# Multiple Aufgabenzerlegung von konzeptuellen Modellen

W. Schmitz

Diplomarbeit an der Universität Kaiserslautern

1994

Betreuer:
Dr. F. Maurer
Prof. Dr. M.M. Richter

# KAPITEL 1 Einleitung und Überblick

*Dieses Kapitel führt in das Thema dieser Diplomarbeit ein. Es wird die Aufgabenstellung erläutert und ein Überblick über die Gliederung der Arbeit gegeben.*

## 1.1 Einleitung

Nicht nur im Bereich der Forschung, sondern auch in der industriellen Anwendung wird die Informationsversorgung - die Bereitstellung der für Arbeitsabläufe benötigten Daten - immer wichtiger. Soll mit mehreren Personen im Team eine Aufgabe gelöst werden, so müssen die dazu benötigten Daten rechtzeitig in der erforderlichen Form zur Verfügung stehen. Die Kommunikation zwischen den Mitarbeitern und die Koordination der anfallenden Tätigkeiten ist mitentscheidend für die Leistung des Teams. Besonders wichtig wird dies, wenn die kooperierenden Personen nicht nur verschiedene Arbeitsplätze haben, beispielsweise verschiedene Bildschirme, sondern diese Arbeitsplätze weit voneinander entfernt sind. So können sich die Arbeitsplätze durchaus in verschiedenen Stockwerken eines Firmengebäudes oder an verschiedenen Orten befinden, jedoch müssen die Mitarbeiter trotzdem eng zusammenarbeiten. Die sich dabei herausbildenden Arbeitsprozesse sind keineswegs statisch, sondern müssen - bedingt durch neue Aufgaben, neue Rahmenbedingungen und neue Erkenntnisse - ständig angepaßt werden.

Doch effiziente Abläufe lassen sich nicht aus dem Stand heraus festlegen, hierzu ist ein flexibles Vorgehen nötig, das neben der Organisationsstruktur des Arbeitsteams auch die Ablauf- oder Arbeitsstrukturen und die Datenflüsse beschreibt, koordiniert, plant und weiterentwickelt. Es bietet sich an, diese Aufgaben durch Programme zu unterstützen.

Ein mögliches Einsatzgebiet eines solchen Programmes ist es, komplexe Arbeitsabläufe, bei denen mehrere Personen an räumlich ver-

teilten Arbeitsplätzen zusammenarbeiten, zu erfassen, um so einen strukturierten Überblick über den organisatorischen Ablauf und das zur Aufgabenlösung verwandte Wissen zu erhalten. Dieser Überblick ist die Voraussetzung für die Entwicklung von Informationssystemen, die diese Abläufe verbessern, unterstützen und automatisieren sollen.

Komplexe Aufgabenstellungen sind oft dadurch gekennzeichnet, daß zu ihrer Lösung umfangreiches Wissen nötig ist. Die Arbeitsabläufe lassen sich daher meist keineswegs durch eine ganz bestimmte Beschreibung eindeutig erfassen, sondern oft sind, je nach konkreter Problemstellung, verschiedene Lösungsansätze möglich.

Oft ändert sich die Vorgehensweise zur Lösung einer Aufgabe noch während der Problembearbeitung, wenn Mängel am verfolgten Arbeitsablaufplan erkannt werden. Außerdem sind konkrete Arbeitsprozesse mit mehreren Beteiligten dadurch geprägt, daß Teilaufgaben nicht nur koordiniert und verteilt werden müssen, sondern das es dazu kommen kann, daß Mitarbeiter bestimmte Teilaufgaben nicht lösen können. Dies kann dann Auswirkungen auf andere Bereiche der Problemlösung und die Vorgehensweise haben. Nicht vorhersehbare Umstände können auch dazu führen, daß statt der gewählten Lösungsmethode plötzlich ein anderer Lösungsweg bevorzugt wird.

All diese Beispiele beschreiben Situationen, in denen Annahmen über getroffene Lösungsstrategien ungültig werden, was Auswirkungen auf den gesamten Arbeitsablauf hat. Aus diesen Beobachtungen ergibt sich der Gegenstand dieser Diplomarbeit, deren Aufgabenstellung im nächsten Abschnitt erläutert wird.

## 1.2 Aufgabenstellung

Das CoMo-Kit[1] System unterstützt die Entwicklung von Informationssystemen, indem es erlaubt, zu einer Aufgabenstellung das *konzeptuelle Modell* (Begriffserklärung in Abschnitt 2.2) zu beschreiben. In diesem Modell sind alle Objekte erfaßt, die in der betrachteten komplexen Aufgabenstellung auftreten, daneben auch die Operationen, die das System zur Aufgabendurchführung kennen muß.

*Erste Teilaufgabe*

Erstes Ziel dieser Diplomarbeit ist die Erweiterung des konzeptuellen Modells um neue Strukturen. Einmal sollen Objekte in das konzeptuelle Modell aufgenommen werden, mit denen sich zu einer Aufgabe verschiedene Lösungswege spezifizieren lassen. Die Lösung einer Aufgabe erfolgt in CoMo-Kit durch fortgesetzte Aufgabenzerlegung, dies soll so erweitert werden, daß *multiple*[2] *Aufgabenzerlegung* mög-

(1) Conceptual Model Construction Kit [Maurer 93]

(2) lateinisch: vielfältig, vielseitig, vielfach

lich wird. Außerdem soll die Datenübergabe zwischen Teilaufgaben durch ein verbessertes Konzept unterstützt werden, ein typisiertes Ein- und Ausgabekonzept in Form von formalen Parametern.

CoMo-Kit gestattet neben dieser Beschreibung von Arbeitsabläufen - dem ersten Schritt zur Entwicklung eines Informationssystems - auch die Ausführung und damit die Validierung der im konzeptuellen Modell beschriebenen Strukturen.

*Zweite Teilaufgabe*

Durch die eben aufgezählten Erweiterungen des konzeptuellen Modells wurde eine Neuimplementierung der Ausführungskomponente nötig. Um die multiplen Aufgabenzerlegungen zu berücksichtigen, muß diese Komponente zur Laufzeit die Auswahl einer der alternativen Aufgabenzerlegungen - später Lösungsmethode genannt - ermöglichen.

Dabei muß die Ausführungskomponente, im folgenden auch als Interpreter bezeichnet, zu jedem Zeitpunkt einen Überblick über den aktuellen Stand der Aufgabenlösung haben und dessen Konsistenz gewährleisten. Bei Problemen während des Arbeitsablaufes (z.B. durch Umplanungen, Entscheidungen für andere Lösungsmethoden oder Fehler) kann es zu Inkonsistenzen kommen, die vom Interpreter erkannt und beseitigt werden müssen. Die komplette Neugestaltung der Interpreterkomponente ist das zweite Ziel dieser Arbeit, das gemeinsam mit Barbara Dellen (siehe [Dellen 94]) bearbeitet wurde. Eine Übersicht über den Aufbau dieser Arbeit und die Aufgabenverteilung zwischen Barbara Dellen und mir gibt der nächste Abschnitt.

## 1.3 Übersicht

Ausgangsbasis für diese Arbeit ist das CoMo-Kit Tool, vorliegend in Version CoMo-Kit 2.0. Eine Detailkenntnis des Systems ist für das Verständnis dieser Arbeit nicht erforderlich. In die Grundlagen, die Methodik und die für diese Arbeit relevanten Komponenten von CoMo-Kit wird jedoch eine Einführung gegeben. Außerdem sind die wesentlichen Benutzerschnittstellen abgedruckt und erläutert[3].

Ansonsten stellt die Lektüre dieses Textes keine speziellen Anforderungen an den Leser, lediglich für die theoretische Einordnung der Arbeit ist eine Grundkenntnis in den Gebieten Knowledge-Engineering und Wissensakquisition nötig.

(3) Hilfreich für das Verstehen dieser Arbeit kann jedoch eine praktische Vorführung der neu erstellten Version CoMo-Kit 3.0 sein, sowohl vor der Lektüre als auch nach Lesen der Arbeit. Die Niederschrift kann nämlich die bei der Systembenutzung auftretenden Abläufe nur theoretisch erklären, besser veranschaulicht dies jedoch eine Systemdemonstration.

*Kapitel 2*

Das zweite Kapitel gibt die bereits erwähnte Einführung in CoMo-Kit, außerdem ordnet es die Vorgehensweise in die Methodik der Entwicklung wissensbasierter Systeme ein. Dabei wird auch der Begriff des konzeptuellen Modells erläutert. Dieses Kapitel kann von einem Leser, der mit der Terminologie, Funktionsweise und Architektur von CoMo-Kit vertraut ist, übergangen werden.

Die weiteren Teile der Arbeit beschreiben die neu erstellte Version CoMo-Kit 3.0. Eine graphische Übersicht über die Systemkomponenten und ihre Zuordnung zu den Kapiteln der Arbeit gibt Abbildung 1. Sie soll nur einen ersten Überblick über den Aufbau der Arbeit ermöglichen. Die Bedeutung und Funktionsweise der einzelnen Komponenten wird in den betreffenden Kapiteln detailliert vorgestellt.

*Kapitel 3*

Kapitel 3 behandelt das erste Teilziel der Diplomarbeit. Die notwendigen Erweiterungen des konzeptuellen Modells werden ausführlich motiviert und vorgestellt, außerdem die dadurch bedingten Erweiterungen der Modellierungswerkzeuge.

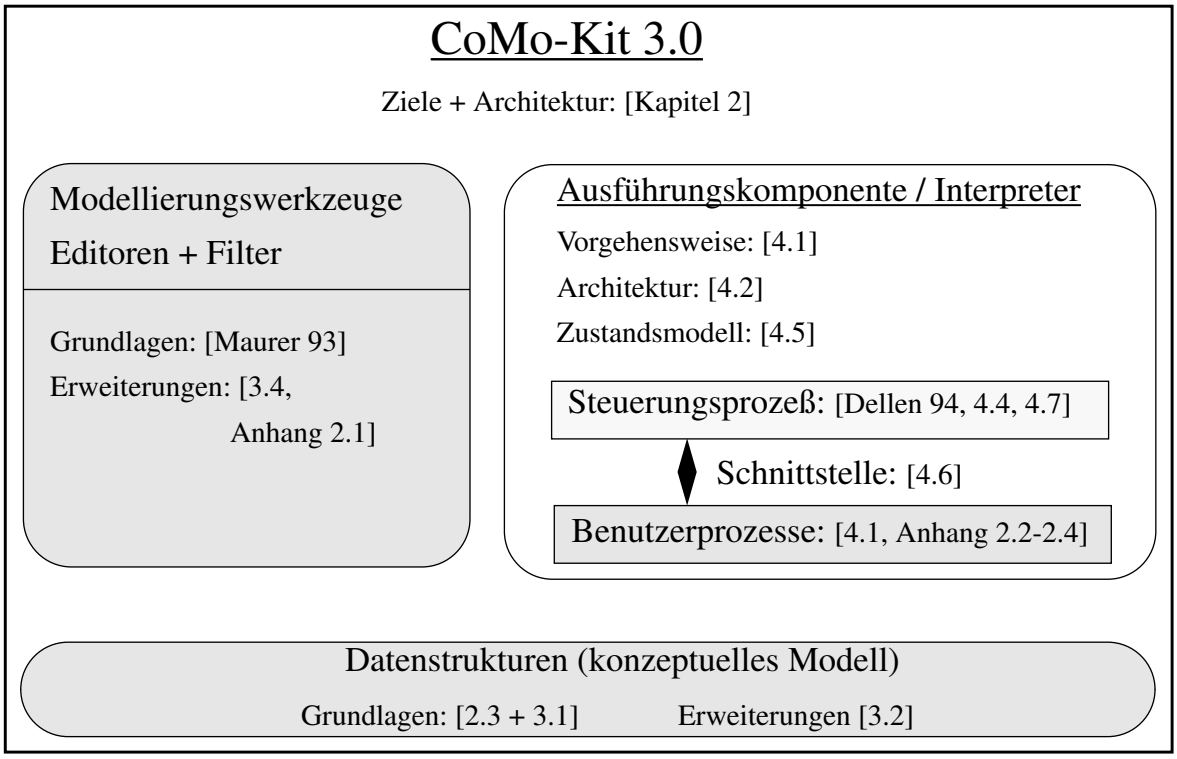



ABBILDUNG 1 Zuordnung der Systemkomponenten zu den Kapiteln der Diplomarbeit: Der hell unterlegte Teil des Interpreters wurde von Barbara Dellen erstellt, die dunkel unterlegten Teile wurden von mir entwickelt (Benutzerprozesse) bzw. erweitert.

*Kapitel 4*

Daran schließt sich der zentrale Teil der Diplomarbeit an. Er beschäftigt sich mit der zweiten Teilaufgabe dieser Arbeit, der Reimplementierung der Ausführungskomponente. Gemeinsam mit Barbara Dellen wurde die Architektur des Interpreters entwickelt und ein

grundlegendes Zustandsmodell spezifiziert. Daran orientiert wurde die Implementierung in zwei Teile aufgespalten. Die Erstellung der zentralen Interpreterkomponente, des Steuerprozesses, übernahm Barbara Dellen. Die Benutzerprozesse wurden von mir implementiert. Zuvor war gemeinsam eine Schnittstelle spezifiziert worden, über welche die Prozeßkommunikation abläuft.

Die in den beiden Diplomarbeiten implementierten Systemkomponenten sind voneinander abhängig und ergeben nur gemeinsam ein lauffähiges Programm. Daher sind, um ein vollständiges Verständnis der neuen Konzeption des Interpreters zu ermöglichen, in Kapitel 4 auch die Systemfunktionen beschrieben, die in den Verantwortungsbereich von Barbara Dellen fielen. Hier ist die Beschreibung allerdings äußerst knapp gehalten, für ein Detailverständnis empfiehlt sich die anschließende Lektüre von [Dellen 94]. Ausführlicher werden die gemeinsam entwickelten Grundlagen und die von mir implementierten Teile erläutert.

*Kapitel 5*

Das letzte Kapitel faßt die Resultate zusammen und gibt einen Ausblick auf mögliche Erweiterungen des erstellten Systems.

*Anhang*

Den Abschluß der Arbeit bildet im Anhang neben dem Literaturverzeichnis eine Zusammenstellung der implementierten Benutzerschnittstellen mit einer Beschreibung ihrer Funktionalität. Außerdem findet sich dort eine Auflistung der erwähnten Schnittstelle zwischen dem Steuerungsprozeß und den Benutzerprozessen.

*Beispiel*

Durchgehend wird ein kleines Demonstrationsbeispiel benutzt und abgedruckt, um die auftretenden Objekte, die Benutzerschnittstellen und die Funktion der Ausführungskomponente zu veranschaulichen. Dieses Beispiel wurde von uns zum Testen des Systems verwendet. Es beschreibt, wie man das Problem der Einnahme eines Mittagessens lösen kann. Allerdings erhebt es keinen Anspruch auf Vollständigkeit und angemessene Modellierung der Realität, es dient lediglich zur Demonstration. Inzwischen wurde allerdings die neu entwickelte Programmversion mit der Modellierung einer realen Domäne - dem Problem der Bebauungsplanerstellung - erfolgreich getestet (siehe [Beste 94b]).

# KAPITEL 2 Das Knowledge Engineering Werkzeug CoMo-Kit

*Dieses Kapitel bietet eine kurze Einführung in die Ausgangsbasis dieser Diplomarbeit, das CoMo-Kit System.*

Im ersten Abschnitt werden schlagwortartig die Aufgabe, die Funktion und die Ziele des CoMo-Kit Systems vorgestellt. Abschnitt 2.2 bettet das System in einen größeren Rahmen ein. Hierzu werden die Phasen zur Entwicklung wissensbasierter Systeme vorgestellt, sowie Ansätze zur Wissensakquisition erläutert. Die mit CoMo-Kit verfolgte Vorgehensweise wird in die modellbasierte Wissensakquisition eingeordnet und mit dem Rapid Prototyping Ansatz verglichen. Im Zusammenhang mit der KADS-Begriffswelt wird der Begriff „konzeptuelles Modell“ eingeführt. Abschnitt 2.3 stellt die zentralen Konzepte vor, die im konzeptuellen Modell erfaßt werden. Der letzte Abschnitt beinhaltet eine grobe Darstellung der Systemarchitektur von CoMo-Kit.

## 2.1 Aufgaben, Funktionen und Ziele von CoMo-Kit

CoMo-Kit ist eine Entwicklungsumgebung für verteilte, wissensbasierte Informationssysteme. Es soll Wissensingenieure und Experten bei der Erstellung solcher Systeme unterstützen. Charakteristisch für die Domänen, in denen die zu entwickelnden Systeme benötigt werden, ist:

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

*Systemfunktionen*

CoMo-Kit ermöglicht, solche komplexen Aufgaben systematisch

- zu analysieren und darzustellen,
- durch Aufgabenzerlegung zu vereinfachen,
- durch Interaktion mit dem System auszuführen,
- damit die verteilte kooperative Problemlösung zu simulieren,
- einzelne Aufgaben nach und nach statt von menschlichen Sachbearbeitern durch den Computer erledigen zu lassen.

*Ziele*

Das Resultat der geschilderten Vorgehensweise ist

- ein besseres Verständnis der modellierten Aufgabe,
- ein besseres Verständnis der Kooperationsprozesse zu ihrer Lösung,
- ein System, das die Simulation der Problemlösung ermöglicht und auch zur realen Problemlösung verwandt werden kann,
- die Unterstützung prototypischer Systementwicklung durch den inkrementellen Übergang von informell beschriebenen zu operationalisierten Aufgaben.

Im nächsten Abschnitt werden verschiedene Methoden zur Entwicklung wissensbasierter Systeme vorgestellt und das CoMo-Kit System innerhalb dieser Ansätze positioniert.

## 2.2 Methoden zur Entwicklung wissensbasierter Systeme

*Knowledge Engineering*

Das CoMo-Kit System ordnet sich in den Rahmen des Knowledge Engineering ein. „Knowledge Engineering umfaßt alle Tätigkeiten, die sich mit der systematischen Erhebung, Strukturierung, Spezifikation, Implementierung und Wartung von Wissensbasen auseinandersetzen“[4]. Analog zum klassischen Softwareengineering läßt sich auch die Entwicklung wissensbasierter Systeme in verschiedene Phasen einteilen. Diese Phasen (Wissenserhebung, Wissensstrukturierung, Systemspezifikation und Implementierung) laufen sukzessive ab, in der Regel sogar mehrfach wiederholt (evolutionärer Entwicklungsprozeß).

*Wissensakquisition*

Wissensakquisition bezeichnet nach [Maurer 93] den Teil des Knowledge Engineering, „der sich mit der *Erstellung* eines wissensbasierten Systems beschäftigt“[5]. Andere Quellen sehen Wissensakquisition auf die Phasen der Wissenserhebung und Wissensstrukturierung beschränkt[6]. Konsens ist aber, daß in der Wissensakquisitionsphase das Expertenwissen zu sammeln und zu beschreiben ist, daß es dabei strukturiert wird und Zusammenhänge aufgedeckt werden. Zur Wis-

(4) [Maurer 93] S. 38

(5) [Maurer 93] S. 39

sensakquisition gibt es verschiedene Ansätze. Im folgenden werden zuerst der Rapid Prototyping Ansatz, danach der systematischere und strukturiertere modellbasierte Wissensakquisitionsansatz vorgestellt.

*Rapid Prototyping Ansätze*

Die „am meisten verbreitete Form der Expertensystementwicklung ist... Rapid Prototyping“[7]. Dabei versucht man, möglichst schnell zu einem lauffähigen Prototypen des Systems zu kommen, zum Beispiel, indem der Experte sein Wissen einem Wissensingenieur diktiert und dieser das Wissen sofort in einen vorbereiteten Wissensrepräsentationsmechanismus oder eine Programmiersprache kodiert. Beginnend mit einem kleinen Ausschnitt werden so immer umfassendere Teile der Systemfunktionalität realisiert.

Problematisch an diesem Vorgehen ist jedoch, daß es in der Regel nicht möglich ist, das Wissen des Experten auf die vorhandenen Wissensrepräsentationsmechanismen abzubilden - sofern der Experte überhaupt in der Lage ist, sein Wissen explizit zu machen. Außerdem kommt es durch die frühe Kodierung zu einer Vermischung von Wissens- und Kontrollstrukturen. Wissensbasierte Systeme sollen sich aber grundsätzlich durch eine Trennung von Wissen und Kontrollmechanismus zu dessen Verarbeitung auszeichnen.

Hauptnachteil der Rapid Prototyping Ansätze ist nach [Angele et al. 90], daß kein explizites Modell des Expertenwissens erstellt wird, da man den Softwareentwicklungsprozeß auf Wissenserhebung und Implementierung einschränkt. Wissensstrukturierung und Systemspezifikation werden also vernachlässigt.

Eine bessere Vorgehensweise erfordert, daß sich der Wissensingenieur - ehe er mit der Implementierung beginnt - einen Überblick über die Gesamtstruktur der Domäne verschafft, ohne hierbei schon Implementierungsdetails zu berücksichtigen.

*Modellbasierte Akquisition*

Aus dieser Motivation heraus wurden modellbasierte Wissensakquisitionsansätze entwickelt. Sie sehen die Konstruktion eines wissensbasierten Systems als Prozeß einer sukzessiven Modellbildung. Die in den verschiedenen Phasen erzeugten Modelle sind Produkte, die gewisse Teilaspekte des Systems modellieren und jeweils von anderen abstrahieren. Sie sind zwar Grundlage für die Implementierung, strukturieren aber den Entwicklungsprozeß so, daß die Wissenserhebung klar von der Spezifikation und der Implementierung getrennt

(6) [Richter et al. 91] S.4: „Die Aufgabe der Wissensakquisition ist die Ermittlung und Aufbereitung des von menschlichen Experten zur Lösung des bearbeiteten Problems verwendeten Wissens bis hin zur formalen Repräsentation.“

(7) [Richter 92] S. 298

ist. Die Vermischung von Wissen mit Kontrollstrukturen wird so verhindert.

*KADS*

Die bekannteste modellbasierte Wissensakquisitionsmethode ist der KADS-Ansatz[8]. KADS beschreibt den Weg vom Expertenwissen zu einem fertigen System durch die Folge von konzeptuellem Modell, Designmodell und Implementationsmodell (siehe Abbildung 2). In der Analysephase wird die grundlegende Struktur der Domäne durch den Experten beschrieben. Dabei werden Wissenstypen und Verarbeitungsmethoden identifiziert und im konzeptuellen Modell festgehalten. Es beschreibt implementierungsunabhängig die Vorgehensweise des Expertensystems, orientiert an der Denkweise des Experten.

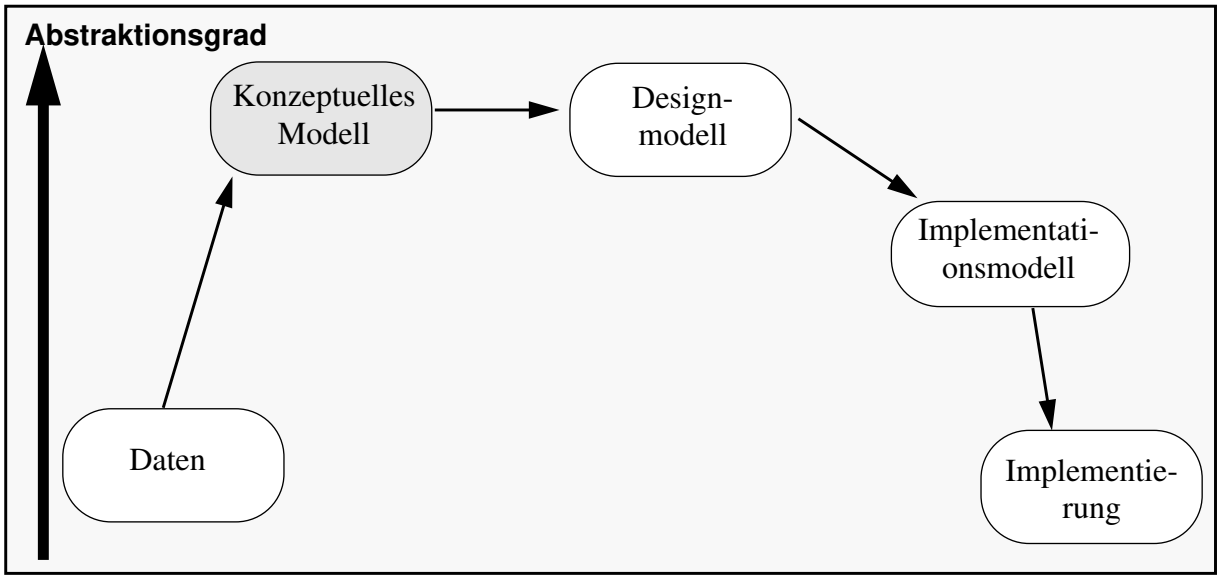



ABBILDUNG 2 KADS-Methodik

Im zweiten Schritt werden die ermittelten Verarbeitungsmethoden und Wissenstypen mit implementierbaren Verfahren in Verbindung gebracht. Man entwickelt das Designmodell, das bereits Formalismen und Methoden enthält, die zur Implementierung auf einem Rechner geeignet sind. Daran schließt sich die Erstellung des Implementationsmodells an, in dem das Designmodell genauer spezifiziert wird, sowie die abschließende Implementierungsphase.

*Konzeptuelles Modell*

Wie schon der Titel dieser Arbeit zeigt, beschäftigen wir uns in erster Linie mit dem konzeptuellen Modell. Dieses besteht aus dem Modell der Expertise[9] und dem Kooperationsmodell[10]. Das Kooperations-

(8) nach dem europäischen Forschungsprojekt KADS (Knowledge Acquisition and Documentation System), siehe [Wielinga et al. 92]

(9) Model of expertise

(10) Model of cooperation

modell beschreibt, wie zur Lösung einzelner Aufgaben Benutzer und das zu erstellende System zusammenarbeiten. „The model of cooperation contains a specification of the functionality of those sub-tasks ... that require a cooperative effort“[11]. Es wird in dieser Arbeit nicht mehr explizit erwähnt.

*Modell der Expertise*

Ziel des Modells der Expertise ist es, das zur Lösung der Aufgaben erforderliche Expertenwissen zu beschreiben. „The model of expertise specifies what kind of knowledge has to be aquired for the expert system“[12]. Die Berücksichtigung dieses Wissens zeichnet nach [Wielinga et al. 92] eben die Entwicklung wissensbasierter Systeme aus. Beim traditionellen Softwareengineering werden in der Analysephase nach [Angele et al. 90] nur zwei Arten von Wissen erfaßt: Wissen über die Objekte und ihre Beziehungen sowie Wissen über die zu lösenden Aufgaben (Systemfunktionen). Die Frage, wie die Aufgaben gelöst werden - also der Problemlösungsprozeß - spielt erst beim Systemdesign eine Rolle. Beim Knowledge Engineering hingegen interessiert Wissen darüber, wie die Aufgabe gelöst wird (inference knowledge) schon bei der Wissenserhebung. Man will ja das darüber vorhandene Expertenwissen ausnutzen. Dazu werden innerhalb des Modells der Expertise vier Wissensebenen (knowledge layers[13]) unterschieden und analysiert.

*Wissensebenen*

Auf der Domänenebene werden die Objekte und ihre Beziehungen beschrieben, das statische Wissen des Gegenstandsbereiches. Die Inferenzebene umfaßt die Problemlösungsschritte (das „Wie“ der Aufgabenlösung). Auf der Aufgabenebene werden die einzelnen Inferenzschritte kombiniert und mit den Objekten der Domänenebene in Verbindung gesetzt. Hier wird somit der Kontrollfluß spezifiziert. Als vierte Ebene wird in KADS noch die Strategieebene beschrieben, die Metawissen über die Auswahl und Kombination der einzelnen Teilaufgaben enthält. Sie beschreibt, warum genau der angegebene Datenfluß eine angemessene Problemlösung darstellt.

CoMo-Kit ermöglicht es, alle Bestandteile des konzeptuellen Modells im System abzubilden. Die dabei verwandten konkreten Objekte werden in Abschnitt 2.3 vorgestellt. Daneben unterstützt CoMo-Kit auch eine frühzeitige Validierung der erfaßten Daten gegenüber den Benutzeranforderungen. Es erlaubt bereits in der Wissensakquisitionsphase die definierten Strukturen zu operationalisieren und so zu überprüfen, ob die gestellten Anforderungen erfüllt

(11) [Wielinga et al. 92] S. 7

(12) [Schmalhofer et al. 91] S. 64

(13) Für die Wissensebenen werden folgende deutsche Übersetzungen zur Bezeichnung verwandt: Domänenebene (statt domain layer), Inferenzebene (inference layer), Aufgabenebene (task layer), Strategieebene (strategic layer).

sind. Dies geschieht, indem die beteiligten Personen die Aufgaben interaktiv mit dem System bearbeiten. Außerdem ist ein inkrementeller Übergang von der informellen Spezifikation zu einer operationalen Struktur möglich, indem schrittweise Aufgaben formalisiert werden und ihre Abarbeitung vom Computer übernommen wird. Eine Übersicht über die hierzu benötigten Systemkomponenten gibt Abschnitt 2.4.

Obwohl CoMo-Kit auf dem modellbasierten Wissensakquisitionsansatz fußt, ermöglicht es also die Vorteile von Prototyping zu nutzen.

## 2.3 Basisobjekte des Knowledge Engineering Prozesses

Zur Beschreibung des konzeptuellen Modells gehören in CoMo-Kit eine Reihe von Basisobjekten. Bisher wurden sie als Wissenstypen und Verarbeitungsmethoden bezeichnet. Dahinter verbirgt sich aber eine durch das CoMo-Kit-System vorgegebene feinere Strukturierung. Folgende Objekte werden im Wissensakquisitions-Prozeß erfaßt:

*Protokolle*

Die Rohdaten des Knowledge-Engineering-Prozesses können in unstrukturierter, nicht formalisierter Form als Protokolle vorliegen. Sie umfassen große Informationsmengen, beispielsweise als Text oder auch als multimediale Daten. Sie werden vom Experten oder Wissensingenieur weiter strukturiert, um die nachfolgend beschriebenen Objekte zu identifizieren. Protokolle tauchen in dieser Arbeit weiter nicht mehr auf.

