# Verwaltung von Abhängigkeiten in flexiblen Arbeitsabläufen

Frank Schepp

Diplomarbeit
Universität Kaiserslautern
Juli 1995


Betreuung:
Prof. Dr. Michael M. Richter
Dr. Frank Maurer
Dipl.-Inform. Gerd Pews

# Erklärung:

Hiermit erkläre ich, Frank Schepp, daß ich die vorliegende Diplomarbeit selbständig verfaßt und keine anderen Hilfsmittel als die von mir angegeben verwendet habe.

Kaiserslautern, den 24. Juli 1995 (Frank Schepp)

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Kapitel 1

# Einleitung

In diesem einleitenden Kapitel erfolgt zunächst eine Motivation der Aufgabenstellung. Anschließend wird die Aufgabenstellung kurz dargestellt und eine Übersicht über den Inhalt der Arbeit gegeben.

## 1.1 Die Motivation der Aufgabenstellung

Der Koordination und Integration mehrerer an einem Projekt arbeitender Personen kommt gerade in letzter Zeit eine immer größere Bedeutung in der betrieblichen Praxis zu.

Die sich durch den global verschärften Wettbewerb ergebende erhöhte Dynamik der Marktveränderungen und deren Qualität führt zu immer kürzeren Produktlebenszyklen, zunehmender Komplexität der Produkte und zwingt die Unternehmen zur Optimierung und Flexibilisierung von Arbeitsabläufen.

Die erhöhte Komplexität von Produkten führt zu einer erhöhten Komplexität der betrieblichen Arbeitsabläufe und der damit verbundenen Datenflüsse. Darüber hinaus erfordern sowohl die kürzeren Produktlebenszyklen als auch die komplexeren Produkte unter anderem den Einsatz einer sich fortwährend vergrößernden Anzahl von Personen, die eine Aufgabe gemeinsam bearbeiten. Dadurch entsteht ein vermehrter Aufwand an Koordination und Verwaltung bei Aufgaben, die kooperatives Design[1] erfordern. Dies macht den Einsatz von den Prozeß des kooperativen Designs unterstützenden Systemen naheliegend.

[1] Kooperatives Design bezeichnet hier in Anlehnung an [Maurer F., 1995] Entwurfs- und Konstruktionsprozesse bei denen mehrere Bearbeiter zeitlich und/oder räumlich verteilt an einem gemeinsamen, nicht vollständig automatisierbaren Entwurf arbeiten. Dabei bestehen komplexe Abhängigkeiten zwischen den Teilentwürfen der einzelnen Bearbeiter und im Verlaufe des Entwurfsprozesses wird auf komplexe und umfangreiche Informationen zugegriffen.

## 1.2 Die Aufgabenstellung

Ein System, welches den Anforderungen des kooperativen Designs Rechnung trägt, ist CoMo-Kit[2]. Im Sinne eines Knowledge-Enginering-Ansatzes [Wielenga *et al.*, 1992] lassen sich unter anderem folgende Anforderungen an ein System für kooperatives Design formulieren:

- Eine bzgl. Simulation adäqate Beschreibung der Arbeitsabläufe in einem konzeptuellen Modell.
- Die Operationalisierung des konzeptuellen Modells, um eine Validierung desselben vornehmen zu können.

In CoMo-Kit werden Arbeitsabläufe durch Datenflüsse beschrieben. Die in realen Arbeitsabläufen auftretenden Datenobjekte sind meist komplexstrukturierte Objekte (Vgl. Frames). Darüber hinaus ist im Zusammenhang mit der Validierung von konzeptuellen Modellen die Vorausplanung innerhalb komplexstrukturierter Arbeitsabläufe erwünscht. Daraus ergibt sich die folgende Aufgabenstellung:

1. Es sollen Datenflüsse modelliert und operationalisiert werden, die Abhängigkeiten enthalten, die zwischen beliebigen Ebenen der Struktur von Datenobjekten und Aufgaben bestehen.
2. Bei komplexstrukturierten Aufgaben[3], soll im Verlauf der Validierung des zugehörigen konzeptuellen Modells unabhängig vom Vorliegen der im konzeptuellen Modell spezifizierten Eingabedaten eine Dekomposition in die zugehörigen Teilaufgaben möglich sein.

Die sich aus Punkt 1) der Aufgabenstellung ergebende erhöhte Komplexität der modellierten Datenflüsse, erfordert eine geeignete Unterstützung des Modellierenden bei der Erstellung von konzeptuellen Modellen. Insbesondere muß eine Konsistenzerhaltung und geeignete Visualisierung der konzeptuellen Modelle erfolgen. Darüber hinaus müssen bei der Simulation der Arbeitsabläufe die jeweils benötigten Datenobjekte erzeugt werden. Um diesen Anforderungen gerecht zu werden, erfolgte eine Erweiterung der Ausdrucksmöglichlichkeiten der konzeptuellen Modelle und eine Anpassung der Modellierungswerkzeuge von CoMo-Kit, um eine Visualisierung dieser zu erreichen. Darüber hinaus wurden Strategien zur Konsistenzerhaltung der konzeptuellen Modelle und zur Erzeugung von Objektstrukturen zur Laufzeit des Systems entwickelt und implementiert.

Im Verlaufe der Erstellung der Arbeit stellte sich heraus, daß eine Unterscheidung zwischen der Ausführbarkeit und Planbarkeit einer Aufgabe sinnvoll wäre. Durch die der Arbeit zugrundeliegenden Implementierung wird nachwievor nur die Ausführbarkeit einer Aufgabe operationalisiert. Auf dieses Thema wird im Ausblick in Abschnitt 5.2 nochmals eingegangen.

[2]Conceptual Model Construktion Kit [Maurer, 1993].

[3]Als komplexstrukturierte Aufgaben werden hier solche Aufgaben bezeichnet, die sich wiederum aus Teilaufgaben zusammensetzen.

## 1.3 Aufbau der Arbeit

Die Aufgabenstellung dieser Arbeit bezieht sich auf das Modellierungswerkzeug CoMo-Kit, welches in Version 3.0 vorliegt.

In Kapitel 2 wird zunächst eine Einführung in die Grundlagen von CoMo-Kit gegeben und es erfolgt eine Präzisierung der Aufgabenstellung. Ansonsten sind keine besonderen Vorkenntnisse zur Lektüre dieser Arbeit notwendig[4]. Im dritten Kapitel werden dann die Konzepte zur Lösung der Probleme der Aufgabenstellung vorgestellt. In Kapitel 4 wird ein Beispiel benutzt, um einige der in Kapitel 3 eingeführten Konzepte zu demonstrieren. Im darauf folgenden Kapitel 5 erfolgt eine Zusammenfassung der Ergebnisse der Arbeit und es wird ein Ausblick auf zukünftige, mögliche Weiterentwicklungen des Systems gegeben, die sich aus einer Anzahl noch offener Probleme ergeben.

[4]Eine praktische Vorführung der, aus der Aufgabenstellung resultierenden, Implementierung kann aber hilfreich für das Verständnis der Arbeit sein.

# Kapitel 2

# Diskussion der Aufgabenstellung im Zusammenhang mit CoMo-Kit

Zunächst erfolgt eine Einordnung von CoMo-Kit in den Kontext des Knowledge-Engineering. Anschließend wird eine Einführung in die Architektur und Basisstrukturen von CoMo-Kit gegeben. Der nächste Abschnitt hat die Architektur und Funktionalität des Interpreters zum Gegenstand. Im darauf folgenden Abschnitt wird auf Probleme der bisherigen Implementierung eingegangen, die die im letzten Abschnitt dargestellte Aufgabenstellung der Arbeit nochmals motivieren.

## 2.1 Allgemeines zum Modellierungswerkzeug CoMo-Kit

CoMo-Kit [Maurer, 1993] ist eine Entwicklungsumgebung für verteilte, wissensbasierte Informationssysteme. Es soll Wissensingenieure und Experten bei der Erstellung solcher Systeme unterstützen. Man kann folgende Merkmale zur Charakterisierung der Domänen, in denen solche Systeme benötigt werden anführen:

- Es liegen komplexe Aufgaben vor,
- Die Lösung erfordert neben einem Inferenzmechanismus Wissen,
- Mehrere Sachbearbeiter lösen die Aufgabe kooperativ,
- Die Lösung erfolgt möglicherweise räumlich und zeitlich verteilt an mehreren Arbeitsplätzen.

Eine modellbasierte Wissensaquisitionsmethode ist der KADS-Ansatz[1]. KADS beschreibt den Weg vom Expertenwissen zu einem lauffähigen System durch die

[1] Nach dem europäischen Forschungsprojekt KADS (Knowledge Acquisition and Documentation System) siehe [Wielenga *et al.*, 1992].

Stationen konzeptuelles Modell, Designmodell und Implementationsmodell. Die Beschreibung der grundlegenden Struktur der Domäne erfolgt in der Analysephase durch einen Experten. Es sind Wissenstypen und Verarbeitungstypen zu identifizieren, die im konzeptuellen Modell festgehalten werden sollen. Das konzeptuelle Modell liefert eine implementationsunabhängige Spezifikation des Expertensystems, die sich an der Vorgehensweise des Experten orientiert.

Alle Bestandteile des konzeptuellen Modells können, mit Hilfe von CoMo-Kit, im System abgebildet werden. Zur Beschreibung von kooperativen, wissensbasierten Systemen [Maurer, 1993; Schmitz, 1994; Dellen, 1995] werden die Basisstrukturen Konzept(concept), Aufgabe(task), Methode(method) und Agent(agent) benutzt. Dazu setzt man bei der Beschreibung von problemspezifischen konzeptuellen Modellen diese Basisstrukturen zueinander in Relation.

Die Operationalisierung und Validierung der mit CoMo-Kit entwickelten konzeptuellen Modelle ermöglicht eine Interpreterkomponente. Dadurch wird eine sukzessive Entwicklung und Validierung des konzeptuellen Modells ermöglicht, das heißt obwohl CoMo-Kit auf einem modellbasierten Wissensaquisitionsansatz fußt, kann man die Vorteile des Rapid-Prototyping nutzen.

Bisherige reale Anwendungsdomänen von CoMo-Kit sind die Bebauungsplanung [Maurer F., 1995] und das Software-Engineering[2].

## 2.2 Architektur und Basisstrukturen von CoMo-Kit

Die Architektur von CoMo-Kit zerfällt in drei Hauptkomponenten (siehe Abb. 2.1 entnommen aus [Dellen, 1995]): Die statische Wissensbasis, die die konzeptuellen Modelle beinhaltet, die Modellierungswerkzeuge für das konzeptuelle Modell in Form von Editoren und Filtern, sowie den Interpreter, der die Operationalisierung des konzeptuellen Modells realisiert. Auf die Architektur und Funktionsweise des Interpreters wird in Abschnitt 2.3 noch näher eingegangen.

In CoMo-Kit werden die folgenden Basisstrukturen [Maurer, 1993; Maurer F., 1995] zur Erstellung von konzeptuellen Modellen verwendet:

1. *Konzepte* beschreiben Daten: Datenstrukturen werden mit einem objektorientierten Ansatz modelliert, wobei man zwischen Konzeptklassen und Konzeptinstanzen unterscheidet. *Konzeptklassen* definieren die Struktur der Instanzen durch eine Menge von Attributen (Slots). Jedem Attribut wird ein Konzeptklasse und eine Kardinalität zugeordnet. *Konzeptinstanzen* lassen sich klassifizieren als Problemfall- und Problemlösewissen [Dellen, 1995]:

   (a) *Problemlösewissen* ist Wissen, das schon bei der Modellierung vorhanden ist und daher im konzeptuellen Modell abgelegt wird. Die Attribute der

[2] Verwendung erfolgt im Rahmen des SFB-501 „Generische Methoden der Softwareentwicklung".

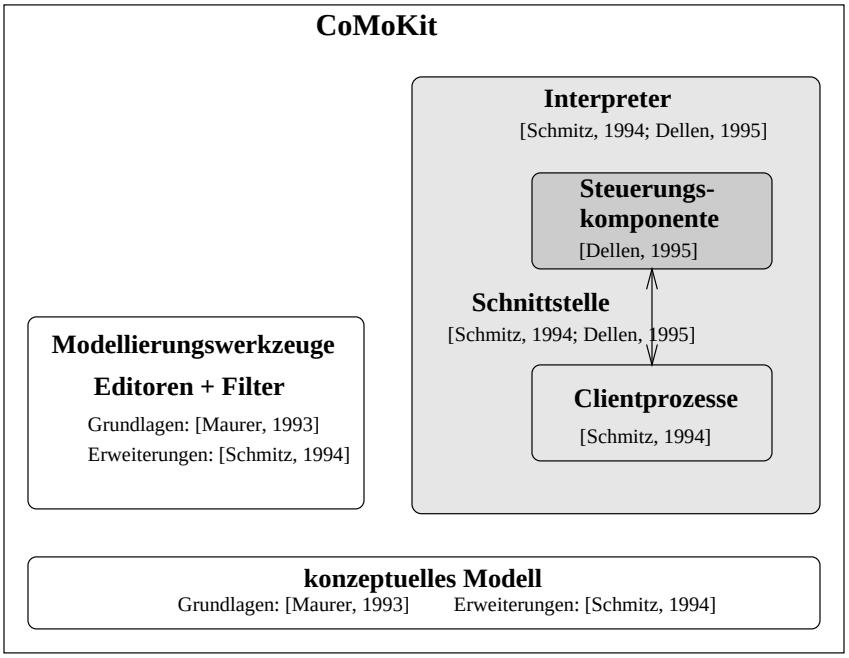


Abbildung 2.1: Die Komponenten von CoMo-Kit

Konzeptinstanzen sind (vollständig) instantiert.

(b) *Problemfallwissen* dagegen wird erst während des Lösungsprozesses generiert und stellt damit eine spezifische Lösung dar. Es muß daher bei der Lösungskomponente (Interpreter) abgelegt und verwaltet werden. Das Problemfallwissen läßt sich durch die alternative Belegung von Variablen im Zuge des Lösungsprozesses beschreiben.

Im Zusammenhang mit der Repräsentation von Problemfallwissen wurde in [Schmitz, 1994] das Konzept der formalen Parameter eingeführt. Diese *formalen Parameter* beschreiben Ein- bzw. Ausgaben von Aufgaben bzw. Methoden. Ein formaler Parameter hat einen Typ (Konzeptklasse) und wird zur Laufzeit mit einer Konzeptinstanz des entsprechenden Typs belegt.

2. *Aufgaben*: Eine Aufgabe wird durch ihre Ziele, ihre Ein- bzw. Ausgaben und die zu ihrer Bearbeitung anwendbaren alternativen Methoden beschrieben.

3. *Methoden*: Eine Methode beschreibt den Weg, wie man das Ziel (die Aufgabe) erreicht. Es wird zwischen atomaren und komplexen (zusammengesetzten) Methoden unterschieden. Atomare Methoden erzeugen Daten und belegen so Variable mit Werten. Eine komplexe Methode wird als Datenflußgraph (siehe Abb. 2.2) beschrieben. Die Knoten des Datenflußgraphen[3] repräsentieren (Unter-) Aufgaben, Variable und zur Problemlösung benötigtes Wissen,

[3] Aufgaben werden in der Graphik durch Ovale dargestellt. Rechtecke zeigen Variable und deren

während die Kanten den Aufgaben Ein- bzw. Ausgaben zuordnen. Es erfolgt eine Zuordnung des Typs (Konzeptklasse) zu einer Variablen (formaler Parameter), von diesem muß zur Laufzeit des Systems eine Instanz erzeugt und dann an die Variable gebunden werden. Eine (Unter-) Aufgabe kann weiter mit Hilfe von Methoden zerlegt werden. Die Bearbeitung von Aufgaben in parallelen Zweigen kann gleichzeitig durch verschiedene Agenten erfolgen[4]. Die Beschrei-

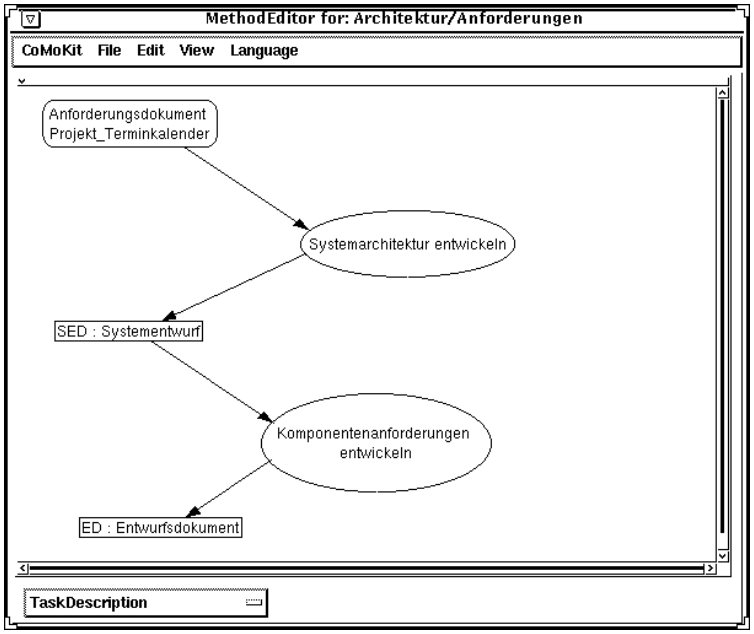


Abbildung 2.2: Durch die Methode *Architektur/Anforderungen* definierter Datenfluß

bung des Problemlöseprozesses erfolgt durch einen UND/ODER-Baum, wobei Aufgaben die UND-Knoten sind und Methoden den ODER-Knoten entsprechen. Durch Datenflußbeziehungen entstehen Reihenfolgeabhängigkeiten, die den Problemlöseprozeß zusätzlich strukturieren, indem sie einen Kontrollfluß definieren. Abb. 2.3 zeigt den Ausschnitt einer Aufgabenzerlegung [5].

4. *Agenten* bearbeiten Aufgaben: Im konzeptuellen Modell wird für jede Aufgabe definiert, welche Agenten fähig sind, sie zu bearbeiten. Welcher Agent sich um eine konkrete Aufgabe kümmern soll, muß erst zur Laufzeit des Systems festgelegt werden. Es werden zwei Arten von Agenten unterschieden: Menschen

Typ. Abgerundete Rechtecke repräsentieren Problemlösewissen.

[4]Nach [Maurer F., 1995] bedeutet dies, daß das System die Datenflußgraphen als gefärbte Petri-Netze interpretiert. Es wird auch darauf hingewiesen, daß in [Gebhardt *et al.*, 1995] ein ähnlicher Ansatz zugrundeliegt. In Kapitel 3.1.1 wird im Rahmen dieser Arbeit näher auf Datenflußabhängigkeiten eingegangen.

[5]Die Abb. 2.2 und 2.3 betreffen den Entwurf eines Softwareprodukts und wurden aus [Dellen, 1995] entnommen.

