesskettenkonnektor mit Durchgangsport

Reiter Konnektor

Das Attribut „Bezeichner" erwartet einen Namen für den **Prozesskettenkonnektor**. Der „Bezeichner" des **Prozesskettenkonnektors** ist für die weitere Modellierung und Analyse nicht relevant und kann zur zusätzlichen Kommentierung verwendet werden.

Reiter Parameter

Die in dieser Liste angezeigten „Parameter" sind die Parameter der angeschlossenen **ProzessID** und spezifizieren die rechten Prozessketten. Das einzelne Parameterrecord besteht aus dem „Parameternamen", dem „Typ" des Parameters, der „Dimension" des Parameter und dem „Wert". Die Attribute „Parametername", „Typ" und „Dimension" können in dem **Prozesskettenkonnektor** nicht verändert werden. Das Attribut „Wert" kann hingegen dem gesetzten Typ entsprechend beliebig gewählt werden.

Reiter Ausgabeparameter

Die in dieser Liste angezeigten „Ausgabeparameter" sind die Parameter der angeschlossenen **ProzessID** und spezifizieren die rechten Prozessketten. Das einzelne Parameterrecord besteht aus dem „Parameternamen", dem „Typ" des Parameters, der „Dimension" des Parameters und dem „Wert". Die Attribute „Parametername", „Typ" und „Dimension" können in dem **Prozesskettenkonnektor** nicht verändert werden. Das Attribut „Wert" kann hingegen dem gesetzten Typ entsprechend beliebig gewählt werden.

Reiter Darstellung

Mit Hilfe des Attributes „Höhe" können Sie die Höhe der Darstellung festlegen.

Über „Anpassung" wird festgelegt, ob die Größe der Darstellung automatisch oder manuell festgelegt werden soll. Bei der Einstellung `AUTO` richtet sich die Darstellung nach der eingestellten Höhe und den ein- und ausgehenden Kanten. Die restlichen Attribute können zum Ein- und Ausblenden des Konnektors genutzt werden (siehe auch Anhang C).

4.12 Funktionseinheit

Funktionseinheiten dienen zur Darstellung und Modellierung von hierarchischen Modellteilen. Eine **Funktionseinheit** verfügt über eine Aussenansicht (Abbildung 4.24) und eine Innenansicht. Die Innenansicht einer **Funktionseinheit** kann Prozessketten und weitere Funktionseinheiten enthalten. Durch die enthaltenen Prozessketten kann eine **Funktionseinheit** Dienste anbieten, die von den umgebenden Modellteilen genutzt werden können.
Zur Vervollständigung der Beschreibung folgen die Attribute, die sich über das Attributfenster (Abbildung 4.25) editieren lassen.

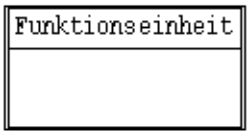


Abbildung 4.24: Funktionseinheit

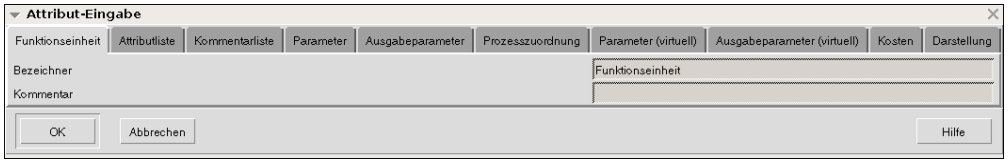


Abbildung 4.25: Attribute der Funktionseinheit

Reiter Funktionseinheit

Das Attribut „Bezeichner” erwartet einen Namen für die **Funktionseinheit**. Auf diesen „Bezeichner” wird unter Umständen in anderen Elementen weiter Bezug genommen, weshalb er eindeutig gewählt werden muss. Eine Überprüfung der Eindeutigkeit des Bezeichners übernimmt der Editor.

Das Attribut Kommentar ermöglicht eine Kurzbeschreibung der **Funktionseinheit**.

Reiter Attributliste

Mit Hilfe dieser „Attributliste” sind Sie in der Lage, eine Liste prägender Attribute zu verwalten. Jedes Listenelement besteht aus einem „Attributnamen”, dem „Typ”, der „Dimension” und einem initialen Attributwert („Init”). Diese Attributliste entspricht der Attributliste aus dem ProC/B-Formalismus.

Reiter Kommentarliste

Die „Kommentarliste” ist zur ausführlichen Kommentierung der **Funktionseinheit** gedacht.

Reiter Parameter

Die in dieser Liste angezeigten „Parameter” sind die Parameter der einzelnen Dienste der **Funktionseinheit**. Das einzelne Parameterrecord besteht aus dem „Prozessnamen”, dem „Parameternamen”, dem „Typ” des Parameter, der „Dimension” des Parameter und dem „Wert”. Die Attribute dieser Liste können in der **Funktionseinheit** nicht verändert werden.

Reiter Ausgabeparameter

Die in dieser Liste angezeigten „Ausgabeparameter" sind die Ausgabeparameter der einzelnen Dienste der **Funktionseinheit**. Das einzelne Parameterrecord besteht aus dem „Prozessnamen", dem „Parameternamen", dem „Typ" des Parameter, der „Dimension" des Parameter und dem „Wert". Die Attribute dieser Liste können in der **Funktionseinheit** nicht verändert werden.

Reiter Prozesszuordnung

Mit Hilfe der Prozesszuordnungsliste können Sie Dienste von **Funktionseinheiten** importieren (siehe **externe Funktionseinheiten**, Abschnitt 4.13). Jedes einzelne Listenelement besteht aus den Attributen „Prozessname", „Funktionseinheit" und „Dienst". Wollen Sie nun einen Dienst einer anderen **Funktionseinheit** innerhalb der **Funktionseinheit** nutzen, dann sollten sie die Attribute der Liste folgendermaßen wählen: Das Attribut „Prozessname" legt den Dienstnamen fest, mit dem Sie den zu importiertenden Dienst innerhalb der **Funktionseinheit** referenzieren möchten. Mit den Attributen „Funktionseinheit" und „Dienst" bestimmen Sie, welcher Dienst einer **Funktionseinheit** importiert werden soll. „Funktionseinheit" und „Dienst" können aus einem Menü gewählt werden, in dem alle möglichen Dienste der Funktionseinheiten aufgeführt sind. Hierbei handelt es sich um die Dienste anderer **Funktionseinheiten**, **Server**, **Counter**, **Storages** und **externer Funktionseinheiten**. Für jeden dieser Fälle müssen für bestimmte Attribute der Zuordnungsliste bestimmte Einträge gemacht werden. Welche Attribute der Zuordnungsliste wie belegt werden müssen, zeigt die folgende Tabelle.

	Prozessname	Funktionseinheit	Dienst
Funktionseinheit	beliebig	jeweiliger Bezeichner	Dienstnamen eines Dienstes dieser FE
Server	beliebig	jeweiliger Bezeichner	*request*
Counter	beliebig	jeweiliger Bezeichner	*change*
Storage	beliebig	jeweiliger Bezeichner	*change*, *alter*, *alter_or_skip*, *content*
externe Funktionseinheit	beliebig	EXTERNAL	Dienstnamen eines Dienstes dieser externen FE

Nachdem Sie die Zuordnungsliste editiert haben, wird innerhalb der **Funktionseinheit** eine **externe Funktionseinheit** mit den jeweiligen Dienstnamen angelegt. Außerdem ist die Außenansicht der **Funktionseinheit** analog zum ProC/B-Formalismus aktualisiert, d.h. unterhalb der gestrichelten Linie sind die Zuordnungen zu sehen.

*Bei jeder Änderung der Zuordnungsliste wird auch die **externe Funktionseinheit** aktualisiert. Ein ausführliches Beispiel zur Modellierung von **externen Funktionseinheiten** ist im Kapitel 6 Anwendungsbeispiele zu sehen.*

Reiter Parameter (virtuell)

Die virtuellen Parameter legen die Aufrufsyntax der jeweiligen Dienste der **externen Funktionseinheit** fest. Das virtuelle Parameterlistenrecord besteht aus den Attributen „Prozessname", „Parametername", „Typ", „Dimension" und „Initial". Mit dem Attribut „Prozessname" bestimmen Sie den Dienst der **externen Funktionseinheit**, auf den sich die virtuellen Parameter beziehen sollen. Mit Hilfe der nachfolgenden Attribute „Parametername", „Typ", „Dimension" und „Initial" können Sie die Aufrufsyntax des Dienstes bestimmen.
Die vituelle Parameterliste wird automatisch gefüllt, wenn in der Prozesszuordnungsliste ein Eintrag vorgenommen wurde.

Reiter Ausgabeparameter (virtuell)

Die virtuellen Ausgabeparameter legen fest, welche Parameter nach Beendigung des jeweiligen Dienstes der **externen Funktionseinheit** übergeben werden sollen. Das virtuelle Ausgabeparameterlistenrecord besteht aus den Attributen „Prozessname", „Parametername", „Typ", „Dimension" und „Initial". Mit dem Attribut „Prozessname" bestimmen Sie den Dienst der **externen Funktionseinheit**. Mit Hilfe der nachfolgenden Attribute „Parametername", „Typ", „Dimension" und „Initial" können Sie die Übergabesyntax des Dienstes bestimmen.
Die virtuelle Ausgabeparameterliste wird automatisch gefüllt, wenn in der Prozesszuordnungsliste ein Eintrag vorgenommen wurde.

Reiter Kosten

Mit dem Attribut „Fixkosten" können Sie die Kosten angeben, die auf jeden Fall für den Betrieb der **Funktionseinheit** anfallen.

Mit dem Attribut „Betriebskosten" können Sie die Kosten angeben, die bei jeder Benutzung der **Funktionseinheit** in Abhängigkeit der Zeit entstehen.

Reiter Darstellung

Mit Hilfe des Attributes „Hintergrundfarbe" kann die farbliche Darstellung eigenen Bedürfnissen (z.B. Markierung, Hervorhebung usw.) angepasst werden. Mit Hilfe des Attributes „Breite" können Sie die Breite der Darstellung festlegen. Mit Hilfe des Attributes „Höhe" können Sie die Höhe der Darstellung festlegen. Über „Anpassung" wird festgelegt, ob die Größe der Darstellung automatisch oder manuell festgelegt werden soll. Die restlichen Attribute können zum Ein- und Ausblenden der Funktionseinheit genutzt werden (siehe auch Anhang C).

4.13 Externe Funktionseinheit

Externe Funktionseinheiten (siehe Abbildung 4.26) sind nur innerhalb von **Funktionseinheiten**, d.h. in deren Innenansicht erlaubt. **Funktioneinheiten** können Dienste auf oberer Ebene importieren. Geschieht dies, so werden die importierten Dienste im Innern der **Funktionseinheit** durch eine **externe Funktionseinheit** repräsentiert. Die

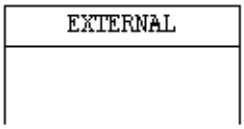


Abbildung 4.26: Externe Funktionseinheit

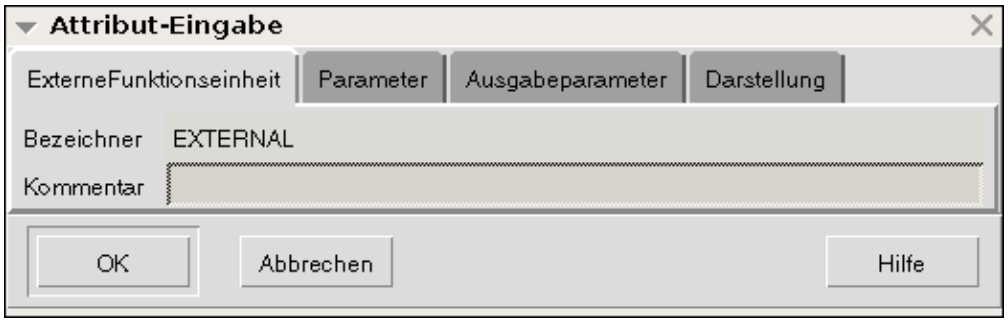


Abbildung 4.27: Attribute der externen Funktionseinheit

Verwendung der importierten Dienste, z.B. die Anbindung an **Prozesskettenelemente**, erfolgt analog zur Nutzung herkömmlicher **Funktioneinheiten**. Die Dienste und deren formale Parameterlisten werden durch die überliegende **Funktionseinheit** fest vorgegeben.

*Im Gegensatz zu anderen Prozesskettenelementen können **Externe Funktionseinheiten** nicht über die Buttonleiste angelegt und auch nicht direkt gelöscht werden. Das Anlegen und Löschen von **Externen Funktionseinheiten** ist ausschliesslich über die Prozesszuordnungsliste der zugehörigen **Funktionseinheit** möglich. Auch die Attribute der **externen Funktionseinheiten** sind nicht editierbar. Änderungen an einer **externen Funktionseinheit** einer Ebene können nur in der überliegenden **Funktionseinheit** durchgeführt werden.*

Zur Vervollständigung der Beschreibung folgen die Attribute, die sich über das Attributfenster (Abbildung 4.27) editieren lassen.

Reiter Externe Funktionseinheit

Das Attribut „Bezeichner” hat stets den Inhalt `EXTERNAL` und ist nicht editierbar.

Das Attribut „Kommentar” ist für eine Kurzbeschreibung der **Externen Funktionseinheit** gedacht.

Reiter Parameter

Die in dieser Liste angezeigten Parameter sind die Parameter aus den importierten Prozessen und spezifizieren die verbundenen Prozessketten. Das einzelne Parameterrecord für jeden Prozess besteht aus dem „Parameternamen”, dem „Typ” des Parameter, der

Abbildung 4.28: Aggregat

„Dimension" des Parameter und dem „Wert". Die Parameterliste kann in der **externen Funktionseinheit** nicht verändert werden. Möchten Sie das Aufrufformat eines Dienstes ändern, so müssen Sie in der überliegenden **Funktionseinheit** die virtuellen Einparameter des entsprechenden Dienstes verändern.

Reiter Ausgabeparameter

Die in dieser Liste angezeigten Ausgabeparameter sind die Ausgabeparameter aus den importierten Prozessen und spezifizieren die verbundenen Prozessketten. Das einzelne Ausgabeparameterrecord besteht aus dem „Ausgabeparameternamen", dem „Typ" des Ausgabeparameter, der „Dimension" des Parameter und dem „Wert". Die Ausgabeparameterliste kann in der **externen Funktionseinheit** nicht verändert werden. Möchten Sie das Ausgabeformat eines Dienstes ändern, so müssen Sie in der überliegenden **Funktionseinheit** die virtuellen Ausgabeparameter des entsprechenden Dienstes verändern.

Reiter Darstellung

Mit Hilfe des Attributes „Hintergrundfarbe" kann die farbliche Darstellung eigenen Bedürfnissen (z.B. Markierung, Hervorhebung usw.) angepasst werden. Mit Hilfe des Attributes „Breite" können Sie die Breite der Darstellung festlegen. Mit Hilfe des Attributes „Höhe" können Sie die Höhe der Darstellung festlegen. Über „Anpassung" wird festgelegt, ob die Größe der Darstellung automatisch oder manuell festgelegt werden soll. Die restlichen Attribute können zum Ein- und Ausblenden der externen Funktionseinheit genutzt werden (siehe auch Anhang C).

4.14 Aggregat

Der ProC/B-Editor bietet die Möglichkeit, Teile einer Prozesskette zu aggregieren. Die erzeugten Ersatzdarstellungen können mit Hilfe des **Aggregats** (siehe Abbildung 4.28) wieder in einem Modell genutzt werden. Nach erfolgreicher Aggregierung über den Menüpunkt „Bearbeiten" $->$ „Aggregieren von..." wird im Editor automatisch ein Aggregatbaustein mit den entsprechenden Ergebnissen erzeugt. Zusätzlich besteht aber auch die Möglichkeit, ein **Aggregat** per Hand anzulegen, um zu einem früheren Zeitpunkt erzeugte Ergebnisse zu nutzen. Zur Vervollständigung der Beschreibung folgen die Attribute des **Aggregats**, die sich über das Attributfenster (Abbildung 4.29) editieren lassen.

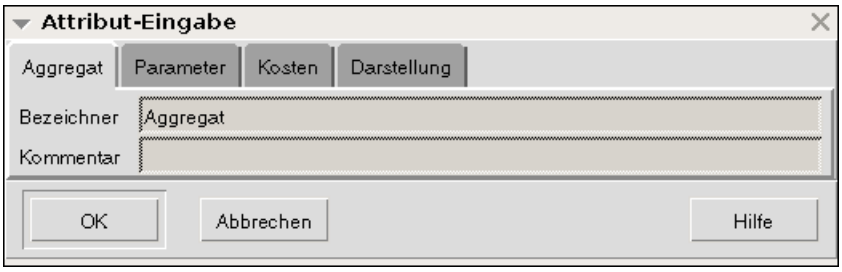


Abbildung 4.29: Attribute des Aggregats

Reiter Aggregat

Das Attribut „Bezeichner" erwartet einen Namen für das **Aggregat**. Auf diesen „Bezeichner" wird unter Umständen in anderen Elementen weiter Bezug genommen, weshalb er eindeutig gewählt werden muss. Eine Überprüfung der Eindeutigkeit des „Bezeichners" übernimmt der Editor. Beim Laden aus einer Datei mit Ergebnissen einer früheren Aggregierung wird der „Bezeichner" automatisch vom Editor gesetzt.

Das Attribut „Kommentar" ist für eine Kurzbeschreibung des **Aggregats** gedacht.

Reiter Parameter

Über den Browse-Button wird ein Menü zum Laden einer Aggregat-Datei geöffnet. Die Ersatzdarstellung wird vollständig in dem **Aggregat** gespeichert. Der angezeigte Pfad dient danach nur noch zur Information, aus welcher Datei die Ersatzdarstellung geladen wurde.

In der Liste wird der Inhalt der Aggregat-Datei angezeigt. Eine übersichtlichere Darstellung kann man sich in der Innenansicht des **Aggregats** anzeigen lassen.

Reiter Kosten

Mit dem Attribut „Fixkosten" können Sie die Kosten angeben, die auf jeden Fall für den Betrieb des **Aggregats** anfallen.

Mit dem Attribut „Betriebskosten" können Sie die Kosten angeben, die bei jeder Benutzung des **Aggregats** in Abhängigkeit der Zeit entstehen.

Mit dem Attribut „Leistungskosten pro Zeiteinheit" können Sie die Kosten angeben, die bei jedem Aufruf in Abhängigkeit der angegebenen Bedienzeit des **Aggregats** anfallen.

Mit dem Attribut „Leistungskosten" können Sie die Kosten angeben, die bei jedem Aufruf unabhängig von der Bedienzeit entstehen.

Reiter Darstellung

Mit Hilfe des Attributes „Hintergrundfarbe" kann die farbliche Darstellung eigenen Bedürfnissen (z.B. Markierung, Hervorhebung usw.) angepasst werden.

Mit Hilfe des Attributes „Breite" können Sie die Breite der Darstellung festlegen.

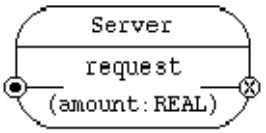


Abbildung 4.30: Server

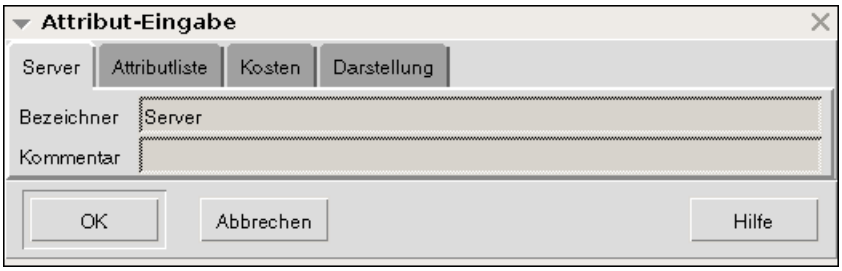


Abbildung 4.31: Attribute des Server

Über „Anpassung” wird festgelegt, ob die Größe der Darstellung automatisch oder manuell festgelegt werden soll. Die restlichen Attribute können zum Ein- und Ausblenden des Aggregats genutzt werden (siehe auch Anhang C).

4.15 Server

Ein **Server** (siehe Abbildung 4.30) kann über den Dienst *request* für bestimmte Zeitspannen angefordert werden und eignet sich zum Beispiel zur Modellierung einer Maschine. Im Folgenden werden die Attribute des **Servers** vorgestellt, die sich über das Attributfenster (Abbildung 4.31) editieren lassen.

Reiter Server

Das Attribut „Bezeichner” erwartet einen Namen für den **Server**. Auf diesen „Bezeichner” wird unter Umständen in anderen Elementen weiter Bezug genommen, weshalb er eindeutig gewählt werden muss. Eine Überprüfung der Eindeutigkeit des „Bezeichners” übernimmt der Editor.

Das Attribut „Kommentar” ist für eine Kurzbeschreibung des **Servers** gedacht.

Reiter Attributliste

Das Attribut „Geschwindigkeit” entspricht dem Attribut SPEED aus dem ProC/B-Formalismus.

Soll die „Geschwindigkeit” von der Population des **Servers** abhängen, so lassen sich auch die zustandsabhängigen Geschwindigkeiten eingeben: Als „SDSPEEDS” sind ein `INT`- und ein `REAL`-Vektor gleicher Länge einzugeben. Abhängig von der

Population des **Servers** wird die „Geschwindigkeit" mit dem entsprechenden REAL-Wert des zweiten Vektors multipliziert. Der INT-Vektor spezifiziert die Intervalle, für die der dazugehörige REAL-Wert des REAL-Vektors gilt. Der INT-Vektor muss bei 1 beginnen. Beispiel: SDSPEEDS$= [1, 7, 8, 12][1.2, 1.1, 0.7, 0.4]$ bedeutet, für Populationen von 1 bis 6 ist die Geschwindigkeit $1,2$-mal der als Attribut „Geschwindigkeit" eingetragene Wert, für die Population von 7 bzw. 8 bis 11 $1,1$- bzw. $0,7$-mal und für Populationen von 12 und größer $0,4$-mal der als Attribut „Geschwindigkeit" eingetragene Wert. Der Standard-Wert ist $[1][1.0]$.

Das Attribut „Disziplin" bietet wie im ProC/B-Modellformalismus eine Auswahl von Bedienstrategien an. Unterstützt werden die Bediendisziplinen FCFS (First-Come-First-Serve), PS (Processor Sharing), IS (Infinite Server), PRIONP (Non Preemptive), PRIONPCAP (Non Preemptive mit Kapazität), PRIOPREP (Preemptive Repeat), PRIOPRES (Preemptive Resume).

Bei FCFS werden die Anforderungen vom **Server** in der Reihenfolge ihres Auftretens abgearbeitet. Überschreitet die Anzahl der Anforderungen die Kapazität, so wird mit der Bearbeitung der letzten Anforderungen gewartet, bis Ressourcen frei werden. Bei PS werden alle Anforderungen gleichzeitig bearbeitet. Eine Überschreitung der Kapazität hat hier zur Folge, dass sich die Bearbeitungszeit für alle Anfragen verlängert. Mit der Bediendisziplin IS werden ebenfalls alle Anfragen gleichzeitig bearbeitet, der **Server** hat allerdings unendliche Kapazität.
Bei den Strategien PRIONP, PRIONPCAP, PIOPREP und PRIOPRES arbeitet der Server als Prio-Server und bedient zunächst die Arbeitsanforderungen mit einer hohen Priorität. Bei mehreren Prozessen mit gleicher Priorität wird die Reihenfolge zufällig gewählt. Als höchste Priorität gilt die Priorität 0, die niedrigste Priorität ist 32767. PRIOPREP unterbricht Prozesse mit niedriger Priorität für Anfragen höherer Priorität. Wird der unterbrochene Prozess später fortgesetzt, so wird die bis dahin verbrauchte Zeit ignoriert und er startet von vorn. PRIOPRES unterbricht Prozesse niedriger Priorität ebenfalls, sie werden allerdings später an der Stelle fortgesetzt, an der sie unterbrochen wurden. PRIONP dagegen unterbricht einmal begonnene Prozesse nicht. Bei der Disziplin PRIONPCAP werden zusätzlich Kapazitäten berücksichtigt. Zusätzliche Dokumentation zu den Disziplinen befindet sich in [5] in Anhang F.1.8.

Das Attribut „Kapazität" erwartet einen positiven ganzen Wert. Die „Kapazität" beschreibt die Anzahl der im **Server** zur Verfügung stehenden, potentiell simultan aktiven Arbeitseinheiten.

Reiter Kosten

Mit dem Attribut „Fixkosten" können Sie die Kosten angeben, die auf jeden Fall für den Betrieb des **Servers** anfallen.

Mit dem Attribut „Betriebskosten" können Sie die Kosten angeben, die bei jeder Benutzung des **Servers** in Abhängigkeit der Zeit entstehen.

Mit dem Attribut „Leistungskosten pro Zeiteinheit" können Sie die Kosten angeben, die bei jedem Aufruf in Abhängigkeit der angegebenen Bedienzeit des **Servers** anfallen.

Mit dem Attribut „Leistungskosten" können Sie die Kosten angeben, die bei jedem Aufruf unabhängig von der Bedienzeit entstehen.

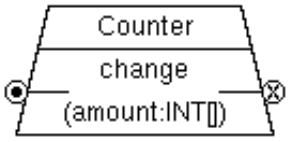


Abbildung 4.32: Counter

Abbildung 4.33: Attribute des Counter

Reiter Darstellung

Mit Hilfe des Attributes „Hintergrundfarbe" kann die farbliche Darstellung eigenen Bedürfnissen (z.B. Markierung, Hervorhebung usw.) angepasst werden.

Mit Hilfe des Attributes „Breite" können Sie die Breite der Darstellung festlegen.

Über „Anpassung" wird festgelegt, ob die Größe der Darstellung automatisch oder manuell festgelegt werden soll. Die restlichen Attribute können zum Ein- und Ausblenden des Servers genutzt werden (siehe auch Anhang C).

4.16 Counter

Ein **Counter** (siehe Abbildung 4.32) eignet sich zur Modellierung eines Lagers oder allgemein gesprochen eines begrenzten Puffers. Abstrakt gesprochen verwaltet ein **Counter** einen mehrdimensionalen diskreten Raum, dessen Elemente für unbestimmte Zeitspannen angefordert werden.

Der **Counter** bietet den Dienst *change* an; dessen Aufruf verändert die Anzahl der Elemente in den Dimensionen des Raumes um den angegebenen Betrag.

Im Folgenden werden die Attribute des **Counters** vorgestellt, die sich über das Attributfenster (Abbildung 4.33) editieren lassen.

Reiter Counter

Das Attribut „Bezeichner" erwartet einen Namen für den **Counter**. Auf diesen „Bezeichner" wird unter Umständen in anderen Elementen weiter Bezug genommen, weshalb er eindeutig gewählt werden muss. Eine Überprüfung der Eindeutigkeit des „Be-

zeichners" übernimmt der Editor. Das Attribut „Kommentar" ist für eine Kurzbeschreibung des **Counters** gedacht.

Reiter Attributliste

Das Attribut „Initialisierung" beschreibt die initiale Belegung (siehe auch Verwendung von Arrays, Abschnitt 4.20).

Das Attribut „Untergrenzen" beschreibt die minimal zulässige Belegung in den vorhandenen Dimensionen (siehe auch Verwendung von Arrays, Abschnitt 4.20).

Das Attribut „Obergrenzen" beschreibt die maximal zulässige Belegung in den vorhandenen Dimensionen. Die Eingaben der „Initialisierung", der „Untergrenzen" und der „Obergrenzen" müssen der Form im ProC/B-Formalismus entsprechen (siehe auch Verwendung von Arrays, Abschnitt 4.20). Für die Eingaben der initialen, minimalen und maximalen Belegung ist eine abkürzende Schreibweise möglich, die ebenfalls unter Verwendung von Arrays erläutert wird.

Das Attribut „Disziplin" legt die Wartedisziplin fest. Zur Zeit werden die Disziplinen `RANDOM`, `FCFS` (First-Come-First-Served) und `PRIO` unterstützt. `RANDOM` steht für zufällige Berücksichtigung einer Anforderung aus einer Menge von wartenden Anforderungen. Bei `FCFS` wird aus der Menge der wartenden und zu dem Zeitpunkt erfüllbaren Anforderungen zuerst die Anforderung berücksichtigt, die die längste Wartezeit hat. Bei `PRIO` wird zunächst die Anforderung ausgewählt, die die höchste Priorität hat. Als höchste Priorität gilt die Priorität 0, die niedrigste Priorität ist 32767. Zusätzliche Dokumentation zu den Disziplinen befindet sich in [5] in Anhang F.3. Die Dienste werden dort mit `CRANDOM`, `CFCFS` und `CPRIO` bezeichnet.

Reiter Kosten

Mit dem Attribut „Fixkosten" können Sie die Kosten angeben, die auf jeden Fall für den Betrieb des **Counters** anfallen.

Mit dem Attribut „Betriebskosten" können Sie die Kosten angeben, die bei jeder Benutzung des **Counters** in Abhängigkeit der Zeit entstehen.

Mit dem Attribut „Leistungskosten" können Sie die Kosten angeben, die bei jedem Aufruf entstehen.

Reiter Darstellung

Mit Hilfe des Attributes „Hintergrundfarbe" kann die farbliche Darstellung eigenen Bedürfnissen (z.B. Markierung, Hervorhebung usw.) angepasst werden.

Mit Hilfe des Attributes „Breite" können Sie die Breite der Darstellung festlegen.

Über „Anpassung" wird festgelegt, ob die Größe der Darstellung automatisch oder manuell festgelegt werden soll. Die restlichen Attribute können zum Ein- und Ausblenden des Counters genutzt werden (siehe auch Anhang C).

4.17 Storage

Ein **Storage** (siehe Abbildung 4.34) erweitert einen Counter um spezielle Überwachungsfunktionen, die helfen Informationen über den Zustand zu gewinnen. Durch

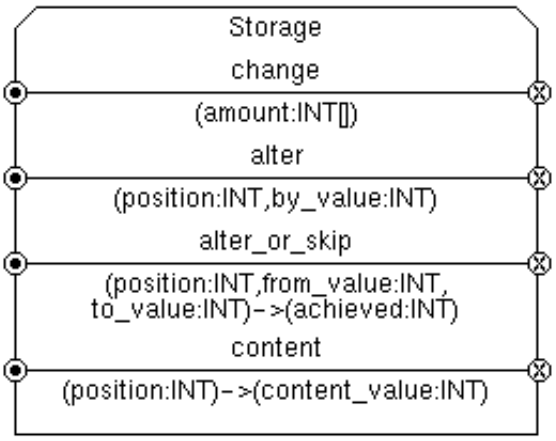


Abbildung 4.34: Storage

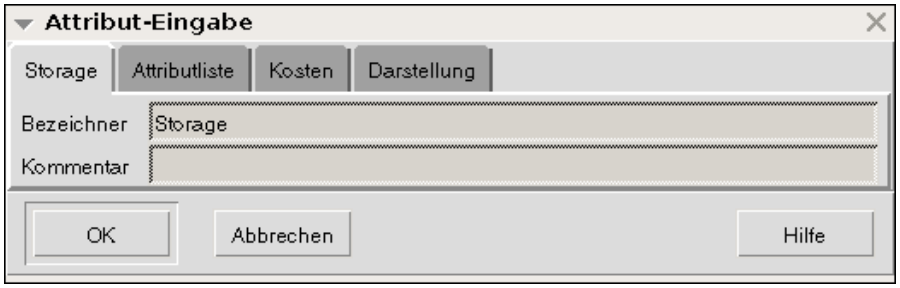


Abbildung 4.35: Attribute des Storage

zusätzliche Dienste wird außerdem die Benutzung vereinfacht. Ein Aufruf des Dienstes *change* ändert die Anzahl der Elemente in den Dimensionen des Raumes um den angegebenen Betrag. Ein Aufruf des Dienstes *alter* ändert die Anzahl an der angegebenen Position um den angegebenen Wert. Der Dienst *alter_or_skip* dient zur Behandlung von out-of-stock-Situationen und Teillieferungen: An der angegebenen Position wird möglichst die Menge „from_value", wenn das nicht möglich ist, so viel wie möglich, aber mindestens die Menge „to_value" entnommen (bzw. eingelagert). Ist auch das nicht möglich, so wird gar nichts entnommen (bzw. eingelagert). Der Rückgabewert informiert über die entnommene (bzw. eingelagerte) Menge. Ein Aufruf des Dienstes *content* liefert den Bestand an der angegebenen Position.

Im Folgenden werden die Attribute des **Storage** vorgestellt, die sich über das Attributfenster (Abbildung 4.35) editieren lassen.

Reiter Storage

Das Attribut „Bezeichner" erwartet einen Namen für das **Storage**. Auf diesen „Bezeichner" wird unter Umständen in anderen Elementen weiter Bezug genommen, wes-

halb er eindeutig gewählt werden muss. Eine Überprüfung der Eindeutigkeit des „Bezeichners" übernimmt der Editor. Das Attribut „Kommentar" ist für eine Kurzbeschreibung des **Storage** gedacht.

Reiter Attributliste

Das Attribut „Initialisierung" beschreibt die initiale Belegung (siehe auch Verwendung von Arrays, Abschnitt 4.20).

Das Attribut „Untergrenzen" beschreibt die minimal zulässige Belegung in den vorhandenen Dimensionen (siehe auch Verwendung von Arrays, Abschnitt 4.20).

Das Attribut „Obergrenzen" beschreibt die maximal zulässige Belegung in den vorhandenen Dimensionen. Die Eingaben der „Initialisierung", der „Untergrenzen" und der „Obergrenzen" müssen der Form wie im ProC/B-Formalismus haben (siehe auch Verwendung von Arrays, Abschnitt 4.20). Für die Eingaben der initialen, minimalen und maximalen Belegung ist eine abkürzende Schreibweise möglich, die ebenfalls unter Verwendung von Arrays erläutert wird.

Als Bedienstrategien stehen wie beim **Counter** `RANDOM`, `FCFS` und `PRIO` zur Verfüfung.

Das Attribut „Anzahl der Messströme" ist schreibgeschützt und wird nur noch für die interne Verarbeitung benutzt.

Reiter Kosten

Mit dem Attribut „Fixkosten" können Sie die Kosten angeben, die auf jeden Fall für den Betrieb des **Storage** anfallen.

Mit dem Attribut „Betriebskosten" können Sie die Kosten angeben, die bei jeder Benutzung des **Storage** in Abhängigkeit der Zeit entstehen.

Mit dem Attribut „Leistungskosten (Input)" können Sie die Kosten angeben, die bei jedem Einlagerungsaufruf anfallen.

Mit dem Attribut „Leistungskosten (Output)" können Sie die Kosten angeben, die bei jedem Auslagerungsaufruf anfallen.

Mit dem Attribut „Betriebskosten (State)" können Sie die Kosten angeben, die für jedes Stück und jede Zeiteinheit entstehen.

Reiter Darstellung

Mit Hilfe des Attributes „Hintergrundfarbe" kann die farbliche Darstellung eigenen Bedürfnissen (z.B. Markierung, Hervorhebung usw.) angepasst werden.

Mit Hilfe des Attributes „Breite" können Sie die Breite der Darstellung festlegen.

Über „Anpassung" wird festgelegt, ob die Größe der Darstellung automatisch oder manuell festgelegt werden soll. Die restlichen Attribute können zum Ein- und Ausblenden des Storages genutzt werden (siehe auch Anhang C).

4.18 Globale Variablen

Mit den **Globalen Variablen** (siehe Abbildung 4.36) lassen sich Variablen oder Rewards deklarieren, deren Existenz nicht von einem speziellen Prozess abhängt. Zusätz-

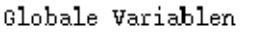


Abbildung 4.36: Globale Variable

Abbildung 4.37: Attribute der globalen Variable

lich können Dateien angegeben werden, aus denen während der Simulation grössere Datensätze in das Modell geladen werden.

So deklarierte Variablen sind in der **Funktionseinheit** bzw. dem Modellteil benutzbar, in der bzw. in dem sie definiert wurden, sowie in den darin enthaltenen **Funktionseinheiten**, sie werden deshalb als globale Variablen bezeichnet. Wird in einer enthaltenen **Funktionseinheit** eine Variable gleichen Namens deklariert, so ist dort nur diese Variable sichtbar, nicht die in einem übergeordneten Modellteil deklarierte Variable gleichen Namens. Rewards können zusätzlich zu den Standard-Messströmen für Messungen an **Funktionseinheiten** verwendet werden. (Siehe auch den Abschnitt 5.3 über Messpunkte und Verursacherpfade.)

Im Folgenden werden die Attribute der **Globalen Variablen** vorgestellt, die sich über das Attributfenster (Abbildung 4.37) editieren lassen.

Reiter Variablen

Das Attribut „Bezeichner" erwartet einen Namen für die **globalen Variablen**. Der „Bezeichner" der **globalen Variablen** ist für die weitere Modellierung und Analyse nicht relevant und kann zur zusätzlichen Kommentierung verwendet werden.

Mit dem Attribut „Typ" kann festgelegt werden, ob globale Variablen oder Rewards deklariert werden sollen.

Reiter Reward-Liste

Hierbei handelt es sich um eine Liste der selbstdefinierten Rewards. Als Typ stehen `COUNT`, `EVENT`, `STATE` zur Auswahl. Rewards vom Typ `EVENT` dienen dazu, aufeinanderfolgende Daten zu sammeln, `COUNT` dient zur Ermittlung von Ereignis-Raten und `STATE`-Rewards beschreiben Zustands-Trajektorien. Eine genauere Beschreibung

der drei Reward-Typen findet sich in den Auszügen aus dem Hi-Slang Reference Manual im Anhang. Die Eingabe der Array-Grenzen ist optional. Das Feld kann leer gelassen werden um einen einzelnen Reward zu deklarieren oder es können Arraygrenzen der Form [a..b] eingegeben werden um einen Array of Reward zu deklarieren.

Reiter Variablenliste

Hierbei handelt es sich um die Liste der global deklarierten Variablen. „Name", „Typ", „Dimension" und „initialer Wert" können festgelegt werden. Weiterhin besteht die Möglichkeit, einen „Kommentar" als Kurzbeschreibung für jede Variable festzulegen.

Reiter Dateivariablen

Hierbei handelt es sich um eine Liste der global deklarierten Dateivariablen. Der „Name" der Dateivariablen kann frei gewählt werden, muss aber eindeutig sein. Als Typ stehen `INFILE` (für Dateien aus denen gelesen werden soll) und `OUTFILE` (für Dateien in die geschrieben werden soll) zur Verfügung.

Reiter Dateizuordnungen

Über diese Liste lassen sich Dateien einem File-Identifier („Linkname") zuordnen. Der „Dateiname" ist dabei ein absoluter oder relativer Pfad zu einer Datei und „Linkname" ein Bezeichner, über den die Datei eindeutig identifiziert wird. Falls kein Dateiname angegeben wird, fordert HIT den Benutzter beim Start der Simulation auf, eine Datei anzugeben. Die hier angegebenen Dateien lassen sich über ein **Code-Element** mit dem Befehl

```
OPEN f, "LINKNAME", LENGTH 80;
```

öffnen. 'f' ist hier eine Dateivariable vom Typ `INFILE` oder `OUTFILE`, `'LINKNAME'` steht für einen File-Identifier, der in dem Reiter Dateizuordnungen angegeben wurde. Die Dateizuordnungen können nur auf der obersten Ebene der Modellhierarchie angegeben werden.

Reiter Darstellung

Mit Hilfe des Attributes „Hintergrundfarbe" kann die farbliche Darstellung eigenen Bedürfnissen (z.B. Markierung, Hervorhebung usw.) angepasst werden. Die restlichen Attribute können zum Ein- und Ausblenden der Variablen genutzt werden (siehe auch Anhang C).

4.19 Kommentar

Kommentare (siehe Abbildung 4.38) dieser Art sind nicht im ProC/B-Modellformalismus festgelegt. Ihre Verwendung ist optional und hat keinen Einfluss auf die Analyse des Modells. Sie dienen zur freien Platzierung von mehrzeiligem Kommentar auf der Zeichenfläche. Im Folgenden werden die Attribute des **Kommentars** vorgestellt, die sich über das Attributfenster (Abbildung 4.39) editieren lassen.

{Kommentar}

Abbildung 4.38: Kommentar

Abbildung 4.39: Attribut des Kommentars

Reiter Kommentar

Das Attribut „Bezeichner" erwartet einen Namen für den **Kommentar**. Der „Bezeichner" des **Kommentars** ist für die weitere Modellierung und Analyse nicht relevant.

Reiter Kommentarliste

Hierbei handelt es sich um die Liste einzelner Zeilen des Kommentars. Die jeweils erste und letzte Zeile eines Kommentars werden bei der Darstellung automatisch um eine öffnende bzw. schließende Klammer ergänzt.

Reiter Darstellung

Mit Hilfe des Attributes „Hintergrundfarbe" kann die farbliche Darstellung eigenen Bedürfnissen (z.B. Markierung, Hervorhebung usw.) angepasst werden. Die restlichen Attribute können zum Ein- und Ausblenden des Kommentars genutzt werden (siehe auch Anhang C).

