Universität Kaiserslautern
Fachbereich Informatik
AG Künstliche Intelligenz: Expertensysteme
Prof. Dr. Michael M. Richter

Diplomarbeit

Dynamische Änderungen an Prozessmodellen

von

Volkmar Pipek

Betreuer:

Prof. Dr. Michael M. Richter
Dr. rer. nat. Frank Maurer
Dipl.-Inform. Barbara Dellen

Kaiserslautern, im November 1996

Inhaltsverzeichnis

1 Einleitung

Der Bereich der *Workflow-Management-Systeme* (WFMS - z.B. [Jab95ab]) wird in jüngerer Zeit in verschiedenen Bereichen der Informatik genauer erforscht. Ziel der Bemühungen ist es, die besonderen Anforderungen , die WFMS an Rechner- und Programmsysteme stellen, zu ermitteln und zu befriedigen.
In dieser Arbeit untersuchen wir Aspekte des Umplanens („Replanning“ bzw. „Remodeling“) während der Abarbeitung eines Workflows. Sie entstand im Rahmen des Projektes CoMo-Kit, im Rahmen dessen Methoden und Werkzeuge entwickelt werden, die die Planung und das Management komplexer Arbeitsabläufe, insbesondere im Entwurfsbereich, unterstützen. Der CoMo-Kit wird seit 1989 am Lehrstuhl für Expertensysteme der Universität Kaiserslautern unter der Leitung von Prof. Dr. M.M. Richter entwickelt.

1.1 Flexibles Workflow-Management

Workflow-Management ist eine Disziplin mit vielen Eltern. In Unternehmen bzw. den Wirtschafts- und Verwaltungswissenschaften wird versucht, durch explizite Formulierung von Arbeitsvorgängen diese besser zu verstehen und zu optimieren. Dabei ist der Computer als Werkzeug sehr gefragt, und er kann auch bei der Abarbeitung von Workflowmodellen durch die Koordination der beteiligten Personen bzw. ihrer Tätigkeiten wertvolle Hilfe leisten. Viele Informatikdisziplinen sind deshalb ebenfalls in die Untersuchung solcher Systeme involviert: Datenbanktechnologie muß sich auf die neuen Anforderungen einstellen, im Bereich Computer-Supported Cooperative Work (CSCW) werden Applikationen entwickelt, mit deren Hilfe teamorientierte Arbeitsabläufe durchgeführt werden können, es wird überprüft, wie Methoden der Künstlichen Intelligenz bei der Entwicklung von WFMS helfen können und im Bereich Software Engineering versucht man die Abläufe eines bestimmten Vorgangs, des Softwareentwicklungs- und -wartungsprozesses, zu erfassen. Da als gemeinsames Merkmal immer eine Prozeßbeschreibung zentraler Angelpunkt der Überlegungen ist, spricht man bei solchen Systemen in der Informatik auch von *Process-Centered Environments* (PCEs).
In klassischen Ansätzen des Workflow-Managements und relativ statischen Anwendungen werden im allgemeinen Planung und Abarbeitung eines Workflows deutlich voneinander getrennt. Dies erscheint auch sinnvoll, denn in Einsatzgebieten wie Geschäftsprozeßmodellierung und Produktionsplanung ist die Intention der Planung mit WFMS die *Automatisierung* von tendentiell routinemäßigen Arbeitsvorgängen mit dem Ziel erhöhter Verläßlichkeit und

Qualität. Bei gewissenhafter Modellierung dieser Prozesse sollten Änderungen dementsprechend die Ausnahme, und nicht die Regel, sein.
Abgesehen von der Tatsache, das „gewissenhafte Modellierung“ in der Praxis nur schwer zuzusichern ist, treten z.B. im Entwurfsbereich andere Intentionen in den Vordergrund. Vom Ziel der Automatisierung kann hier nur auf abstrakteren Ebenen (z.B. 1. Spezifikation; 2. Entwurf; 3. Prototyp; 4. Test) gesprochen werden; für die detailliertere Prozeßplanung müssen derartige WFMS in erster Linie flexibles Planen und Verwalten von Arbeitsvorgängen erlauben. Dabei treten gewisse Probleme häufiger als in klassischen WFM-Domänen auf, so daß die strikte Trennung von Plan und Ausführung nicht mehr wünschenswert erscheint:

- Oft kann erst während der Abarbeitung eines Workflows über die genaue Vorgehensweise bei der Bewältigung einer Teilaufgabe entschieden werden, da zur Entscheidungsfindung Daten benötigt werden, die erst wärend der Abarbeitung des Workflows anfallen;
- Es kann notwendig werden, früher gefällte Entscheidungen über die genaue Vorgehensweise zu revidieren, da sich die Vorraussetzungen, unter denen sie gefällt wurden, geändert haben;
- Entwurfsprozesse dauern z.T. erheblich länger als z.B. Workflows im Produktionsbereich. Daher ist eine Änderung von in der Abarbeitung befindlichen Workflows auch wesentlich wahrscheinlicher als in klassischen Workflow-Bereichen.

Da es sich hierbei um Planungsentscheidungen *während* der Abarbeitung eines Workflows handelt, wollen wir solche Entscheidungsprozesse als *dynamische* Umplanungsprozesse bezeichnen[1].