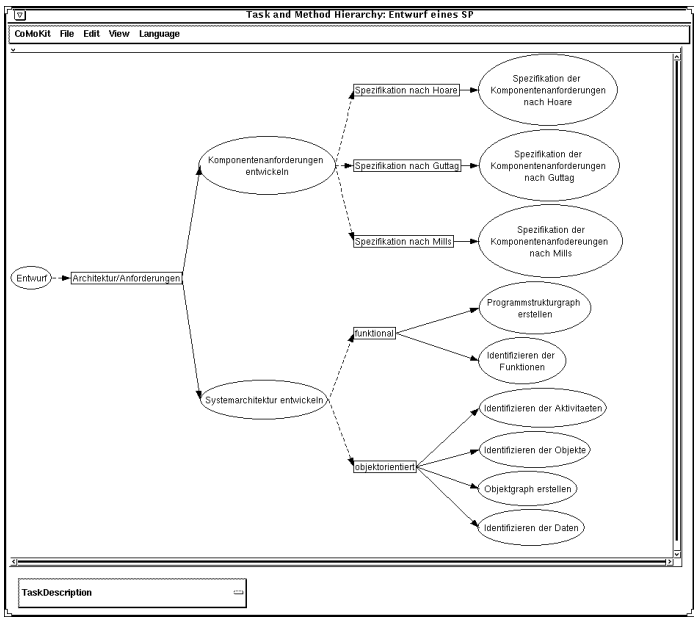


Abbildung 2.3: Aufgabenzerlegung für den Entwurf eines Softwareproduktes

und Rechner. Ein Mensch löst eine Aufgabe, indem er die Editoren des Systems benutzt und der Rechner führt Programme aus.

Bevor in den Abschnitten 2.4 und 2.5 auf die Probleme der bisherigen Implementierung eingegangen bzw. eine detailierte Darstellung der Aufgabenstellung erfolgt, wird im nächsten Abschnitt noch auf die Architektur, Funktionalität und Abhängigkeitsverwaltung des Interpreters eingegangen.

## 2.3 Die Architektur, Funktionalität und Abhängigkeitsverwaltung des Interpreters

Der Interpreter realisiert die Operationalisierung von konzeptuellen Modellen und ermöglicht eine Validierung derselben begleitend zu deren Erstellung. Zunächst soll die Architektur des Interpreters dargestellt werden. Im Anschluß daran wird auf die Mechanismen eingegangen, die dessen Funktionalität realisieren. Zum Schluß wird kurz auf die TMS-Strukturen eingegangen, die die Abhängigkeitsverwaltung des Schedulers realisieren.

### 2.3.1 Die Architektur des Interpreters

Der Interpreter (siehe Abb. 2.4) ist als Client-Server-Architektur realisiert. Der Server entspricht dem Scheduler und hat die Aufgabe den aktuellen Stand der Problemlösung zu verwalten. Dies beinhaltet eine Zustandsverwaltung der Aufgaben

mit Hilfe einer Menge von Listen, welche die verschiedenen Abarbeitungszustände von Aufgaben repräsentieren. Darüber hinaus müssen die gültigen Variablenbindungen verwaltet werden und zur Verwaltung von Abhängigkeiten zwischen Aufgaben müssen Abhängigkeitsstrukturen aufgebaut werden, die ein zielgerichtetes Backtracking unterstützen. Die vom Scheduler zur Bearbeitung freigegebenen Aufgaben werden von den Clients (Agenten) bearbeitet, indem diese eine Methode anwenden. Dabei entstehende Ergebnisse werden zum Scheduler zurücktransferiert. Eine detailierte Beschreibung der Schnittstelle zwischen Server und Client findet man in [Schmitz, 1994].

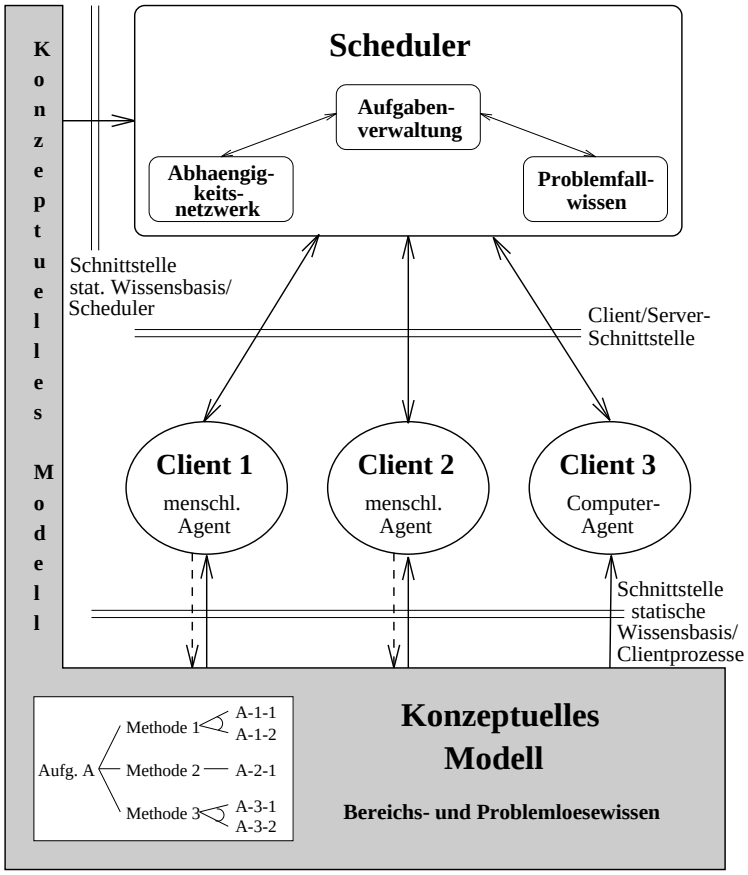


Abbildung 2.4: Die Architektur des Interpreters

### 2.3.2 Die Funktionalität des Interpreters

Durch die Anwendung von Methoden werden die im Laufe des Lösungsprozesses anfallenden Aufgaben gelöst. Wird einem Agenten eine Aufgabe vom Scheduler zur Barbeitung zugeteilt, muß sich der Agent für eine der alternativen Methoden entscheiden. Die durch das Auswählen von Methoden hergeleiteten (Teil-) aufgaben

und Konzeptinstanzen stellen den gültigen Lösungsraum dar, die Entscheidungen den Lösungsweg. Der Scheduler verwaltet den aktuellen Stand des Problemlösungsprozesses und die entstehenden Abhängigkeiten. Zu seinen Aufgaben gehören im einzelnen [Dellen, 1995]:

- Die Verwaltung der Zustände, welche die Aufgaben jeweils im Laufe des Lösungsprozesses einnehmen. Der Zustand einer Aufgabe gibt Aufschluß darüber, ob diese in Bearbeitung, gelöst oder ungültig ist.

- Die durch den Datenfluß gegebenen Abhängigkeiten zwischen den Aufgaben müssen verwaltet werden (Insbesondere müssen die Abhängigkeiten zwischen der Anwendung (atomarer) Methoden und die damit verbundenen Variablenbelegungen dargestellt werden).

- Die Protokollierung der Beziehungen, die sich aus der Zerlegung von Aufgaben in Teilaufgaben ergeben.

- Die Verwaltung der durch die Anwendung von atomaren und komplexen Methoden entstehenden Entscheidungen.

- Die Überwachung der Delegierung von Aufgaben an ihre Bearbeiter muß erfolgen.

Über ein Unterstützung bei der Abarbeitung von Aufgaben hinausgehend, erfolgt durch den Scheduler die Unterstützung des Rückzugs von Entscheidungen, die zu einer inkonsistenten Lösung führen. In der Regel hat der Rückzug von Entscheidungen globale Auswirkungen auf den Lösungsprozeß:

- Bei der Anwendung einer komplexen Methode auf eine Aufgabe wird diese in Teilaufgaben zerlegt. Wird die Zerlegung zurückgezogen, verlieren die aus dieser resultierenden Aufgaben und Lösungen ihre Gültigkeit. Dieser Vorgang pflanzt sich bis zu den Blättern des Lösungsbaumes fort.

- Eine atomare Methode belegt Variablen mit Werten, die die Grundlage für weitere Entscheidungen sind. Verliert eine bisherige Belegung ihre Gültigkeit durch den Rückzug der zugehörigen Entscheidung, muß jede darauf beruhende Lösung zurückgenommen und überdacht werden.

- Ist keine der einer Aufgabe zugeordneten Methoden (mehr) anwendbar, kann die Aufgabe nicht mehr bearbeitet werden. Für die dadurch resultierende Blockade wird ein ursachengesteuertes Backtracking durchgeführt.

Sind Entscheidungen von der Gültigkeit anderer Entscheidungen abhängig, werden entsprechende Rückzugsbedingungen formuliert. So können die Ursachen einer Inkonsistenz ermittelt werden und zur Auflösung von Blockaden durch abhängigkeitsgerichtetes Backtracking verwendet werden. Im folgenden Abschnitt soll kurz auf die Realisierung der Abhängigkeitsverwaltung durch den Scheduler eingegangen werden.

### 2.3.3 Die Abhängigkeitsverwaltung des Schedulers

Als Basismodell zur Modellierung der Abhängigkeitsverwaltung dient das allgemeine Planung- und Designmodell REDUX [Petrie, 1991]. Es ermöglicht die Dekomposition von Aufgaben, den Entscheidungsrückzug und abhängigkeitsgerichtetes Backtracking. Die dabei auftretenden Abhängigkeiten stellen logische Implikationen dar. Zur Verwaltung der Abhängigkeiten, die sich während des Lösungsprozesses ergeben, wird ein TMS [Doyle, 1979] eingesetzt, wobei alle Aufgaben, Entscheidungen und deren Abhängigkeiten auf TMS-Strukturen abgebildet werden. Auf die TMS-Strukturen wird in dieser Arbeit nur soweit eingegangen, wie dies für das Verständnis erforderlich ist (näheres siehe [Dellen *et al.*, 1995; Dellen, 1995]). Zunächst sollen die Begriffe Datenflußbeziehung und Datenflußabhängigkeit präzisiert werden:

**Datenflußbeziehungen** Diese entstehen zwischen den Aufgaben und den formalen Parametern[6] eines Datenflußgraphen, indem ein formaler Parameter als Ein- bzw. Ausgabe einer Aufgabe definiert wird.

**Datenflußabhängigkeiten** Diese bestehen zwischen den Aufgaben eines Datenflußgraphen, wenn Aufgaben die Ausgaben von anderen Aufgaben als Eingaben benutzen. Oder anders formuliert entstehen sie durch Verkettung von Datenflußbeziehungen.

**Direkte Datenflußabhängigkeit** Diese liegt vor, wenn die Aufgabe $T_2$ eine Ausgabe der Aufgabe $T_1$ als Eingabe benutzt.

**Indirekte Datenflußabhängigkeit** Diese liegt zwischen zwei Aufgaben $T_1$, $T_2$ vor, wenn eine weitere Aufgabe $T_3$ existiert, die eine Ausgabe der Aufgabe $T_1$ als Eingabe benutzt und zwischen den Aufgaben $T_3$, $T_2$ eine direkte oder indirekte Datenflußabhängigkeit vorliegt.

Wie bereits erwähnt verwaltet der Scheduler die Zustände, welche die Aufgaben jeweils im Verlauf des Lösungsprozesses einnehmen. Dabei werden die einzelnen Zustände über TMS-Stukturen modelliert (näheres siehe [Dellen, 1995]). Ob eine Aufgabe zur Bearbeitung durch den Scheduler freigegeben wird, wird über den *executable*-Knoten einer Aufgabe bestimmt. Ist dieser gültig, wird die Aufgabe zur Bearbeitung freigegeben. Diese kann dann von einen Agenten akzeptiert und bearbeitet werden. Damit der *executable*-Knoten einer Aufgabe $T$ gültig ist, müssen dessen Rechtfertigungen gültig sein (siehe Abb. 2.5 entnommen aus [Dellen, 1995]). Als Rechtfertigung wird zunächst dessen *valid-delegation*-Knoten eingetragen. Dieser ist gültig, wenn eine gültige Delegierungsentscheidung für die Aufgabe existiert. Darüber hinaus werden bisher alle Eingaben der Aufgabe in Form von *assigned*-Knoten als

[6] Datenflußbeziehungen bestehen auch zwischen Konzeptinstanzen und Aufgaben, diese werden aber bei der Begriffsbestimmung ausgenommen, weil diese als Problemlösewissen bereits zu Beginn des Lösungsprozesses vorhanden sind (siehe Abschnitt 2.2 unter 1.a)) und deshalb nicht in die Abhängigkeitsverwaltung des Schedulers einbezogen wurden.

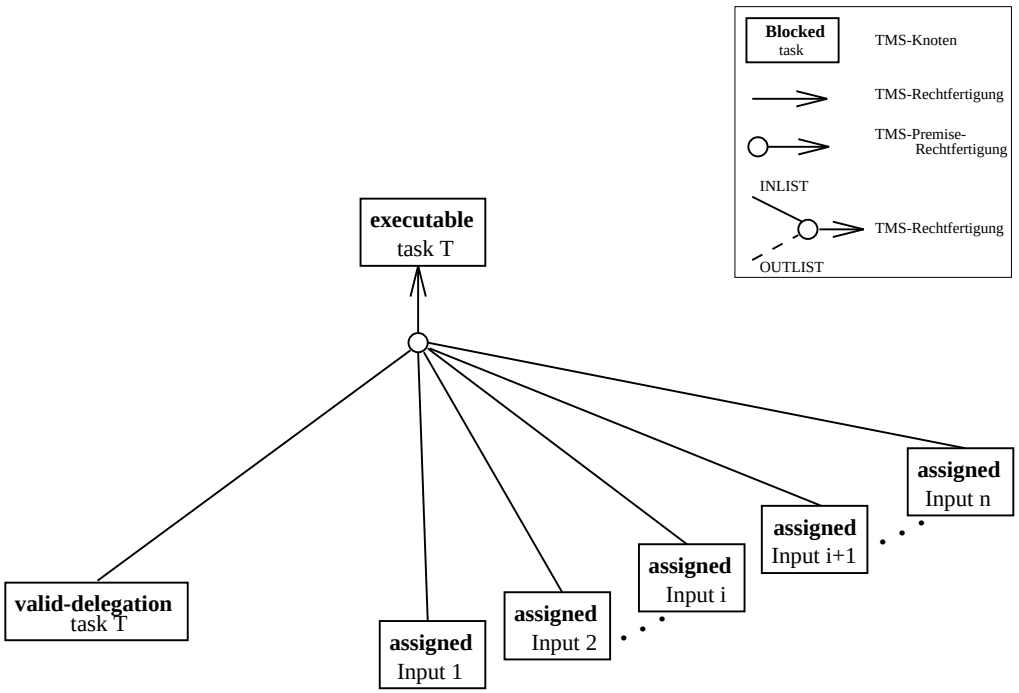


Abbildung 2.5: TMS-Strukturen zur Rechtfertigung des *executable*-Knotens einer Aufgabe $T$

Rechtfertigung eingetragen (zu den Änderungen, die der jetzigen Implementierung zugrundeliegen siehe Abschnitt 3.2). Die Gültigkeit des *assigned*-Knotens eines formalen Parameters gibt an, daß eine gültige Zuweisung für diesen existiert (siehe Abb. 2.6 entnommen aus [Dellen, 1995]). Dies wird wiederum dadurch modelliert, daß alle Zuweisungen an den formalen Parameter, die im Verlaufe des Lösungsprozesses erfolgen als Rechtfertigung in Form von *assignment*-Knoten bei dem *assigned*-Knoten eingetragen werden. Im Abb. 2.6 sind zum Beispiel bei dem *assigned*-Knoten des formalen Parameters $L$ die *assignment*-Knoten für $L = l_1$ und $L = l_2$ als Rechtfertigung eingetragen. Dabei werden die *assignment*-Knoten durch den *decision*-Knoten eines Operators gerechtfertigt. Die Gültigkeit des *decision*-Knotens eines Operators bzw. einer Methode bedeutet, daß im Lösungsprozeß eine Entscheidung für den Operator getroffen wurde, um die zugehörige Aufgabe zu bearbeiten, und diese nicht zurückgezogen wurde. Im Beispiel wurde zur Bearbeitung der Aufgabe $T_1$ eine Entscheidung für die Anwendung von Methode $m_1$ getroffen, die auch gültig ist. Wird ein *assignment*-Knoten durch die Entscheidung für einen Operator gerechtfertigt, dann bedeutet dies, daß die zum *assignment*-Knoten gehörende Instanz Ausgabe derjenigen Aufgabe ist, die durch die Anwendung des Operators, gelöst wurde. Beispielsweise wird der *assignment*-Knoten für $L = l_1$ durch den *decision*-Knoten von Methode $m_1$ gerechtfertigt. Ob die Notwendigkeit für den Rückzug der Entscheidung für eine Methode vorliegt, wird durch deren *rejected-decision*-Knoten angezeigt. Ist

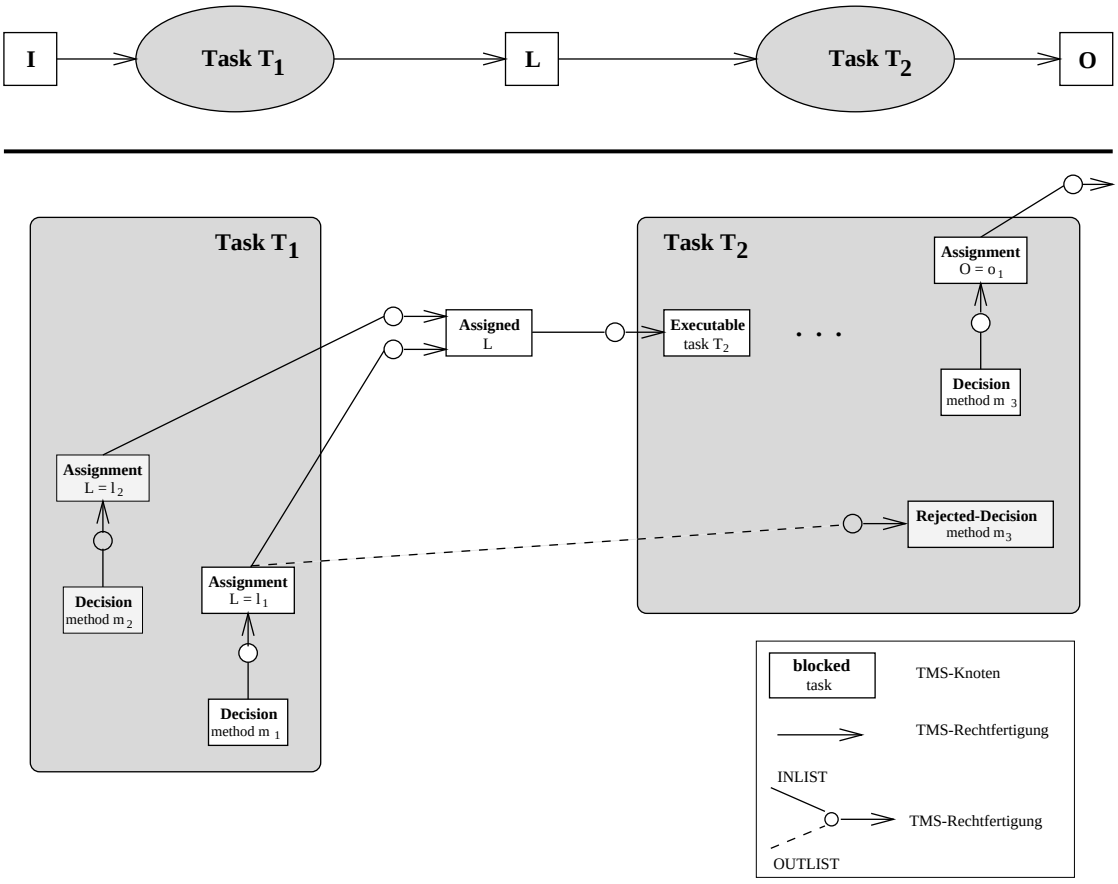


Abbildung 2.6: TMS-Strukturen für einen Datenfluß

dieser gültig, so bedeutet dies, daß Gründe vorliegen die Entscheidung für die jeweilige Methode zurückzuziehen. In die *OUTLIST* des *rejected-decision*-Knotens eines Operators werden bisher diejenigen *assignment*-Knoten eingetragen, die im konzeptuellen Modell als Eingabe der zum Operator gehörenden Methode spezifiziert sind (zu den Änderungen, die der jetzigen Implementierung zugrundeliegen siehe Abschnitt 3.1.3.4). Hier wird der *assignment*-Knoten für $L = l_1$ in die *OUTLIST* des *rejected-decision*-Knotens der Methode $m_3$ eingetragen. Das heißt, sobald eine der Eingaben des Operators ungültig ist, führt dies zur Gültigkeit des *rejected-decision*-Knotens und somit zur Ungültigkeit des *decision*-Knotens. Wenn also eine Eingabe ungültig wird, führt dies dazu, daß die zugehörige Aufgabe nicht mehr als reduziert gilt und falls möglich durch die Anwendung eines anderen Operators gelöst werden muß.