4.20 Verwendung von Arrays

Der nachfolgende Abschnitt beschreibt die Definition (Festlegung der Dimension) und Initialisierung von Arrays. Für die Dimension des Arrays gibt es folgende Notationen:

- Eindimensionale Arrays: $[x_1..x_n]$, wobei $x_1..x_n$ ganze Zahlen sind. Beispiel: $[4..16]$
- Mehrdimensionale Arrays: $[a_1..a_n, b_1..b_n, .., x_1..x_n]$, wobei $a_1, ..a_n, b_1..b_n$, $x_1..x_n$ ganze Zahlen sind. Beispiel: $[1..3, 1..4, 3..5]$

Um eine Initialisierung der Arrays vorzunehmen, gibt es zwei verschiedene Notationen:

1. Die Hi-Slang-Notation, die für eindimensionale Arrays folgendermaßen aussieht $[Wert_1, Wert_2, .., Wert_N]$. Eine Initialisierung von einem zweidimensionalen Array wird an dem folgenden Beispiel erläutert. In unseren Beispiel sei die Dimension $[1..3, 1..4]$, d.h. es müssen $3 * 4$ Werte initialisiert werden. Eine mögliche Wertzuweisung für unser Beispiel muss also 3 Arrays der Länge 4 initialisieren, beispielsweise $[[11, 12, 13, 14], [21, 22, 23, 24], [31, 32, 33, 34]]$.

2. Die verkürzende Schreibweise für die Arrayinitalisierung, die für ein eindimensionales Array zum Beispiel so verwendet wird: $[\{1, 2\} : 2, 3, 4]$. Der Inhalt der geschweiften Klammern wird stets vervielfacht. Die Zahl nach dem Doppelpunkt hinter der schliessenden Klammer gibt die Anzahl der Vielfachung an. Das Ergebnis des Beispiels sieht in der Hi-Slang-Notation so aus : $[1, 2, 1, 2, 3, 4]$. Natürlich können die Ausdrücke beliebig verschachtelt werden. Für ein Array mit der Dimension $[1..2, 1..8]$ wird die Kurzschreibweise $[[\{\{1\} : 3, 3\} : 2], [\{\{2, 3\} : 2\} : 2]]$ zum Beispiel in $[[1, 1, 1, 3, 1, 1, 1, 3], [2, 3, 2, 3, 2, 3, 2, 3]]$ umgewandelt. Die vorgestellte Kurzschreibweise soll eine bequeme Möglichkeit bieten, lange Arrays mit gleichem Wert zu initialisieren.

4.21 Wahrscheinlichkeitsverteilungen

Einige ProC/B-Elemente verfügen über Parameter, die das Verhalten der Prozessketten beeinflussen, zum Beispiel Quellen, die die Entstehung von Prozessen beschreiben. Die Werte dieser Parameter können durch Wahrscheinlichkeitsverteilungen festgelegt werden. Im Editor werden die Verteilungen von Hi-Slang unterstützt:

1. Cox-Verteilung
   cox (A, B: REAL): REAL
   definiert eine COX-verteilte Pseudozufallszahl mit Rate A (reziproker Mittelwert) und Variationskoeffizient B. Dabei muss A $\geq$ 0 und B $\geq$ 0.1 sein.

   coxg (A: ARRAY OF REAL): REAL
   definiert eine COX-verteilte Pseudozufallszahl. A ist ein zweidimensionaler REAL-Array. Die erste Zeile gibt die Service-Raten an, die zweite Zeile die Wahrscheinlichkeiten für den schrittweisen Ablauf. Das letzte Array-Element der zweiten Zeile muss 0 sein, alle anderen Elemente positiv.

2. discrete (A: ARRAY OF REAL): INTEGER
   Das eindimensionale Array A, der aus Sicherheitsgründen durch das Element 1 ergänzt wird, wird als Treppenfunktion des Index interpretiert, die eine diskrete kumulative Verteilungsfunktion beschreibt. Alle Array-Elemente müssen in aufsteigender Reihenfolge aus dem Intervall [0, 1] stammen. Das Ergebnis ist der kleinste Index i, so dass A[i] $\geq$ u, wobei u der Wert einer einfachen Ziehung ist.

3. Bernoulli-Variable
   draw (A: REAL): BOOLEAN
   Das Ergebnis ist TRUE mit der Wahrscheinlichkeit A und FALSE mit der Wahrscheinlichkeit 1-A.

4. Erlang-Verteilung
   erlang (A, B: REAL): REAL
   definiert eine Erlang-Verteilung mit Mittelwert 1/A und Standardabweichung 1/(A*sqrt(B)). Die Rate A und die Phasenzahl B müssen beide größer als 0 sein.

5. histd (A: ARRAY OF REAL): INTEGER
   Der eindimensionale Array A wird als Histogramm interpretiert, das die relative Häufigkeit der Array-Elemente angibt. Die relativen Häufigkeiten müssen ≤ 1 sein, die Summe aller Häufigkeiten gleich 1. Als Ergebnis wird ein Index aus dem Bereich A.lower_bounds[1]..A.upper_bounds[1] geliefert.

6. Linear-Verteilung
   linear (A, B: ARRAY OF REAL): REAL
   A und B müssen REAL-Arrays mit den gleichen Unter- und Obergrenzen lb und ub sein. Sie definieren eine kumulative Verteilung f. Das Ergebnis wird durch lineare Interpolation in der nicht-äquidistanten Verteilungstabelle ermittelt und ist definiert durch A und B.
   Die Funktion f ist definiert durch A[i] = f(B[i]) für lb $\leq$ i $\leq$ ub
   Folgende Bedingungen müssen eingehalten werden:
   A[lb] = 0, A[ub] = 1 und beide Arrays müssen monoton sein, z.B. für lb $\leq$ i $\leq$ ub gilt A[i] $\leq$ A[i+1] und B[i] $\leq$ B[i+1]

7. Negativ-exponentiale Verteilung
   negexp (A: REAL): REAL
   definiert eine negativ-exponentiale Verteilung mit Mittelwert 1/A (bzw. Rate A) Dies entspricht der Wartezeit zwischen zwei Ankünften eines Poisson-verteilten Ankunftsprozesses mit Erwartungswert A für die Anzahl der Ankünfte je Zeiteinheit. A muss größer als 0 sein.

8. Normalverteilung
   normal (A, B: REAL): REAL
   definiert eine Normalverteilung mit Mittelwert A und Standardabweichung B. B darf nicht negativ sein.

9. Poisson-Verteilung
   poisson (A: REAL): INTEGER
   definiert eine Poisson-Verteilung mit Parameter A. Das Ergebnis n wird definiert durch n+1 Basisziehungen u(i), so dass n die kleinste nicht-negative Zahl mit u(0)*u(1)*...*u(n) < exp(-A) ist. Für negative A ist das Ergebnis 0.

10. Gleichverteilung
    randint (A, B: INTEGER): INTEGER
    definiert eine Gleich-Verteilung über die ganzen Zahlen A, A+1, ..., B-1, B mit A $\leq$ B.

    uniform (A, B: REAL): REAL
    definiert eine Gleich-Verteilung über die reellen Zahlen im Intervall [A, B] mit A $\leq$ B.

11. Beta-Verteilung
    beta (A1, A2: REAL): REAL
    definiert eine Beta-Verteilung mit den Formparametern A1 und A2.
    Falls A1 ≤ 0 oder A2 ≤ 0 bricht die Simulation mit einem Fehler ab. Falls sich A1 oder A2 in der Epsilon-Umgebung von 0 befinden, werden sie so behandelt als wären sie 0.
    Als Ergebnis wird der Quotient von Gamma-Verteilungen zurückgegeben:
    $beta(A1, A2) \sim gamma(A1, 1)/(gamma(A1, 1) + gamma(A2, 1))$
    Wenn die Summe der beiden Gamma-Verteilungen Null ist, ist auch das Ergebnis der Beta-Verteilung Null.

12. Gamma-Verteilung
    gamma (A, B: REAL): REAL
    definiert eine Gamma-Verteilung mit Formparameter A und Skalierungsparameter B.
    Falls A ≤ 0 oder B ≤ 0 bricht die Simulation mit einem Fehler ab. Falls sich A oder B in der Epsilon-Umgebung von 0 befinden, werden sie so behandelt als wären sie 0.

13. Weibull-Verteilung
    weibull (A, B: REAL): REAL
    definiert eine Weibull-Verteilung mit Formparameter A und Skalierungsparameter B.
    Falls A ≤ 0 oder B ≤ 0 bricht die Simulation mit einem Fehler ab. Falls sich A oder B in der Epsilon-Umgebung von 0 befinden, werden sie so behandelt als wären sie 0.

Kapitel 5

Experimente

Dieses Kapitel beschreibt das Spezifizieren eines Experiments.

5.1 Experimentserie

Nachdem das Erstellen eines Prozesskettenmodells in den vorangegangenen Kapiteln beschrieben wurde, kommen wir nun zur Analyse des Modells. Bevor eine Analyse des Modells erfolgen kann, muss erst einmal ein Experiment spezifiziert werden, d.h. dass festgelegt werden muss, welche Werte an welchen Elementen des Modells gemessen werden sollen.

Der ProC/B-Editor ist lediglich in der Lage, aus der Experimentspezifikation eine procb-Datei zu generieren. Eine Analyse des spezifierten Experiments ist außerhalb des Editors angedacht, zum Beispiel im ProC/B-Analysator. In [6] wird beispielhaft beschrieben, wie eine solche Analyse durchgeführt werden kann.

5.2 Auswahl der Analyse

Der erste Schritt zum Anlegen eines Experiments ist die Auswahl der gewünschten Analyse. Eine Experimentserie wird über das Hierarchiefenster (Teil des Hauptfensters, s. Abschnitt 3.1.1) angelegt. Dazu klicken Sie mit der linken Maustaste auf das Symbol derjenigen Funktionseinheit, in der Sie eine Experimentserie durchführen wollen.

Nun erscheint ein Menü, indem Sie den Eintrag ’Experimentserie anlegen’ anwählen. Daraufhin erscheint ein Untermenü, indem Sie ’ProC/B-Analyse’ auswählen können. Zur Zeit ist die ’ProC/B-Analyse’ der einzige zur Verfügung stehende Löser.

Mit der Wahl der Funktionseinheit für die Experimentserie sind auch alle untergeordneten Funktionseinheiten mitausgewählt. Funktionseinheiten, die nicht in der Hierarchie der gewählten Funktionseinheit sind, können in der neuen Experimentserie nicht berücksichtigt werden. Beim Anlegen einer Experimentserie werden alle geöffneten Editierfensterfenster in Experimentbeschreibungsfenster umgewandelt (siehe Abbildung 5.2) und das Hauptfenster wird in ein Experimentfenster umgewandelt (siehe Abbildung 5.1).

Möchten Sie nun Änderungen an ihrem Modell durchführen, müssen Sie erst das Experiment schließen. Während der Definition des Experimentes sind keine modell-

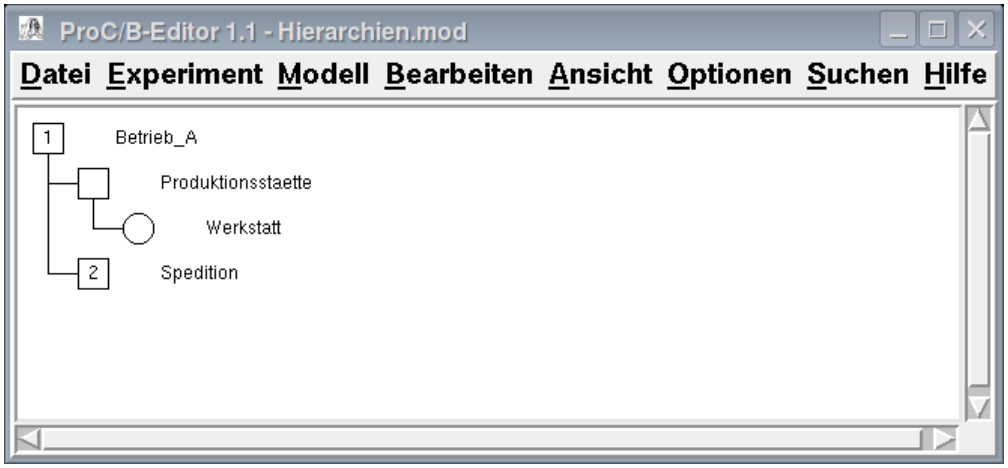


Abbildung 5.1: Hauptfenster

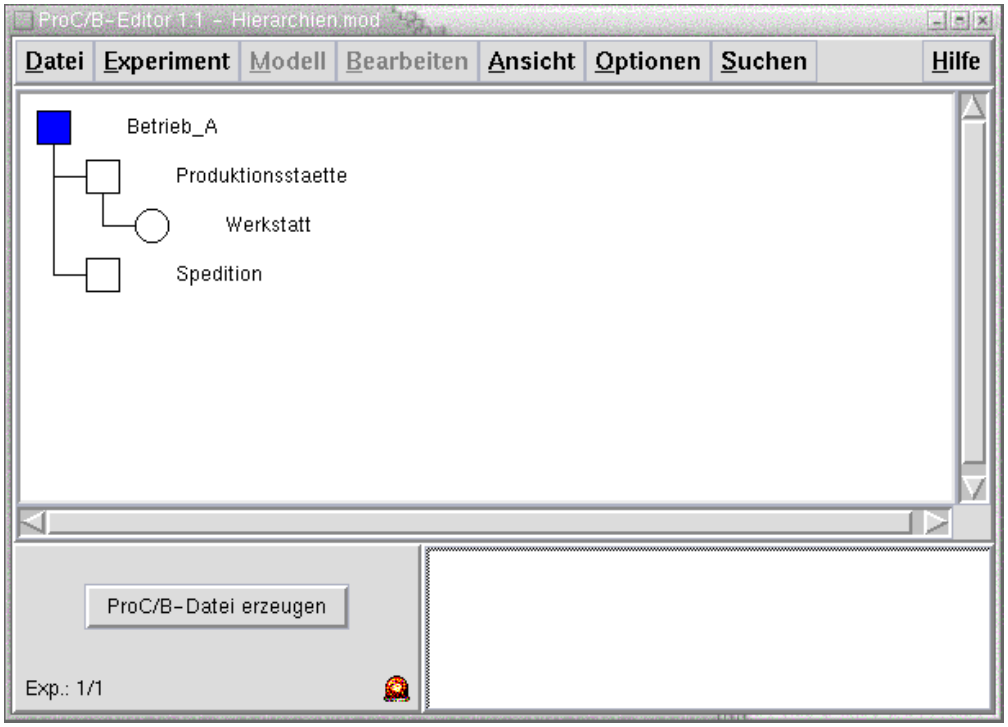


Abbildung 5.2: Experimentfenster

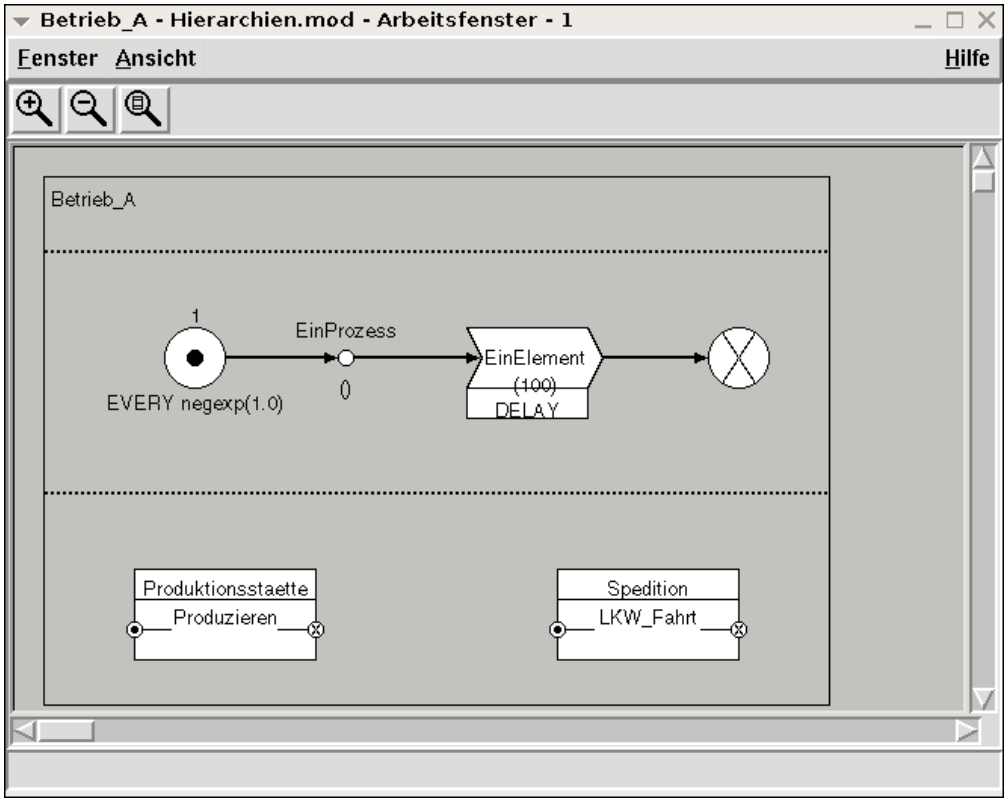


Abbildung 5.3: Experimentbeschreibungsfenster

spezifischen Änderungen möglich. Sie sollten sicherstellen, dass Ihr Prozesskettenmodell zuvor abgespeichert ist. Durch das Speichern wird sichergestellt, dass sich bei einem späteren Laden eines Experiments das zugehörige Modell zuverlässig identifizieren läßt.

5.3 Messpunkte und Verursacherpfade definieren

Zur Beschreibung eines Experiments ist es notwendig, Messpunkte und Verursacherpfade anzulegen. Messpunkte können an jeder **Funktionseinheit**, an jedem **Aggregat** und an jedem **Storage**, **Counter** oder **Server** definiert werden. Ein entsprechendes Menü bekommen Sie, indem Sie mit der Maus auf eine Funktionseinheit gehen und dann die linke Maustaste drücken.

Gemessen werden kann an jeder **Funktionseinheit**, jedem **Aggregat**, jedem **Storage**, jedem **Server** und jedem **Counter** der `Throughput`, die `Population`, die `Turnaroundtime` und die `Occupation`. `Throughput` beschreibt den Durchsatz, `Population` die durchschnittliche Anzahl Prozesse, die den Baustein nutzen, und `Turnaroundtime` die Verweildauer der Prozesse.

Zusätzlich kann an jedem **Server** noch die `Utilization` und an jedem **Storage** können die Werte `State`, `In`, `Out`, `OutOfStock_Calls`, `OutOfStock_Amount`, `OutOfSpace_Calls` und `OutOfSpace_Amounts` gemessen werden.

`Utilization` beschreibt die Auslastung des **Servers**. `State` dient zur Messung des durchschnittlichen Zustandes des **Storage**, `In` und `Out` zur Messung der Ein- bzw. Auslagerungen. `OutOfStock_Calls` und `OutOfStock_Amount` dienen zum Erfassen von Out-of-Stock-Situationen, wie sie bei der Benutzung des Dienstes `alter_or_skip` des **Storage** auftreten können. `OutOfStock_Calls` misst, wie oft der Lagerbestand nicht ausreichte, um die Anfrage zu bedienen, mit `OutOfStock_Amount` wird gemessen, wieviel Elemente jeweils gefehlt haben. Analog dienen `OutOfSpace_Calls` und `OutOfSpace_Amount` zum Erfassen von Situationen, in denen die Lagerkapazität nicht ausreichte, um eine Einlagerung vollständig durchzuführen.

An **Funktionseinheiten** können neben den Standard-Messströmen auch selbstdefinierte Messströme (Rewards) gemessen werden. Über das Element **Globale Variablen** (siehe Abschnitt 4.18) lassen sich innerhalb einer **Funktionseinheit** Rewards deklarieren. Im Experimentfenster stehen diese Rewards bei der Auswahl der Messpunkte ebenfalls zur Verfügung. Updates an den Rewards muss der Nutzer mit Hilfe eines **Code-Elementes** (siehe Abschnitt 4.5) im Modell selbst vornehmen.
Es folgt eine Zusammenfassung der Standard-Messströme. Eine ausführlichere Beschreibung dieser Messströme befindet sich in [5].

5.3.1 Throughput (Durchsatz)

- Messbar an: **Funktionseinheit, Server, Storage, Counter, Aggregat**
- Bedeutung: Bei der Simulation liefert der Mittelwertschätzer des Throughputs die mittlere Anzahl der Prozesse pro Zeiteinheit, die die Funktionseinheit verlassen. Der Schätzer der Standardabweichung bezieht sich auf die Zwischen-

abgangszeiten. Der Messstrom wird jeweils aktualisiert, wenn ein Prozess die Funktionseinheit verlässt.

- Besonderheiten: Aufrufe des Dienstes `content` beim **Storage** haben keine Auswirkungen auf den Throughput.

5.3.2 Population (Anzahl)

- Messbar an: **Funktionseinheit, Server, Storage, Counter, Aggregat**
- Bedeutung: Bei der Simulation liefert der Mittelwertschätzer Informationen über die mittlere Anzahl der Prozesse in der Funktionseinheit pro Zeiteinheit.

5.3.3 Tournaroundtime (Verweildauer)

- Messbar an: **Funktionseinheit, Server, Storage, Counter, Aggregat**
- Bedeutung: Bei einer Simulation liefert der Mittelwertschätzer die mittlere Zeitspanne, die ein Prozess in der Funktionseinheit verbringt. Der Messstrom wird aktualisiert, wenn ein Prozess die Funktionseinheit verlässt.

5.3.4 Utilization (Auslastung)

- Messbar an: **Server**
- Bedeutung: Die Utilization misst die Bediengeschwindigkeit, die Prozessen gewidmet wird. Diese entspricht der Anzahl der Prozesse, die der betreffende Server pro Zeiteinheit abwickelt, geteilt durch die Anzahl der Prozesse, die der Server pro Zeiteinheit abwickeln kann. Bei einer Simulation liefert der Mittwelwertschätzer die mittlere Auslastung der Funktionseinheit. Bei SD-Servern werden bei der Berechnung der Utilization die SD-Speeds berücksichtigt.

5.3.5 Occupation

- Messbar an: **Funktionseinheit, Server, Storage, Counter, Aggregat**
- Bedeutung: Bei der Simulation misst der Mittwelwertschätzer der Occupation die Wahrscheinlichkeit dafür, dass die Funktionseinheit belegt ist. Der Messstrom wird aktualisiert, wenn ein Prozess eine FE erreicht, in der sich zuvor kein anderer Prozess befunden hat, oder wenn der letzte Prozess die FE verlässt.

5.3.6 State (Zustand)

- Messbar an: **Storage**
- Bedeutung: Der Messstrom misst für jede Lager-Komponente die Belegung dieser Komponente.

5.3.7 In (Einlagerungen)

- Messbar an: **Storage**
- Bedeutung: Der Messstrom misst für jede Lager-Komponente die Einlagerungsmenge pro Einlagerung. Die Einlagerung von 0 gilt nicht als Einlagerung.

5.3.8 Out (Auslagerungen)

- Messbar an: **Storage**
- Bedeutung: Der Messstromm misst für jede Lager-Komponente die Auslagerungsmenge pro Auslagerung. Die Auslagerung von 0 gilt nicht als Auslagerung.

5.3.9 OutOfStockCalls (OutOfStock-Aufrufe)

- Messbar an: **Storage**
- Bedeutung: Der Messstromm misst die Anzahl der Aufrufe pro Zeiteinheit, bei denen die Lagermenge nicht ausreichte, um die Auslagerung vollständig durchzuführen.

5.3.10 OutOfStockAmount (OutOfStock-Menge)

- Messbar an: **Storage**
- Bedeutung: Der Messstromm misst die Menge, die bei Auslagerungen nicht ausgelagert werden konnte, pro nicht vollständig ausgeführter Auslagerung.

5.3.11 OutOfSpaceCalls (OutOfSpace-Aufrufe)

- Messbar an: **Storage**
- Bedeutung: Der Messstromm misst die Anzahl der Aufrufe pro Zeiteinheit, bei denen die Lagerkapazität nicht ausreichte, um die Einlagerung vollständig aufzunehmen.

5.3.12 OutOfSpaceAmount (OutOfSpace-Menge)

- Messbar an: **Storage**
- Bedeutung: Der Messstromm misst die Menge, die bei Einlagerungen nicht eingelagert werden konnte, pro nicht vollständig ausgeführter Einlagerung.

Sie haben nun zwei Möglichkeiten: Entweder Sie geben für den Messpunkt einen vordefinierten Verursacherpfad an oder Sie messen ALL. Mit Hilfe von Verursacherpfaden können Sie verursacherbezogene Messungen durchführen. Verursacherpfade bestehen aus einem Tripel: Component, Service, UsedService. Verursacherpfade können Sie in jeder Funktionseinheit und Standard-FE (Counter, Server usw.) definieren. Dazu gehen Sie mit der Maus zum Beispiel auf eine Funktionseinheit und drücken die linke Maustaste und wählen anschließend den Menüeintrag „neuer Verursacherpfad”.

Es erscheint nun ein kleines Fenster, in das Sie den Namen des Verursacherpfades eingeben sollten. Eine Überprüfung des Pfadnamens auf Eindeutigkeit und auf syntaktische Korrektheit übernimmt der Editor. Ausgehend von dieser Funktionseinheit können Sie diesen erzeugten Verursacherpfad stetig erweitern, indem Sie den Verursacherpfad durch die Hinzunahme von Funktionseinheiten, ProzessIDs und AufrufPKEs ergänzen.

Eine Ergänzung des Pfades kann folgendermaßen erfolgen: Gehen Sie in einem Experimentfenster mit der Maus auf eine Funktionseinheit, eine ProzessID oder ein AufrufPKE und drücken Sie die linke Maustaste. Sie sehen in dem erscheinenden Menü zum einen die aktuelle Pfadspezifikation, d.h. welche Elemente des Modells bereits in dem aktuellen Verursacherpfad enthalten sind, und zum anderen den Eintrag „ergänzen mit <Element-Name>". Für Funktionseinheiten kann dieses Menü auch direkt im Hauptfenster des Editors ausgewählt werden. Falls Sie den Eintrag „ergänzen mit <Element-Name>" wählen, wird das entsprechende Element in die Pfadspezifikation aufgenommen. Die Reihenfolge, in der die Elemente in den Pfad aufgenommen werden, spielt hierbei keine Rolle, da der ProC/B-Editor den Pfad richtig sortiert. Möchten Sie die Pfadspezifikation des aktuellen Verursacherpfads beenden, dann gehen Sie erneut auf die Funktionseinheit für die der Pfad angelegt wurde und wählen dann den Menüeintrag „ergänzen mit <Element-Name>". Falls der gewählte Pfad unvollständig ist, weil er z.B. nicht aus jeder Ebene der Hierarchie bis zu der FE, an der gemessen werden soll, ein Element enthält, wird er von dem Editor automatisch ergänzt. Ein Fenster informiert über die angelegten Pfade. Wird jedoch eine Funktionseinheit in den Pfad aufgenommen, von der in einer anderen FE Dienste importiert werden, kann es vorkommen, dass mehrere Teilpfade möglich sind. In diesem Fall kann durch einen Dialog ausgewählt werden, welcher Teilpfad übernommen werden soll. Ein Beispiel zeigt Abbildung 5.4. Ein Dienst der Funktionseinheit *Spedition* wird in der FE *Fabrik* über eine externe Funktionseinheit genutzt. Durch Aufnahme der FE *Spedition* in den noch leeren Verursacherpfad entstehen zwei mögliche Pfade: Der Verursacherpfad, bei dem die *Spedition* über die externe FE erreicht wird und der Pfad, bei dem die FE direkt erreicht wird. Der Nutzer kann in diesem Fall auswählen, welche Pfade übernommen werden sollen.

Beim Anlegen von Verursacherpfaden wird der Nutzer durch die farbige Darstellung der FEs im Hauptfenster unterstützt (siehe Abbildung 5.5). Die Funktionseinheit, an der der Pfad angelegt wird, ist rot dargestellt. Bereits im Pfad enthaltene Funktionseinheiten werden gelb markiert und FEs, die noch aufgenommen werden können, werden blau gezeichnet.

Sind Sie mit der Beschreibung des Experiments fertig, dann haben Sie die Möglichkeit die Ergebnisse abzuspeichern. Dazu wählen Sie im Hauptfenster den Menüpunkt „Experiment" und davon den Unterpunkt „Experiment speichern".

5.4 Experimentserie starten

Nachdem die Lastpfade erfolgreich definiert worden sind, können Sie die Experimentserie starten. Bevor eine .procb-Datei aus dem Modell erzeugt wird, sollten Sie jedoch sicher stellen, dass die Experimentoptionen richtig eingestellt sind. Den dafür benötigten Dialog erhalten sie über das Menü Experiment unter dem Eintrag Experi-

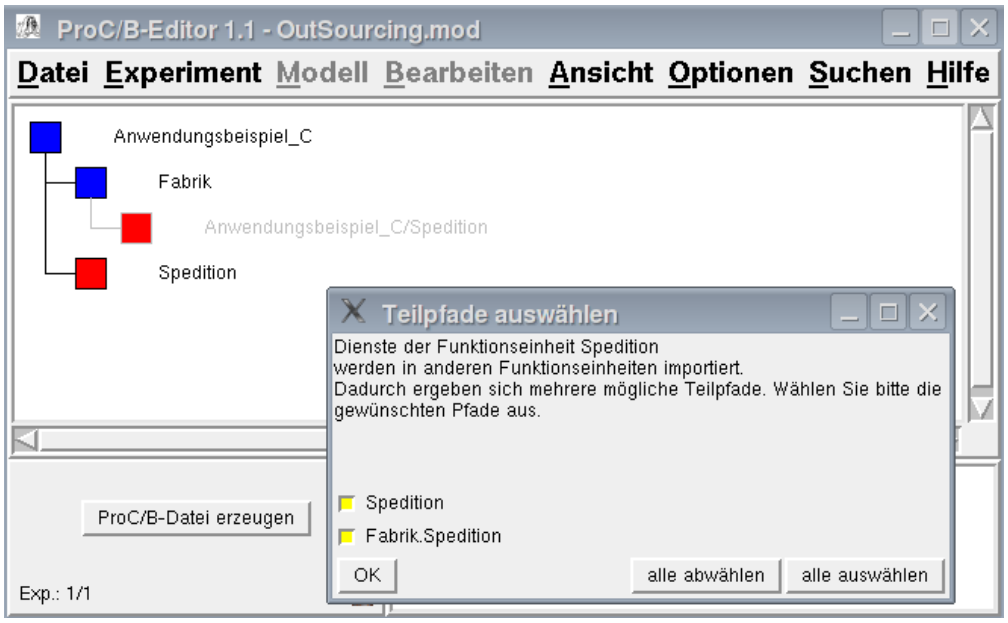


Abbildung 5.4: Auswahl von Teilpfaden

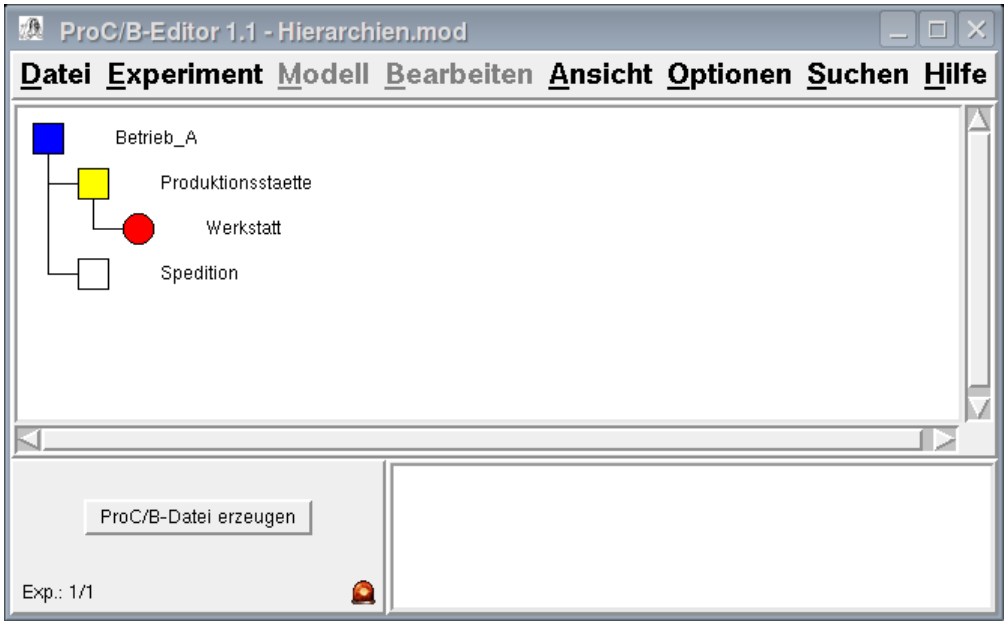


Abbildung 5.5: Farbige Darstellung der FEs im Hauptfenster

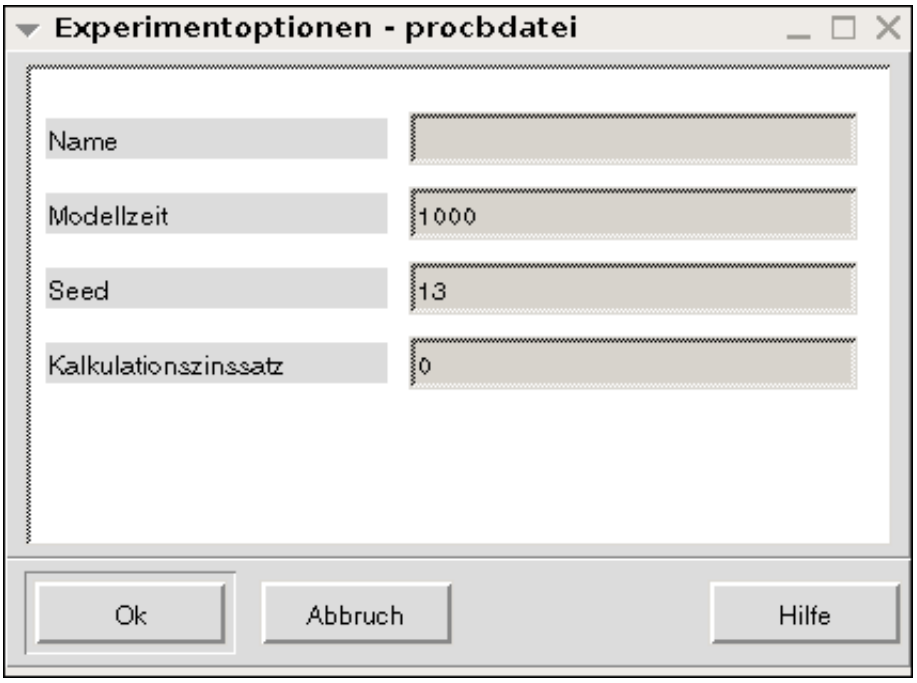


Abbildung 5.6: Experimentoptionen

mentoptionen. Zu den möglichen Einstellungen gehören die Modellzeit, der Seed und der Kalkulationszinssatz.

Durch Aktivieren des Buttons „ProC/B-Datei erzeugen" in der Steuerleiste des Hauptfensters wird das Experiment gestartet.
Das Starten der Experimentserie wird zur Zeit nur von der Linux- und der Sun-Version des ProC/B-Editors unterstützt.

5.5 Experimentserie laden

Prinzipiell haben Sie auch die Möglichkeit, eine zuvor gespeicherte Experimentserie (d.h. deren Einstellungen und Ergebnisse) zu laden. Falls die Experimentserie nicht zu dem gerade geöffneten Modell passt, erscheint ein Hinweis, ob das Experiment für dieses Modell übernommen oder das zu dem Experiment gehörende Modell geladen werden soll. Eine Übernahme des Experiments erscheint z.B. dann sinnvoll, wenn das Experiment mit einer älteren Version des Modells erstellt wurde und sich inzwischen die Hierarchiestruktur des Modells geändert hat. In diesem Fall kann das alte Experiment so weiterhin genutzt werden, einige Verursacherpfade, die in dem Modell nicht mehr gültig sind, werden aber eventuell ignoriert.

5.6 Ergebnisdarstellung

Zur Zeit ist keine besondere Ergebnisdarstellung realisiert. Die erzeugte .procb-Datei kann aber von anderen Tools importiert und analysiert werden. Analysen mit Ergebnisdarstellung können zum Beispiel mit dem Analysator aus dem ProC/B-GUI durchgeführt werden.

Kapitel 6

Anwendungsbeispiele

In diesem Kapitel werden drei Anwendungsbeispiele vorgestellt. Ein weiteres Anwendungsbeispiel befindet sich in [6], wo die Modellierung dieses Modells sowie dessen Auswertung beschrieben wird.

6.1 Ein einfaches Beispiel: Kartoffelküche

In diesem Beispiel soll eine Kartoffelküche modelliert werden, in der Kartoffeln zubereitet werden. Die Modelldateien zu diesem Beispiel befindet sich als
`beispiel.kartoffelkueche.schritt1.mod` und
`beispiel.kartoffelkueche.schritt2.mod`
im Ordner `Beispielmodelle` im Editorverzeichnis.
In dem ersten Modellierungsschritt besteht die Zubereitung der Kartoffeln nur aus den beiden Schritten 'Schaelen' und 'Kochen', die jeweils durch ein Delay-PKE mit einem Zeitverbrauch von 7 Zeiteinheiten dargestellt werden (siehe Abbildung 6.1).

In einem zweiten Modellierungsschritt (Abbildung 6.2) wird das Modell verfeinert. Durch einen Oder-Konnektor wird zwischen der Zubereitung von Salzkartoffeln und Pellkartoffeln unterschieden. Die einzelnen Zubereitungsschritte werden wieder durch Delay-PKEs dargestellt.

6.2 Ein komplexeres Beispiel: Güterverteilzentrum (GVZ)

An dieser Stelle erfolgt nur eine kurze Bechreibung des Güterverteilzentrums. Ausführlichere Erläuterungen befinden sich in [3]. Die zugehörige Modelldatei finden Sie in
`Beispielmodelle/beispiel.gvz.mod`
Abbildung 6.3 zeigt die Aussenansicht des GVZ. In regelmässigen Abständen erreichen LKWs, Züge und Huckepack-Verkehr das GVZ und werden von der FE 'Terminal' bearbeitet. In der Innenansicht (Abbildung 6.4) der FE 'Terminal' werden die Be- und Entladevorgänge beschrieben.