Dynamisches Umplanen im Entwurfsbereich

Die Modellierung von zielgerichteten Arbeitsvorgängen geschieht im Allgemeinen über die Methode *Divide and Conquer* (Teile und Herrsche). Im Entwurfsprozeß wird damit eine komplexe Entwurfsaufgabe in einfachere Teilaufgaben zerlegt, und auch das zu entwerfende Produkt sowie die Anforderungen die an dieses gestellt werden, werden zerlegt. Im Kontext dieser Zerlegungen entstehen zwischen den Prozessen (den Workflows) und den Produkten (d.h. den Datenstrukturen, die Prozeßergebnisse und Prozeßressourcen repräsentieren) verschiedene Abhängigkeiten:

- **Datenflußabhängigkeiten zwischen (Sub-)Workflows** entstehen dadurch, daß ein Workflow zu seiner Abarbeitung Produkte benötigt, die ein anderer Workflow erst erstellt haben muß.
- **Kontrollflußabhängigkeiten zwischen Workflows** werden explizit dann formuliert, wenn zwei Workflows trotz fehlender Datenflußabhängigkeit nur in einer ganz bestimmten zeitlichen Relation (z.B. Reihenfolge) abgearbeitet werden sollen. So muß z.B. vor der Nutzung eines Softwarepaketes dieses erst korrekt konfiguriert werden. Man kann jedoch wiederum nicht sagen, daß die Existenz eines Konfigurationsberichts ein Eingabeprodukt für den Nutzungsprozeß selber darstellt. Ein anderes Beispiel sind die einem hierarchischen Workflow-Ansatz innewohnenden Abhängigkeiten zwischen einem Workflow und seinen Subworkflows, z.B. daß der Subworkflow erst startet darf, wenn der Superworkflow bereits gestartet ist.

1. In anderen Publikationen redet man auch von „dynamischer Modellierung“ [WKM+95].

- **Begründungsbezogene Abhängigkeiten** entstehen durch Entscheidungen, deren Basis frühere Designentscheidungen sind, z.B. die Erstellung von bestimmten Workflow-Produkten oder die Auswahl bestimmter Designmethoden. Gerade diese Abhängigkeiten sind im Entwurfsbereich wichtig, um bei Revision früherer Entscheidungen die Folgen abschätzen zu können.

Für den Bearbeiter (Agenten) eines Workflows ist es wichtig, von allen für ihn relevanten Veränderungen bei Produkten und Prozessen, von denen er bzw. der von ihm bearbeitete Workflow abhängig ist (z.B. wenn frühere Designentscheidungen revidiert wurden) zu erfahren. Eines der Hauptziele im CoMo-Kit ist die Unterstützung flexiblen Workflow-Managements durch Verwaltung dieser Abhängigkeiten. In der CoMo-Kit-Abhängigkeitsverwaltung werden Abhängigkeiten zwischen getroffenen und anstehenden Dekompositions- (Welche Zerlegung wähle ich?) und Wertbelegungsentscheidungen (Welchen Wert weise ich einer Produktdatenstruktur zu?) verwaltet. Diese Verwaltung garantiert, daß ein Agent von einer Änderung der Produkte, auf denen seine Arbeit aufbaut, auch erfährt[1].
Bestimmte Problemzerlegungen sind von vorneherein spezifizierbar. Durch diese Spezifikationen (*Methoden* im CoMo-Kit) werden auch bestimmte Abhängigkeitsstrukturen determiniert. Hat sich eine Methode als unzureichend bezüglich der Erfüllung des Zieles ihres Superworkflows erwiesen, so können ihre Effekte zurückgesetzt und eine andere Methode ausgewählt werden. Diese Art von dynamischen Umplanungen wird mit Hilfe der Abhängigkeitsverwaltung im CoMo-Kit bereits ermöglicht.
In der Praxis kann es jedoch vorkommen, daß Problemzerlegungen nicht von vorneherein spezifizierbar sind, sondern ihre genaue Ausgestaltung erst im Laufe der Workflow-Abarbeitung festgelegt werden kann. Oder es können Modellierungsfehler bei der Problemzerlegung oder der Produktmodellierung vorgekommen sein. Dies Fälle sind besonders problematisch, da sich *während der Abarbeitung* die Abhängigkeiten im Workflow-Modell ändern. Praktische Folgen solcher Änderungen können z.B. auch der Wegfall von Teilaufgaben sein, deren Erledigung sich damit erübrigt, oder der Wegfall von Beziehungen, die z.B. die Auswahl einer bestimmten Zerlegung eines Teilproblems verhindert haben.
Die Verwaltung solcher die Abhängigkeiten in einem Workflow-Modell modifizierenden Ummodellierungen ist das zentrale Thema dieser Arbeit.

Anforderungen an ein Ummodellierungs-Management

Für den ausführenden Agenten ist wichtig, daß er auch bei Ummodellierungen zum frühestmöglichen Zeitpunkt von für ihn relevanten Veränderungen erfährt. Prinzipiell wünscht er auch eine hohe Verfügbarkeit des WFMS, so daß die technischen Aspekte der durch die Ummodellierung verursachten Revision der Abhängigkeitsstrukturen möglichst zügig und für den Benutzer transparent ablaufen sollten.
Für den Workflow-Designer ist es wichtig, daß er möglichst frei ist beim Design der Ummodellierungen, daß er Unterstützung erhält bezüglich der Erhaltung der Modellkonsistenz und daß ihm die (potentiell notwendige) Interaktion mit der in Abarbeitung befindlichen Instanz des Workflow-Modells möglichst einfach gemacht wird. Auch ihn interessiert die genaue Art der durch seine Ummodellierungen verursachten Modifikation der Abhängigkeitsstrukturen nicht, sie sollte möglichst weit automatisiert sein.
Für den Projekt-Manager eines Workflow-Modells ist es wichtig, daß nach der Ummodellierung möglichst viel der bereits erledigten Arbeit wiederverwendet werden kann. Diese Wiederverwendung sollte möglichst weit automatisiert und möglichst transparent geschehen.