## 2.4 Probleme der bisherigen Implementierung

Die im folgenden beschriebenen beiden Probleme der bisherigen Implementierung werden im Rahmen dieser Arbeit aufgegriffen:

1. Mit Hilfe der Modellierungswerkzeuge von CoMo-Kit ist es möglich geschachtelte Objektstrukturen, in denen auch mengenwertige Attribute vorkommen, zu definieren. Die meisten in realen Aufgabenzerlegungen vorkommenden Datenflüsse enthalten komplexe Objekte in obigem Sinne. Die dabei auftretenden Datenflußabhängigkeiten zwischen Datenobjekten und Aufgaben können aber auf beliebigen Ebenen der jeweiligen Objektstruktur angesiedelt sein. In der bisherigen Implementierung von CoMo-Kit können diese Datenflüsse weder modelliert werden noch erfolgt zur Laufzeit des Systems eine Erzeugung der notwendigen und geeigneten Objekt- und Abhängigkeitsstrukturen.

2. Die Anwendung einer komplexen Methode geschieht dadurch, daß die durch die Aufgabenzerlegung entstehenden Teilaufgaben an geeignete Agenten delegiert werden. Während die Anwendung einer atomaren Methode dazu führt, daß die für die zugehörige Aufgabe spezifizierten Ausgaben gültig werden bzw. zur Verfügung stehen. In der bisherigen Implementierung können Aufgaben unabhängig davon, ob sie durch die Anwendung von komplexen oder atomaren Methoden gelöst werden, immer erst dann bearbeitet werden, wenn alle zu ihrer Bearbeitung notwendigen Eingaben[7] vorliegen. Bei realen Aufgabenzerlegungen sollen aber Delegierungsentscheidungen zum Zwecke der Vorausplanung jederzeit möglich sein.

[7]Die zur Bearbeitung notwendigen Eingaben sind diejenigen Eingaben, die für diese Aufgabe im konzeptuellen Modell als Eingaben spezifiziert wurden.

## 2.5 Detailierte Darstellung der Aufgabenstellung

An dieser Stelle erfolgt eine konkretere Darstellung der Aufgabenstellung, so wie sie sich aus den Problemen, die in Abschnitt 2.4 geschildert werden, ergibt:

1. Um die Modellierung von Datenflüssen zwischen Datenobjekten und Aufgaben auf allen Ebenen der jeweiligen Objekthierarchie zu realisieren, müssen im einzelnen folgende Aspekte betrachtet werden:

   (a) Es soll eine für die Operationalisierung geeignete Erweiterung der Ausdrucksmöglichkeiten des konzeptuellen Modells vorgenommen werden.

   (b) Die sich aus a) ergebenden erweiterten Ausdrucksmöglichkeiten des konzeptuellen Modells machen eine Anpassung der Modellierungswerkzeuge von CoMo-Kit erforderlich. Die Anpassung soll so erfolgen, daß der Modellierende soweit wie möglich bei der Erstellung von konzeptuellen Modellen unterstützt wird. Insbesondere sollte er sich möglichst wenig mit der Erhaltung der Konsistenz des konzeptuellen Modells beschäftigen müssen, da dies bei komplexeren Aufgabenzerlegungen einen erheblichen Aufwand bedeuten kann.

   (c) Die Mechanismen der Interpreterkomponente müssen so angepaßt bzw. erweitert werden, daß eine Operationalisierung der konzeptuellen Modelle aus a) erfolgt.

2. Eine Implementierung der gewünschten Funktionalität soll über eine Anpassung der Abhängigkeitsverwaltung von Aufgaben erfolgen. In Kapitel 5 wird ein Ausblick auf einen möglichen allgemeineren Ansatz gegeben, auf dessen Implementierung aber verzichtet wurde, da dieser den Rahmen einer Diplomarbeit sprengen würde.

# Kapitel 3

# Konzepte zur Lösung der Probleme der Aufgabenstellung

In den folgenden beiden Abschnitten werden die Wege dargestellt, die bei der Lösung der Probleme der Aufgabenstellung eingeschlagen wurden. Zunächst werden in Abschnitt 3.1 die Erweiterungen vorgestellt, die die Modellierung von Datenflüssen unter CoMo-Kit betreffen. Anschließend wird in Abschnitt 3.2 der Lösungsweg beschreiben, der eingeschlagen wurde, um jederzeit Delegierungsentscheidungen treffen zu können, unabhängig davon, ob die jeweils notwendigen Eingaben der Aufgabe vorliegen.

## 3.1 Konzepte zur Erweiterung der Modellierung von Datenfüssen

In diesem Abschnitt sollen die Konzepte dargestellt werden, die es ermöglichen im konzeptuellen Modell Datenflußbeziehungen zwischen einer beliebigen Unterstruktur eines komplexen Objekts und einer Aufgabe zu definieren und das so entstehende Modell mit Hilfe des Interpreters zu operationalisieren[1]. Darüber hinaus spielt die geeignete Visualisierung und die Konsistenzerhaltung des konzeptuellen Modells bei der Erweiterung der Modellierungswerkzeuge von CoMo-Kit eine wichtige Rolle. Zunächst wird beschrieben, welche Erweiterungen am konzeptuellen Modell zur Lösung der Probleme vorgenommen wurden. Dann werden die daraus resultierenden Erweiterungen bzw. Änderungen der Modellierungswerkzeuge vorgestellt. Zum Schluß wird dargestellt, wie die Operationalisierung der Erweiterung des konzeptuellen Modells durch eine Anpassung bzw. Erweiterung des Schedulers realisiert wurde.

[1] In [Oberweis, 1994] wird ein „auf höheren Petri-Netzen basierter graphischer Beschreibungsformalismus“ vorgestellt, „der es erlaubt, in intergrierter Form komplex strukturierte Objekte, Operationen auf diesen Objekten und mit diesen Operationen gebildete Abläufe adäquat zu beschreiben“. Eine „Workflow-Engine“ kann das jeweilige „Petri-Netz-Schemata unmittelbar als Grundlage für die Ablaufkontrolle und -steuerung“ einsetzen. Dieser Ansatz ist allerdings nicht auf CoMo-Kit übertragbar.

### 3.1.1 Erweiterung der Ausdrucksmöglichkeiten des konzeptuellen Modells

In CoMo-Kit können Arbeitsabläufe durch Aufgabenzerlegung und zugehörige Datenflüsse modelliert werden. Die Aufgabenzerlegung wird durch einen UND/ODER-Baum beschrieben. Die Aufgaben werden durch UND-Knoten und die (komplexen) Methoden[2] durch ODER-Knoten repräsentiert. Das bedeutet, daß die einer Methode zugeordneten **Aufgaben** bei der Anwendung der Methode alle bearbeitet werden müssen, während die verschiedenen **Methoden**, die einer Aufgabe zugeordnet sind, Alternativen darstellen um diese zu bearbeiten. Eine komplexe Methode wird im konzeptuellen Modell durch einen Datenflußgraphen beschrieben. Die Knoten sind Aufgaben, Variable (die Problemfallwissen repräsentieren) und Problemlösewissen[3]. Durch die Kanten wird der Datenfluß zwischen den Aufgaben beschrieben. Die Anfangsknoten von Kanten, die auf eine Aufgabe gerichtet sind, stellen Eingaben der Aufgabe dar, während die Endknoten, der von einer Aufgabe ausgehenden Kanten, Ausgaben der Aufgabe spezifizieren. Als Eingaben kommen Problemlösewissen in Form von Konzeptinstanzen und Problemfallwissen in Form von formalen Parametern, die als typisierte Variable fungieren, in Frage. Als Ausgaben können nur formale Parameter spezifiziert werden.

Wie bereits in Abschnitt 2.4 ausgeführt wurde, können diese Datenabhängigkeiten zwischen formalen Parametern und Aufgaben aber nur auf der obersten Ebene der Objektstruktur des jeweiligen formalen Parameters formuliert werden. Im folgenden wird beschrieben in welcher Weise das konzeptuelle Modell erweitert werden mußte, um diese Enschränkung aufzuheben. Zur Erläuterung der in diesem Abschnitt dargestellten Konzepte wird nach und nach ein Beispiel eingeführt. Das Beispiel soll die Erfassung von Personendaten modellieren. Abb. 3.1 zeigt zunächst den UND/ODER-Baum, der die Aufgabenzerlegung des Beipiels darstellt. Beispielsweise wird die Aufgabe *Personendaten erfassen* mit Hilfe der Methode *standard* in die (Teil-) aufgaben *postalische Daten* und *sonstige Daten* zerlegt.

[2]Im Aufgaben-/Methodenzerlegungsbaum werden die atomaren Methoden nicht explizit dargestellt.

[3]siehe auch Abschnitt 2.2.

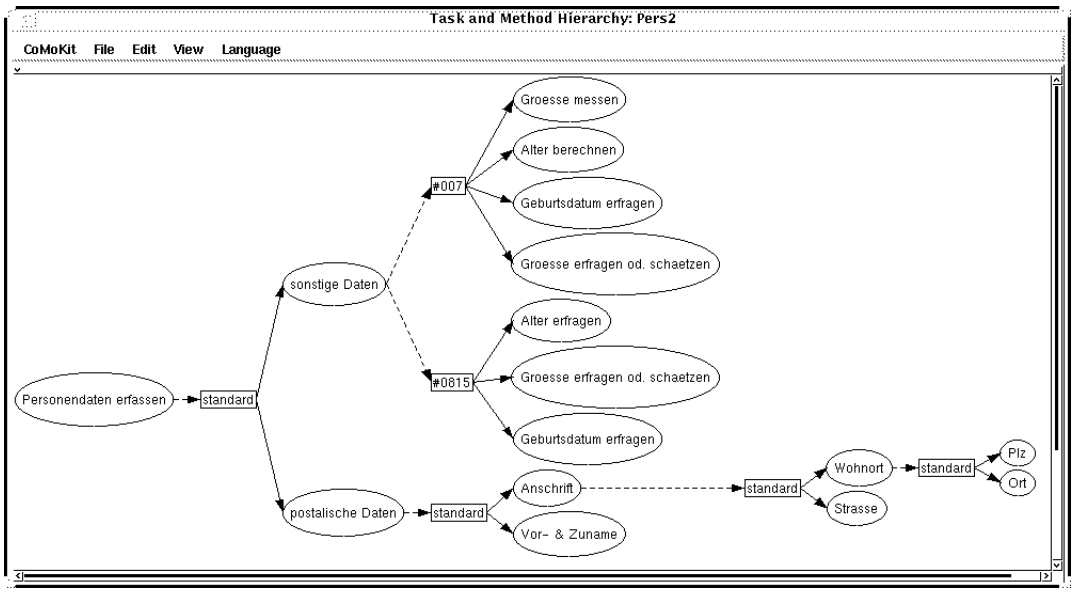


Abbildung 3.1: Der UND/ODER-Baum für das Beispiel zur Erfassung von Personendaten

#### 3.1.1.1 Pämisse bei der Modellierung von Datenflüssen

Ausgangspunkt der folgenden Überlegungen ist die Prämisse, daß Datenobjekte erst dann als Eingabe einer Aufgabe verwendet werden können, wenn sie bereits im Laufe der Problemlösung erzeugt worden sind. Problemlösewissen ist bereits zu Beginn des Problemlöseprozesses vorhanden, während das Problemfallwissen erst im Verlaufe desselben erzeugt wird. Das bedeutet bezogen auf das konzeptuelle Modell, daß die Modellierungsmöglichkeiten bzgl. der Spezifikation von formalen Parametern als Ein- bzw. Ausgabe von Aufgaben geeignet einzuschränken sind. An dieser Stelle soll die Einführung einiger nützlicher Begriffe erfolgen:

**Kontext** Die Kontexte eines Aufgabenzerlegungsbaumes sind gegeben durch dessen komplexe Methoden und die jeweils zu diesen gehörenden Datenflußgraphen. Der aktuelle Kontext bezeichnet denjenigen Kontext, der augenblicklich Gegenstand der Betrachtung ist.

**Kontexte einer Aufgabe** Die Kontexte einer Aufgabe sind diejenigen Kontexte, deren Methoden diese Aufgabe referenzieren. Der aktuelle Kontext einer Aufgabe bezeichnet einen bestimmten Kontext der Kontexte einer Aufgabe auf den sich die augenblickliche Betrachtung bezieht.

Die Datenflüsse der Methoden sind analog zu den Aufgaben bzw. Methoden der Aufgabenzerlegung hierarchisch strukturiert, das heißt mit fortschreitender Aufgabenzerlegung erfolgt eine sukzessive Verfeinerung der Datenflüsse. Aus dieser Beob-

achtung werden die Begriffe des abhängigen bzw. übergeordneten Kontexts abgeleitet:

**Abhängiger Kontext** Die abhängigen Kontexte eines Kontexts sind alle Kontexte, die sich im Unterbaum des Kontexts befinden.

**Übergeordneter Kontext** Die übergeordneten Kontexte des aktuellen Kontexts sind alle Kontexte, die sich auf dem Weg vom aktuellen Kontext zur Wurzel des UND/ODER-Baums befinden.

Bezogen auf die Erfüllung der Prämisse bedeutet dies, daß ein Datenobjekt, das als Eingabe einer Aufgabe spezifiziert werden soll, entweder in dem aktuellen Kontext der Aufgabe als Ausgabe einer anderen Aufgabe spezifiziert ist oder in einem übergeordneten Kontext des aktuellen Kontexts Ausgabe einer Aufgabe ist, die im UND/ODER-Baum nicht auf dem Weg vom aktuellen Kontext zur Wurzel liegt. Das heißt, die Aufgabe, die das Datenobjekt als Eingabe hat, gehört nicht zu dem Unterbaum der Aufgabe, die das Datenobjekt als Ausgabe hat. Beipielsweise benötigt man zur Berechnung des Alters einer Person deren Geburtsdatum und das Tagesdatum. Bezogen auf das Beispiel aus Abb. 3.1 bedeutet dies, daß Geburtsdatum entweder Ausgabe einer Aufgabe im Kontext der Methode *#007* ist oder Ausgabe der Aufgabe *postalische Daten* im Kontext der Methode *standard*[4], die zum einzigen übergeordneten Kontext von Methode *#007* gehört.

Diese Strategie findet unter anderem Eingang in die Mechanismen, die zur Konsistenzerhaltung des konzeptuellen Modells verwendet werden, worauf in Abschnitt 3.1.2 eingegangen werden soll.

#### 3.1.1.2 Der Parameterbaum

Das Problemfallwissen wird im konzeptuellen Modell durch formale Parameter spezifiziert. Einem formalen Parameter ist eine Konzeptklasse als Typ zugeordnet. Diese Konzeptklasse kann eine komplexe Struktur aufweisen. In der bisherigen Implementierung wurde bei der Definition eines solchen formalen Parameters lediglich dieser selbst und ein Verweis auf die als Typ spezifizierte Konzeptklasse in das konzeptuelle Modell aufgenommen. Jetzt soll in so einem Fall ein Parameterbaum erzeugt werden, indem zusätzlich für jede Unterstruktur der Konzeptklasse, welche als Typ spezifiziert wurde, ein formaler Parameter mit zugehörigem Verweis auf seinen Typ ins konzeptuelle Modell aufgenommen wird. Die Knoten des Baumes, die formale Parameter repräsentieren, sind durch gerichtete, beschriftete Kanten[5] analog zur Konzepthierarchie verknüpft. Die Beschriftung gibt Auskunft darüber, welches Attribut (Slot)

[4]Der Methodenname *standard* kommt im UND/ODER-Baum der Abb. 3.1 mehrfach vor, unter CoMo-Kit wird aber jedes Vorkommen der Methode als ein Objekt verwaltet, zu dem ein Datenfluß definiert wird.

[5]Diese Kanten werden im konzeptuellen Modell durch Formal-Parameter-Context-Links repräsentiert.

der zum Anfangsknoten gehörenden Konzeptklasse der Endknoten repräsentiert. An dieser Stelle sollen wieder einige Begriffe eingeführt werden:

**Pfad** Wenn man im Parameterbaum alle Beschriftungen der Kanten auf dem Weg von der Wurzel zu einem Knoten aufsammelt, erhält man den Pfad, den man durch den Parameterbaum gegangen ist.

**Spezialisierungen eines Knotens** Die Menge der Knoten, welche man durch Betrachtung aller Endknoten der von diesem Knoten des Parameterbaumes ausgehenden Kanten erhält, bezeichnet man als dessen Spezialisierungen.