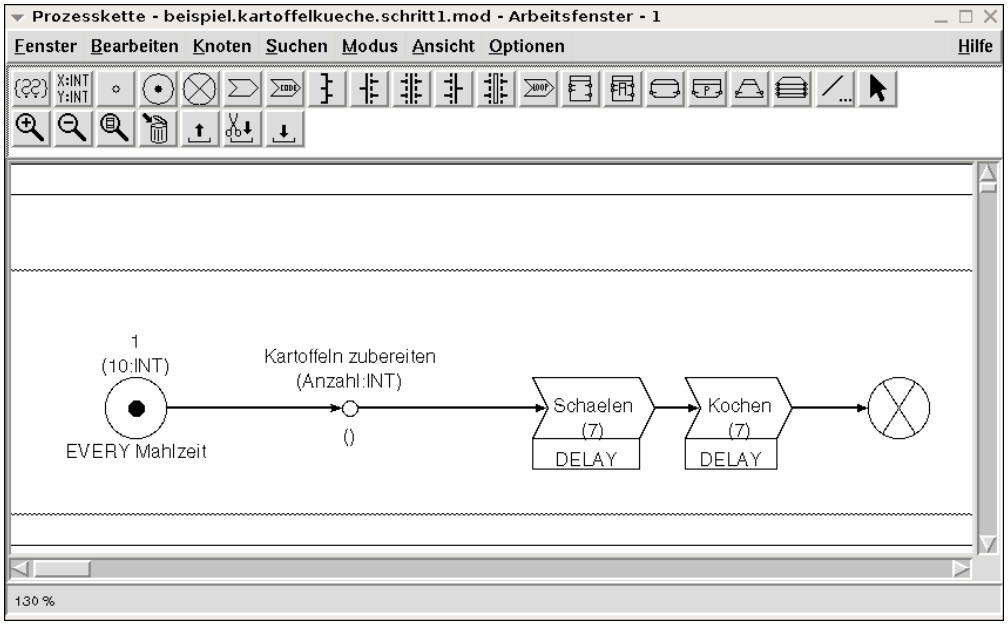


Abbildung 6.1: Kartoffelküche Schritt 1

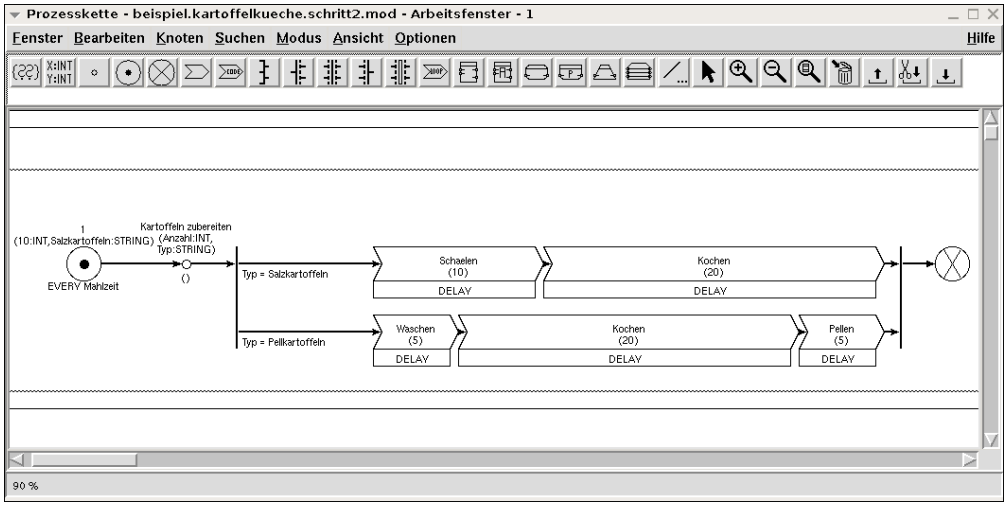


Abbildung 6.2: Kartoffelküche Schritt 2

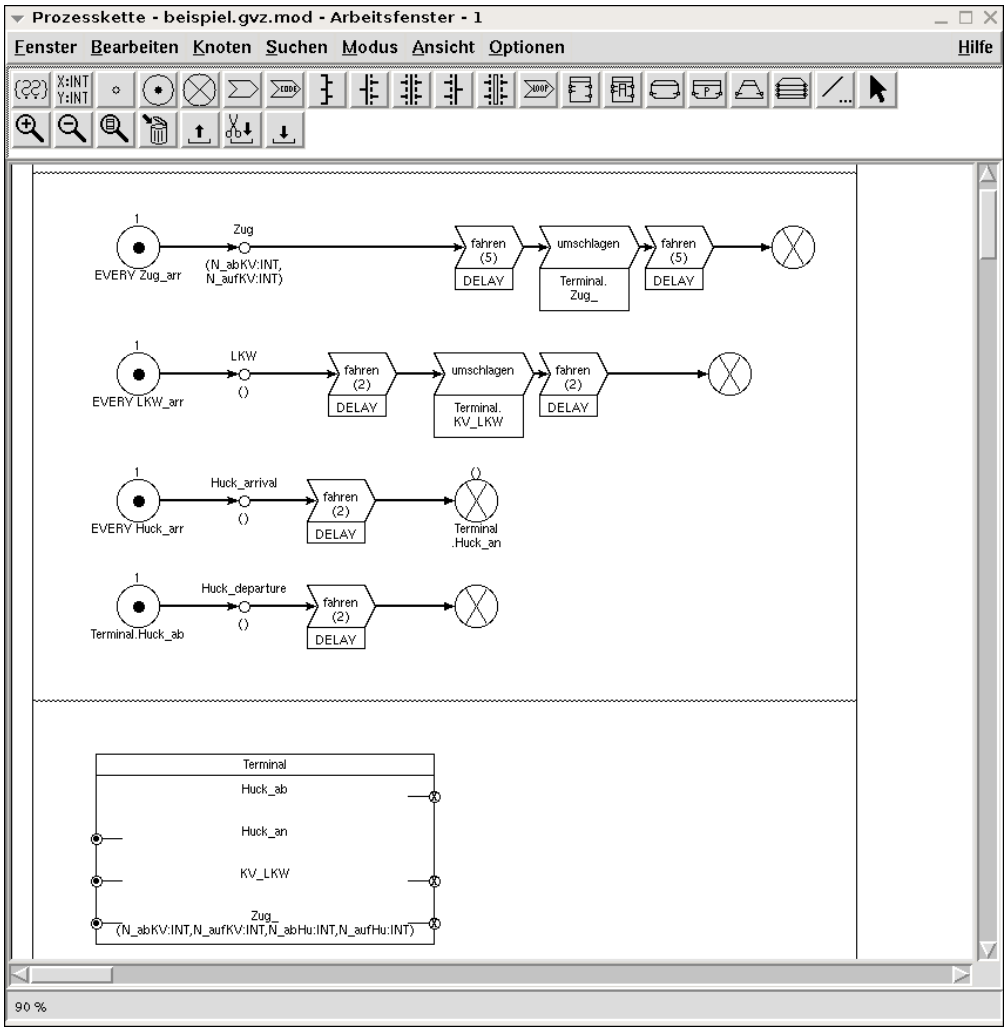


Abbildung 6.3: Güterverteilzentrum Aussenansicht

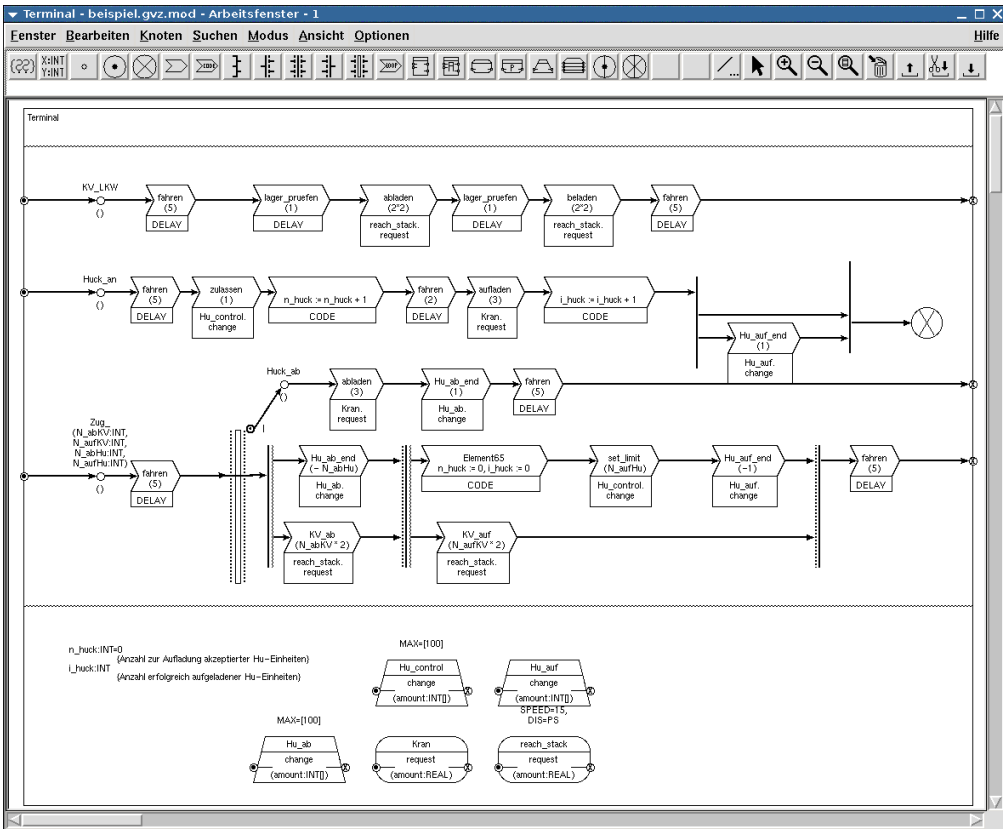


Abbildung 6.4: Güterverteilzentrum Innenansicht

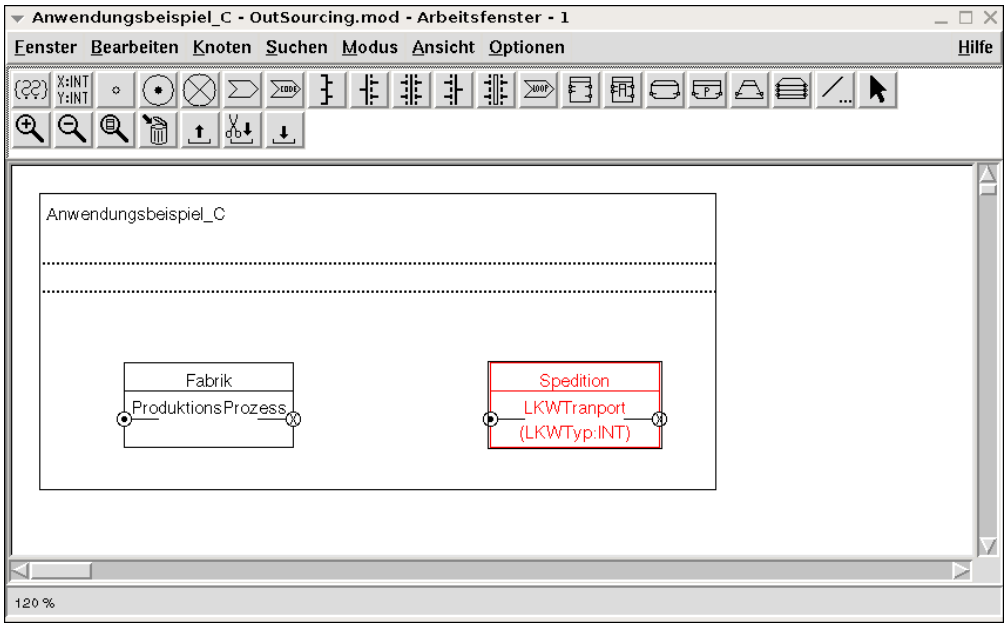


Abbildung 6.5: Spedition und Fabrik

6.3 Outsourcing

Das nächste Beispiel soll detailierter beschreiben, wie externe Funktionseinheiten verwendet werden können. Die zugehörige Modelldatei befindet sich in `Beispielmodelle/OutSourcing.mod`.
Die folgende Abbildung 6.5 zeigt zwei Funktionseinheiten Spedition und Fabrik.

Die Funktionseinheit Fabrik möchte nun den Dienst LKWTransport der Funktionseinheit Spedition nutzen. Um den Dienst LKWTransport in die Fabrik zu importieren, müssen Sie das Attribut Zuordnungsliste der Funktionseinheit Fabrik ergänzen. Die Ergänzungen müssen dann folgendermaßen aussehen (siehe folgende Abbildung 6.6).

Nach dem Sie den OK-Button bestätigt haben, ist der angegebene Dienst in der Zuordnungsliste importiert. Um nun die gleiche Aufrufsyntax zwischen dem Dienst OutsourcingLKW der externen Funktionseinheit und dem Dienst LKWTransport der Spedition herzustellen, müssen noch die virtuellen Parameter eingegeben bzw. angepasst werden (siehe Abbildung 6.7).

Es sollte innerhalb der Funktionseinheit Fabrik eine externe Funktionseinheit mit dem Dienst OutsourcingLKW und der spezifierten Aufrufsyntax zu sehen sein (siehe Abbildung 6.8).

Der Dienst der externen Funktionseinheit kann wie bei “normalen” Funktionseinheiten durch Aufrufprozesskettenelemente benutzt werden (siehe Abbildung 6.9).

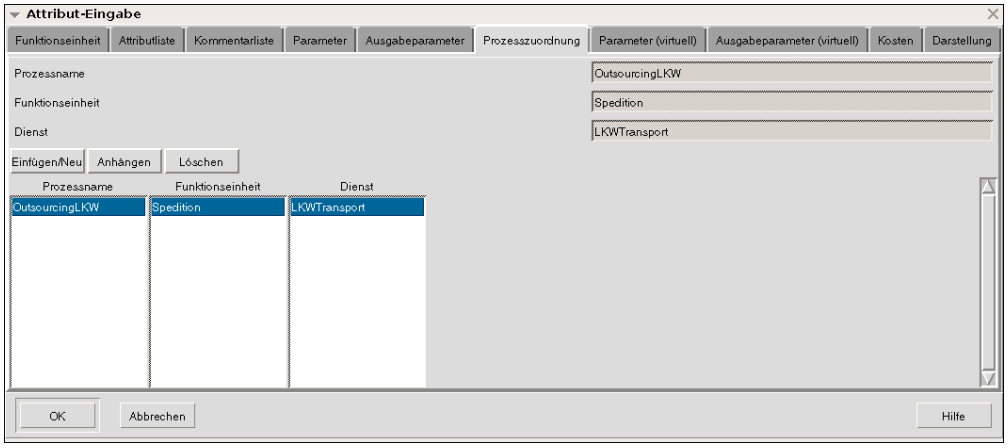


Abbildung 6.6: Zuordnungsliste

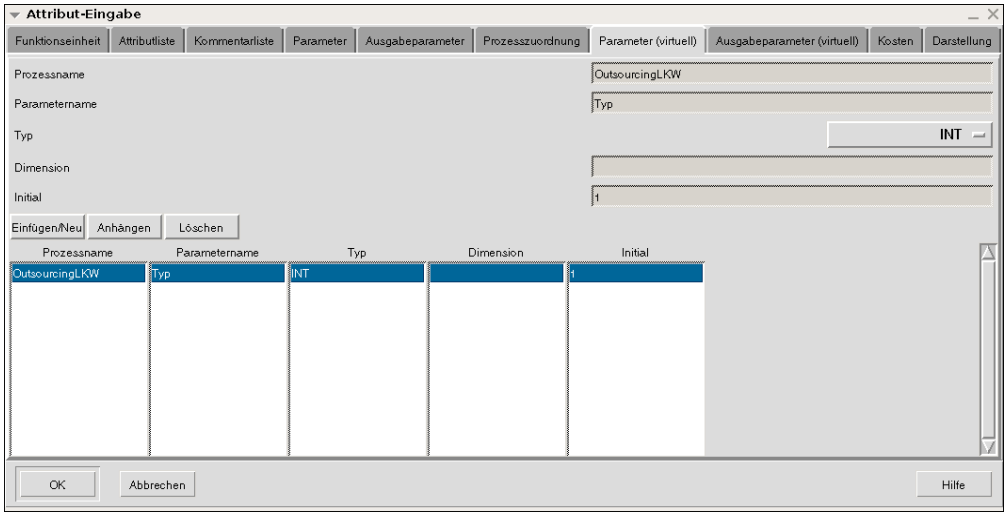


Abbildung 6.7: Virtuelle Parameterliste

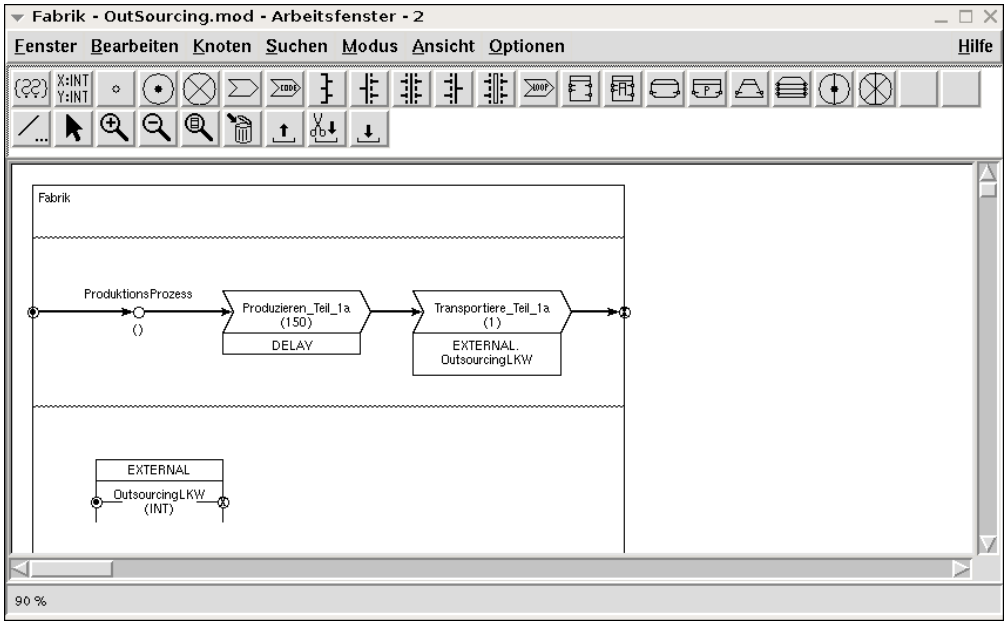


Abbildung 6.8: Fabrik mit externer Funktionseinheit

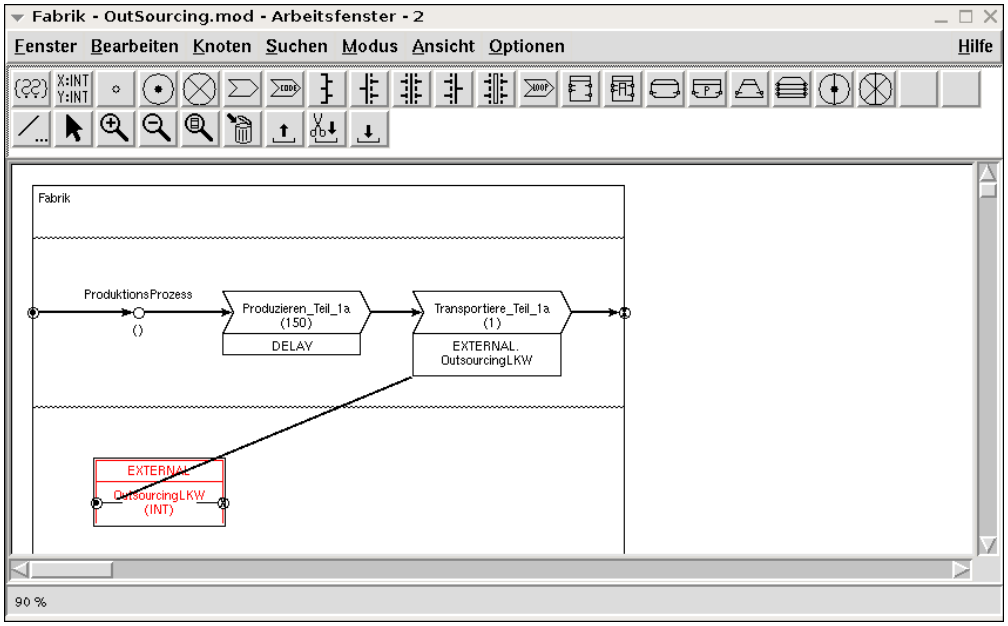


Abbildung 6.9: Anbindung an Dienst OutsourcingLKW

Kapitel 7

Typische Modellierungsfehler

Dieses Kapitel listet die häufigsten Fehlermeldungen auf, die beim Benutzen der Tools doggetoprocb, procbtohislang und HIT auftreten. Die ausgegebenen Fehlermeldungen sind unter Umständen nicht unmittelbar verständlich, deshalb werden die möglichen Ursachen zu jedem Fehler angegeben.

7.1 DoggeToProC/B-Fehlermeldungen

1. - Fehlermeldung:
   `Fehler beim Laden des Modells: Falsche Modellversion!`
   - Ursache: Die Modelldatei wurde mit einer älteren Editorversion erstellt und muss auf die zu DoggeToProC/B passende Version konvertiert werden.

2. - Fehlermeldung:
   `Ausgabeparameter: Zielvariable ... nicht vorhanden in ...`
   - Mögliche Ursachen:
     (a) Die lokale oder globale Variable, auf die zugegriffen werden soll, existiert nicht.
     (b) Der Präfix data. fehlt vor einer lokalen Variablen.
     (c) Die ProzessID hat einen syntaktisch nicht korrekten Bezeichner (z.B. enthält ein Bezeichner einige Sonderzeichen).

7.2 HIT-Fehlermeldungen

1. - HIT-Fehlermeldung:
   `Declaration for 'variable' missing or not within valid block.`
   - Ursache: `data.` vor <Variable> vergessen
     Lokale Variablen müssen im Gegensatz zu globalen Variablen mit den

Präfix `data.` versehen werden. Lokale Variablen werden bei der Definition von Prozessen deklariert (vgl. Abschnitt 4.3) und beispielsweise in Code-PKEs und als Übergabeparameter benutzt.

2.
   - HIT-Fehlermeldung:
     `Actual parameter is inconsistent with formal parameter 'amount'.`
   - Ursache: Übergabeparameter bei Countern und andere Arrays müssen bei der Eingabe mit eckigen Klammern eingeschlossen werden. Die Klammern werden bei einzelnen Einträgen oft übersehen.

3.
   - HIT-Fehlermeldung:
     `Declaration for 'data' missing or not within valid block. Illegal construction for expression on left side of CREATE statement`
   - Ursache: Prozesslokale Variablen werden an Kanten von Prozessskettenkonnektoren verwendet. Prozesslokale Variablen sind außerhalb von Prozessen, d.h. auch im Prozesskettenkonnektor unbekannt. Deswegen dürfen auf den Kanten keine lokalen Variablen als Vielfachheiten (Anzahl) stehen. Dies kann umgangen werden, indem stattdessen globale (d.h. FE-lokale) Variablen benutzt werden. Ggf. muss die lokale Variable per Code-PKE in eine globale umkopiert werden.

4.
   - HIT-Fehlermeldung:
     `The type of the PROB label is not REAL`
   - Ursache: Ein Boolscher Oder-Konnektor hat den falschen Typ (probabilistisch bei Bedingungen).

5.
   - HIT-Fehlermeldung:
     `Missing symbol: ')'. Last scanned symbol was: ','. Too many actual parameters for 'uniform'.`
   - Ursache: Ein Dezimalkomma wird statt eines Dezimalpunkts verwendet.

6.
   - HIT-Fehlermeldung:
     `Illegal symbol for PROCEDURE_CALL_OR_ASSIGNMENT:'='.`
   - Ursache: In einer Zuweisung wird statt := nur = verwendet.

7.
   - HIT-Fehlermeldung:
     `Illegal symbol for SIMPLE_EXPRESSION: 'TRUE'.`
   - Ursache: Dies liegt daran, dass HIT `TRUE` und `FALSE` nicht als einfachen Ausdruck (simple_expression) erkennt. Bei der Modellierung muss `TRUE` entweder geklammert oder in if-Abfragen ganz weggelassen werden.
     Statt `if variable = TRUE then ...` kann z.B. `if variable then ...` und statt `if variable = FALSE then ...` kann `if NOT variable then ...` geschrieben werden.

8.
   - HIT-Fehlermeldung:
     `Incompatible elements within equating.`

```
SERVICE/PROCEDURE call aufruf is not referred to a
provided SERVICE/PROCEDURE.
```

- Ursache: Parameter von Bedingten Senken werden als Ausgabeparameter verwendet.

Anhang A

Konsistenzprüfungen im ProC/B-Editor

Dieser Abschnitt über die Durchführung von Konsistenzprüfungen wurde aus [7] entnommen.

A.1 Durchführung von Konsistenzprüfungen

Die Konsistenzprüfungen für ProC/B-Modelle können sowohl aus dem Hauptfenster des ProC/B-Editors als auch aus den Arbeitsfenstern gestartet werden. Im Hauptfenster können die Tests aus dem Datei-Menü über den Menüpunkt „Konsistenzprüfung durchführen“ aufgerufen werden (s. Abbildung A.1). In einem Arbeitsfenster erfolgt der Aufruf über das Funktionseinheit-Menü (s. Abbildung A.2).

Im daraufhin erscheinenden Auswahlfenster (s. Abbildung A.3) kann festgelegt werden, welche Konsistenzprüfungen durchgeführt werden sollen. Das Fenster zeigt für jede der zur Auswahl stehenden Konsistenzprüfungen eine kurze Beschreibung an. Unterhalb der Beschreibung können zusätzliche Optionen für die Prüfung festgelegt werden.
Im Einzelnen stehen die folgenden Konsistenzprüfungen zur Auswahl:

- **Überprüfung der Syntax**: Bei der Syntaxüberprüfung wird die Struktur des Prozesskettenmodells auf Korrektheit getestet. Dabei werden z.B. die fehlerhafte Schachtelung von Konnektoren, unvollständige Prozessketten, fehlende Dienstanbindungen oder nicht genutzte Modellelemente entdeckt. Zusätzlich können auch die Attribute der einzelnen Modellelemente geprüft werden. Dieser Test entdeckt nicht-deklarierte Variablen, fehlerhafte Ausdrücke (z.B. als Kantenbeschriftungen oder Zeitangaben von Quellen) und unpassende Parameter bei Dienstaufrufen. Außerdem werden auch nicht genutzte Variablen gemeldet. Die Untersuchung der Attribute funktioniert allerdings nur bei Ausdrücken in Hi-Slang-Syntax.
  Die Überprüfung der Syntax steht sowohl bei Aufruf aus dem Hauptfenster als auch bei Aufruf aus einem Arbeitsfenster zur Verfügung. Im ersten Fall wird das gesamte Modell untersucht im zweiten Fall nur die Funktionseinheit, die in dem aktuellen Arbeitsfenster dargestellt wird, und eventuell darin enthaltene Funktionseinheiten.

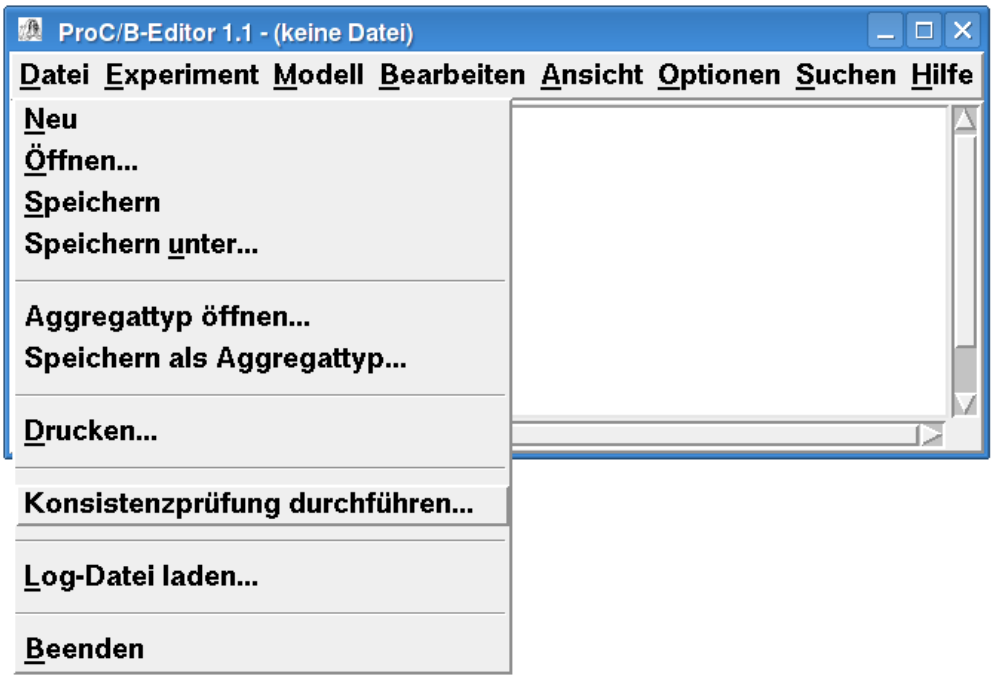


Abbildung A.1: Aufruf der Konsistenzprüfungen aus dem Hauptfenster

Die Arbeitsweise der Syntaxüberprüfung kann durch folgende Optionen genauer festgelegt werden:

- *Folgefehler ignorieren*: Falls diese Option ausgewählt ist, wird pro Prozesskette nur der erste gefundene Fehler in der Struktur gemeldet, da die weiteren Fehler häufig eine Folge dieses Fehlers sind.
- *nur Struktur untersuchen*: Über diese Option kann die Untersuchung der Attribute der Modellelemente abgestellt werden. Dies ist z.B. wünschenswert, wenn das ProC/B-Modell nicht nach Hi-Slang, sondern in die Eingabesprache einer anderen Simulationssoftware übersetzt werden soll.
- *enthaltene Funktionseinheiten untersuchen*: Mit dieser Option kann festgelegt werden, ob nur die aktuelle Funktionseinheit untersucht werden soll oder auch alle in ihr enthaltenen Funktionseinheiten. Die Option kann nur ausgewählt werden, wenn die Konsistenzprüfung aus einem Arbeitsfenster aufgerufen wurde. Bei einem Aufruf aus dem Hauptfenster wird immer das gesamte Modell untersucht.

- **Überprüfung auf Nicht-Stationarität**: Dieser Test wandelt das ProC/B-Modell in ein Petri-Netz um und prüft das erzeugte Netz auf E-Sensitivität. Da E-Sensitivität ein starker Hinweis auf Nicht-Ergodizität ist, sollten e-sensitive Modelle vor der Simulation einer genaueren Betrachtung unterzogen werden. Dieser Test wird für das gesamte Modell durchgeführt, falls der Aufruf der Konsistenzprüfung aus dem Hauptfenster des Editors erfolgte. Bei einem Aufruf aus einem Arbeitsfenster wird die in dem Fenster dargestellte Funktionseinheit untersucht.

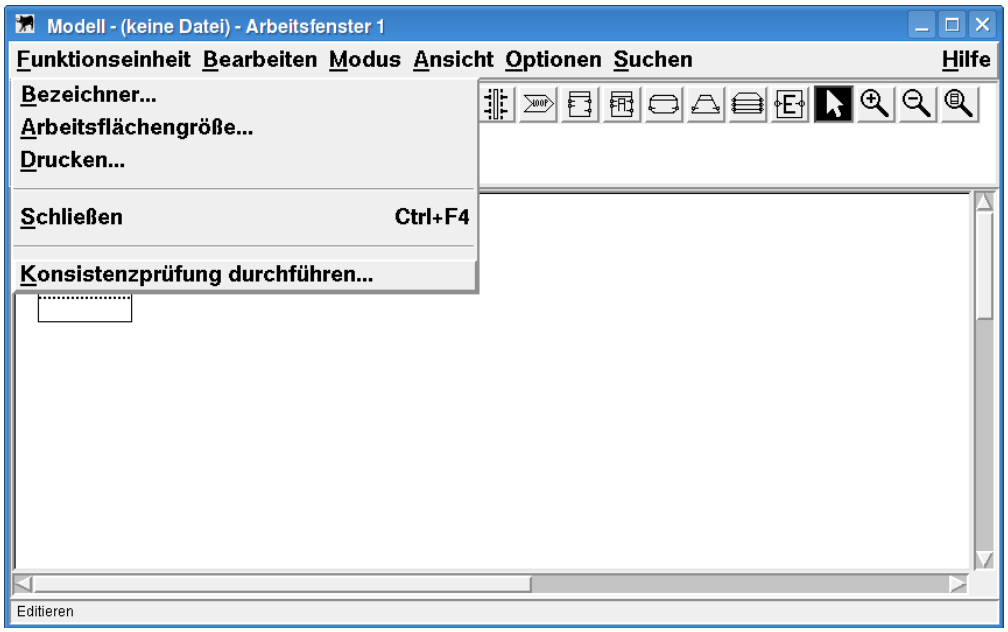


Abbildung A.2: Aufruf der Konsistenzprüfungen aus dem Arbeitsfenster

Über die Option *enthaltene Funktionseinheiten untersuchen* kann im zweiten Fall angegeben werden, dass auch alle in dieser Funktionseinheit enthaltenen weiteren Funktionseinheiten untersucht werden.

- **Umwandlung in das APNN-Format**: Hiermit wird das ProC/B-Modell in ein Petri-Netz umgewandelt und in dem Format für die APNN-Toolbox abgespeichert. Über den Button „Speichern unter...“ kann der Dateiname der APNN-Datei ausgewählt werden. Die Umwandlung in das APNN-Format kann über weitere Optionen beeinflusst werden:
  - *Senken mit Quellen kurzschliessen*: Diese Option kann ausgewählt werden, wenn der Zustandsraum des erzeugten Petri-Netzes beschränkt sein soll. In diesem Fall können bedingte Quellen vom Typ EVERY nur eine begrenzte Anzahl von Prozessen erzeugen. Die Erzeugung weiterer Prozesse ist erst dann möglich, wenn ein Prozess beendet wurde. Um dies sicherzustellen erhalten die Transitionen, die die Quelle repräsentieren, eine Stelle im Vorbereich. Damit das Netz lebendig bleibt, wird die Senke mit dieser Stelle verbunden.
  - *enthaltene Funktionseinheiten untersuchen*: Mit dieser Option kann festgelegt werden, ob nur die aktuelle Funktionseinheit umgewandelt werden soll oder auch alle in ihr enthaltenen Funktionseinheiten. Die Option kann nur ausgewählt werden, wenn die Konsistenzprüfung aus einem Arbeitsfenster aufgerufen wurde. Bei einem Aufruf aus dem Hauptfenster wird immer das gesamte Modell untersucht.

Durch einen Klick auf den OK-Button werden die ausgewählten Konsistenzprü-

fungen durchgeführt und anschließend die Ergebnisse im Ergebnisfenster (s. Abbildung A.4) dargestellt. Das Ergebnisfenster besteht aus drei Teilen: Einer Auswahlliste im linken oberen Bereich, einigen Buttons zur Sortierung und Filterung der Ergebnisse im linken unteren Bereich und dem rechten Fenster, in dem die Resultate angezeigt werden.

Die Ergebnisse können nach Konsistenzprüfungen oder Funktionseinheiten sortiert werden. Bei einer Sortierung nach Konsistenzprüfungen enthält die Auswahlliste alle durchgeführten Tests (Abbildung A.4). Im Resultatfenster werden dann alle Ergebnisse dieser Prüfung angezeigt. Dabei werden zur besseren Übersicht alle Meldungen, die zu derselben Funktionseinheit gehören, zusammengefasst. Bei einer Sortierung nach Funktionseinheiten (Abbildung A.5) enthält die Auswahlliste alle Funktionseinheiten, für die Ergebnisse vorliegen, und im Resultatfenster werden alle Ergebnisse für diese Funktionseinheit angezeigt.

Über die Anzeige- und Filter-Optionen können zusätzlich einige der Meldungen ein- bzw. ausgeblendet werden. So ist es beispielweise möglich, sich nur Fehler oder nur Warnungen anzeigen zu lassen. Zusätzlich kann man sich auch nur Meldungen anzeigen lassen, die z.B. die Struktur des Modells oder die Attribute einzelner Elemente betreffen.

Wenn alle Einstellungen für die Sortierung und Anzeige der Ergebnisse vorgenommen worden sind, kann das Ergebnisfenster über den Button „Anzeige aktualisieren“ neu geladen werden.

Wie bereits erwähnt, erhält der rechte Bereich des Fensters jeweils eine Auswahl der Ergebnisse der Konsistenzprüfungen. Jedes Ergebnis besteht dabei aus einem Typ (z.B. Fehler), der farblich hervorgehoben wird, und einer Beschreibung des Fehlers, die auch die betroffenen Modellelemente enthält. Die Modellelemente sind in der Beschreibung blau hervorgehoben. Bei einem Klick mit der linken Maustaste auf den Namen wird ein Arbeitsfenster mit der Funktionseinheit, zu der das Element gehört, geöffnet und das Element markiert und zentriert im Arbeitsfenster dargestellt. Ein Klick mit der rechten Maustaste auf den Namen des Elements öffnet ein Kontextmenü (s. Abbildung A.5). Hiermit kann das Element ebenfalls wie bei einem Linksklick angezeigt werden. Außerdem kann aber auch direkt das Attributfenster des jeweiligen Modellelements geöffnet werden, um schnell Fehler in den Attributen beseitigen zu können.

Die Darstellung der Fehlermeldungen orientiert sich also sowohl optisch als auch von der Funktionsweise her stark an Links in HTML-Seiten und sollte deshalb intuitiv zu benutzen sein.

A.1.1 Vorbereitung der Konsistenzprüfungen

Damit die Überprüfung auf Nicht-Stationarität sinnvoll durchgeführt werden kann, ist es notwendig, dass der Modellierer das Modell entsprechend vorbereitet. Für die Überprüfung wird das Modell in ein Petri-Netz umgewandelt. Da Variablen sich nur mit großen Einschränkungen in einem Petri-Netz darstellen lassen, werden Variablen und auch Aufrufparameter von Diensten bei der Umwandlung ignoriert. Da eine häufige Ursache für Nicht-Stationarität der Zugriff mehrerer Prozessketten auf dasselbe Lager ist, kann der Modellierer durch die Eingabe zusätzlicher Attribute dafür sorgen, dass diese Situationen bei der Umwandlung korrekt behandelt werden. Jedes Prozessketteneelement hat ein zusätzliches Attribut (siehe Abbildung A.6), mit dem der Nutzer

den Zugriff auf das Lager passend angeben kann. Negative Zahlen für das Attribut bedeuten, dass eine Auslagerung stattfindet, positive Zahlen stehen für eine Einlagerung. Durch den Wert der Zahl kann der Zugriff außerdem gewichtet werden. Ist bei einem Aufruf-PKE der Wert 1 eingetragen und bei einem zweiten der Wert 3, so bedeutet dies, dass der Modellierer erwartet, dass bei einer Nutzung des Lagers durch das zweite Aufruf-PKE durchschnittlich dreimal so viel eingelagert wird wie bei einer Nutzung durch das erste Aufruf-PKE.