1. Natürlich nur, sofern die entsprechenden Abhängigkeiten auch modelliert wurden.

Wir gehen auf diese Anforderungen zu Beginn unsere Betrachtungen über Ummodellierungen in Abschnitt 5.2 nochmals ein.

1.2 Aufgabenstellung der Diplomarbeit

Im Rahmen dieser Diplomarbeit soll die Verwaltung von dynamischen Ummodellierungsprozessen in WFMS sowohl theoretisch untersucht werden als auch eine Ummodellierungsmanagement-Komponente für den CoMo-Kit entwickelt werden.
Bei der theoretischen Erarbeitung eines Verwaltungsmodells müssen wir uns Eigenschaften von Ummodellierungsprozessen erarbeiten, die es erlauben, im Kontext der oben genannten Anforderungen eine Ummodellierung hinreichend genau zu beschreiben.
Für die Entwicklung einer Ummodellierungskomponente werden wir uns mit technischen Aspekten von Modelländerungen und ihrer Umsetzung in der Abwicklungskomponente des CoMo-Kit auseinandersetzen. Dazu gehören auch Mechanismen zur Vermeidung der Fortsetzung überflüssig gewordener Workflows und zur Wiederverwendung bereits erledigter Arbeitsschritte nach der Ummodellierung.
Wir werden diese Aufgabenstellung in Abschnitt 2.5 noch einmal speziell für den CoMo-Kit konkretisieren.

1.3 Aufbau der Dokumentation

Allgemein setzen wir voraus, daß der Leser mit der Problematik der Modellierung und Abwicklung von Workflow-Modellen vertraut ist.
In Abschnitt 2 führen wir kurz das Projekt CoMo-Kit und seine Begriffswelt ein. Dieser Abschnitt ist auch deshalb wichtig, weil wir (in Ermangelung einer allgemein anerkannten Terminologie für flexible WFMS) die eingeführte Begriffswelt auch bei der Entwicklung eines Konzeptes für eine Ummodellierungskomponente verwenden werden.
In Abschnitt 3 untersuchen wir allgemein die Begriffswelt von Umplanungsprozessen und die Möglichkeiten des Umgangs mit Ummodellierungen. In Abschnitt 4 beschreiben den Stand der Forschung auf den verschiedenen beteiligten Gebieten der Wissenschaft. In Abschnitt 5 erarbeiten wir ein Modell zur Verwaltung von Umplanungsprozessen im CoMo-Kit und seine Operationalisierung in der aktuellen CoMo-Kit-Implementierung.
Zum Abschluß geben wir eine Zusammenfassung unserer Ergebnisse und einen Ausblick auf mögliche Weiterentwicklungen im CoMo-Kit.

2 CoMo-Kit als flexibles WFMS

Im folgenden beschreiben wir CoMo-Kit. Nach einer Einordnung des CoMo-Kit in das Gebiet der PCEs (Abschnitt 2.1) beschreiben wir zunächst die Systemarchitektur (Abschnitt 2.2), dann gehen wir auf das CoMo-Kit-Modell und seine Begriffswelt ein (Abschnitt 2.3), gefolgt von einer einer Beschreibung der Planabarbeitung- und überwachung (Abschnitt 2.4). Am Ende dieses Abschnittes (Abschnitt 2.5) gehen wir nochmals auf die Problematik von Umplanungen im CoMo-Kit ein und beschreiben sowohl die bereits vor unserer Arbeit möglichen Umplanungen als auch eine Konkretisierung der Aufgabenstellung unter Berücksichtigung der besonderen Gegebenheiten im CoMo-Kit.
Abschnitt 2.3 ist auch deshalb wichtig, weil wir uns auch im weiteren Verlauf dieser Arbeit der CoMo-Kit-Terminologie bedienen werden. Um die Details des in spezifizierten Algorithmus nachvollziehen zu können, sollte man auch die Beschreibung der Abwicklung eines Prozessmodells (insbesondere die des Zustandsmodells) in Abschnitt 2.4 gelesen haben.

2.1 CoMo-Kit im Kontext von PCEs

Der Conceptual Modelling Kit ([Maur93], neuere Aspekte in [Maur96ab]) ist ein Werkzeug zur Erstellung und Abarbeitung von verteilten, kooperativen und wissensbasierten Prozeßmodellen. Hauptanwendungsgebiete waren bisher die städtische Bebauungsplanung (Überblick in [MP95], [MP96]) und - seit neuerem - Software Engineering (Kopplung von Planung und Ausführung in [DM96], Synthese mit MVP-E in [VDMM96] analysiert, fortgesetzt in der CoMo-Kit-Erweiterung MILOS, beschrieben in [DMMV96]) innerhalb des SFB 501 („Entwicklung großer Systeme mit generischen Methoden“). Die aktuellste Übersicht über CoMo-Kit findet man in [DMP96], auf die wir uns auch hier beziehen.
Ein PCE beinhaltet zwei wesentliche Funktionalitäten: die Möglichkeit, Arbeitsabläufe, deren Eingaben und Ergebnisse sowie die benötigten Ressourcen zu modellieren, und die Möglichkeit, diesen Ablaufplan in einem konkreten Projekt zu instanziieren und seine Abarbeitung zu unterstützen und zu überwachen. In CoMo-Kit sind diese Funktionalitäten separat in Systemkomponenten umgesetzt worden.