*Konzepte*

Die in der Domäne vorliegenden Daten werden als Konzepte erfaßt. Sie können sowohl als Ausgangsdaten, als Ergebnis oder als Zwischenresultate bei der Lösung des Problems eine Rolle spielen. Im Detail vorgestellt und weiter untergliedert werden die Konzepte in Abschnitt 3.1.1., sie repräsentieren die Domänenebene nach KADS.

*Aufgaben*

Die einzelnen Schritte, die zur Lösung des gestellten Problems durchzuführen sind, sind Aufgaben, auch Tasks genannt. Sie werden spezifiziert durch

- ein Ziel, das erreicht werden soll (das goal)
- eine Vorgehensweise (wie auch das Ziel durch eine für den Bearbeiter leicht verständliche, informelle Beschreibung definiert)
- die benötigten Eingaben
- die erzeugten Ausgaben

Aufgaben können in Unteraufgaben aufgespalten werden, so daß durch fortschreitende Zerlegung eine baumartige Aufgabenstruktur entsteht. Daraus resultiert die Unterscheidung nach atomaren und

zusammengesetzten, komplexen Aufgaben (siehe Abschnitt 3.1.2). Durch Erfassen der Aufgaben beschreibt man die Inferenzebene nach KADS. Ferner muß zu jeder komplexen, weiter zerlegten Aufgabe ein Datenfluß spezifiziert werden, der beschreibt, wie die zur Lösung der Aufgabe benötigten Daten von Unteraufgabe zu Unteraufgabe weiter gegeben werden. Zur Datenflußmodellierung wird der Objekttyp „formaler Parameter“ benötigt und in Abschnitt 3.2.2 eingeführt.

*Agenten*

Die Bearbeitung der erfaßten Aufgaben erfolgt durch Agenten. Dabei werden Menschen und Computer unterschieden. Menschen lösen Aufgaben, die ihnen vom System zur Bearbeitung angeboten werden, indem sie nach einer Arbeitsvorschrift die Eingaben in Ausgaben transferieren. Computer lösen Aufgaben selbständig, allerdings ist hierzu eine Formalisierung der Aufgabe nötig. Alle, die an der Problemlösung in der Domäne mitarbeiten oder später mitarbeiten sollen, müssen als Basisobjekt „Agent“ im System abgebildet werden. Dabei lassen sich, zum Beispiel aufgrund gemeinsamer Fähigkeiten einzelner Personen, Agentengruppen definieren.

## 2.4 Die Grobarchitektur von CoMo-Kit

Die in Abbildung 3 gezeigte Architektur wurde auch in der neu entwickelten Version CoMo-Kit 3.0 beibehalten. Alle Komponenten

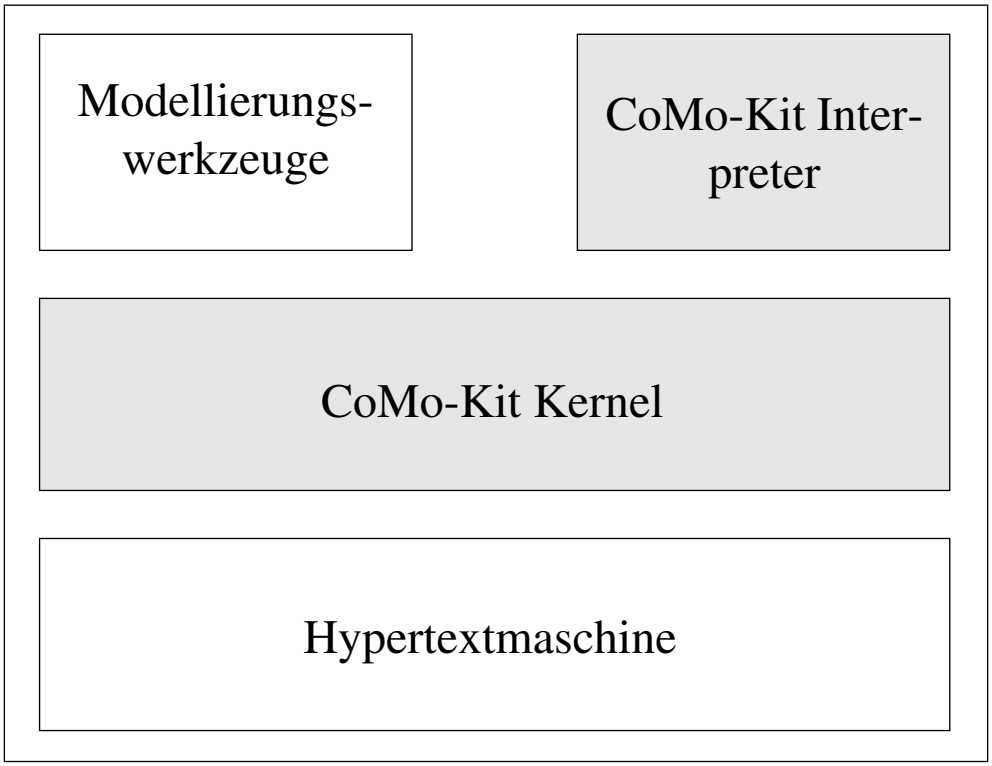



ABBILDUNG 3 CoMo-Kit - Architektur

wurden unter Smalltalk entwickelt, wobei das Werkzeug VisualWorks zur Implementierung der graphischen Benutzerschnittstellen verwandt wurde.

*Hypertextmaschine*

CoMo-Kit setzt auf einer Hypertextmaschine[14] auf, die Hypermedianetze verwaltet. Sie stellt Funktionen zur Verfügung, um Objekte in Form von Knoten und Beziehungen zwischen den Objekten in Form von Kanten zu speichern. Dabei können die Knoten multimediale Information enthalten, neben Texten beispielsweise auch Bilder und Videos. Das Hypertextsystem bietet die Möglichkeit, solche Objekte zu editieren und durch Ausfiltern eine Untermenge der Objekte anzuzeigen. Diese Funktionen werden von den höheren Schichten des Gesamtsystems benutzt und weiter ausgebaut.

*CoMo-Kit-Kernel*

Der CoMo-Kit-Kernel gestattet mit den dafür vorgesehenen Editoren die Entwicklung des konzeptuellen Modells und dessen Verwaltung. Er besteht im wesentlichen aus einem „Knowledge Repository“, in dem die Wissenseinheiten abgelegt und auf das Hypermedianetz abgebildet werden. Somit können alle Benutzer darauf zugreifen. In das Knowledge Repository werden die schon in Abschnitt 2.3 vorgestellten Objekte aufgenommen, also Protokolle, Aufgaben und Konzepte, ferner auch die in Kapitel 3 neu eingeführten Methoden und Parameter. Die Beziehungen zwischen den Objekten werden durch Links (Kanten) repräsentiert, die der Benutzer jeweils zwischen zwei Objekten einfügen kann. Zum Beispiel kann er einer Aufgabe eine Konzeptinstanz als Eingabe zuordnen, woraufhin - wie in Abbildung 4 zu erkennen - ein InputToTask-Link erzeugt wird.

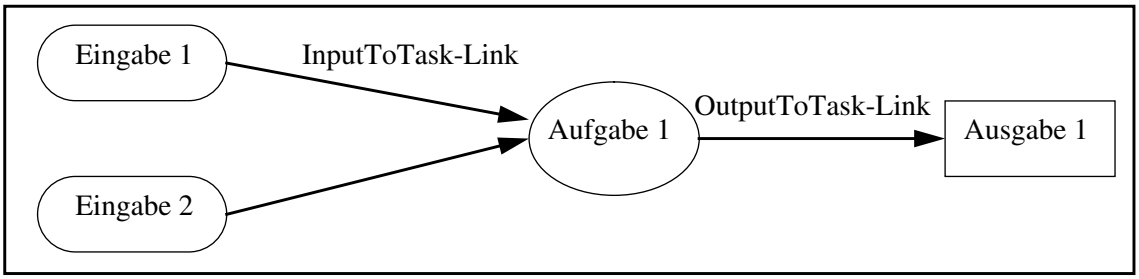



ABBILDUNG 4 Verwendung von Knoten und Links: Man erkennt zwei Eingabeknoten, einen Aufgabe- und einen Ausgabeknoten und entsprechende Links.

Aufgabe des CoMo-Kit Kernel ist in erster Linie, die in CoMo-Kit verwandten Objekte auf das Hypermedianetz abzubilden und dem Benutzer für eine Modellierung zur Verfügung zu stellen. Eine weitere Kernelkomponente erzeugt automatisch Benutzerschnittstellen zur Wissensakquisition in Form von Bildschirmmasken. Hierzu gehören insbesondere die Eingabemasken, mit denen bei der Aufgabenbearbeitung Resultate erfaßt werden können (siehe [Beste 94]).

(14) HAM (Hypertext Abstract Machine), siehe [Maurer 93]

*Modellierungswerkzeuge* Hierunter werden alle Werkzeuge zusammengefaßt, die dem Benutzer erlauben, die Objekte des Kernels zu erzeugen, anzusehen und zu verändern. Es gibt eine Vielzahl von Editoren, jeweils abgestimmt auf einen bestimmten Objekttyp. Filter dienen dazu, jeweils bestimmte Objekte (beispielsweise die Aufgabenhierarchie) anzusehen. Durch die Erweiterungen des CoMo-Kit Kernels wurden auch Erweiterungen der Editoren nötig. Die neu entworfenen Benutzerschnittstellen werden in Anhang 2 vorgestellt.

*CoMo-Kit-Interpreter* In dieser Arbeit wurde die Ausführungskomponente, der CoMo-Kit-Interpreter, neu entwickelt. Er dient dazu, die im konzeptuellen Modell definierte Aufgabenstruktur in Interaktion mit dem Benutzer abzuarbeiten. Damit kann die Problemlösung simuliert werden, was insbesondere zum Validieren des konzeptuellen Modells - beispielsweise gegenüber den Benutzeranforderungen - sinnvoll ist. Die Architektur und Funktionsweise dieses Interpreters wird in Kapitel 4 detailliert vorgestellt.

Der Interpreter nutzt eine weitere Funktion, die der CoMo-Kit-Kernel zu Verfügung stellt. Diese ermöglicht es, einzelne Aufgaben zu operationalisieren, indem ihnen eine ausführbare Beschreibung in einer Programmiersprache zugeordnet wird. Hierzu steht eine Bibliothek zur Verfügung, die anpaßbare Quellcodes für bestimmte typische Aufgaben enthält.

Für weitere Einzelheiten bezüglich der hier nicht näher beschriebenen Komponenten sei auf [Maurer 93] verwiesen.

KAPITEL 3

# Erweiterungen der Modellierungswerkzeuge

*Dieses Kapitel beschreibt, welche Änderungen am CoMo-Kit Kernel im Rahmen dieser Diplomarbeit vorgenommen wurden. Neue Modellierungskonzepte ergänzen die bereits vorgestellten Basisobjekte des Knowledge Engineering Prozesses.*

Dieses Kapitel motiviert, warum an den Modellierungswerkzeugen von CoMo-Kit konzeptionelle Erweiterungen nötig waren und stellt diese Erweiterungen - mit denen die erste Teilaufgabe der Diplomarbeit gelöst wird - vor. Abschnitt 3.1 beschreibt, welche Datenstrukturen bisher zur Modellierung der Verarbeitungsmethoden und der Wissenseinheiten im konzeptuellen Modell verwendet wurden. Dabei traten jedoch Probleme auf. Diese werden im zweiten Abschnitt erläutert. Dort werden neue Modellierungskonzepte (Parameter und Methoden) eingeführt, welche die Darstellungsmöglichkeiten von CoMo-Kit erweitern. In Abschnitt 3.3 wird erklärt, warum die neue Modellierung als Und/Oder-Baum aufgefaßt werden kann. Der letzte Teil schließlich geht auf notwendige Ergänzungen der Editoren des konzeptuellen Modells ein.

## 3.1 Modellierung der Aufgabenstruktur und der Wissenseinheiten

### 3.1.1 Beschreibung des Domänenwissens durch Konzepte

Beim Modellieren der Domäne werden die auftretenden Daten durch Konzepte beschrieben. Es werden Konzeptklassen und Konzeptinstanzen unterschieden.

*Konzeptklassen*

Konzeptklassen definieren die Struktur der Objekte, die in der Domäne auftreten, sie stellen die Spezifikation der Objekte dar. Jede Klasse besteht aus einer Attributmenge, wobei jedem Attribut ein Typ zugeordnet ist. Als Typen können entweder wiederum Konzeptklassen selbst oder vordefinierte Basistypen (Integer, String, Real,

Sound, Video...) verwendet werden. Zusätzlich ist ein Defaultwert sowie die Kardinalität des Attributs beschreibbar. Konzeptklassen gehören zum Bereichswissen, da sie die Struktur der auftretenden Wissenseinheiten beschreiben. Sie dienen als Schablonen zur Erzeugung von Konzeptinstanzen[15].

*Konzeptinstanzen*

Eine Instanz belegt die Attribute der Konzeptklassen mit Werten. Wird kein expliziter Wert eingetragen, so wird der Defaultwert benutzt. Konzeptinstanzen werden einerseits erzeugt, um unveränderliches Wissen in den Problemlöseprozeß einzubringen (Problemlösewissen). Andererseits erzeugt man auch Instanziierungen, um während der Problemlösung generiertes Wissen (Problemfallwissen) zu verwalten.

### 3.1.2 Beschreibung der Aufgabenstruktur

Die zu lösenden Aufgaben (eingeführt auf Seite 12) werden in atomare und komplexe Aufgaben unterschieden.

*Komplexe Aufgaben*

Komplexe Aufgaben lassen sich in mehrere Unteraufgaben zerlegen. Zur Lösung einer solchen Aufgabe müssen alle Teilaufgaben gelöst werden. Dazu delegiert der Bearbeiter einer komplexen Aufgabe die Unteraufgaben an von ihm auszuwählende Agenten und überwacht ihre Ausführung. Die Ausführung einer komplexen Aufgabe instanziiert somit keine Ergebnisse, sondern ihr Resultat ist die Zerlegung in Unteraufgaben und deren Weitergabe an andere Agenten.

*Atomare Aufgaben*

Atomare Aufgaben entsprechen einzelnen, unzerlegten Problemlösungsschritten. Der Inferenzschritt besteht darin, daß der Bearbeiter ausgehend von Eingabedaten bestimmte Ausgaben erzeugt.

Eingabe einer Aufgabe ist eine Menge von Konzeptklassen und Konzeptinstanzen, wobei von jeder Konzeptklasse bei Ausführung der Aufgabe genau eine konkrete Instanz erwartet wird. Die Konzeptinstanzen beschreiben statisches Problemlösewissen, das bereits bei Spezifikation des konzeptuellen Modells instanziiert wurde. Hingegen werden die Konzeptklassen problemfallspezifisch verwandt, sie spezifizieren nur, von welchem Typ (welcher Klasse) das zu erwartende Objekt ist.

Als Ausgabe ist jeder Aufgabe ebenfalls eine Menge von Konzeptklassen zugeordnet. Resultat atomarer Aufgaben ist eine Instanziierung der Ausgabekonzepte. Abbildung 5 zeigt beispielhaft eine atomare Aufgabe mit zwei Eingabeinstanzen und einem Ausgabekonzept.

(15) Konzeptklassen entsprechen etwa den in der Expertensystemtechnologie oft verwendeten „Frames“

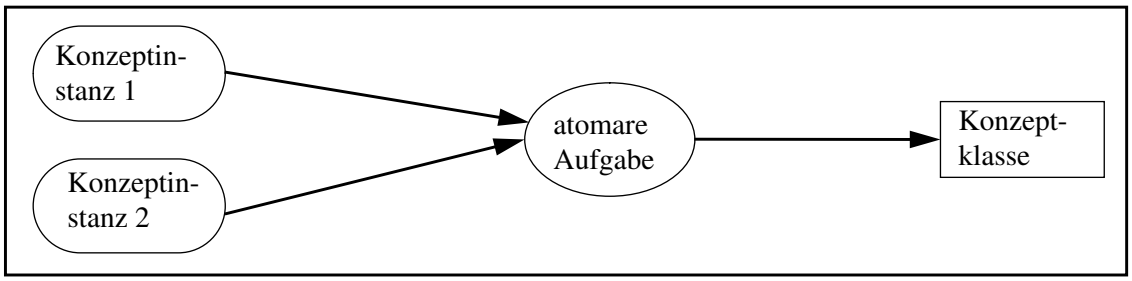



ABBILDUNG 5 Atomare Aufgabe

## 3.2 Erweiterung der vorhandenen Modellierungskonzepte

### 3.2.1 Mängel bei der Verwendung von Konzepten und Instanzen

Durch Spezifikation der Aufgabenbeschreibungen und Zuordnen der Ein- und Ausgaben zu den Aufgaben entsteht ein Datenflußgraph, der durch fortschreitende Zerlegung beliebig detailliert werden kann. Schon bisher konnten in CoMo-Kit beliebig viele *Konzeptinstanzen* einer Konzeptklasse in diesen Datenflußgraph eingebunden werden, nachdem die Konzeptklasse und all diese Instanzen mit entsprechenden Eingabemasken erzeugt worden waren (siehe [Beste 94]). Über diese Konzeptinstanzen konnte somit beliebig viel zu der „Schablone“ passendes Wissen in den Problemlösungsprozeß eingebracht werden. Dieses war jedoch während der Problemlösung nicht modifizierbar.

Die *Konzeptklasse* hingegen konnte (da jede Klasse nur einmal existiert) nur *einmal* verwendet werden. Sie konnte zwar mehreren Aufgaben auf verschiedenen Stufen der Aufgabenhierarchie als Eingabe zugeordnet sein. Jedoch konnte nur eine Aufgabe das im Datenflußgraph eingesetzte Konzept als Ausgabe benutzen, also instanziieren. Da alle Aufgaben während des Problemlösungsprozesses pro spezifizierter Eingabe-Konzeptklasse genau eine Instanz erwarten, und pro spezifizierter Ausgabe eine Instanz erzeugen, ließen sich nicht mehrere „Schablonen“ eines Konzeptes mit *verschiedenen* Werten belegen. CoMo-Kit erlaubt bisher beispielsweise nicht die Abbildung der einfachen Aufgabe „Addition zweier reeler Zahlen“. Dazu hätten nämlich zwei unabhängige Instanzen des Konzeptes reele Zahl verwaltet werden müssen.

Als weiteres Beispiel sieht man in Abbildung 6 den Datenfluß zwischen den Aufgaben „Telefonnummer suchen“, „Bestellung aufgeben“, „Platz schaffen“ und „Pizza essen“. Verschiedene Instanzen (Notizzettel - Preise, Notizzettel - Wünsche) des Konzeptes Notizzet-

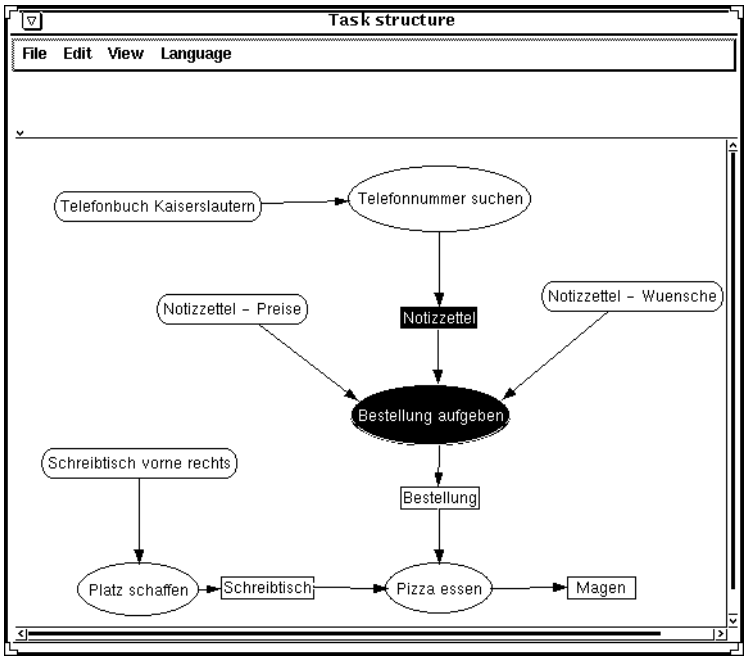



ABBILDUNG 6

Alte Aufgabenstruktur in CoMo-Kit 2.0: Ovale stellen Aufgaben dar, Rechtecke entsprechen Konzeptklassen, Rechtecke mit abgerundeten Ecken Konzeptinstanzen. Dieses Beispiel beschreibt, wie die Aufgabe Mittagessen gelöst werden kann, indem man ein Essen beim Pizzaservice bestellt. Dazu muß zuerst die Telefonnummer gesucht werden, diese kann man sich notieren. Mit Hilfe weiterer Notizzettel wird die Bestellung aufgegeben. Parallel hierzu kann der Schreibtisch aufgeräumt und danach die Pizza gegessen werden.

tel werden als Eingabe für die Aufgabe „Bestellung aufgeben“ benutzt. Ferner ist die Konzeptklasse Notizzettel als Ausgabe von „Telefonnummer suchen“ zu erkennen. Zur Laufzeit wird eine Instanz des Notizzettels an die Aufgabe „Bestellung aufgeben“ weitergegeben. Jedoch bestand keine Möglichkeit, noch einen weiteren Notizzettel als Ausgabe der Aufgabe „Telefonnummer suchen“ einzutragen. Auch konnte keine andere Aufgabe einen Notizzettel beschreiben.

Zusammengefaßt: Vom Einbringen des Wissens durch Konzeptinstanzen abgesehen, ließ sich die Konzeptklasse nur ein einziges Mal in einem Datenfluß verwenden, dynamisch konnte nur eine einzige Instanz erzeugt werden. In vielen Domänen resultieren aber aus *verschiedenen* Aufgaben unterschiedliche Instanzen einer Klasse. Oder *eine einzige* Aufgabe erzeugt mehrere unterschiedliche Instanzen einer Klasse. Um diese Defizite zu beheben, mußte ein neuer Objekttyp in CoMo-Kit eingeführt werden, der formale Parameter.

### 3.2.2 Einführung formaler Parameter

*Begriffsbestimmung*: **Formale Parameter** verknüpfen Konzeptklassen mit Aufgaben. *Ein* Parameter ordnet jeweils eine Konzeptklasse allen Aufgaben zu, die genau *dieselbe* Instanz dieser Klasse als Ein- oder Ausgabe verwenden (siehe Abbildung 7).

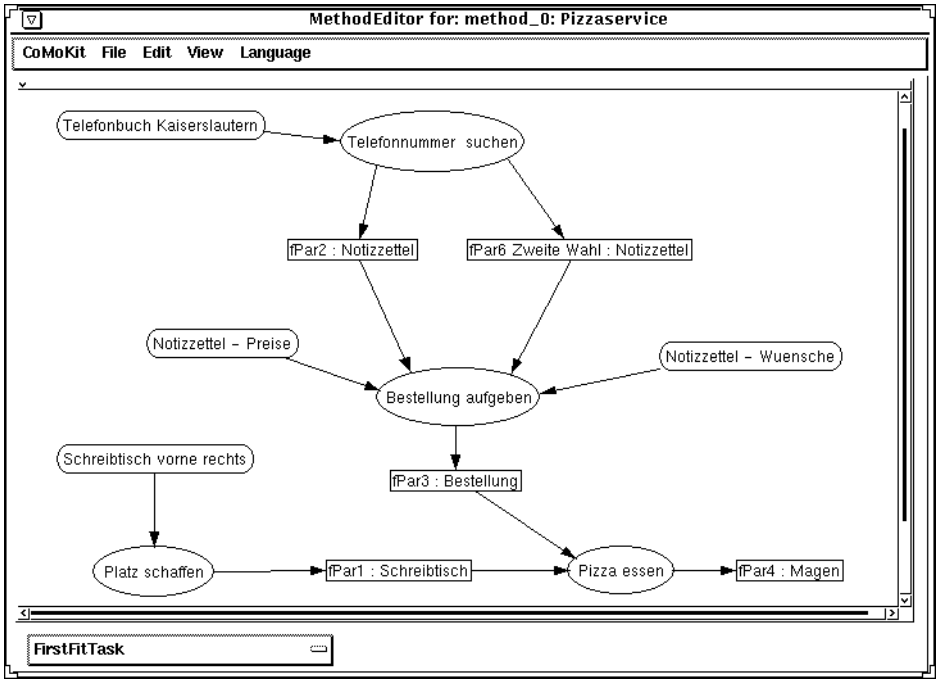



ABBILDUNG 7

Aufgabenstruktur in CoMo-Kit 3.0 mit formalen Parametern: Rechtecke stellen nun Parameter dar, Rechtecke mit abgerundeten Ecken entsprechen weiterhin den Konzeptinstanzen. Im Unterschied zu Abbildung 6 sieht man nun, daß die Aufgabe „Telefonnummer suchen“ zwei Instanzen des Notizzettels beschreiben kann.

Der Parameter stellt - ähnlich wie Übergabe-Parameter in prozeduralen Programmiersprachen - einen Verweis dar, für den im Problemlöseprozeß eine Instanziierung der benötigten Konzeptklasse eingesetzt wird. Dazu muß bei Erzeugung des Parameters angegeben werden, welche Objektstruktur er repräsentieren soll. „Parametertyp“ ist entweder eine der definierten Konzeptklassen oder einer der definierten Standardtypen. Der Parametertyp spezifiziert damit, von welcher Form die einzusetzende Instanz ist. Das so realisierte Typkonzept veranschaulicht die Graphik in Abbildung 8.

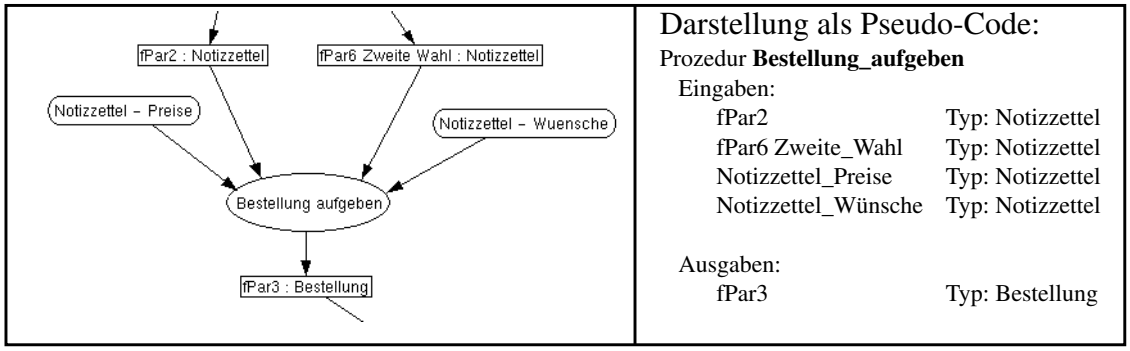



ABBILDUNG 8

Verwendung von Parametern als typisierte Ein- und Ausgaben: Die Aufgabe „Bestellung aufgeben“ benutzt als Eingabe zwei Parameter, die Ausgaben einer vorhergehenden Aufgabe sind, daneben noch zwei Konzeptinstanzen, alle vom Typ Notizzettel.

Entscheidender Fortschritt ist, daß mehrere Parameter vom gleichen Typ sein können. Somit können bei der Problemlösung *mehrere unabhängige* Instanziierungen *einer* Konzeptklasse erzeugt und weiterverarbeitet werden. Auch kann nun eine Aufgabe mehrere Instanziierungen *einer* Konzeptklasse nutzen oder erzeugen. Durch das Objekt „formaler Parameter“ sind die für verschiedene Instanzen vorbereiteten „Schablonen“ unterscheidbar. Zusätzlich trägt jeder Parameter spezifische Informationen (z.B. Namen zur Bildschirmanzeige, graphische Information), die eben nur diese Instanz der zugehörigen Klasse betreffen.

### 3.2.3 Mängel bei der Aufgabenmodellierung

CoMo-Kit gestattet zwar, Aufgaben in beliebig viele Unteraufgaben zu zerlegen, um durch deren Ausführung das Gesamtergebnis zu erhalten. Diese Zerlegung läßt sich beliebig verfeinern. Jedoch läßt sich jede Aufgabe auf höchstens eine Art und Weise zerlegen, das heißt, es kann nur eine „Methode“ spezifiziert werden, mit der die Aufgabe zu lösen ist. Insbesondere muß diese Entscheidung schon während der Spezifikation des konzeptuellen Modells getroffen werden. Bei Ausführung der Aufgabe muß sich der Bearbeiter genau an der spezifizierten Zerlegung orientieren, er hat keine Wahlmöglichkeiten. Abbildung 9 zeigt beispielhaft eine solche Zerlegung.

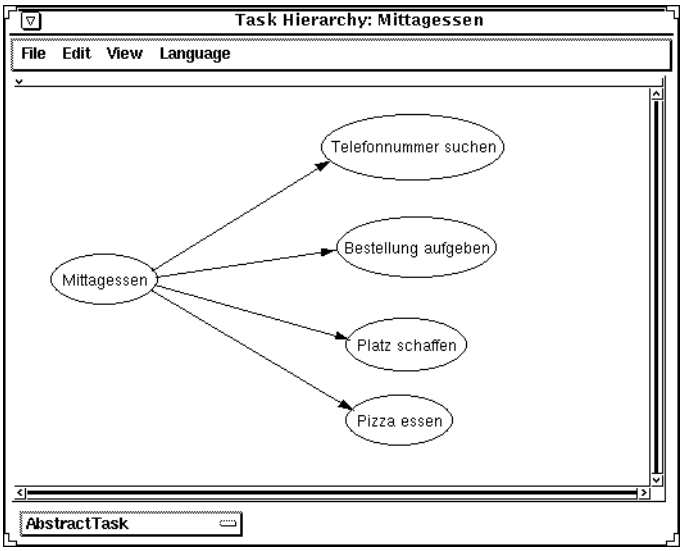



ABBILDUNG 9 Statische Aufgabenzerlegung in CoMo-Kit 2.0: Die Aufgabe „Mittagessen“ wird in die vier Teilaufgaben zerlegt. Der Datenfluß innerhalb dieser Teilaufgaben wurde bereits in Abbildung 7 dargestellt.