**Generalisierung eines Knotens** Betrachtet man eine Kante des Parameterbaumes, so bezeichnet man deren Anfangsknoten als die Generalisierung von deren Endknoten.

Demnach hat die Wurzel des Parameterbaumes keine Generalisierung und die Blätter des Baumes haben keine Spezialisierung. Im Kontext von Methode *standard* zur Bearbeitung der Aufgabe *Personendaten erfassen*, wird jetzt durch Auswahl des entsprechenden Menuepunkts der formale Parameter *Person* mit der Konzeptklasse *PERSON* definiert. Dieser ist dann in allen abhängigen Kontexten zur Definition von Ein-/Ausgabestrukturen verfügbar (siehe auch Abschnitt 3.1.2). Im konzeptuellen Modell wird für den formalen Parameter *Person* entsprechend der Objektstruktur die Konzeptklasse *PERSON* der in Abb. 3.2 dargestellte Parameterbaum aufgebaut.

#### 3.1.1.3 Harte und weiche Links zur Spezifikation des Ein- bzw. Ausgabeverhaltens

An dieser Stelle sollen neue Link-Typen eingeführt werden, die die Visualisierung der im konzeptuellen Modell spezifizierten Datenflüsse unterstützen bzw. zusätzlich deren Operationalisierung ermöglichen. Die bisher im konzeptuellen Modell verwendeten Link-Typen zur Spezifikation von Ein- bzw. Ausgaben von Aufgaben bzw. Methoden sollen durch speziellere Link-Typen ersetzt bzw. ergänzt werden. Bei den bisher verwendeteten Link-Typen erfolgt jetzt eine Unterscheidung zwischen harten (solid) und weichen (soft) Links. Die harten Links dienen zur Spezifikation des Ein- bzw. Ausgabeverhaltens und übernehmen damit zur Laufzeit die Rolle der bisherigen Link-Typen. Die weichen Links hingegen finden lediglich im Rahmen der Modellierung von konzeptuellen Modellen Verwendung. Sie sollen den Modellierenden bei der Spezifikation von Ein- bzw. Ausgabestrukturen für einen formalen Parameter unterstützen, wenn er sich auf den verschiedenen Ebenen der Objektstruktur des formalen Parameters bewegt (näheres siehe Abschnitt 3.1.2.2).

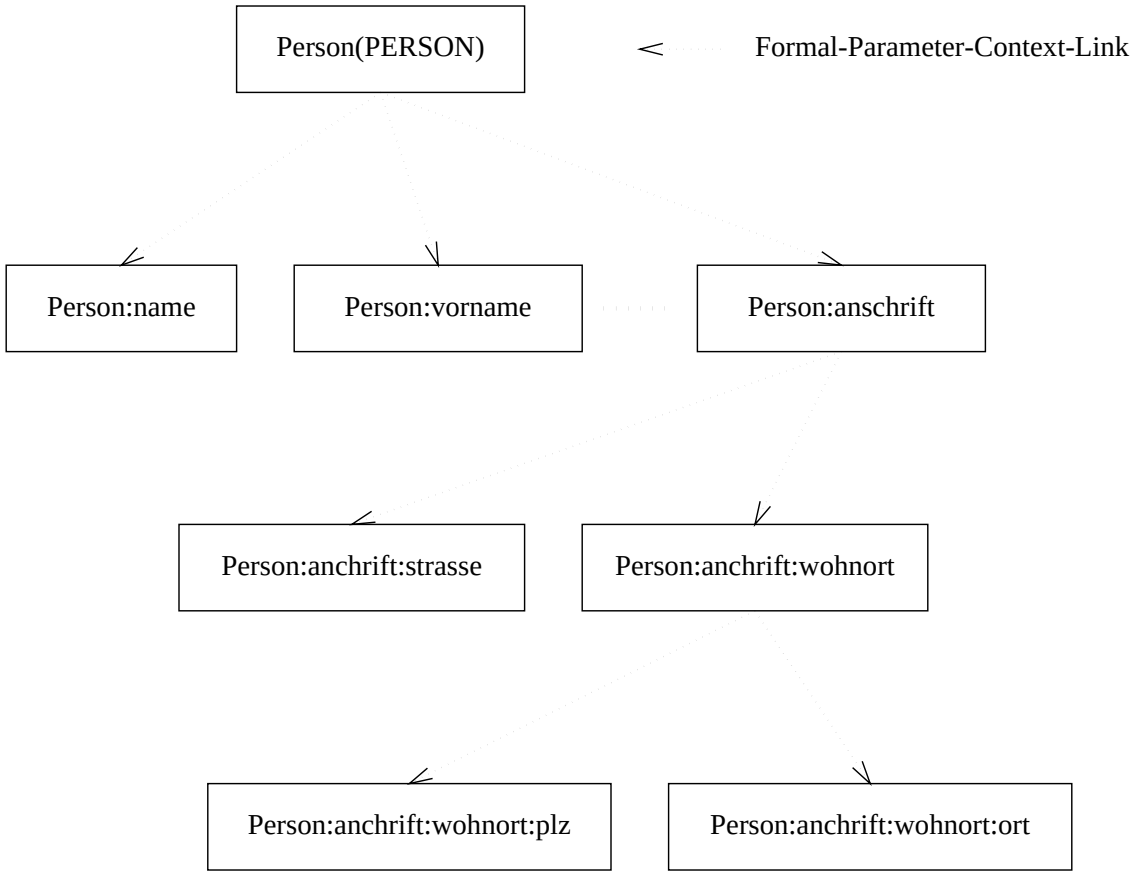


Abbildung 3.2: Der Parameterbaum des formalen Parameters *Person*

#### 3.1.1.4 Problemfälle bei der Modellierung von Datenflüssen

Bei der Modellierung von Datenflüssen kann man einige kritische Fälle identifizieren, da diese nicht eindeutig bzw. ohne Anpassung des Interpreters operationalisiert werden können oder keine sinnvollen Datenflüsse darstellen:

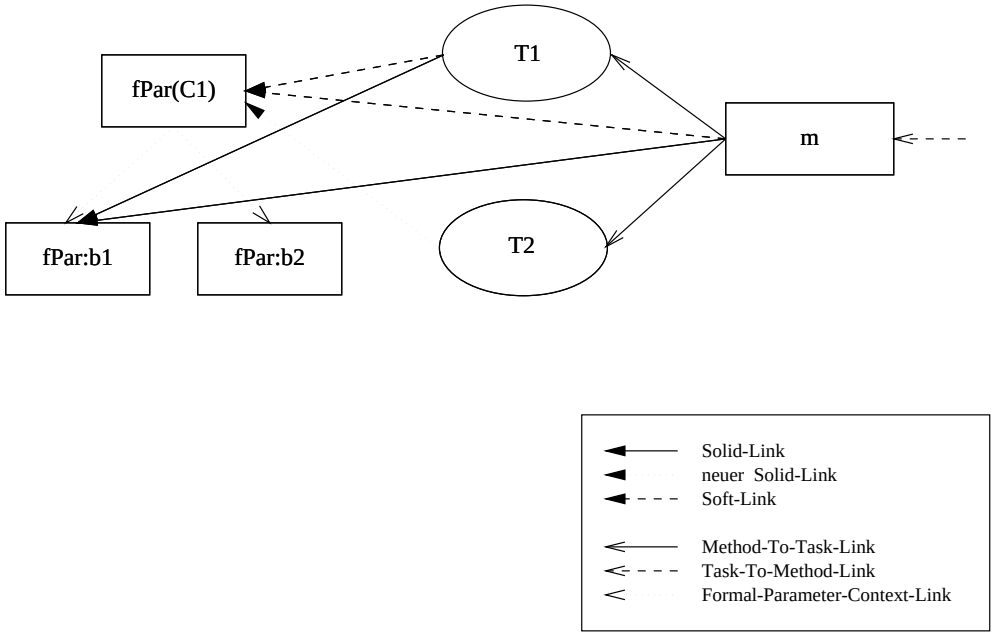


Abbildung 3.3: Erster Problemfall

1. In der im folgenden gegebenen Situation, wird ein formaler Parameter von zwei verschiedenen Aufgaben als Ausgabe spezifiziert (siehe Abb. 3.3): Der formale Parameter *fPar* soll als Ausgabe von Aufgabe $T_2$ definiert werden. Für Aufgabe $T_1$ ist aber der formale Parameter $fPar{:}b_1$ bereits Ausgabe. Darüber hinaus sollen keine zusätzlichen direkten oder indirekten Datenflußabhängigkeiten zwischen den beiden Aufgaben $T_1$ und $T_2$ bestehen. Es ergibt sich die folgende Problematik: Sowohl $T_1$ als auch $T_2$ müß bei der Anwendung von Methode $m$ bearbeitet werden. Da zwischen $T_1$ und $T_2$ nach Voraussetzung keine Datenflußabhängigkeiten bestehen, ist die Reihenfolge der Bearbeitung nicht festgelegt und je nachdem, welche der beiden Aufgaben zuerst bearbeitet wird, ergibt sich eine andere Ausgabe und die Ausgabe der zuerst bearbeiteten Aufgabe wird (teilweise) überschrieben ohne vorher in den Datenfluß einzugehen. Deshalb wäre es beispielsweise denkbar, daß die eigentlich intendierte Modellierung derjenigen von Abb. 3.4 entspricht.

2. Wir betrachten jetzt folgende Situation, in der der spezifizierte Datenfluß keine eindeutige Semantik hat (siehe Abb. 3.5): *fPar* ist Ausgabe von Aufgabe $T_1$,

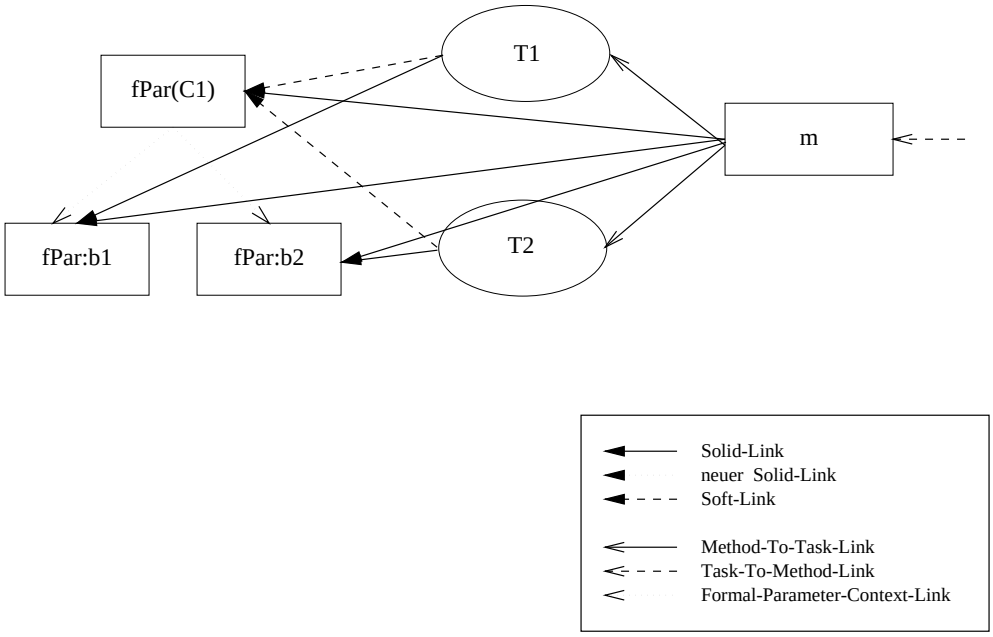


Abbildung 3.4: Intendierte Modellierung des ersten Problemfalles

Eingabe von Aufgabe $T_2$ und sowohl Eingabe als auch Ausgabe von Aufgabe $T_3$. Der Prämisse aus Abschnitt 3.1.1.1 folgend, daß ein formaler Parameter im Datenfluß Ausgabe einer Aufgabe sein muß, bevor er als Eingabe einer anderen Aufgabe verwendet werden kann, muß zunächst Aufgabe $T_1$ bearbeitet werden. Aber dann ergeben sich die beiden alternativen Möglichkeiten zur Interpretation des Datenflusses:

(a) Die Aufgaben $T_2$ und $T_3$ benutzen die Ausgabe von Aufgabe $T_1$ als Eingabe und können parallel bearbeitet werden. Aufgabe $T_3$ verändert dabei den Wert von *fPar*.

(b) Die Aufgabe $T_3$ benutzt zunächst die Ausgabe von Aufgabe $T_1$ als Eingabe und überschreibt die Ausgabe von Aufgabe $T_1$. Der neue Wert von *fPar* wird dann von Aufgabe $T_2$ als Eingabe benutzt.

Es ergibt sich also keine eindeutige Semantik des Datenflusses, weil $T_2$ in a) und b), falls $T_3$ in b) den Wert von *fPar* verändert, mit einer anderen Eingabe bearbeitet wird und deshalb evtl. eine andere Ausgabe liefert bzw. die Bearbeitung anderes verläuft.

3. Der jetzt dargestellten Ausgangssituation kann eine eindeutige Semantik zugeordnet werden (siehe Abb. 3.6): *fPar* ist Ausgabe von Aufgabe $T_1$ und sowohl Eingabe als auch Ausgabe von Aufgabe $T_2$. Im Sinne der in Abschnitt 3.1.1.1 eingeführten Prämisse, ergibt sich folgende Semantik des Datenflusses:

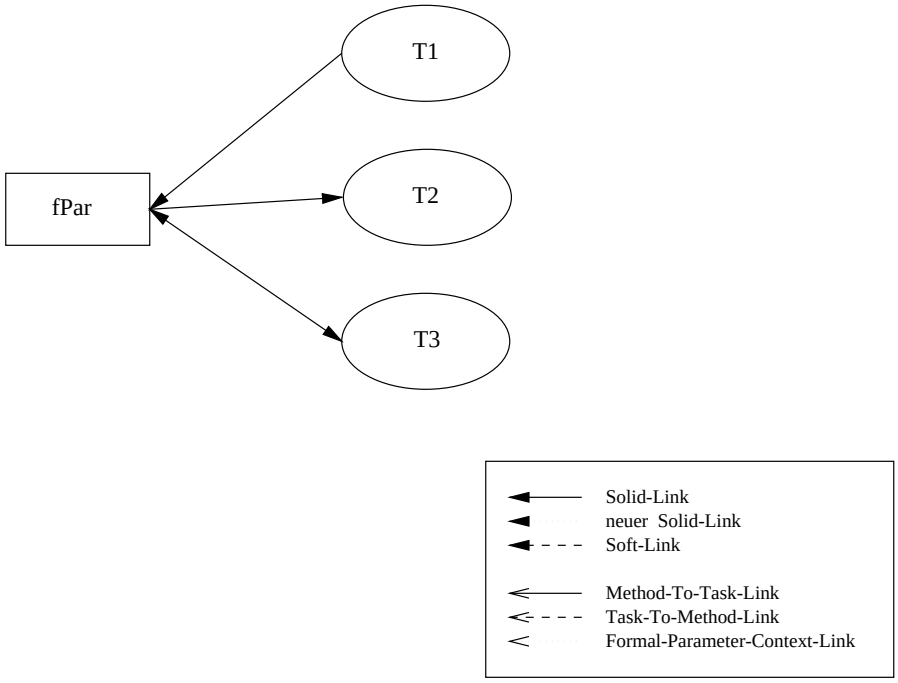


Abbildung 3.5: Zweiter Problemfall

(a) Aufgabe $T_1$ bearbeiten, liefert einen Wert für den formalen Parameter *fPar*.

(b) Aufgabe $T_2$ wird bearbeitet unter der Benutzung der Ausgabe von Aufgabe $T_1$ und verändert den Wert von *fPar*. Es ergibt sich die folgende Problematik: Zur Laufzeit muß die Ausgabe von Aufgabe $T_1$ ungültig werden, sobald die Ausgabe von Aufgabe $T_2$ gültig wird. Dazu ist eine Anpassung des Schedulers erforderlich (näheres siehe Abschnitt 3.1.3.4).

Die in 1)- 3) beispielhaft angeführten Datenflüsse deuten an, daß der sich aus dem Datenfluß unmittelbar ergebende Kontrollfluß nicht immer geeignet ist, um eine eindeutige bzw. sinnvolle Operationalisierung durch den Scheduler zu gewährleisten. Denkbar wäre eine Erweiterung von CoMo-Kit, die es erlaubt einen Kontrollfluß zu definieren, der unabhängig von dem sich durch den Datenfluß ergebenden Kontrollfluß ist. In [Jablonski, 1995] wird im Rahmen der abhängigkeitsbezogenen Aspekte des dort vorgestellten Referenzmodells eines Workflow-Management-Systems[6] auf diese Problematik eingegangen. Hier kann diese Problematik nicht vollständig bzw. zufriedenstellend gelöst werden, da dies den Rahmen einer Diplomarbeit sprengen würde. Deshalb werden zum Beispiel die in 1) und 2) beschriebenen Fälle bei der

[6]Dort wird ein Workflow-Management-System bzw. Aktivitäten-Management-System als ein System beschrieben, das zur Koordinierung von Benutzern dient, die räumlich verteilt an der Lösung von Aufgaben arbeiten.

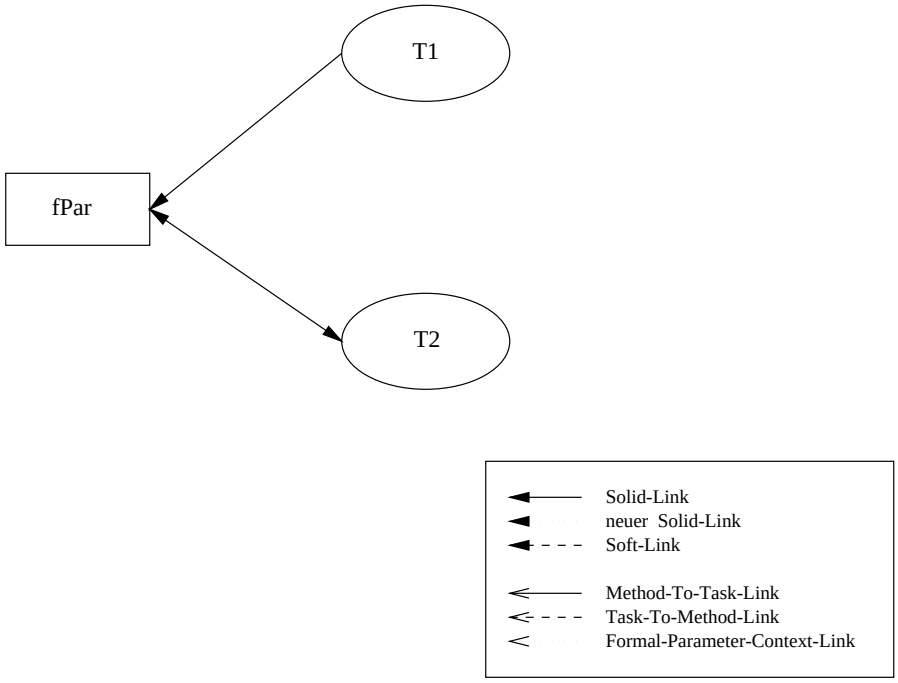


Abbildung 3.6: Dritter Problemfall

Modellierung von konzeptuellen Modellen nicht zugelassen (näheres siehe Abschnitt 3.1.2).