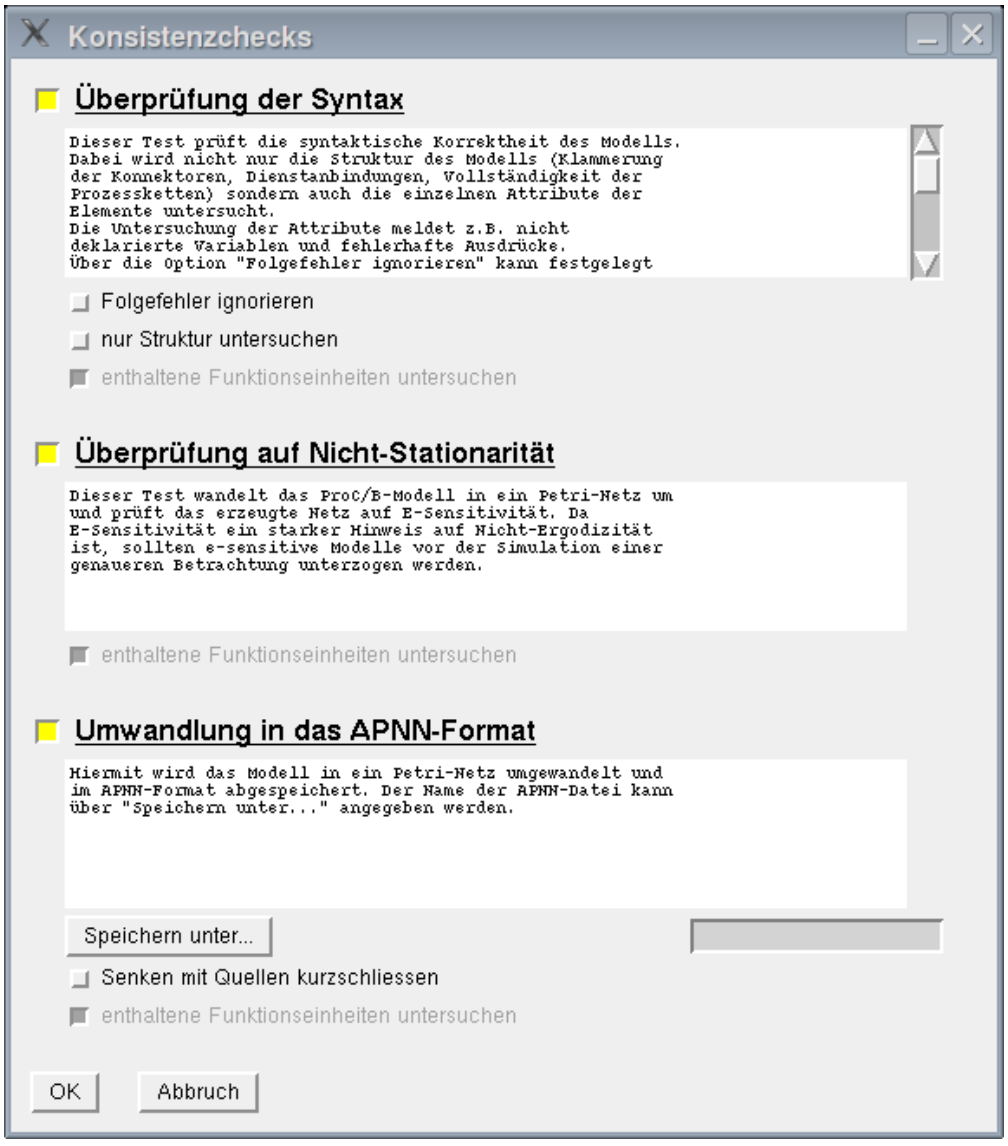


Abbildung A.3: Auswahlfenster für Konsistenzprüfungen

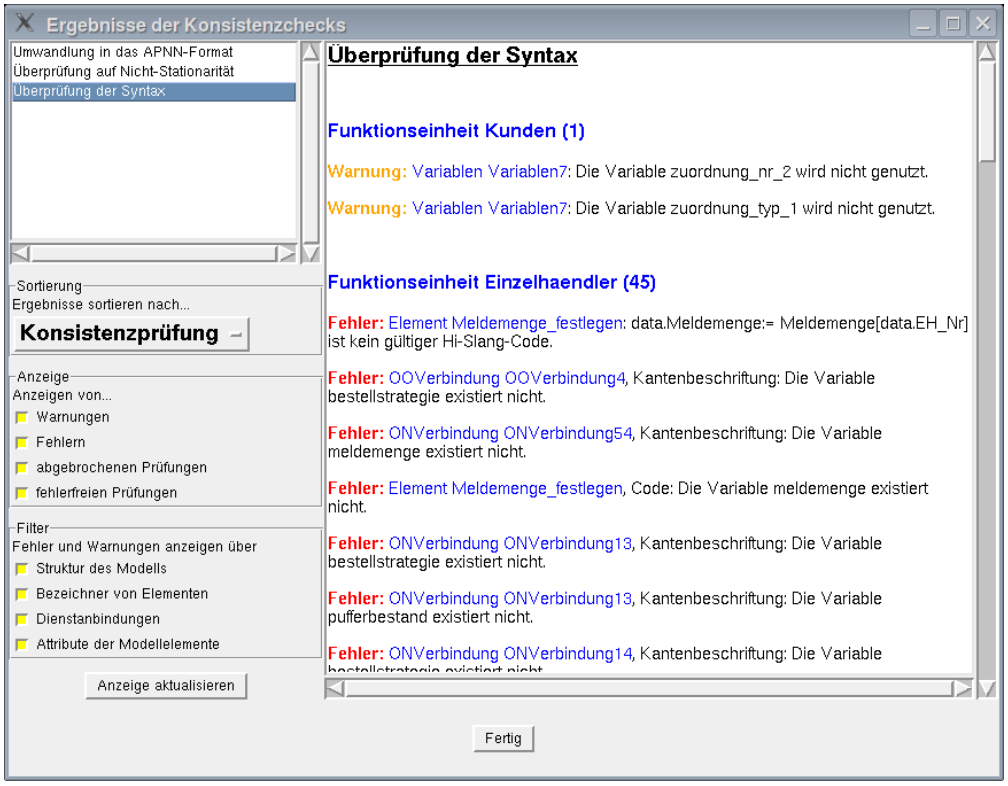


Abbildung A.4: Ergebnisdarstellung der Konsistenzprüfungen (nach Eigenschaft sortiert)

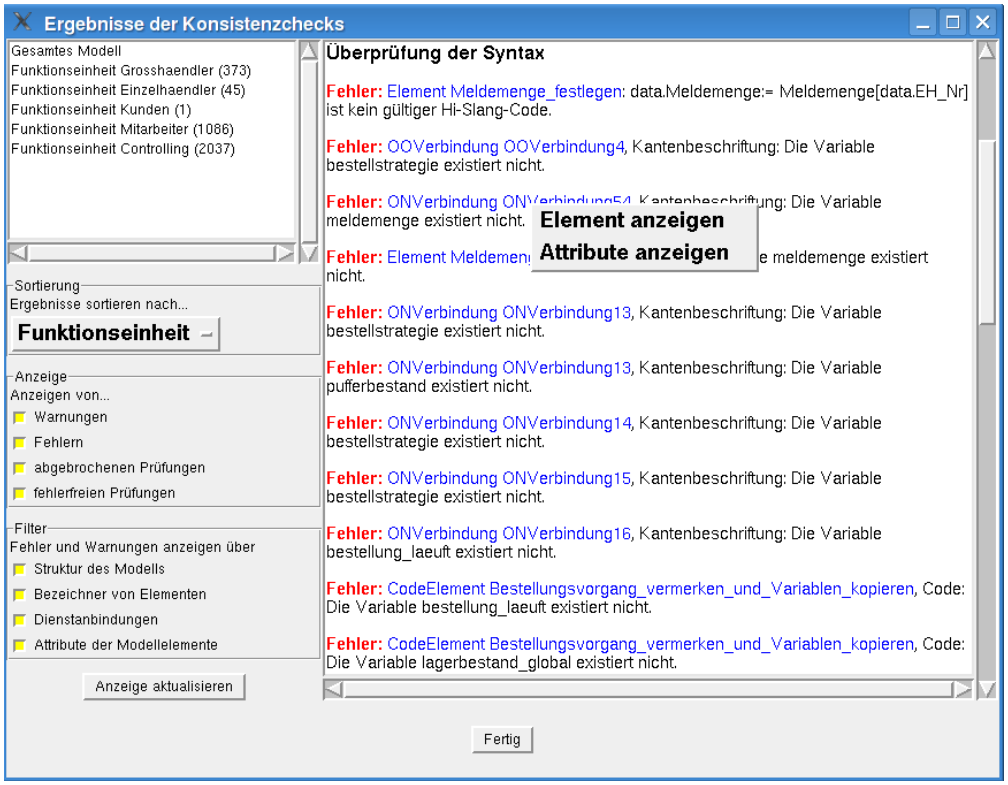


Abbildung A.5: Ergebnisdarstellung der Konsistenzprüfungen (nach Funktionseinheiten sortiert)

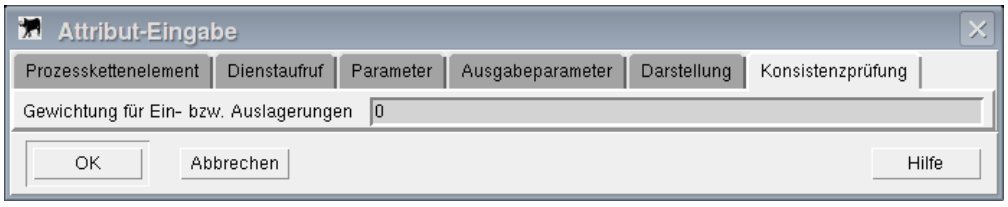


Abbildung A.6: Attributfenster eines Aufruf-PKEs

Anhang B

Auszüge aus dem Hi-Slang Reference Manual

Es folgen einige Auszüge aus dem Hi-Slang Reference Manual, die für die Modellierung im ProC/B-Editor hilfreich sein können. Die Nummerierung der Abschnitte entspricht der Nummerierung aus dem Reference Manual.
In Abschnitt 3.2 wird erläutert, welche Arten von Ausdrücken in HiSlang erlaubt sind. Bei der Modellierung können diese Ausdrücke zum Beispiel verwendet werden, um Variablen einen Wert zuzuweisen oder als Parameter bei Funktionsaufrufen.
Abschnitt 3.3 behandelt Variablenzuweisungen und if-then-else-Bedingungen in Hi-Slang, die sich zum Beispiel in Code-Elementen nutzen lassen.
Abschnitt 5.1 behandelt Streams, die im Editor `Rewards` genannt werden. Rewards lassen sich im Editor als globale Variablen selbst anlegen. An den selbstdefinierten Rewards und einigen Standardmessströmen können im Rahmen der Experimentserie Messungen durchgeführt werden. Der Abschnitt gibt einen Überblick über die verschiedenen Reward-Typen und die Standardmessströme.
Abschnitt D.1 erläutert schliesslich, in welcher Reihenfolge Hi-Slang-Operatoren in Ausdrücken ausgewertet werden. In den restlichen Abschnitten werden arithmetische Funktionen aufgezählt, die Hi-Slang zur Verfügung stellt. Diese Funktionen können im Editor zum Beispiel in Ausdrücken verwendet werden, um eine Variable mit einem Zufallswert zu belegen.

3.2. Expressions

An expression is a formula that describes the computation of a value. The value has a (standard) data type which depends on the types of the operands and the operators occuring in the expression. We do not regard expressions and operations on records and pointers here (they are treated in Section 3.7.4.). The OF operator within primary below will be introduced in Section 4.2.2.3. All other operators are known from other programming languages.

3.2.1. Arithmetic Expressions

An arithmetic expression results in an INTEGER or a REAL value. The nonterminal simple_real_expression denotes arithmetic expressions in the following syntax:

```
   simple_real_expression ::= ...
      | [unary_operator] term [adding_operator...]

   adding_operator ::=
      + | -

   term ::=
      factor [multiplying_operator ...]

   multiplying_operator ::=
      * | / | // | MOD

   factor ::=
      primary [** ...]

   primary ::=
      identifier [OF identifier]
      | number
      | (simple_real_expression)

   identifier ::=
      {name [[simple_real_expression [, ...]]] [actual_parameters]} [. ...]


Examples:

   2.0
   a        {a is REAL or INTEGER variable}
   (b [3])  {b is REAL or INTEGER array}
   f (3)    {f is INTEGER procedure}
```

Numbers, variables and constants, calls of procedures and services returning one result value, accesses to elements of structured objects (arrays, records) and INTEGER or REAL expressions in parentheses can appear as operands. There are two unary operators for arithmetic expressions: ’+’ and ’-’:

unary operator	type of operand	type of result
+	INTEGER	INTEGER
	REAL	REAL
-	INTEGER	INTEGER
	REAL	REAL

```
Examples:

   -a
   +2.0                  {equivalent to 2.0}
   -(a+b)
   -(5 * matrix [6, 1])
   -rank (’a’)           {rank is a standard procedure}
```

Binary operators in arithmetic expressions may be addition operators (’+’, ’-’) and multiplication operators (’*’, ’/’, ’//’, ’**’, MOD). Arithmetic expressions can appear as their operands. The following tables show the admissible operators and the resulting types of the expressions, which depend on the used operand types:

Addition:				Subtraction:		
+	INTEGER	REAL		-	INTEGER	REAL
INTEGER	INTEGER	REAL		INTEGER	INTEGER	REAL
REAL	REAL	REAL		REAL	REAL	REAL

Multiplication:				Division:		
*	INTEGER	REAL		/	INTEGER	REAL
INTEGER	INTEGER	REAL		INTEGER	REAL	REAL
REAL	REAL	REAL		REAL	REAL	REAL

Integer division:				Modulo:		
//	INTEGER	REAL		MOD	INTEGER	REAL
INTEGER	INTEGER	INTEGER		INTEGER	INTEGER	INTEGER
REAL	INTEGER	INTEGER		REAL	INTEGER	INTEGER

Exponentiation:		
**	INTEGER	REAL
INTEGER	REAL	REAL
REAL	REAL	REAL

Expressions, when used as operands, need only be parenthesized if the desired sequence of computation is not achieved by precedence rules. Parentheses are permitted for means of clearness, too. Expressions with operators of the same precedence will be evaluated from left to right.

For normal division, integer division and the modulo operation one has to observe that the operand on the right hand side may not be equal to 0. Integer division and modulo demand operands of type INTEGER. Here REAL values are rounded and converted to INTEGER values.

Examples:

```
-(a+b)
a*c+b           {will be evaluated just like (a*c)+b}

e**f*g          {will be evaluated just like (e**f)*g}
e**f**g         {will be evaluated just like (e**f)**g}

-3+4+5-1        {yields the same result as}
(((-3)+4)+5)-1

3.4 MOD 1.5     {will be evaluated like ...}
3 MOD 2         {and yields 1}

2.5 MOD 2.4     {will be evaluated like ...}
3 MOD 2         {and yields 1}
```

Exponentiation with negative values in both arguments is not checked by the HI-SLANG compiler, but depends on the run time system of the SIMULA compiler.

3.2.2. CHARACTER Expressions

CHARACTER expressions yield a value of type CHARACTER. Operands of them may be CHARACTER constants or CHARACTER variables, CHARACTER elements of structured objects, the standard procedure char, or calls of procedures and services which result in exactly one CHARACTER value. The operands of a CHARACTER expression may be parenthesized. Of course INTEGER expressions may occur within CHARACTER array indices and expressions of the corresponding type may serve as actual parameters of CHARACTER procedures (see syntax of identifier).

```
simple_expression ::= ...
   | character
   | identifier
   | (expression)


Examples:

   'a'
   char (10)           {the standard procedure char yields the 10th}
                       {ASCII or EBCDIC character in this case}

   (char_variable)

   char (rank (f (x))  {rank is the invers of char;
                        (f is a CHARACTER function)}

   char_array [x+y]
```

3.2.3. TEXT Expressions

TEXT expressions yield a value of type TEXT. Operands of them may be string constants (with a maximal length of 70 characters) or string variables, TEXT elements of structured objects, and calls of procedures and services which result in exactly one TEXT value. The operands of a TEXT expression may be parenthesized. The only operator in TEXT expression is the binary concatenation operator '&'.

```
   simple_text_expression ::=
      simple_text [& ...]

   simple_text ::=
      string
      | identifier
      | (simple_text_expression)


Examples:

   "texts"
   "tex" & "ts"          {= "texts"}
   ("texts")
   (("tex") & ("ts"))
```

```
days [1] & "DAY"       {= "SUNDAY";
                        see also examples in Section 3.1.2.}
"H" & "A" & "L" & "T"  {= "HALT"}
```

3.2.4. BOOLEAN Expressions

Boolean expressions result in TRUE or FALSE. They may contain operator NOT or binary operators, especially relational operators:

```
boolean_expression  ::=
   TRUE
   | FALSE
   | disjunction [EQV disjunction]

disjunction  ::=
   conjunction [or_else ...]

or_else  ::=
   OR [ELSE]

conjunction  ::=
   {[NOT] relation } [and_then ...]

and_then  ::=
   AND [THEN]

relation  ::=
   simple_expression [relational_operator simple_expression]

relational_operator  ::=
   =  |  <  |  >  |  <=  |  >=  |  <>  |  #

simple_expression  ::= ...
   | identifier
   | ( boolean_expression )
```

Operands of boolean expressions are constants or variables of type BOOLEAN, boolean elements of structured objects, relations and calls of procedures and services that yield exactly one boolean value. The keywords TRUE, FALSE are also allowed as boolean constants.

Examples:

```
VARIABLE bit : BOOLEAN;
```

```
bit_vector   : ARRAY [1..8] OF BOOLEAN;

TRUE
bit
bit_vector [1]
```

The only applicable unary operator in boolean expressions is the NOT operator. It negates the value of a boolean operand, which must be a boolean expression that may be parenthesized.

NOT	TRUE	FALSE
	FALSE	TRUE

Examples:

```
NOT bit_vector [1]
NOT (a=b)
NOT (TRUE)
```

Note: The keywords TRUE and FALSE, when used as operands must be parenthesized.

Binary operands in boolean expressions may either be relational operators (’=’, ’<’, ’>’, ’<=’, ’>=’, ’<>’, and ’#’) or logical operators (’EQV’, ’AND’, ’OR’). Logical operators allow only boolean expressions as operands . Relational operators allow arithmetic expressions, CHARACTER or TEXT expressions; ’=’, ’<>’ and ’#’ also allow boolean expressions. Terms consisting of relational operators and their operands are boolean expressions.

EQV / =	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	TRUE

AND	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

<>, #	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	FALSE

OR	TRUE	FALSE
TRUE	TRUE	TRUE
FALSE	TRUE	FALSE

The operators EQV or '=' compare two boolean expressions for equivalence and yield either TRUE or FALSE. The operators '<>' and '#' (which may be read as XOR) compare two boolean expressions for inequality and yield either TRUE or FALSE. The AND operator delivers TRUE, if all operands have the value TRUE. The OR operator delivers TRUE, if at least one operand has the value TRUE. All operands of a boolean expression are evaluated. The sequence of evaluation depends on the HIT installation. This may cause side-effects.

To avoid these side-effects the operators AND THEN and OR ELSE are provided. Their semantics correspond to those of AND and OR, respectively, but only the first operand is evaluated if it defines the result of the expression clearly. Using AND THEN leads to the next operand not being evaluated unless the previous one results in TRUE. OR ELSE leads to the next operand not being evaluated unless the previous one results in FALSE. Boolean expressions with the operators AND THEN and OR ELSE are called conditional boolean expressions.

Examples:

```
VARIABLE  flag1, flag2 : BOOLEAN;

flag1 EQV flag2
      {= TRUE, because flag1 and flag2 are initialized by FALSE}

flag1 AND (TRUE)               {= FALSE}
flag1 OR flag2 OR (TRUE)       {= TRUE}
(flag1 AND flag2) OR (TRUE)    {= TRUE}

flag1 AND THEN flag2
      {= FALSE; 2. operand will not be evaluated}

NOT flag1 OR ELSE NOT flag2
      {= TRUE; 2.operand will not be evaluated}
```

Comparing two operands requires them to be of the same type, except for the comparison of INTEGER and REAL numbers. In this case, the INTEGER number

is converted to REAL and the comparison deals with REAL values. The result of a comparison always is of type BOOLEAN.

Operator	Meaning	
=	TRUE	both operands are equal
	FALSE	else
<	TRUE	the left operand is less than the right one
	FALSE	else
>	TRUE	the left operand is greater than the right one
	FALSE	else
<=	TRUE	the left operand is less than or equal to the right one
	FALSE	else
>=	TRUE	the left operand is greater than or equal to the right one
	FALSE	else
<>, #	TRUE	both operands are not equal
	FALSE	else

Two real expressions should not be compared using '=' or '<>' (resp. '#') (because of possible inaccuracies in REAL representations).

The comparison of TEXT expressions depends on the internal character coding. Two text elements are equal if they coincide in their length and in all of their characters. Text comparison means that all characters of the text are compared, regarding their respective positions. Text elements are compared from left to right. The first different character determines the decision which of the text strings is smaller or greater.

The text with the smaller character with regard to character coding, or the text with less characters, is the smaller text.

Examples:

```
"TEXTS" > "TEXT"
"text" <> "TEXT"
"texts" > "texti"
"abc" <> "abc "                {note the blank}
"Attention" < "C"
"attention" < "C"              {the EBCDIC case}
"attention" > "C"              {the ASCII case}
"T"&"E"&"X"&"T"&"S" = "TEXTS"
"A" > ""
```

All of the above examples result TRUE. CHARACTER expressions can also be compared with each other based on the internal coding:

```
Examples:

   CONSTANT a, c : INTEGER DEFAULT 2;
   CONSTANT b    : BOOLEAN DEFAULT TRUE;

   'a' = 'b'                    {= FALSE}
   a < c                        {= FALSE}
   a+1 >= c+1                   {= TRUE}
   a > c AND b                  {= FALSE}
   b OR (c+3 < 5)               {= TRUE}
   "text"&"s" <> "text"         {= TRUE}
   "text" < "texts" AND b       {= TRUE}


Examples:

   VARIABLE i : INTEGER DEFAULT 2;
            j : INTEGER DEFAULT 4;
            k : REAL    DEFAULT 0.5;

   j <> 0 AND THEN i/j <> k   {= FALSE; 2. operand will be evaluated}
```

Just as in arithmetic expressions, operands in boolean expressions can be parenthesized if the desired sequence of evaluation differs from that specified by precedence rules. Parentheses for the purpose of clearness are allowed. Boolean expressions with operators of the same precedence are evaluated from left to right.

3.2.5. Precedence Rules for Evaluating Expressions

Precedence rules determine the process of computing an expression. The syntax of HI-SLANG determines the following priorities for the operators of HI-SLANG:

(1) = lowest priority EQV
(2) OR, OR ELSE
(3) AND, AND THEN
(4) NOT
(5) =, <, >, <=, >=, <>, #
(6) +, -, &
(7) *, /, //, MOD
(8) **
(9) = highest priority (expression), OF

Operators of higher priority are applied prior to operators of lower priority. Ope-

rators of the same precedence are applied from left to right. Expressions embraced in parentheses are of highest priority. Expressions within parentheses are also computed according to the precedence order described above. Using parentheses, the programmer is able to define the desired order of subexpression evaluation.

Examples:

```
VARIABLE a, b, c: INTEGER DEFAULT 3;

a+b*c        {yields the value 12}
(a+b)*c      {yields the value 18}
a*b*c        {yields the value 27}
(a*b)**c     {yields the value 729}
```

3.3. Statements

This chapter describes the application of assignment statements, conditional statements, loop statements and block statements. Other statements, such as procedure calls, result assignments, input and output statements etc., will be described in following chapters.

```
sequence_of_statements ::=
   statement [ ...]

statement ::= ...
   | simple_statement
   | compound_statement

simple_statement ::= ...
   | assignment_statement
   | empty_statement

compound_statement ::= ...
   | conditional_statement
   | loop_statement
   | block_statement

empty_statement ::=
   ;
```

The execution of an empty statement entails no action. The statements in a sequence of statements are executed one after the other.

3.3.1. Assignment and Type Conversion

An assignment replaces the actual value of a variable or of an element of a structured object by a new value described by an expression or an aggregate. By using a multiple assignment, one value can be assigned to several variables or elements of structured objects simultaneously. Assignments involving records or pointers are not treated in this section. They are dealt with in Section 3.7.4.

The variables or elements of a structured object on the left hand side of an assignment and the expression or aggregates on its right hand side must be of the same type or, otherwise they must be convertible.

Assignments made to elements of a structured object require the object to be defined as a variable. Assignments to array variables demand consideration of the array´s dimension and its index bounds.

```
assignment_statement ::= ...
   | common_assignment

common_assignment ::=
   identifier [, ...] := expression_or_aggregate ;
```

An assignment will be executed by computing the expression on the right hand side first. Next, the computed value is given to the variable, unless the variable denotation contains expressions like array indices or procedure calls. These expressions determining the indices of the array element are calculated first (i.e., before computing the right hand side expression and performing the assignment). Of course this calculation has to yield values within the array bounds.

Multiple assignment is accomplished in the following manner: the first variable on the left side of the assignment operator ':=' acquires the value of the right hand side. Now, from right to left, the variables on the left hand side are successively assigned the new value, the latest updated variable serving each time as the right hand side of the next assignment. The same is valid for assigning a value to array elements.

Type conversion is allowed for INTEGER and REAL variables only. The conversion from INTEGER to REAL will not change the interpretation of the value as real number. The conversion from REAL to INTEGER occurs by rounding the REAL value to an INTEGER value and requires the REAL value to be representable as INTEGER.

Note: If there are array variables or expressions in the list on the left and right hand side of a multiple assignment, then all indices and expressions will be computed first (left to right) before any assignment is performed (as seen in the examples).

```
Examples:

   VARIABLE i       : INTEGER;
            x, y    : REAL;
            a, b, c : ARRAY [1..6] OF INTEGER;

   x, i, y := 1.2;    {corresponds to:}

   y := 1.2;
   i := y;            {i = 1,   conversion REAL - INTEGER}
   x := i;            {x = 1.0, conversion INTEGER - REAL}

   a[i], i := a[i] + 4;  {corresponds to:}

   i := a[i] + 4;        {i = 4, since a[1] = 0}
   a[1] := i;            {a[1] = 4, since index is evaluated first}

   c := [1, 2, 3, 4, 5, 6];
   a := 0;               {all array elements of a are set to 0}
   a, b := c;
```

3.3.2. Conditional Statements

Conditional statements offer the possibility of controlling the execution of the program with regard to one or more conditions.

```
   conditional_statement ::=
      if_statement
      | case_statement
      | branch_statement
```

3.3.2.1. IF Statement

In an IF statement, either one sequence of statements or no statements at all are selected for execution. The selection depends on the value of a boolean expression.

```
   if_statement ::=
      IF boolean_expression
      THEN sequence_of_statements
      [ELSE sequence_of_statements]
      END IF ;
```

An IF statement is executed by computing the boolean expression following the keyword IF first. If the evaluation of the expression results in TRUE, the statements of the THEN branch are carried out. If the result is FALSE and an ELSE branch exists, the statements of the ELSE branch are executed. The IF statement is terminated by END IF. Both the THEN branch and the ELSE branch may in turn contain IF statements themselves. The IF-END IF pairs clearly define the nesting structure.

Note: Since every statement is terminated by a semicolon, there is always a semicolon before ELSE or END IF.

Examples:

```
VARIABLE a, b, c : INTEGER;

IF a < b THEN                    {1}
   a := a + 1;
   IF a = b THEN                 {2}
      a := 0;
   ELSE                          {2}
      IF a < b AND c > 0 THEN    {3}
         c := 0;
      END IF;                    {3}
      c := c + 1;
   END IF;                       {2}
ELSE                             {1}
   a := a - 1;
END IF;                          {1}
```

5.1. Streams

In HIT, so-called streams form the most important interface between a model and its evaluation. From the analytical point of view a stream represents a kind of stochastic process described by the model. From the statistical point of view a stream is a multivariable time series, from the simulative point of view it is a list of pairs, generated during the simulation. Each pair (ti, xi) consists of a time stamp (or a counter) ti and a numerical value xi. The interpretation of xi (and therefore its evaluation) depends on the type of the stream. There are three stream types, COUNT, EVENT and STATE, which are described below. For any solver a standard set of streams for analyzing the model is available (see Section 5.1.2.). By the simulative solver additionally user-defined streams can be measured. In contrast to standard streams, which can simply be evaluated by MEASURE statements, for user-defined streams the user additionally has to

- declare a stream of the appropriate type (Section 5.1.1.), and to

- update the stream, i.e., generate a new pair (UPDATE statement; Section 5.1.3.)

in his model description.

5.1.1. Declaration of Streams

Streams may be declared in the declaration part of component and model types. Such streams are called user-defined streams in contrast to predefined standard streams, which may not be declared. The declaration of user-defined streams is only allowed when using simulation.

```
modelling_declaration  ::= ...
   | stream_declaration

stream_declaration  ::=
   STREAM
   {stream-name [, ...] : stream_type ; } [ ...]

stream_type  ::=
   COUNT
   | EVENT
   | STATE
```

The stream-name is the name of the stream (any identifier allowed by the rules for constructing names and according to the scope of validity for identifiers), by which this stream can be accessed in the body of all services.

Examples:

```
STREAM creek  : STATE;
       s1, s2 : COUNT;

STREAM cycletime : EVENT;
```

Note: Due to historical reasons streams may as well be declared in services, but for such streams a COLLECT block has additionally to be provided in the surrounding component type. Declaring a stream in a service will produce a warning.

5.1.1.1. Type EVENT

The type EVENT covers event streams for serially collecting values. The generated pairs (either explicitly by an UPDATE statement or implicitly for standard streams)

consist of a number, which is interpreted as an observed value and a consecutive counter starting at 1. The numerical value must be a REAL. The counter value is generated by the system. The estimator MEAN for streams of type EVENT is defined by

$$x = \frac{1}{N} \cdot \sum_{i=1}^{N} x_i$$

where N is the number of pairs and x_i (i = 1, ..., N) the numerical value. Thus the estimator MEAN is the mean of the observed values, if no start condition is specified. An estimator for the standard deviation is computed with respect to this mean value, an estimator for the confidence interval is computed with respect to this mean estimator. If %parm=minmax is used within the control file (see Chapter 8), the minimum and maximum values shown in the table file are the smallest and largest ones of the observed values.

Example:

```
STREAM MY_TURNAROUNDTIME : EVENT;
```

5.1.1.2. Type STATE

The type STATE covers streams of states which may take different values but are constant while no update occurs. Continuously changing states cannot be described. The pairs generated (either explicitly by UPDATE statements or implicitly for standard streams) consist of a numerical value and a time stamp. For user-defined streams the numerical value is computed by evaluating the expression given in the UPDATE statement. The time stamp is generated by the system. The numerical value is interpreted as the difference to the previously observed state value, i.e., the new state is computed by adding the value just calculated to the previous state. The range of values of the numerical value is REAL. Streams of type STATE are initialized by 0.0 at model time 0.0. The estimator MEAN for streams of type STATE is defined by

$$x = \frac{1}{T} \cdot \left[\left(\sum_{i=1}^{n} (t_i - t_{i-1}) \cdot x_{i-1} \right) + (T - t_n) \cdot x_n \right]$$

where T is the observation time starting at t_0=0 with x_0=0, n is the number of state changes until T, ti the time of a state change and xi the new state value (not the difference between new and old value, which is given in the UPDATE statement). Thus the estimator for the mean is the mean of the observed values, weighted by the length of the intervals the value has been observed, if no special start condition is specified. An estimator for the standard deviation is computed with respect to this mean value, an estimator for the confidence interval is computed with respect to this mean estimator. If %parm=minmax is used within the control file (see Chapter 8), the minimum and maximum values shown in the table file are the lowest and highest values of the state trajectory.

Example:

```
STREAM MY_POPULATION,
       MY_OCCUPATION,
       MY_UTILIZATION : STATE;
```

5.1.1.3. Type COUNT

Streams of type COUNT count events for estimating rates. The pairs generated (either explicitly by UPDATE statements or implicitly for standard streams) always consist of the value 1 and a time stamp. If the expression given in the UPDATE statement does not yield 1, it is set to 1 by the system. If the counter value shall be increased by more than 1, the appropriate number of UPDATE statements have to be executed. The time stamp is generated by the system. Streams of type COUNT are initialized by 0 at model time 0.0. The estimator MEAN for streams of type COUNT is defined by

$$x = \frac{N}{T}$$

where T is the observation time (start t = 0) and N the number of the pairs generated. Thus, the estimator for the mean is the number of events observed, divided by the length of the observation interval, if no special start condition is specified. An estimator for the standard deviation is computed with respect to the inter-event time. Confidence intervals are given for mean rates. If %parm=minmax is used within the control file (see Chapter 8), the minimum and maximum values shown in the table file are the smallest and largest interevent time observed.

Example:

```
STREAM MY_THROUGHPUT : COUNT;
```

5.1.2. Standard Streams

The standard streams THROUGHPUT, TURNAROUNDTIME, POPULATION, UTILIZATION, OCCUPATION, SCHEDULE_RATE and PREEMPT_RATE are predefined for every component (area) and the model itself. They must neither be declared nor updated. THROUGHPUT, TURNAROUNDTIME, POPULATION and UTILIZATION may be used in analytical and simulative evaluations, whereas OCCUPATION, SCHEDULE_RATE and PREEMPT_RATE are allowed for simulation only. UTILIZATION is permissible for components of type server only. As models have no control procedures, it does not make sense to measure SCHEDULE_RATE or PREEMPT_RATE for them. Furthermore both streams are not allowed for component areas. These streams can be obtained for every model and component as well as for special component areas. The latter is only possible for simulations, and the word component

has to be replaced by the special component area (and the control procedure names have to be omitted) in the following explanations of these standard streams:

- THROUGHPUT (type COUNT)
The stream is updated whenever a process is leaving the component
The MEAN estimator yields the number of processes leaving the component in one time unit. The STANDARDDEVIATION estimator renders the standard deviation of the interdeparture times, not of the throughput.

- TURNAROUNDTIME (type EVENT)
The stream is updated whenever a process is leaving the component.
TURNAROUNDTIME measures the time this process has spent in the component, i.e., the difference between departure time and arrival time. The MEAN estimator yields the mean time a process is spending in the component.

- POPULATION (type STATE)
The stream is updated whenever a process is entering the component or leaving it by +1 or -1 resp.
The MEAN estimator yields the mean number of processes resident in the component.

- UTILIZATION (type STATE)
This stream is allowed (and does only make sense) for components of type server and prioserver only, component areas are not permitted. UTILIZATION measures the service speed given to the processes. It is updated whenever the speed of a process changes. The parameter speed of the standard dispatch control procedures shared, sdshared equal and sdequal (see also Appendix F.3.4.) defines the standard speed of a component. For this component UTILIZATION is normalized to this standard speed by dividing the actual speeds by the standard speed. For shared and equal, the standard speed corresponds to the actual speed of the component. For sdshared and sdequal, the standard speed is multiplied by the appropriate entry of the speeds-array, yielding the actual speed. The MEAN estimator gives the utilization of the component (type server or prioserver). Values greater than 1 are posssible, due to actual speeds being greater than standard speed and/or multiple processes in service.In the case of simulation the evaluation of UTILIZATION is more CPU-time consuming compared to the other standard measures. In some cases, there are alternative streams resulting the same measures. For dispatch control procedure shared UTILIZATION is identical to OCCUPATION of the service area. For the dispatch procedure equal the alternative measure depends on the schedule procedure. POPULATION of the server can be used with immediate, POPULATION of SERVICE_AREA with fcfs and lcfs and OCCUPATION with lcfspr, fcfs(1) and lcfs(1).

- OCCUPATION (type STATE)
This stream is allowed in case of simulation only. It is updated, if a component, which was previously empty, takes a process into itself or if the last process in the component is leaving. Arrivals (and departures) that do not find the component empty (or do not leave it empty, resp.) are ignored. The stream only knows about the two states ”component is occupied” and ”component is empty”. The MEAN estimator yields the

probability of finding the component occupied.

- SCHEDULE_RATE (type COUNT)
This stream is only allowed when using simulation. It is updated whenever the control procedure schedule puts a process from the entry area to the service area. The MEAN estimator yields the transition rate from the entry area to the service area, thus this stream must not be applied for evaluation objects referring to an area.

- PREEMPT_RATE (type COUNT)
This stream is only allowed when using simulation. It is updated whenever the control procedure schedule preempts a process (from the service area to the entry area). The MEAN estimator yields the transition rate from the service area to the entry area, thus this stream must not be applied for evaluation objects referring to an area.

5.1.3. Update Statement

The UPDATE statement is applied to generate a new value for user-defined streams. It is allowed only in the body of the service which declares the stream (old concept) or which is contained in the component type declaring the stream.

```
simple_statement  ::= ...
   | update_statement

update_statement  ::=
   UPDATE stream-name BY simple_real_expression ;
```

The stream-name is the name of the stream updated. The expression must be of type INTEGER or REAL. The interpretation of this value (and as a consequence the evaluation of the stream) depends on the type of the stream. For streams of type STATE, the value is the difference between new and previous state. For streams of type EVENT, it takes the value which was actually observed. For streams of type COUNT the value is ignored and the internal counter is incremented (by 1). Nevertheless the expression may not be omitted. The corresponding time stamps or counters are supplied by the system automatically. User-defined streams are updated at the time the UPDATE statement is executed, standard streams are updated implicitly. The number of updates which have occured for a stream can be displayed (in a table) for all kinds of streams by using %parm=updates in the control file.

```
Examples:

   UPDATE my_stream BY time - started_at;
   UPDATE s1        BY 5;
```

6.2. Scopes of Identifiers

All local identifiers declared within one block must be unambiguous. Multiple declarations of an identifier within one block leads to a HI-SLANG compile error message. The name of the EXPERIMENT block, the model name within an EVALUATE statement and the formal parameters of procedures, model types, component types and services are local in that block. These identifiers may not be redeclared within the declaration part of that block. Regardless of the sequence of notation, every declared identifier is known in the whole declaration part and the statement part of the block. It is also known in every inner block unless the identifier has been newly declared there. If it has, only the latest declaration is valid for this block as well as for all following inner blocks (in terms of the block structure). Newly declaring an identifier is thus allowed and replaces (the validity of) all previous declarations in outer blocks. Identifiers declared in the declaration part preceding the EXPERIMENT block are known in the EXPERIMENT block itself. Identifiers declared in the modelling environment (e.g., predefined procedures and constants) are global to the entire source. Relevant in terms of the scope of identifiers in procedures, record types and any type of modelling object is only the declaration environment, not the block in which a procedure has been activated or an object has been generated. Outside of a block its local declarations are not known. This is true for embracing blocks as well as parallel blocks. Some exceptions to this rule are discussed later on. Services and procedures which are local within a component type may be made available to the next-higher component type by use of the PROVIDE declaration. The upper level component type can take access to one of these provided objects using a special mechanism, the REFER part. The procedures popul, popul_announce, popul_entry, popul_service and popul_exit are known in any component type without explicitely being provided. Services provided by a component type are also known and available within the statement part of the AGGREGATE statement. The identifiers of utilized services or procedures being used within a service or procedure definition are specified in the USE declaration part, the interface of services and procedures. Just as the provided objects, they can only be used in the REFER part of the component type they belong to. Access to block-local objects and types is limited within CONCURRENT statements in services. See Section 4.1.5.1. The identifiers of components, services and used services may be accessed in the definition of evaluation objects and load filtering hierarchies. They must, however, be completely specified and qualified by navigating from the model down to the evaluation object. References to files (by link names) are not influenced by the block concept. Therefore, all used link names must be unambiguous and globally declared in the control file. Regarding the control file, they may refer only to one file or mobase object.

D.1. Operators and Precedence Rules

The following tables list the type of result (or signatures) of all HI-SLANG operators, depending on the type of their arguments.

unary operators:

	operator	
operand	+	-
INT	INT	INT
REAL	REAL	REAL

binary operators:

		operator						
left operand	right operand	+	-	*	/	//	MOD	**
INT	INT	INT	INT	INT	REAL	INT	INT	REAL
INT	REAL	REAL	REAL	REAL	REAL	INT	INT	REAL
REAL	INT	REAL	REAL	REAL	REAL	INT	INT	REAL
REAL	REAL	REAL	REAL	REAL	REAL	INT	INT	REAL

		operator				
left operand	right operand	EQV	AND	AND THEN	OR	OR ELSE
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE
FALSE	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE
FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE

		operator
left operand	right operand	&
TEXT	TEXT	TEXT

The result of a comparison is of type BOOLEAN. Both arguments to be compared must have the same type. There is one exception: Comparing operands of type REAL and INTEGER is possible, as the INTEGER operand is converted to REAL.

Precedence rules for operators:

(1) = lowest precedence EQV
(2) OR, OR ELSE
(3) AND, AND THEN
(4) NOT
(5) =, <, >, <=, >=, <>, #
(6) +, -, &
(7) *, /, //, MOD

(8) **
(9) = highest precedence (expression), OF

With n<m operators of precedence operators of class n are evaluated after those of class m. Operators of the same class are evaluated from the left to the right (left associative).

D.2. Predefined Procedures

name	type of result	name of 1. par	type of 1. par	name of 2. par	type of 2. par	usable array bounds	for	comments
abs	REAL	X	REAL	-	-	y		
arccos	REAL	X	REAL	-	-	y		
arcsin	REAL	X	REAL	-	-	y		
arctan	REAL	X	REAL	-	-	y		
cos	REAL	X	REAL	-	-	y		
cosh	REAL	X	REAL	-	-	y		
entier	INT	X	REAL	-	-	y		
exp	REAL	X	REAL	-	-	y	arithmetic	
ln	REAL	X	REAL	-	-	y	functions	
log	REAL	X	REAL	-	-	n		
maxint	INT	-	-	-	-	y		
maxreal	REAL	-	-	-	-	y		
sign	INT	X	REAL	-	-	y		
sin	REAL	X	REAL	-	-	y		
sinh	REAL	X	REAL	-	-	y		
sqrt	REAL	X	REAL	-	-	y		
tan	REAL	X	REAL	-	-	y		
tanh	REAL	X	REAL	-	-	y		
undefined		REAL	-	-	-	-	y	
char	CHAR	I	INT	-	-	-		
digit	BOOL	C	CHAR	-	-	-	character	
letter	BOOL	C	CHAR	-	-	-	functions	
rank	INT	C	CHAR	-	-	y		
cox	REAL	A	REAL	B	REAL	n		
coxg	REAL	A	ARRAY	-	-	n		
discrete		INT	A	ARRAY	-	-	n	
draw	BOOL	A	REAL	-	-	n		
erlang	REAL	A	REAL	B	REAL	n		
histd	INT	A	ARRAY	-	-	n	random	
linear	REAL	A	ARRAY	B	ARRAY	n	number	

```
negexp  REAL    A       REAL    -       -       n       generators
normal  REAL    A       REAL    B       REAL    n
poisson INT     A       REAL    -       -       n
randint INT     A       INT     B       INT     n
uniform REAL    A       REAL    B       REAL    n

cpu_time        REAL    -       -       -       -       n
get_result      REAL    *       *       *       *       n
last_seed       INT     -       -       -       -       n
put_header      -       link    TEXT    info    TEXT    -
put_footer      -       link    TEXT    info    TEXT    -
put_result      -       *       *       *       *       -       modelling
set_seed        -       new_seed        INT     -       -       -  support
stop_evaluation -       message TEXT    -       -       -       procedures
time    REAL    -       -       -       -       n
trace_off       -       -       -       -       -       -
trace_on        -       -       -       -       -       -
trace_state     -       -       -       -       -       -
transfer_results        -       -       -       -       -       -
was_message     BOOL    kind    CHAR    no      INT     -

eof     BOOL    F       INFILE  -       -       -
eoln    BOOL    F       INFILE  -       -       -
lastitem        BOOL    F       INFILE  -       -       -
lowten  -       C       CHAR    -       -       -       I/O support
sysin   INFILE  -       -       -       -       -       procedures
sysout  OUTFILE -       -       -       -       -
tracefile       OUTFILE -       -       -       -       -



Legend:
   ARRAY <=> ARRAY {[}...{]} OF REAL
       - <=> non existent
       * <=> more than two parameters, see below
```

D.2.1. Arithmetic Functions

The exact definitions (concerning precision, allowed parameter values, etc.) for most arithmetic functions are implementation defined. All functions return best possible approximations to the exact mathematical results. For argument values where the mathematical functions are not defined a run time error occurs.

D.2.1.1. Trigonometric Functions

All trigonometric functions deal with angles expressed in radians. The following functions are available:

```
sin     (X: REAL) RESULT REAL
sinh    (X: REAL) RESULT REAL
cos     (X: REAL) RESULT REAL
cosh    (X: REAL) RESULT REAL
tan     (X: REAL) RESULT REAL
tanh    (X: REAL) RESULT REAL
arcsin  (X: REAL) RESULT REAL
arccos  (X: REAL) RESULT REAL
arctan  (X: REAL) RESULT REAL
```

D.2.1.2. Other Arithmetic Functions

The following arithmetic functions are built into the HIT system:

```
abs    (X: REAL) RESULT REAL
       The result is the magnitude of X.

entier (X: REAL) RESULT INTEGER
       The result is the integer "floor" of the real X,
       the value always being less than or equal to X.
       Thus entier (1.8) returns 1, while entier (-1.8)
       returns -2.

exp    (X: REAL) RESULT REAL
       The result is eX.

ln     (X: REAL) RESULT REAL
       The result is ln X. X must be positive.

log    (X: REAL) RESULT REAL
       The result is log10 X. X must be positive.

sign   (X: REAL) RESULT INTEGER
       The result is zero if X is zero, +1 if X
       is positive and -1 if X is negative.

sqrt   (X: REAL) RESULT REAL
       The result is square root of X.
       X must not be negative.
```

D.2.1.3 Installation-dependent Functions

These functions result values defined by the underlying hardware or Simula system.

maxreal RESULT REAL

The maxreal value is the maximal real representable by your hardware. If this value is exceeded during REAL calculation an overflow results from the SIMULA Run Time System.
The smallest REAL is - MAXREAL.

undefined RESULT REAL

The undefined value is defined as maxreal / 4 (to avoid calculation overflows). It is especially used in dump files for performance measures which could not be determined (in table files the word ”Undefined” is displayed instead). If you read values from a dump file by get_result you should take into account that they may be undefined.

maxint RESULT REAL

The maxint value is the maximal integer representable by your hardware. If this value is exceeded during INTEGER calculation an integer overflow results from the SIMULA Run Time System.