Diese statische Aufgabenzerlegung ist der Realität nicht angemessen. Reale Probleme lassen sich meist auf mehrere Arten lösen. Wobei das genaue Vorgehen oft erst dann festlegbar ist, wenn die konkrete Problemstellung (also die Aufgabe mit ihren Eingabedaten) vorliegt. Dazu wurde ein neuer Objekttyp, die „Methode“, in CoMo-Kit einge-

führt. Im folgenden werden die beiden Arten von Methoden vorgestellt.

### 3.2.4 Einführung von Methoden zur multiplen Aufgabenzerlegung

*Begriffsbestimmung*: Eine **komplexe Methode** beschreibt einen möglichen Lösungsweg einer komplexen Aufgabe. Durch komplexe Methoden werden Aufgaben in Unteraufgaben zerlegt. Im zugehörigen Datenfluß werden die auftretenden Wissenseinheiten den Teilaufgaben als Ein- oder Ausgaben zugeordnet. Dies spezifiziert implizit auch den Kontrollfluß. Die Unteraufgaben können wieder weiter durch Methoden zerlegt werden.

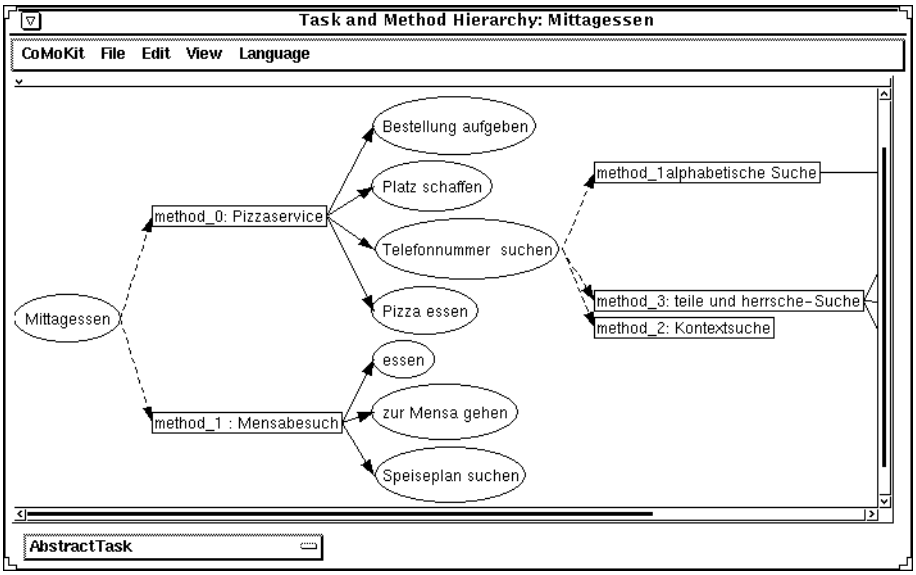



ABBILDUNG 10 Multiple Aufgabenzerlegung in CoMo-Kit 3.0: Für die Aufgabe „Mittagessen“ sind nun zwei mögliche Zerlegungen spezifiziert, auch „Telefonnummer suchen“ kann auf mehrere Arten zerlegt werden.

Einer komplexen Aufgabe können mehrere Methoden zugeordnet sein (wie der Aufgabe „Mittagessen“ in Abbildung 10). Erst bei der Ausführung muß aus dieser Menge der möglichen Aufgabenzerlegungen eine ausgewählt werden. Die geeignete Zerlegung wird also bei Bearbeitung des konkreten Problems anhand des Problemfallwissens ausgewählt. Neben der Methodenauswahl ist der Bearbeiter auch für das Delegieren der Unteraufgaben zuständig. Das heißt, er muß entscheiden, wie die Aufgaben auf die möglichen Bearbeiter verteilt werden, die Bearbeitung überwachen und bei möglichen Problemen (wenn zum Beispiel die Bearbeitung einer Teilaufgabe mißlingt) reagieren.

Wegen einer einheitlichen Begriffsbildung wurden neben den bisher beschriebenen komplexen Methoden noch atomare Methoden eingeführt.

*Begriffsbestimmung*: **Atomare Methoden** sind den atomaren Aufgaben zugeordnet. Sie beschreiben keine Aufgabenzerlegung, sondern eine Lösung der Aufgabe durch Instanziierung der Ausgabekonzeptklassen. Der Benutzer führt eine atomare Methode aus, indem er Resultate vom festgelegten Ausgabetyp erzeugt. Zu einer atomaren Aufgabe kann es nur genau eine atomare Methode geben. Atomare und komplexe Methoden können nie gleichzeitig zu *einer* Aufgabe gehören.

Eine atomare Aufgabe wird jedoch vom System als komplexe Aufgabe aufgefaßt, sobald der Benutzer eine oder mehrere komplexe Methoden zu dieser Aufgabe definiert. Entsprechend wird nach Löschen aller komplexen Methoden eine Aufgabe vom System wieder als atomare Aufgabe betrachtet.

Atomare Methoden sind sowohl für den Wissensingenieur beim Systementwurf als auch für die Benutzer keine „sichtbaren“ Objekte. Sie werden vom System selbständig solchen Aufgaben zugeordnet, zu denen noch keine komplexen Methoden definiert sind oder deren komplexe Methoden wieder gelöscht wurden. Diese Zuordnung illustriert Abbildung 11.

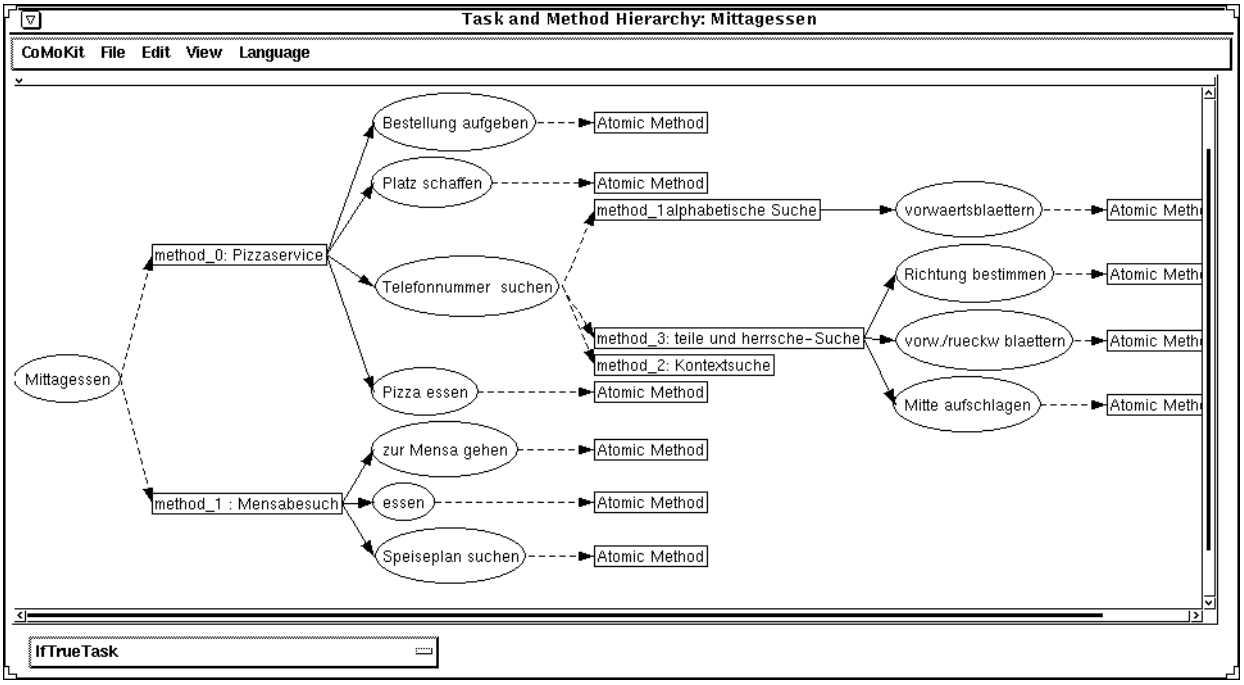



ABBILDUNG 11 Aufgabenzerlegung aus Abbildung 10 mit atomaren Aufgaben: Die in dieser Abbildung erkennbaren atomaren Methoden sind normalerweise für den Benutzer nicht sichtbar.

Wenn auch der Benutzer somit die Existenz atomarer Methoden nicht bemerkt, so sind diese Objekte doch (auch aus Implementierungs-Gesichtspunkten) nötig. So sind nämlich Routinen der Smalltalk-Klasse „Methode“ - egal ob es sich um die atomare oder komplexe Unterklassen handelt - allein für die tatsächliche Ausführung der Aufgabe zuständig. Außerdem entsprechen sich nun die Begriffs-

paare atomare Aufgabe / komplexe Aufgabe sowie atomare Methode / komplexe Methode.

Im folgenden wird meist vereinfachend nur noch von atomaren Aufgaben gesprochen, wobei dann auch die zugehörige atomare Methode gemeint ist.

## 3.3 Die Aufgabenzerlegung als Und/Oder-Baum

Die durch Aufgaben und Methoden definierte Zerlegung (Beispiel in Abbildung 10) kann man nach [Nilsson 80] als Und/Oder-Baum auffassen. Die Verbindungskanten zwischen Methoden und ihren Unteraufgaben nennt man Und-Kanten, da zur Ausführung einer komplexen Methode *sämtliche* Unteraufgaben gelöst werden müssen. Hingegen muß zur Ausführung einer Aufgabe *nur eine* der Methoden vollständig durchgeführt werden. Die Kanten von den Aufgaben zu den Methoden sind somit die Oder-Kanten. Die tatsächlich gewünschte Aufgabenzerlegung wird erst zur Laufzeit bestimmt. Die Abarbeitung einer Aufgabe kann man als Suche nach der dem aktuellen Problem angemessenen Aufgabenzerlegung im potentiellen „Lösungsraum“ begreifen. Dabei ist der Lösungsraum eben durch den Baum aus Aufgaben und Methoden beschrieben. Eine Problemlösung entspricht einem Hyperpfad[16] des Und/Oder-Baumes. Bei einer Suche kann man bekanntlich auf Irrwege geraten, so daß bestimmte Entscheidungen (z.B. Methodenauswahl) zurückgezogen und neue Entscheidungen getroffen werden müssen. Hiermit beschäftigt sich Abschnitt 4.7.

## 3.4 Editieren des konzeptuellen Modells

Die Spezifikation des konzeptuellen Modells erfolgt in der Regel, indem zuerst die Domänenkonzepte erfaßt werden und danach der Aufgaben- und Methodenbaum durch fortgesetzte Zerlegung der Aufgaben aufgebaut wird. Dann werden durch Zuordnen der Konzepte zu den Aufgaben die Datenflüsse spezifiziert. Dabei ist jedoch zu beachten, daß der *Datenfluß innerhalb der Zerlegung* einer Aufgabe konsistent ist mit dem *Datenfluß, in den die Aufgabe selbst eingeordnet ist*. Abschnitt 3.4.1 beschreibt, inwieweit der Benutzer hierfür selbst verantwortlich ist und welche Unterstützung durch das System geboten werden kann.

(16) Analog zu Pfaden in normalen Bäumen spricht [Nilsson 80] in Und/Oder-Bäumen von Hyperpfaden. Ein Hyperpfad beginnt an einem Wurzelknoten. Gehen von einem Knoten Und-Kanten aus, so wird der Hyperpfad durch all diese Und-Kanten fortgesetzt. Gehen von einem Knoten Oder-Kanten aus, so ist genau eine Kante als Fortsetzung des Hyperpfades auszuwählen.

Beim Erstellen des Aufgaben-Methoden-Baumes kann es ferner nötig sein, Teile zu löschen oder an eine andere Stelle des Baumes zu kopieren oder zu verschieben. Abschnitt 3.4.2 erläutert die hierzu angebotene Systemunterstützung.

### 3.4.1 Überwachen der Konsistenz von Ein- und Ausgaben

Werden zu einer Aufgabe Methoden definiert, und dann der Datenfluß innerhalb dieser Methoden spezifiziert, so sind bestimmte Regeln zu beachten, damit man einen konsistenten, der Domäne angemessenen Entwurf erhält. Dies ist insbesondere auch erforderlich, damit der in Kapitel 4 vorgestellte Interpreter das Netz korrekt verarbeiten kann. Innerhalb des Datenflusses müssen genau die Objekte vorkommen, die Eingabe oder Ausgabe der Aufgabe sind, die durch die Methode näher beschrieben wird. Diese Aufgabe wird im folgenden als übergeordnete Aufgabe der Methode bezeichnet. Dabei darf eine *Eingabe* der übergeordneten Aufgabe nur als *Eingabe* von Teilaufgaben verwandt werden. Entsprechend dürfen *Ausgaben* der übergeordneten Aufgabe nicht als Eingaben von Teilaufgaben Verwendung finden, sondern nur als Ausgaben.

Bei der Verwendung von Parametern ist zu beachten, daß genau der Parameter in den Datenfluß eingesetzt wird, der auch Ein- oder Ausgabe der übergeordneten Aufgabe war. Nur so werden später immer genau die richtigen Instanzen der zum Parameter gehörenden Konzeptklasse referenziert und von Aufgabe zu Unteraufgabe (als Eingabe) und wieder von Unteraufgabe zu übergeordneter Aufgabe (als Ausgabe) durchgereicht.

*Interne Parameter*

Innerhalb des Datenflusses dürfen neben diesen unbedingt zu benutzenden Parametern auch interne Parameter benutzt werden. Interne Parameter sind solche, die als Ausgabe einer Unteraufgabe und Eingabe einer anderen Unteraufgabe Daten weitergeben, aber auf höherer Ebene noch nicht benutzt wurden. Sie können vom Benutzer innerhalb eines Datenfluß in beliebiger Zahl erzeugt und verwendet werden. Abbildung 12 veranschaulicht den Begriff des internen Parameters.

Einfacher als bei den Parametern ist die Konsistenzkontrolle bei den Konzeptinstanzen. Diese können - da sie ja statisches Wissen in den Problemlöseprozeß einbringen - nur Eingabe von Teilaufgaben sein. Aber Konzeptinstanzen treten niemals als Ausgabe auf.

Diese notwendigen Bedingungen für einen konsistenten Systementwurf wurden bisher von CoMo-Kit nicht überwacht. Daher habe ich eine *neue Strategie* gesucht, mit der das System einen konsistenten Systementwurf unterstützt. Beim Erzeugen einer Methode wird überprüft, welche Ein- und Ausgaben die übergeordnete Aufgabe hat. Die Methode „merkt“ sich dann diese Konzeptinstanzen und Parameter,

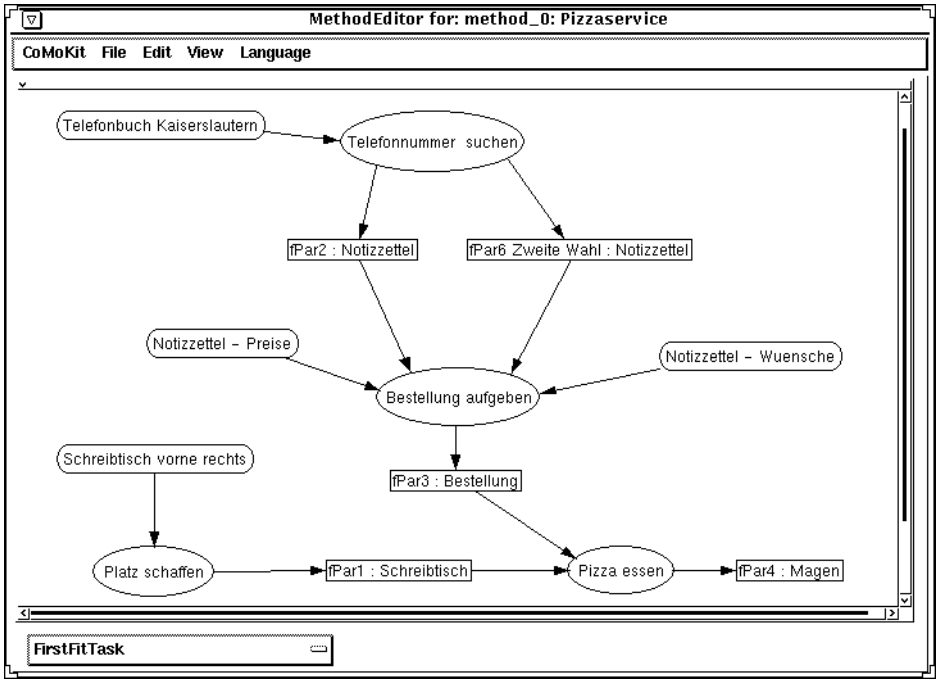



ABBILDUNG 12

Aufgabenstruktur in CoMo-Kit 3.0: Interne Parameter sind fPar1, fPar2, fPar3 und fPar6. Die Konzeptinstanzen „Telefonbuch Kaiserslautern“, „Schreibtisch vorne rechts“, „Notizzettel-Preise“ und „Notizzettel-Wünsche“ werden nur als Eingabeobjekte verwendet, der Parameter fPar4 als Ausgabeobjekt. Alle Methoden zur Aufgabe „Telefonnummer suchen“ müssen die Instanz „Telefonbuch Kaiserslautern“ im Datenfluß als Eingabeobjekt enthalten, entsprechend auch beide Parameter vom Typ Notizzettel als Ausgabeobjekt.

ebenso wie sie auch spätere Änderungen registriert. Bearbeitet der Entwickler den Datenfluß der Methoden, so stehen ihm diese Ein- und Ausgabeobjekte bereits innerhalb des Datenflusses zur Verfügung. Die Methode überwacht, daß die Parameter nur entsprechend ihrer Bestimmung verwendet werden, nicht jedoch als interne Parameter. Konzeptinstanzen können, wie erläutert, nur Eingabe von Aufgaben sein. Der Benutzer kann die automatisch eingefügten Objekte nicht aus dem Datenfluß löschen. Er muß erst die Zuordnung des Objektes zu der übergeordneten Aufgabe löschen. Damit verschwindet es dann auch (rekursiv fortschreitend) aus den Datenflüssen aller Methoden.

Bedingt durch diese Restriktion lassen sich allerdings keine Aufgaben modellieren, bei denen die zugehörigen Methoden jeweils unterschiedliche Ein- und Ausgaben benötigen. CoMo-Kit stellt also die Anforderung an die zu modellierende Domäne, daß alle Methoden zur Lösung einer Aufgabe gleiche Ein- und Ausgaben benutzen. Diese Restriktion ist in realen Domänen sicher oft nicht erfüllt. Sie zu umgehen hieße jedoch, dem Benutzer fast sämtliche Verantwortung für die Konsistenz des spezifizierten Netzes zu übertragen. Außerdem würden größere Änderungen am CoMo-Kit-Interpreter nötig.

### 3.4.2 Löschen und Wiederverwenden von Aufgaben und Methoden

Das Vorgehen bei der Entwicklung des wissensbasierten Informationssystems intendiert, daß der Systementwickler oder der Experte iterativ den Aufgaben- und Methodenbaum verfeinert.

*Löschen*

Dabei kann es durchaus vorkommen, daß bei Korrekturen bestimmte Aufgaben oder Methoden gelöscht werden müssen. Um zu verhindern, daß durch versehentliches Löschen ganze Zweige des Und/Oder-Baumes verloren gehen, wurde eine restriktive Strategie implementiert. Das Löschen kann nur von den Blättern des Baumes hin zur Wurzel erfolgen (konträr zur Erzeugung, bei welcher der Baum aus Aufgaben und Methoden top down von der Wurzel her aufgebaut wird). Eine Aufgabe kann nur gelöscht werden, wenn keine komplexen Methoden dazu definiert sind. Ebenso kann eine Methode nur gelöscht werden, wenn keine Unteraufgaben definiert sind.

*Aufgaben-Wiederverw.*

Soll eine Aufgabe mit ihren Methoden an anderer Stelle des Aufgaben-Methoden-Baumes wiederverwendet werden, bedeutet dies - aufgrund der fortschreitenden Zerlegung - eventuell das Kopieren mehrerer Ebenen des Baumes. Ganze Zweige könnten so wiederverwendet werden, um dem Benutzer eine Neueingabe dieser bereits definierten Strukturen zu ersparen. Trotzdem unterstützt CoMo-Kit bisher nicht die Wiederverwendung von Aufgaben und Methoden, da dabei folgende Probleme auftreten:

Jede Aufgabe referenziert Ein- und Ausgaben. Diese werden, nach der in Abschnitt 3.4.1 geschilderten Vorgehensweise, auch von den zugehörigen Methoden benutzt. Kopiert man jedoch eine Aufgabe an eine andere Stelle innerhalb der Aufgabenhierarchie, so geht die Zuordnung der Ein- und Ausgaben verloren. Demnach machen dann auch die Methoden keinen Sinn mehr, da ihnen ja nun die in ihrer Modellierung verwendeten Ein- und Ausgaben nicht mehr zur Verfügung stehen.

*Methoden-Wiederverw.*

Analoge Probleme treten beim Kopieren von Methoden mit den zugehörigen Teilaufgaben auf. Dadurch geht nämlich die Zuordnung der Methode zu der übergeordneten Aufgabe und folglich auch zu deren Ein- und Ausgabedaten verloren. Die kopierte Methode geht von bestimmten Eingaben aus und stellt bestimmte Ausgaben zur Verfügung. Im neuen Kontext werden aber nicht unbedingt genau diese Parameter bzw. Konzeptinstanzen auch angeboten und übernommen.

*Implementierung*

Da CoMo-Kit bisher die Wiederverwendung von Aufgaben und Methoden nicht unterstützt, müssen diese jeweils gänzlich neu erzeugt und definiert werden. Denkbar wäre aber eine Strategie, bei der Aufgaben oder Methoden, insbesondere nach Löschen der übergeordneten Objekte, an anderer Stelle wieder eingesetzt werden kön-

nen. Dabei könnten die problemverursachenden Ein- und Ausgabeobjekte rekursiv fortschreitend aus dem zu kopierenden Zweig gelöscht werden. Es blieben dann nur die internen Parameter erhalten.

Eine andere Strategie ist, keine Objekte aus dem zu kopierenden Zweig zu löschen. Der Benutzer ist dann selbst verantwortlich, daß das kopierte Objekt nur an einer Stelle eingesetzt wird, an der er wieder konsistente Zuordnungen treffen kann. Diese Strategie läßt dem Benutzer weitgehende Möglichkeiten, allerdings stellt sie höhere Ansprüche an ihn, da er die Konsistenz seiner Modellierung selbst überwachen muß.

Eine bessere Benutzerunterstützung würde das System gewährleisten, wenn es das Kopieren von Aufgaben und Methoden an Stellen gestattet, die von den Ein- und Ausgabetypen her kompatibel sind. Für die genaue Zuordnung bliebe weiter der Benutzer verantwortlich. Durch die Typüberprüfung würde aber eine gewisse Konsistenz garantiert.

KAPITEL 4

# Der Interpreter zur Operationalisierung des konzeptuellen Modells

*Dieses Kapitel beschreibt den neu entwickelten CoMo-Kit Interpreter. Dieser ermöglicht die Ausführung des zuvor spezifizierten konzeptuellen Modells.*

Im ersten Abschnitt wird der Ablauf der Operationalisierung von konzeptuellen Modellen erläutert. Dabei wird besonders auf die Aufgaben der Benutzer eingegangen. Abschnitt 4.2 stellt die Gesamtarchitektur des Interpreters vor, die Funktionen der einzelnen Komponenten werden grob erklärt. Danach wird die Prozeßverwaltung erläutert, die für das Zusammenwirken der Komponenten des Interpreters mit den Benutzern des Systems verantwortlich ist. Abschnitt 4.4 beschreibt, wie das bei der Aufgabenlösung anfallende Problemfallwissen behandelt wird. Der fünfte Abschnitt erklärt das von uns entwickelte Zustandsmodell für Aufgaben. Die Schnittstelle zwischen den Hauptkomponenten des Interpreters wird in 4.6 vorgestellt. Der letzte Abschnitt gibt einen groben Überblick über die Aufgaben und die grobe Funktionsweise der Komponente zur Abhängigkeitsverwaltung.

## 4.1 Ablauf der Operationalisierung

Die Ausführungskomponente, der CoMo-Kit Interpreter, wird zur interaktiven Abarbeitung der spezifizierten Aufgaben eingesetzt. Durch Ausführung wird das konzeptuelle Modell validiert, damit können die Anwender des zu entwickelnden Informationssystems testen, ob die Wissensakquisition und -modellierung korrekt verlaufen ist. Außerdem wird mit Hilfe des Interpreters die reale Ausführung der Aufgaben simuliert. Voraussetzung ist, daß die bereits beschriebenen Spezifikationen des konzeptuellen Modells der betrachteten Domäne vorliegen (auf während der Problemlösung noch mögliche Ergänzungen des konzeptuellen Modells geht

Abschnitt 5.3 ein). Dazu müssen folgende Schritte durchgeführt worden sein:

- Beschreiben der Wissenseinheiten (Konzepte, Konzeptinstanzen)
- Beschreiben der Aufgaben, beliebige Detaillierung durch Aufgabenzerlegung mit Hilfe komplexer Methoden
- Beschreiben der Datenflüsse innerhalb einer komplexen Methode (durch Zuordnen von Konzeptinstanzen und formalen Parametern als Ein- und Ausgaben zu den Teilaufgaben)
- Zuordnen einer Gruppe von Agenten zu jeder Aufgabe (die Agenten, die diese Aufgabe bearbeiten können)
- Formalisierung einzelner, atomarer Aufgaben, die vom Computer durchgeführt werden sollen

Intention von CoMo-Kit ist es, eine kooperative, verteilte Problemlösung zu ermöglichen. Unter der Kontrolle einer Steuerungskomponente arbeiten die Agenten zwar gemeinsam, aber an verschiedenen Arbeitsplätzen (Terminals, Rechnern), eventuell sogar an verschiedenen Orten. An den Arbeitsplätzen müssen also Programme ablaufen, welche die Benutzerprozesse darstellen und mit einer zentralen Steuerungskomponente kommunizieren. Diese kann als Programm an einem zentralen Rechner installiert sein. Die Wissensbasis, die alle im konzeptuellen Modell definierten Objekte enthält, kann ebenfalls an einer zentralen Stelle vorhanden sein. Denkbar - und in einer älteren CoMo-Kit Version bereits realisiert - ist aber auch, die Wissensbasis als verteilte Datenbank zu repräsentieren.

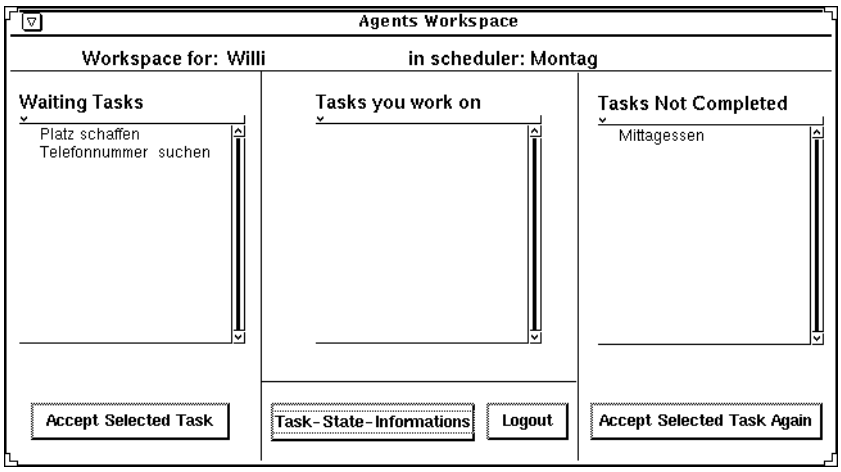



ABBILDUNG 13 Arbeitsfenster eines Agenten: Man sieht die Liste der für den Benutzer Willi ausführbaren Aufgaben. Daneben werden auch die in Bearbeitung befindlichen Aufgaben (Tasks you work on) und Aufgaben, deren Bearbeitung unterbrochen und zurückgestellt wurde (Tasks not Completed) angezeigt. Außerdem kann der Benutzer weitere Funktionen aufrufen, die in Anhang 2.2.1 beschrieben sind.

Der grobe Ablauf der Ausführung eines konzeptuellen Modells gestaltet sich dann als Wechselspiel zwischen den Benutzern und der zentralen Steuerungskomponente. Ein ausgezeichneter Systembenut-

zer (siehe Abschnitt 4.3) startet die Bearbeitung einer Aufgabe, indem er sie an einen oder mehrere Bearbeiter (die sie ausführen können) delegiert und damit zur Bearbeitung freigibt. Das System übernimmt danach die Verwaltung des weiteren Bearbeitungsprozesses. Formalisierte Aufgaben werden automatisch an die ihnen zugeordneten Computer übergeben und von diesen berechnet[17]. Die von menschlichen Agenten zu bearbeitenden Aufgaben werden im Dialog mit den jeweiligen Sachbearbeitern ausgeführt. Dazu werden dem Bearbeiter an seinem jeweiligen Arbeitsplatz die Aufgaben, die er bearbeiten kann, angezeigt (siehe Abbildung 18, Liste „Waiting Tasks“). Dies geschieht natürlich erst dann, wenn zu einer Aufgabe alle zu ihrer Bearbeitung benötigten Eingaben vorhanden sind. Der Benutzer kann die Aufgabe auswählen, die er als nächste bearbeiten will. Er muß danach folgende Tätigkeiten ausführen:

Bei der Bearbeitung einer komplexen Aufgabe:

- Auswahl der zu benutzenden Methode im Methoden-Auswahl Fenster (siehe Abbildung 14).