Ziele des Projektes
Ein typisches Problem bei PCEs ist die Verwaltung von Abhängigkeiten. Abhängigkeiten entstehen beispielsweise, wenn ein Workflow Ergebnisse eines anderen Workflows verwendet, oder durch Dekomposition in einem hierarchischen Workflow-Modell. Werden in diesen Bei-

spielen Entscheidungen zurückgezogen, die Basis des ergebnisproduzierenden Workflows bzw. des Superworkflows waren, so werden sie und ihre Ergebnisse invalidiert und somit werden auch dem ergebnisverwertenden Workflow bzw. dem Subworkflow die Arbeitsgrundlagen entzogen.
Ein wesentliches Ziel des CoMo-Kit ist es, solche Abhängigkeiten zum einen durch eine komfortable Modellierung einfach spezifizierbar zu machen, zum anderen auch die Abarbeitung von Prozessen mit Hilfe dieser Abhängigkeiten übersichtlicher zu gestalten.
CoMo-Kit unterscheidet sich deshalb von den meisten anderen PCE-Arten (WFMS, Process-Centered Software Engineering Environments) hauptsächlich in den erweiterten Möglichkeiten zur Verwaltung solcher Abhängigkeiten zwischen Arbeitsabläufen und ihren Ergebnissen. Dabei geht es im Wesentlichen um die Verwaltung von Planungsentscheidungen, die während der Abarbeitung von Arbeitsabläufen getroffen wurden und damit Grundlage des weiteren Vorgehens wurden. Werden diese Entscheidungen im weiteren Planverlauf widerrufen, so hat dies u.U. weitreichende Folgen für andere Teile des Ablaufplanes. Diese Abhängigkeiten zu verwalten, ist eines der Hauptziele im CoMo-Kit. Damit eignet sich CoMo-Kit besonders für Domänen, in denen ein hohes Maß an Flexibilität des Prozeßschedulings verlangt wird.

2.2 Die CoMo-Kit-Architektur

Die wesentlichen Komponenten des CoMo-Kits sind eine Modellierungskomponente, der *Modeler*, in der sämtliche Aspekte des Prozeßmodells manipuliert und gespeichert werden können, und der *Scheduler* ([DMP95], [Dell95]), der den Ablauf eines Prozeßmodells steuert und überwacht (siehe Abb. 1).

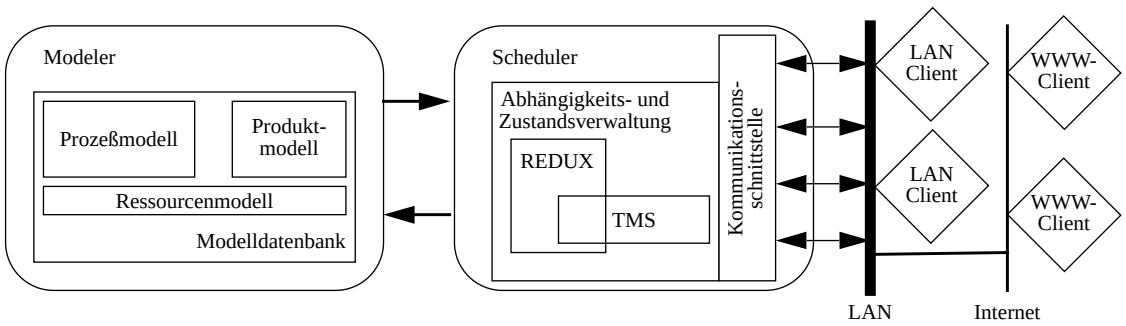


Abb. 1: Die CoMo-Kit-Architektur

Der Scheduler verwaltet dabei die für die Ablaufplanung zwischen den Modellkomponenten entstehenden Abhängigkeiten mit Hilfe der Planungskomponente REDUX ([Petr91], kurz auch in [Dell95]), die eine Verwaltung von voneinander abhängigen Entscheidungen und Zuweisungen inklusive deren Rückzug mit Hilfe eines *Truth Maintenance Systems* (TMS; [Doyl79]) erlaubt. Auch das Zustandsmodell des Scheduling ist auf der Basis des TMS realisiert ([DMP95]).
Für den Scheduler existiert neben der normalen grafischen Benutzeroberfläche auch ein WWW-Interface. Alle Komponenten sind in Smalltalk implementiert und benutzen die Objektorientierte Datenbank Gemstone zur Datenspeicherung.

Wir werden in dieser Arbeit sowohl den Modeler um bestimmte Ausdrucksmöglichkeiten beim Editieren der Modelle erweitern als auch den Scheduler um eine Komponente zu Transformation der Abhängigkeiten bereichern.

2.3 Modellierung von Prozessen in CoMo-Kit

In CoMo-Kit werden drei Modelle zur vollständigen Beschreibung von Arbeitsabläufen benutzt. Ein Prozeßmodell, mit dem spezifiziert wird, wie die Arbeitsabläufe aussehen; ein Produktmodell, in dem die Datenstrukturen, die bei der Abarbeitung des Ablaufplanes Verwendung finden, definiert werden und ein Ressourcenmodell, in dem die Ressourcen beschrieben werden können, die bei der Abarbeitung des Ablaufplanes zum Einsatz kommen (Dies sind im wesentlichen die menschlichen und maschinellen Agenten, die letztendlich die Prozesse abarbeiten).
CoMo-Kit übernimmt die Überwachung der aus diesen Modellen ableitbaren Abhängigkeiten während der Planausführung. Darüber hinaus mag es aber noch wünschenswert sein, solche Abhängigkeiten „von Hand“ in das Abarbeitungsmodell einzufügen oder zu löschen oder „externe“ textuelle Begründungen einzuführen. Dies wird im CoMo-Kit durch die Verwendung von *Design Rationales* möglich, auf die wir erst in Abschnitt 2.4 eingehen, da ihre Spezifikation eher in die Prozeßabarbeitung fällt als in die Prozeßmodellierung.