### 3.1.2 Anpassung der Modellierungswerkzeuge von CoMo-Kit

Die in Abschnitt 3.1.1 eingeführten Erweiterungen des konzeptuellen Modells machen eine Anpassung bzw. Erweiterung der Modellierungswerkzeuge von CoMo-Kit notwendig.

Wie bereits in Abschnitt 2.5 bei der Darstellung der Aufgabenstellung unter 1.b) formuliert wurde, erfolgt dies mit der Zielsetzung, den Modellierenden bei der Erstellung von konzeptuellen Modellen soweit wie möglich zu unterstützen. Dabei spielt die Konsistenzerhaltung des konzeptuellen Modells während des Modellierungsprozesses eine wichtige Rolle, da die Komplexität des Modellierungsprozesses durch die Möglichkeit für Unterstrukturen von komplexen Datenobjekten Datenflußbeziehungen zu formulieren, beträchtlich zugenommen hat.

Da der Modellierende bei komplexeren Aufgabenzerlegungen, die bei realen Anwendungen die Regel sind, dann mit der Konsistenzerhaltung des konzeptuellen Modells wahrscheinlich überfordert wäre, wird die Konsistenzerhaltung durch die Modellierungswerkzeuge übernommen.

Im folgenden werden nun die Mechanismen beschrieben, die aus der Erweiterung

des konzeptuellen Modells resultieren und die Konsistenzerhaltung desselben betreffen.

#### 3.1.2.1 Manipulation der Sichten auf einen Parameterbaum

Die Definition von formalen Parametern erfolgt im `MethodEditor` von CoMo-Kit durch die Auswahl des entsprechenden Menuepunkts. Daraufhin wird der Modellierende im einem Dialog aufgefordert den Namen und Typ des neu zu definierenden formalen Parameters zu spezifizieren. Als Typ eines formalen Parameters können (alle im konzeptuellen Modell bereits definierten) Konzeptklassen angegeben werden. Wie bereits in Abschnitt 3.1.1.2 beschrieben, wird anhand der zu einer Konzeptklasse evtl. gehörenden Konzepthierarchie ein Parameterbaum erzeugt. Dieser Parameterbaum wächst proportional zur Komplexität der jeweiligen Konzepthierarchie.

Um den Modellierenden nicht mit überflüssiger Information bei der Erstellung von Datenflüssen zu Methoden zu konfrontieren, soll er bei formalen Parametern jeweils nur eine Sicht auf diejenige Ebene der Objektstruktur des Parameters erhalten, welche für ihm momentan von Interesse ist. Die Manipulation der Sichten auf einen formalen Parameter kann durch den Modellierenden über die Menuepunkte `unfold formal Parameter` bzw. `fold formal Parameter` des `MethodEditors` erfolgen. Bei der Anwahl des Menuepunkts `unfold formal Parameter` wird der jeweils selektierte formale Parameter entfaltet, d. h. im `MethodEditor` wird der Parameter durch alle seine Kinder im Parameterbaum ersetzt, vorausgesetzt diese existieren. Bei der Anwendung des Menuepunkts `fold formal Parameter` wird der dem formalen Parameter übergeordnete Parameter in der Hierarchie des Parameterbaums sichtbar im `MethodEditor` zugunsten des zu ihm gehörenden Unterbaums. In den beiden Abb. 3.7 und 3.8 sind zwei Sichten auf den in Abb. 3.2 dargestellten Parameterbaum zu sehen.

#### 3.1.2.2 Verwendung von harten bzw. weichen Links bei der Modellierung

In Abschnitt 3.1.1.3 wurden harte und weiche Links als neue Link-Typen zur Spezifikation von Ein- bzw. Ausgabestrukturen eingeführt. Die Spezifikation von Ein- bzw. Ausgabestrukturen im Rahmen der Erstellung des zu einer Methode gehörenden Datenflußgraphen erfolgt durch die Auswahl des entsprechenden Menuepunkts im `MethodEditor`. Geschieht dies, muß sich der Modellierende entscheiden, ob der selektierte formale Parameter Eingabe oder Ausgabe der ebenfalls selektierten Aufgabe werden soll. Ist diese Entscheidung getroffen worden, wird anschliessend, aus Gründen der Konsistenzerhaltung des konzeptuellen Modells, überprüft, ob dieser formale Parameter überhaupt als Ein- bzw. Ausgabe der betroffenen Aufgabe zulässig ist. Auf die dabei zugrundegelegten Kriterien wird noch später in Abschnitt 3.1.2.3 eingegangen. Ist der formale Parameter als Ein- bzw. Ausgabe der jeweiligen Aufgabe zulässig, wird das konzeptuelle Modell in folgender Weise verändert:

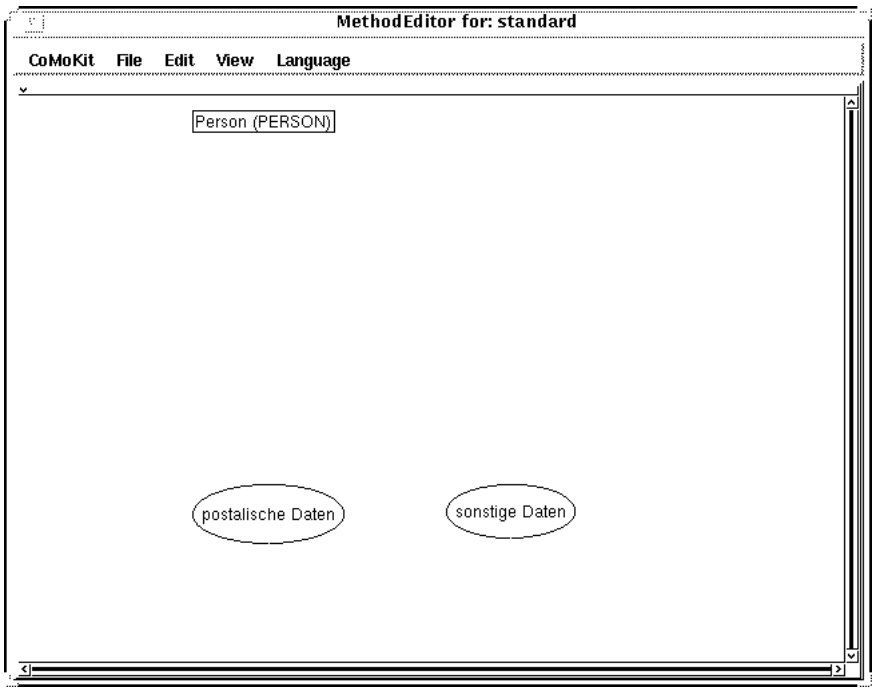


Abbildung 3.7: Sicht auf die oberste Ebene der Objektstruktur des formalen Parameters *Person*

1. Es wird ein harter Ein- bzw. Ausgabe-Link zwischen dem formalen Parameter und der betroffenen Aufgabe angelegt. Zusätzlich werden Links des gleichen Typs zwischen den zum Unterbaum des Parameters gehörenden formalen Parametern und der Aufgabe angelegt.

2. Darüber hinaus ist aus Gründen der Konsistenzerhaltung je nach Situation die Anlage von harten oder weichen Ein- bzw. Ausgabe-Links zwischen den dem jeweiligen Parameter im Parameterbaum übergeordneten formalen Parametern notwendig (näheres hierzu siehe auch Konsistenzüberlegungen, die noch später in Abschnitt 3.1.2.3 folgen).

In den Abb. 3.9 und 3.10 ist beispielhaft dargestellt, welche Link-Strukturen angelegt werden, wenn die formalen Parameter *Person (PERSON):name* und *Person (PERSON):vorname* als Ausgabe von Aufgabe *Vor- & Zuname* definiert werden.

#### 3.1.2.3 Konsistenzerhaltung des konzeptuellen Modells beim Hinzufügen von Ein-/Ausgabestrukturen

Bei der Definition von neuen Ein-/Ausgabestrukturen werden Aufgaben über Links mit formalen Parametern, die eine geschachtelte Objektstruktur aufweisen können, verknüpft. Wie im letzten Abschnitt 3.1.2.2 bereits ausgeführt, wird ein entsprechender Menuepunkt im `MethodEditor` zur Definition von neuen Ein-/Ausgabestrukturen

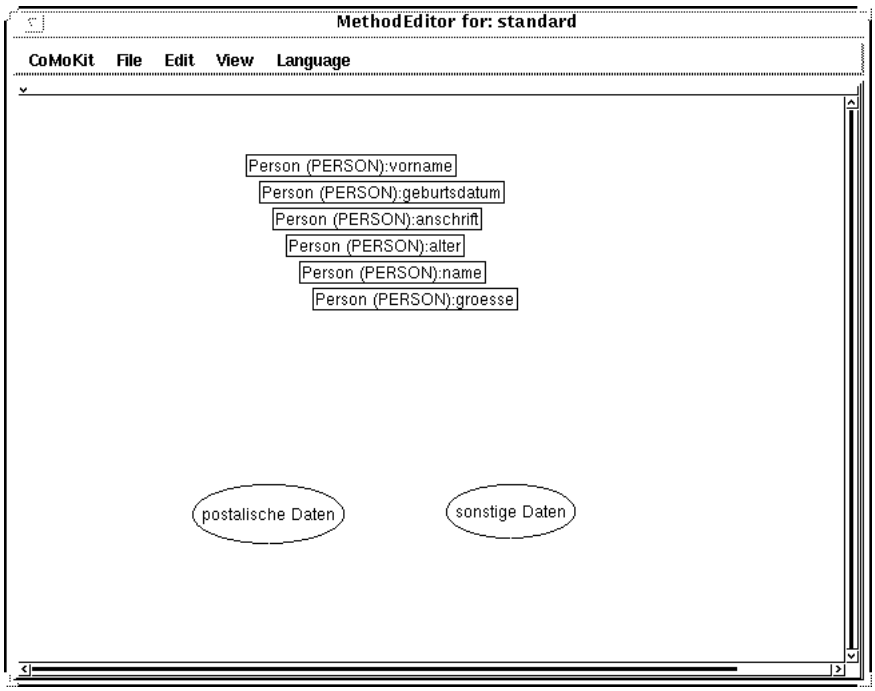


Abbildung 3.8: Sicht auf die unter der obersten liegende Ebene der Objektstruktur von *Person*

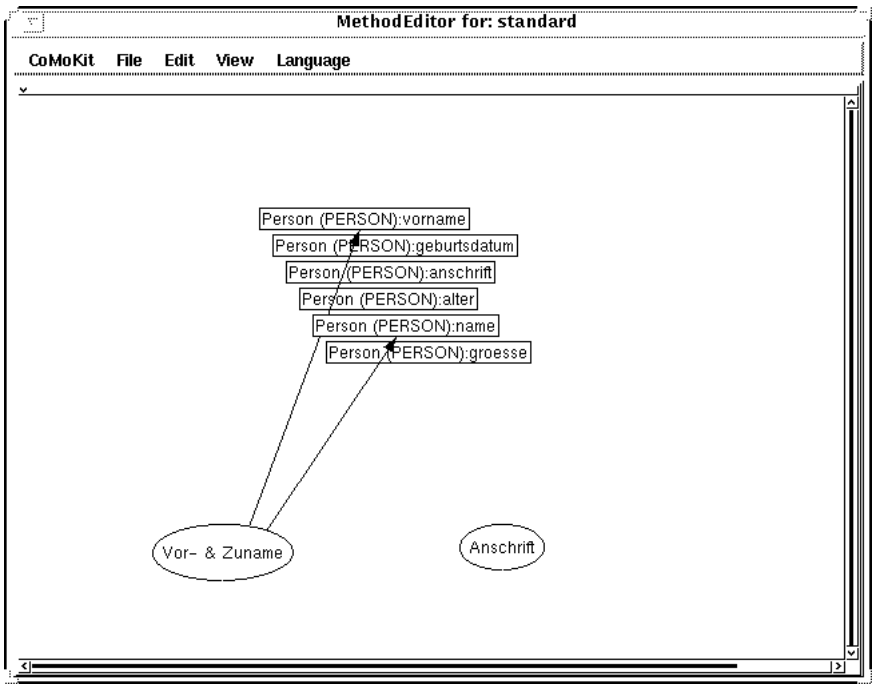


Abbildung 3.9: Darstellung der Link-Strukturen, die zur Spezifikation angelegt werden

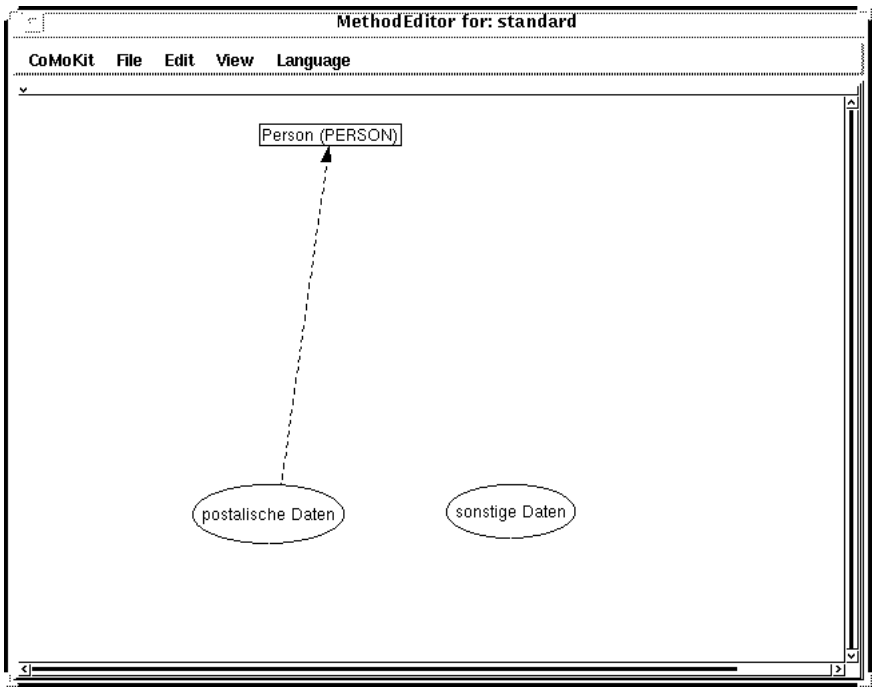


Abbildung 3.10: Darstellung der Link-Strukturen, die auf der obersten Ebene der Objektstruktur des formalen Parameters angelegt werden

verwendet. Dabei erfolgt eine Überprüfung der Zulässigkeit der spezifizierten Ein-/Ausgabestrukturen. Sind diese nicht zulässig, werden diese aus Gründen der Konsistenzerhaltung nicht in das konzeptuelle Modell aufgenommen und es erfolgt eine Fehlermeldung, die auf die Ursache der Inkonsistenz schließen läßt. In diesem Abschnitt wird nun auf die Mechanismen eingegangen, die zur Konsistenzerhaltung des konzeptuellen Modells bei der Definition von neuen Ein-/Ausgabestrukturen, benutzt werden.

An dieser Stelle soll der Begriff des Definitionskontextes eines formalen Parameters eingeführt werden:

**Definitionskontext eines formalen Parameters** Dieser bezeichnet denjenigen Kontext der Aufgabenzerlegung, in welchem der formale Parameter in das konzeptuelle Modell aufgenommen wurde. Für den Modellierenden ist ein formaler Parameter in seinem Definitionskontext und zusätzlich in allen vom Definitionskontext abhängigen Kontexten sichtbar.

Die Idee ist nun, daß bei der Definition von neuen Ein-/Ausgabestrukturen diese an den jeweiligen übergeordneten Kontext weitergereicht werden. Der Vorgang der Propagierung von Ein-/Ausgabestrukturen an übergeordnete Kontexte stoppt, wenn über die Zulässigkeit bzw. Nichtzulässigkeit der spezifizierten Ein-/Ausgabestrukturen entschieden werden kann. Die Propagierung von

Ein-/Ausgabestrukturen an übergeordnete Kontexte verfolgt also das Ziel der Konsistenzerhaltung des konzeptuellen Modells.

**Propagierung von Ein-/Ausgabestrukturen** Es erfolgt jetzt eine detailierte Darstellung der Propagierung von Ein-/Ausgabestrukturen:

- Die Propagierung erfolgt rekursiv, indem jeweils der übergeordnete Kontext eines Kontexts aufgefordert wird, die für diesen spezifizierten Ein-/Ausgabestrukturen zu übernehmen.

- Der Ausgangspunkt der Rekursion ist derjenige Kontext, in dem die Spezifikation der Ein-/Ausgabestukturen durch den Modellierenden erfolgt.

- Die Rekursion stoppt entweder wenn der Definitionskontext eines formalen Parameters erreicht ist oder wenn bereits im aktuellen Kontext entschieden werden kann, ob die Ein-/Ausgabestrukturen zulässig sind oder nicht.

- Kann im aktuellen Kontext eine Entscheidung bezüglich der Zulässigkeit bzw. Nichtzulässigkeit[7] der spezifizierten Ein-/Ausgabestrukturen getroffen werden, wird folgendermaßen vorgegangen:

  1. Falls die Definition der spezifizierten Ein-/Ausgabestrukturen im aktuellen Kontext nicht zulässig ist, werden diese dort nicht erzeugt und der betroffene abhängige Kontext erhält die Mitteilung über die Nichtzulässigkeit.

  2. Falls die Definition des spezifizierten Ein-/Ausgabeverhaltens im aktuellen Kontext zulässig ist, werden diese dort angelegt und der betroffene abhängige Kontext erhält die Mitteilung über die Zulässigkeit.

**Aufnahme neuer Ein-/Ausgabestrukturen in einen Kontext** Sollen neue Ein-/Ausgabestrukturen in einen Kontext aufgenommen werden, ergibt sich die folgende Vorgehensweise:

1. Die Zulässigkeit der spezifizierten Ein-/Ausgabestrukturen wird überprüft, falls diese nicht zulässig sind, dann STOP.

2. Ermittle diejenigen Links, die erzeugt werden müssen, unter Berücksichtigung der folgenden Konsistenzbedingungen für den Parameterbaum des formalen Parameters:

[7]Die Kriterien für die Un- bzw. Zulässigkeit von Eingaben, werden unter den Ausführungen zur Definition eines formalen Parameters als Eingabe aufgeführt. Und die Kriterien für die Un- bzw. Zulässigkeit von Ausgaben, werden unter den Ausführungen zur Definition eines formalen Parameters als Ausgabe dargestellt.