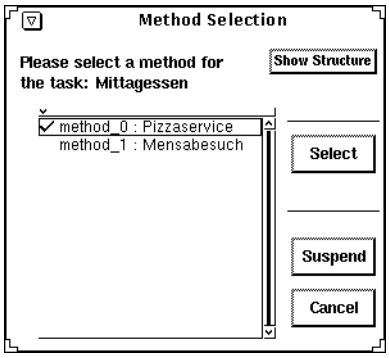



ABBILDUNG 14 Methoden-Auwahl Fenster

- Verteilen der Arbeit durch Delegieren der Teilaufgaben (Abbildung 15, Liste „Subtasks to delegate“) an eine oder mehrere Personen aus der Liste der möglichen Bearbeiter (Abbildung 15, Liste „Possible Agents“)
- Überwachen des Lösungsprozeß (Abbildung 15, Listen am unteren Bildrand)
- Auflösen von eventuell auftretenden Blockierungen, die beispielsweise dadurch entstehen können, das einzelne Teilaufgaben von ihren Bearbeitern nicht innerhalb einer erwarteten Bearbeitungszeit fertiggestellt werden oder überhaupt nicht durchgeführt werden können.

Bei der Bearbeitung einer atomaren Aufgabe:

(17) Die Bearbeitung formalisierter Aufgaben ist in CoMo-Kit 3.0 noch nicht implementiert.

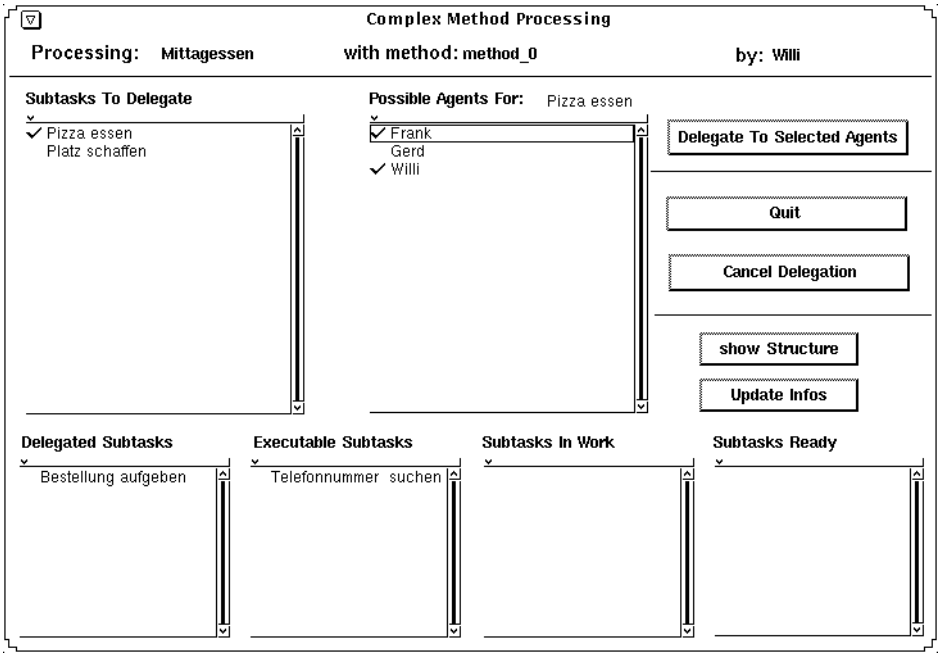



ABBILDUNG 15 Fenster zur Bearbeitung einer komplexen Methode

- Bearbeitung der Aufgabe nach einer als Hypertext gespeicherten Bearbeitungsvorschrift durch die Eingabe der Resultate (siehe Abbildung 16)
- Auflösen von eventuell auftretenden Blockierungen, die eine Rücknahme bereits erzeugter Resultate erforderlich machen

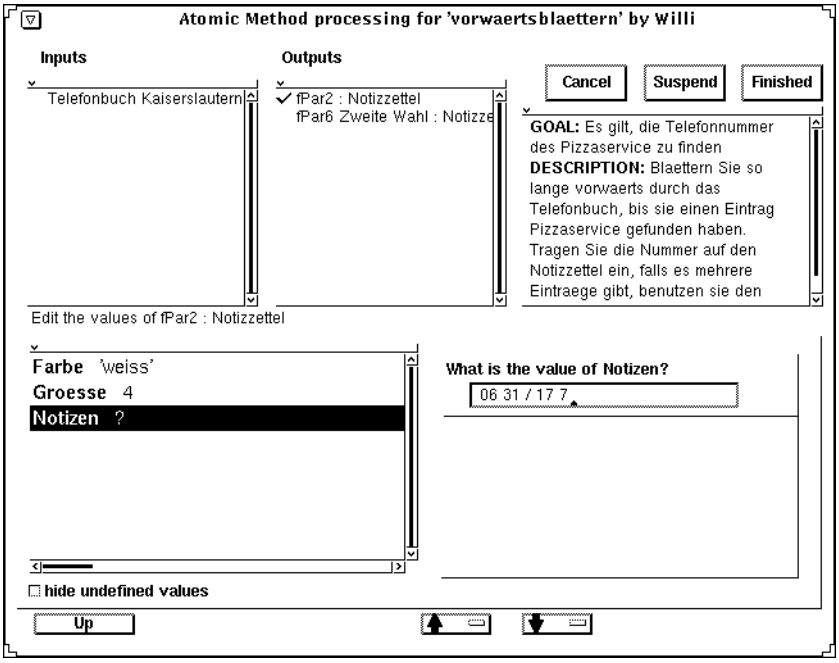



ABBILDUNG 16 Fenster zur Bearbeitung einer atomaren Methode

Jederzeit kann der Benutzer die Bearbeitung einer komplexen oder einer atomaren Aufgabe unterbrechen. Das entsprechende Bearbeitungsfenster verschwindet daraufhin und die Aufgabe wird im Arbeitsfenster (Abbildung 18) in der Liste „Tasks Not completed“ angezeigt. Dort kann sie der Benutzer erneut akzeptieren und die Bearbeitung zu Ende führen. Von diesem zeitweiligen Unterbrechen der Bearbeitung ist der Abbruch einer Aufgabe (Cancel-Button) zu unterscheiden. Nach einem Abbruch wird die Aufgabe als nicht lösbar betrachtet, was - wie noch zu erläutern ist - Auswirkungen auf den gesamten Problemlösungsprozeß hat.

Die abgebildeten Fenster zur Bearbeitung der Aufgaben sind in Anhang 2.3 und Anhang 2.4 näher beschrieben. Die Steuerungskomponente des Systems lenkt mit einer Reihe von Dienstleistungen den gesamten Arbeitsablauf. Diese Dienste werden im weiteren Verlauf dieses Kapitels beschrieben. Zuerst aber wird im nächsten Abschnitt eine Übersicht über die Interpreter-Architektur gegeben.

## 4.2 Architektur des Interpreters

Die Übersicht über die Architektur der Interpreters in Abbildung 17 läßt drei wesentliche Teile erkennen. Einmal die Scheduler-Komponente, daneben die an der Problemlösung beteiligten Agenten, in der Zeichnung als Clients bezeichnet. Weiter erkennt man das konzeptuelle Modell, das die Ausgangsdaten für den Interpreter enthält.

*konzeptuelles Modell*

Im konzeptuellen Modell ist, wie bereits in Kapitel 2 erläutert, das gesamte Wissen über die zu lösende Aufgabe, die dazu benötigten Wissenseinheiten und das „Wie“ der Aufgabenlösung abgelegt. Auf dieses während der Problemlösung in der Regel statische Wissen greifen die übrigen Komponenten lesend zu, da sie dort die zur Operationalisierung notwendigen Informationen finden. Daneben haben die Clients noch die (mit der gestrichelten Linie angedeutete) Möglichkeit, durch einen Schreibzugriff das konzeptuelle Modell zu verändern. Diese Systemerweiterung, die zur Änderung des Aufgaben- und Methodenbaumes während der Ausführungszeit dient, ist in Abschnitt 5.3 näher erläutert. Im folgenden wird das konzeptuelle Modell auch als die Wissensbasis des Interpreters bezeichnet.

*Scheduler*

Der Scheduler ist der bisher als Steuerungskomponente bezeichnete zentrale Teil des Interpreters. Er steuert und verwaltet den Problemlösungsprozeß, stellt also einen Problemlöser dar, der die Benutzer bei der gemeinsamen Bearbeitung der Aufgaben anleitet. Hierzu bietet er umfangreiche Dienstleistungen, die von den Agenten genutzt werden. Funktionell lassen sich drei Hauptaufgaben der Problemlösungskomponente unterscheiden.

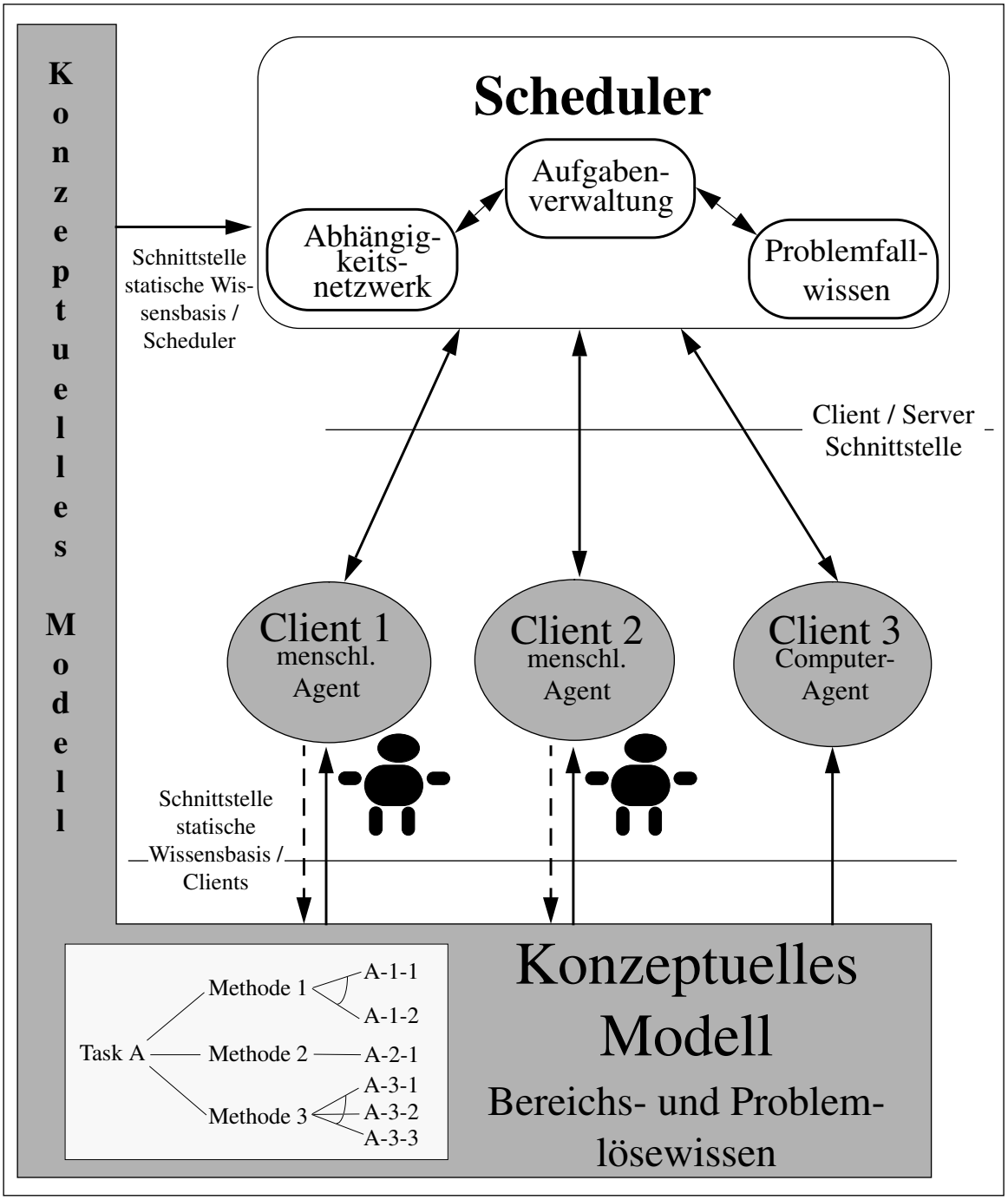



ABBILDUNG 17 Interpreter - Architektur

Einmal muß sie das während der Problemlösung erzeugte Problemfallwissen (die Eingaben und Resultate atomarer Aufgaben) verwalten. Dies wird in Abschnitt 4.4 erläutert.

Als zweites obliegt dem Scheduler die Verwaltung der Aufgaben. Er muß kontrollieren, welche Aufgaben sich in welchen Bearbeitungs-

zuständen befinden und die Übergänge zwischen den Zuständen kontrollieren (siehe Abschnitt 4.5). Diese Übergänge werden in erster Linie durch Entscheidungen der Benutzer (beispielsweise bei Methodenauswahl oder Delegierung) und die durch den Datenfluß festgelegten zeitlichen Abhängigkeiten zwischen den Aufgaben verursacht.

Außerdem ist der Scheduler verantwortlich, die Konsistenz der Lösung zu prüfen und zu erhalten. Dazu wird ein Netzwerk verwaltet, in dem die Datenflußabhängigkeiten sowie die Abhängigkeiten aufgrund von Entscheidungen abgebildet werden. Bei auftretenden Inkonsistenzen kann mittels dieses Netzwerks wieder ein konsistenter Zustand hergestellt werden (siehe Abschnitt 4.7).

*Schnittst. zur Wissensbasis* Zur Erfüllung dieser Aufgaben muß der Scheduler alle spezifizierten Teile des konzeptuellen Modells kennen. Dazu wurde eine Schnittstelle zwischen der Wissensbasis und dem Scheduler definiert. Der Zugriff ist nur lesend, die Problemlösungskomponente kann das abgelegte Wissen nicht verändern. Gelesen werden sukzessive die aktuell benötigten Informationen, was bei Voranschreiten des Lösungsprozeß zu jeweils neuen Datenzugriffen führt.

*Clients* Mit dem Scheduler arbeitet eine beliebige Anzahl von Agenten zusammen. Abbildung 17 zeigt beispielhaft zwei menschliche und einen Computer-Agenten. Diese müssen innerhalb des konzeptuellen Modells als Basisobjekt „Agent“ spezifiziert sein, aber es müssen nicht unbedingt alle dort definierten Agenten an der Problemlösung mitarbeiten. Sie führen eine verteilte Problemlösung durch und arbeiten, meist an verschiedenen Arbeitsplätzen, gemeinsam an der Lösung des Problems. Ihre Arbeit wird vom Scheduler gesteuert, der ihnen alle Informationen über den aktuellen Problemlösungsprozeß zur Verfügung stellt, die von ihnen benötigt werden. Informationen, die für einen bestimmten Agenten nicht relevant sind, werden vom System ausgeblendet. Der Scheduler ist somit ein Server, der mit mehreren Clients - den Agenten - kommuniziert.

*Client/Server-Schnittstelle* Die Client-Server-Kommunikation wird über eine weitere Schnittstelle abgewickelt. Deren Spezifikation ist in Anhang 3 abgedruckt, sie wird in Abschnitt 4.6 näher erläutert. Mit den dort definierten Nachrichten werden von den Agenten Informationen über den Stand der Problemlösung beim Scheduler abgefragt, Zustandsänderungen vom Scheduler den Agenten mitgeteilt und Ergebnisse an den Scheduler geschickt. Der Informationsaustausch bezieht sich somit auf die aktuelle Problemlösung - das Problemfallwissen und die Verwaltung der Aufgaben im Scheduler.

Sofern die Agenten Informationen benötigen, die zum Bereichs- und Problemlösewissen gehören, müssen diese aus der Wissensbasis gelesen werden. Dazu wurde die Schnittstelle zwischen dem konzeptuellen Modell als statischer Wissensbasis und den Clients eingerich-

tet. Sie verhindert, daß große Informationseinheiten aus der Wissensbasis über den Umweg des Schedulers an die Clients laufen. Wenn nämlich die Benutzer an räumlich verteilten Arbeitsplätzen sitzen, würde dieser Nachrichtenfluß eine erhebliche Auslastung der Kommunikationswege zwischen Scheduler und Clients zur Folge haben. Um einen sparsamen Nachrichtenaustausch zu ermöglichen, gibt der Scheduler stattdessen nur Kennungen an die Clients weiter. Mit deren Hilfe können die Clients die Informationen direkt aus der Wissensbasis lesen, die wie bereits erläutert als verteilte Datenbank implementiert werden kann. Dadurch werden die über die Client/Server-Schnittstelle laufenden Informationspakete größenmäßig minimiert.

Nicht durch solche Kennungen referenziert wird das Problemfallwissen, das wie erläutert nur innerhalb des Schedulers, nicht aber in der Wissensbasis vorgehalten wird. Eingaben und Ergebnisse atomarer Aufgaben werden somit direkt über die Client/Server-Schnittstelle ausgetauscht.

Der nächste Abschnitt beschreibt die Auswirkungen der gewählten Architektur auf die Implementierung des Interpreters als Mehrprozeßsystem.

## 4.3 Prozeßverwaltung von CoMo-Kit

Bereits erläutert wurde, daß die verteilte Problemlösung mehrerer Agenten in Zusammenarbeit mit einem zentralen Scheduler bedingt, dezentrale Client-Prozesse einzurichten. Diese laufen an den Arbeitsplatzrechnern der Benutzer ab. Zur Unterstützung des dezentralen Zugriffs sollten die im konzeptuellen Modell abgelegten Daten mittels eines verteilten Datenbanksystem verwaltet werden.

In CoMo-Kit 3.0 werden die Verteilung der Wissensbasis und das Mehrprozeßsystem nur simuliert, das gesamte System läuft jedoch innerhalb eines Smalltalk-Prozeß ab. Dies hat zur Folge, daß alle Agenten an einem einzigen Bildschirm arbeiten müssen - eine echte Verteilung der Benutzer auf mehrere Arbeitsplätze wird nicht unterstützt. Dies führt schnell dazu, daß die Darstellung an diesem Bildschirm vollkommen unübersichtlich wird. Um diesen Mangel zu beheben, wurden bereits mehrere mögliche Strategien zur Verteilung der Benutzerprozesse entwickelt, diese werden in Abschnitt 5.2 vorgestellt.

Die geplante Verteilung der Client-Prozesse auf verschiedene Rechner wurde beim Entwurf des Interpreters bereits berücksichtigt. Einmal wurde die Client/Server-Schnittstelle bereits auf eine geplante Verteilung hin optimiert (siehe Abschnitt 4.6). Weiter wurden, unterstützt durch das objektorientierte Systemdesign, bereits die Client-

funktionen in simulierten „Prozessen“ zusammengefaßt. Somit konnte eine Prozeßverwaltung realisiert werden, die leicht zu einem tatsächlichen Mehrprozeß-System hin erweitert werden kann.

Ausgangspunkt für die Prozeßverwaltung ist das konzeptuelle Modell. Es ist als CoMo-Kit Netzwerk realisiert, das die verschiedenen Basisobjekte enthält und wiederum auf ein Netzwerk der Hypertextmaschine abgebildet wird (siehe Abschnitt 2.4).

Ein ausgezeichneter Benutzer - der CoMo-Kit Superuser - kann einen zentralen Prozeß starten, der die Netzwerke und Scheduler verwaltet. Dieser Prozeß ermöglicht es, Netzwerke zu erzeugen und zu editieren, also das konzeptuelle Modell zu spezifizieren (Abbildung 18, links). Ferner kann dieser Benutzer einem Netzwerk beliebig viele Scheduler zuordnen, diese repräsentieren dann unterschiedliche Ausführungen der im konzeptuellen Modell beschriebenen Aufgaben. Der Superuser startet einen Schedulerprozeß, indem er angibt, welche Aufgaben initialisiert und an welche Bearbeiter diese delegiert werden sollen (Abbildung 18, rechts).

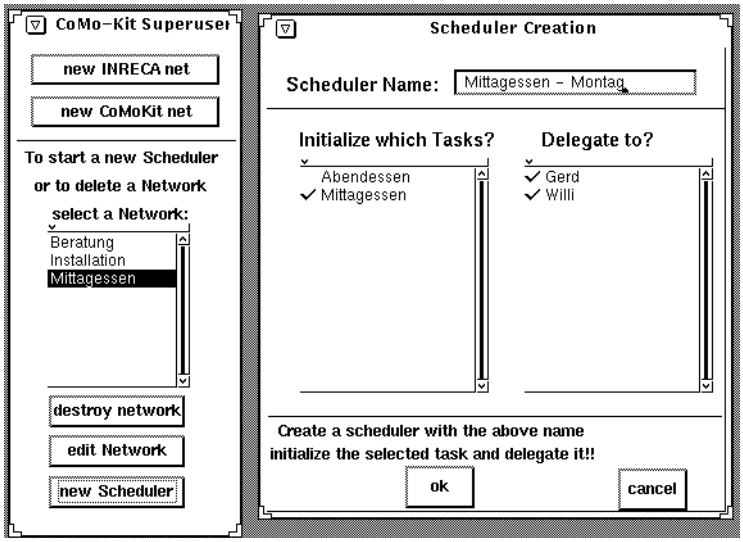



ABBILDUNG 18 Arbeitsfenster des Superuser: Im linken Fenster erzeugt und verwaltet der Superuser die Netzwerke, im rechten Fenster erzeugt er neue Scheduler

Die einzelnen Benutzerprozesse sind vom zentralen Schedulerprozeß isoliert. Sie erhalten jedoch über eine gemeinsame Datenstruktur[18], in der alle Netzwerke und Scheduler verzeichnet sind, vom Superuser-Prozeß die Information über laufende Scheduler, also über im Moment stattfindende Aufgabenbearbeitungen. Über seinen Benutzerprozeß kann sich ein Agent bei beliebig vielen Scheduler anmelden, um an der jeweiligen Aufgabenlösung mitzuarbeiten. Dazu muß

(18) die Smalltalk-Klassenvariable NetworkList CurrentNetList mit der Zugriffsmethode NetworkList currentNetList

er sich über einen Login-Vorgang identifizieren, außerdem überprüft der Scheduler, ob der Agent im bearbeiteten konzeptuellen Modell erfaßt ist und somit zur Problemlösung beitragen kann. Anschließend kann der Agent die für ihn anstehenden Aufgaben bearbeiten. Dazu dienen die bereits in Abschnitt 4.1 beschriebenen Fenster, die der Benutzerprozeß verwaltet.

## 4.4 Problemfallwissen

Zur Bearbeitung einer atomaren Aufgabe benötigt der Benutzer Eingaben und er erzeugt Ausgaben, zusammengefaßt das *Problemfallwissen*. Es „enthält die Informationen, die im Verlauf der Lösung eines konkreten Problems (eines Falles) erhoben bzw. abgeleitet werden. Problemfallwissen bildet die dynamische Wissensbasis“[19]. Ihm gegenüber steht in der statischen Wissensbasis - dem konzeptuellen Modell - das Bereichs- und das Problemlösewissen. Dieses wird bereits bei der Wissensakquisition erhoben. Das *Bereichswissen* besteht aus den grundlegenden Domänenstrukturen. „Es definiert den Rahmen der Wissensrepräsentation. *Problemlösewissen* umfaßt die Fakten und Zusammenhänge der Anwendung. Es füllt sozusagen den durch das Bereichswissen aufgespannten Rahmen.“[20]

Das Problemfallwissen muß für jede Problemlösung separat verwaltet werden, daher wurde in den Scheduler eine Komponente eingebaut, die das Problemfallwissen speichert. Konkret handelt es sich bei den dort abzulegenden Daten um formale Parameter und um Konzeptinstanzen, die Eingaben oder Ergebnisse atomarer Aufgaben sind.

*formale Parameter*

Parameter referenzieren eine Konzeptklasse, während der Problemlösung wird für jeden Parameter eine Instanz dieser Klasse erzeugt. Dies übernimmt der Scheduler, er stellt dem Bearbeiter der entsprechenden Aufgabe die Instanz zur Verfügung. Nach Abschluß der Aufgabenbearbeitung übernimmt der Scheduler wieder die Instanz - nun sind die Slots mit den Ergebnissen belegt - und speichert sie. Dabei verwaltet er die abgespeicherten Instanzen so, daß bei späteren Referenzen des gleichen Parameters auch die richtige Instanz zur Verfügung gestellt wird.

*Konzeptinstanzen*

Die Verwaltung der in die Datenflüsse eingebundenen Konzeptinstanzen reduziert sich für den Scheduler auf ein einfaches Kopieren der jeweiligen Instanz aus dem konzeptuellen Modell und Weitergabe an die Benutzer.

(19) [Maurer 93] S. 26

(20) [Haupenthal 93] S. 12

Wichtig ist, daß es durch Rücknahme von Entscheidungen und Umplanungen dazu kommen kann, daß Ergebnisse einer Aufgabe ungültig werden. Dies berücksichtigt der Scheduler, wobei er solche ungültigen Instanzen weiterhin verwaltet, so daß er sie bei Bedarf wieder benutzen kann. Die hierzu benötigten Rückzugsmechanismen werden in Abschnitt 4.7 erläutert.

Die Verwaltung des Problemfallwissens dient nicht nur dazu, für Bearbeiter von Aufgaben die Eingabedaten zur Verfügung zu stellen. Ein wichtiger Nebenaspekt ist, daß damit auch die durch die Datenflüsse gegebenen zeitlichen Abhängigkeiten zwischen den Aufgaben modelliert werden können. Teilaufgaben dürfen nämlich erst dann bearbeitet werden, wenn die Eingabedaten vorliegen. Dies zu überwachen gehört ebenfalls zu den Aufgaben der Abhängigkeitsverwaltung, beschrieben in Abschnitt 4.7. Zunächst wird aber im nächsten Abschnitt die grundlegende Aufgabenverwaltung vorgestellt.

## 4.5 Aufgabenverwaltung und Bearbeitungszustände der Tasks

Während der Problemlösung muß der CoMo-Kit Interpreter den aktuellen Zustand des Lösungsprozeß verwalten. Diese Aufgabe wird von der zentralen Interpreterkomponente, dem Scheduler, übernommen. Er speichert zu jeder Aufgabe den aktuellen Zustand und überwacht die Zustandsübergänge. Abbildung 19 zeigt alle möglichen Zustände, in denen sich eine Aufgabe befinden kann, außerdem die Übergänge mit auslösenden Ereignissen.

*delegated*

Der Scheduler bezieht eine im konzeptuellen Modell definierte Aufgabe dann in seine Verwaltung ein, wenn diese initialisiert und delegiert worden ist. Dies geschieht entweder ausgelöst durch den CoMo-Kit Superuser beim Initialisieren eines Schedulers (siehe Abschnitt 4.3) oder bei der Bearbeitung einer komplexen Methode (siehe Abschnitt 4.1). Durch Delegieren wird abhängig von der aktuellen Situation festgelegt, welche Agenten die Aufgabe bearbeiten sollen. Die Aufgabe befindet sich dann im Zustand `delegated`. Ob die Aufgabe nun bereits ausführbar ist, hängt von den Daten ab, die zur Ausführung benötigt werden.

*waiting*

Eine Aufgabe ist dann ausführbar, wenn alle Eingabedaten zur Verfügung stehen. Es muß für alle Eingabewerte im Problemfallwissen eine gültige Zuordnung vorhanden sein. Dies ist erst dann der Fall, wenn alle im Datenfluß vor dieser Aufgabe stehenden Aufgaben erfolgreich bearbeitet wurden und Ausgabewerte erzeugt haben. Die Aufgabe geht dann vom Zustand `delegated` in den Zustand `waiting` über. Sie wartet darauf, von einem Benutzer bearbeitet zu werden.

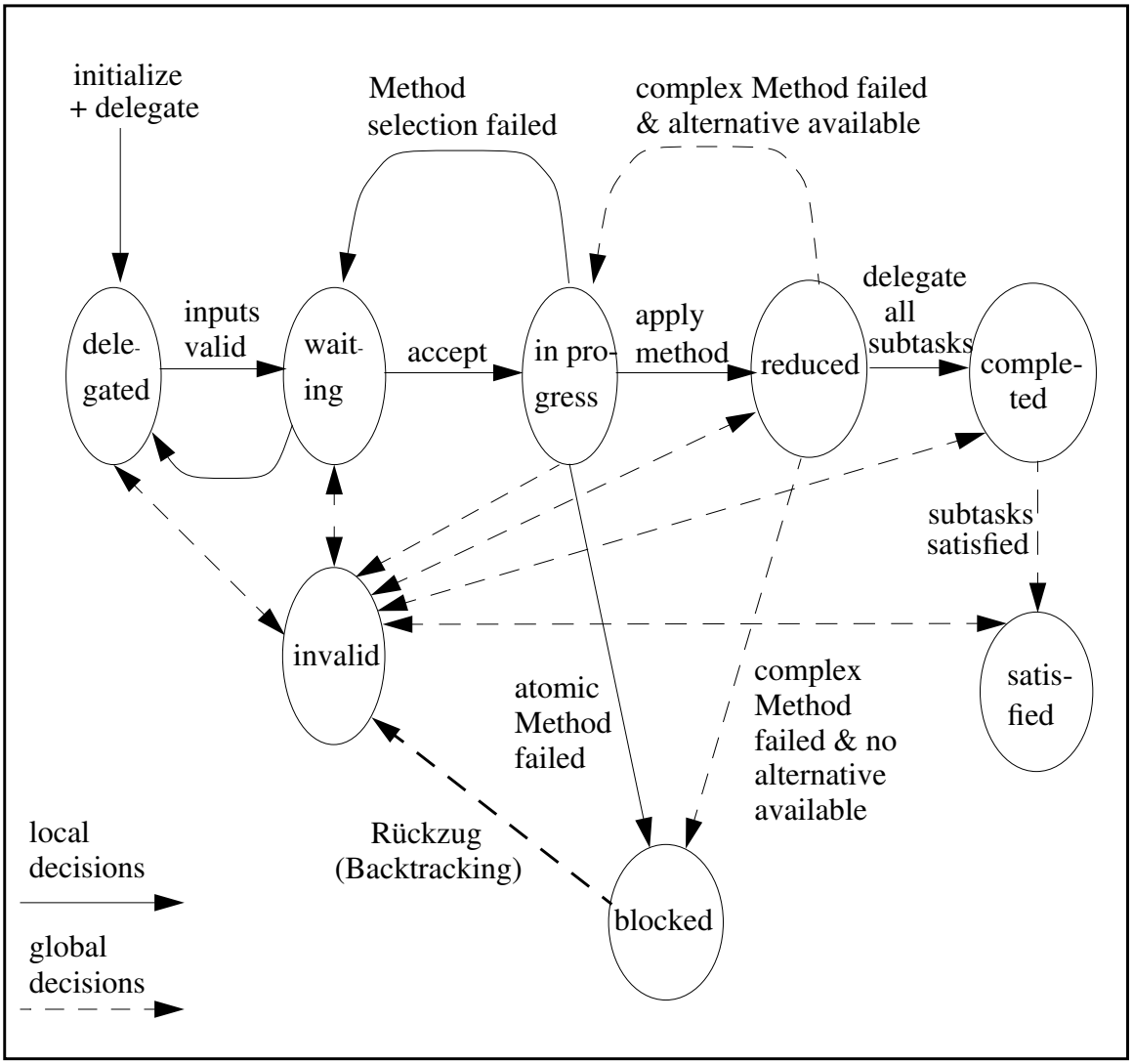



ABBILDUNG 19 Zustandsdiagramm für Aufgaben

Sobald eine Aufgabe im Zustand `waiting` ist, wird sie allen Agenten, die beim Scheduler angemeldet sind, als ausführbar angezeigt - sofern sie an den entsprechenden Benutzer delegiert wurde. Benutzer, die sich neu anmelden, erhalten ebenfalls in ihrem Arbeitsfenster die Übersicht über alle für sie ausführbaren Aufgaben.