2.3.1 Das Prozeßmodell

CoMo-Kit verwendet ein durch Dekomposition (*Divide and Conquer*) strukturiertes hierarchisches Prozeßmodell ([Schm94]). Für jeden Prozeßtyp muß festgelegt werden, *was* gemacht werden soll, *wie* es gemacht werden soll, *wann* es gemacht werden soll und *wer* den Prozeß ausführen bzw. überwachen soll.
Hauptmerkmal eines Prozeßtyps (Tabelle 1) ist seine *Aufgabe* bzw. sein Ziel, in dem beschrieben wird, was gemacht werden soll. Zur Erfüllung dieser Aufgabe stehen ein oder mehrere diesem Prozeß zugeordnete *Methoden* (Tabelle 2) zur Verfügung, mit deren Hilfe spezifiziert wird, wie das Prozeßziel erreicht werden soll. Im Falle einer *atomaren Methode* kann das Ziel in einem Schritt erreicht werden (d.h. die Abarbeitung besteht lediglich aus der Erstellung des Prozeßergebnisses); in der Methode wird das Vorgehen als *Prozeßskript* (mit Handlungsanweisungen für menschliche Agenten) oder als *Prozeßprogramm* (mit formalen Vorgehensanweisungen für maschinelle Agenten) beschrieben. Bei *komplexen Methoden* wird das Prozeßziel in Teilziele bzw. der Prozeß in entsprechende Subprozesse zerlegt (Dekomposition).
Ein Prozeß benötigt Informationen bzw. Daten und erzeugt Informationen bzw. Daten (als Prozeßergebnis), die wir allgemein *Produkte* nennen und deren Datenstrukturen im Produktmodell (siehe Abschnitt 2.3.2) festgelegt werden. Konsumierte Informationen nennen wir die *Inputprodukte*, produzierte Informationen die *Outputprodukte* eines Prozesses. Als Grundprinzip des Zusammenhanges zwischen Inputs und Outputs nehmen wir an, daß es eine kausale Abhängigkeit der Outputs eines Prozesses von seinen Inputs gibt.
Der durch konsumierte und produzierte Produkte entstehende Datenfluß wird bei einer komplexen Methode für den jeweiligen Prozeß gespeichert. Diese Datenflußbeschreibung bildet zusammen mit der Dekomposition das wesentliche Merkmal einer Methode.

Attribut	Bedeutung
Aufgabe	Informelle Beschreibung der vom Prozeß zu lösenden Aufgabe
Parameter: Planning Input	Input, der zur Planung des Prozesses benötigt wird
Parameter: Execution Input	Input, der zur Abarbeitung des Prozesses benötigt wird
Parameter: Optional Input	nicht benötigter, aber möglicherweise hilfreicher Input
Parameter: Optional Output	optionaler Output
Parameter: Required Output	Output, welcher erstellt werden muß, um Aufgabe zu lösen
Context Information	Referenzen auf Informationen, die sich während der Prozeßabarbeitung nicht ändern. Beispiel: Dokumentationsrichtlinien bei der Softwareerstellung
Precondition	Formale, über Prozeß- und Produktparameter und -attribute sowie temporale Prädikate definierte Bedingung, die zum Start der Prozeßabarbeitung erfüllt sein muß. Beispiel: Anforderungen an Wertbelegungen von Inputs
Invariant	Formale, über Prozeß- und Produktparameter und -attribute sowie temporale Prädikate definierte Bedingung, die während der Prozeßabarbeitung immer erfüllt sein muß. Beispiel: Zeitlimit für die Prozeßabarbeitung
Postcondition	Formale, über Prozeß- und Produktparameter und -attribute sowie temporale Prädikate definierte Bedingung, die bei Abschluß der Prozeßabarbeitung immer erfüllt sein muß (Wird als Teil der Methoden-Postcondition überprüft). Beispiel: Qualitätsanforderungen an die Outputs
Required Competences	Kompetenzen, die ein Agent, der diesen Prozeß bearbeiten soll, mitbringen muß
Methods	Methoden, die zur Abarbeitung dieses Prozesses benutzt werden können
weitere	*Es können noch weitere Attribute, z.B. zum Spezifizieren der Prozessdauer, definiert werden.*

Tabelle 1: Prozeßeigenschaften im CoMo-Kit

Der Datenfluß beeinflußt über die Planning-Inputs der Prozesstypen auch den Zeitpunkt der Abarbeitung eines Prozesses, da erst bei Vorliegen aller Planning-Inputs mit der Prozeß startbar ist. Darüberhinaus können Kontrollflußspezifikationen über die *Precondition* eines Prozesses spezifiziert werden, die erfüllt sein muß, um einen Prozeß starten zu können. Dort sind sowohl zeitabhängige Prädikate als auch vom Wert eines Inputs abhängige Prädikate einsetzbar. Weitere Bedingungskonstrukte sind *Invarianten*, mit denen einem Prozeß für die ganze Dauer seiner Laufzeit bestimmte Umweltbedingungen zugesichert werden können[1] und *Postconditions*, die z.B. Eigenschaften einzelner Outputs oder Constraints über mehrere Outputs zusichern können[2].