D.2.2. Character Functions

All character functions deal with characters. In HI-SLANG the set of characters is either coded in ASCII or EBCDIC (see Appendix C.3.).

char (I: INTEGER) RESULT CHARACTER

The result is the character obtained by converting I according to the implementation defined coding of characters. I must be in the range 0..255, otherwise a run time error occurs. The inversion of char is rank. char (rank (c)) = c always holds.

digit (C: CHARACTER) RESULT BOOLEAN

The result is TRUE if the character C represents a decimal digit.

letter (C: CHARACTER) RESULT BOOLEAN

The result is TRUE if the character C is a letter of the alphabet (’a’..’z’, ’A’..’Z’).

rank (C: CHARACTER) RESULT INTEGER

The result is an integer in the range 0..255, obtained by converting C according to the implementation defined character code. The inversion of rank is char. For i in 0..255

rank (char (i)) = i always holds.

D.2.3. Random Number Generators

All (pseudo-)random drawing procedures of HI-SLANG are based on the technique of obtaining ”basic drawings” from the uniform distribution in the interval]0,1[. A basic drawing (call of a random number generator) replaces the value of the predefined integer variable seed by a new value according to an implementation defined algorithm, e.g.,
seed (i+1) = mod (seed (i) * 5** (2*p+1), 2**n)
and generates a stream of pseudo-random numbers in the interval [1,(2**n)-1]. The result of the i+1th basic drawing called u(i+1) is calculated by
u(i+1) = seed (i+1) / 2**n
The n is an integer related to the size of a computer word (e.g., 35) and p is a positive integer (e.g., 6). The initial value of seed is 13 within a globally declared procedure or within the experiment block. Within a model, component or service the predefined parameter seed (with default 13) of the model type denoting the root of the hierarchy is used (see chapter 4.).

It can be proved that if seed (= seed (0)) is a positive odd integer, the same holds for seed (i), and the sequence seed (0), seed (1), ... is cyclic with period 2**n-2.

Of course the user may write his own random drawing procedures based on those listed below, e.g.,

```
PROCEDURE geometric_distribution (m : REAL) RESULT REAL;
   VARIABLE res : INTEGER;
   BEGIN
      WHILE draw (m) LOOP res := res + 1; END LOOP;
      RESULT res;
   END PROCEDURE;
```

The following random drawing procedures are available in HI-SLANG:

cox (A, B: REAL) RESULT REAL
The result is a COX-distributed pseudo-random number with rate A (reciprocal mean value) and variation coefficient B. If A <= 0 or B < 0.1, a run time error occurs.

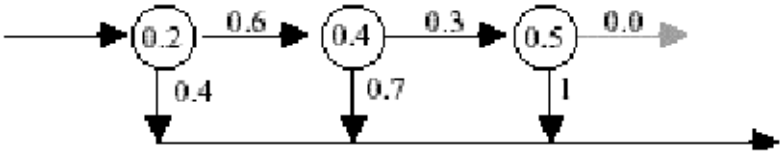
coxg (A: ARRAY OF REAL) RESULT REAL
The result is a COX-distributed pseudo-random number. A must be a two dimensional ARRAY OF REAL or ARRAY aggregate denoting step-progress-probabilities for all service rates given in the first ”line” of the array.

Example:

The array

A	1	2	3
1	0.2	0.4	0.5
2	0.6	0.3	0.0

or aggregate [[0.2,0.4, 0.5], [0.6, 0.3, 0.0]]
describes the following COX-distribution:



The values on the vertical arrows are not found in the array; they are the complement of the values in the second sub array.
Please note that the last value (in the second sub array) must be zero, while all the other array elements must be positive, otherwise a run time error occurs.

discrete (A: ARRAY OF REAL) RESULT INTEGER
The one-dimensional ARRAY OF REAL A, augmented by the element 1 to the right (for safety reasons), is interpreted as a step function of the subscript, defining a discrete (cumulative) distribution function. All array elements must be in the interval [0, 1] and in ascending order. The result of the function is the smallest index i such that A[i] > u, where u is the value of the basic drawing. The result thus is an integer in the range A.lower_bounds[1] .. A.upper_bounds[1]+1.

draw (A: REAL) RESULT BOOLEAN
The result is TRUE with the probability A, FALSE with the probability 1-A. For A <= 0 the result is always FALSE, for A >= 1 the result is always TRUE.

erlang (A, B: REAL) RESULT REAL
The result is a drawing from the Erlang distribution with mean 1/A and standard deviation 1/(A*sqrt(B)). Both the rate A and the number of phases B must be greater than zero.

histd (A: ARRAY OF REAL) RESULT INTEGER
The one-dimensional ARRAY OF REAL A is interpreted as a histogram defining the relative frequencies of the array elements. The relative frequencies must be less or equal 1, the sum over all elements must be one. The result of the function is an index, that is an integer in the range A.lower_bounds[1] ..A.upper_bounds[1].

linear (A, B: ARRAY OF REAL) RESULT REAL

A and B must be one-dimensional ARRAYs OF REAL with equal lower and upper bounds lb and ub. They define a cumulative distribution f. The result of type REAL is determined by linear interpolation in the non-equidistant distribution table, defined by A and B. The function f is defined by
A[i] = f(B[i]) for lb <= i <= ub
To avoid run time errors or installation dependent results, the following conditions must hold:
A[lb] = 0 and A[ub] = 1 and both arrays must be monotonous, i.e., for lb<= i <= ub, both, A[i] <= A[i+1] and B[i] <= B[i+1] hold.

negexp (A: REAL) RESULT REAL
The result is a drawing from the negative exponential distribution with mean 1/A (rate A), defined by - ln(u)/A, where u is the basic drawing. This is the same as a random ”waiting time” in a Poisson distributed arrival pattern with expected number of arrivals per time unit equal to A. If A is non-positive, a run time error occurs.

normal (A, B: REAL) RESULT REAL
The result is normally distributed with mean A and standard deviation B. B must be non-negative.

poisson (A: REAL) RESULT INTEGER
The result is a drawing from the Poisson distribution with parameter A. The result n is defined by n+1 basic drawings u(i), such that n is the smallest non-negative integer with u(0)*u(1)*...*u(n) ¡ exp(-A). For negative A, the result is defined to be zero. When A is greater than some implementation defined value, e.g., 20, the result may be approximated by the maximum of entier (normal (A, sqrt(A) + 0.5)) and zero.

randint (A, B: INTEGER) RESULT INTEGER
The result is one of the integers A, A+1,..., B-1, B with equal probability. If B < A, an error occurs.

uniform (A, B: REAL) RESULT REAL
The result is a uniformly distributed pseudo-random REAL number in the interval [A..B]. If B < A, an error occurs.

Anhang C

Modellierung und Bewertung von Supply Chain-Modellen unter Berücksichtigung variierender Strukturen

Der folgende Abschnitt stammt aus [9]. Die Seitenzahlen und die Nummerierung der Abschnitte wurden dabei aus dieser Arbeit übernommen.

Kapitel 5

Modifikationen im ProC/B-Editor

Dieses Kapitel beschäftigt sich mit den im Rahmen der Diplomarbeit vorgenommenen Änderungen im ProC/B-Editor. Sämtliche Änderungen sind dabei so gestaltet, dass sie sich nicht auf die nachgelagerten Tools auswirken. Dies bedeutet, dass die Übersetzung nach B1 sowie die Transformation nach HIT, die Eingabesprache des Simulators, unverändert übernommen werden können.
Die Modifikationen ergeben sich aus den Anforderungen der Darstellung von Supply Chains. Wie bereits in Kapitel 2 dargestellt, handelt es sich bei einer Supply Chain um ein komplexes Netzwerk rechtlich selbständiger Unternehmen. Für die Umsetzung im ProC/B-Editor bedeutet dies, dass sich die Akteure auf einer Hierachieebene befinden. Eine Hierarchie, die sich aus den Prozessabläufen und Aufrufrelationen ergibt, erleichtert zwar in einigen Fällen die Modellierung, spiegelt aber nicht die Realität exakt wider. Als ein Beispiel einer solchen an den Abläufen orientierten Hierarchie sei hier auf *Kaczmarek und Völker* (2003, S. 18) verwiesen. Die rechtliche Selbständigkeit und die sich im ersten Schritt daraus ergebende flache Hierarchie führen zu einer vermehrten Verwendung von externen Funktionseinheiten. Externe Funktionseinheiten ermöglichen die Nutzung von Diensten, die von Funktionseinheiten derselben Ebene angeboten werden. Im Gegensatz zu einem internen Dienstaufruf innerhalb der Hierarchie, bei dem der Dienst von einer enthaltenen Funktionseinheit angeboten wird und damit sichtbar ist, kann so auf Dienste zugegriffen werden, die lokal nicht sichtbar sind.[1] Zur Verdeutlichung erfolgt in Abbildung 5.1 eine Gegenüberstellung der Nutzung von Diensten externer und interner Funktionseinheiten. Dargestellt wird ein Modell bestehend aus zwei Akteuren. Der Lieferant bietet einen Dienst „transportieren“ an. Der OEM, der aufgrund der rechtlichen Gleichstellung auf derselben Hierarchieebene liegt, verfügt über eine interne Funktionseinheit „Produktion“, die den Dienst „produzieren“ anbietet. Dieser Dienst ist gemäß der Enthaltensein-Relation im OEM sichtbar und nutzbar. So kann dort in der Prozessket-

[1] Vgl. Beilner u.a. (1999), S. 66 f..

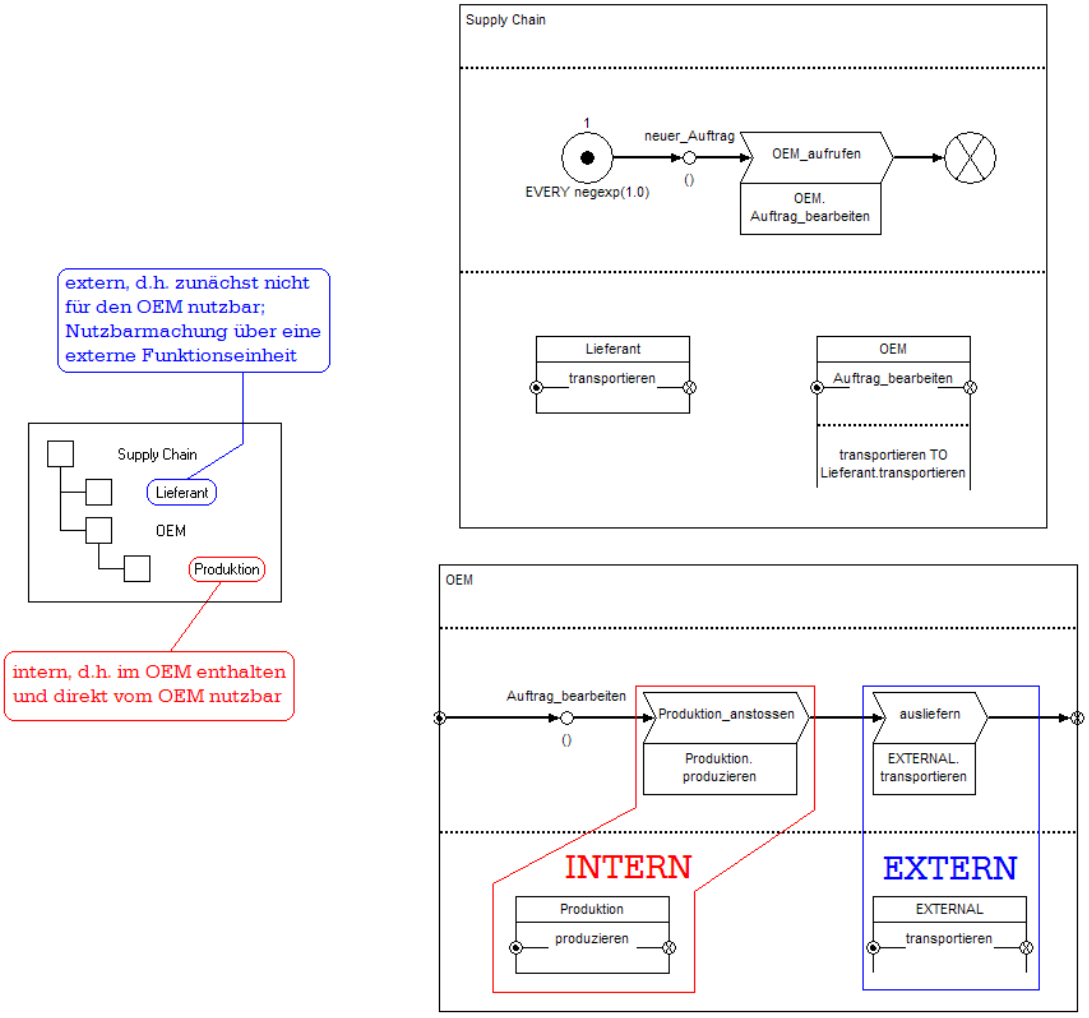


Abbildung 5.1: Gegenüberstellung interner und externer Funktionseinheiten; links der Hierarchiebaum mit der Aussenansicht der Funktionseinheiten, rechts die Innenansicht der Supply Chain und des OEM. (Quelle: eigene Darstellung)

te „Auftrag_bearbeiten“ auf diesen Dienst zugegriffen werden. Der Dienst „transportieren“ des Lieferanten ist bisher nur in der Modellumwelt „Supply Chain“ sichtbar. Um ihn auch dem OEM anbieten zu können, muss in der Aussenansicht des OEM dieser Dienst importiert werden. In der Innenansicht wird eine Funktionseinheit „EXTERNAL“ angelegt, die den Zugriff auf den importierten Dienst des Lieferanten ermöglicht.

Neben der rechtlichen Selbständigkeit führt die Anzahl der Unternehmen zu einer Besonderheit bei Supply Chain-Modellen. Die Vielzahl an Akteuren und ihrer Austauschbeziehungen resultiert in komplexen Modellen. Die Komplexität bezieht sich zum einen auf die Länge einzelner Prozesse und zum anderen auf die Menge an Prozessen. Prozessketten in einer Supply Chain sind i. d. R. lang, d. h. sie umfassen viele Aktivitäten, da sie den gesamten Leistungserstellungsprozess vom

ersten Lieferanten bis zum Kunden darstellen müssen. Neben diesen Hauptprozessen existiert eine Vielzahl von Prozessen, die der Steuerung, Kontrolle oder Unterstützung der Hauptprozesse dienen. So ist ein Kundenauftrag, der beim Einzelhändler eingeht und über einen Großhändler an den OEM und eventuell die Lieferanten weitergeleitet wird, ein Beispiel für einen Hauptprozess, der sich über die gesamte Supply Chain erstreckt. Ein Hilfsprozess ist die Wartung einer Maschine. Vereinfacht kann gesagt werden, dass sich die Hauptprozesse auf oberster Unternehmensebene im Modell ereignen, während sich die Hilfsprozesse in die Tiefe der Hierarchie eines Unternehmens erstrecken. Dies führt bei Supply Chain-Modellen dazu, dass sich sowohl horizontale Prozesse über eine Anzahl von Akteuren als auch vertikale Prozesse, die sich in einem Unternehmen abspielen, bilden und so sehr umfangreiche Modelle entstehen.

Bei Supply Chains handelt es sich zwar um feste Strukturen, jedoch muss aus vielfachen Gründen, von denen Kapitel 3 einige näher betrachtet, die Struktur des Netzwerkes modifiziert werden. Unternehmen verlassen den Verbund, neue Unternehmen kommen hinzu, Prozesse und Produkte ändern sich ebenso wie die Kundennachfrage, kurz gesagt, die reale Struktur wandelt sich im Verlauf der Zeit. Hieraus ergeben sich zwei Anforderungen an die Modellierung im ProC/B-Editor. Zunächst soll eine Unterstützung des Modellierers zu einer vereinfachten Umgestaltung bereits erstellter Modelle erfolgen. Es entstehen mehrere Modelle einer Supply Chain, so dass unterschiedliche Strukturen in Simulationsläufen untersucht werden können. Dabei gilt es zu beachten, dass jedes Modell separat ausgewertet wird. Bei einer zweiten Modellierungsart werden die Strukturveränderungen in einem einzigen Modell dargestellt. Dies ermöglicht die Untersuchung von Strukturveränderungen während eines Simulationslaufes. Beispielsweise kann analysiert werden, wie sich eine eingespielte Supply Chain verhält, wenn Akteure wegfallen oder ersetzt werden. Hier sei noch einmal auf die Beispiele Just-in-time (siehe Kapitel 3.5) und E-Business (siehe Kapitel 3.6) verwiesen. Anzumerken ist, dass in dieser Diplomarbeit eine Konzentration auf die Modellierung variierender Strukturen erfolgt. Auf eine konkrete Auswertung der Simulationsläufe wird verzichtet.

Nach dieser Einführung in die Besonderheiten der Umsetzung von Supply Chains in ProC/B-Modelle folgt eine Erläuterung der durchgeführten Änderungen. Zunächst wird das Ein- und Ausblenden von Modellteilen vorgestellt (Kapitel 5.1). Dies kann von Hand, in Abhängigkeit einer Variablen oder im Zusammenhang mit einem Oder-Konnektor geschehen. In Kapitel 5.2 wird dargelegt, inwieweit Änderungen in Bezug auf externe Funktionseinheiten stattgefunden haben. Dabei ist anzumerken, dass diese Änderungen durchaus auch ohne die Verwendung externer Funktionseinheiten von Nutzen sein können. Daran schließt sich in Kapitel 5.3 die Vorstellung der Modifikationen

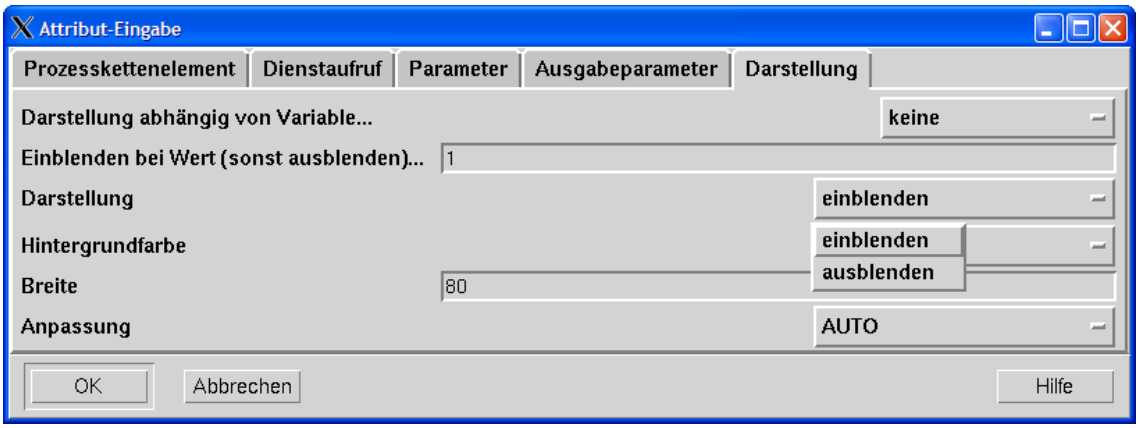


Abbildung 5.2: Attributmenü zum Ein- und Ausblenden. (Quelle: eigene Darstellung)

an, die eine vereinfachte Umgestaltung der Modellstruktur erlauben. Abschließend wird das Entscheidungselement eingeführt (Kapitel 5.4). Es ermöglicht eine Ausgliederung von Teilstrukturen aus einer Prozesskette, um diese übersichtlich zu halten und nicht mit alternativen Abläufen zu überladen.

Die folgenden Beschreibungen der Änderungen können als eine Art Handbuch verstanden werden. Durch die Erläuterung des Grundes der Modifikation sowie des Vorgehens anhand von Beispielen auf verbale und grafische Weise kann die Verwendung der Neuerungen gut nachvollzogen werden.

5.1 Ein- und Ausblenden von Teilstrukturen

Das Ein- und Ausblenden von Teilstrukturen verfolgt das Ziel der vereinfachten Präsentation komplexer Modelle. Die Komplexität der Supply Chain-Modelle beruht, wie in der Einleitung erwähnt, auf der Vielzahl der Prozesse, aber auch gerade im Rahmen der Modellierung von Veränderungen in der Darstellung alternativer Strukturen in einem Modell. Zur Realisation mehrerer Alternativen in einem Modell eignen sich Oder-Konnektoren. Dabei stellt sich das Problem, dass sich eine stark verzweigte Struktur ergibt, die wenig übersichtlich ist. Um diesem entgegen zu wirken, werden schrittweise Änderungen vorgestellt, die es erlauben, komplexe Strukturen vereinfacht darzustellen.

5.1.1 Ein- und Ausblenden einzelner Elemente

In einem ersten Schritt wird es dem Modellierer ermöglicht, über die Attribute eines Elementes die Darstellung dieses Elementes ein- bzw. auszuschalten.[2] Er kann so die Elemente ausblenden, die

[2] Vgl. Abbildung 5.2.

in seiner aktuellen Modellierungsphase nicht wichtig sind. Dies verschafft ihm gerade bei großen Funktionseinheiten mehr Übersicht und verringert dadurch Modellierungsfehler. Ein zweiter Vorteil besteht in der nachfolgenden Präsentation der Modelle. In der Regel werden die Modelle auch Personen vorgestellt, die mit dem Modellierungstool[3] oder der dargestellten Struktur[4] nicht vertraut sind. Das Ausblenden und fallweise Einblenden von Elementen reduziert die Komplexität und erlaubt es dem Modellierer, sein Modell vor der Präsentation komplett zu erstellen und während der Präsentation die Strukturen schrittwiese zu erläutern. So kann er den Prozessablauf, wie er sich in der Realität ergibt, in kleinen Abschnitten vorstellen. Für die beiden genannten Personengruppen bietet sich der Vorteil eines besseren Verständnisses, da sie nicht sofort mit dem Gesamtmodell überfordert werden, sondern durch die Entwicklung der Struktur den Überlegungen des Modellierers einfacher folgen können.
Zum Vorgehen beim Ausblenden ist anzumerken, dass die Elemente noch gezeichnet werden, ihre Vordergrundfarbe aber gleich ihrer Hintergrundfarbe ist. In der Regel ist die Zeichenfläche ebenso wie der Hintergrund weiss, so dass beim Ausblenden das gesamte Element weiss dargestellt wird und somit nicht mehr sichtbar ist, da es sich nicht vom Hintergrund abhebt. Durch Anklicken der Position kann das Element jedoch markiert werden und erscheint in rot, so dass jederzeit das entsprechende Kontextmenü aufgerufen und das Element wieder eingeblendet werden kann. Das Ausblenden funktioniert also auf diese Weise nur, wenn die Farbe der Zeichenfläche mit der Hintergrundfarbe des auszublendenden Elementes übereinstimmt.

5.1.2 Festlegen von Abhängigkeiten

Das Ein- und Ausblenden von Elementen wird in einem zweiten Schritt um das Festlegen von Abhängigkeiten erweitert.[5] Wiederum über die Attribute des entsprechenden Elementes kann eine Variable ausgewählt werden, deren Wert die Darstellung des Elementes beeinflusst. In einem Menü werden alle Variablen aufgeführt, die für eine solche Abhängigkeit in Frage kommen. Sie müssen in der Funktionseinheit definiert und vom Typ „INT“, also ganzzahlig, sein. Zum Umgehen einer solchen Abhängigkeit gibt es in dem Menü eine Option „keine“, die die Darstellung zurücksetzt. Hat der Modellierer eine Variable ausgewählt, so kann er den Wert festlegen, den die Variable einnehmen muss, damit das Element eingeblendet wird. Bei allen anderen Werten wird es ausgeblendet. Liegt eine Abhängigkeit vor, so kann im gleichen Schritt zwar noch ausgewählt werden, ob das Element ein- oder ausgeblendet werden soll, die Auswahl findet jedoch

[3] Im Rahmen des Supply Chain Managements können dies z. B. Betriebswirte oder Logistiker sein.
[4] Im Rahmen des Supply Chain Managements können dies z. B. Informatiker sein.
[5] Vgl. Abbildung 5.3.

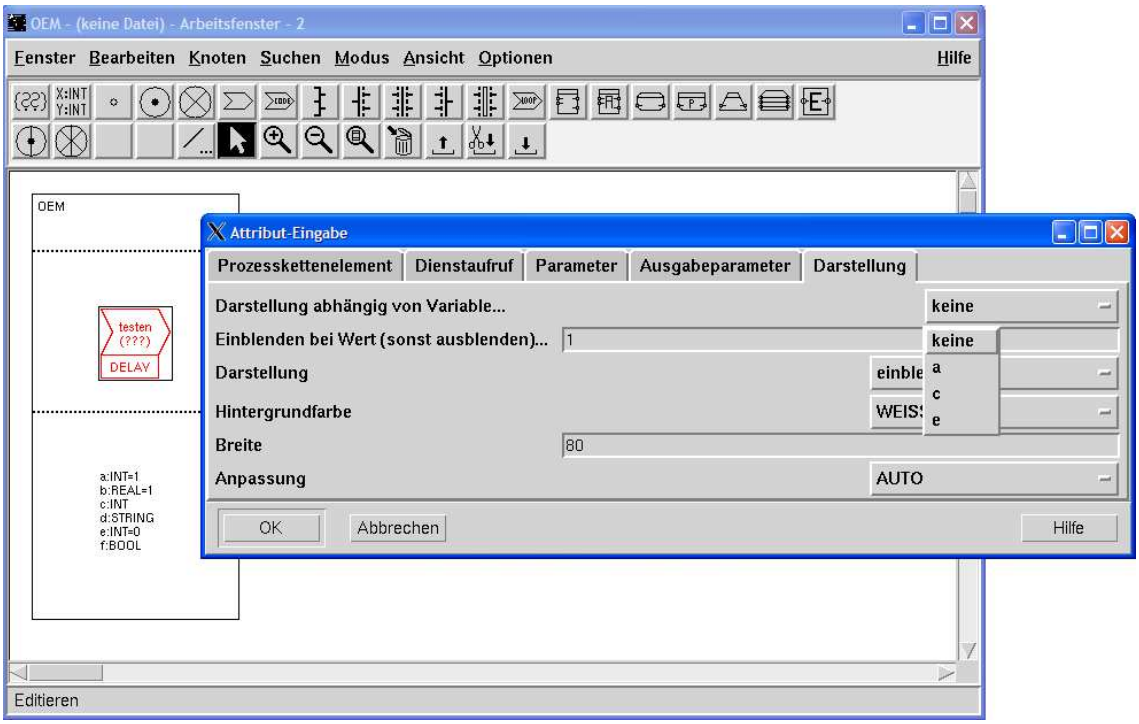


Abbildung 5.3: Attributmenü zum Festlegen von Abhängigkeiten. Es können alle Variablen der Funktionseinheit gewählt werden, die vom Typ „INT“ sind, in diesem Fall also a, c und e. Wird der Punkt „keine“ gewählt, werden eventuell vorhandene Abhängigkeiten gelöscht. (Quelle: eigene Darstellung)

keine Berücksichtigung mehr. Wird das Attributfenster geschlossen und erneut geöffnet, so ist die Auswahlmöglichkeit deaktiviert. Ändert der Modellierer nun den Wert einer solchen Variablen, so wird die Darstellung der abhängigen Elemente aktualisiert. Wird eine solche Variable entfernt oder ihr Typ ungleich „INT“ gesetzt, so werden die betroffenen Elemente wieder eingeblendet und die Variable, von der ihre Darstellung abhängig sein soll, auf „keine“ zurückgesetzt.

Eine Verwendung solcher Abhängigkeitsvariablen versetzt den Modellierer in die Lage, die Darstellung mehrerer Elemente gleichzeitig zu beeinflussen. So kann er beispielsweise Prozessketenabschnitte ein- oder ausblenden. Diese Funktion bietet sich vor allem bei der Verwendung von Konnektoren an. Indem der Modellierer alle Elemente eines Zweiges von einer Variablen abhängig macht, kann eine differenzierte Visualisierung des Modells je nach aktueller Variablenbelegung erfolgen. Auf diese Weise kann eine Komplexitätsreduzierung gerade bei großen Modellen mit vielen alternativen Prozessabläufen erreicht werden. Dies wirkt sich aufgrund der größeren Übersichtlichkeit vorteilhaft bei der Erstellung, Überprüfung und Präsentation der Modelle aus.

5.1.3 Gruppen ein- und ausblenden

Das Ein- und Ausblenden einzelner Elemente ist gerade bei einer großen Anzahl mühsam. Um diese Arbeit zu vereinfachen, kann ein Rahmen um die betroffenen Elemente gezogen werden, so dass alle markiert werden. Erfolgt nun ein Doppelklick auf die erste Maustaste im Rahmen auf eine Stelle, die nicht durch ein Element besetzt ist, so öffnet sich ein Kontextmenü. Die markierten Elemente können ein- oder ausgeblendet werden. Ebenso kann eine Abhängigkeit der Darstellung der markierten Elemente festgelegt werden. Wird dieser Punkt gewählt, so steht wiederum die Liste aller möglichen Variablen zur Verfügung. Hat der Modellierer eine Variable angeklickt, so erfolgt eine Abfrage, bei welchem Wert die Elemente eingeblendet werden sollen. Vorherige Abhängigkeiten werden dabei gelöscht.[6]

Sollen ganze Prozessketten ausgeblendet werden, muss bei den FEQuellePorts und den FESenke-Ports eine Besonderheit beachtet werden. Ihre Darstellung ist abhängig von der ein- bzw. ausgehenden Kante. Ist sie ausgeblendet, wird auch der entsprechende Port ausgeblendet. Das Ausblenden der Ports findet jedoch erst statt, wenn sie neu gezeichnet werden müssen. Sollten die Ports also dargestellt sein, obwohl die entsprechende Kante ausgeblendet ist, kann der Modellierer den entsprechenden Port anklicken. Seine Darstellung wird aufgehoben, der Rahmen wird durch eine Linie vervollständigt. Der Vorgang beim Einblenden verläuft analog.

Abbildung 5.4 zeigt anhand eines konkreten Beispiels das Ausblenden einer Gruppe und das anschließende Einblenden einzelner Elemente.

5.1.4 Der Oder-Konnektor als Alternativenbaustein

Der Gedanke des fallweisen Ein- und Ausblenden wird auch auf den Oder-Konnektor übertragen. Zur Realisation mehrerer alternativer Strukturen in einem Modell bietet sich die Verwendung von Oder-Konnektoren an. Je nach erfüllter Bedingung wird einem Zweig des Konnektors gefolgt, so dass sich die Prozesskette ihren Weg durch die verzweigte Struktur sucht. So kann beispielsweise beim Vorhandensein mehrerer Logistikdienstleister über eine Variable in Verbindung mit einem Oder-Konnektor festgelegt werden, welcher Dienstleister gewählt werden soll. Dies ergibt eine Vielzahl alternativer Prozessreihenfolgen und komplexer Prozessketten. Um sich einen Überblick über die aktuell gewählte Struktur zu verschaffen, muss an jedem Oder-Konnektor geprüft werden, welche Bedingung erfüllt ist. Dies erhöht die Komplexität eines in der Regel schon umfangreichen Modells.

An diesem Punkt setzt die Erweiterung des Oder-Konnektors an. In seinem Attributmenü kann

[6] Vgl. zum Festlegen von Abhängigkeiten von Gruppen das Beispiel in Abbildung 6.16.

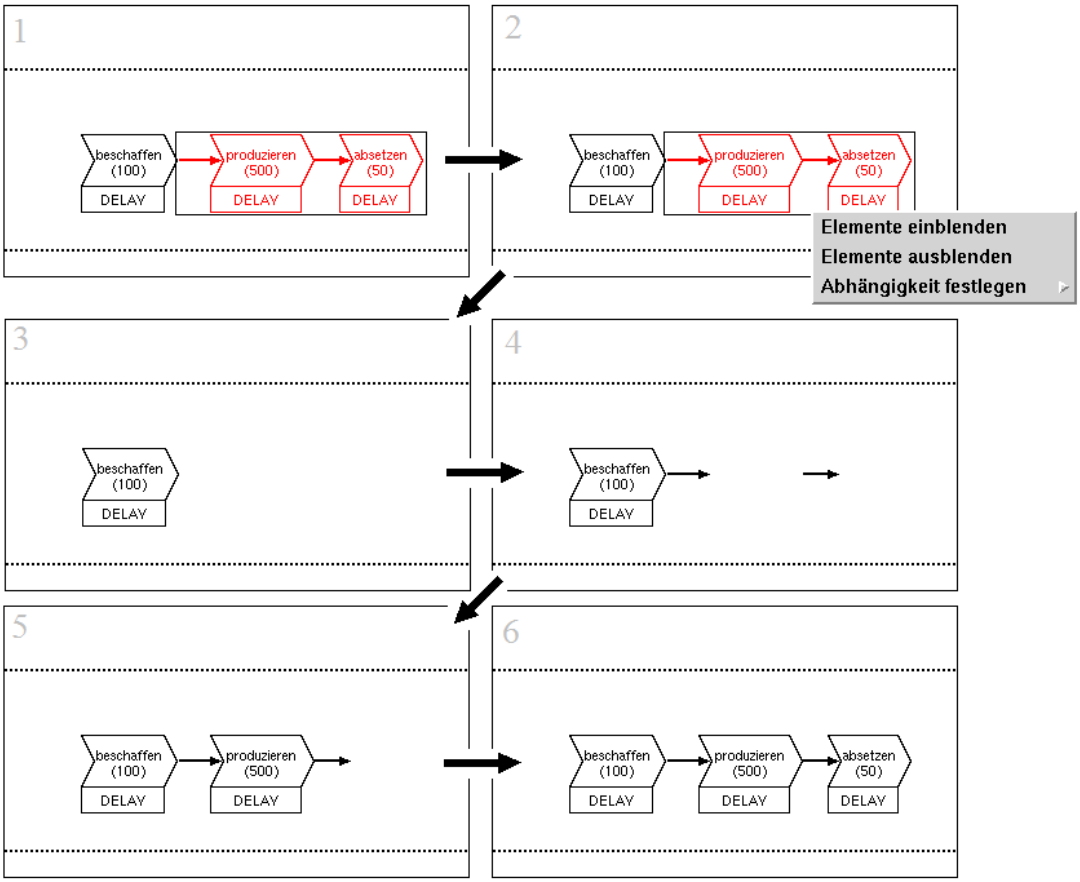


Abbildung 5.4: Ein- und Ausblenden von Elementen. Der Modellierer legt zunächst die Elemente an und markiert dann diejenigen, die er ausblenden möchte (1). Im Kontextmenü wählt er den Punkt „Elemente ausblenden“ (2), worauf sich Bild (3) ergibt. Zur Verdeutlichung, dass die Elemente existieren, aber in der Hintergrundfarbe dargestellt sind, sind in Schritt (4) die Kanten wieder eingeblendet. Schritte (5) und (6) zeigen das aufeinander folgende Einblenden der Elemente, wie es beispielsweise bei einer Präsentation erfolgen könnte. (Quelle: eigene Darstellung)

festgelegt werden, ob er als „STANDARD“ oder als „ALTERNATIVENBAUSTEIN“ verwendet werden soll. Im Fall „STANDARD“ erfolgt keine Änderung zur bisherigen Verwendung. Die Verwendung als „ALTERNATIVENBAUSTEIN“ erlaubt das fallweise Ein- und Ausblenden der nachfolgenden Zweige. Wird diese Option gewählt, wird geprüft, ob Variablen vorhanden sind, die Einfluss auf die Bedingungen der Zweige des Oder-Konnektors haben können. Ist dies nicht der Fall, so wird seine Verwendung auf „STANDARD“ zurückgesetzt. Ist wenigstens eine solche Variable vorhanden, so wird ihr Wert mit den Bedingungen verglichen. Ist bei der aktuellen Belegung die Bedingung erfüllt, so wird die nachfolgende Kante samt aller Elemente des Zweiges bis zum schließenden Oder-Konnektor eingeblendet. Im negativen Fall werden diese Elemente ausgeblendet. Ist keine Bedingung erfüllt, so wird auf einen „else“-Zweig getestet. Ist dieser vorhanden, so wird er eingeblendet. Ist auch kein „else“-Zweig zu finden, so wird die Verwendung des Oder-Konnektors ebenso auf „STANDARD“ zurückgesetzt.
Als weitere Option kann der Modellierer wählen, ob der eingeblendete Zweig verschoben werden soll. Eine Verschiebung erfolgt in der Weise, dass versucht wird, die Prozesskette in einer Linie darzustellen. Abbildungen 5.5 bis 5.8 geben abschließend ein Beispiel zur Verwendung des Oder-Konnektors als Alternativenbaustein.

5.2 Ergänzungen im Rahmen externer Funktionseinheiten

Im Rahmen der Nutzung externer Funktionseinheiten werden zwei wesentliche Änderungen vorgenommen. Die erste Modifikation bezieht sich auf das Anlegen externer Funktionseinheiten und bietet dem Modellierer hierbei Unterstützung (siehe Abschnitt 5.2.1). Die zweite Ergänzung ermöglicht es, Dienste tiefer gelegener Funktionseinheiten auch auf Hierarchieebenen anzubieten, die nicht direkt übergeordnet sind. Abschnitt 5.2.2 geht näher auf diese Änderung ein.