*in progress*

Ein Benutzer kann eine Aufgabe in seinem Arbeitsfenster auswählen. Er akzeptiert diese Aufgabe und wird damit zu ihrem Bearbeiter. Die Aufgabe gelangt in den Zustand `in progress`. Gleichzeitig sorgt der Scheduler durch eine Mitteilung an alle angemeldeten Benutzer dafür, daß kein anderer Agent mehr diese Aufgabe akzeptieren kann. Der Bearbeiter muß nun die Aufgabe lösen. Dazu muß er eine Methode anwenden.

*reduced*

Der Zustand `reduced` wird durch erfolgreiche Methodenanwendung erreicht. Ist die Aufgabe atomar, so gibt es genau eine ihr zugeordnete atomare Methode. Diese wird angewandt, indem der Benutzer im vom Scheduler zur Verfügung gestellten Fenster (Atomic Method Processing Window, siehe Anhang 2.4.1) die Resultate einträgt. Wählt der Benutzer den Finished-Button an, so gilt die Anwendung der atomaren Methode als abgeschlossen.

Einer komplexen Aufgabe sind eine oder mehrere komplexe Methoden zugeordnet. Bei einer Methode ist die Auswahl trivial und wird automatisch durchgeführt. Bei mehreren komplexen Methoden wird der Benutzer vom Scheduler veranlaßt, eine auszuwählen (Method Selection Window, siehe Anhang 2.3.1). Durch die Methodenwahl wird die komplexe Aufgabe in den Zustand `reduced` überführt.

Bearbeitete Aufgaben gelangen also einheitlich in den Zustand `reduced`, der Zustandsübergang manifestiert sich aber unterschiedlich, je nach Aufgabentyp. Das Resultat einer atomaren Aufgabe, die in den Zustand `reduced` gelangt, ist die Zuordnung konkreter Werte an die Ausgabekonzepte. Aus dem Übergang einer komplexen Aufgabe nach `reduced` resultiert hingegen die durch die Methodenauswahl bewirkte Aufspaltung in Teilaufgaben. Dabei werden neben den neuen Teilaufgaben auch die Konzeptspezifikationen in die Verwaltungskomponente übernommen, deren Instanzen später das Problemfallwissen aufnehmen.

*completed*

Zur Bearbeitung einer komplexen Aufgabe gehört neben der Methodenwahl das Delegieren der Unteraufgaben, in welche die Methode die Aufgabe aufspaltet (Complex Method Processing Window, siehe Anhang 2.3.2). Erst wenn alle Unteraufgaben delegiert sind, ist die Bearbeitung der komplexen Methode vollständig abgeschlossen. Die Aufgabe ist dann `completed`.

*satisfied*

`Completed` beschreibt nur, daß alle Teilaufgaben delegiert wurden. Es trifft jedoch keine Aussage über den Bearbeitungsstand der Teilaufgaben. Dazu dient `satisfied`. Eine komplexe Aufgabe gelangt in diesen Zustand, wenn alle ihre Teilaufgaben vollständig bearbeitet wurden, wenn also auch alle Teilaufgaben `satisfied` sind.

Atomare Aufgaben sind `satisfied`, sobald die atomare Methode angewendet ist. Damit decken sich für atomare Aufgaben die Zustandsübergänge nach `satisfied` und nach `reduced`. Der `completed` Zustand muß für atomare Aufgaben nicht betrachtet werden, da dort keine Delegationen durchzuführen sind.

Zusammengefaßt führt die erfolgreiche und vollständige Ausführung einer Aufgabe, einschließlich aller eventuell vorhandenen Teilaufgaben, zum Zustand `satisfied`.

Der bisher beschriebene Ablauf und die erläuterten Zustände treten so nur bei Aufgabenbearbeitungen ohne Komplikationen auf. Es kann jedoch zu vielfältigen Ausnahmesituationen kommen, die ebenfalls im Zustandsmodell abgebildet sind.

*Abbruch d. Methodenwahl* Ausnahmesituationen ergeben sich einmal während der Ausführung einer Aufgabe durch den entsprechenden Agenten. Hat dieser eine komplexe Aufgabe akzeptiert, so ist diese `in progress`. Gelingt es dem Benutzer dann aber nicht, eine von mehreren Methoden auszuwählen (Cancel-Button im Method Selection Window), so gelangt die Aufgabe erneut in den Zustand `waiting`. Jener Benutzer oder ein anderer Agent, an den die Aufgabe ebenfalls delegiert ist, kann sie später nochmals akzeptieren und erneut ihre Bearbeitung versuchen.

*Abbruch des Delegieren* Hat der Benutzer zu einer komplexen Aufgabe eine Methode ausgewählt, so ist die Aufgabe `reduced`. Nun kann aber der Benutzer die Aufgabenbearbeitung während des Delegierens abbrechen, beispielsweise wenn er sich nicht in der Lage sieht, alle Teilaufgaben an Bearbeiter zu delegieren (Cancel Delegation Button im Complex Method Processing Window). Die Methodenanwendung schlägt fehl, alle bereits delegierten Teilaufgaben werden ungültig (siehe `invalid`). Falls es noch andere mögliche Methoden gibt, so setzt das System die Aufgabe in den Zustand `in progress`, der Benutzer kann eine der noch verbleibenden Methoden als Lösungsalternative wählen. Gibt es jedoch keine andere Methode mehr, so geht die Aufgabe in den Zustand `blocked`.

Ebenso kann eine atomare Aufgabe blockiert werden, wenn der Benutzer während der Anwendung der atomaren Methode (im Zustand `in progress`) diesen Vorgang abbricht (Cancel-Button im Atomic Method Processing Window). Grund hierfür kann beispielsweise sein, daß er sich mit den gegebenen Eingaben und der Aufgabenbeschreibung nicht zu einer Lösung im Stande sieht.

*blocked* Blockiert sind Aufgaben somit, wenn keine Lösungsmethode mehr anwendbar ist. Es gibt keine Möglichkeit mehr, die Aufgabe nach `completed` zu überführen. Damit kann aber auch das übergeordnete Problem - zu dessen Lösung der Scheduler gestartet wurde - nicht gelöst werden.

*Rückzüge* Der gesamte Problemlösungsprozeß gerät durch eine blockierte Aufgabe in einen inkonsistenten Zustand. Dies kann nur durch den Rückzug (Backtracking) einer Entscheidung aufgelöst werden. Eine zuvor getroffene Entscheidung muß zurückgenommen werden und stattdessen muß eine neue Entscheidung getroffen werden, die den Lösungsprozeß weiterbringen kann. Beispiele, wie solche Rückzüge ablaufen können, findet man in Anhang 3.2.

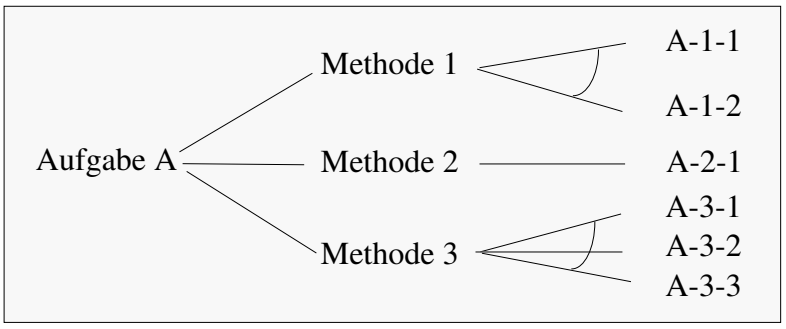



ABBILDUNG 20 Ausschnitt eines Aufgaben- und Methodenbaumes

Beispielsweise könnte in Abbildung 20 eine Blockierung von Aufgabe A-1-2 die Lösung der übergeordneten Aufgabe A verhindern. Doch könnte die zuvor getroffene Auswahl von Methode 1 (zu der ja A-1-2 gehört) zurückgezogen werden und stattdessen die Methode 3 gewählt werden. Hier könnte dann unter Umständen die Lösung der Aufgabe A ohne Blockierung möglich werden.

Backtracking-Prozesse müssen vom Benutzer explizit ausgelöst werden. Allerdings bietet der Scheduler hierzu erhebliche Unterstützung an. Er überwacht, wann Rückzüge nötig werden, macht den betroffenen Agenten darauf aufmerksam und bietet ihm Vorschläge an, wie der Rückzug durchgeführt werden kann. Insbesondere verwaltet er dann die durch den Rückzug mittelbar und unmittelbar ausgelösten Zustandsänderungen. Er bringt den Lösungsprozeß wieder in einen konsistenten Zustand.

Die Durchführung von Rückzügen kann nicht immer durch den Agenten erfolgen, der die Blockierung der Aufgabe ausgelöst hat. Im Beispiel von Abbildung 20 kann die Blockierung der atomaren Aufgabe A-1-2 durch einen beliebigen Bearbeiter, bezeichnet als Benutzer B, ausgelöst worden sein (Cancel im Bearbeitungsfenster). Wird nun ein Rückzug von Methode 1 als notwendig angesehen, muß dieser durch den Bearbeiter von Aufgabe A, Benutzer A, durchgeführt werden. Dessen Aufgabe ist ja, die Teilaufgaben zu delegieren und ihre Lösung zu überwachen. Nun muß er eventuell eine andere Methode auswählen oder er muß klären, warum die soeben gescheiterte Methode mißlang. Eventuell kann er dann auch diese Methode neu- oder umdefinieren (siehe Abschnitt 5.3). Der Benutzer A, der den Rückzug ausführt, ist somit nicht unbedingt mit dem auslösenden Benutzer B identisch.

*invalid*

Wenn Entscheidungen zurückgezogen werden, so hat dies erhebliche Auswirkungen auf die anderen Teile des Problemlösungsprozeß. Es können zuvor existierende Aufgaben aus einem beliebigen anderen Zustand nach `invalid` gelangen, sie werden ungültig. Dies bedeutet, daß die Problemlösung ohne diese Aufgabe möglich ist, ihre

Bearbeitung trägt beim aktuellen Stand des Lösungsprozeß nicht mehr zur Bearbeitung der Gesamtaufgabe bei. Bereits durchgeführte Aktionen zur Lösung der `invalid`-Aufgabe sind ungültig, bereits delegierte Teilaufgaben werden ebenfalls ungültig. Da deren Bearbeitung bereits begonnen haben kann, pflanzt sich der Zustandsübergang nach `invalid` eventuell über mehrere Ebenen des Aufgaben/ Methoden-Baumes fort.

In Fortführung des Beispiels zu Abbildung 20 wird durch den Rückzug von Methode 1 die blockierte Aufgabe A-1-2, aber auch die Aufgabe A-1-1, die sich noch in jedem anderen Zustand befinden kann, ungültig.

Werden Aufgaben ungültig, die gerade in Bearbeitung sind, so erhält der ausführende Agent eine Mitteilung. In der Regel wird er daraufhin die Bearbeitung der ungültigen Aufgabe abbrechen. Auf Wunsch kann er sie aber auch fortsetzen und fertigstellen, wenn auch das Resultat momentan nicht benötigt wird.

*lokale Entscheidungen*

In Abbildung 19 sieht man, daß zwei Arten von auslösenden Ereignissen für die Zustandsübergänge unterschieden werden. Es gibt globale und lokale Entscheidungen, die Übergänge auslösen. Zu den lokalen Entscheidungen ist nur Wissen über den betroffenen Task und dessen Bearbeitung notwendig. Mehr Wissen wird für globale Entscheidungen benötigt.

*globale Entscheidungen*

Im Zustandsdiagramm erkennt man, daß die Auslöser einiger Zustandsübergänge nach `invalid` und `blocked` als globale Entscheidungen gekennzeichnet sind (gestrichelte Linien). Global bedeutet, daß hierzu Wissen über den gesamten Problemlösungsprozeß nötig ist. Es muß der globale Stand der Problemlösung berücksichtigt werden. Beispielsweise muß, um eine Aufgabe zu blockieren, bekannt sein, daß keine andere Methode mehr zur Lösung zur Verfügung steht, alle Methoden müssen bereits gescheitert sein.

Solche globalen Entscheidungen - in [Maurer, Paulokat 94] auch als Meta-Entscheidungen bezeichnet - werden vom Benutzer getroffen und vom Scheduler verwaltet. Als zentrale Instanz überwacht dieser die Problemlösung und hat somit Metawissen über den gesamten Lösungsprozeß. Nähere Erläuterungen hierzu finden sich in [Dellen 94].

Nachdem nun erläutert worden ist, welche Informationen der Scheduler bezüglich des Problemfallwissens und der Aufgaben kennen muß und wozu diese benutzt werden, sind die Grundlagen gelegt, um im nächsten Abschnitt die Schnittstelle zwischen dem Scheduler und den Agenten zu behandeln. Denn über diese Schnittstelle werden unter anderem auch diese Informationen des Schedulers ausgetauscht.

## 4.6 Schnittstelle zwischen Scheduler und Clients

Über diese Schnittstelle kommunizieren die Agenten mit dem Scheduler. Sämtliche zu diesem Zweck spezifizierten Nachrichten sind in Anhang 3 abgedruckt, hier wird nur das Prinzip der Nachrichtenübermittlung erläutert. Die ausgetauschten Informationen lassen sich - nach ihrer Funktion und der Richtung des Datenverkehrs - in drei Gruppen gliedern.

Zum einen gibt es Nachrichten, mit denen die Clients beim Scheduler konkrete Informationen abfragen. Die zweite Gruppe enthält Mitteilungen der Clients an den Scheduler. Daneben gibt es noch Mitteilungen, die der Scheduler an die Clients verschickt, ohne daß diese explizit abgefragt werden müssen.

*Anfragen der Clients*

Zur ersten Gruppe gehören Aufrufe, die als Rückgabewert Daten liefern, die von den Fenstern des Benutzerprozesses benötigt werden. Entweder werden diese zum Initialisieren der Fenster benötigt, oder sie werden eingesetzt, um auf Anforderung des Benutzers ein Fenster zu aktualisieren. Übertragen werden Objekte, die in den Anzeigelisten des Arbeitsfensters und des Informationsfensters des Benutzers sowie in den Bearbeitungsfenstern für komplexe und atomare Aufgaben benötigt werden. Außerdem liefert der Scheduler auch die Informationen, die vom Benutzer zum Auflösen von Blockierungen gebraucht werden.

Der Scheduler kann die Informationen dadurch zur Verfügung stellen, daß er für alle in Abschnitt 4.5 vorgestellten Aufgabenzustände Listen führt. Diese Listen enthalten alle Aufgaben, die sich in einem bestimmten Zustand befinden. Auf Anforderung eines Agenten muß er eine solche Liste nach den für diesen Benutzer relevanten Aufgaben durchsuchen.

Weitere Informationen entnimmt der Scheduler aus dem Problemfallwissen. Hier ist noch auf die Besonderheit hinzuweisen, daß bei der Bearbeitung einer atomaren Aufgabe neben den Eingabeinstanzen auch die Ausgaben beim Scheduler angefordert werden. Dies beruht darauf, daß nur im Scheduler die Umsetzung der Parameter aus den Datenflüssen in konkrete Instanzen vorgenommen wird. Der Bearbeiter einer atomaren Aufgabe erhält vom Scheduler „leere Instanzen“ - also Konzeptspezifikationen -, in die er die erzeugten Werte durch einfaches Belegen der Slots einträgt.

*Mitteilungen der Clients*

Zur zweiten Gruppe von Nachrichten, den Mitteilungen der Clients an den Scheduler, gehören die Anmeldung und Abmeldung des Benutzers (Login und Logout). Daneben Benutzeraktionen, auf die der Scheduler reagieren muß. Hierunter fällt das Akzeptieren einer Aufgabe, das Lösen einer Aufgabe durch Anwendung einer Methode, das Delegieren von Teilaufgaben und das Scheitern

bestimmter Aktionen, beispielsweise der Abbruch einer Aufgabenlösung. Außerdem gehören hierzu auch Mitteilungen über Rückzüge und Aufhebungen von Rückzügen. Anhand der übergebenen Informationen kann der Scheduler seine Verwaltungskomponente aktualisieren. Sämtliche Nachrichten dieser Gruppe haben reinen Mitteilungscharakter, sie liefern keinen Rückgabewert.

*Mitteilungen des Scheduler*

Der letzte Nachrichtenkomplex enthält ebenfalls nur Mitteilungen. Sie dienen dazu, betroffene Benutzerprozesse über Zustandsänderungen zu informieren. Konkret handelt es sich um Mitteilungen, daß Aufgaben ausführbar werden, nicht mehr ausführbar sind, gelöst werden dürfen, ungültig oder blockiert sind. Diese Informationen entstehen durch automatisches Propagieren von Zustandsänderungen innerhalb der Aufgabenverwaltung des Schedulers. Sie werden nur an die Clients verschickt, auf deren Benutzerprozesse sie Auswirkungen haben.

*übertragene Datenobjekte*

Überwiegend werden als Inhalte der Nachrichten keine vollständigen Objekte (beispielsweise die vollständigen Aufgabenbeschreibungen) übertragen, sondern nur Kennungen. Mittels dieser Kennungen können die Objekte von den Clients bzw. dem Scheduler in der Wissensbasis referenziert werden. Dies geschieht über die Schnittstelle Scheduler / Wissensbasis bzw. Client / Wissensbasis. Diese Vorgehensweise ist dadurch gerechtfertigt, daß das konzeptuelle Modell als verteilte Wissensbasis so implementiert werden kann, daß es allen Benutzern für direkte Zugriffe zur Verfügung steht.

Wäre diese Realisierung als verteilte Datenbank nicht möglich, entfielen also direkte Zugriffe über die Schnittstelle Client / statische Wissensbasis, so müßte ein anderes Vorgehen gewählt werden. Statt der Kennungen müßte der Scheduler vollständige Objekte aus der Wissensbasis auslesen und über die Client / Server Schnittstelle an die Agenten verschicken. Aus verschiedenen, noch zu erläuternden Gründen wäre dies sehr unvorteilhaft. Allerdings läßt sich das System leicht auf dieses Vorgehen umstellen, indem in der Schnittstellenspezifikation für die Kennungen konkrete Objekte eingesetzt werden. Dies hat nur geringe Änderungen an den Benutzerprozessen zur Folge.

*Kennungen*

Wir verwenden jedoch statt der Objekte als Kennungen eindeutige Objekt-Identifier, die von der Hypertextmaschine, auf der das CoMoKit System aufsetzt, zur Verfügung gestellt werden[21]. An einigen Stellen läßt es sich allerdings nicht vermeiden, statt der Kennungen doch komplexe Objekte zu übertragen. Es handelt sich um die Nachrichten, die Problemfallwissen referenzieren. Konkret betrifft es die

(21) Die hamID, die sich aus dem Namen des Objekts und einer eindeutigen, fortlaufend vergebenen Nummer zusammensetzt.

Anforderung der für die Ausführung von atomaren Aufgaben benötigten Konzeptinstanzen, sowie die Rückgabe von Ergebnissen dieser Aufgaben an den Scheduler. Da das Problemfallwissen nur innerhalb des Schedulers vorliegt, können die Objekte auch nur auf dem direkten Weg zwischen Scheduler und Client übertragen werden.

*Entwurfsziele*

Beim Entwurf der Schnittstelle haben wir die geplanten Einsatzbedingungen des Systems berücksichtigt. Wie bereits in Abschnitt 4.1 erläutert, laufen die Benutzer- und Schedulerprozesse an verschiedenen Rechnern ab, in der Regel räumlich weit voneinander entfernt. Der Datenaustausch zwischen den Prozessen muß über eine noch festzulegende Methode erfolgen (hierzu Abschnitt 5.3), beispielsweise über ein Mail-System. Allen möglichen Methoden ist gemeinsam, daß die Interprozeßkommunikation zeit- und kostenmäßig einen höheren Aufwand verursacht als die Vorgänge innerhalb eines Prozesses. Daher war es wichtig, die Schnittstelle möglichst wenig zu belegen, um so die Belastung der Kommunikationskanäle möglichst gering zu halten. Dies haben wir beim Entwurf beachtet und in folgende konkrete Maßnahmen umgesetzt:

- Ziel 1: Zahl der ausgetauschten Nachrichten minimieren

  Dies wird dadurch erreicht, daß nur wirklich benötigte Informationen verschickt werden. Eine große Reduzierung der Nachrichtenaufrufe ergibt sich insbesondere dadurch, daß der Scheduler Zustandsänderungen selektiv nur an die betroffenen Benutzer propagiert, nicht aber an alle angemeldeten Agenten.

  Außerdem werden die Übersichtsanzeigen zur Information des Benutzers im TaskStateInfo-Fenster und im ComplexMethodProcessingWindow nur auf Anforderung des Benutzers aktualisiert (Update-Buttons). Müßte jede Änderung, die Auswirkungen auf diese Fenster hat, sofort propagiert werden, würden erheblich mehr Nachrichten verschickt - obwohl die Änderungen im Fenster vom Benutzer eventuell gar nicht wahrgenommen würden. Hier wird also bewußt in Kauf genommen, daß Anzeigen kurzzeitig nicht dem tatsächlichen Bearbeitungsstand entsprechen, bis der Benutzer manuell eine Aktualisierung anfordert.
- Ziel 2: Größe der ausgetauschten Nachrichten minimieren

  Wie bereits geschildert werden, soweit nicht das im Scheduler vorliegende Problemfallwissen betroffen ist, statt komplexer Objekte nur Kennungen der Objekte übertragen. Dadurch reduziert sich die Größe der Nachrichten erheblich.
- Ziel 3: Flexibilität gegenüber möglichen Erweiterungen

  Bei zusätzlichem Informationsbedarf oder Erweiterungen der Fensteranzeigen in den Schedulerprozessen kann die Schnittstelle entsprechend erweitert werden. Es stehen im Scheduler schon jetzt mehr Informationen zur Verfügung, als von den Clientprozessen ausgenutzt werden. Es wurden auch bereits Nachrichten definiert, die bisher nicht benutzt werden.

- Ziel 4: Toleranz gegenüber Verzögerungen bei der Nachrichtenübermittlung / Ausfällen einzelner Komponenten

  Der Nachrichtenaustausch wurde möglichst so angelegt, daß es auch dann nicht zu Blockierungen des gesamten Problemlösungsprozesses kommt, wenn eine Nachricht bei der Übertragung verzögert wird. Hierzu wurden die Nachrichtenaufrufe und die Routinen der Benutzerprozesse möglichst wenig verzahnt. Damit wird nicht der gesamte Benutzerprozeß blockiert, wenn er auf das Eintreffen einer Nachricht wartet. Nicht verhindert werden kann dies allerdings bei denjenigen Nachrichten, die als Funktionsaufrufe einen Rückgabewert liefern. Hier ist die aufrufende Routine blockiert, bis der Rückgabewert - eine Antwortnachricht - eintrifft.

  Bei Ausfall eines Benutzerprozeß können die anderen Benutzer ohne Verzögerung weiterarbeiten. Allerdings müßte durch eine Erweiterung sichergestellt werden, daß der Scheduler solche Totalausfälle eines Benutzerprozesses registrieren und entsprechende Umplanungen vornehmen kann. Eine Toleranz gegenüber dem Ausfall des Schedulerprozeß ist nicht erreichbar, in diesem Fall muß die gesamte Problemlösung neu beginnen.

In den letzten Abschnitten ist die Verwaltungskomponente des Schedulers für Aufgaben und Problemfallwissen erläutert worden. Ebenso wurden die Funktionen der Clientprozesse und die Schnittstelle zwischen dem Scheduler und den Clients vorgestellt. Im nächsten Abschnitt soll nun noch ein Überblick über die Aufgaben der Abhängigkeitsverwaltung gegeben werden. Die Funktionen dieser Komponente repräsentieren den größten Fortschritt des Interpreters in CoMo-Kit 3.0 gegenüber dem Interpreter in der Version 2.0.

## 4.7 Abhängigkeitsverwaltung des CoMo-Kit Interpreters

Die Komponente zur Abhängigkeitsverwaltung ist der wichtigste Teil des Problemlösers, da sie die wesentlichen Aufgaben der Problemlösungskomponente realisiert und koordiniert. Zu den Aufgaben des Problemlösers gehört es nämlich

- die durch den Datenfluß gegebenen zeitlichen Abhängigkeiten zwischen den Aufgaben zu modellieren,
- den aktuellen Zustand des Problemlöseprozesses zu verwalten
- und die Konsistenz der Lösung zu überwachen.

Welche Anforderungen die Zustandsverwaltung der Aufgaben (siehe Abschnitt 4.5) und des Problemfallwissens (siehe Abschnitt 4.4) stellt, ist bereits erläutert worden. Nun geht es darum, was es heißt, einen konsistenten Lösungsverlauf sicherzustellen.

*Entscheidungspunkte*

Die Problemlösung ist in ihrem Verlauf durch eine Reihe von Entscheidungspunkten gekennzeichnet. Es sind dies

- die Zerlegung von Aufgaben in Teilaufgaben durch Auswahl einer komplexen Methode,
- das Delegieren der Teilaufgaben
- und das Belegen der Ausgabevariablen atomarer Aufgaben.

Sieht man den Baum aus Aufgaben und Methoden als Suchraum des Problemlösers an, so definieren die Entscheidungspunkte einen Hyperpfad (siehe Abschnitt 3.3) durch diesen Baum. Sie beschreiben den Lösungsweg. Die einzelnen Entscheidungen müssen jedoch nicht immer richtig sein, zumindest nicht immer optimal. Sie beruhen eventuell auf unvollständigem Wissen. Der durch die Entscheidungen definierte Pfad führt somit nicht immer zu einer (optimalen) Lösung. Die Problemlösungskomponente muß es daher ermöglichen, verschiedene Lösungswege zu testen. Sie muß es erlauben, solange neue Lösungswege zu begehen, bis eine ausreichend gute Lösung gefunden ist oder feststeht, daß eine Lösung nicht möglich ist.

*Rückzüge*

Der Problemlöser muß dazu gestatten, an jedem Entscheidungspunkt die getroffene Entscheidung zurückzunehmen und eine andere Entscheidung zu treffen (Backtracking). Ein einfaches *chronologisches Backtracking* würde bedeuten, daß immer die zuletzt getroffene Entscheidung zurückgezogen wird und die nächste mögliche Alternative ausprobiert wird. So könnte zwar der ganze Lösungsraum abgesucht werden. Dieses „ziellose“ Herumsuchen im Aufgaben/Methoden-Baum wäre aber sehr ineffizient und würde außerdem von den Benutzern viele unnötige Arbeitsgänge verlangen.

Daher werden die Rückzüge getroffener Entscheidungen und Umplanungen besser in den Händen der ausführenden Benutzer gelassen. Diese können Entscheidungen zurückziehen unter Berücksichtigung anderer Entscheidungspunkte, mit dem Wissen über Zusammenhänge, aber insbesondere auch mit Hintergrundwissen über ihre Aufgabe und die Domäne. Beispielsweise hat der Bearbeiter einer komplexen Aufgabe Hintergrundwissen, warum er genau eine bestimmte Methode ausgesucht hat und welche Alternative er im Falle eines notwendigen Rückzugs der Entscheidung ausprobieren würde. Er kann daher eine viel sinnvollere Umplanung vornehmen als dies der Scheduler bei chronologischem Backtracking tun würde. Man bezeichnet dieses Vorgehen als *dependency-directed backtracking*[22].

Allgemein erfordert ein Backtracking-Prozeß nicht nur die Rücknahme einer alten und das Bestimmen einer neuen Entscheidung.

(22) abhängigkeitsgesteuerte Rückverfolgung

Rückzüge können Auswirkungen auf den gesamten Problemlösungsprozeß haben. Wird zum Beispiel eine Entscheidung zurückgenommen, die Ausgabevariablen einer atomaren Aufgabe belegt hat, so werden auch alle Aufgaben ungültig, die diese Ausgabewerte bereits benutzt haben. Wird eine Delegierungsentscheidung hinfällig, so müssen auch die delegierten und eventuell bereits bearbeiteten Teilaufgaben zurückgenommen werden. Werden diese Auswirkungen von Rückzügen nicht beachtet, kommt der Bearbeitungsprozeß in einen inkonsistenten Zustand. Daher ist es eine der wichtigsten Aufgaben des Schedulers, die Konsistenz des Lösungsprozesses zu überwachen und bei Verletzungen Maßnahmen zu ergreifen, um die Konsistenz wiederherzustellen.