1. Insbesondere müssen in der Precondition spezifizierte Bedingungen nur zum Zeitpunkt des Starts der eigentlichen Methodenabarbeitung erfüllt sein. Soll etwas über die ganze Dauer eines Prozesses zugesichert werden, so muß dies als Invariante spezifiziert werden.
2. Es sei darauf hingewiesen, daß Outputs für nachfolgende Prozesse bereits sichtbar werden, obwohl die Postcondition (noch) nicht erfüllt ist. Natürlich kann es dann passieren, daß sich bereits in der Weiterverarbeitung befindliche Outputs nochmals zurückgenommen werden.

Attribut	Bedeutung
Process Type	Prozeßtyp, zu dem die Methode gehört
Parameter: Execution Input	Input, der zur Abarbeitung der Methode benötigt wird
Parameter: Optional Input	nicht benötigter, aber möglicherweise hilfreicher Input
Parameter: Output	Output, welcher von der Methode erstellt wird
Precondition	Formale, über Prozeß- und Produktparameter und -attribute sowie temporale Prädikate definierte Bedingung, die zum Start der Methodenabarbeitung erfüllt sein muß.
Invariant	Formale, über Prozeß- und Produktparameter und -attribute sowie temporale Prädikate definierte Bedingung, die während der Methodenabarbeitung immer erfüllt sein muß.
Postcondition	Formale, über Prozeß- und Produktparameter und -attribute sowie temporale Prädikate definierte Bedingung, die bei Abschluß der Methodenabarbeitung erfüllt sein muß (Enthält Postcondition des Prozesses).
Required Competences	Kompetenzen, die ein Agent, der diesen Methode benutzen soll, mitbringen muß
Process Script	(nur bei atomaren, von menschlichen Agenten zu bearbeitenden Methoden:) Beschreibung der Vorgehensweise bei der Erstellung der Outputs
Process Program	(nur bei atomaren, von maschinellen Agenten zu bearbeitenden Methoden:) Formale Beschreibung der Vorgehensweise bei der Erstellung der Outputs
Decomposition	Subprozesse, in die der dieser Methode zugeordnete Prozeß zerlegt wird
Product Flow Graph	Datenfluß der Input- und Outputprodukte der Subprozesse. Der Graph besteht im wesentlichen aus den Subprozessen, den Formalen Parametern der Subprozesse und der Relationen zwischen ihnen.

Tabelle 2: Methodeneigenschaften im CoMo-Kit

Die von einem Prozeß benötigten Ressourcen sind *Agenten* (menschliche oder maschinelle), die einen Prozeß bzw. eine Methode abarbeiten und *Tools*, die bei der Abarbeitung benötigt werden. Welche Agenten einen Prozeßtyp abarbeiten können, wird durch *Kompetenzen* spezifiziert, die ein Prozeß von seinen Bearbeitern fordert. Die verwendeten Tools werden bei den Methoden spezifiziert.
Methoden verfügen über ähnliche Attribute wie Prozesse. Bei den Bedingungskonstrukten Invariante und Postcondition sowie bei den Kompetenzen handelt es sich um methodenspezifische Spezialisierungen der entsprechenden Prozeßattribute, bei den Parametern um Teilmengen der Prozeßparameter. Die genaue Art und Weise, in der eine Methode einen Prozeß in Subprozesse zerlegt, wird in den Attributen *Decomposition* (Welche Subprozesse?) und *Product Flow Graph* (Durch welche Datenflußabhängigkeiten hängen diese zusammen?) festgehalten.
Durch die Dekomposition bildet sich bei der Spezifikation des Prozeßmodells ein UND-ODER-Baum aus Methoden und Prozeßtypen. Bei Methoden müssen alle Subprozesse abgearbeitet werden (AND-Ebene), um die Methode als bearbeitet zu betrachten, bei Prozessen muß eine Methode ordnungsgemäß abgearbeitet werden (OR-Ebene), damit einen Prozeß als erfolgreich gilt.

Dies sind die wesentlichen Eigenschaften des Prozeßmodells. Tabelle 1 und Tabelle 2 geben Auskunft auch über weitere Eigenschaften von Methoden und Prozeßtypen. Es sei noch darauf hingewiesen, daß wir im Weiteren (insbesondere in Abschnitt 3) des öfteren die Bezeichnungen „Aufgabe“ oder „Task“ für den Prozeßtyp wählen werden, um Mehrdeutigkeiten um den Begriff „Prozeß“ zu vermeiden, der in unserer Domäne an vielen Stellen auftreten kann.

2.3.2 Das Produkt-Datenmodell

Das Produkt-Datenmodell folgt weitgehend einem objektorientierten Ansatz. Neben den üblichen Abstraktionsbeziehungen Generalisierung/Spezialisierung ist auch die Abstraktionsbeziehung Aggregation modellierbar. Ein Produktyp kann entweder ein Grundtyp (wie `Real`, `String`, `Text`, etc.) oder komplex sein (wir reden dann auch von einer Konzeptbeschreibung). Komplexe Produkttypen erlauben die Spezifikation von Slots, die eine part-of-Beziehung spezifizieren, und Attributen, die lediglich beschreibenden Charakter haben. Als Slottypen stehen sowohl die Grundtypen zur Verfügung als auch andere Produkttypen, so daß auf diese Weise auch komplexe Produkthierarchien definiert werden können. Jeder in einem Prozeßmodell vorkommende formale Parameter (d.h. alle Inputs und Outputs) hat einen Produkttyp. Die Dekomposition einer komplexen Produkthierarchie muß konsistent mit der Aufgabendekomposition des Prozeßmodells, die diese Produkte erstellt, sein.