(a) Zur Vermeidung von Redundanz werden prinzipiell nur Ein-/Ausgabestrukturen in Form von Links in das konzeptuelle Modell aufgenommen, die nicht schon existieren. Sollen harte Ein-/Ausgabestrukturen definiert werden, für die schon entsprechende weiche Ein-/Ausgabestrukturen existieren, werden die weichen Ein-/Ausgabestrukturen gelöscht.

(b) Es werden ebenfalls entsprechende Ein-/Ausgabestrukturen in Form von harten Links für den kompletten zum formalen Parameter gehörenden Unterbaum erzeugt.

(c) Falls nicht für alle übrigen Spezialisierungen der Generalisierung des formalen Parameters entprechende harte Ein-/Ausgabestrukturen existieren, muß, falls dieser noch nicht existiert, ein entsprechender weicher Link für die Generalisierung angelegt werden.

(d) Falls für alle übrigen Spezialisierungen der Generalisierung des formalen Parameters bereits entsprechende harte Ein-/Ausgabestrukturen existieren, wird ein entsprechender harter Link für die Generalisierung des formalen Parameters erzeugt und ein evtl. schon existierender entsprechender weicher Link wird gelöscht.

(e) Die unter c) und d) beschriebenen Vorgehensweisen setzen sich unter Umständen über mehrere Ebenen des Parameterbaums fort.

3. Erzeuge die unter 2) ermittelten Links für die mit den Ein-/Ausgabestrukturen spezifizierte Aufgabe.

4. Erzeuge entsprechende Links für die Methoden, die laut der Aufgabenzerlegung zur Lösung der betreffenden Aufgabe angewendet werden können.

Hier erfolgt eine Darstellung der Konsistenzbedingungen, die bei der Überprüfung der Zulässigkeit von Ein- bzw. Ausgabestrukturen in einem bestimmten Kontext verwendet werden.

**Definition eines formalen Parameters als Eingabe** Zunächst soll der formale Parameter *fPar* als Eingabe einer Aufgabe im aktuellen Kontext definiert werden. Dabei müssen folgende Fälle unterschieden werden:

1. In den folgenden beiden Fällen ist *fPar* als Eingabe der jeweiligen Aufgabe nicht zulässig. Deshalb erfolgt eine Fehlermeldung, der entsprechende Link wird nicht angelegt und der Aufruf zur Spezifikation von *fPar* als Eingabe einer Aufgabe in diesem Kontext liefert deren Unzulässigkeit zurück:

   (a) Der in Abschnitt 3.1.1.4 unter 2) beschriebene Fall liegt vor.

   (b) Es existiert bereits ein entsprechender Link zwischen *fPar* und der betreffenden Aufgabe (siehe Abb. 3.11).

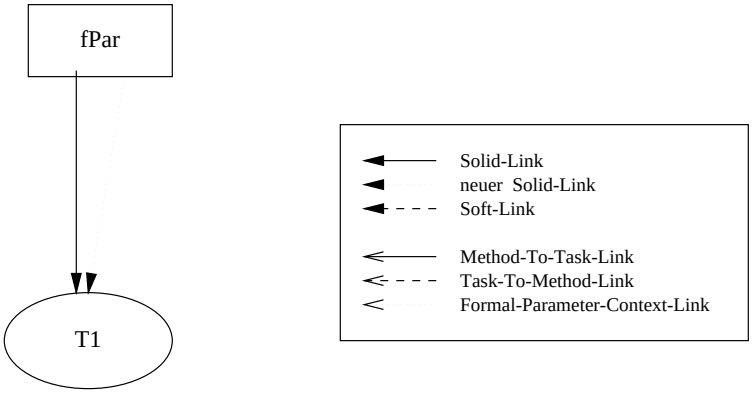


Abbildung 3.11: Der spezifizierte Link existiert bereits

2. Soll die Spezifikation von *fPar* als Eingabe in dessen Definitionskontext stattfinden, werden die folgenden beiden Situationen unterschieden:

   (a) Es gibt keine andere Aufgabe im Definitionskontext, die *fPar* als Ausgabe hat. Dann wird kein Link angelegt, es erfolgt eine entsprechende Fehlermeldung und der Aufruf liefert die Unzulässigkeit zurück.

   (b) Es existiert eine andere Aufgabe im Definitionskontext, die *fPar* als Ausgabe hat (siehe Abb. 3.12). In diesem Fall wird der Link, wie oben dargestellt, angelegt und der Aufruf liefert die Zulässigkeit zurück.

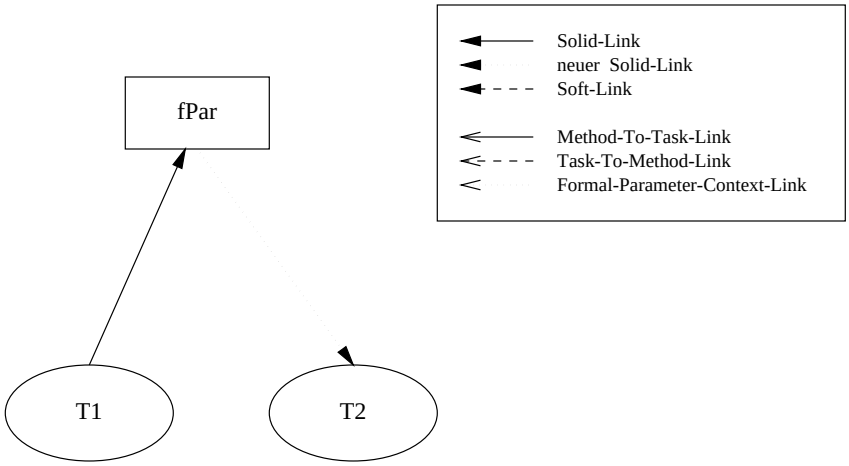


Abbildung 3.12: Eine andere Aufgabe im Definitionskontext hat *fPar* als Ausgabe

3. Liegen nicht die unter 1) und 2) beschriebenen Fälle vor, sind wiederum folgende drei Fälle zu unterscheiden:

(a) Der formale Parameter *fPar* ist weder Eingabe noch Ausgabe der zum Kontext gehörenden Methode. Dann erfolgt ein rekursiver Aufruf zur Überprüfung der Zulässigkeit. Dies geschieht dadurch, daß für die Aufgabe, die durch Anwendung der Methode des aktuellen Kontexts, gelöst wird in dem übergeordneten Kontext die Zulässigkeit von *fPar* als Eingabe überprüft wird. Liefert der rekursive Aufruf die Zulässigkeit, wird der Link angelegt und der Aufruf für den aktuellen Kontext gibt ebenfalls die Zulässigkeit zurück. Andernfalls wird der Link nicht angelegt und der Aufruf liefert die Unzulässigkeit zurück.

(b) Ist *fPar* nur als Ausgabe der Methode des aktuellen Kontexts spezifiziert, dann gibt es zwei Möglichkeiten:

i. Existiert im aktuellen Kontext keine andere Aufgabe, die *fPar* als Ausgabe hat (siehe Abb. 3.13), dann wird der Link nicht angelegt, es erfolgt eine Fehlermeldung und der Aufruf liefert die Unzulässigkeit zurück.

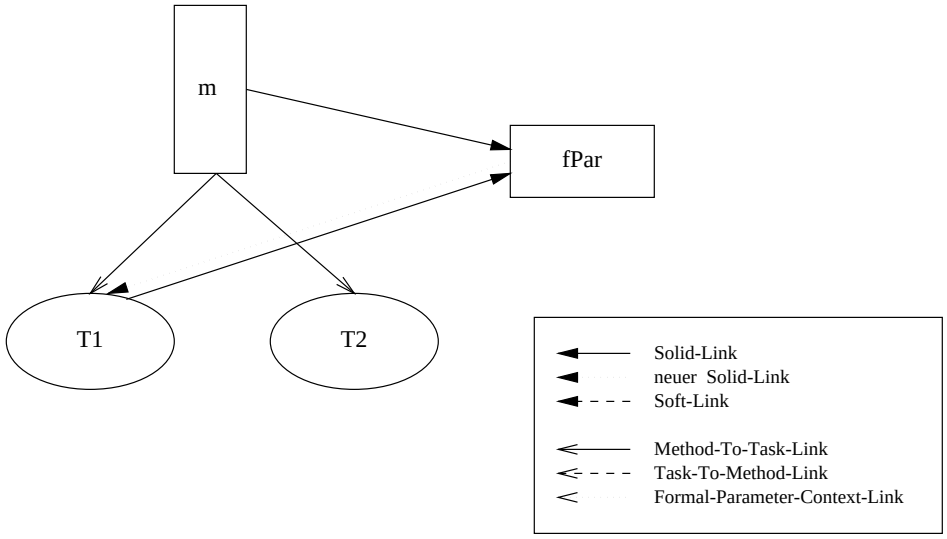


Abbildung 3.13: Der formale Parameter ist für keine andere Aufgabe als Ausgabe spezifiziert

ii. Existiert eine solche Aufgabe doch (siehe Abb. 3.14), wird der Link angelegt und die Zulässigkeit wird zurückgegeben.

(c) Falls *fPar* mindestens Eingabe und evtl. zusätzlich Ausgabe des aktuellen Kontexts ist, wird der Link angelegt und der Aufruf gibt die Zulässigkeit zurück.

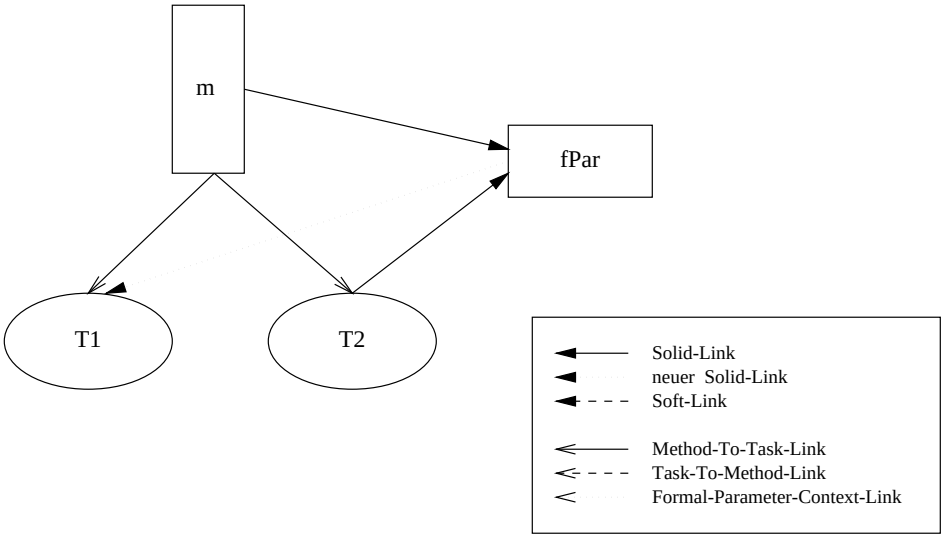


Abbildung 3.14: Der formale Parameter ist für eine andere Aufgabe als Ausgabe spezifiziert

**Definition eines formalen Parameters als Ausgabe** Jetzt soll der formale Parameter *fPar* als Ausgabe einer Aufgabe in einem bestimmten Kontext definiert werden. Dann werden die folgenden Fälle unterschieden:

1. In den folgenden drei Fällen wird kein entsprechender Link angelegt, es erfolgt eine geeignete Fehlermeldung und der Aufruf gibt die Unzulässigkeit zurück:
   (a) Der formale Parameter *fPar* ist nicht schon Eingabe derjenigen Aufgabe für die er als Ausgabe spezifiziert werden soll und es existiert noch eine andere Aufgabe im aktuellen Kontext, die *fPar* als Ausgabe hat (siehe Abb. 3.15).
   (b) Außer der Aufgabe für die *fPar* als Ausgabe spezifiziert werden soll, existiert noch eine weitere Aufgabe im aktuellen Kontext, die *fPar* als Eingabe hat (siehe Abb. 3.16).
   (c) Der formale Parameter *fPar* ist bereits Ausgabe der betreffenden Aufgabe.
2. Es erfolgt kein rekursiver Aufruf, der Link wird angelegt und die Zulässigkeit wird zurückgegeben, falls folgende Situationen vorliegen:
   (a) Der aktuelle Kontext ist der Definitionskonext von *fPar*.
   (b) *fPar* ist bereits Ausgabe der zum aktuellen Kontext gehörenden Methode.
3. Falls ein rekursiver Aufruf erfolgt, dann werden folgende Fälle unterschieden:
   (a) Der rekursive Aufruf ist erfolgreich, dann wird der Link im aktuellen Kontext erzeugt und die Zulässigkeit für den aktuellen Aufruf zurückgegeben.

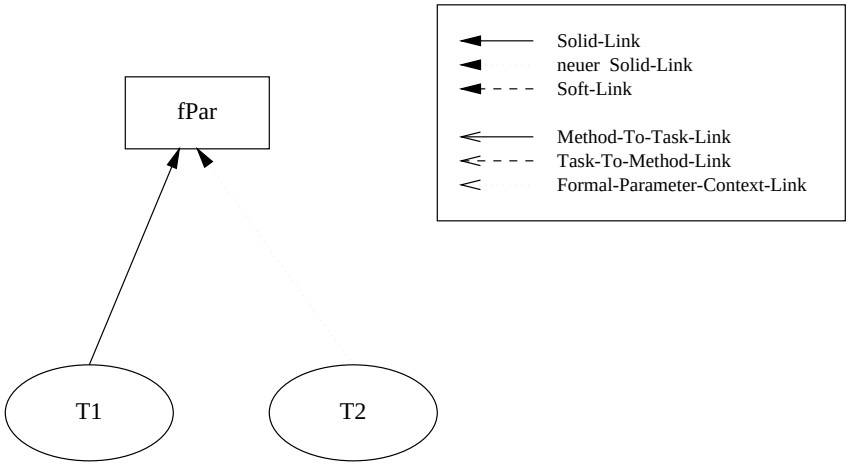


Abbildung 3.15: Der formale Parameter ist nicht schon Eingabe der Aufgabe

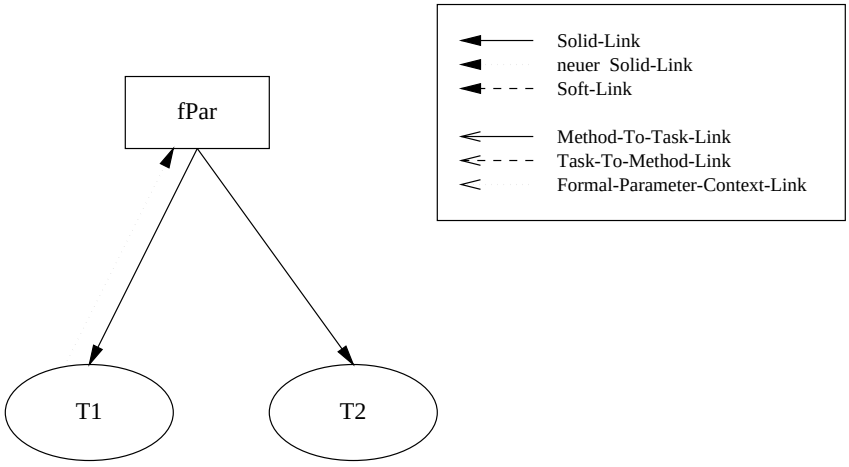


Abbildung 3.16: Es existiert noch eine weitere Aufgabe, die den formalen Parameter als Eingabe hat

(b) Ist rekursive Aufruf nicht erfolgreich, dann wird kein Link angelegt und die Unzulässigkeit zurückgegeben.

#### 3.1.2.4 Konsistenzerhaltung des konzeptuellen Modells beim Löschen von Ein-/Ausgabestrukturen

Das Löschen von Ein-/Ausgabestrukturen erfolgt im `MethodEditor` durch Auswahl des entsprechenden Menuepunkts. Auch das Löschen von Ein-/Ausgabestrukturen soll ausgeschlossen werden, wenn es zu einer Inkonsistenz des konzeptuellen Modells führt. Im folgenden wird die prinzipielle Vorgehensweise beim Löschen von Ein-/Ausgabestrukturen beschrieben:

- Sollen Ein-/Ausgabestrukturen aus dem aktuellen Kontext gelöscht werden, wird zunächst die Zulässigkeit des Löschvorgangs geprüft. Nicht zugelassen sind Löschvorgänge, die zu Situationen führen, wie sie in Abschnitt 3.1.1.4 unter 1) und 2) beschrieben werden.

- Die zu löschenden Ein-/Ausgabestrukturen werden ausgehend vom aktuellen Kontext ebenfalls in allen abhängigen Kontexten gelöscht.

- Beim Löschen von Ein-/Ausgabestrukturen muß darauf geachtet werden, daß die für den Parameterbaum des betreffenden formalen Parameters im jeweiligen Kontext spezifizierten Ein-/Ausgabestrukturen durch den Löschvorgang nicht inkonsistent werden[8].

[8] Vgl. dazu die Konsistenzerhaltung des Parameterbaums bei den Erläuterungen zur Vorgehensweise bei der Aufnahme von neuen Ein-/Ausgabestrukturen in einen Kontext unter 2) in Abschnitt 3.1.2.3.

### 3.1.3 Operationalisierung der Erweiterung des konzeptuellen Modells

Wie bereits in Abschnitt 2.4 dargestellt wurde, erfolgt die Operationalisierung der konzeptuellen Modelle durch die Interpreterkomponente von CoMo-Kit. Die in Abschnitt 3.1.1 beschriebene Erweiterung des konzeptuellen Modells erfordert, wie bereits bei der Darstellung der Aufgabenstellung in Abschnitt 2.5 erläutert, eine Anpassung bzw. Erweiterung der Mechanismen der Interpreterkomponente, wobei diese hauptsächlich den Scheduler betreffen.

#### 3.1.3.1 Probleme

Im einzelnen sind bei der Operationalisierung folgende Probleme zu lösen:

1. Erzeugung der Objektstrukturen und zugehörigen Abhängigkeitsstrukturen im TMS während des Lösungsprozesses.

2. Aufbau bzw. Abbau von Objektstrukturen, der sich durch Gültig- bzw. Ungültigwerden der zugehörigen Abhängigkeitstrukturen im TMS ergibt. Der Aufbau von Objektstrukturen geschieht in der Form, daß beispielsweise neu erzeugte Instanzen eines formalen Parameters den entsprechenden Slots ihres Rahmenobjekts (d. h. der Generalisierung des formalen Parameters) zugewiesen werden. Beim Abbau von Objektstrukturen werden die ungültig gewordenen Instanzen eines formalen Parameters aus den Slots ihres Rahmenobjekts entfernt, indem der betreffende Slot auf `nil` gesetzt wird.

#### 3.1.3.2 Alternativen bei der Erzeugung von Objektstrukturen

Bei der Erzeugung von Objektstrukturen bieten sich prinzipiell zwei Möglichkeiten an:

1. Mit dem Fortschreiten des Lösungsprozesses werden die nach einer Strategie jeweils notwendigen Objektstrukturen sukzessive erzeugt.