5.2.1 Das Anlegen externer Funktionseinheiten

Wie in der Einleitung zu diesem Kapitel bereits erwähnt, ist die Verwendung externer Funktionseinheiten bei der Modellierung von Supply Chains in nur wenigen Fällen zu umgehen. Aus diesem Grund wird der Umgang mit ihnen erleichtert. Dazu ist zunächst der bisherige Vorgang zu erläutern. Um einen Dienst importieren zu können, müssen die Funktionseinheit, die den Dienst anbietet, und die Funktionseinheit, die den Dienst nutzen soll, auf einer Hierachieebene liegen. Dies bedeutet, sie müssen im selben Subgraphen angelegt werden. Ist dies der Fall, kann in der Aussenansicht der Funktionseinheit, die den Dienst importiert, d. h. in der Innenansicht ihres Vaters, das Kontextmenü geöffnet und der Punkt „Attribute“ ausgewählt werden. Unter dem Reiter

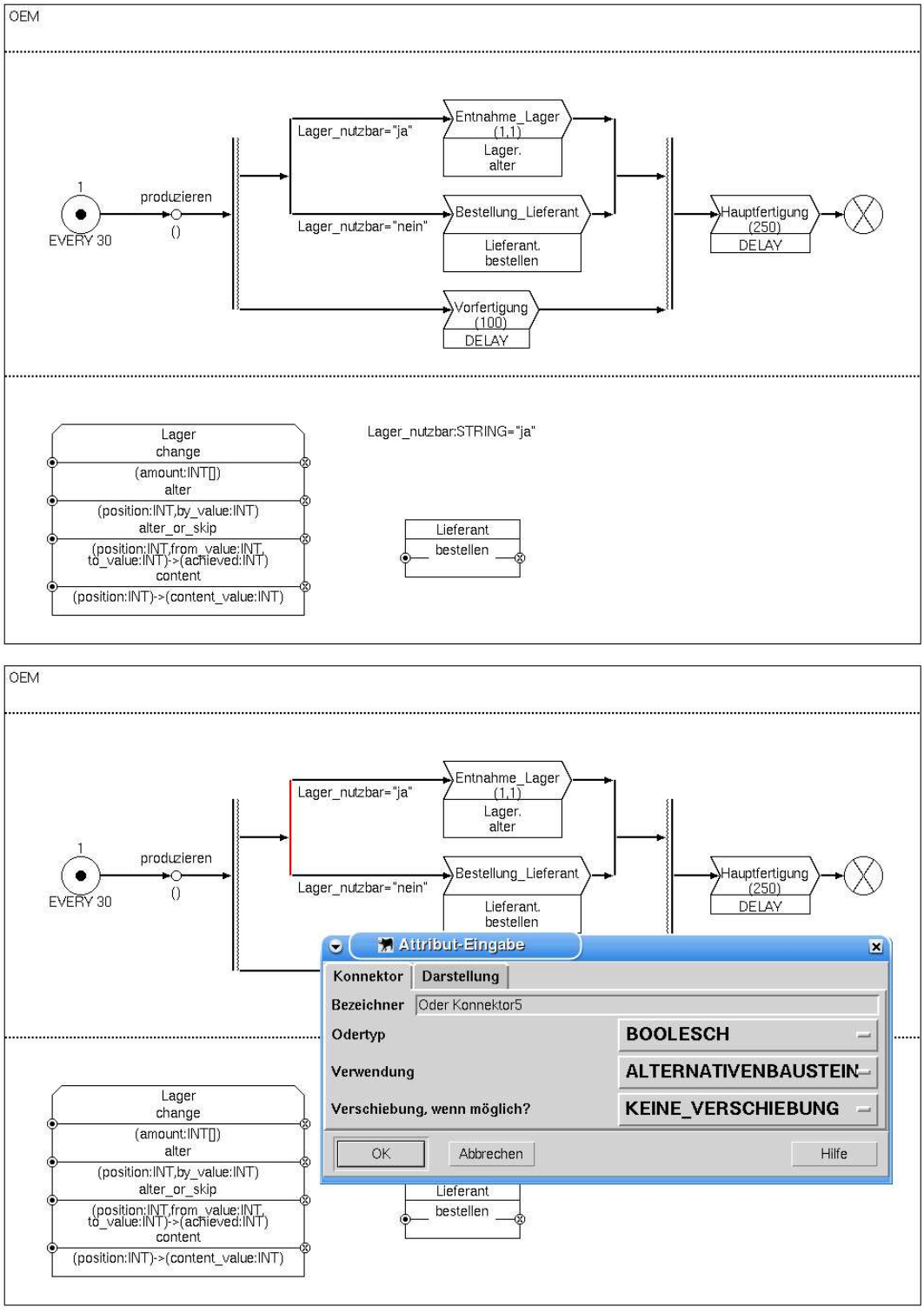


Abbildung 5.5: Der Oder-Konnektor als Alternativenbaustein, Teil 1.
Oben: Ausgangsstruktur.
Unten: Markieren des Oder-Konnektors, Aufrufen des Attributmenüs und Auswahl der Punkte „ALTERNATIVENBAUSTEIN“ und „KEINE_VERSCHIEBUNG“. (Quelle: eigene Darstellung)

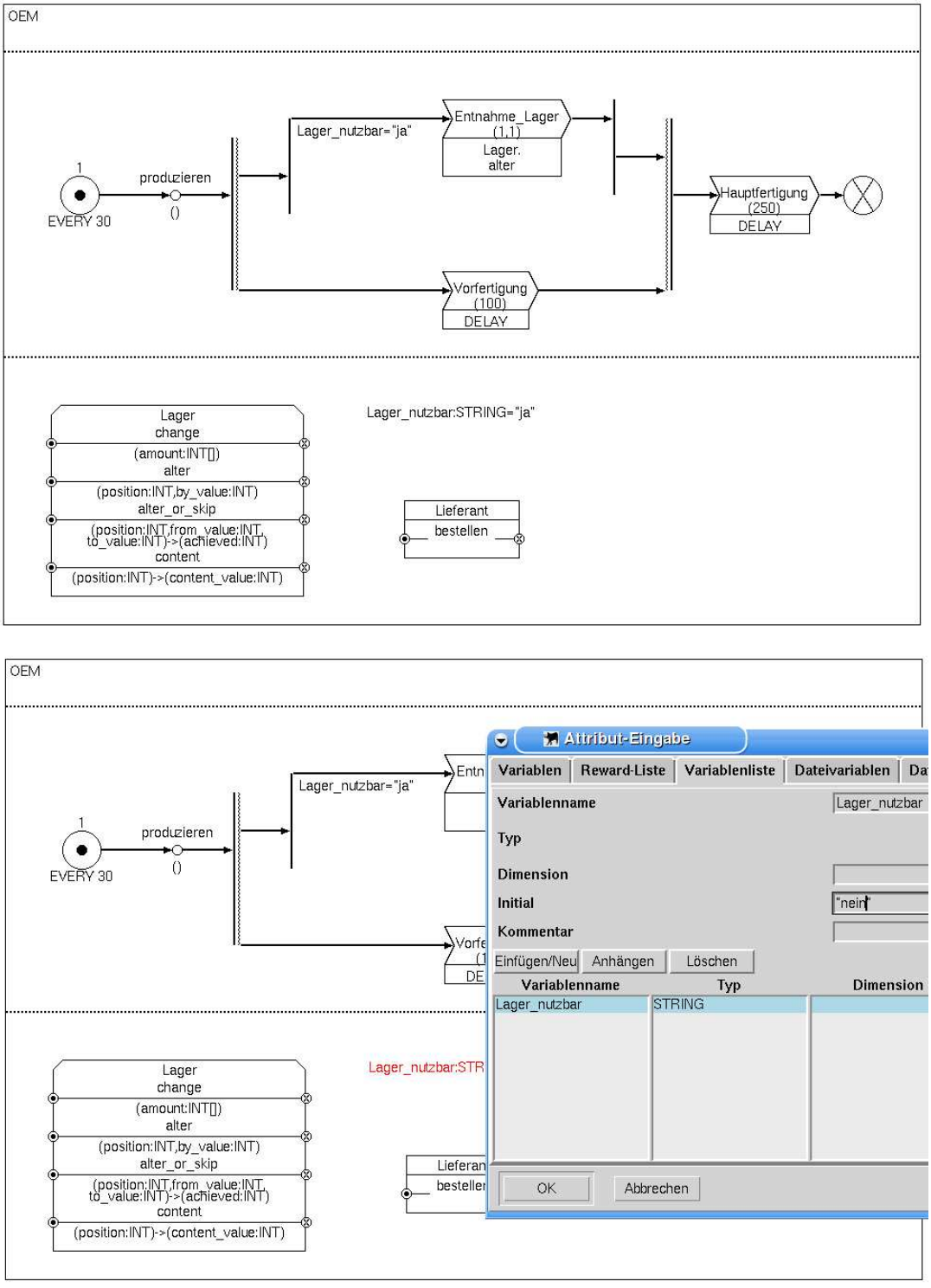


Abbildung 5.6: Der Oder-Konnektor als Alternativenbaustein, Teil 2.
Oben: Modell bei Verwendung des Oder-Konnektors als Alternativenbaustein ohne Verschiebung.
Unten: Ändern der Variablen „Lager_nutzbar“ von „ja“ auf „nein“. (Quelle: eigene Darstellung)

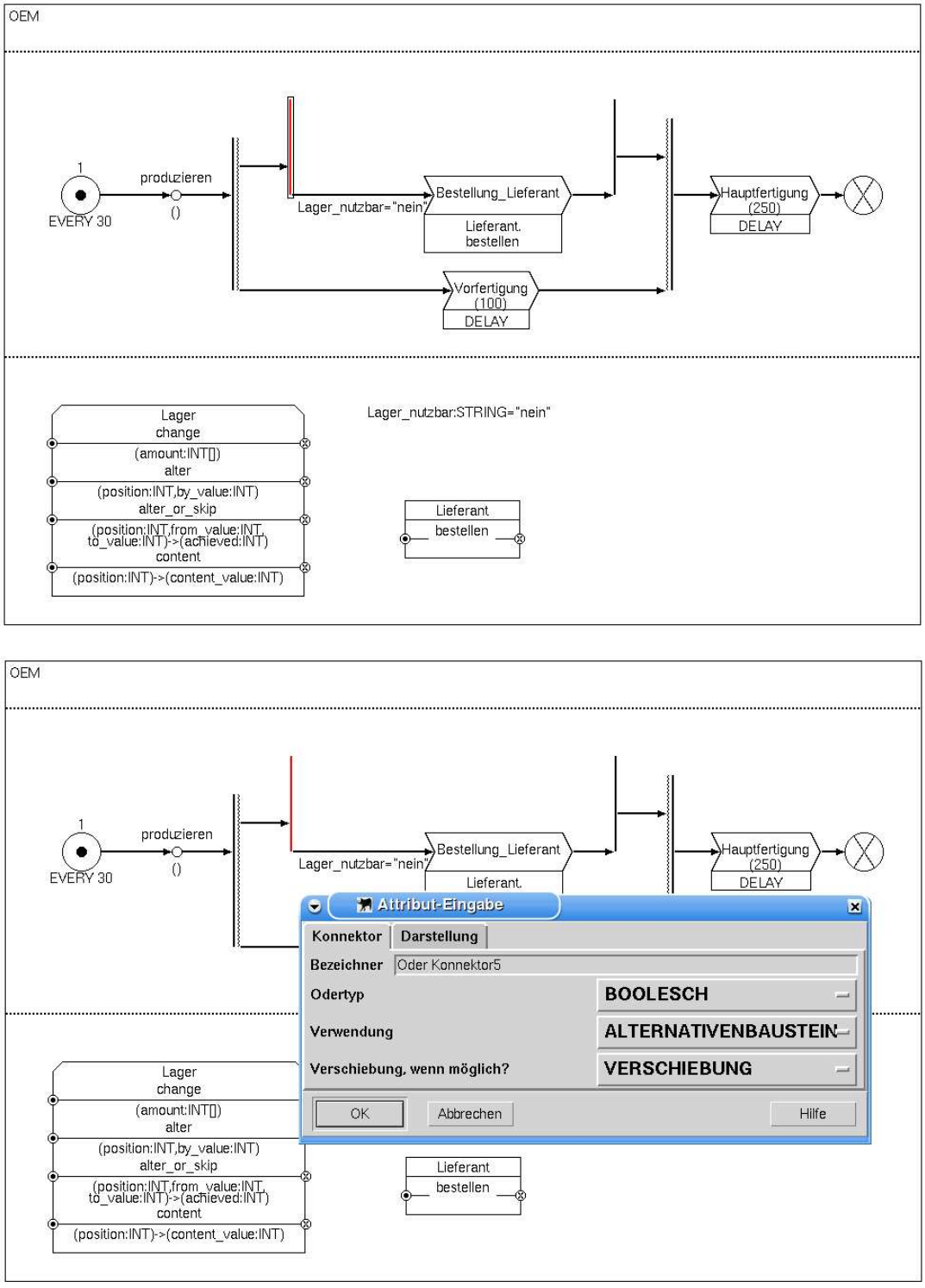


Abbildung 5.7: Der Oder-Konnektor als Alternativenbaustein, Teil 3.
Oben: Ergebnis der Änderung der Variablenbelegung.
Unten: Markieren des Oder-Konnektors, Aufrufen des Attributmenüs und Auswahl des Punktes „VERSCHIEBUNG“. (Quelle: eigene Darstellung)

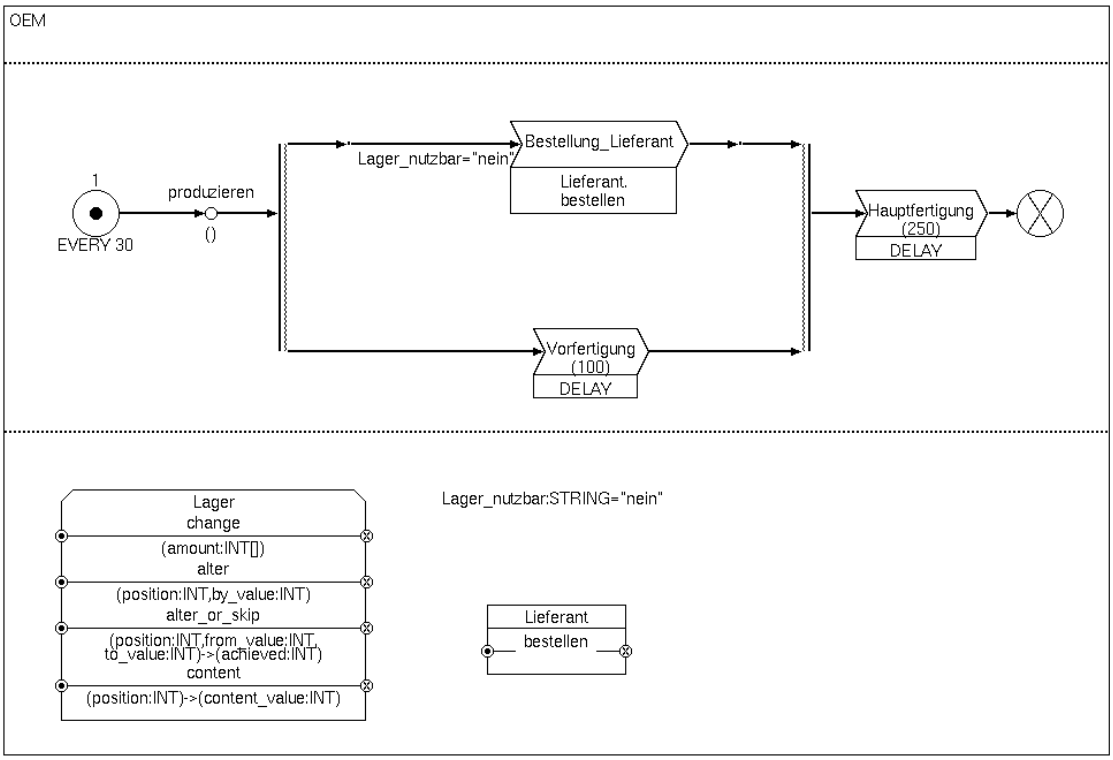


Abbildung 5.8: Der Oder-Konnektor als Alternativenbaustein, Teil 4. Endstruktur: Verwendung des Oder-Konnektors als Alternativenbaustein mit Verschiebung. (Quelle: eigene Darstellung)

„Prozesszuordnung“ können nun von Hand der Prozessname, die Funktionseinheit und der Dienst angegeben werden.[7] Unter dem Prozessnamen wird der Name verstanden, unter dem der importierte Dienst in der Funktionseinheit ansprechbar sein soll. Die Funktionseinheit meint diejenige Funktionseinheit, die den Dienst exportiert, und schließlich bezeichnet Dienst den Prozessnamen des exportierten Dienstes. Ist dies korrekt eingegeben, so müssen je nach Parametern des exportierten Dienstes über zwei Eingabemasken[8] diese Parameter an den importierten Dienst übertragen werden, bis der Dienst endlich genutzt werden kann. Wie sich aus der Beschreibung entnehmen lässt, ist das Anlegen externer Funktionseinheiten gerade für ungeübte Modellierer keine einfache Aufgabe. Um den Modellierer hierbei zu unterstützen, werden zwei Hilfen angeboten.

Bei der Eingabe der Funktionseinheit und des Dienstes, der exportiert werden soll, in der Maske „Prozesszuordnung“ stellt sich die Frage, welche Dienste überhaupt genutzt werden dürfen. Ist diese beantwortet, so müssen die Bezeichnungen exakt übertragen werden, so dass der Modellierer eventuell häufiger zwischen der Attributmaske und der Darstellung des Subgraphen wechseln muss, um Fehler zu vermeiden. Um diesen Vorgang zu vereinfachen und Fehler zu unterbinden,

[7] Vgl. Abbildung 5.9.
[8] Parameter (virtuell) und Ausgabeparameter (virtuell).

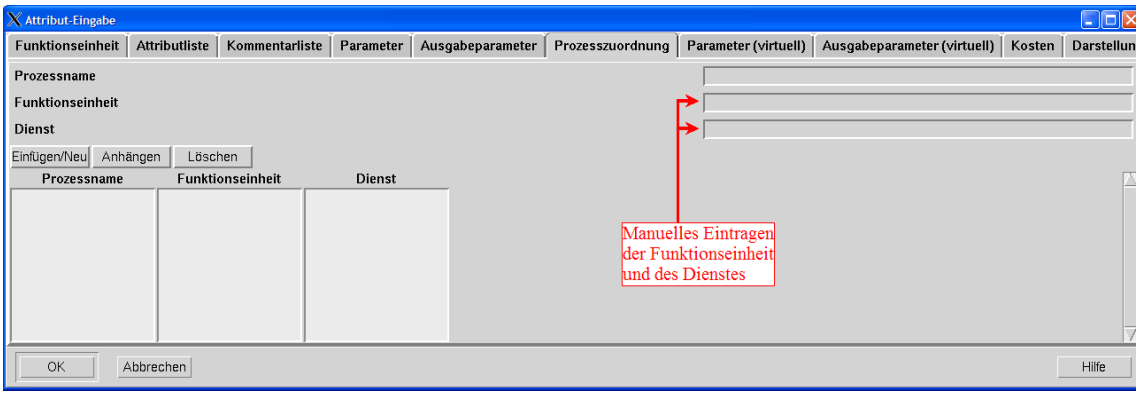


Abbildung 5.9: Bisherige Maske zum Anlegen externer Funktionseinheiten. (Quelle: eigene Darstellung)

erfolgt nun die Eingabe dieser beiden Felder nicht mehr manuell. Der Modellierer kann nun aus einem Menü die gewünschte Funktionseinheit und den zu exportierenden Dienst auswählen. So ist stets die korrekte Schreibweise gesichert. Ebenso wird gewährleistet, dass nur Dienste importiert werden können, die auch tatsächlich genutzt werden dürfen. Abbildung 5.10 stellt die neue Maske dar.

In einer zweiten Änderung wird der gesamte Prozess des Anlegens externer Funktionseinheiten vereinfacht. Der Modellierer muss nicht mehr über die Attributmaske die notwendigen Informationen eingeben. Er kann nun vielmehr in der Aussenansicht der beiden betrachteten Funktionseinheiten über das Kontextmenü den Vorgang durchführen. Dazu wählt er zunächst den zu exportierenden Dienst aus und öffnet das Kontextmenü des damit verbundenen FEQuellePorts. Dort gibt es einen neuen Punkt „Dienst exportieren“. Wird dieser Punkt angeklickt, so wird der Dienst im Editor gemerkt. In diesem Schritt geschieht nichts weiteres. Ist ein Dienst so gemerkt, kann die Funktionseinheit ausgewählt werden, die den Dienst importieren soll. Wird das Kontextmenü geöffnet und kann der Dienst importiert werden[9], so erscheint der Menüpunkt „Dienst ... von ... importieren“, wobei die ersten Punkte durch den Dienstnamen und die zweiten Punkte durch den Namen der Funktionseinheit, die den Dienst anbietet, ersetzt sind. Über die Auswahl dieses Menüpunktes wird der Dienst mit seinen Parametern und Ausgabeparametern importiert. Dabei wird zunächst der gewünschte Prozessname, d. h. der Name, unter dem der importierte Dienst angesprochen werden kann, abgefragt.[10] Wird kein Name vergeben, so wird er automatisch generiert und setzt sich aus dem Namen der Funktionseinheit, die den Dienst exportiert, und dem Dienstnamen zusammen, getrennt durch einen Unterstrich. Wenn der Modellierer Änderungen vornehmen

[9] Dienste können nicht in einen Server, PrioServer, Counter oder Storage importiert werden. Ebenso wird ein Importieren in die Funktionseinheit unterbunden, die den Dienst anbietet.

[10] Vgl. Abbildung 5.11.

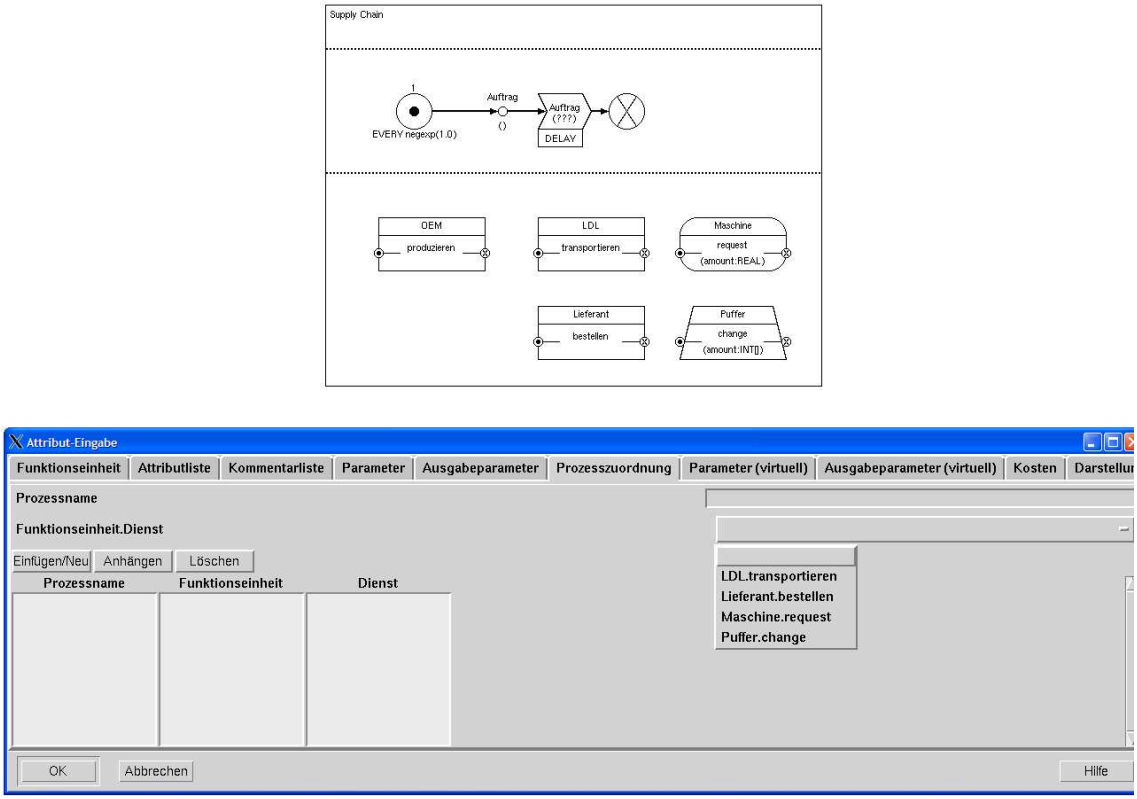


Abbildung 5.10: Anwendung der modifizierte Maske zum Anlegen externer Funktionseinheiten. Zu sehen ist die Attributmaske der Funktionseinheit „OEM“. Im Auswahlmenü stehen alle Dienste aller Funktionseinheiten zur Verfügung, die importiert werden dürfen. (Quelle: eigene Darstellung)

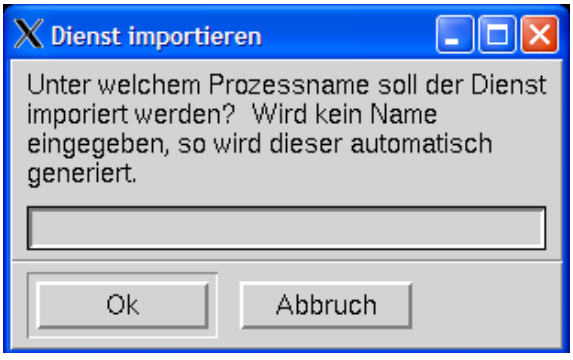


Abbildung 5.11: Möglichkeit zur Eingabe eines Namens für den zu importierenden Dienst. (Quelle: eigene Darstellung)

möchte, so kann dies wie gewohnt über die Attributmaske in den entsprechenden Unterpunkten geschehen.

Abbildungen 5.12 bis 5.14 fassen das Vorgehen grafisch an einem Beispiel zusammen. In diesem Modell möchte der OEM die Dienste „Abfrage_freie_Kapazitaeten“ und „Transportauftrag“ des Logistikdienstleisters LDL in Anspruch nehmen. Der Dienst „Abfrage_freie_Kapazitaeten“ gibt als Ausgabeparameter einen String mit den freien Kapazitäten zurück. Für den Dienst „Transportauftrag“ wird die zu transportierende Menge übergeben. Als Rückgabe erhält der OEM die Rechnung für den Transport. Das Anlegen der externen Funktionseinheit erfolgt auf die eben beschriebene Weise. Die Ergebnisse werden sowohl in der Aussenansicht als auch in den betroffenen Masken dargestellt.

5.2.2 „Durchschleusen“ von Prozessen

Im Rahmen von Supply Chains ist es die Regel, dass die Unternehmen Informationen austauschen und gegenseitig Dienste nutzen. Der dafür notwendige Umgang mit externen Funktionseinheiten wird im vorherigen Abschnitt 5.2.1 beschrieben. Diese ermöglichen die Nutzung von Diensten, die in der exportierenden Funktionseinheit angeboten werden. Dieser Vorgang entspricht der Realität insoweit, dass die Unternehmen zunächst Kontakt auf oberster Ebene knüpfen, d. h. es gibt wohldefinierte Schnittstellen, über die miteinander kommuniziert wird. Hat sich der Kontakt aber gefestigt, so sollte ein direkter Kontakt der beteiligten Abteilungen möglich sein. Um das Ganze zu verdeutlichen, wird es an einem konkreten Beispiel erläutert.

In einer Supply Chain arbeiten ein Lieferant und der OEM zusammen. Der OEM verfügt über eine Produktionsabteilung, die wiederum ein Lager leitet, in dem alle zur Produktion benötigten Teile vorhanden sind. In regelmäßigen Abständen nutzt der OEM den Dienst „Teile_bestellen“ des Lieferanten, um dieses Lager wieder aufzufüllen. Wenn der OEM mit der Qualität der gelieferten Teile zufrieden ist, kann er planen, die Supply Chain-Strategie zu ändern. Denkbar ist ein Umstieg auf ein Vendor Managed Inventory[11], d. h. er überträgt die Verantwortung für den Lagerbestand an den Lieferanten. Damit dieser der Verantwortung gerecht werden kann, muss er z. B. in der Lage sein, den aktuellen Bestand abzufragen. Dieser vom Lager angebotene Dienst kann aber nicht vom Lieferanten genutzt werden, da er „zu tief“ in der Hierarchie des OEM liegt. Damit nun der Lieferant doch auf die Lagerbestandskontrolle zugreifen kann, muss der Dienst schrittweise über die Produktionsabteilung als Dienst des OEM realisiert werden. Sind diese Weiterleitungen erzeugt, kann der Lieferant den neu geschaffenen Dienst des OEM nutzen. Auf Ebene des OEM und der

[11] Vgl. zum Vendor Managed Inventory bspw. Kaczmarek, Völker (2003), S. 16 f..

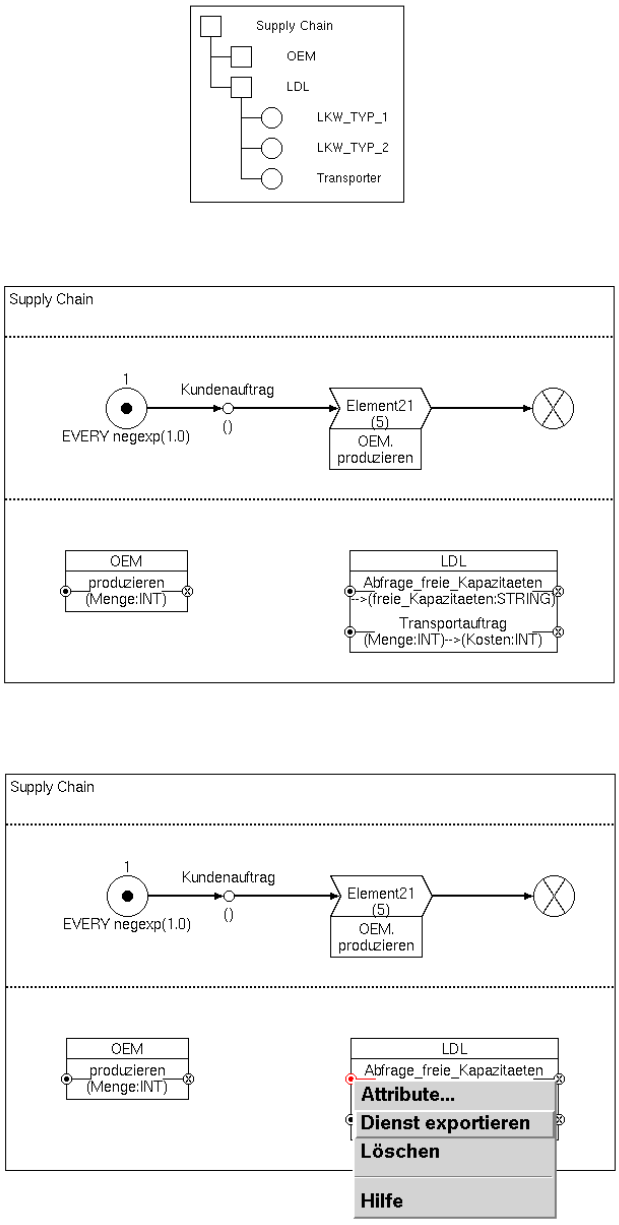


Abbildung 5.12: Vereinfachtes Anlegen externer Funktionseinheiten, Teil 1.
Oben: Hierachiebaum des Modells.
Mitte: Innenansicht des Wurzelknotens „Supply Chain“.
Unten: Markieren des FEQuellePorts des Dienstes „Abfrage_freie_Kapazitaeten“, Aufrufen des Kontextmenüs und Auswahl des Punktes „Dienst exportieren“. (Quelle: eigene Darstellung)

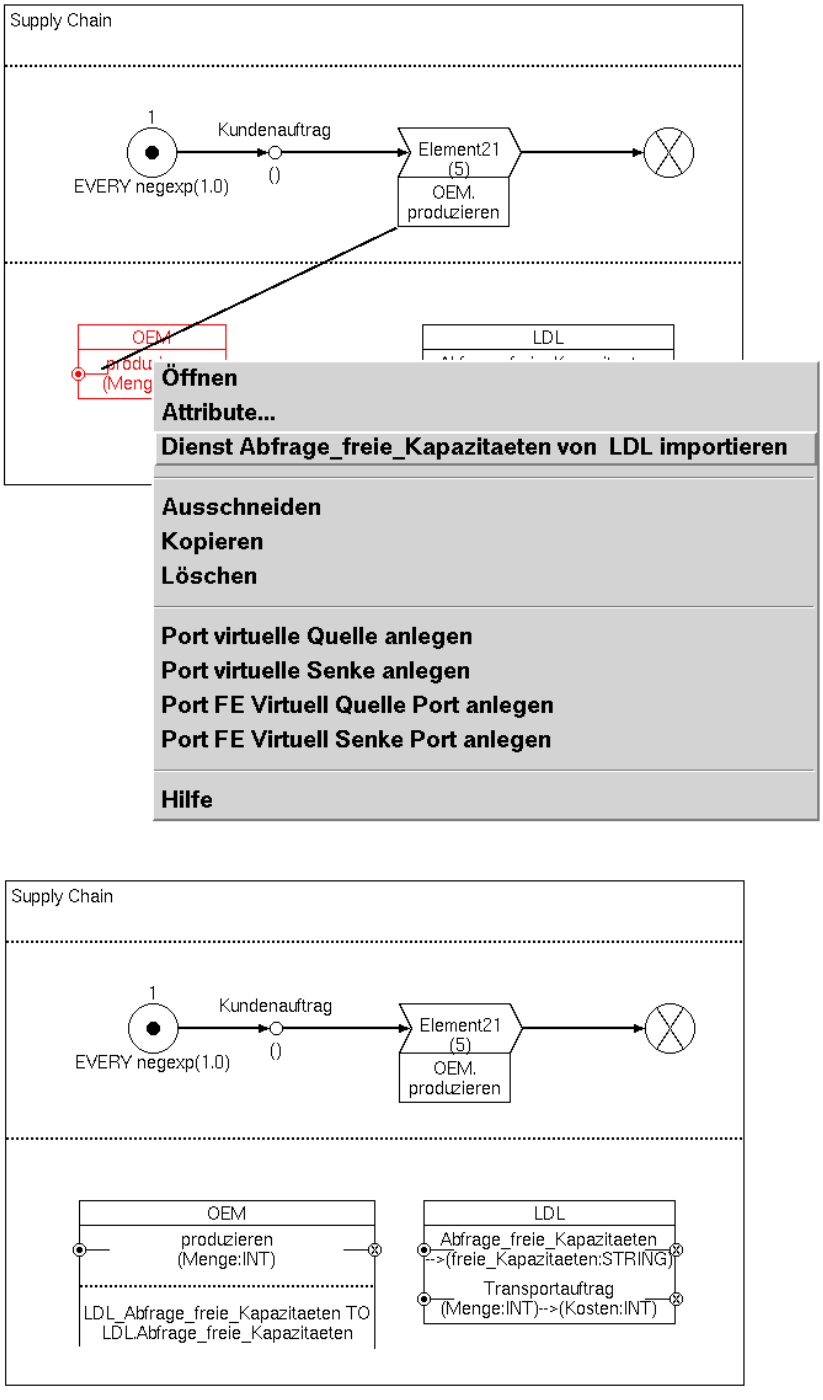


Abbildung 5.13: Vereinfachtes Anlegen externer Funktionseinheiten, Teil 2.
Oben: Aufruf des Punktes „Dienst Abfrage_freie_Kapazitaeten von LDL importieren“ im Kontextmenü der Funktionseinheit OEM.
Unten: Ansicht nach Import des Dienstes. Auf die Vergabe eines eigenen Namens wurde verzichtet, so dass automatisch der Name „LDL_Abfrage_freie_Kapazitaeten“ erzeugt wurde. (Quelle: eigene Darstellung)

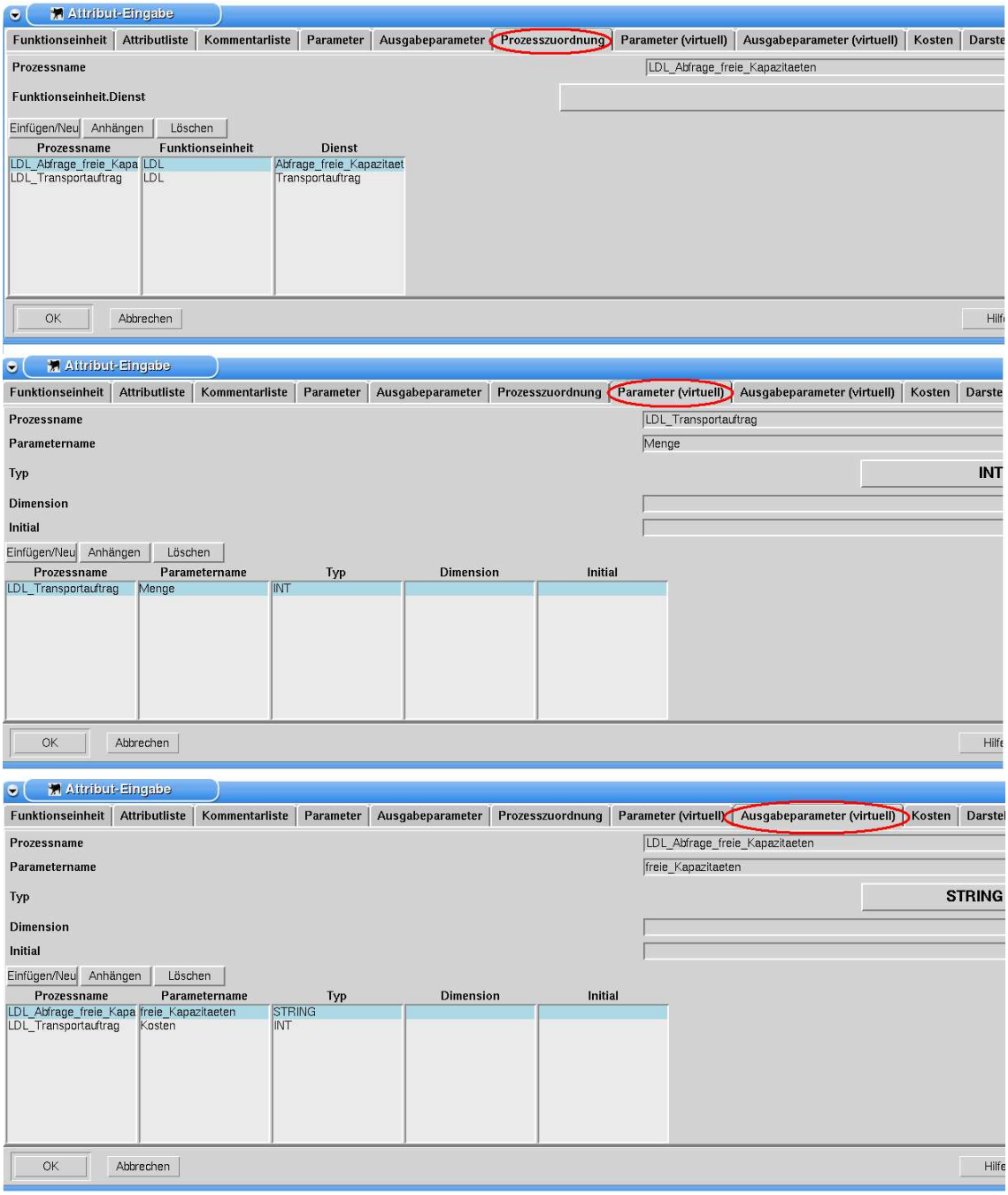


Abbildung 5.14: Vereinfachtes Anlegen externer Funktionseinheiten, Teil 3.
Ansicht der Masken „Prozesszuordnung“, „Parameter (virtuell)“ und „Ausgabeparameter (virtuell)“, nachdem beide Dienste des LDL in den OEM importiert worden sind. Automatisch werden alle Parameter generiert. (Quelle: eigene Darstellung)

Produktionsabteilung erfolgt lediglich eine Weiterleitung des Aufrufes an das Lager.[12]
Durch diese internen Weiterleitungen werden zwei Vorteile erreicht. Zum einen können Dienste genutzt werden, die tiefer in der Hierarchie liegen, d. h. die eigentlich auf der aktuellen Ebene nicht sichtbar sind, zum anderen wird in Verbindung mit externen Funktionseinheiten die Schnittstelle zwischen zwei Funktionseinheiten erweitert.
Das Erzeugen solcher interner Weiterleitungen zum Durchschleusen von Aufrufen ist aufwendig, weil in jeder betroffenen Funktionseinheit eine neue Prozesskette angelegt werden muss, die den Aufruf mit allen Parametern weiterreicht. Hier setzt eine Erweiterung des ProC/B-Editors an, indem der Modellierer von dieser Aufgabe befreit wird. Handelt es sich bei dem Prozess, den er nach oben in der Hierarchie durchschleusen möchte, um einen Prozess eines Servers, PrioServers, Counters oder Storages, erreicht er in der Aussenansicht dieser Funktionseinheiten über das Kontextmenü des verbundenen FEQuellePorts einen Auswahlpunkt „Weiter oben anbieten“.[13] Wählt

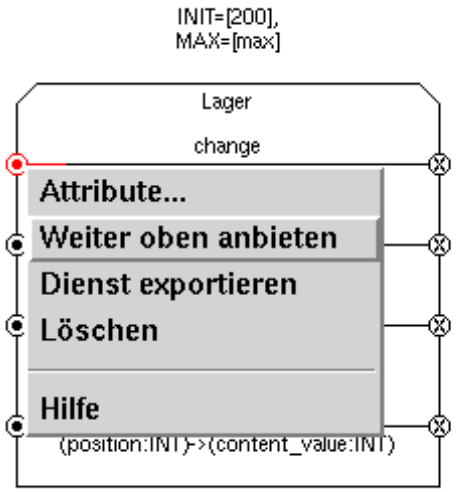


Abbildung 5.15: Modifiziertes Kontextmenü an einem Storage. (Quelle: eigene Darstellung)

er diesen Punkt aus, so erscheint eine Liste aller Funktionseinheiten, in denen der Dienst angeboten werden kann. Hat er sich für eine Funktionseinheit entschieden, werden die Weiterleitungen mit allen Parametern automatisch generiert. Dazu ist anzumerken, dass die Weiterleitungen nur bis zu der Funktionseinheit angelegt werden, die direkt in der Hierarchie unterhalb der gewählten Funktionseinheit liegt. Durch dieses Vorgehen ist der Dienst in der gewählten Funktionseinheit nutzbar, d. h. er wird zur Nutzung angeboten. Übertragen auf die Verwendung externer Funktionseinheiten bedeutet dies, dass der Dienst immer bis zu der Funktionseinheit angeboten werden muss, die auch die beiden Funktionseinheiten enthält, die ihre Dienste ex- bzw. importieren möch-

[12] Das Beispiel des Vendor Managed Inventory wird in Kapitel 6.1 erneut aufgegriffen.
[13] Vgl. Abbildung 5.15.

ten. Im Beispiel des OEM und des Lieferanten ist das die Funktionseinheit „Supply Chain“. Ebenso ist ein Durchschleusen von Prozessen von konstruierten Funktionseinheiten möglich. Das Vorgehen unterscheidet sich von dem geschilderten Vorgehen nur insoweit, dass der Auswahlpunkt „Weiter oben anbieten“ in der Innenansicht über den entsprechenden Knoten „ProzessID“ erreichbar ist. Abbildung 5.16 zeigt das modifizierte Kontextmenü an einem solchen Knoten an einem Beispiel. In diesem Modell existiert ein LDL als abhängiges Unternehmen eines Großspediteurs. Der Dienst „Abfrage_freie_Kapazitaeten“ des LDL soll dem OEM zugänglich gemacht werden. Dafür muss er auf Ebene des Großspediteurs realisiert werden, d. h. er muss in der Supply Chain nutzbar sein. Ist dies der Fall, kann der OEM den Dienst auch importieren.[14]

Bei der Umsetzung dieser Idee entstand die Frage, wie die Weiterleitungen dargestellt werden

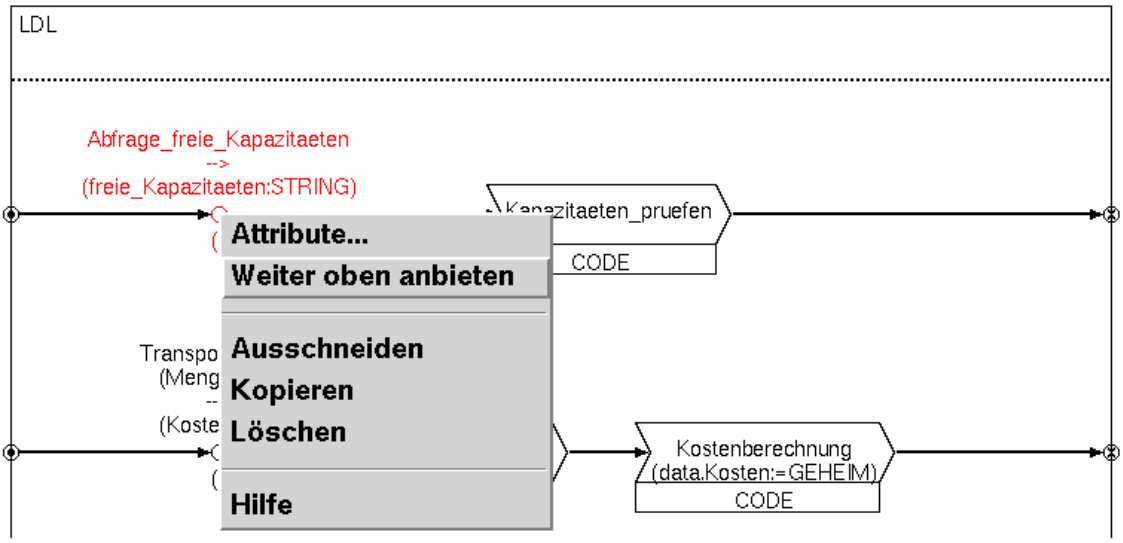


Abbildung 5.16: Beispiel zum Durchschleusen von Prozessen, Teil 1.
Innenansicht des LDL mit aufgerufenem Kontextmenü des Knotens „Abfrage_freie_Kapazitaeten“. (Quelle: eigene Darstellung)

sollen. Es wurde überlegt, die Weiterleitungen zu verstecken, indem sie nicht dargestellt werden, und die Funktionseinheit, dessen Dienst durchgeschleust wird, in der gewählten Funktionseinheit mit einer speziellen Kennzeichnung darzustellen. Von beiden Überlegungen wurde Abstand genommen, da der Modellierer, wenn er es möchte, die Weiterleitungsprozesse von Hand ausblenden kann (siehe Kapitel 5.1), und ein zusätzliches Einblenden der Funktionseinheit, wenn auch speziell gekennzeichnet, eine falsche Struktur suggerieren würde. Stattdessen wurde der jeweilige Prozessname auf jeder Stufe der Weiterleitung so gewählt, dass der Modellierer den Weg der Weiterleitung schon anhand des Namens erkennen kann.[15] Durch diese Namensgebung kann der

[14] Vgl. zu diesem Beispiel Abbildungen 5.16 bis 5.18. Sie stellen einen Auszug aus der in Kapitel 6.3 geschilderten Übernahme von Unternehmen dar.
[15] Vgl. das Beispiel in Kapitel 6.1.