2.3.3 Das Ressourcenmodell

Als Ressourcen werden im CoMo-Kit Agenten und Tools betrachtet. Bei Agenten werden neben beschreibenden Attributen im wesentlichen ihre Kompetenzen spezifiziert. Wir unterscheiden für menschliche Agenten zwischen Qualifikationen, die die Fähigkeiten der Agenten beschreiben, Rollen, die die Funktionen von Agenten beschreiben und organisationsbezogene Kompetenzen, die ihre Rechte aufgrund ihrer Stellung in der Organisation beschreiben.
Die Beschreibung von Tools ist noch nicht weiter ausgearbeitet, sie erschöpft sich zur Zeit in der Spezifikation der Aufrufsequenz für das betreffende Tool.

2.4 Die Ausführung von Prozessen

Zunächst wollen wir die Ausführung von Prozessen in CoMo-Kit beschreiben, bevor wir auf das Zustandsmodell, welches die Prozessabarbeitung operationalisiert, seine Verwaltung und die Verwaltung der Abhängigkeiten eingehen.

Das Szenario aus Anwendersicht

Wir gehen von einem Prozeß aus, der als zu bearbeitend („valid“) gekennzeichnet ist. Wenn alle zur Planung benötigten Inputs vorliegen und die Precondition des Prozesses zu `True` evaluiert, wird der Prozeß den Agenten, die aufgrund ihrer Kompetenzen und der Delegationsentscheidung (s.u.) im eventuell vorhandenen Superprozeß dieses Prozesses zur Bearbeitung dieses Prozesses legitimiert sind, als bearbeitbar gemeldet. Akzeptiert einer dieser Agenten den Prozeß, so tritt dieser in die Planungsphase, in der dieser Agent (wir nennen ihn den *Pla-*

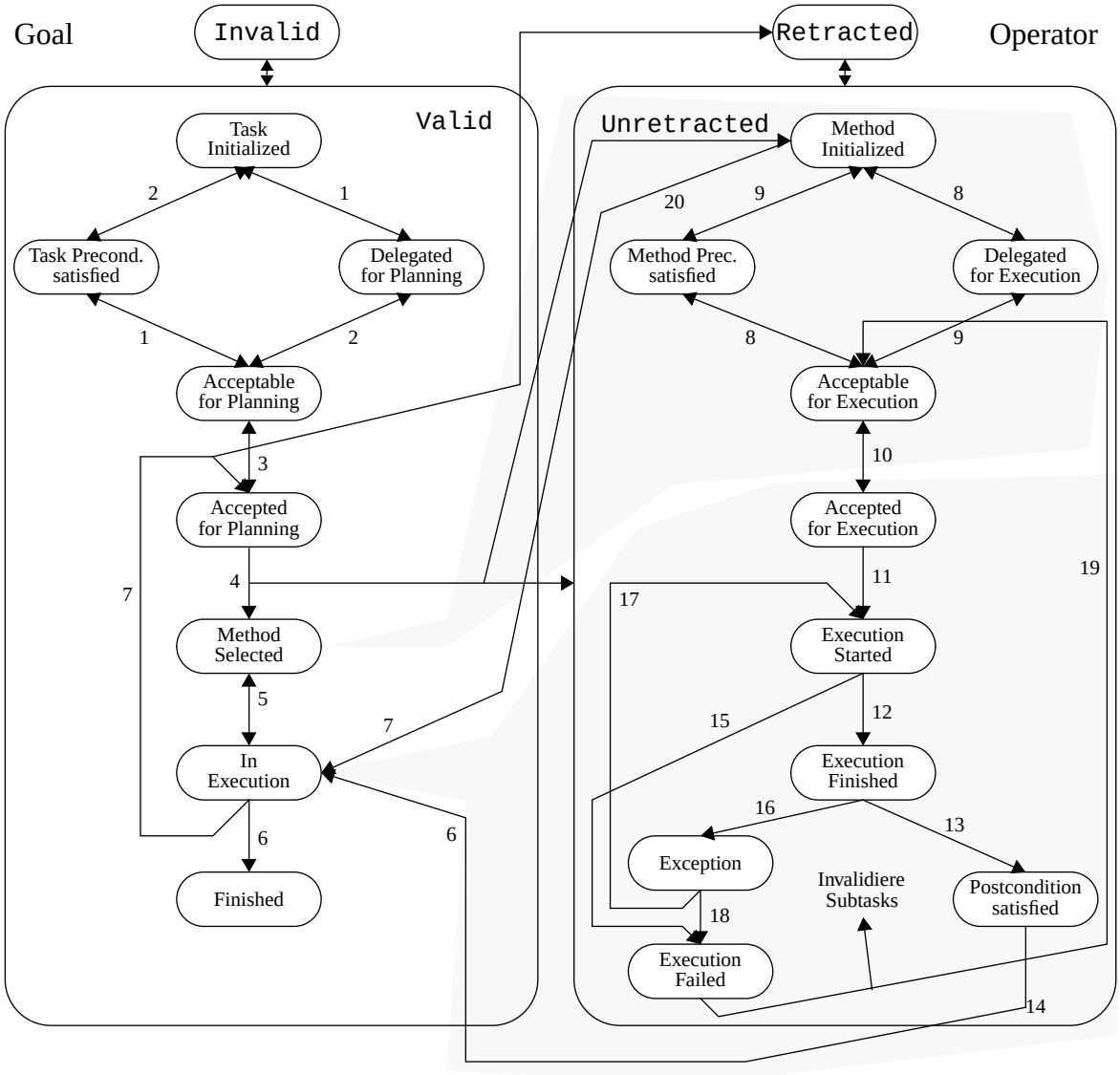
ner dieses Prozesses) eine dem Prozeß zugeordnete Methode auswählen muß, nach der dieser Prozeß dann bearbeitet werden soll. Hat der Agent eine Methode ausgewählt (*Dekompositionsentscheidung*), so tritt der Prozeß in die Ausführungsphase. Die ausgewählte Methode wird, sobald alle für die Ausführung notwendigen Inputs vorliegen, die Precondition der Methode erfüllt ist und der Planer mögliche Bearbeiter dieser Methode spezifiziert hat (*Delegationsentscheidung*), allen dafür kompetenten Agenten als ausführbar gemeldet. Übernimmt ein Agent die Ausführung (wir nennen ihn den *Bearbeiter* des Prozesses), so werden die in der Methode vorkommenden Subprozesse als zu bearbeitend gekennzeichnet und der Bearbeiter kann mögliche Planer für die Subprozesse spezifizieren (*Delegationsentscheidung*). Danach werden diese Subprozesse wie beschrieben abgearbeitet. Liegen alle Outputprodukte der Subprozesse vor, und sind alle ihre Postconditions erfüllt, so wird geprüft, ob die Methoden-Postcondition erfüllt ist. Ist dies nicht der Fall, kann der Berarbeiter entscheiden, ob die Methode nochmals abgearbeitet werden soll oder der Planer aufgefordert werden soll, möglicherweise eine andere Methode auszuwählen. Ist die Methoden-Postcondition erfüllt, so kann (nach Überprüfung der Prozeß-Postcondition) der Prozeß als abgearbeitet gelten und die Outputs werden an eventuelle übergeordnete Prozesse weitergegeben.