*REDUX*

In CoMo-Kit 3.0 wurde die Abhängigkeitsverwaltung durch Barbara Dellen (siehe [Dellen 94]) implementiert. Sie setzt auf der REDUX-Architektur (siehe [Petrie 91]) auf. REDUX stellt Möglichkeiten zur Verfügung, einen Suchraum darzustellen und ihn zur Problemlösung zu benutzen. Es unterstützt die Zerlegung von Aufgaben innerhalb des Suchraumes und kann Zustände von Aufgaben und Entscheidungen abbilden. Ferner gibt es Mechanismen, die für das Erkennen von blockierten Aufgaben und den Rückzug von Entscheidungen benutzt werden können. Die Auswirkungen von Rückzügen werden durch das System weiterpropagiert. REDUX mußte erweitert werden um Aspekte, mit denen sich die Delegierung von Aufgaben, die Datenflußabhängigkeiten und die Zustandsübergänge im Verlauf des Lösungsprozeß abbilden ließen. Danach konnte REDUX in den Scheduler eingebunden und mit den übrigen Komponenten des Interpreters verbunden werden.

REDUX ist implementiert unter Benutzung eines TMS-Systems (Truth Maintenance System), mit dem die verschiedensten Abhängigkeiten zwischen Entscheidungen verwaltet und dependency-directed Backtracking unterstützt wird. Neben Entscheidungen wie Auswahl einer Methode werden in dieser Struktur auch Zuordnungen von Werten an die Variablen (Ein- und Ausgaben der atomaren Aufgaben) abgebildet. Damit lassen sich die Zustände, in denen sich Aufgaben befinden, bestimmen. Daneben gestatten es die vorhandenen Mechanismen, solche Zuordnungsentscheidungen zurückzuziehen. Trotzdem gehen die angelegten Variablenbindungen nicht verloren, sie sind nur ungültig. Sollten aber durch Rückzüge und Wiedereinsetzen von Entscheidungen die Zuordnungen nochmals benötigt werden, so können die Wertezuordnungen leicht wieder gültig gemacht werden. Die Ergebnisse sind also noch vorhanden, ohne die entsprechende Aufgabe nochmals berechnen zu müssen.

Die Abbildung der Variablenzuordnungen in das TMS unterstützt natürlich auch die Überwachung des Datenflusses zwischen den Aufgaben. Weitere Einzelheiten zu den aufgebauten Strukturen und der Funktionsweise finden sich in [Dellen 94].

KAPITEL 5

# Zusammenfassung und Ausblick

*Dieses Kapitel faßt die Inhalte der Diplomarbeit kurz zusammen und erläutert mögliche Erweiterungen des implementierten Systems.*

## 5.1 Zusammenfassung

Das erste Teilziel dieser Diplomarbeit waren Ergänzungen des konzeptuellen Modells von CoMo-Kit. Um eine sinnvolle Systembenutzung zu gestatten, war der Objekttyp „formaler Parameter“ unbedingt notwendig. Erst jetzt kann die Datenübergabe zwischen verschiedenen Teilaufgaben angemessen modelliert werden, wenn verschiedene Instanzen eines Objekttyps im Problemlösungsprozeß benötigt werden. Die Einführung des Objekttyps „Methode“ hingegen vervollständigt nicht nur das bisherige Modellierungsvorgehen. Durch Methoden werden die Möglichkeiten, die CoMo-Kit zur Analyse komplexer Arbeitsabläufe bietet, grundsätzlich erweitert. Auch diese Ergänzung ist insofern notwendig, als sich viele Domänen sicher nur mit Hilfe multipler Aufgabenzerlegungen vernünftig in ein Modell abbilden lassen. Es gibt allerdings auch Domänen, bei denen Aufgaben immer eindeutig zu beschreiben sind, multiple Aufgabenzerlegungen also nicht notwendig sind. Hier stellt aber die neue Modellierungsform mit Methoden kein Hindernis dar, es fällt durch die Methodendefinition lediglich ein geringer Mehraufwand in der Wissensakquisitionsphase an.

Beide Ergänzungen des konzeptuellen Modells waren wichtig, um die Benutzerakzeptanz und damit die Einsatzmöglichkeiten von CoMo-Kit zu steigern. Denn nur, wenn der Benutzer die von ihm modellierte Domäne zufriedenstellend in das konzeptuelle Modell abbilden kann und dabei ausreichend vom System unterstützt wird, kann er die weiteren Phasen der Systementwicklung in CoMo-Kit durchführen. Nur wenn ihm die Modellierung vollständig gelingt, kann er die Operationalisierung und interaktive Ausführung des kon-

zeptuellen Modells in Angriff nehmen. In diesem Zusammenhang ist noch zu erwähnen, daß in der bisherigen Implementierung die Restriktion an die Domäne gestellt wird, daß alle Methoden, die zu einer Aufgabe gehören, die gleichen Ein- und Ausgaben benutzen müssen (siehe Seite 27). Durch diese Einschränkung wird in vielen Domänen die Erstellung eines konzeptuellen Modells unmöglich gemacht oder erschwert. Hier müßte noch eine Lösung gesucht werden, die dem Benutzer auch in diesen Fällen den Einsatz des Systems erlaubt.

Die Einführung von Methoden hatte jedoch nicht nur Einfluß auf den Ablauf der Wissensakquisition mit CoMo-Kit, sondern bedingte auch eine komplette Neugestaltung der Ausführungskomponente. Der Interpreter ermöglicht es nun, Methoden zur Ausführungszeit auszuwählen und interaktiv mit dem System zu bearbeiten. Dazu wurden alle Benutzerschnittstellen und die Benutzerprozesse neu entworfen.

Die Neuimplementierung des Interpreters kann noch keinen letztendlichen Stand erreichen, da sie die verteilte Aufgabenbearbeitung an mehreren Rechnern lediglich simuliert. Hier sind noch Ergänzungen notwendig, um eine reale verteilte Ausführung zu gestatten (siehe Abschnitt 5.2). Jedoch wird die Vorgehensweise des Interpreters bereits deutlich. Alle notwendigen Mechanismen außer der Interprozeßkommunikation über eine verteilte Schnittstelle und der Bearbeitung formalisierter Aufgaben durch Rechner sind bereits vorhanden.

Wichtigstes Resultat der Diplomarbeiten von Barbara Dellen und mir ist die Ergänzung der Ausführungskomponente um einen Teil, der Benutzerentscheidungen während des Lösungsprozesses nicht nur - wie bisher - verwaltet, sondern mittels abhängigkeitsgesteuertem Backtracking Rückzüge getroffener Entscheidungen und Umplanungen erlaubt und die Konsistenz des Lösungsprozesses überwacht.

Die hierzu von Barbara Dellen entworfene und implementierte Komponente zur Abhängigkeitsverwaltung stellt - integriert in die übrige Interpreterarchitektur - ein voll funktionsfähiges und einsetzbares System dar. Es sind auch hier noch Erweiterungen denkbar (siehe Abschnitt 5.3), jedoch sind die geforderten Kernfunktionen vollständig realisiert.

Mit dem neuen Interpreter verlagert sich der Schwerpunkt bei den Einsatzmöglichkeiten von CoMo-Kit meines Erachtens mehr in Richtung Simulation einer realen Problemlösung. Der Interpreter unterstützt nun die kooperative, verteilte Problemlösung erheblich besser. Durch die Rückzugsmechanismen werden Möglichkeiten geboten, die über eine reine Validierung des spezifizierten konzeptuellen Modells hinausgehen.

Es wird nun berücksichtigt, daß bei der realen Problemlösung viele Entscheidungen auf unvollständigem Wissen beruhen, und somit nicht optimal oder falsch sind. Diese Entscheidungen können zurückgenommen oder geändert werden. Zurückgenommene Entscheidungen können auch wiedereingesetzt werden. Hiermit steht ein größerer Funktionsumfang zur Verfügung, den der Benutzer auch vom zu entwickelnden Informationssystem erwartet. CoMo-Kit ist damit nicht mehr nur in erster Linie ein Wissensakquisitionstool, sondern vorallem auch eine vielseitig einsetzbare Umgebung für die Ausführung der Spezifikationen im konzeptuellen Modell mit Mehrbenutzer- und Backtrackingunterstützung.

Die in der Einleitung beschriebenen Einsatzmöglichkeiten von CoMo-Kit zur Erfassung von kooperativen Arbeitsabläufen kann man also ergänzen um den Einsatz als Simulationsumgebung, mit der einem Team in einer frühen Phase der Systementwicklung die Kernfunktionen des zu erstellenden Systems demonstriert werden können. Der spätere Einsatz kann mit Hilfe von CoMo-Kit durch die zukünftigen Systembenutzern erprobt und simuliert werden, CoMo-Kit koordiniert und steuert die Abarbeitung der Spezifikationen im konzeptuellen Modell. Durch die Rückzugs- und Umplanungsmöglichkeiten nähert sich das Vorgehen sehr nah an reale Arbeitsabläufe an, die auch von häufigen Unterbrechungen, Fehlversuchen und Umplanungen geprägt sind.

Die folgenden Abschnitte beschäftigen sich mit Ergänzungen, die das implementierte System vervollständigen und seinen Funktionsumfang erweitern können.

## 5.2 Strategien zur Verteilung der Client-Server Schnittstelle

Bereits in Abschnitt 4.3 wurde erwähnt, daß die Verteilung der Benutzerprozesse und damit auch die Client / Server-Schnittstelle bisher nur simuliert werden. Tatsächlich handelt es sich bei den „Prozessen“ bisher nur um verschiedene Smalltalk-Objekte, die gegenseitig über den Austausch von Nachrichten miteinander kommunizieren. Nachrichtenaustausch entspricht in Smalltalk dem Aufruf von Methoden. Die in Anhang 3 aufgeführten Nachrichten sind somit Methoden, die vom Benutzer- bzw. Schedulerprozeß implementiert und vom Kommunikationspartner aufgerufen werden. Will beispielsweise der Schedulerprozeß einen Benutzerprozeß über eine ungültig gewordene Aufgabe informieren, so ruft er die Methode „invalid:“ der entsprechenden Instanz des Benutzerprozesses[23] auf.

Als einer der nächsten Schritte bei der Weiterentwicklung des CoMo-Kit Systems ist jedoch geplant, diese Prozesse tatsächlich zu trennen

und an verschiedenen Rechnern ablaufen zu lassen. An jedem Rechner muß dann ein Smalltalk-Image existieren. Dieses kann jedoch nicht einfach Methoden von Objekten innerhalb eines anderen Images aufrufen. Es muß eine Möglichkeit gefunden werden, die Kommunikation abzubilden.

Hierzu wurden von uns drei Ansätze diskutiert und deren möglicher Einsatz bereits beim Entwurf des Interpreters berücksichtigt.

Erste Realisierungsmöglichkeit wäre, die Nachrichten über eine verteilte Datenbank auszutauschen. Das konzeptuelle Modell soll, wie bereits erwähnt, in eine verteilte Datenbank abgebildet werden. In einer früheren Version von CoMo-Kit wurde dies bereits mit dem Datenbanksystem Gemstone getestet. Die über die Schnittstelle auszutauschenden Nachrichten könnten dann mit Hilfe dieser Datenbank weitergegeben werden. Auch die Objekte des Problemfallwissens könnten in der verteilten Datenbank abgelegt werden. Damit ständen sie allen Benutzern zur Verfügung.

Eine zweite Realisierungsmöglichkeit wäre es, die Socketschnittstelle von Smalltalk[24] auszunutzen. Dieses Konzept gestattet es, durch Smalltalk-Aufrufe Daten mit Hilfe des UNIX-Betriebssystem auszutauschen. Die Daten werden in Streams (Dateien) geschrieben, die an die Sockets gebunden sind. Verwendet man hierzu Schreib-/Lese-Streams, so können die Daten auch von anderen Smalltalk-Prozessen, die ebenfalls über eine Socketanbindung an den entsprechenden Stream verfügen, wieder eingelesen werden. Bei entsprechender Realisierung bliebe das Socketkonzept für den Benutzer unsichtbar, das System könnte aber Daten mit anderen Benutzern an den vernetzten Rechnern austauschen.

Eine noch weitergehende Möglichkeit wäre, statt einfacher UNIX-Dateien zum Datenaustausch ein Mail-System zu nutzen. Die Nachrichten könnten als Electronic-Mails zwischen Benutzern und Scheduler hin und her geschickt werden. Ein Problem bei diesem Vorgehen liegt in einer realen Domäne sicherlich in den dabei erreichbaren Datenaustauschgeschwindigkeiten. Problematisch ist ferner, daß bei Mail-Kommunikation neben der Anbindung von Smalltalk an das Betriebssystem auch noch die Umsetzung in Mails und die Weitergabe der auszutauschenden Objekte an das Mail-System realisiert werden muß.

Bei allen geschilderten Methoden ergibt sich ein Hauptproblem. Leicht zu transportieren sind sicher solche Nachrichten, die nur aus

(23) Benutzerprozesse sind Instanzen der Klasse CoMoKitClient, Schedulerprozesse sind Instanzen der Klasse CoMoKitSystem.

(24) unter Zugrundelegung des TCP/IP-Protokolls

einem Identifikator für den Nachrichtentyp und Kennungen für die übertragenen Objekte bestehen[25]. Auch solche Nachrichten, die einen Rückgabewert eines einfachen Datentyps (z.B. nur eine Bestätigung oder Objektkennung) liefern, lassen sich leicht abbilden[26]. Hier muß nur die Rückgabe des Wertes als separate Nachricht aufgefaßt werden, die dann als Antwort zurückgeschickt wird. Daher wurden beim Schnittstellenentwurf möglichst viele Nachrichten benutzt, die diese Kriterien erfüllen.

Schwieriger wird es jedoch, wenn mit den Nachrichten komplexere Objekte verschickt werden müssen. Wie bereits in Abschnitt 4.6 geschildert, ließ sich dies nicht immer vermeiden (aufgrund des nur im Scheduler vorliegenden Problemfallwissens). Um solche komplexen Objekte - beispielsweise Konzeptinstanzen - austauschen zu können, müssen diese erst in ein Format gebracht werden, in dem der Datenaustausch möglich wird. Als Smalltalk-Objekte lassen sie sich nicht transferieren. Hier müßte also noch eine entsprechende Syntax entwickelt werden, außerdem müßte berücksichtigt werden, das durch die Größe der resultierenden Objektbeschreibungen die Schnittstelle doch erheblich belastet würde.

Unter Berücksichtigung dieses Aspektes erscheint es mir insgesamt am sinnvollsten, den ersten Lösungsansatz zu benutzen und sowohl die Nachrichten als auch die Objekte des Problemfallwissens auf eine verteilte Datenbank abzubilden.

Begonnen wird zur Zeit mit der Implementierung der zweiten Möglichkeit, der Kommunikation über die Socketschnittstelle. Ein genaues Realisierungskonzept ist aber noch nicht erstellt

## 5.3 Definition / Änderung von Methoden zur Laufzeit

Nach dem auf Seite 33 geschilderten Ablauf kann der Benutzer bei komplexen Aufgaben eine der vorhandenen Methoden auswählen und diese ausführen. Dabei ist er dann an die im konzeptuellen Modell festgehaltene Spezifikation der Methodenstruktur gebunden. Dieses ist nur dann sinnvoll, wenn der CoMo-Kit Interpreter zur reinen Validierung der vorhandenen Spezifikation benutzt wird. Soll jedoch das zu entwickelnde Informationssystem simuliert werden oder CoMo-Kit zur realen Problemlösung eingesetzt werden, so ist diese statische Vorgehensweise nicht optimal. Benutzerfreundlicher

(25) beispielsweise alle Nachrichten in Anhang 3.3 und alle Nachrichten in Anhang 3.2 außer „login:“, „logout:“ und „reducedTask: method: output:“

(26) alle Nachrichten in Anhang 3.1 mit Ausnahme der zur Anforderung der Ein- und Ausgabevariablen dienenden

ist es, wenn die spezifizierten Strukturen noch zur Laufzeit des Interpreters abgeändert werden können.

Daher besteht eine interessante Systemerweiterung darin, dem Benutzer noch zur Ausführungszeit zu gestatten, neue Methoden zu definieren und bestehende Methoden abzuändern. Letzteres kann zum Beispiel dadurch geschehen, daß zu einer Methode neue Teilaufgaben definiert oder die Datenflüsse abgeändert werden.

Hierzu wurde das Methodenauswahl-Fenster (siehe Abbildung 14) erweitert. Der Benutzer kann nun zu einer ausführbaren Methode einen Editor aufrufen, in dem er die Methodenstruktur ändern kann. Oder er kann einen Editor einblenden, in dem er neue Methoden definieren kann, die er dann natürlich noch genauer spezifizieren muß. Nach Abschluß der Änderungen teilt der Benutzer dies dem System mit, die geänderten Definitionen werden daraufhin vom Scheduler übernommen und im Methodenauswahl-Fenster angezeigt. Das so erweiterte Methodenauswahl-Fenster ist in Anhang 2.3.1 beschrieben. Zur Ergänzung bzw. Änderung der Spezifikation werden die Editoren aus Anhang 2.1 verwendet.

Problematisch an dieser Systemerweiterung ist, daß die Änderungen im konzeptuellen Modell durchgeführt werden, welches normalerweise vor dem Start der Ausführungskomponente vollständig vorliegt und nicht mehr verändert wird. Während nun ein Benutzer diese Spezifikationen ändert, dürfen keine anderen Benutzer auf die noch nicht vollständig geänderten Teile zugreifen. Bisher gibt es im konzeptuellen Modell allerdings keinen Mechanismus zur Definition von Zugriffsrechten. Es ist nicht möglich, bestimmte Objekte für Benutzerzugriffe zu sperren. Im Falle der Änderung oder Ergänzung von Methoden ist aber dieses Problem nicht relevant, da gleichzeitig immer nur ein Agent eine bestimmte Aufgabe bearbeiten kann. Da das Ändern der Definition aber ein Bearbeitungsschritt ist, kann es nicht dazu kommen, das verschiedene Benutzer parallel eine Methode ändern und anwenden wollen.

Soll allerdings das CoMo-Kit System so weiterentwickelt werden, daß zur Ausführungszeit noch weitergehendere Änderungen am konzeptuellen Modell möglich werden, so müßte über einen Mechanismus nachgedacht werden, mit dem das konzeptuelle Modell vor unberechtigten Änderungs- und Lesevorgängen geschützt würde. Im Rahmen einer Implementierung mittels einer verteilten Datenbank würden diese Schutzmechanismen eventuell vom Datenbanksystem geliefert.

ANHANG 1

# Literaturverzeichnis

[Angele et al. 90]
J. Angele, D. Fensel, R. Studer: Applying Software Engineering Methods and Techniques to Knowledge Engineering. Forschungsbericht der Universität Karlsruhe, Nr. 204, Karlsruhe, September 1990

[Bernardi et al. 91]
A. Bernardi, H. Boley, Ph. Hanschke, K. Hinkelmann, Ch. Klauck, O. Kühn, R. Legleitner, M. Meyer, M. M. Richter, F. Schmalhofer, G. Schmidt, W. Sommer: ARC-TEC: Acquisition, Representation and Compilation of Technical Knowledge. Research Report RR-91-27, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Kaiserslautern, September 1991

[Beste 94]
E. Beste: Automatisches Generieren von Eingabemasken unter CoMo-Kit, Projektarbeit im Fachbereich Informatik der Universität Kaiserslautern, Kaiserslautern, Juli 1994

[Beste 94b]
E. Beste: Evaluierung und Erweiterung von CoMo-Kit am Beispiel des Bebauungsplanes, Diplomarbeit im Fachbereich Informatik der Universität Kaiserslautern, Fertigstellung voraussichtlich Herbst 94

[Dellen 94]
B. Dellen: Operationalisierung von konzeptuellen Modellen, Diplomarbeit im Fachbereich Informatik der Universität Kaiserslautern, Fertigstellung voraussichtlich Herbst 94

[Haupenthal 93]
W. Haupenthal: Entwicklung einer objektorientierten Sprache zur Operationalisierung konzeptueller Modelle, Diplomarbeit im Fachbereich Informatik der Universität Kaiserslautern, Kaiserslautern, 1993

[Maurer 93]
F. Maurer: Hypermediabasiertes Knowledge Engineering für verteilte wissensbasierte Systeme, Dissertation Universität Kaiserslautern, Kaiserslautern, 1993

[Maurer, Paulokat 94]
F. Maurer, J. Paulokat: Operationalizing Conceptual Models Based on a Model of Dependencies. In: A. Cohn (ed.): ECAI 94, 11th European Conference on AI, John Wiley & Sons, Ltd., 1994

[Nilsson 80]
N. Nilsson: Principles of Artificial Intelligence, Springer Verlag, Palo Alto, 1980

[Petrie 91]
Ch. Petrie: Planning and Replanning with Reason Maintenance, Dissertation, University of Texas, Austin, 1991

[Richter 92]
M. Richter: Prinzipien der künstlichen Intelligenz, Teubner Verlag, Stuttgart, 1992

[Richter et al. 91]
M. Richter, A.Bernardi, C. Klauck, R. Legleitner: Akquisition und Repräsentation von technischem Wissen für Planungsaufgaben im Bereich der Fertigungstechnik. Research Report RR-91-23, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Kaiserslautern, Juli 1991

[Schmalhofer et al. 91]
F. Schmalhofer, R. Bergmann, O. Kühn, G. Schmidt: Using Integrated Knowledge Acquisition to Prepare Sophisticated Expert Plans for their Re-Use in Novel Situations. In: Th. Christaller (ed.): GWAI-91, Springer Verlag, Berlin 1991

[Wetter 90]
T. Wetter: First Order Logic Foundation of the KADS Conceptual Model. In: Wielinga B. et al: Current Trends in Knowledge Acquisition, IOS Press, Amsterdam, Mai 1990

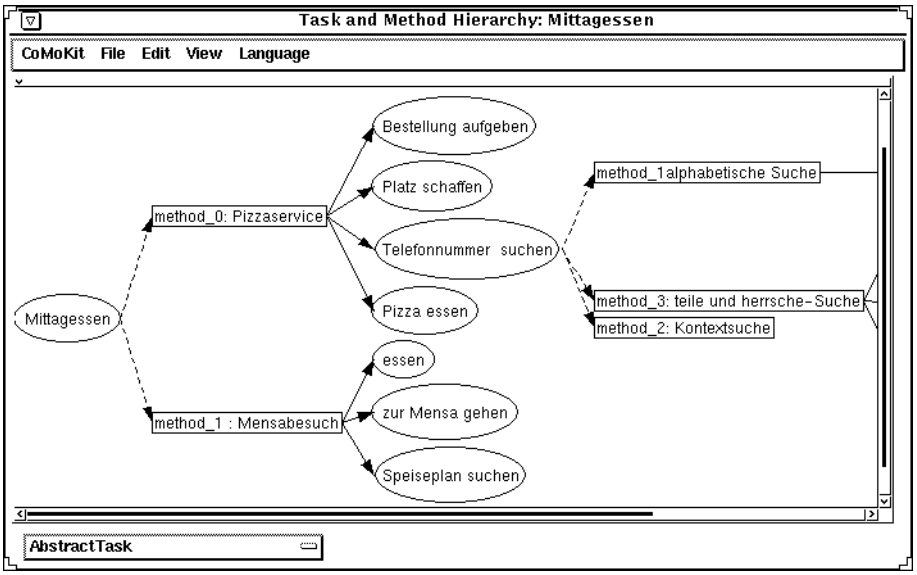
[Wielinga et al. 92]
B. Wielinga, G. Schreiber, J. Breuker: KADS a modelling approach to knowledge engineering. In: G. Schreiber (Ed.): Special Issue: The KADS approach to knowledge engineering, Knowledge Acquisition, Vol. 4 No. 1, Academic Press, März 1992

# ANHANG 2 Dokumentation der implementierten Benutzerschnittstellen

*In diesem Kapitel werden die geänderten und neu implementierten Benutzerschnittstellen als Bildschirmabdrucke vorgestellt. Der Aufruf der Fenster, die Funktionen der Anzeigen, Buttons und Mausmenüs werden aufgelistet. Es handelt sich um reine Funktionsbeschreibungen, Details zur Implementierung entnimmt man dem Klassenkommentar sowie den Kommentaren zu den Methoden im Programm.*

## Anhang 2.1 Editoren zur Spezifikation des Aufgaben- und Methodenbaumes

### Anhang 2.1.1 TaskAndMethodFilter




*Smalltalk-Klasse:* `TaskAndMethodFilterDisplay`

*Aufruf:* `(TaskAndMethodFilter on: network) open`

wobei als `network` das anzuzeigende CoMo-Kit Netz übergeben wird. Die Instanz des TaskAndMethodFilter ermittelt die anzuzeigen-

den Objekte und übergibt diese an das TaskAndMethodFilterDisplay, das für die graphische Anzeige verantwortlich ist.

*Beschreibung:* Fenster, in dem der Entwickler eine Übersicht über alle im konzeptuellen Modell bisher definierten Aufgaben und Methoden erhält. Wird aufgerufen, indem der CoMoKit-Superusers in seinem Fenster zum Editieren des Netzwerks[27] „Tasks“ anwählt.

Im Fenster können initiale Aufgaben direkt eingegeben werden. Weitere Aufgaben sowie Methoden müssen in den entsprechenden Editoren (siehe Anhang 2.1.2, Anhang 2.1.3) erzeugt werden, werden dann aber in dieser Gesamtübersicht mit angezeigt.

*Anzeigen:* `Aufgaben- und Methodenhierarchie:` In Form eines Baumes mit Wurzel am linken Bildrand, Blättern am rechten Bildrand, werden alle Aufgaben und komplexe Methoden angezeigt. Ellipsen entsprechen Aufgaben, Rechtecke stellen Methoden dar. Außerdem erscheinen die sie verbindenden Kanten, wobei die Kanten von den Aufgaben zu den Methoden[28] gestrichelt, die Kanten von Methoden zu Aufgaben[29] durchgezogene Linien sind.

`Typ-Feld:` Selektionsfeld am unteren Fensterrand, in dem ein Aufgabentyp angewählt werden kann. Der aktuell im Feld stehende Wert wird als Typ bei der Erzeugung neuer Aufgaben benutzt.

*Pop-Down Menüs:* CoMoKit:

- About: Copyright-Information
- Help: Hilfefenster

File:

- Print all: PostScript-Ausdruck aller angezeigten Objekte (Snapshot)
- Print Selection: Ausdruck der selektierten Objekte mit Slotinhalt
- Quit: Schließen des Fensters

Edit:

- Add Task: Hinzufügen einer Aufgabe als initiale Aufgabe (Wurzel eines Aufgaben- und Methodenbaumes). Die Aufgabe wird Instanz des Aufgabentyps, der im Typ-Feld angezeigt ist.
- Show Only Tasks: Anzeigen einer Liste, in der nicht die Methoden, sondern nur die Aufgaben in alphabetischer Reihenfolge aufgeführt sind.

(27) `CoMoKitLauncher`, erzeugt durch `CoMoKitNetwork browseInitial`

(28) TaskToComplexMethodLinks, die Oder-Kanten nach Abschnitt 3.3

(29) HAMContextLinks, die Und-Kanten nach Abschnitt 3.3

- Edit Object: Editieren des selektierten Objektes, ruft entweder den Methoden-Editor (siehe Anhang 2.1.3) oder den Aufgaben-Editor (siehe Anhang 2.1.2) auf.
- Change Task Type: Ordnet selektierter Aufgabe einen anderen Aufgabentyp zu, wobei der neue Aufgabentyp durch die Anzeige im Typ-Feld bestimmt ist
- Delete: Löschen der selektieren Objekte und Kanten
- Choose Agent: mögliche Agenten zu einer Aufgabe festlegen
- Related Show: alle mit dem selektierten Objekt verbundenen Netzwerk-Objekte anzeigen
- Inspect: Aufrufen eines Smalltalk-Inspektors zu dem selektierten Objekt
- Select All: Selektieren aller angezeigten Objekte
- Deselect All: Deselektieren aller Objekte

View:

- Display Graphics: Anzeige der Objekte als Graphik (Baumstruktur)
- Display List: Anzeige der Objekte als Liste
- Refresh View: Neuaufbau der graphischen Anzeige
- Recalculate View: Neuberechnung der angezeigten Objekte mit Neuaufbau der Anzeige
- Arrangement Tools: Zum Positionieren und gleichmäßigen Verteilen selektierter Objekte in Anzeigefenstern mit freier Objektpositionierung
- Layout: Graphische Anzeige in horizontaler oder vertikaler Form

Language:

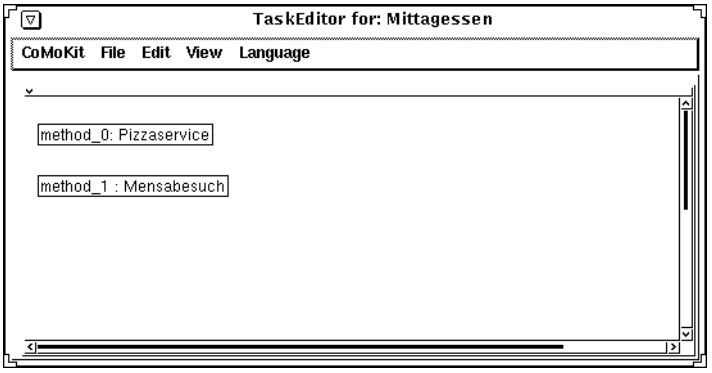
Auswahl einer Sprache, in der die Namen der Objekte angezeigt werden sollen

*Mausmenüs:* mittleres Mausmenü: alle Funktionen, die auch über das Pop-Down Menü Edit ausgelöst werden können.

*Schließen des Fensters* Erfolgt entweder durch Anklicken von Quit im File-Menü oder über Close im Window-Menü[30].

(30) Menü, daß durch Drücken der rechten Maustaste erreicht werden kann

## Anhang 2.1.2 TaskEditor




*Smalltalk-Klasse:* `CoMoKitTaskEditor`

*Aufruf:* `(CoMoKitTaskEditor for: task) open`

wobei als `task` die Aufgabe übergeben wird, die angezeigt und bearbeitet werden sollen.