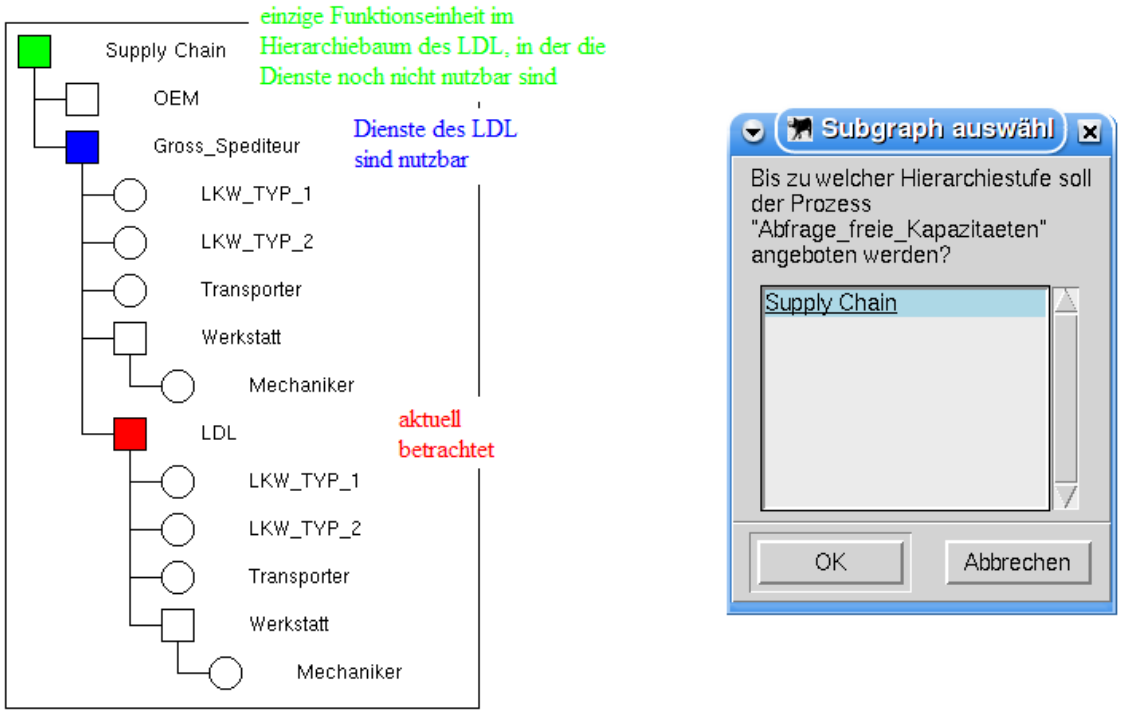


Abbildung 5.17: Beispiel zum Durchschleusen von Prozessen, Teil 2.
Zugrunde liegender Hierarchiebaum und sich daraus ergebende Abfrage nach Aufruf des Punktes „Weiter oben anbieten“ in Abbildung 5.16. (Quelle: eigene Darstellung)

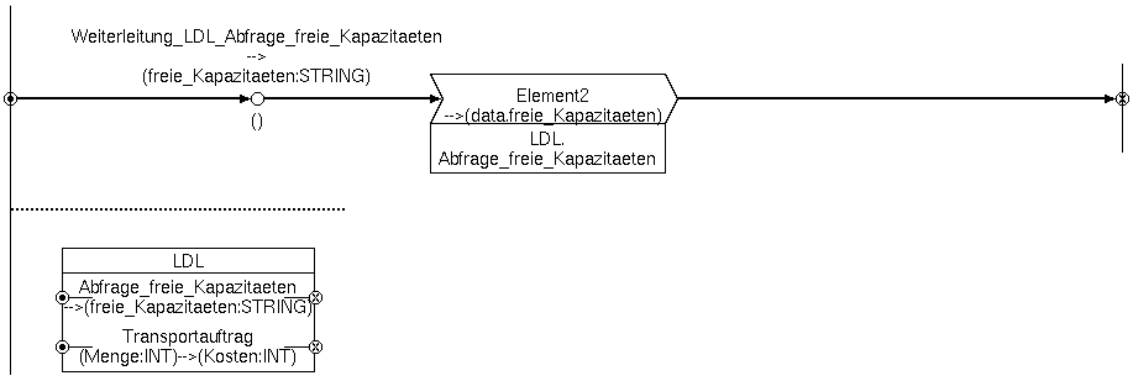


Abbildung 5.18: Beispiel zum Durchschleusen von Prozessen, Teil 3.
Neu erzeugter Prozess als Weiterleitung im Groß_Spediteur. (Quelle: eigene Darstellung)

ursprüngliche Dienst genau lokalisiert werden, so dass eine gewisse Übersichtlichkeit gewährleistet ist. Nachteilig wirkt sich jedoch aus, dass die Namen schnell sehr lang werden. In diesem Fall bleibt es dem Modellierer überlassen, durch eine eigene Vergabe der Prozessnamen die für ihn beste Lösung umzusetzen.

5.3 Veränderungen der Strukturen

Supply Chain-Strukturen sind häufigen Änderungen unterworfen. Die Zusammensetzung der beteiligten Akteure ändert sich, neue Aufgaben kommen hinzu, alte Prozesse entfallen etc.. Aber auch die Aufbauorganisation eines einzelnen Unternehmens kann sich im Zeitverlauf verändern. Abteilungen fallen weg, werden zusammengelegt oder neu geschaffen, Mitarbeiter, Teams und deren Aufgaben werden anderen Abteilungen zugeordnet, durch In- bzw. Outsourcing werden Dienste in das Unternehmen integriert bzw. aus dem Unternehmen ausgelagert, usw. Solche Veränderungen müssen auch bei der Modellierung berücksichtigt werden, um die Modelle den aktuellen Gegebenheiten schnell anpassen zu können.
Um eine Vorstellung einer solchen Veränderung zu bekommen, soll zunächst ein Beispiel angeführt werden. Abbildung 5.19 zeigt einen Auszug aus einer Supply Chain. Im OEM stehen strukturelle Veränderungen an. Die Abteilung „Werkstatt“ soll geschlossen werden. Da der Meister nicht entlassen werden kann, soll er in den AbsatzOEM versetzt werden. Entfallen die Dienste der Werkstatt, so müssen die Abteilungen „Teilefertigung“, die bisher direkt auf die Dienste der Werkstatt zugreifen konnte, und „Lager“, das Dienste der Werkstatt importiert hat, angepasst werden. Bei der bisherigen Modellierung muss die Versetzung des Meisters explizit durchgeführt werden, da ein Entfernen einer Funktionseinheit alle unterliegenden Elemente auch löscht. Dazu muss zunächst die Funktionseinheit „Werkstatt“ geöffnet werden, um dort den „Meister“ zu kopieren oder auszuschneiden. Anschließend muss die Funktionseinheit „AbsatzOEM“ geöffnet werden. Dort kann dann der „Meister“ eingefügt werden. Ähnliches gilt neben dem Personal auch für andere Ressourcen wie Maschinen, die in der Realität entweder veräußert (→ löschen) oder in anderen Abteilungen eingesetzt werden (→ verschieben wie beim „Meister“). Ebenso kann der Fall auftreten, dass Funktionseinheiten gelöscht werden und ihre Unterabteilungen eine Hierarchieebene aufsteigen sollen. In dem konkreten Beispiel bedeutet dies, dass die „Werkstatt“ entfällt und der „Meister“ ihre Position einnimmt, d. h. direkt der „Teilefertigung“ unterstellt wird. Ein anderes Beispiel aus der Praxis ist Lean Management, bei dem die mittleren Hierarchieebenen entfallen und die Aufgaben direkt den unteren Hierarchieebenen übertragen werden.
Wird eine Funktionseinheit entfernt, ändern sich eventuell Dienstanbindungen. Eine übergeordne-

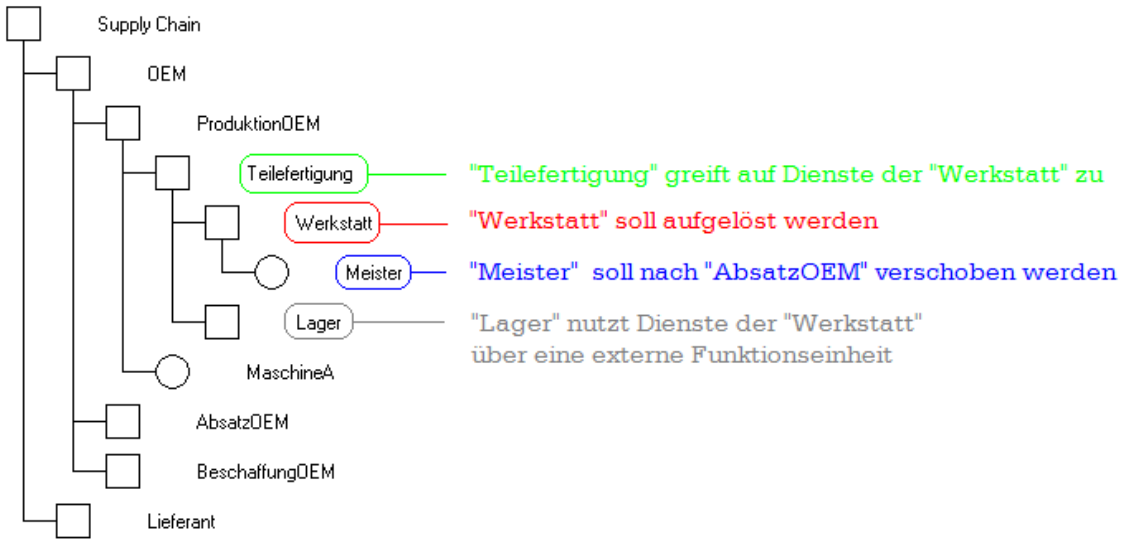


Abbildung 5.19: Auszug aus einer Supply Chain, die Änderungen unterworfen ist. (Quelle: eigene Darstellung)

te Funktionseinheit kann die Dienste direkt nutzen (im Beispiel ist das die „Teilefertigung“) und Funktionseinheiten derselben Hierarchieebene können über externe Funktionseinheiten auf die Dienste zugreifen. Bisher werden solche verlorenen Dienstanbindungen im ProC/B-Editor nicht speziell gekennzeichnet, so dass der Modellierer gezwungen ist, in den betroffenen Funktionseinheiten alle Elemente zu überprüfen.

Um die Durchführung dieser Strukturveränderungen zu erleichtern, werden einige Modifikationen im Editor vorgenommen. Dazu wird ein Vervielfachen, Verschieben und Löschen im Hierachiebaum ermöglicht. Die Verschiebung dieser Operationen aus den Funktionseinheiten in den Hierachiebaum bietet den Vorteil, dass dieser die größte Übersicht über das Modell bietet und alle Aktionen in einem Fenster ausgeführt werden können. Schrittweise werden nun die einzelnen Operationen vorgestellt.

5.3.1 Vervielfachen

In einem ersten Schritt wird in das Kontextmenü jedes Knotens des Hierachiebaums ein Auswahlpunkt „FE vervielfachen“ eingebunden. Wird dieser Punkt ausgewählt, erfolgt eine Abfrage, wie oft der Knoten kopiert werden soll. Wird ein ganzzahliger positiver Wert eingegeben, werden entsprechend viele Kopien des Knotens in dem Subgraphen angelegt, in dem auch der Knoten zu finden ist. Anders gesagt, die Kopien werden auf derselben Hierachieebene wie das Original angelegt. Wird kein ganzzahliger positiver Wert eingetragen. erfolgt ein Abbruch der Aktion mit einer entsprechenden Fehlermeldung. Der Modellierer kann nach Festlegung der Anzahl wählen,

ob er selbst Namen für die Kopien vergeben möchte. Ist dies der Fall, wird er nach einem Namen für jede Kopie befragt. Dabei wird auf eine leere Eingabe sowie bereits in der Funktionseinheit vergebene Namen geprüft. In beiden Situationen erfolgt eine Warnmeldung mit der Aufforderung, einen anderen Namen zu wählen. Möchte der Modellierer keine eigenen Namen eingeben, so werden diese automatisch generiert, indem der ursprüngliche Name um einen Unterstrich und eine Zahl ergänzt wird. Werden Dienste des Originals mittels externer Funktionseinheiten in anderen Funktionseinheiten genutzt, so erfolgt eine weitere Abfrage, ob die Dienste der Kopien ebenfalls importiert werden sollen. Bei der Beantwortung dieser Frage sind zwei Alternativen denkbar. Sind die Kopien angelegt worden, um eine Situation zu modellieren, in der mehrere einheitliche Funktionseinheiten existieren, so macht ein Importieren Sinn. Dies ist z. B. der Fall, wenn ein Pool von Logistikdienstleistern besteht, von denen jeder einen Dienst „transportieren“ anbietet. Nutzt ein OEM alle Dienstleister, so muss er sämtliche Dienste „transportieren“ importieren. Wird nun ein Muster-Logistikdienstleister erzeugt, dessen Dienst importiert wird, so kann über ein Vervielfachen schnell die gewünschte Situation hergestellt werden. Der andere Fall tritt dann auf, wenn ein Vervielfachen durchgeführt wird und anschließend die Kopie(n) verschoben werden soll(en). Dabei wird ein Importieren der Dienste überflüssig sein.
Abbildungen 5.20 bis 5.23 stellen die Schritte des Vervielfachens grafisch an einem Beispiel dar.

5.3.2 Verschieben und Einfügen

Dem Modellierer wird die Möglichkeit eingeräumt, Knoten im Hierarchiebaum zu verschieben. Dieser Vorgang setzt sich aus zwei Teilschritten zusammen. Zunächst muss im Kontextmenü des Knotens, der verschoben werden soll, der Punkt „Knoten verschieben“ ausgewählt werden. Geschieht dies, wird intern die Identität dieses Knotens bestimmt. Es passieren keine weiteren sichtbaren Änderungen. In einem zweiten Schritt öffnet der Modellierer das Kontextmenü des Knotens, in den eingefügt werden soll. Ist ein Einfügen des vermerkten Knotens möglich[16], erscheint ein Punkt „... einfügen“, wobei „...“ für den Namen des einzufügenden Knoten steht. Durch Auswahl des Punktes beginnt der Verschiebe-Vorgang. Der Knoten wird von seiner ursprünglichen Position im Hierarchiebaum gelöscht. Da dadurch seine Dienste dort nicht mehr verfügbar sind, werden im Fall, dass Elemente im Vaterknoten auf diese zugegriffen haben oder gleich gestellte Funktionseinheiten Dienste importiert haben, entsprechende Warnmeldungen ausgegeben und die betroffenen Elemente farblich markiert, so dass der Modellierer die gelöschten Dienstanbindungen sofort er-

[16] Ein Einfügen ist nicht möglich, wenn der Zielknoten im zu verschiebenen Knoten liegt. Dies ist z. B. in Abbilbung 5.19 der Fall, wenn die „Teilefertigung“ in die „Werkstatt“ verschoben werden soll. Dieses Verbot ergibt sich aus der Tatsache, dass der zum Verschieben ausgewählte Knoten mit allen in ihm enthaltenen Knoten verschoben wird, im Beispiel also die „Teilefertigung“ inklusive der „Werkstatt“ und des „Meisters“.

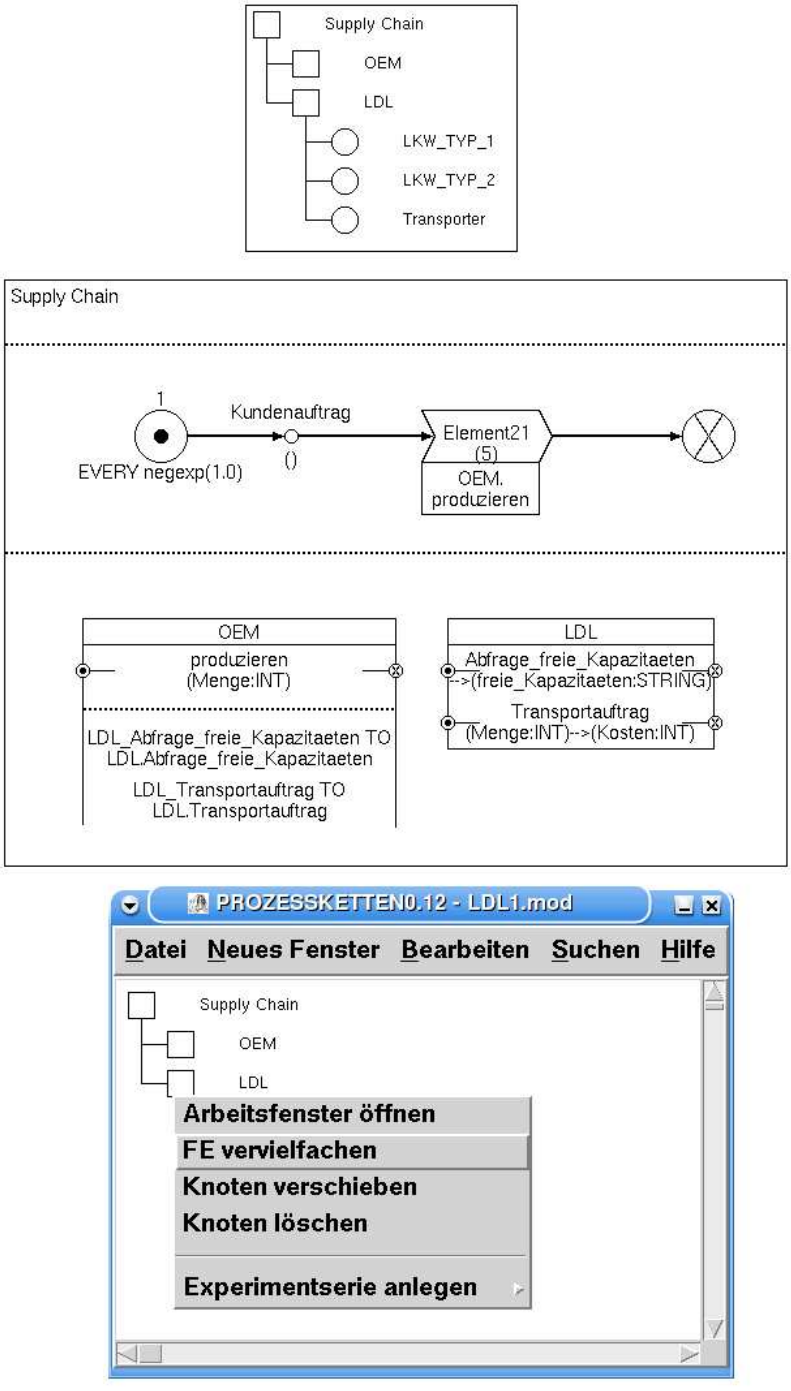


Abbildung 5.20: Vervielfachen von Funktionseinheiten, Teil 1.
Oben: Hierachiebaum des Modells.
Mitte: Innenansicht des Wurzelknotens „Supply Chain“.
Unten: Aufruf des Punktes „FE vervielfachen“ im Kontextmenü der Funktionseinheit LDL. (Quelle: eigene Darstellung)

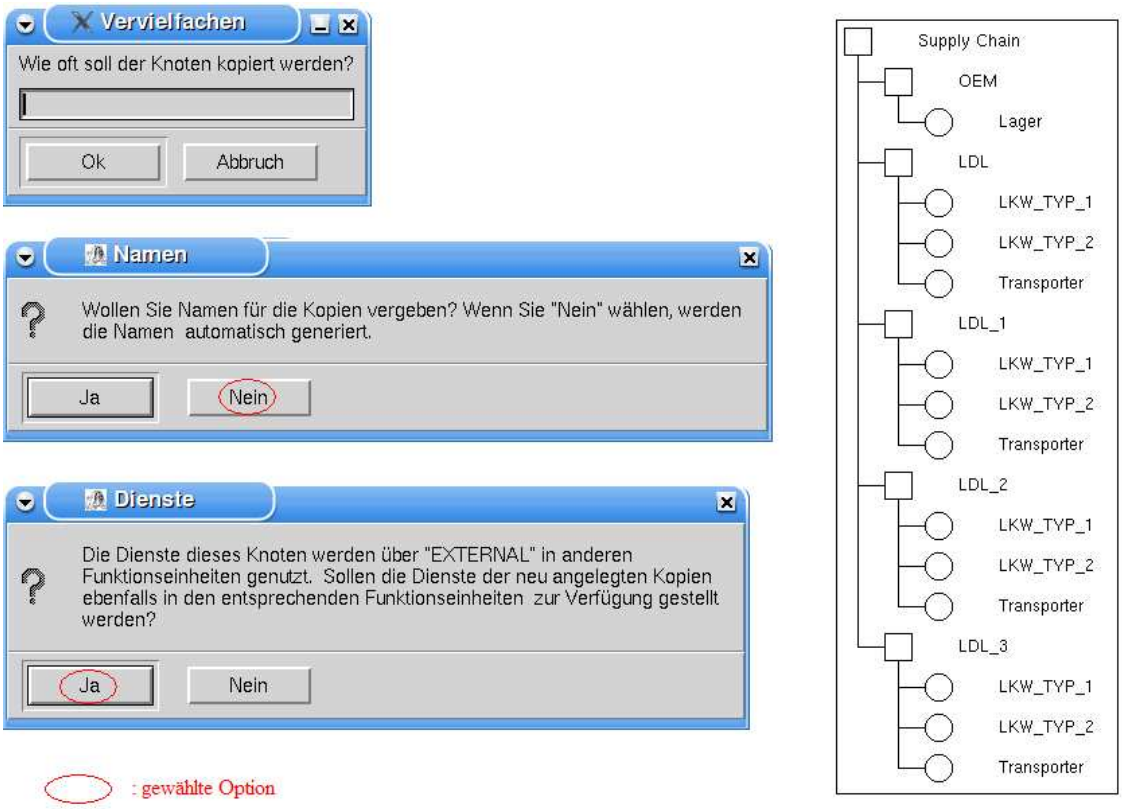


Abbildung 5.21: Vervielfachen von Funktionseinheiten, Teil 2.
Links: Abfragen.
Rechts: Ergebnis nach dreifacher Vervielfältigung mit den gewählten Optionen. (Quelle: eigene Darstellung)

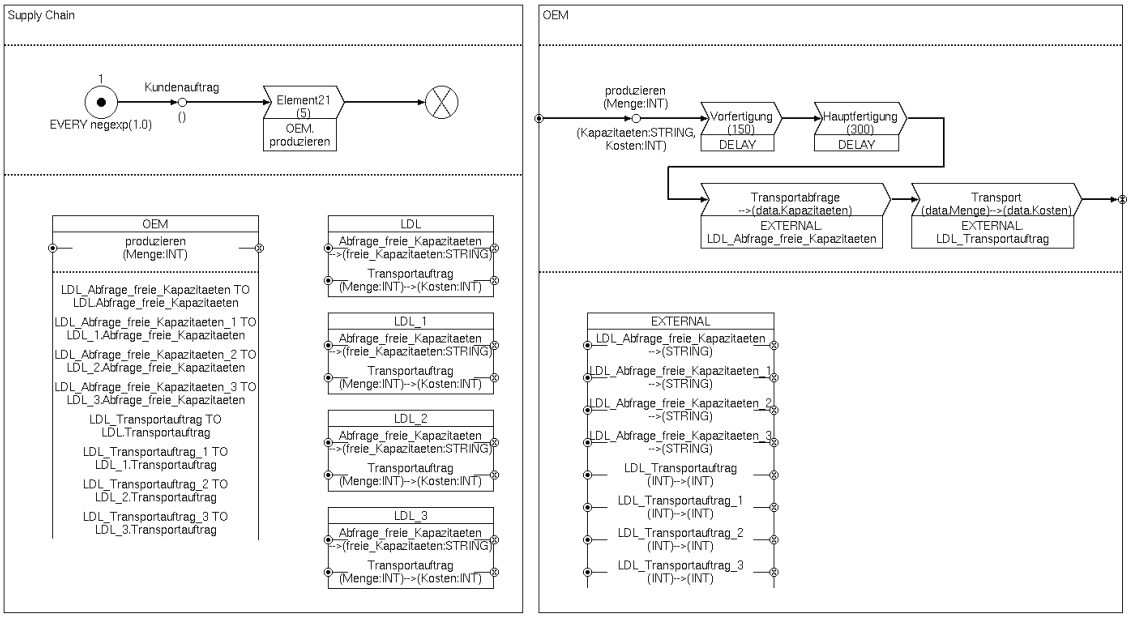


Abbildung 5.22: Vervielfachen von Funktionseinheiten, Teil 3.
Links: Innenansicht der Funktionseinheit Supply Chain nach Vervielfältigung.
Rechts: Innenansicht der Funktionseinheit OEM, wenn die Dienste ebenfalls importiert wurden. (Quelle: eigene Darstellung)

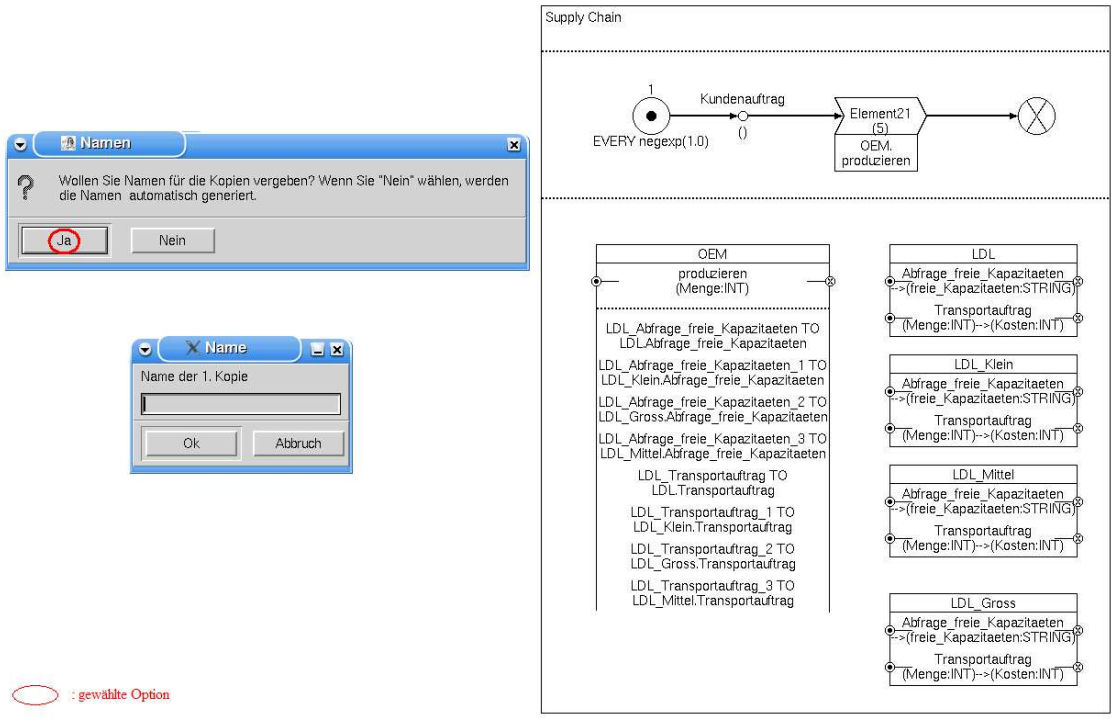


Abbildung 5.23: Vervielfachen von Funktionseinheiten, Teil 4.
Links: Abfragen zur Vergabe eigener Namen für die Kopien.
Rechts: Innenansicht der Funktionseinheit Supply Chain nach Vervielfältigung mit eigener Namensvergabe. (Quelle: eigene Darstellung)

kennen kann. Anschließend wird der Knoten in die ausgewählte Funktionseinheit kopiert.

Das Vorgehen des Verschiebens und Einfügens wird an einem Beispiel vorgeführt.[17] Abbildung 5.24 zeigt die zugrunde liegende Modellstruktur. Ebenso sind die Innenansichten der Funktionseinheiten Supply Chain und OEM zu sehen. Beide nutzen die Dienste des LDL, wobei in der Supply Chain direkt auf sie zugegriffen werden kann, da der LDL eine enthaltene Funktionseinheit ist, während der Zugriff im OEM über eine externe Funktionseinheit realisiert ist.

Abbildung 5.25 fasst den Vorgang des Verschiebens zusammen. Im Kontextmenü des LDL im Hierarchiebaum wird der Punkt „Knoten verschieben“ ausgewählt. Anschließend entsteht im Kontextmenü des Groß_Spediteurs ein Punkt „LDL einfügen“, der ebenfalls ausgewählt wird.

Es folgen Warnmeldungen, dass Dienstanbindungen sowohl im OEM als auch in der Supply Chain ungültig geworden sind, da durch das Verschieben des LDL seine Dienste in den beiden Funktionseinheiten nicht mehr verfügbar sind. Es bleibt dem Modellierer überlassen, dies zu überprüfen.[18]

Abschließend präsentiert Abbildung 5.27 den neuen Hierarchiebaum und die Innenansichten der Supply Chain und des OEM. In beiden Funktionseinheiten sind diejenigen Elemente rot markiert,

[17] Dieses Beispiel stellt einen Auszug aus der in Kapitel 6.3 geschilderten Übernahme von Unternehmen dar.
[18] Vgl. Abbildung 5.26.

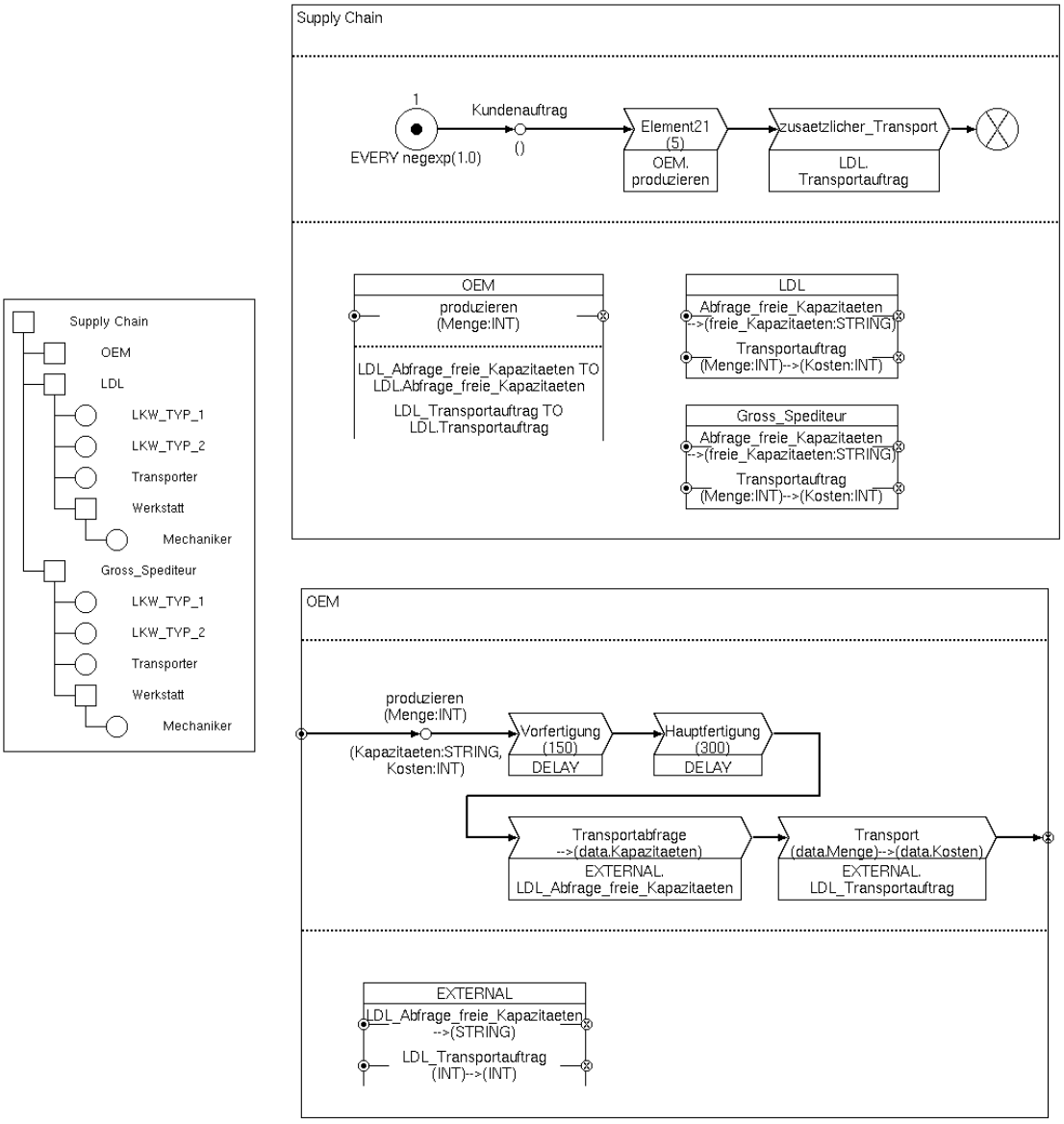


Abbildung 5.24: Verschieben von Funktionseinheiten, Teil 1.
Links: Hierarchiebaum.
Rechts: Innenansicht der Funktionseinheiten Supply und Chain OEM. Beide nutzen die Dienste des LDL. (Quelle: eigene Darstellung)

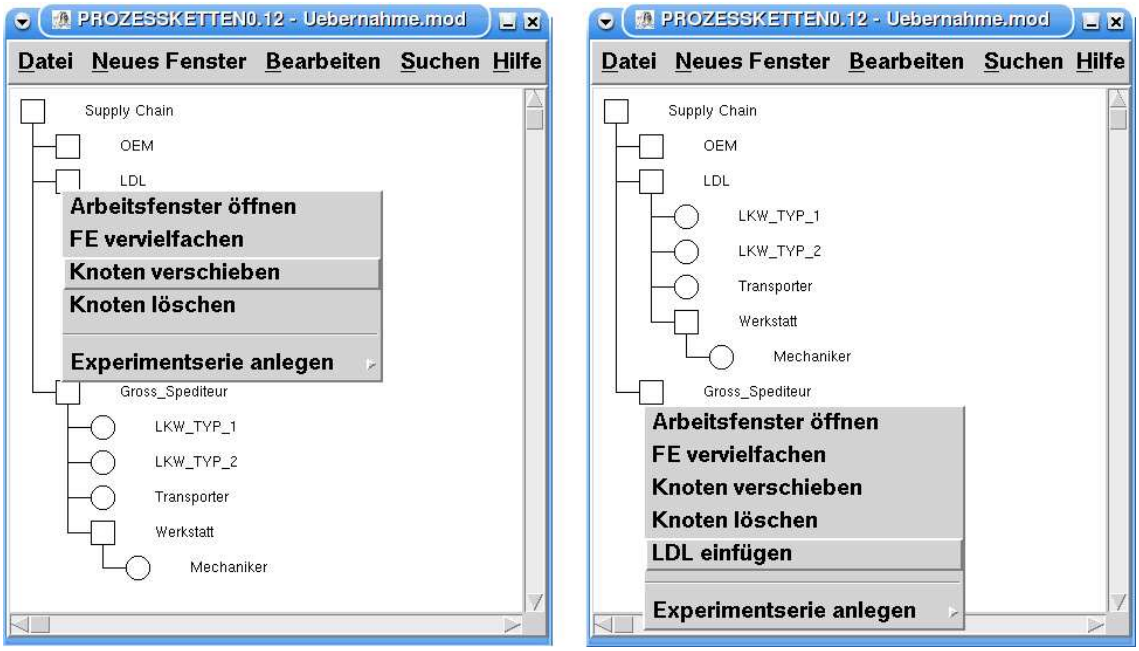


Abbildung 5.25: Verschieben von Funktionseinheiten, Teil 2.
Links: Verschieben.
Rechts: Einfügen. (Quelle: eigene Darstellung)

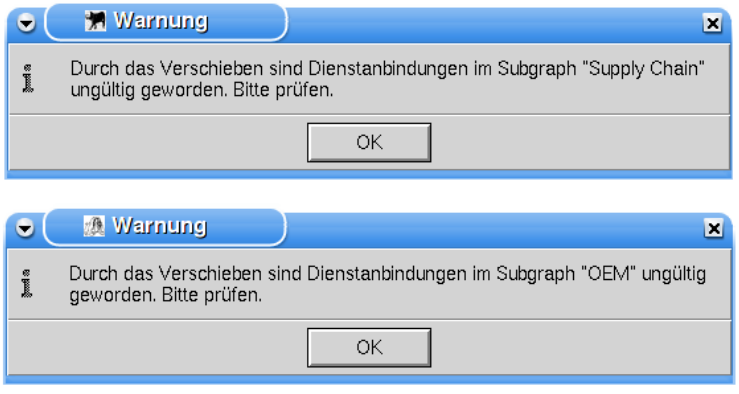


Abbildung 5.26: Verschieben von Funktionseinheiten, Teil 3.
Warnmeldungen. (Quelle: eigene Darstellung)

die bisher die Dienste des LDL genutzt haben. Um die Markierung zu entfernen, kann der Modellierer eine neue Dienstanbindung anlegen, einen Delay-Wert eingeben oder das Element als Codeelement verwenden.