2. Die Objektstrukturen werden erst dann erzeugt, wenn sie im Lösungsprozeß benötigt werden.

In der der Arbeit zugrundeliegenden Implementierung wurde die erste der beiden Möglichkeiten gewählt. Die Entscheidung für diesen Ansatz erfolgte aus den folgenden Gründen:

- Es ist eine natürlichere Behandlung von rekursiven Datenstrukturen im Lösungsprozeß möglich, da die jeweiligen Objektstrukturen begleitend zu jedem Rekursionsschritt erzeugt werden können[9].

[9]Die Modellierung von rekursiven Datenstrukturen ist unter CoMo-Kit zwar möglich, die Verwendung von diesen macht jedoch keinen Sinn, da bisher keine Möglichkeit besteht rekursive Auf-

- Die Erzeugung von Objektstrukturen orientiert sich an der Hierarchie der Datenflüsse des konzeptuellen Modells. Deshalb kann man sich eine Strategie für die sukzessive Erzeugung von Objektstrukturen zurechtlegen, so daß man jeweils nicht mehr überprüfen muß, ob evtl. schon Teile der Objekthierarchie, d. h. formale Parameter in Form von Knoten des jeweiligen Parameterbaums, erzeugt worden sind.

#### 3.1.3.3 Strategie zur sukzessiven Erzeugung von Objektstrukturen

Jetzt soll die Strategie vorgestellt werden, mit deren Hilfe entschieden wird, welche Objektstrukturen beim aktuellen Stand der Problemlösung zu erzeugen sind:

1. Wird ein atomarer Operator zur Bearbeitung einer Aufgabe angewendet, werden folgende Schritte durchgeführt:

   (a) Zunächst werden Instanzen derjenigen formalen Parameter zur Verfügung gestellt, die Ausgabe der entsprechenden atomaren Aufgabe bzw. Methode im konzeptuellen Modell sind.

   (b) Dann wird die atomare Aufgabe durch einen Agenten abgearbeitet und die für die Ausgaben erzeugten Instanzen evtl. mit Werten belegt.

   (c) Zum Schluß werden *assignment*-Knoten für die oben erzeugten Instanzen der formalen Parameter erzeugt und diese durch die zum atomaren Operator gehörende Entscheidung gerechtfertigt. Falls formale Parameter sowohl als Eingaben als auch Ausgaben des Operators spezifiziert sind, müssen hier zusätzliche Abhängigkeitsstrukturen aufgebaut werden[10].

2. Erfolgt die Auswahl eines komplexen Operators zur Bearbeitung einer Aufgabe, werden jeweils für bestimmte formale Parameter neue Instanzen erzeugt. Dabei findet die folgende Vorgehensweise zur Bestimmung der Menge *NEWINST*, die diejenigen formalen Parameter enthält für die neue Instanzen erzeugt werden müssen, Anwendung:

   (a) Zunächst werden diejenigen Ausgaben des Operators anhand des konzeptuellen Modells ermittelt und in *NEWINST* aufgenommen, welche nicht zugleich Eingaben sind[11].

   (b) Anschließend werden alle Objektstrukturen in die Menge *NEWINST* aufgenommen, die im Kontext der zum Operator gehörenden Methode neu definiert werden und in den Datenfluß des Kontexts eingehen.

---

gabenstrukturen zu modellieren. In Kapitel 5 wird nochmals auf dieses Thema im Rahmen offener Probleme eingegangen.

[10] Näheres hierzu siehe Ausführungen zum Problemfall aus Abschnitt 3.1.1.4 in Abschnitt 3.1.3.4.

[11] Der Grund hierfür ergibt sich aus der Prämisse, die in Abschnitt 3.1.1.1 eingeführt wurde: Objektstrukturen, die als Eingabe einer Aufgabe bzw. Methode spezifiziert sind, sind zu dem Zeitpunkt, da die Aufgabe zur Bearbeitung ansteht bzw. die Methode angewendet wird, bereits vorhanden, weil sie zuvor Ausgabe einer anderen Aufgabe waren.

(c) Aus der Menge *NEWINST* werden dann alle Objektstrukturen entfernt, welche als Ausgabe mindestens einer Aufgabe des Kontextes spezifiziert sind.

(d) Im letzten Schritt werden für alle Elemente aus *NEWINST* die entsprechenden Instanzen erzeugt und die notwendigen Abhängigkeitsstrukturen im TMS aufgebaut. Insbesondere wird der zur Instanz gehörende *assignment*-Knoten durch die zum Operator gehörende Entscheidung gerechtfertigt. Damit ist die Erzeugung und Gültigkeit von Objektstrukturen immer an bestimmte Entscheidungen gebunden[12]. Sobald dann die zur Instanz gehörende Entscheidung gültig wird, wird auch die Instanz gültig und diese muß ihrem jeweiligen Rahmenobjekt zugewiesen werden.

#### 3.1.3.4 Operationalisierung eines Problemfalls

Im folgenden soll auf die Operationalisierung des in Abschnitt 3.1.1.4 vorgestellten Problemfalls eingegangen werden (siehe Abb. 3.17):

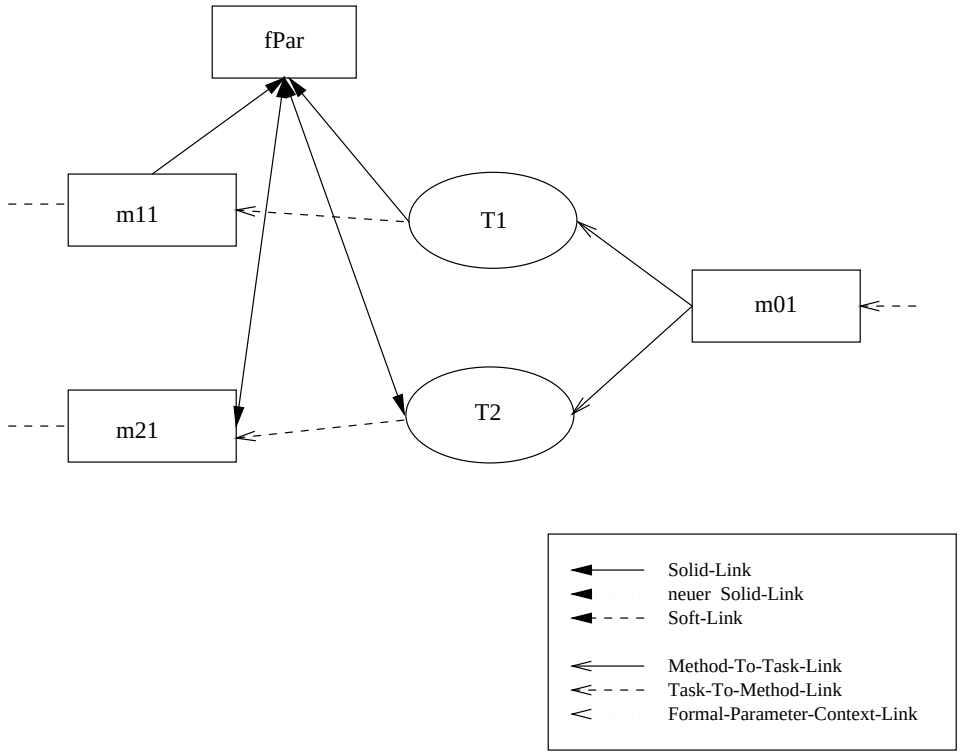


Abbildung 3.17: Der Problemfall

Aufgabe $T_1$ liefert als Ausgabe eine Instanz des formalen Parameters *fPar*, diese

[12]Näheres zu den im TMS aufgebauten Abhängigkeitsstrukturen und deren Verwendung siehe auch Abschnitt 2.3.3.

wird von Aufgabe $T_2$ als Eingabe benutzt. Aufgabe $T_2$ hat ebenfalls den formalen Parameter *fPar* als Ausgabe und erzeugt, für den Fall daß sie atomar ist, eine weitere Instanz des formalen Parameters *fPar*, die jetzt der Wert von *fPar* ist. Diese Abhängigkeit von Entscheidungen untereinander bzw. zwischen Entscheidungen und der Gültigkeit von Werten eines formalen Parameters müssen vom Scheduler verwaltet werden. In der dieser Arbeit zugrundeliegenden Implementierung wurden die Abhängigkeiten zum Zwecke der Operationalisierung wie folgt auf TMS-Strukturen abgebildet (siehe auch Abb. 3.18):

1. Der *rejected-decision*-Knoten wird nicht mehr über die *assignment*-Knoten, welche der Methode ($method_{21}$) als Eingabe (*assignment*-Knoten für $fPar = v_1$) dienen, gerechtfertigt, sondern von den Enscheidungen (*decision*-Knoten für $method_{11}$), die ebendiese *assignment*-Knoten rechtfertigen. Der Grund für diese Änderung ist, daß man vermeiden will, daß der *decision*-Knoten für die Methode $m_{21}$ ebenfalls ungültig wird, wenn der *assignment*-Knoten für $fPar = v_1$ ungültig wird. Diese Lösung hat den Nachteil, daß im Falle der Existenz von mehreren Rechtfertigungen für einen *assignment*-Knoten, diese alle entsprechend bei dem *rejected-decision*-Knoten als Rechtfertigungen eingetragen werden müßten.

2. Für den Fall, daß $T_2$ eine atomare Aufgabe ist, d. h. die zu $T_2$ gehörende Methode $m_{21}$ ist eine atomare Methode, müssen noch TMS-Strukturen mit der folgenden Wirkung angelegt werden: Sobald der Wert von *fPar* überschrieben wird, d. h. sobald die Entscheidung für Methode $m_{21}$ gültig wird, muß der alte Wert von *fPar* ungültig werden, d. h. der *assignment*-Knoten für $fPar = v_1$ wird ungültig.

3. Der Aufbau der TMS-Strukturen unter 2) entfällt, falls $m_{21}$ eine komplexe Methode ist, weil dann nicht nochmals durch deren Anwendung eine neue Instanz von *fPar* erzeugt wird (siehe Abschnitt 3.1.3.3).

Im Gegensatz zu [Dellen, 1995] wird hier, unter den einschränkenden Bedingungen des Problemfalls aus Abb. 3.17, zugelassen, daß sowohl atomare als auch komplexe Aufgaben eine Variable mit verschiedenen Werten belegen. Im Problemfall wird für den zugehörigen Datenfluß eine bestimmte Semantik unterstellt, durch die festgelegt wird, welche Belegung der Variablen gültig sein muß. Die Abhängigkeiten zwischen den Entscheidungen selbst und zwischen diesen und den verschiedenen Belegungen der Variablen werden auf geeignete TMS-Strukturen abgebildet.

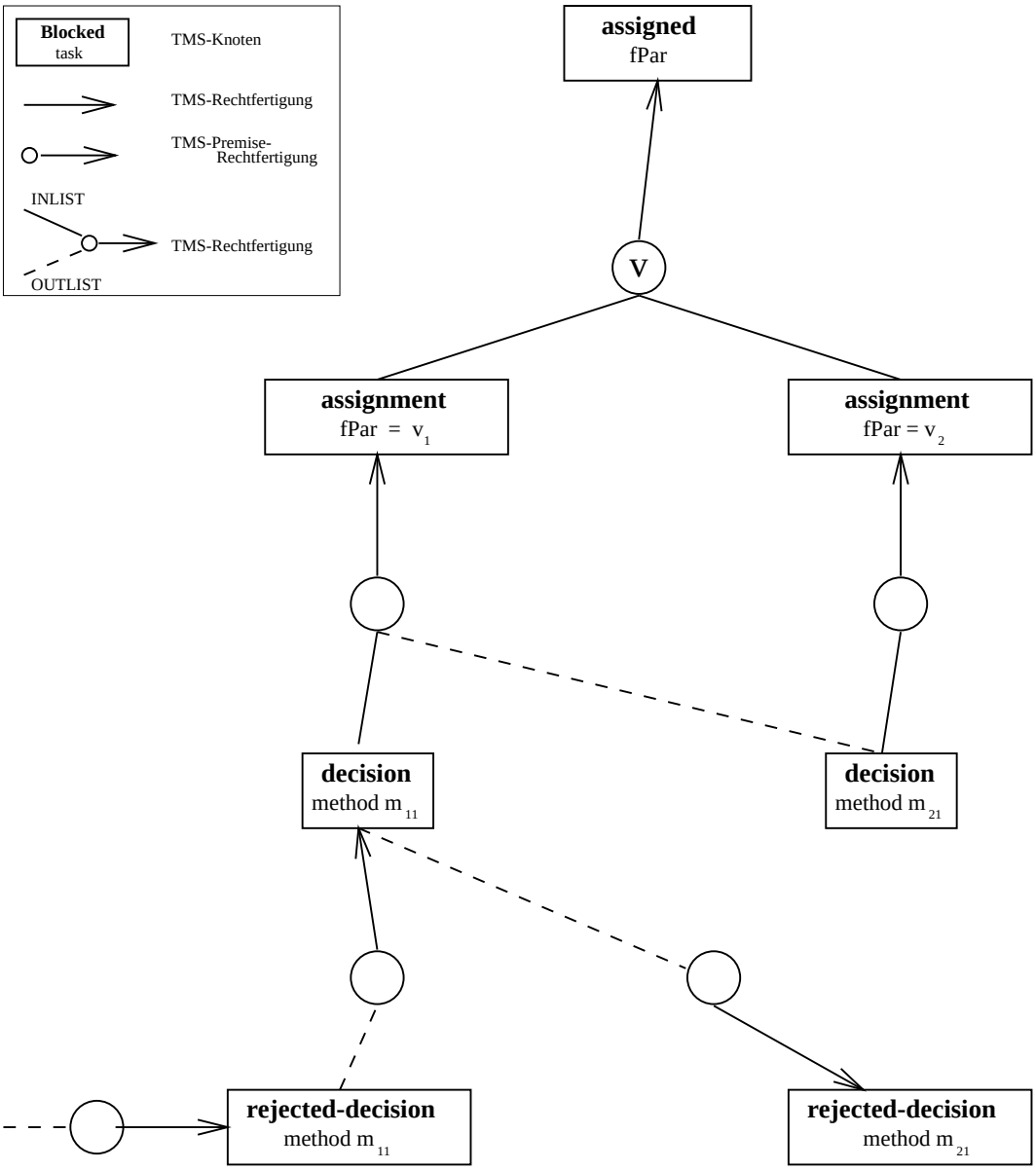


Abbildung 3.18: TMS-Strukturen zur Operationalisierung des Problemfalls

## 3.2 Unabhängigkeit der Delegierungsentscheidungen vom Vorliegen der Eingaben der Aufgabe

Um jederzeit zum Zwecke der Vorausplanung Delegierungsentscheidungen bei Aufgaben durchführen zu können, die durch Anwendung von komplexen Methoden gelöst werden, wird die Abhängigkeitsverwaltung solcher Aufgaben derart angepaßt, daß diese unabhängig vom Vorliegen der im konzeptuellen Modell spezifizierten Eingaben abgearbeitet werden können. Für die Implementierung bedeutet dies folgendes:

1. Über den *executable*-Knoten einer Aufgabe wird gesteuert, ob diese vom Scheduler zur Bearbeitung freigegeben wird. Erst wenn der *executable*-Knoten einer Aufgabe gültig ist, kann diese zur Bearbeitung freigegeben werden. Unabhängig davon, ob die jeweilige Aufgabe atomar oder komplex ist, wird dieser in der bisherigen Implementierung dann gültig, wenn der *valid-delegation*-Knoten gültig ist und alle Eingaben der Aufgabe vorliegen (siehe Abb. 2.5). Werden die Eingaben als Rechtfertigungen des *executable*-Knotens von komplexen Aufgaben weggelassen, können diese unabhängig vom Vorliegen der spezifizierten Eingaben bearbeitet werden. Insbesondere können dann Delegierungsentscheidungen für deren Teilaufgaben getroffen werden.

2. Aufgrund der in Abschnitt 3.1.1 beschriebenen Erweiterung des konzeptuellen Modells ergibt sich darüber hinaus folgende Änderung beim Aufbau der Abhängigkeitsstrukturen des *executable*-Knotens: Als Rechtfertigungen des *executable*-Knotens von atomaren Aufgaben dienen nur diejenigen Eingaben, die atomare formale Parameter[13] repräsentieren.

[13] Atomare formale Parameter sind diejenigen Parameter im Parameterbaum, die keine Spezialisierung haben, also Blätter sind.

# Kapitel 4

# Demonstration einiger Konzepte am Beispiel

In diesem Kapitel sollen einige der in Kapitel 3 dargestellten Konzepte unter Verwendung des dort eingeführten Beipiels demonstriert werden.

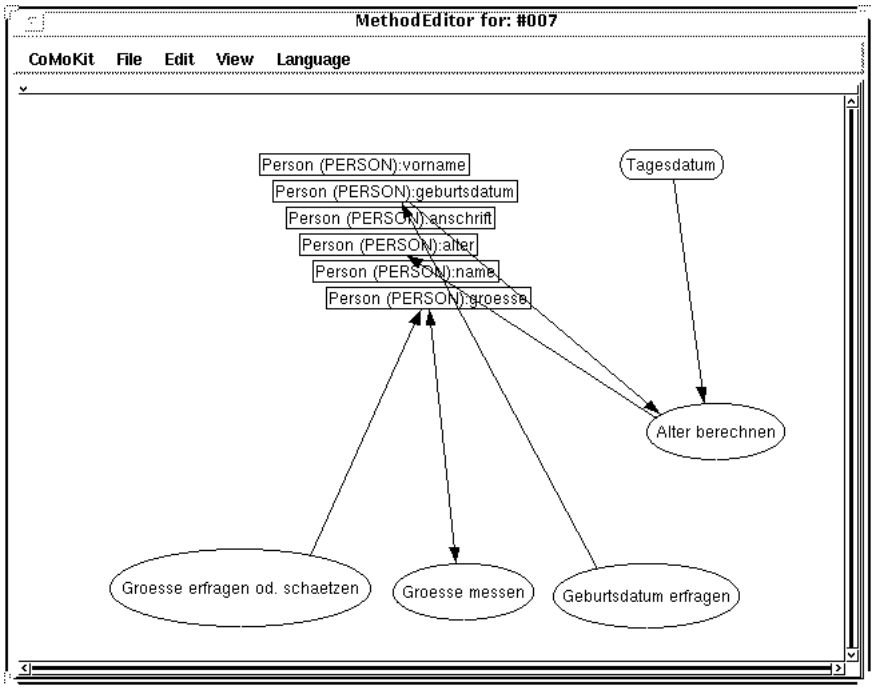


Abbildung 4.1: Definition des Datenflusses zu Methode *#007*

## 4.1 Propagierung von Ein-/Ausgabestrukturen an übergeordnete Kontexte

Zunächst einmal erfolgt eine vollständige Modellierung der für das Beispiel vorgesehenen Datenflüsse. Mit Hilfe der Methode *#007* und dem zugehörigen Datenfluß werden die Größe, das Geburtsdatum und das Alter der Person ermittelt (siehe Abb. 4.1): Der jeweilige Mitarbeiter kann zum Beispiel zunächst die Größe ermitteln, indem er diese von der Person erfragt oder kurzerhand schätzt, weil dadurch die anschließende Messung der Größe erleichtert wird. Anschließend kann er das Geburtsdatum erfragen und unter Zuhilfenahme des Tagesdatums das Alter der Person berechnen.