*Beschreibung:* Fenster, in dem der Entwickler alle zu einer Aufgabe relevanten Aspekte ansehen und editieren kann. Insbesondere ermöglicht der Editor, Methoden zu erzeugen und der bearbeiteten Aufgabe zuzuordnen. Wird aufgerufen, wenn im TaskAndMethodFilter (siehe Anhang 2.1.1) oder im MethodEditor (siehe Anhang 2.1.3) eine Aufgabe selektiert und der Edit-Befehl angewählt wird..

*Pop-Down Menüs:* CoMoKit + File + View + Language:

siehe Anhang 2.1.1

Edit:

Modusumschaltung zwischen:

- Contents: Einblenden eines Editors für Ziel und Beschreibung der Aufgabe[31] (TaskDescriptionEditor)
- Denotation: Einblenden eines Editors für den Namen der Aufgabe und zugeordnetes Icon (HAMObjectDenotationEditor)
- ST-Specification: Einblenden eines Editors für Smalltalk-Formalisierung der Aufgabe (OperationalSpecification)
- Methods: Standardeinstellung bei Aufruf des Editors, Anzeige aller zur editierten Aufgabe gehörenden Methoden, Möglichkeit zur Erzeugung weiterer Methoden (TaskStructureFilterDisplay)

(31) Beschreibungen eines Objektes werden in dessen contents-Slot abgelegt, das Ziel im Slot mit dem Namen goal.

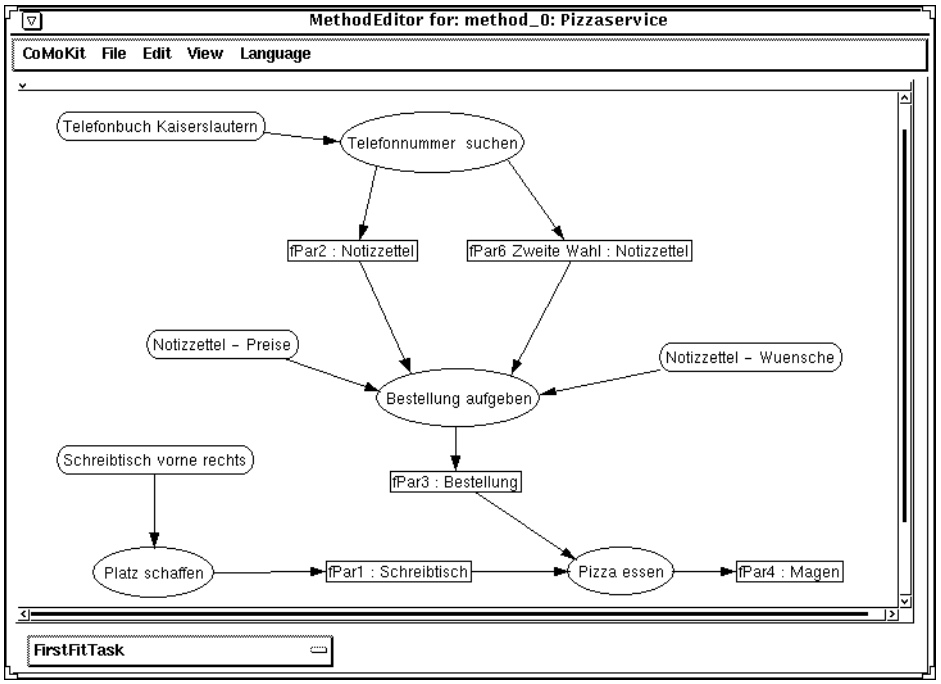
*Anzeigen Standardmodus:* `Aufgabenstruktur` (TaskStructureFilterDisplay): Alle zu der Aufgabe definierten Methoden werden als Rechtecke angezeigt. Der TaskStructureFilter bestimmt die für die Anzeige relevanten Methoden, diese müssen über TaskToMethodLinks mit der bearbeiteten Aufgabe verbunden sein.

*Mausmenüs:* mittleres Mausmenü im Standardmodus:

- Add Method: Hinzufügen einer Methode
- Edit Method: Editieren des selektierten Objektes, ruft den Methoden-Editor (siehe Anhang 2.1.3) auf.
- Delete Methods: Löschen der selektieren Objekte
- Inspect: Aufrufen eines Smalltalk-Inspektors zu dem selektierten Objekt
- Select All: Selektieren aller angezeigten Objekte
- Deselect All: Deselektieren aller angezeigten Objekte

*Schließen des Fensters* Erfolgt entweder durch Anklicken von Quit im File-Menü oder über Close im Window-Menü.

## Anhang 2.1.3 MethodEditor




*Smalltalk-Klasse:* `CoMoKitMethodEditor`

*Aufruf:* `(CoMoKitMethodEditor for: method) open`

wobei als `method` die Methode übergeben wird, die angezeigt und bearbeitet werden sollen.

*Beschreibung:* Fenster, in dem der Entwickler alle zu einer Methode relevanten Aspekte ansehen und editieren kann. Insbesondere ermöglicht der Editor, den Datenfluß zu der Methode zu spezifizieren. Wird aufgerufen, wenn im TaskAndMethodFilter (siehe Anhang 2.1.1) oder im TaskEditor (siehe Anhang 2.1.2) eine Methode selektiert und der Edit-Befehl angewählt wird.

*Pop-Down Menüs:* CoMoKit + File + View + Language:

siehe Anhang 2.1.1

Edit:

Modusumschaltung zwischen:

- Description: Einblenden eines Editors für eine Beschreibung der Methode (MethodDescriptionEditor)
- Denotation: Einblenden eines Editors für den Namen der Methode und zugeordnetes Icon (HAMObjectDenotationEditor)
- Structure: Standardeinstellung bei Aufruf des Editors, Anzeige der Methodenstruktur mit den Teilaufgaben und dem Datenfluß (MethodStructureFilterDisplay)

*Anzeigen Standardmodus:* `Methodenstruktur` (MethodStructureFilterDisplay): Hier wird der vollständige Datenfluß zu der Methode angezeigt und ergänzt. Ellipsen entsprechen Aufgaben, abgerundete Kästchen Konzeptinstanzen, Rechtecke stellen formale Parameter dar. Außerdem erscheinen die Kanten zwischen den genannten Objekten[32]. Alle angezeigten Objekte sind mit der bearbeiteten Methode über HAM-ContextLinks verbunden, anhand dieses Kriteriums ermittelt der MethodStructureFilter die für die Anzeige relevanten Objekte.

`Typ-Feld:` Selektionsfeld am unteren Fensterrand, in dem ein Aufgabentyp angewählt werden kann. Der aktuell im Feld stehende Wert wird als Typ bei der Erzeugung neuer Aufgaben benutzt.

*Mausmenüs:* mittleres Mausmenü im Standardmodus:

- Add Task: Hinzufügen einer Aufgabe. Die Aufgabe wird Instanz des Aufgabentyps, der im Typ-Feld angezeigt ist.
- Add Concept-Instance: Einfügen von Konzeptinstanzen, es erscheint eine Auswahlliste mit allen existierenden Konzeptinstanzen. Bereits im Datenfluß eingebundene Instanzen sind selektiert, die neu einzufügenden können vom Benutzer selektiert werden. Durch Deselektieren können Instanzen aus dem Datenfluß entfernt werden (siehe auch „Remove“).
- Add formal Parameter: Einfügen eines formalen Parameters, es muß der Name der Parameters angegeben werden und der Parametertyp ausgewählt werden.
- Add Link: Zuordnen von Parametern und Konzeptinstanzen als Ein- oder Ausgabe zu einer Aufgabe durch Ziehen einer Kante. Die zu verbindenden Objekte müssen selektiert sein. Das System überwacht die korrekte Verwendung von Ein- und Ausgaben der Methode (siehe Abschnitt 3.4.1). Nur erlaubte Verbindungsrichtungen werden zugelassen, bei Mehrdeutigkeit (interne Parameter, siehe Abschnitt 3.4.1) wird die Richtung der Kante vom Benutzer erfragt.
- Edit Object: Editieren des selektierten Objektes, ruft den Aufgaben-Editor (siehe Anhang 2.1.2), den Parameter-Editor (siehe Anhang 2.1.4) oder den Editor für Konzeptinstanzen auf.
- Remove: Entfernen der selektierten Konzeptinstanz aus dem Datenfluß. Die Instanz wird nicht gelöscht, sie gehört nur nicht mehr in den Datenfluß dieser Methode. Nicht entfernen kann man Instanzen, die zu den Eingaben der Methode gehören[33].
- Delete: Löschen des selektieren Objektes, nur auf Aufgaben, Parameter und Kanten anwendbar. Parameter, die zu den Eingaben oder den Ausgaben der Methode gehören, können nicht gelöscht werden. Ebenso können keine Aufgaben gelöscht werden, zu denen noch Methoden existieren. Das Löschen von Kanten setzt sich rekursiv fort, d.h. die an den Kanten hängenden Instanzen werden auch aus den Datenflüssen der Aufgabe am anderen Kantenende gelöscht (siehe Anhang 3.4.1).
- Choose Agent: mögliche Agenten zu einer Aufgabe festlegen

(32) Instanzen von Unterklassen von InputToTaskLink oder TaskToOutputLink

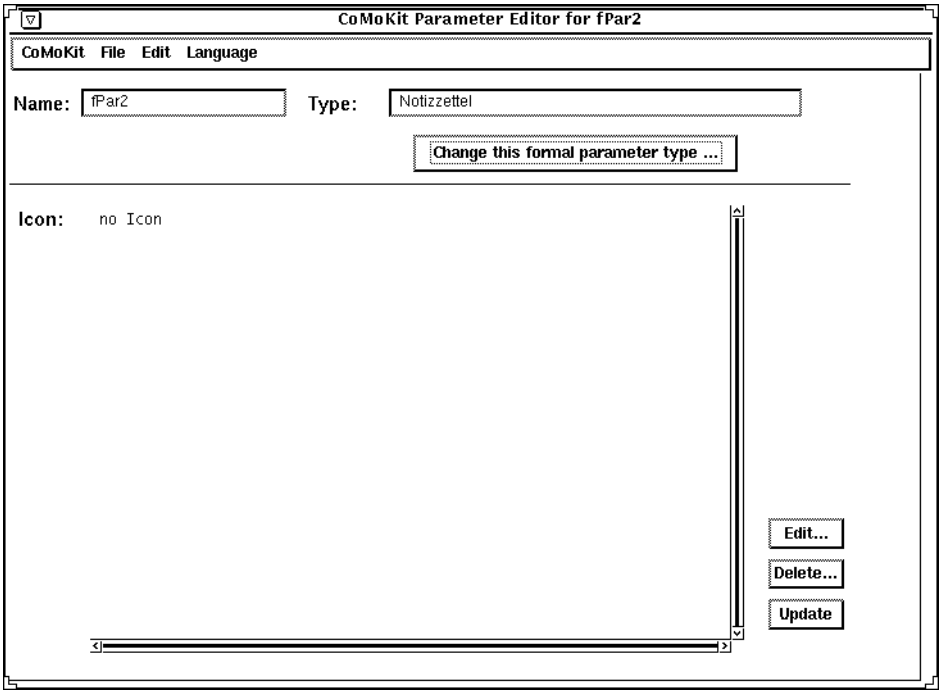
(33) Die Kennungen von Objekten, die Eingabe einer Methode sind, stehen im input-Slot der Methode, ebenso die Ausgaben im output-Slot.

- Show: Möglichkeit zur Anzeige verschiedener Netzwerkobjekte, die mit dem selektierten Objekt über Kanten verbunden sind.
- Change task type: Ordnet der selektierten Aufgabe einen anderen Aufgabentyp zu, wobei der neue Aufgabentyp durch die Anzeige im Typ-Button bestimmt ist.
- Inspect: Aufrufen eines Smalltalk-Inspektors zu dem selektierten Objekt
- Select All: Selektieren aller angezeigten Objekte
- Deselect All: Deselektieren aller angezeigten Objekte

*Schließen des Fensters*

Erfolgt entweder durch Anklicken von Quit im File-Menü oder über Close im Window-Menü.

### Anhang 2.1.4 Parameter - Editor




*Smalltalk-Klasse:* `CoMoKitParamterEditor`

*Aufruf:* `(CoMoKitParameterEditor for: parameter) open`

wobei als `parameter` der formale Parameter übergeben wird, der angezeigt und bearbeitet werden sollen.

*Beschreibung:* Fenster, in dem der Entwickler alle zu einem Parameter relevanten Aspekte ansehen und editieren kann.

*Pop-Down Menüs:* CoMoKit + File + View + Language:

siehe Anhang 2.1.1

Edit:

Modusumschaltung zwischen:

- Description: Einblenden eines Editors für die Parameterbeschreibung
- Denotation: Standardeinstellung, Einblenden eines Editors für den Namen des Parameters, das zugeordnete Icon und den Parametertyp. Der Name kann editiert werden (jeweils in allen über das Language-Menü einstellbaren Sprachen), außerdem kann der Parametertyp geändert werden.

*Anzeigen Standardmodus:* Es werden der Parametername in der im Language-Menü eingestellten Sprache sowie der Parametertyp angezeigt[34]. Außerdem erscheint in der unteren Fensterhälfte das zugeordnete Icon.

*Buttons Standardmodus:* `Change this formal parameter type:` Es erscheint eine Auswahlliste mit allen möglichen Parametertypen (alle definierten

Konzeptklassen und Typen). Auf dieser Liste kann ein Objekt ausgewählt werden, welches dann dem Parameter als Parametertyp zugewiesen wird[35].

`Edit:` Aufruf eines Fensters zum Editieren des Icons, das den Parameter symbolisieren soll.

`Delete:` Löschen des angezeigten Icons.

`Update:` Editiertes Icon dem Parameter zuweisen.

*Mausmenüs:* keine

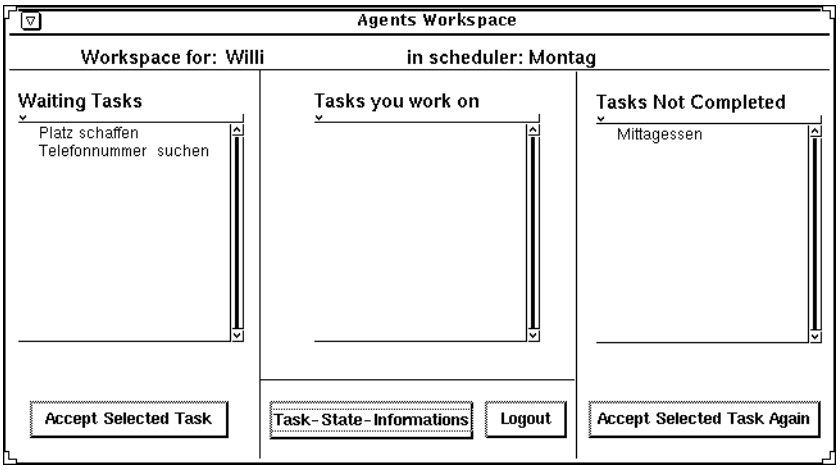
*Schließen des Fensters* Erfolgt entweder durch Anklicken von Quit im File-Menü oder über Close im Window-Menü.

(34) In den übrigen Displays wird bei der Darstellung von Parametern der eigentliche Parametername konkateniert mit dem Namen des Parametertyps angezeigt.

(35) durch Eintrag im Slot parameterType des Parameters

## Anhang 2.2 Arbeitsfenster der Clients

### Anhang 2.2.1 AgentsWorkspace




*Smalltalk-Klasse:* `ClientWorkspace`

*Aufruf:* `(ClientWorkspace newForClient: clientprocess) open`

wobei clientprocess die das Fenster aufrufende Instanz der Klasse `CoMoKitClient` bezeichnet.

*Beschreibung:* Dient als Arbeitsfenster für einen Benutzer. Es wird nach dem Login vom Benutzerprozeß erzeugt und erst beim Logout geschlossen. Das Fenster zeigt die Namen anstehender und in Bearbeitung befindlicher Aufgaben an. Im Fenster kann der Benutzer Aktionen auslösen, um die Bearbeitung der anstehenden Aufgaben zu beginnen bzw. fortzusetzen. Außerdem kann er zusätzliche Informationen abrufen.

*Anzeigen:* `Waiting Tasks:` Enthält die Bezeichnungen aller Aufgaben, die für den Benutzer ausführbar sind (die sich im Zustand „waiting“ befinden). Die Liste wird automatisch durch Mitteilungen des Schedulers aktualisiert.

`Tasks you work on:` Ermöglicht dem Benutzer eine Übersicht über alle von ihm aktuell bearbeiteten Aufgaben, auch wenn er das Bearbeitungsfenster mit der Maus iconisiert hat. Die Liste enthält die Bezeichnungen aller Aufgaben, die im Zustand „in progress“ oder „reduced“, aber noch nicht in „completed“ oder „satisfied“ sind. Außerdem Aufgaben, die „invalid“ sind, wenn der Benutzer ausdrücklich gewünscht hat, diese Aufgabe weiterzubearbeiten. Aus der Liste entfernt werden atomare Aufgaben dann, wenn der Benutzer durch den „Finished“-Button das Ende der Bearbeitung signalisiert hat, komplexe Aufgaben dann, wenn alle Unteraufgaben delegiert sind.

`Tasks Not Completed:` Ermöglicht dem Benutzer eine Übersicht über alle Aufgaben, deren Bearbeitung von ihm unterbrochen worden ist. Der Benutzer kann die Bearbeitung zu einem späteren Zeitpunkt aber wieder aufnehmen. Der Scheduler hat keine Kenntnis über die Unterbrechung, es handelt sich um einen intern vom Benutzerprozeß verwalteten Status. Die Aufgaben befinden sich in den gleichen Zuständen wie die auf der „Tasks you work on"-Liste. Allerdings wird durch Unterbrechen einer Aufgabe das Bearbeitungsfenster geschlossen. Zu den Aufgaben auf der „Tasks you work on"-Liste gibt es hingegen jeweils ein aktives Bearbeitungsfenster, auch wenn dieses eventuell iconisiert sein kann.

In der `Tasks Not Completed`-Liste werden auch nach dem Login alle Aufgaben abgelegt, die der Benutzer bei seiner letzten Arbeitssitzung bereits akzeptiert, aber nicht fertiggestellt hatte.

*Buttons:*

`Accept Selected Task:` Die in der „waiting Tasks"-Liste angewählte Aufgabe wird akzeptiert, das heißt, der Scheduler erhält die Mitteilung, daß der Agent diese Aufgabe bearbeiten will. Die Funktion dieses Button ist Default, sie kann auch durch die Return-Taste ausgelöst werden.

`Accept Selected Task Again:` Die in der „Tasks Not Completed"-Liste angewählte Aufgabe wird zur Weiterbearbeitung ausgewählt. Deren Bearbeitungsfenster wird neu erzeugt. Dieser Vorgang läuft intern innerhalb des Benutzerprozesses ab, er löst keine Zustandsänderung beim Scheduler aus.

`Logout:` Dem Scheduler wird das Logout des Client mitgeteilt. Alle dem Benutzer zugeordneten Fenster[36] werden geschlossen. Zu einem späteren Zeitpunkt kann sich der Benutzer erneut anmelden und die noch nicht fertiggestellten Aufgaben zu Ende bearbeiten.

`Task State Informations:` Aufruf des TaskStateInfo-Fensters (vergleiche Anhang 2.2.2) zur detaillierteren Übersicht über den Bearbeitungsstand aller den Benutzer betreffenden Aufgaben, sortiert nach den Zuständen, die die Aufgabenverwaltung des Schedulers unterscheidet.

*Mausmenüs:*

In der Liste `Waiting Tasks` kann über Mausmenü die Funktion des Accept-Buttons ausgelöst werden.

In der Liste `Tasks Not Completed` kann über Mausmenü die Funktion des Accept Again-Buttons ausgelöst werden.

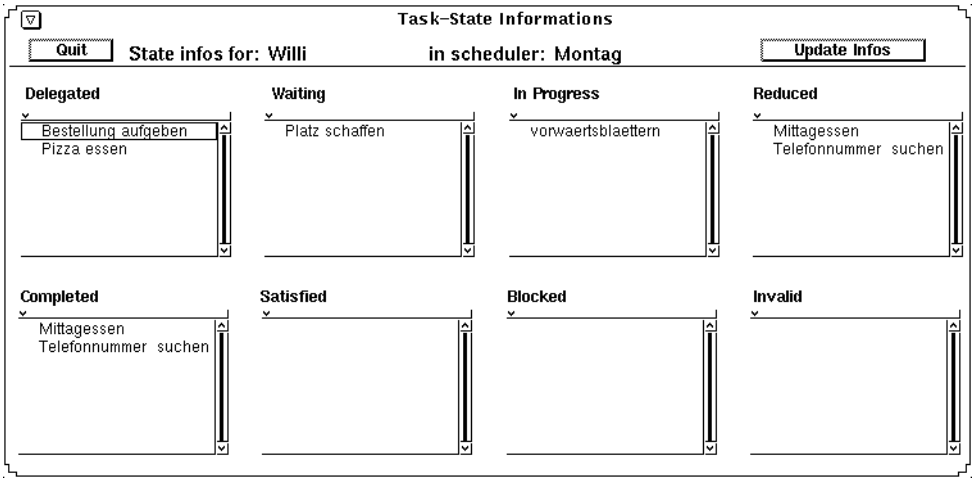
(36) Die zu einem Clientprozeß gehörenden Fenster sind in der Instanzenvariable dependentWindows der Klasse CoMoKit-Client gespeichert.

In allen Anzeigelisten kann über den Mausmenüeintrag „inspect“ ein Smalltalk-Inspector für die angewählte Aufgabe aufgerufen werden.

*Schließen des Fensters*

Erfolgt entweder durch Anklicken von Logout oder über Close im Window-Menü. Close in diesem Menü löst die gleichen Aktionen wie der Logout-Button aus.

## Anhang 2.2.2 TaskStateInfo-Fenster




*Smalltalk-Klasse:* `TaskStateInfos`

*Aufruf:* `(TaskStateInfos newForClient: clientprocess) open`

wobei clientprocess die das Fenster aufrufende Instanz der Klasse `CoMoKitClient` (Benutzerprozeß) bezeichnet.

*Beschreibung:* Dient als Übersichtsfenster für einen Benutzer. Es wird durch Anwahl des Task-State-Information Button im AgentWorkspace aufgerufen. Im Fenster sind alle Aufgaben aufgelistet, die für den Fensterbenutzer bearbeitbar sind, von ihm bearbeitet werden oder bearbeitet worden sind. Die Auflistung erfolgt sortiert nach den Zuständen, wie sie in der Aufgabenverwaltung des Schedulers erfaßt sind. Aktualisiert werden die Einträge nur auf ausdrücklichen Wunsch des Benutzers nach Anwahl des Update-Button.

Sämtliche Datenanforderungen und Meldungen an den Scheduler werden über den Benutzerprozeß veranlaßt und weitergeleitet.

*Anzeigen:* `Delegated:` Enthält die Bezeichnungen aller Aufgaben, die an den Benutzer delegiert worden sind (Aufgaben im Zustand „delegated“).

`Waiting:` Enthält die Bezeichnungen aller Aufgaben, die für den Benutzer ausführbar sind (Aufgaben im Zustand „waiting“).

`In Progress:` Enthält die Bezeichnungen aller Aufgaben, die der Benutzer akzeptiert hat, für die aber noch keine Methode angewandt wurde (Aufgaben im Zustand „in Progress“).

`Reduced:` Enthält die Bezeichnungen aller Aufgaben, auf die der Benutzer erfolgreich eine Methode angewandt hat (Aufgaben im Zustand „reduced“).

`Completed:` Enthält die Bezeichnungen aller komplexen Aufgaben, bei denen Benutzer alle Teilaufgaben delegiert hat (Aufgaben im Zustand „completed“).

`Satisfied:` Enthält die Bezeichnungen aller Aufgaben, die vom Benutzer bearbeitet wurden und die nun einschließlich ihrer Teilaufgaben vollständig gelöst sind (Aufgaben im Zustand „satisfied“).

`Blocked:` Enthält die Bezeichnungen aller Aufgaben, die der Benutzer bearbeitet hat, die aber nun blockiert sind (Aufgaben im Zustand „ blocked“). Sie sind dadurch gekennzeichnet, daß alle Methoden, die zu den Aufgaben definiert sind, bereits zurückgezogen sind. Um eine solche Aufgabe weiter zu bearbeiten, muß der Benutzer einen solchen Rückzug aufheben (siehe Mausmenüs).

`Invalid:` Enthält die Bezeichnungen aller Aufgaben, die der Benutzer bearbeitet hat, die aber nun ungültig sind (Aufgaben im Zustand „invalid“).

*Buttons:*

`Update Infos:` Fordert für alle Anzeigelisten neue Informationen beim Scheduler an und aktualisiert damit die Listen. Dieser Button ist Default, er kann auch durch die Return-Taste angewählt werden.

`Quit:` Schließt das Fenster.

*Mausmenüs:*

In allen Listen kann über den Mausmenüeintrag „inspectTask“ ein Smalltalk-Inspector für die angewählte Aufgabe aufgerufen werden.

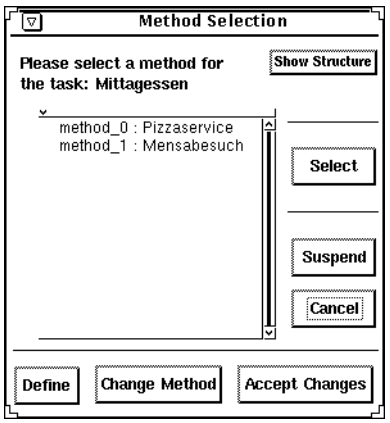
In der `Blocked`-Liste kann über den Mausmenüeintrag „solve goal block“ ein Fenster aufgerufen werden, in dem die Blockierung aufgelöst werden kann (siehe Anhang 3.2, unrejectMethod: ofBlokkedTask:).

*Schließen des Fensters*

Erfolgt entweder durch Anklicken des Quit-Button, automatisch beim Logout des zugehörigen Benutzers oder über Close im Window-Menü.

## Anhang 2.3 Bearbeitung komplexer Aufgaben

### Anhang 2.3.1 Method Selection Window




*Smalltalk-Klasse:* `MethodSelectionWindow`

*Aufruf:* `(MethodSelectionWindow newFor: clientprocess task: taskID methods: methodList) open`

wobei clientprocess die das Fenster aufrufende Instanz der Klasse `CoMoKitClient` (Benutzerprozeß) bezeichnet, taskID die zu bearbeitende Aufgabe identifiziert und methodList eine Liste aller auswählbaren Methoden ist.

*Beschreibung:* Dient zur Auswahl einer komplexen Methode. Wird vom Benutzerprozeß aufgerufen, nachdem dieser vom Scheduler die Nachricht „inProgress:“ (siehe Anhang 3.2) erhalten hat. Methodenauswahl ist nur dann erforderlich, wenn es sich um eine komplexe Aufgabe handelt, der mehrere komplexe Methoden zugeordnet sind. Ist nur eine komplexe Methode vorhanden oder handelt es sich um eine atomare Aufgabe, so ist die Auswahl trivial und wird automatisch durchgeführt.

Neben der Auswahl von existierenden Methoden kann über dieses Fenster auch die Neudefinition einer Methode zu der bearbeiteten Aufgabe oder die Änderung einer Methode veranlaßt werden.

Sämtliche Datenanforderungen und Meldungen an den Scheduler werden über den Benutzerprozeß weitergeleitet.

*Anzeigen:* `Methodenliste:` Enthält die Bezeichnungen aller anwählbaren Methoden.

*Buttons:* `Select:` Selektierte Methode wird ausgewählt, dies wird dem Scheduler mitgeteilt. Danach können im Complex Method Processing-Window (siehe Anhang 2.3.2) die durchzuführenden Delegatio-

nen bearbeitet werden. Dieser Button ist Default, er kann auch durch die Return-Taste angewählt werden.

`Cancel:` Schließt das Fenster, die Methodenauswahl gilt als gescheitert, der Scheduler erhält hierüber eine Mitteilung.

`Suspend:` Schließt das Fenster, die Methodenauswahl wird unterbrochen, kann aber später durch Anwahl der Aufgabe in der „Tasks Not Completed“-Liste fortgesetzt werden.

`Show Structure:` Es wird ein MethodStructure-Filter (siehe Anhang 2.1.3) aufgerufen, indem sich der Bearbeiter den Datenfluß zu der selektierten Methode ansehen kann. Er kann sich damit vor der Entscheidung Informationen über die möglichen Methoden einholen.

Folgende Buttons wurden aufgrund der in Abschnitt 5.3 vorgestellten Erweiterungen eingefügt:

`Define:` Es wird ein TaskStructure-Filter (siehe Anhang 2.1.2) aufgerufen, in dem der Bearbeiter eine neue Methode zur bearbeiteten Aufgabe anlegen kann. Er muß zu dieser Methode im entsprechenden Methodeneditor einen Datenfluß spezifizieren. Anschließend kann er über den Accept Changes-Button den Scheduler veranlassen, diese Methode im die Methodenliste aufzunehmen und somit auswählbar zu machen.