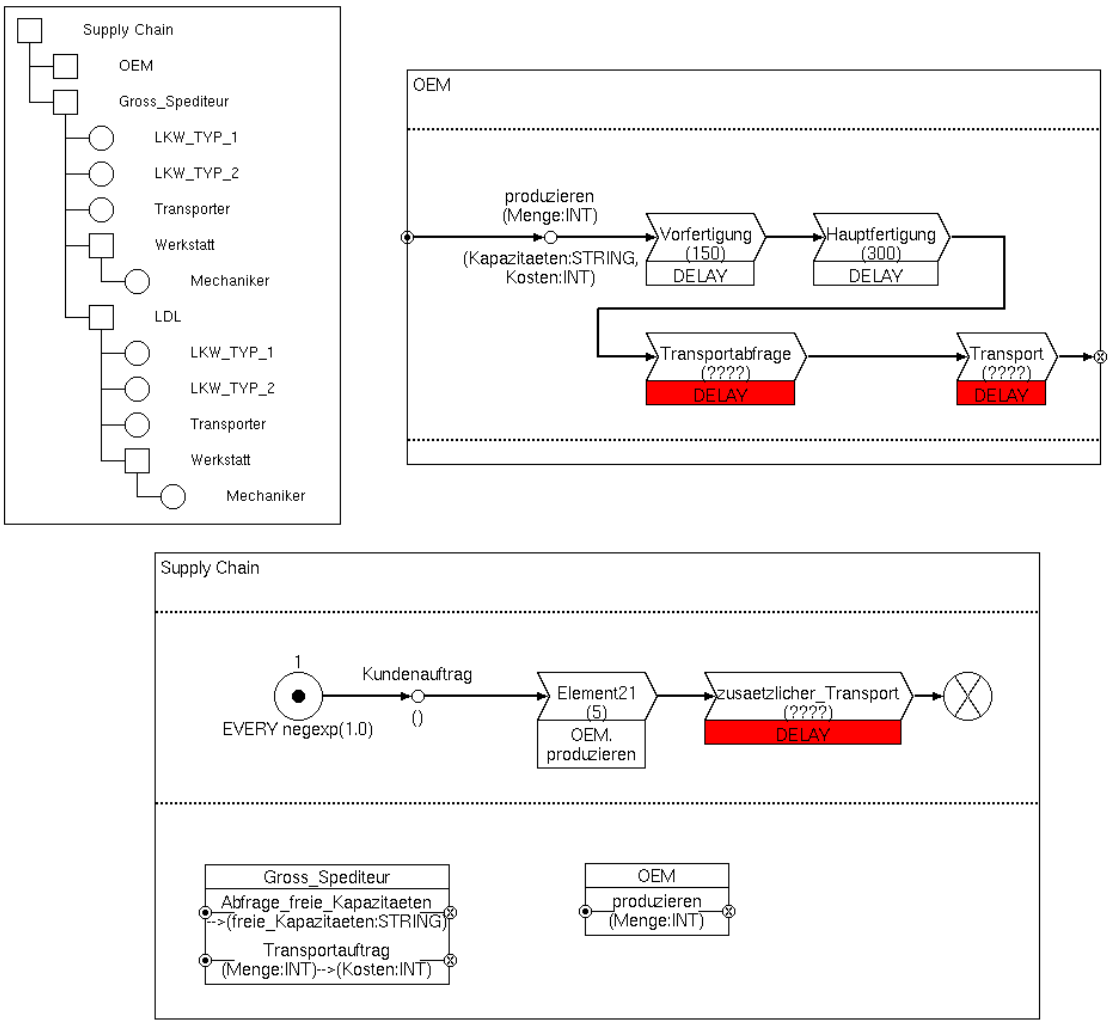


Abbildung 5.27: Verschieben von Funktionseinheiten, Teil 4.
Links oben: Neuer Hierarchiebaum.
Rechts oben: Innenansicht des OEM mit markierten Elementen.
Unten: Innenansicht der Supply Chain mit markierten Elementen. (Quelle: eigene Darstellung)

5.3.3 Löschen

Im Gegensatz zu einem Löschen einer Funktionseinheit in der Innenansicht des Vaterknotens, bei dem auch sämtliche Funktionseinheiten, die in der zu löschenden Funktionseinheit enthalten sind, mit entfernt werden, erfährt das Löschen aus dem Hierarchiebaum eine Modifikation. Vor dem Löschen wird für alle Knoten, die von dieser Aktion direkt betroffen sind, abgefragt, ob diese

Abbildung 5.28: Löschen von Funktionseinheiten, Teil 1.
Aufruf des Punktes „Knoten löschen“ und anschließende Sicherheitsabfrage. (Quelle: eigene Darstellung)

ebenfalls entfernt werden sollen. Wird „Nein“ ausgewählt, so wird der Knoten auf die Hierarchieebene des Vaterknotens verschoben. Dieser Vorgang tritt in der Realität z. B. dann auf, wenn Abteilungen entfallen und die Mitarbeiter in die übergeordnete Abteilung versetzt werden. Wie bereits beim Verschieben erläutert, so werden auch hier Warnungen ausgegeben, wenn Dienstanbindungen ungültig werden. Betroffene Elemente werden markiert.

Auch das Löschen wird exemplarisch dargestellt in Abbildungen 5.28 bis 5.30. Grundlage ist das Modell aus Abbildung 5.24.[19] Hier soll der LDL nicht verschoben sondern gelöscht werden. Dafür wird in seinem Kontextmenü der Punkt „Knoten löschen“ ausgewählt, worauf eine zusätzliche Sicherheitsabfrage gestellt wird, um ein unbeabsichtigtes Löschen zu verhindern.[20] Wird diese mit „Ja“ beantwortet, wird für jede enthaltene Funktionseinheit einzeln abgefragt, ob diese ebenfalls gelöscht werden soll. Wenn dies nicht der Fall sein soll, wird die entsprechende Funktionseinheit in den Vaterknoten des zu löschenden Knotens verschoben, in diesem Fall also in die Supply Chain. Im Beispiel werden die Werkstatt und der LKW_TYP_2 nicht entfernt.[21] Da die Supply Chain und der OEM die Dienste des LDL bisher genutzt haben, folgen wie bereits für das Ver-

[19] Da der Groß_Spediteur nicht relevant für dieses Beispiel ist, wurde er entfernt.
[20] Vgl. Abbildung 5.28.
[21] Vgl. Abbildung 5.29.

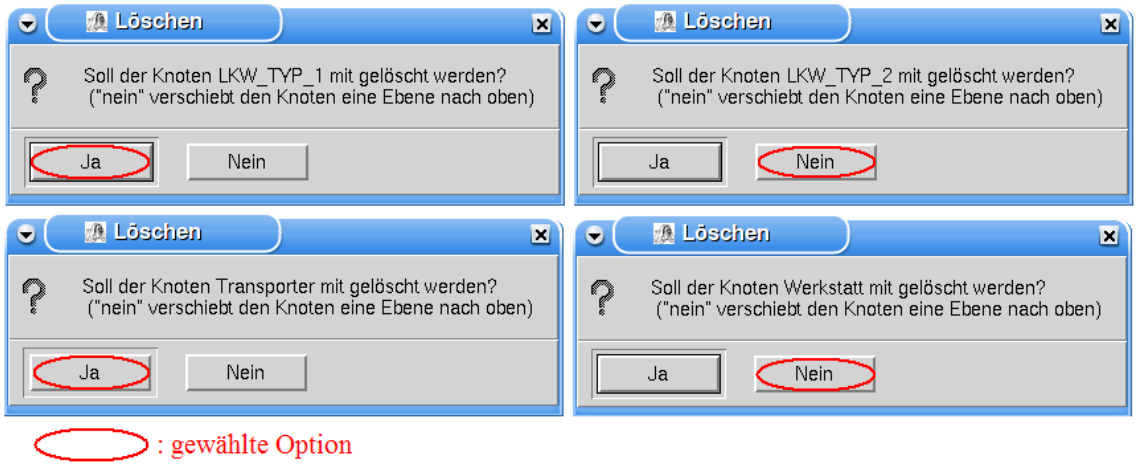


Abbildung 5.29: Löschen von Funktionseinheiten, Teil 2.
Abfragen zum Umgang mit enthaltenen Funktionseinheiten. (Quelle: eigene Darstellung)

schieben in Abbildung 5.26 dargestellt Warnmeldungen. Die sich ergebende Struktur ebenso wie die neuen Innenansichten der Supply Chain und des OEM zeigt Abbildung 5.30.

Diese Änderungsmöglichkeiten im Hierarchiebaum vereinfachen die Umstrukturierung von Modellen. Sind bisher viele Schritte notwendig gewesen, die in exakter Reihenfolge durchgeführt werden mussten[22], so wird der Modellierer jetzt durch Abfragen, Warnungen und Markierungen besser unterstützt.

5.4 Das Entscheidungselement

Mit dem Entscheidungselement wird ein neuer Baustein in den Editor integriert, der eine Modularisierung von Prozessketten erlaubt. Wie bereits in der Einleitung zu diesem Kapitel erwähnt, sind die im Rahmen einer Supply Chain modellierten Prozessketten in der Regel umfangreich. Möchte der Modellierer zusätzlich alternative Strukturen einführen, so wird das Modell unübersichtlich. Diese Komplexität soll an einem Beispiel erläutert werden.
In einer Supply Chain kann der OEM zwischen fünf LDL wählen, wobei er je nach Größe, Dringlichkeit, Zielort und Verfügbarkeit seine Entscheidung trifft. Für die Modellierung bedeutet dies, dass der OEM eine Reihenfolge der LDL anhand der Kriterien Größe, Dringlichkeit und Zielort erstellen muss. Anschließend wird die Verfügbarkeit des LDL mit der höchsten Priorität abgefragt. Kann er liefern, so wird ihm der Auftrag erteilt. Im negativen Fall wird der nächste LDL

[22] So musste vor dem Löschen einer Funktionseinheit überlegt werden, was mit enthaltenen Knoten geschehen sollte. Da diese Aktion aber in der Aussenansicht der Funktionseinheit durchgeführt wurde, fehlte die Übersicht über die interne Struktur, d. h. es konnte leicht passieren, dass unbeabsichtigt Knoten entfernt wurden.

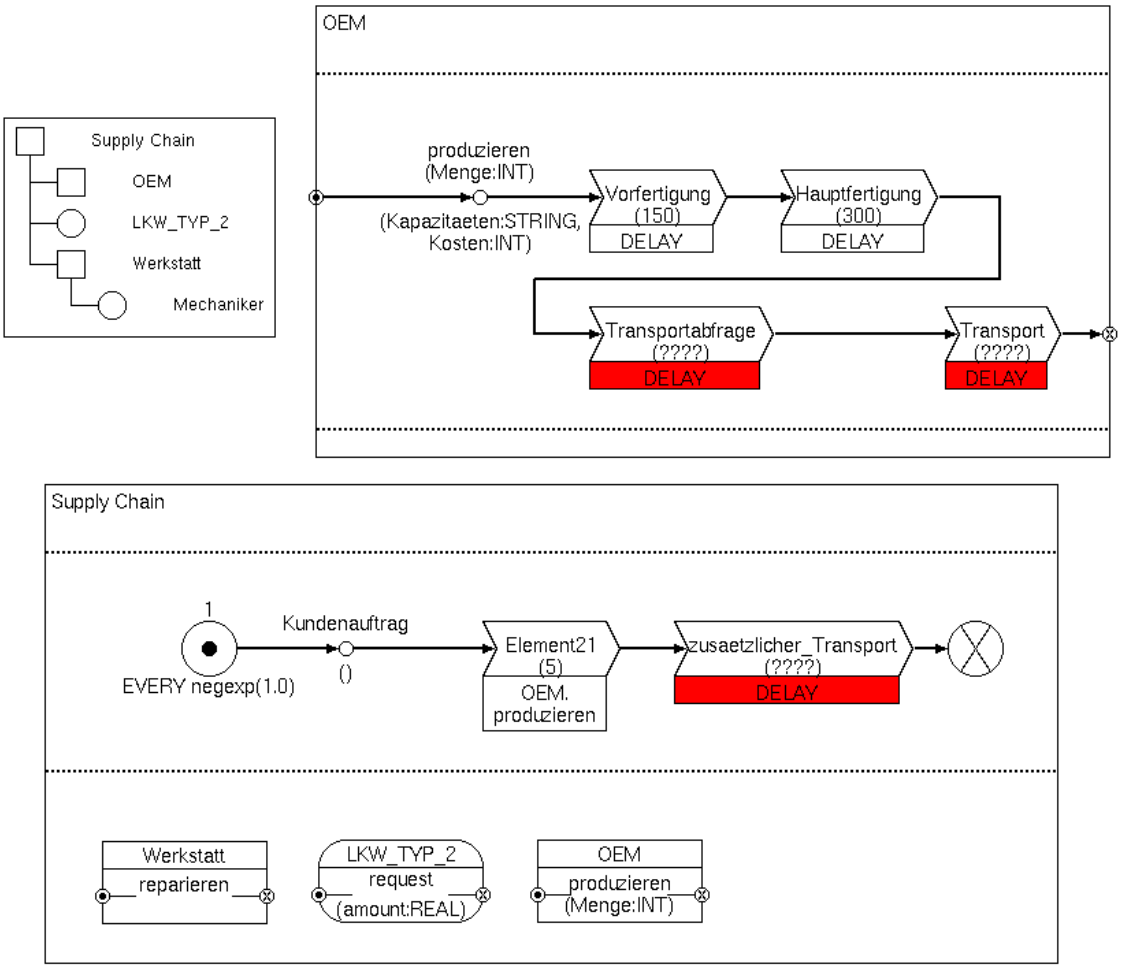


Abbildung 5.30: Löschen von Funktionseinheiten, Teil 3.
Links oben: Neuer Hierarchiebaum.
Rechts oben: Innenansicht des OEM mit markierten Elementen.
Unten: Innenansicht der Supply Chain mit markierten Elementen. (Quelle: eigene Darstellung)

Abbildung 5.31: Button zum Anlegen eines Entscheidungselementes (rot umrandet). (Quelle: eigene Darstellung)

kontaktiert, bis schließlich ein lieferfähiger LDL gefunden ist. Kann keiner der LDL den Auftrag übernehmen, muss ein „Notfallplan“ bestehen, um den Transport abzuwickeln. Wie anhand der verbalen Beschreibung schon zu erkennen ist, wird die Modellierung dieser Situation mit vielen Abfragen und Verzweigungen, d. h. Oder-Konnektoren, verbunden sein. Dabei stellt diese Transportentscheidung aber nur einen Teil der Prozesskette dar, da ebenso beispielsweise der Auftragseingang, die Produktion und die Beschaffung modelliert werden müssen. Wenn zusätzlich in diesen komplexen Ablauf noch die Einführung von E-Commerce als alternative Struktur integriert werden soll, kommen weitere Abfragen und Verzweigungen hinzu, so dass der Modellierer leicht die Übersicht verlieren kann. Ein weiteres Problem bei dieser Art von Modellierung ist die Verwischung der Hauptaktivitäten. Durch die große Anzahl an Prozesskettenelementen, die für Entscheidungen und alternative Abläufe benötigt werden, also Nebentätigkeiten darstellen, tritt die Hauptstruktur der Prozesskette in den Hintergrund.

An dieser Stelle setzt das Entscheidungselement an, indem es dem Modellierer erlaubt, Teilprozesse aus der Prozesskette auszugliedern. Die Verwendung erfolgt analog zu derjenigen von Funktionseinheiten. Abbildung 5.31 zeigt einen Ausschnitt aus der Buttonleiste in der Innenansicht von Funktionseinheiten mit dem hinzugefügten Button zum Anlegen eines Entscheidungselementes. Ist ein Entscheidungselement erzeugt, so kann es mit einer virtuellen Kante mit einem Element verbunden werden. Die Anzahl der möglichen Verbindungen ist auf eins beschränkt.

Der weiteren Betrachtung des Entscheidungselementes liegt das schon mehrfach angeführte Beispiel einer Supply Chain mit einem OEM und einem LDL zugrunde.[23] Die einzige Ergänzung besteht darin, dass der OEM über ein Lager vom Typ Storage verfügt.

Durch die Erzeugung eines Entscheidungselementes wird ein neuer Subgraph angelegt. Dieser verfügt direkt nach dem Anlegen bereits über eine Prozesskette „EGraph“[24] mit einem FEQuellePort, einer entsprechenden ProzessID und einem FESenkePort. Neben dieser Prozesskette werden die Variablen, die enthaltenen Funktionseinheiten sowie die Counter, (Prio)Server und Storages des umgebenden Subgraphen eingeblendet. Diese Bausteine können nicht modifiziert werden. Dieser Umstand wird durch eine graue Umrandung des unteren Drittels der Zeichenfläche mit einer

[23] Vgl. z. B. Kapitel 5.3.1.

[24] Aus diesem Grund wird im Folgenden auch vom Entscheidungsgraph gesprochen, wenn dieser Subgraph gemeint ist.

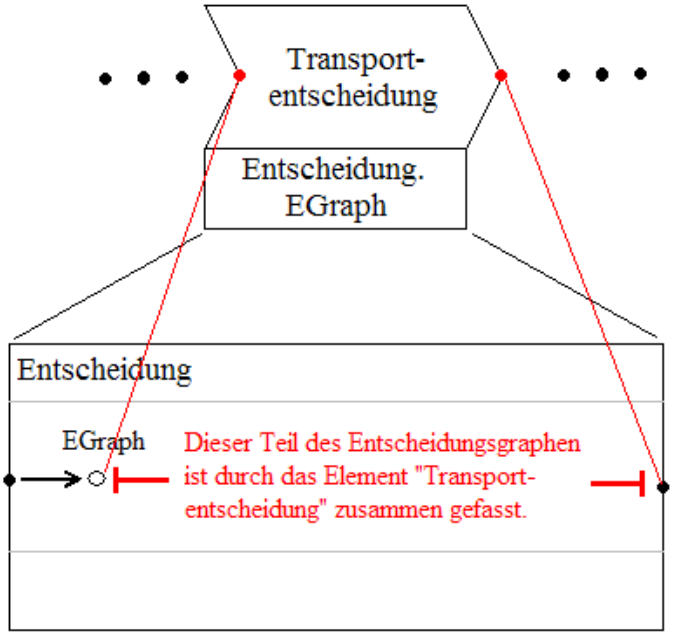


Abbildung 5.32: Idee hinter dem Entscheidungselement. (Quelle: eigene Darstellung)

Anmerkung „Keine Änderungen möglich“ gekennzeichnet. Soll einer dieser Knoten geändert werden, so muss dies im umgebenden Subgraphen geschehen. Eine Übergabe von Parametern an den Prozess „EGraph“ ist nicht vorgesehen. Vielmehr werden die Parameter, Ausgabeparameter und Variablen des Prozesses angezeigt, der das Prozesskettenelement enthält, das wiederum mit dem Entscheidungselement verbunden ist. Dies hat mit der Idee, die hinter dem Entscheidungselement steckt, zu tun. Es extrahiert einen Teil aus einer Prozesskette, d. h. das Element, das mit ihm verbunden ist, steht stellvertretend für die Elemente des Entscheidungsgraphen. Der Abschnitt zwischen der ProzessID und dem FESenkePort in diesem Graphen füllt quasi das betroffene Element aus, anders gesagt, die ProzessID und der FESenkePort sind die Schnittstellen zum Hauptprozess. Abbildung 5.32 stellt diese Idee grafisch dar. In diesem Sinn stellt der Entscheidungsgraph keinen eigenen Prozess „EGraph“ zur Verfügung.[25] Da es sich um einen Ausschnitt aus dem Hauptprozess handelt, stehen im Entscheidungsgraph dieselben Parameter und Variablen zur Verfügung. Aus diesem Grund können auch die anderen Funktionseinheiten des umgebenden Subgraphen verwendet werden. Dieses Verhältnis zu den anderen Funktionseinheiten liefert die Begründung, warum auf ein Anzeigen der Entscheidungselemente im Hierarchiebaum verzichtet wird. Sie stellen keine enthaltenen Knoten dar, stehen aber auch nicht mit dem umgebenden Knoten auf einer Stufe. Die Situation nach dem Anlegen eines Entscheidungselementes sowie nach der Erzeugung einer Verbindung zu einem Element zeigt Abbildung 5.33.

[25] Die Verwendung der Ports und der ProzessID liegt in der Implementierung begründet.

Der Abbildung 5.33 ist zu entnehmen, dass im Entscheidungsgraphen beschränkte Modellierungsmöglichkeiten vorliegen. Es steht nur ein Teil der möglichen Knoten zur Verfügung. Der Modellierer kann Kommentare, Elemente, Codeelemente, Schleifen und Konnektoren anlegen. Das Anlegen weiterer ProzessIDs wird unterbunden, weil der Entscheidungsgraph genau einen definierten Start- und Endpunkt haben muss. Diese sind wie oben erwähnt bereits vorgegeben. Aus diesem Grund können auch keine Quellen, Senken, FEQuellePorts und FESenkePorts erzeugt werden. Ebenso können keine neuen Funktionseinheiten und Variablen angelegt werden. Diese Bausteine müssen, wenn benötigt, im umgebenden Subgraphen erstellt werden. Um den Import der erwähnten Knoten zu unterbinden, ist es nicht möglich, in einen Entscheidungsgraphen einzufügen, d. h. die Punkte „Einfügen“ und „Einfügen aus“ im Menü „Bearbeiten“ sind wie auch der entsprechende Button in der Buttonleiste entfernt. Für den Zweck, für den die Entscheidungselemente konzipiert sind, reichen die restlichen Knoten aus. Es ist möglich, Verzweigungen, Abfragen und Vergleiche durchzuführen, um so Teile des Hauptprozesses auszulagern und seine Darstellung auf die Kernaktivitäten zu reduzieren. Im Beispiel ist die Transportentscheidung, die sehr einfach ausfällt, separat modelliert. Abbildung 5.34 zeigt den entsprechenden Entscheidungsgraphen mit der Nutzung der externen Funktionseinheit.

Hat der Modellierer das Modell mit Entscheidungselementen erzeugt, kann es sinnvoll sein, die entsprechenden Graphen im Hauptprozess einzublenden, d. h. das Element, das bisher mit dem Entscheidungselement verbunden ist, wird ersetzt. Um diesen Vorgang umzusetzen, ist das Menü „Bearbeiten“ in der Prozessketten-Editor-Oberfläche ergänzt worden.[26] Mittels der Punkte „Entscheidungselemente expandieren“ und „Entscheidungselemente komprimieren“ kann zwischen den beiden Ansichten hin- und hergeschaltet werden. Dabei bedeutet „expandieren“, dass die Entscheidungselemente erweitert werden, d. h. ihr Prozess „EGraph“ wird in den Hauptprozess eingefügt. Dabei werden die vorgegebenen Ports und die ProzessID nicht berücksichtigt. Die eingefügten Elemente des Entscheidungsgraphen werden farblich gekennzeichnet. Um die Konsistenz des Modells zu bewahren, sind Änderungen in dieser expandierten Sichtweise nicht möglich.[27] Wird das Modell wieder komprimiert, wird die Darstellung zurückgesetzt. Die Speicherung des Modells erfolgt in der expandierten Weise, da so eine Struktur vorliegt, die auch von Tools benutzt werden kann, die die Entscheidungselemente nicht kennen. Diese Darstellungsform wird ebenfalls für das Anlegen von Experimentserien gewählt. Abbildung 5.37 präsentiert das Beispiel in der beschriebenen Situation. Die Elemente des Entscheidungsgraphen sind ebenso markiert wie das Entscheidungselement selbst, und ihre Dienstaufrufe sind wieder hergestellt.

[26] Vgl. Abbildung 5.35.
[27] Vgl. Abbildung 5.36.

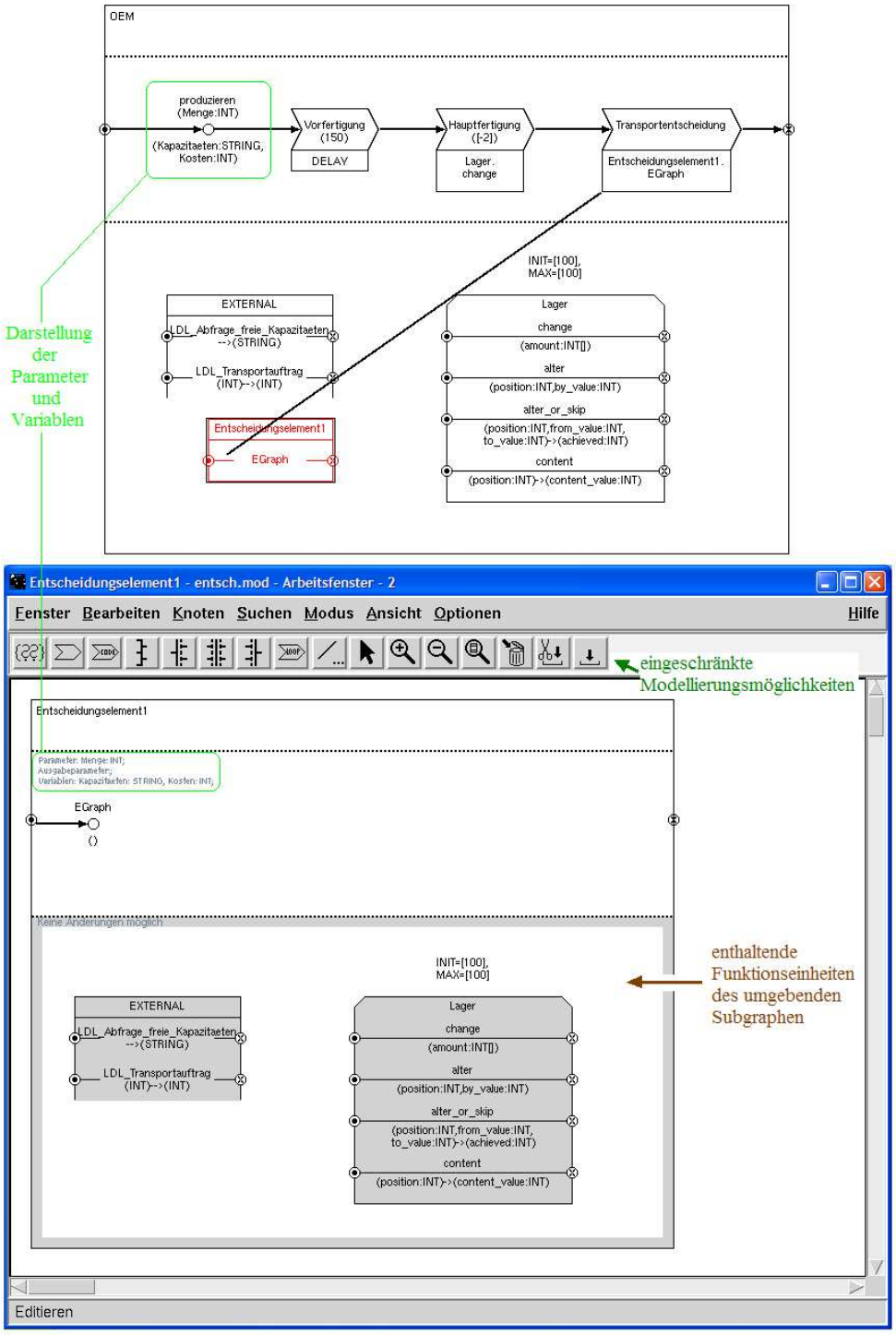


Abbildung 5.33: Situation nach Anlegen eines Entscheidungselement.
Oben: Innenansicht des OEM mit dem angelegten Entscheidungselement, das mit einem Element im Prozess „produzieren“ verbunden ist.
Unten: Innenansicht des Entscheidungselementes direkt nach dem Anlegen. (Quelle: eigene Darstellung)

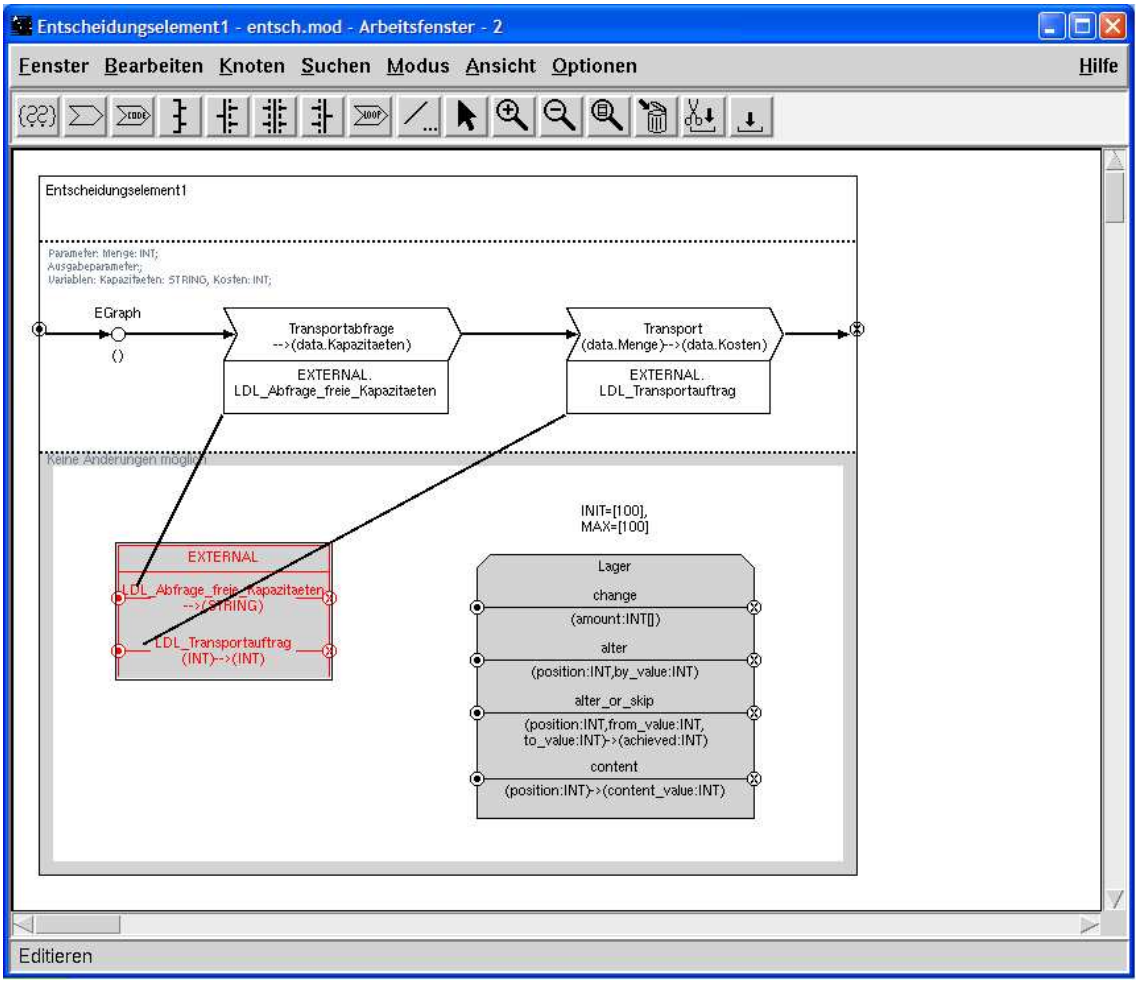


Abbildung 5.34: Beispiel eines einfachen Entscheidungsgraphen. (Quelle: eigene Darstellung)

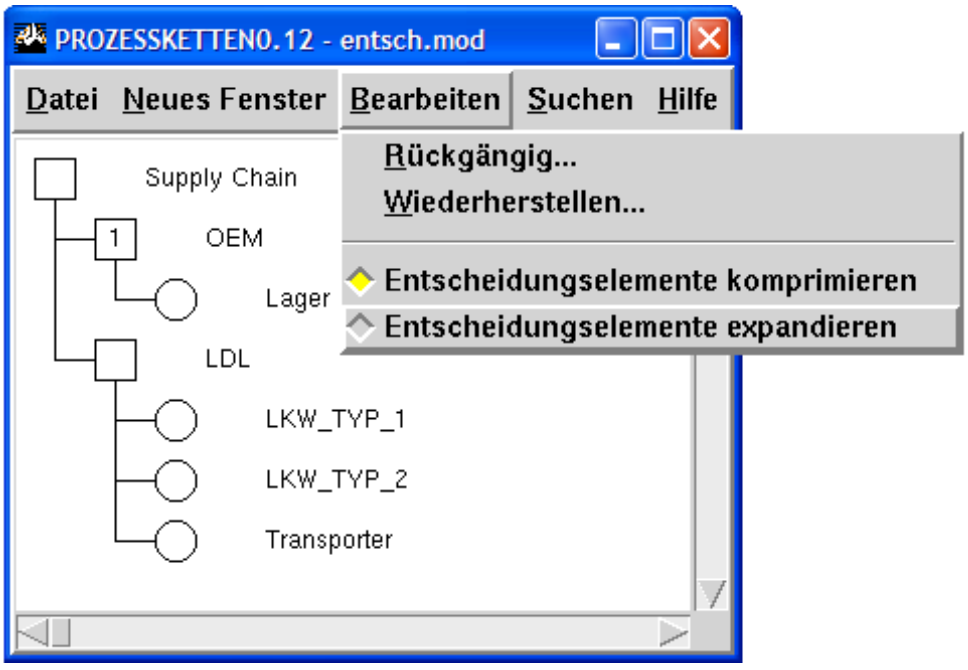


Abbildung 5.35: Schalter zum Expandieren und Komprimieren. (Quelle: eigene Darstellung)

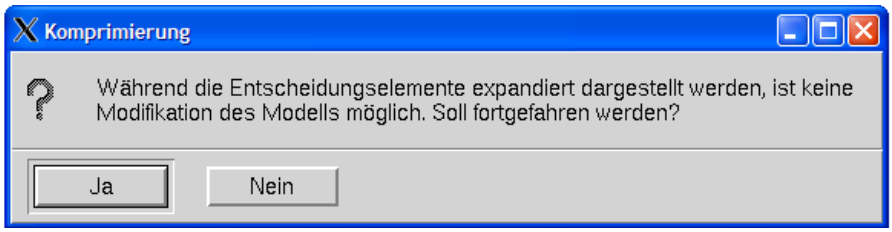


Abbildung 5.36: Warnmeldung vor Durchführung der Expandierung. (Quelle: eigene Darstellung)

Insgesamt kann festgehalten werden, dass Entscheidungselemente eine übersichtliche Gestaltung

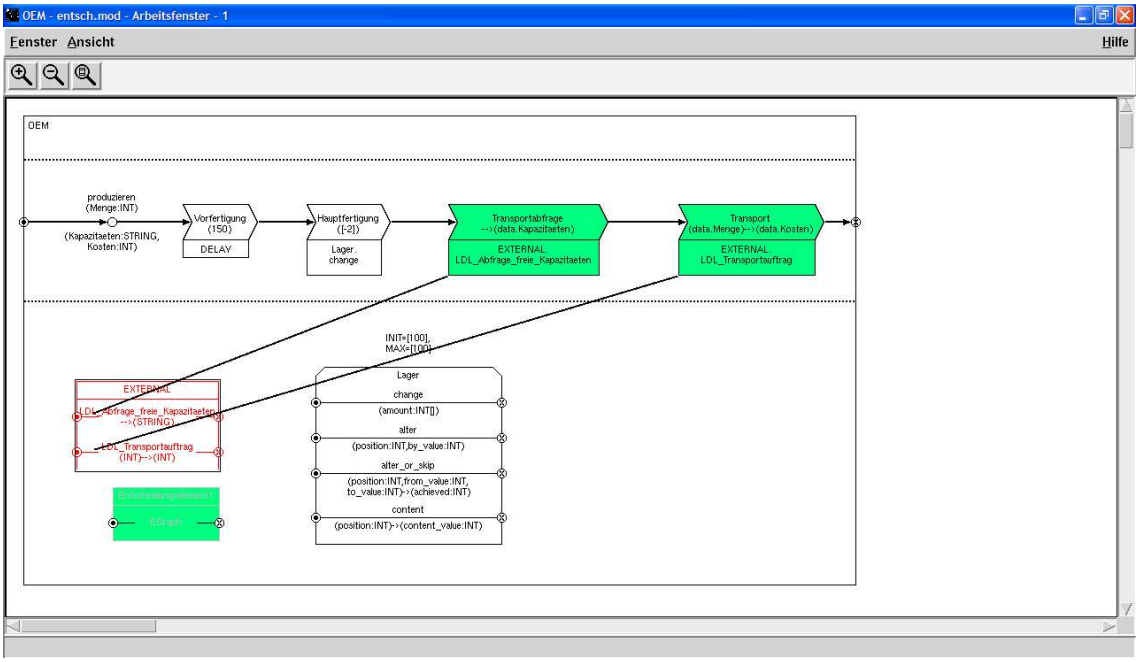


Abbildung 5.37: Innenansicht des OEM in der expandierten Sichtweise. (Quelle: eigene Darstellung)

komplexer Graphen, wie sie sich im Supply Chain Management ergeben können, ermöglicht. Sie versetzen den Modellierer in die Lage, wichtige Aktivitäten der Prozessketten hervorzuheben und weniger wichtige Aktivitäten, die zum Teil auch nur auf Grund der Modellierung benötigt werden (z. B. Abfragen oder Codeelemente), auszulagern. Anzumerken ist abschließend, dass bei der Vergabe von Namen von Knoten auf Eindeutigkeit zu achten ist. Dies bezieht sich auf den Entscheidungsgraphen und seine umgebende Funktionseinheit. Werden keine eindeutigen Namen vergeben, so führt dies bei der Expandierung und Komprimierung zu Fehlern und zu einem eventuellen Verlust des Modells.

Literaturverzeichnis

[1] F. Bause, H. Beilner, M. Fischer, P. Kemper, M. Völker: *The ProC/B Toolset for the Modelling and Analysis of Process Chains*, 12th International Conference on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation, TOOLS 2002, London (UK), in: T. Field, P.G. Harrison, J. Bradley, U. Harder (eds): Computer Performance Evaluation, Modelling Techniques and Tools, Lecture Notes in Computer Science, No 2324, Springer, pp. 51-70, 2002.

[2] F. Bause, H. Beilner, M. Schwenke: *Semantik des ProC/B-Paradigmas*, Technical Report SFB 559 Nr. 03001, ISSN 1612-1376

[3] H. Beilner, F. Bause, H. Tatlitürk, A. van Almsick, M. Völker: *Zum B-Modellformalismus, Version B1*, SFB-Bericht Nr. 99002.

[4] J. Finzel: *Toolunterstützung zur Aggregatbildung in Prozessketten-orientierten Modellwelten*. Diplomarbeit, Universität Dortmund, Fachbereich Informatik, Lehrstuhl Informatik IV, Dortmund, 2003.

[5] M. Buttner (ed.): *HI-SLANG Reference Manual*, Universität Dortmund, Lehrstuhl Informatik IV, 1993, www4.cs.uni-dortmund.de/HIT/HITDOCUPDF/HISLANG-Refman.pdf.

[6] M. Eickhoff, M. Hierweck, M. Schwenke: *Hands On ProC/B-Tools*, Technical Report SFB 559 Nr. 06003, ISSN 1612-1376

[7] J. Kriege: *Konsistenzprüfung von ProC/B-Modellen zur Vorbereitung einer simulativen Analyse*. Diplomarbeit, Universität Dortmund, Fachbereich Informatik, Lehrstuhl Informatik IV, Dortmund, 2006.

[8] C. Tepper: *Anwendung simulativer Aggregierung bei der Analyse eines Güterverkehrszentrums*, Technical Report SFB 559 Nr. 03018, ISSN 1612-1376

[9] G. Terhardt: *Modellierung und Bewertung von Supply Chain-Modellen unter Berücksichtigung variierender Strukturen*. Diplomarbeit, Universität Dortmund, Fachbereich Informatik, Lehrstuhl 4, Dortmund, 2005.

Sonderforschungsbereich 559

Bisher erschienene Technical Reports

03032 Marco Motta, Iwo Riha, Stefan Weidt: Simulation eines Regionallagerkonzeptes

03034 Frank Laakmann, Iwo Riha, Niklas Stracke, Stefan Weidt: Workbenchgestützte Konstruktion von Beschaffungsketten

03035 Iwo Riha, Stefan Weidt: Entwicklung einer Bewertungssystematik für Beschaffungsketten

04001 André Alberti, Bernd Hellingrath, Stefan Weidt, Markus Witthaut: Ergebnisse und Schlussforderungen der Simulationsexperimente im Szenario Automobilindustrie

04002 Kay Hömberg, Dirk Jodin, Maren Leppin: Methoden der Informations- und Datenerhebung

04003 Carsten Tepper: Prozessablauf-Visualisierung von ProC/B-Modellen

05001 Jochen Bernhard, Miroslav Dragan, Sigrid Wenzel: Evaluation und Erweiterung der Kriterien zur Klassifizierung von Visualisierungsverfahren für GNL

05002 Bernd Hellingrath, Sana Mehicic-Eberhardt, Markus Witthaut: Entwicklung eines Anaylserahmens für die Untersuchung organisatorischer Aspekte in der Supply Chain

05003 Dennis Müller, Mathias Stöber, Axel Thümmler: Einsatz der Response Surface Methode zur Optimierung komplexer Simulationsmodelle

05004 Dirk Jodin, Andreas Mayer: Automatisierte Methoden und Systeme der Datenerhebung

05005 Thomas Fender, Anne Krampe, Sonja Kuhnt: Kriterien für die Kategorisierung statistischer Methoden im Rahmen eines Methodennutzungsmodells zur Informationsgewinnung in GNL

05006 Kay Hömberg, Dirk Jodin, Maik Langenbach, Christian Kellner: Konzept einer logistischen Informationsbedarfsanalyse mit Hilfe von Basisprozessen und standardisierten Logistikdaten

05007 Hans-Werner Graf: Festlegung der Abfahrts- und Ankunftszeiten (Fahrplangestaltung)

06001 Iwo Riha: Grundlagen des Cost-Benefit-Sharing

06002 Jens Finzel, Michael Hierweck, Andreas van Almsick, Jan Sören Kriege, Mathias Schwenke: ProC/B-Editor – Handbuch

Alle Technical Reports können im Internet unter
http://www.sfb559.uni-dortmund.de/
abgerufen werden. Für eine Druckversion wenden Sie
sich bitte an die SFB-Geschäftsstelle
e-mail: andrea.grossecappenberg@iml.fraunhofer.de