Die Abb. 4.2 zeigt, daß der Datenfluß der für Methode *#007* definiert wurde in den Datenfluß der Methode *standard* übernommen wurde.

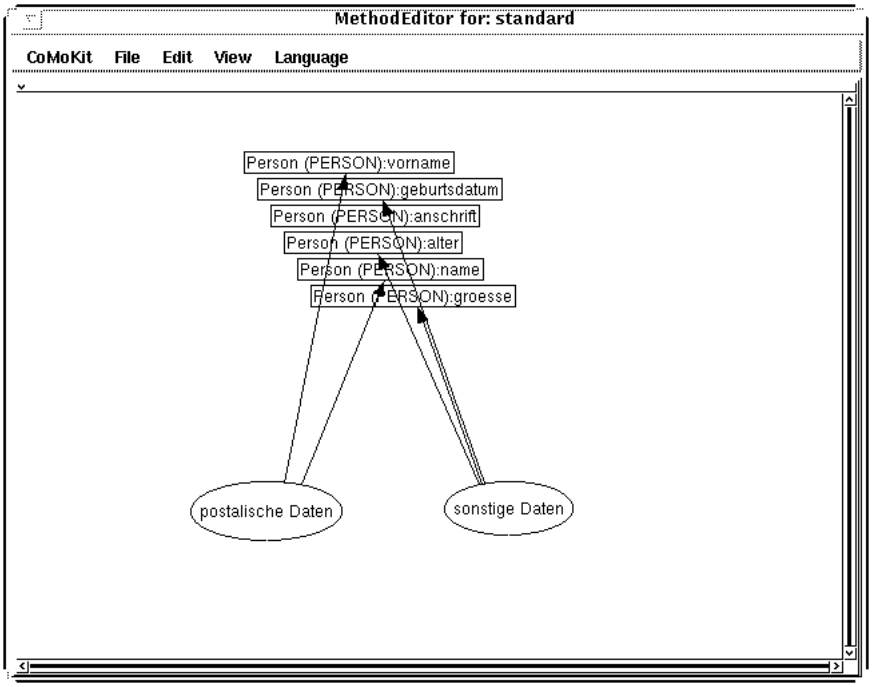


Abbildung 4.2: Der in Methode *#007* definierte Datenfluß ist in den Datenfluß von Methode *standard* übernommen worden

Die Erfassung der postalischen Daten erfolgt mit Hilfe der Methode *standard* und dem zugehörigen Datenfluß (siehe Abb. 4.3). Dort ist jetzt der formale Parameter *Person (PERSON):anschrift* als Ausgabe der Aufgabe *Anschrift* definiert worden.

Die Abb. 4.4 zeigt wiederum, daß der neue Datenfluß aus dem abhängigen Kontext übernommen worden ist.

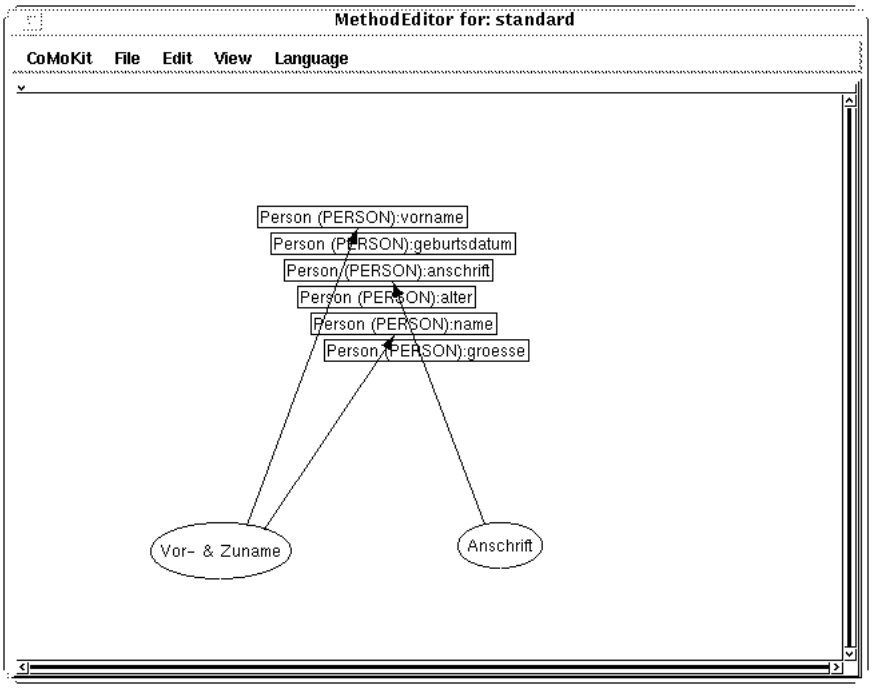


Abbildung 4.3: Der formale Parameter *Person(PERSON):anschrift* ist als Ausgabe der Aufgabe *Anschrift* definiert worden

## 4.2 Sukzessive Erzeugung von Objektstrukturen

Jetzt soll noch beispielhaft auf die in Abschnitt 3.1.3.3 vorgestellte Strategie zur sukzessiven Erzeugung von Objektstrukturen während des Lösungsprozesses eingegangen werden:

1. Zur Bearbeitung der Aufgabe *Personendaten erfassen* wird die komplexe Methode *standard* ausgewählt. Für den in diesem Kontext neu definierten formalen Parameter *Person (PERSON)* sind alle Knoten des Unterbaumes als Ausgabe von Aufgabe *postalische Daten* bzw. *sonstige Daten* definiert. Deshalb wird nur für den formalen Parameter *Person (PERSON)* eine Konzeptinstanz der Klasse *PERSON* erzeugt.

2. Bei der Bearbeitung der Aufgabe *postalische Daten* wird die komplexe Methode *standard* angewendet. Es werden keine Objekte erzeugt, da alle Ausgaben von Methode *standard* auch Ausgaben der Aufgaben *Vor- & Zuname* bzw. *Anschrift* sind.

3. Bei der Bearbeitung von Aufgabe *sonstige Daten* werden aus den gleichen Gründen wie unter 2) angeführt keine Objekte erzeugt. Diese werden erst bei der Anwendung der zu den atomaren Aufgaben gehörenden atomaren Methoden erzeugt. Dabei wird sowohl von Aufgabe *Groesse erfragen od. schaetzen*

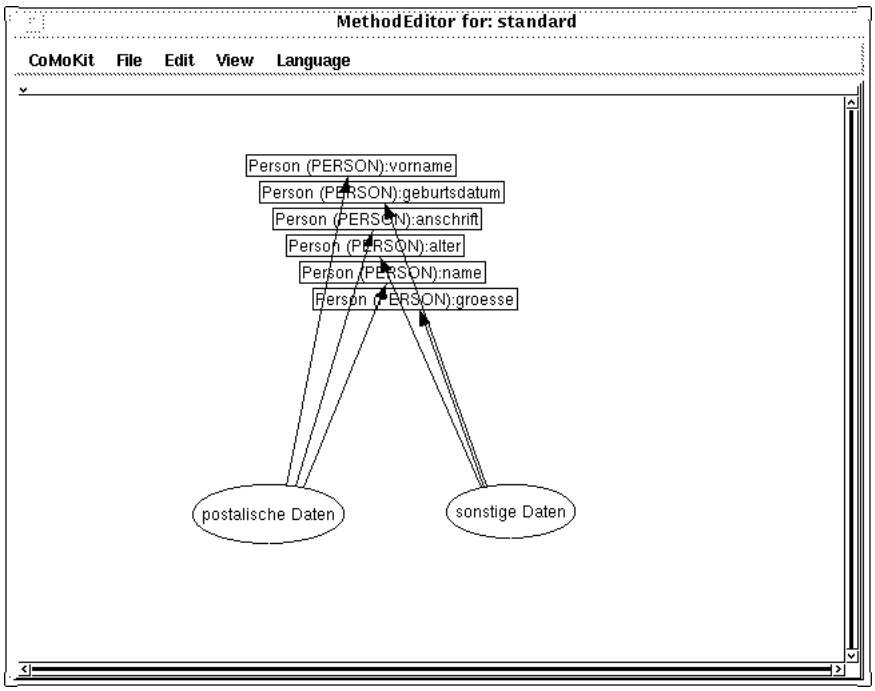


Abbildung 4.4: Die Datenflüsse aus den abhängigen Kontexten sind wieder übernommen worden

als auch von Aufgabe *Groesse messen* eine Instanz des formalen Parameters *Person(PERSON):groesse* erzeugt (siehe Abschnitt 3.1.3.4).

# Kapitel 5

# Zusammenfassung und Ausblick

In diesem Kapitel erfolgt zunächst eine Zusammenfassung der Ergebnisse der Arbeit. Anschließend wird ein Ausblick auf mögliche Weiterentwicklungen von CoMo-Kit gegeben.

## 5.1 Zusammenfassung

Der Gegenstand dieser Arbeit ist die Anpassung bzw. Erweiterung von CoMo-Kit, um die folgende Funktionalität des Systems zu erreichen:

1. Für die in Datenflüssen vorkommenden Objekte mit komplexer Objektstruktur sollen Ein-/Ausgabestrukturen definiert werden können, die sich auf einer beliebigen Ebene der jeweiligen Objektstruktur befinden.

2. Für komplexe Aufgaben können unabhängig vom Vorliegen der im konzeptuellen Modell spezifizierten Eingaben Delegierungsentscheidungen für die jeweils zugehörigen Teilaufgaben getroffen werden, während dies bisher nur beim Vorliegen der Eingaben möglich war.

Diese Forderungen an die Funktionalität des Systems ergeben sich zum einen aus der Tatsache, daß die in realen Arbeitsabläufen auftretenden Datenobjekte meist eine komplexe Struktur haben und zum anderen, daß im Rahmen der Abarbeitung komplexer Arbeitsabläufe Vorausplanung möglich sein muß, indem man Delegierungsentscheidungen trifft, ohne daß bereits alle zur Bearbeitung der Aufgabe notwendigen Daten vorliegen.

Die unter 1) dargestellten Anforderungen werden durch eine Erweiterung des konzeptuellen Modells erreicht, die eine Anpassung bzw. Erweiterung der Modellierungswerkzeuge von CoMo-Kit erfordert und entsprechend durch den Interpreter operationalisiert werden muß. Wie bereits in Abschnitt 3.1.3.2 erwähnt erlaubt CoMo-Kit die Definition von rekursiven Datenstrukturen, diese können jedoch bisher noch nicht sinnvoll im konzeptuellen Modell verwendet werden. Darüber hinaus übernehmen die Methoden einer Aufgabe weiterhin deren Ein-/Ausgabeverhalten (siehe

[Schmitz, 1994]). Das heißt, es können keine Domänen durch ein konzeptuelles Modell modelliert werden, die Methoden enthalten, deren Ein-/Ausgabeverhalten von dem der zugehörigen Aufgabe abweicht.

## 5.2 Ausblick

In diesem Abschnitt werden mögliche Weiterentwicklungen des Systems beschrieben, die unter anderem auf die in Abschnitt 5.1 dargestellten Probleme eingehen.

### 5.2.1 Einführung von Makros in das konzeptuelle Modell

Die Einführung von Makros[1] in das konzeptuelle Modell ermöglicht unter anderem eine vereinfachte Beschreibung der Operationalisierung von Aufgaben, Methoden und Datenflüssen.

Man könnte Makros zur Modellierung von Kontrollstrukturen im konzeptuellen Modell einsetzen. Beispielhaft dafür werden hier folgende Makros angeführt:

1. Ein Makro, das zur Spezifizierung der Bearbeitung von rekursiven Datenstrukturen dient, damit diese vom Scheduler operationalisiert werden können.

2. Ein Makro, das die Operationalisierung einer beliebig oft auszuführenden Aufgabe ermöglicht. Als reale Anwendung sei hier ein Beispiel aus der Bebauungsplan angeführt (siehe [Maurer F., 1995]): In einem Bebauuungsgebiet soll die Eintragung beliebig vieler Straßen möglich sein.

Darüber hinaus ist die Modellierung von Methoden sinnvoll, die die Aufgabenzerlegungen, die zu einzelnen Methoden einer Aufgabe gehören, kombinieren bzw. diese teilweise übernehmen. Als Anwendung wird hier wieder ein Beispiel aus der Bebauungsplanung angeführt: Man kann für eine Grundstücksfläche eine textliche, zeichnerische oder eine aus diesen kombinierte Festsetzung treffen. Ist bespielsweise schon eine zeichnerische Festsetzung für eine Fläche getroffen worden, die aber nicht vollständig ist, soll diese durch eine textliche Festsetzung ergänzt werden können. Das heißt die Lösung, die durch Anwendung der Methode für die zeichnerische Festsetzung entstanden ist, wird bei der Anwendung der Methode, die eine kombinierte Festsetzung ermöglicht, übernommen und durch die textliche Festsetzung ergänzt.

[1] Ein Beispiel für ein Makro ist eine komplexe Aufgabe, durch die zugehörigen komplexen Methoden sind alternative Möglichkeiten für deren Abarbeitung gegeben, indem die zu den komplexen Methoden gehörenden Datenflußgraphen durch den Interpreter operationalisiert werden. Eine komplexe Aufgabe stellt also eine abkürzende Schreibweise im konzeptuellen Modell dar, die operationalisiert werden kann.

### 5.2.2 Unterscheidung von Planbarkeit und Ausführbarkeit einer Aufgabe

In der jetzigen Implementierung ist die Ausführbarkeit einer komplexen Aufgabe unabhängig vom Vorliegen der im konzeptuellen Modell spezifizierten Eingaben (siehe Abschnitt 3.2). Die Ausführung einer komplexen Aufgabe geschieht durch Anwendung einer ihrer komplexen Methoden, die die Aufgabe in Teilaufgaben zerlegt, welche an entsprechende Agenten delegiert werden müssen. Somit ist die Delegierung an die Methodenanwendung gebunden. Es wäre aber sinnvoll, wenn man Delegierungsentscheidungen zurücknehmen könnte, ohne daß dies Auswirkungen auf bereits getroffene Entscheidungen für Methoden und dabei erstellte Lösungen hat. Dies macht eine Entkopplung von Methodenanwendung und Delegierung notwendig.

Darüber hinaus ist eine Unterscheidung in Planbarkeit und Ausführbarkeit einer Aufgabe sinnvoll. Dabei wird die Planbarkeit bzw. Ausführbarkeit einer Aufgabe über deren Methoden bestimmt:

1. Eine Aufgabe ist planbar, falls eine ihrer Methoden anwendbar ist bezüglich Planung.

2. Eine Aufgabe ist ausführbar, falls eine Methode anwendbar ist bezüglich Ausführbarkeit.

Für die Ausführbarkeit einer Aufgabe bzw. Anwendbarkeit einer Methode in der Ausführungsphase gilt:

1. Bei atomaren Methoden werden die Datenflußabhängigkeiten ausschließlich über Design-Rationales[2] definiert.

2. Bei komplexen Methoden werden die Datenflußabhängigkeiten definiert über:

   (a) Design-Rationales.

   (b) Einer Untermenge der Eingaben der zugehörigen Aufgabe, die die Obermenge darstellen, von denen die Anwendbarkeit ihrer Methoden abhängt.

Hingegen gilt für die Anwendbarkeit einer Methode in der Planungsphase, daß deren Anwendbarkeit unabhängig vom Vorliegen der im konzeptuellen Modell spezifizierten Eingaben ist. Darüber hinaus können bei der Planung und Ausführung verschiedene Delegierungsentscheidungen getroffen werden.

[2]Durch die Einführung von Design-Rationales sollen Argumente für oder gegen eine Methode angegeben werden können. Dabei werden drei Typen von Design-Rationales verwendet: Textuelle Begründungen, Begründungen durch andere Entscheidungen und Eingabedaten. Näheres siehe auch [Kohler, 1995].

# Literaturverzeichnis


Dellen, B. 1995 (Jan.). *Operationalisierung von konzeptuellen Modellen.* Diplomarbeit, Universität Kaiserslautern, Fachbereich Informatik, Kaiserslautern.

Dellen, B., Maurer, F. und Paulokat, J. 1995. *Verwaltung von Abhängigkeiten in kooperativen wissensbasierten Arbeitsabläufen.* Veröffentlichung auf der XPS-95.

Doyle, J. 1979. A Truth Maintenance System. *Journal on Artificial Intelligence*, **12**(3), 231–272.

Gebhardt, F., Gross, E., Hemmann, Th. und Voss, H. 1995. Knowledge Engineering mit MoMo. *KI 1/95, interdata Verlag.*

Jablonski, S. 1995. Workflow-Management-Systeme: Motivation, Modellierung, Architektur. *Informatik Spektrum*, **18**, 13–24.

Kohler, K. 1995. *Design Rationale bei der Modellierung und Abwicklung von Entwurfsprozessen.* Diplomarbeit, Universität Kaiserslautern, Fachbereich Informatik, Kaiserslautern.

Maurer, Frank. 1993. *Hypermediabasiertes Knowledge-Engeneering für verteilte wissensbasierte Systeme.* Dissertation, Universität Kaiserslautern, Kaiserslautern.

Maurer F., Pews G. 1995. *Ein Knowledge-Engineering-Ansatz für kooperatives Design am Beispiel der Bebauungsplanung.* Universität Kaiserslautern, Kaiserslautern.

Oberweis, A. 1994. *Verteilte betriebliche Abläufe und Objektstrukturen: Integriertes Modellierungskonzept für Workflow-Managementsysteme.* Dissertation, Universität Karlsruhe.

Petrie, C. 1991. *Planning and Replanning with Reason Maintenance.* Dissertation, MCC AI Lab, Austin, TX.

Schmitz, Willi-Gerd. 1994 (Aug.). *Multiple Aufgabenzerlegung von konzeptuellen Modellen.* Diplomarbeit, Universität Kaiserslautern, Fachbereich Informatik, Kaiserslautern.

WIELENGA, B., SCHREIBER, G. UND BREUKER, J. 1992. KADS a modelling approach to knowledge engineering. *Aus:* SCHREIBER, G. (Hrsg.), *Special Issue: The KADS approach to knowledge engineering.* Knowledge Acquisition, Band 4, Nr. 1. Academic Press.