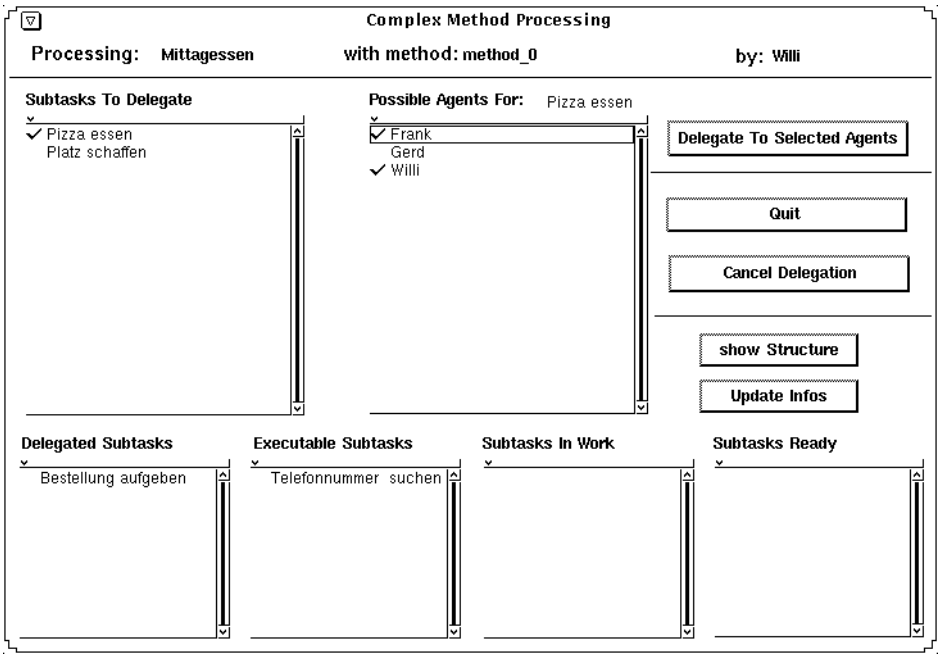
`Change Method:` Es wird ein MethodStructure-Filter (siehe Anhang 2.1.3) aufgerufen, in dem sich der Bearbeiter den Datenfluß zu der selektierten Methode ansehen und ihn ändern kann. Er kann somit die Spezifikation einer Methode noch zur Laufzeit des Interpreters nach seinen Vorstellungen aktualisieren.

`Accept Changes:` Durch Anwahl dieses Buttons teilt der Benutzer dem Scheduler mit, daß er die mit „Define“ oder „Change Method“ begonnenen Änderungen oder Erweiterungen abgeschlossen hat, der Scheduler liest daraufhin die zur bearbeiteten Aufgabe gehörenden Methoden neu aus dem konzeptuellen Modell ein. Die angezeigt Methodenliste wird neu aufgebaut.

*Mausmenüs:* keine

*Schließen des Fensters* Erfolgt entweder durch Anklicken des Select-Button, automatisch beim Logout des zugehörigen Benutzers oder über Close im Window-Menü. Close im Window-Menü bedeutet, genau wie der Suspend-Button, ein Unterbrechen der Methodenauswahl.

## Anhang 2.3.2 Complex Method Processing Window




*Smalltalk-Klasse:* `ComplexMethodInterpreter`

*Aufruf:* `(ComplexMethodInterpreter newFor: clientprocess task: taskID method: methodID) open`

wobei clientprocess die das Fenster aufrufende Instanz der Klasse `CoMoKitClient` bezeichnet, taskID die zu bearbeitende Aufgabe und methodID die angewandte Methode identifiziert.

*Beschreibung:* Dient zur Durchführung einer komplexen Methode. Wird aufgerufen, nachdem die anzuwendende komplexe Methode vom Benutzer ausgewählt wurde. Aufruf erfolgt, indem an diese Methode (Klasse ComplexMethod) ein „doItFor: clientprocess“ geschickt wird. Im Fenster erhält der Benutzer die zu delegierenden Teilaufgaben angezeigt. Nach Anwahl einer Teilaufgabe werden die möglichen Bearbeiter eingeblendet, eine Teilmenge dieser Agenten kann ausgewählt werden und die Aufgabe kann an sie delegiert werden. Außerdem ermöglich das Fenster, den Verlauf der Bearbeitung der Teilaufgaben mitzuverfolgen. Die dazu am unteren Fensterrand vorhandenen Listen werden nur auf Wunsch des Benutzers aktualisiert.

Sämtliche Datenanforderungen und Meldungen an den Scheduler werden über den Benutzerprozeß weitergeleitet.

*Anzeigen:* `Subtasks to Delegate:` Liste der zu delegierenden Teilaufgaben

`Possible Agents For:` Liste der möglichen Bearbeiter für die in der „Subtasks to Delegate“-Liste angewählte Aufgabe. Sollten im

konzeptuellen Modell zu dieser Aufgabe keine möglichen Bearbeiter definiert sein, so werden alle im konzeptuellen Modell definierten Agenten als mögliche Bearbeiter angezeigt.

`Delegated Subtasks:` Liste der bereits delegierten Teilaufgaben, die noch nicht ausführbar geworden sind. In diese Liste aufgenommen werden Teilaufgaben, sobald durch den Delegate-Button die Aufgabe delegiert ist, ansonsten wird die Liste nur bei Anwahl des Update-Button aktualisiert.

`Executable Subtasks:` Liste der bereits delegierten Teilaufgaben, die ausführbar sind. Aktualisiert nur bei Anwahl des Update-Button.

`Subtasks In Work:` Liste der delegieren Teilaufgaben, die von einem Bearbeiter akzeptiert wurden und in Bearbeitung sind (Aufgaben im Zustand „in progress“). Aktualisiert nur bei Anwahl des Update-Button.

`Subtasks Ready:` Liste der delegierten Teilaufgaben, die bereits bearbeitet sind (Aufgaben im Zustand „reduced“). Aktualisiert nur bei Anwahl des Update-Button.

*Buttons:*

`Delegate To Selected Agents:` Die ausgewählte Teilaufgabe wird an die ausgewählten Agenten delegiert, die Teilaufgabe verschwindet daraufhin aus der Liste „Subtasks to delegate“ und erscheint in der „Delegated Subtasks“ bzw. der „Executable Subtasks“-Liste.

`Quit:` Schließt das Fenster, die komplexe Aufgabe wird vom Benutzerprozeß als nicht fertiggestellt betrachtet und im Arbeitsfenster (siehe Anhang 2.2.1) als „Task Not Completed“ angezeigt. Sie kann dort zur Fertigstellung erneut ausgewählt werden. Dies gilt nur, solange es noch zu delegierende Teilaufgaben gibt. Sind alle Teilaufgaben delegiert, wird das Fenster nicht mehr in der „Tasks Not Completed“-Liste angezeigt.

`Cancel Delegation:` Schließt das Fenster, die Bearbeitung der komplexen Methode gilt als gescheitert, der Scheduler erhält hierüber eine Mitteilung. Diese Funktion ist auch dann möglich, wenn bereits Teilaufgaben delegiert sind, diese werden dann ungültig.

`Show Structure:` Es wird ein MethodStructure-Filter (siehe Anhang 2.1.3) aufgerufen, in dem sich der Bearbeiter den Datenfluß der bearbeiteten komplexen Methode ansehen kann. Er kann sich damit vor den Delegierungsentscheidungen Informationen über den genauen Ablauf der Bearbeitung einholen.

`Update Infos:` Die vier Anzeigelisten am unteren Fensterrand, in denen die weitere Bearbeitung der delegierten Teilaufgaben mit-

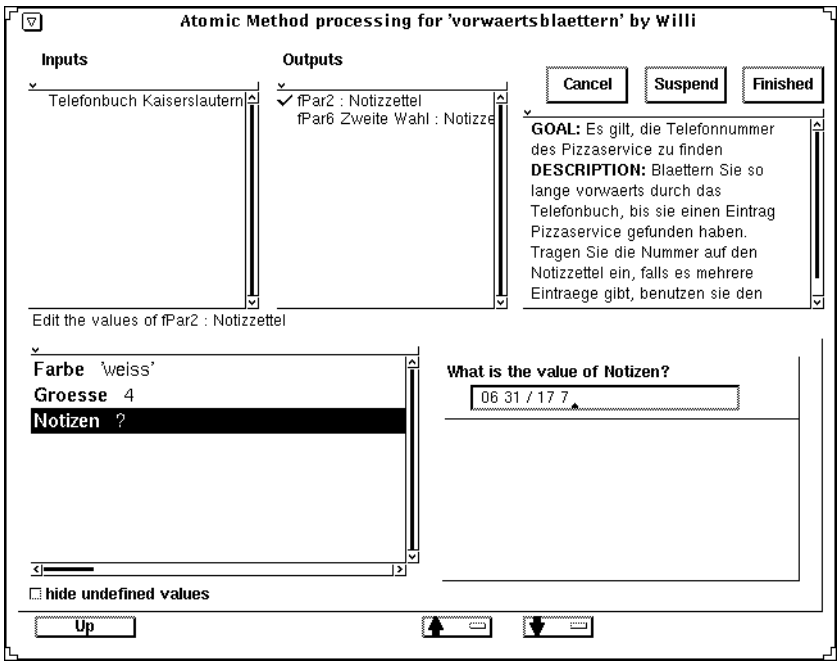
verfolgt werden kann, werden durch neue Anforderung der Daten aktualisiert.

*Mausmenüs:* In allen Anzeigelisten für Aufgaben kann über den Mausmenüeintrag „inspect“ ein Smalltalk-Inspector für die angewählte Aufgabe aufgerufen werden.

*Schließen des Fensters* Erfolgt entweder durch Anklicken des Quit-Button, Anklicken des Cancel Delegation-Button, automatisch beim Logout des zugehörigen Benutzers oder über Close im Window-Menü. Close im Window-Menü löst die gleichen Folgeaktionen wie der Quit-Button aus.

## Anhang 2.4 Bearbeitung atomarer Aufgaben

### Anhang 2.4.1 Atomic Method Processing Window




*Smalltalk-Klasse:* `AtomicMethodInterpreter`

*Aufruf:*

```
(AtomicMethodInterpreter newFor: clientprocess
task: taskID method: methodID inputs: inList
output: outList) open
```

wobei clientprocess die das Fenster aufrufende Instanz der Klasse `CoMoKitClient` (Benutzerprozeß) bezeichnet, taskID die zu bearbeitende Aufgabe und methodID die angewandte Methode identifiziert. Die Liste inList enthält die Konzeptinstanzen, die Eingabe der Aufgabe sind. Die Liste outList enthält die Konzeptinstanzen, die Ausgabe der Aufgabe sind, wobei die Slots der Ausgaben noch nicht belegt sind.

*Beschreibung:* Dient zur Durchführung einer atomaren Methode. Wird aufgerufen, nachdem vom Benutzer eine Aufgabe akzeptiert wurde, zu der nur eine atomare Methode existiert. Aufruf erfolgt, indem an diese Methode (Instanz der Klasse AtomicMethod) ein „doItFor: clientprocess“ geschickt wird. Im Fenster erhält der Benutzer das Ziel der Aufgabe und eine Aufgabenbeschreibung angezeigt. Außerdem in zwei Listen die Eingaben und die zu erzeugenden Ausgaben. Er kann sich die Slotbelegungen der Eingabeinstanzen in der unteren Fensterhälfte (HyperstackConceptEditor) ansehen. Dort kann er auch die Slots der Ausgabekonzepte mit Werten belegen.

Sämtliche Datenanforderungen und Meldungen an den Scheduler werden über den Benutzerprozeß weitergeleitet.

*Anzeigen:*

`Inputs:` Liste der Konzeptinstanzen, die Eingabe der Aufgabe sind. Hierzu gehören auch formale Parameter, die Umsetzung der Parameter in Instanzen erfolgt im Scheduler.

`Outputs:` Liste der Konzeptinstanzen, die Ausgabe der Aufgabe sind. Es handelt sich ausschließlich um formale Parameter, die Umsetzung der Parameter in Instanzen erfolgt im Scheduler.

`Goal:` Ziel der Aufgabe, wird im TaskEditor (siehe Anhang 2.1.2), Modus „Contents“ eingegeben.

`Description:` Aufgabenbeschreibung, wird im TaskEditor (siehe Anhang 2.1.2), Modus „Contents“ eingegeben.

`untere Fensterhälfte:` Instanz der Klasse HyperstackConceptEditor, dient zur Anzeige und zum Editieren von Konzeptinstanzen.

*Buttons:*

`Finished:` Die bearbeitete Aufgabe wird als abgeschlossen betrachtet, dies wird dem Benutzerprozeß mitgeteilt. Über diesen werden die Ausgabekonzeptinstanzen an den Scheduler geschickt.

`Suspend:` Schließt das Fenster, die Aufgabe wird vom Benutzerprozeß als nicht fertiggestellt betrachtet und im Arbeitsfenster (siehe Anhang 2.2.1) als „Task Not Completed“ angezeigt. Sie kann dort zur Fertigstellung erneut ausgewählt werden.

`Cancel:` Schließt das Fenster, die Bearbeitung der komplexen Aufgabe gilt als gescheitert, der Scheduler erhält hierüber eine Mitteilung.

*Mausmenüs:* keine

*Schließen des Fensters* Erfolgt entweder durch Anklicken des Cancel-Button, Anklicken des Finished-Button, automatisch beim Logout des zugehörigen Benutzers oder über Close im Window-Menü. Close im Window-Menü löst die gleichen Folgeaktionen wie der Suspend-Button aus.

# ANHANG 3 Client-Server Schnittstelle

*Dieser Teil enthält eine Auflistung und Erklärung aller Nachrichten, die über die Schnittstelle zwischen der Steuerungskomponente und den Benutzerprozessen ausgetauscht werden.*

Als Parameter in den nachfolgend aufgelisteten Nachrichtenaufrufen werden, soweit nicht anders angegeben, in der Regel nur Objektidentifikatoren benutzt. Es handelt sich um die „hamIDs“, Kennungen, mit denen die Objekte im zugrundeliegenden Hypermedianetz identifiziert werden. In der folgenden Auflistung sind diese Identifikatoren jeweils mit ID bezeichnet, nämlich taskID, agentID, methodID. Es wird nicht mehr explizit erwähnt, daß hiermit nicht das Objekt selbst, sondern nur ein Objektidentifikator übergeben wird.

Weitere Erläuterungen zur Schnittstelle finden sich in Abschnitt 4.6.

## Anhang 3.1 Anfragen der Clients an den Scheduler

- Datenanforderung für Arbeitsfenster des Benutzers (Agents Workspace, siehe Anhang 2.2.1):
  - getWaitingTasksFor: agentID
    Funktion dient zum Füllen der Anzeigeliste „Waiting Tasks“ bei Initialisierung des Fensters. Rückgabewert ist eine Liste mit den IDs der für den Absender der Anforderungen ausführbaren Aufgaben. Änderungen der Anzeigen im Arbeitsfenster werden durch die Nachrichten in Anhang 3.3 ausgelöst.
  - getTasksInProgressFor: agentID
    Rückgabewert ist eine Liste mit den IDs aller Aufgaben, die nach der letzten Arbeitssitzung der Benutzers in Zustand „inProgress“ waren. Diese Aufgaben werden bei Initialisierung des Fensters in der Anzeigeliste „Tasks Not Completed“ angezeigt, damit sie vom Benutzer durch Anwahl des Button „Accept Selected Task Again“ fertiggestellt werden können.
  - getReducedTasksFor: agentID
    Rückgabewert ist eine Liste mit den IDs aller Aufgaben, die nach der letzten Arbeitssitzung der Benutzers im Zustand „reduced“, aber noch nicht im

Zustand „completed“ waren. Diese Aufgaben werden bei Initialisierung des Fensters in der Anzeigeliste „Tasks Not Completed“ angezeigt, damit sie vom Benutzer durch Anwahl des Button „Accept Selected Task Again“ fertiggestellt werden können.

- Datenanforderung für Informationsfenster des Benutzers (Task-StateInfo-Fenster, siehe Anhang 2.2.2):
  - getDelegatedTasksFor: agentID
    Funktion dient zum Füllen der Liste „delegated“ bei Initialisierung des Fensters sowie zum Aktualisieren der Liste bei Anwahl des „Update“-Button. Rückgabewert ist eine Liste mit den IDs aller an den Fensterbenutzer delegierten, aber noch nicht ausführbaren Aufgaben.
  - getWaitingTasksFor: agentID
    analog für Liste „waiting“ - ausführbare Aufgaben (Dieser Aufruf wird auch vom Arbeitsfenster benutzt - siehe oben.)
  - getTasksInProgressFor: agentID
    analog für Liste „in progress“ - in Bearbeitung befindliche Aufgaben (Dieser Aufruf wird auch vom Arbeitsfenster benutzt - siehe oben.)
  - getReducedTasksFor: agentID
    analog für Liste „reduced“ - Aufgaben, bei denen bereits eine Methode angewandt wurde (Dieser Aufruf wird auch vom Arbeitsfenster benutzt - siehe oben.)
  - getCompletedTasksFor: agentID
    analog für Liste „completed“ - komplexe Aufgaben, bei denen alle Teilaufgaben delegiert wurden
  - getSatisfiedTasksFor: agentID
    analog für Liste „satisfied“ - Aufgaben, bei denen alle Teilaufgaben vollständig gelöst wurden
  - getBlockedTasksFor: agentID
    analog für Liste „blocked“ - blockierte Aufgaben
  - getInvalidTasksFor: agentID
    analog für Liste „invalid“ - ungültige Aufgaben

- Datenanforderung zur Aufgabenbearbeitung:
  - possibleMethodsFor: taskID
    Liefert als Rückgabewert eine Liste aller zu der Aufgabe taskID noch ausführbaren Methoden. Falls es sich um mehrere Methoden handelt, so muß der Benutzer eine dieser Methoden im MethodSelectionWindow (siehe Anhang 2.3.1) auswählen.
  - getValidDecisionForTask: taskID agent: agentID
    Mitteilung, daß der Benutzer die Aufgabe taskID fertigstellen möchte, die er bereits einmal akzeptiert und eventuell schon teilweise bearbeitet hatte. Wird ausgelöst durch Anwahl einer Aufgabe in der „Tasks not completed“-Liste des Arbeitsfensters.
    Falls zu der Aufgabe eine gültige komplexe Methode vorliegt, wird deren Kennung als Rückgabewert geschickt. Es kann dann mittels dieser Methode weitergearbeitet werden. Im Bearbeitungsfenster werden als zu delegierende Teilaufgaben nur noch die angezeigt, die beim vorhergehenden Arbeitsgang noch nicht delegiert wurden.
    Falls es keine gültige Methode gibt, ist der Rückgabewert „nil“, dann muß eine neue Methodenauswahl durchgeführt werden. In diesem Fall erscheint entweder das Methodauswahlfenster oder - falls es nur eine Methode gibt - das Fenster zur Bearbeitung dieser atomaren oder komplexen Methode.

- Datenanforderung für Bearbeitungsfenster atomarer Aufgaben (Atomic Method Processing Window, siehe Anhang 2.4.1):

  Übergeben werden jeweils Kennungen der bearbeiteten Aufgabe (taskID) und der ausgewählten Methode (methodID).

  - getInputVariablesForTask: taskID method: methodID
    Liefert als Rückgabewert eine Liste aller Eingaben der atomaren Aufgabe taskID. Die anzuwendende atomare Methode ist bezeichnet durch methodID. Es handelt sich dabei ausschließlich um Konzeptinstanzen, die formalen Parameter sind bereits in Instanzen umgewandelt.
    Achtung: Rückgabewert sind keine IDs, sondern die konkreten Instanzen!
  - getOutputVariablesForTask: taskID method: methodID
    Liefert entsprechend als Rückgabewert eine Liste aller Ausgaben für die atomare Aufgabe. Die formalen Parameter sind ebenfalls bereits in Instanzen umgewandelt. Die Slots der Instanzen sind nicht gefüllt, sie werden erst durch Bearbeitung der atomaren Methode belegt.
    Achtung: Rückgabewert sind keine IDs, sondern die konkreten Instanzen!

- Datenanforderung für Bearbeitungsfenster komplexer Aufgaben (Complex Method Processing Window, siehe Anhang 2.3.2):

  Übergeben werden jeweils Kennungen der bearbeiteten Aufgabe (taskID), der ausgewählten Methode (methodID) und des anfragenden Agenten (agentID).

  - getSubtasksToBeDelegatedOfTask: taskID method: methodID for: agentID
    Funktion dient zum Füllen der Anzeigeliste „Subtasks To Delegate“ bei Initialisierung des Fensters. Rückgabewert ist eine Liste mit den IDs der zu delegierenden Teilaufgaben.
  - getDelegatedSubtasksOfTask: taskID method: methodID for: agentID
    Funktion dient zum Füllen der Anzeigeliste „Delegated Subtasks“ bei Initialisierung des Fensters sowie zum Aktualisieren der Liste bei Anwahl des „Update Infos“-Button. Rückgabewert ist eine Liste mit den IDs der bereits delegierten Teilaufgaben, die noch nicht ausführbar sind.
  - getWaitingSubtasksOfTask: taskID method: methodID for: agentID
    analog für Liste „executable Subtasks“ - bereits delegierte und ausführbar gewordene Teilaufgaben
  - getSubtasksInWorkOfTask: taskID method: methodID for: agentID
    analog für Liste „Subtasks in Work“ - bei einem beliebigen Benutzer in Bearbeitung befindliche Teilaufgaben
  - getReadySubtasksOfTask: taskID method: methodID for: agentID
    analog für Liste „Subtasks ready“ - bereits bearbeitete Teilaufgaben (Teilaufgaben im Zustand reduced)

- Datenanforderung für die Fenster zum Auflösen von Blockierungen (siehe [Dellen 94]):

  Übergeben wird jeweils eine Kennung des blockierten Prozeß.

  - blockedTaskID: processID
    Liefert die Kennung der blockierten Aufgabe.
  - agentOfBlockedTask: processID
    Liefert die Kennung des Bearbeiters der blockierten Aufgabe.
  - superTaskID: processID
    Liefert die Kennung der zur blockierten Aufgabe übergeordneten Aufgabe.
  - agentOfSuperTask: processID
    Liefert die Kennung des Bearbeiters der übergeordneten Aufgabe.

- rejectedMethodIDs: processID
  Liefert die Kennungen aller zurückgezogenen Methoden zu der blockierten Aufgabe.
- rejectedMethodByUserIDs: processID
  Liefert die Kennungen aller Methoden, die durch den Bearbeiter der Blokkierung zurückgezogen wurden.

## Anhang 3.2 Mitteilungen der Clients an den Scheduler

- login: agentID client: process
  Mitteilung, daß der Agent agentID am Schedulingprozeß teilnehmen will, liefert „false“, falls der Agent bereits angemeldet ist (es ist stets nur eine Anmeldung des Benutzers an einen Scheduler möglich). Liefert „true“ falls der Agent noch nicht angemeldet ist. Als Wert von process wird der Benutzerprozeß übergeben. Der Scheduler benötigt diesen als Adresse für die Nachrichten, die vom Scheduler im Laufe der Bearbeitung verschickt werden.
- logout: agentID client: process
  Entfernt den Agenten aus der Verwaltung des Schedulers. Danach wird der Cientprozeß beendet, alle noch zu diesem Prozeß gehörenden Fenster werden geschlossen.
- acceptedTask: taskID agent: agentID
  Mitteilung, daß der Benutzer agentID die Aufgabe taskID akzeptiert hat. Der Scheduler setzt daraufhin diese Aufgabe vom Zustand executable nach in progress, alle anderen Agenten werden mit der Nachricht „notWaiting:“ informiert, daß die Aufgabe für sie nicht mehr ausführbar ist.
  Die Bearbeitung der Aufgabe darf der Client erst beginnen, wenn der Scheduler die „acceptedTask:“-Nachricht mit der Mitteilung „inProgress:“ (siehe Anhang 3.3) bestätigt hat.
- reducedTask: taskID method: methodID
  Mitteilung, daß zu der Aufgabe taskID vom Benutzer eine komplexe Methode ausgewählt worden ist und nun das Delegieren der Teilaufgaben beginnt.
- reducedTask: taskID method: methodID output: (Set of Instances)
  Mitteilung, daß zu der Aufgabe taskID eine atomare Methode erfolgreich bearbeitet wurde und die übergebenen Resultate erzeugt wurden.
  Achtung: Als output werden keine IDs übergeben, sondern die konkreten Instanzen!
- methodChooseFailed: taskID
  Mitteilung, daß der Benutzer die Methodenauswahl zu der komplexen Aufgabe taskID abgebrochen hat.
- delegateTask: taskID to: (Set of AgentIDs)
  Mitteilung, daß die Unteraufgabe taskID an die mitgelieferte Menge von Agenten delegiert wurde.
- methodFailed: methodID inTask:taskID
  Mitteilung, daß die Bearbeitung einer komplexen oder atomaren Aufgabe vom Benutzer abgebrochen wurde und die Methode somit als fehlgeschlagen gilt.
- rejectMethod: methodID forBlockedTask:taskID
  Diese Nachricht wird bei der Bearbeitung einer Blockierung ausgelöst. Der Wert taskID bezeichnet eine blockierte Aufgabe. Der Bearbeiter will die

Blockade auflösen, indem er die Methode, deren Anwendung die nun blokkierte Aufgabe erzeugt hat, zurückzieht.
MethodID bezeichnet die Methode, die zurückgezogen werden soll.

- unrejectMethod: methodID ofBlockedTask: taskID
  Diese Nachricht wird bei der Bearbeitung einer Blockierung ausgelöst. Der Wert taskID bezeichnet eine blockierte Aufgabe. Zu ihr gibt es keine anwendbare Methode mehr. Alle Methoden sind bereits gescheitert und zurückgezogen worden. Der Bearbeiter will nun die Blockade auflösen, indem er eine der gescheiterten Methoden erneut anzuwenden versucht, er hebt den Methodenrückzug auf und macht die Methode wieder gültig.
  MethodID bezeichnet die Methode, deren Rückzug aufgehoben werden soll. Durch die Rücknahme des Methodenrückzuges wird die Methode wieder anwendbar. Bei einer atomaren Methode heißt dies, daß der Benutzer erneut die Möglichkeit erhält, diese atomare Aufgabe zu bearbeiten. Dabei muß er die selben Eingabedaten wie zuvor benutzen. Bei einer komplexen Methode bedeutet es, daß der Benutzer nochmals den Versuch machen muß, die Teilaufgaben zu delegieren.

Eine Sonderstellung hat folgende Nachricht, da sie nicht von einem gewöhnlichen Benutzer, sondern nur vom CoMo-Kit Superuser an den Scheduler geschickt wird:

- initializedTask: taskID delegatedTo: (Set of AgentIDs)
  Mitteilung des CoMo-Kit Superusers an den Scheduler, daß die Aufgabe taskID initialisiert und an die aufgelisteten Agenten delegiert wurde. Die Aufgabe wird vom Scheduler in die Aufgabenverwaltung eingefügt und an die Agenten delegiert.

## Anhang 3.3 Mitteilungen über Zustandsänderungen vom Scheduler an Clients

Nachfolgende Nachrichten werden vom Scheduler an alle von der Änderung betroffenen Benutzer geschickt. Sie beinhalten jeweils Zustandsänderungen einer Aufgabe.

- waiting: taskID
  Mitteilung, daß die Aufgabe taskID für den Empfänger ausführbar ist. Die Aufgabe wird daraufhin im Arbeitsfenster als „waitingTask“ angezeigt und kann zur Ausführung vom Benutzer akzeptiert werden.
- notWaiting: taskID
  Mitteilung, daß die Aufgabe taskID für den Empfänger nicht mehr ausführbar ist. Die Aufgabe ist von einem anderen Agenten akzeptiert worden und wird von diesem bearbeitet. Sie darf nun im Arbeitsfenster nicht mehr als „waitingTask“ angezeigt werden.
- inProgress: taskID
  Mitteilung, daß die Aufgabe taskID vom Empfänger bearbeitet werden kann. Dazu muß der Empfänger diese Aufgabe zuvor akzeptiert haben und dies dem Scheduler mit der Nachricht acceptedTask mitgeteilt haben. Die Aufgabe wird nun aus der Anzeige der „waiting Tasks“ im Arbeitsfenster entfernt. Sie wird im Arbeitsfenster als „Task you work on“ angezeigt und die Bearbeitung beginnt.
  Dazu fordert der Client mit der Nachricht possibleMethodsFor: (siehe Anhang 3.1) die zu der Aufgabe spezifizierten Methoden an. Falls es nur eine atomare Methode gibt, wird das AtomicMethodProcessingWindow

aufgerufen. Dieses fordert vom Scheduler das nötige Problemfallwissen an (getInputVariables, getOutputVariables). Bei mehreren möglichen Methoden wird das MethodSelectionWindow gestartet, dort kann der Benutzer eine komplexe Methode auswählen.

- invalid: taskID
  Mitteilung, daß die Aufgabe taskID ungültig geworden ist. Die Reaktion des Benutzerprozesses hängt davon ab, wieweit der Benutzer bereits mit der Bearbeitung der Aufgabe vorangekommen ist. Hat er die Aufgabe noch nicht akzeptiert, so wird diese von der Anzeigeliste „waitingTasks“ im Arbeitsfenster genommen und der Benutzer darüber informiert. Ist die Aufgabe jedoch in Bearbeitung oder bereits teilweise fertiggestellt, so wird der Benutzer gefragt, ob er die Bearbeitung zu Ende führen möchte. Ihm wird mitgeteilt, daß die Resultate momentan nicht benötigt werden.
- solveGoalBlock: taskID
  Mitteilung, daß die vom Nachrichtenempfänger delegierte Teilaufgabe mit der Kennung taskID blockiert ist. Der Benutzer ist nun aufgefordert, diese Blockierung aufzulösen. Dies geschieht in einem speziellen Fenster, das vom Clientprozeß am Bildschirm aufgebaut wird (siehe [Dellen 94]). Dort kann der Benutzer die Methode, zu deren Teilaufgaben die blockierte Aufgabe gehört, zurückziehen (siehe Abschnitt 3.2: „rejectMethod: forBlokkedTask“).
  Sofern es nun zu der Aufgabe, die der blockierten Aufgabe übergeordnet ist, noch andere Methoden gibt, kann er anschließend an den Rückzug eine neue Methode auswählen. Gibt es keine alternative Methode mehr, so ist auch die übergeordnete Aufgabe blockiert.
  Um weiterarbeiten zu können, müßte der Agent in diesem Fall in seinem Informationsfenster in der blocked-Liste die Aufgabe selektieren. Nach Anwahl des Befehl „solve goal block“ (siehe Seite 75) baut der Benutzerprozeß dann ein weiteres Bearbeitungsfenster auf, in dem der Benutzer eine der zurückgezogenen Methoden wieder anwendbar machen kann (er zieht den Rückzug zurück, siehe Abschnitt 3.2: „unrejectMethod: ofBlokkedTask“).