2.4.1 Das Zustandsmodell der Prozeßabarbeitung

Das Zustandsmodell im CoMo-Kit zeichnet im wesentlichen die oben beschriebene Abarbeitung nach. Über das oben beschriebene hinaus haben Planer zu jedem Zeitpunkt die Möglichkeit, ihre Dekompositionsentscheidung (d.h. die Methodenauswahl) zurückzunehmen und sich anders zu entscheiden. Da damit auch die an diesen Methoden hängenden Subprozesse (und die bei deren Abarbeitung getroffenen Entscheidungen) betroffen sind, wird eine Abhängigkeitsverwaltung notwendig, die im nächsten Abschnitt beschrieben wird.
Im Zustandsmodell werden Prozesse durch *Goals* und Methoden durch *Operatoren* repräsentiert. Goals haben die Zustände *valid* und *invalid*; die Operatoren die Zustände *retracted* und *unretracted*. In beiden Fällen beschreiben die Zustände, ob die entsprechenden Prozesse/ Methoden gerade Teil des Planes sind (valid/unretracted) oder nicht (invalid/retracted). Die Zustände unretracted und valid haben Subzustände, die den Bearbeitungszustand des zugeordneten Prozesses bzw. der Methode beschreiben. Die Zustände und die Zustandsübergänge sind in Abb. 2 aufgezeigt. Das Zustandsmodell atomarer Methoden unterscheidet sich vom obigen Operatorzustandsmodell nur durch den Wegfall der Delegierungsentscheidung.
Auch Produktvariablen haben Zustände, die angeben, ob eine Variable noch nicht belegt wurde (*Unassigned*), nicht belegt wird (*Irrelevant*) oder einen gültigen Wert hat (*Assigned*).
Nähere Informationen (insbesondere eine verfeinerte Sicht auf die Zusammenhänge zwischen den Zustandsmodellen von Goals und Operatoren) findet man in [DMP96] und detailgenauer in [Chri96].

2.4.2 Der Einsatz des REDUX-Systems, des TMS und der Design Rationales

REDUX ist ein Planungstool, dessen wesentliche Aufgabe darin besteht, die in einem Plan getroffenen Entscheidungen zu protokollieren und ein Abhängigkeitsnetzwerk aufzubauen, mit dessen Hilfe die Folgen des Rückzuges einer Entscheidung für den Plan verwaltet werden können. REDUX benutzt zum Aufbau dieses Netzwerkes ein TMS, daß in CoMo-Kit auch über



Legende der Zustandsübergänge in der Task:
1. Delegationsentscheidung des Bearbeiters der Supertask
2. Vorbedingungen erfüllt
3. Agent (Planer) erklärt sich für Task zuständig
4. Planer wählt Methode aus (Dekompositionsentscheidung)
5. Methode von einem Agenten (Bearbeiter) akzeptiert (s. 10)
6. Methode erfolgreich beendet
7. Methode fehlgeschlagen

Doppelpfeile beinhalten auch die inversen Ereignisse.

Legende der Zustandsübergänge im Operator:
8. Delegationsentscheidung des Planers
9. Vorbedingung erfüllt
10. Agent (Bearbeiter) erklärt sich für Methode zuständig (s. 5)
11. Beginn der Methodenabarbeitung
12. Alle Subgoals ordnungsgemäß bearbeitet
13. Nachbedingung erfüllt
14. Methode erfolgreich beendet
15. Fehler während der Methodenabarbeitung
16. Nachbedingung nicht erfüllt
17. Weiteren Bearbeitungsversuch starten
18. Keinen weiteren Bearbeitungsversuch starten
19. Abarbeitung durch anderen Bearbeiter / Methodenfehlschlag
20. Methodenabarbeitung endgültig fehlgeschlagen

Abb. 2: Das CoMo-Kit-Zustandsmodell

REDUX hinaus Verwendung findet. Die folgende Beschreibung konzentriert sich auf das wesentliche, für weitere Informationen sei auf [Petr91] und [Dell95] verwiesen.

REDUX-Grundlagen

Die grundlegenden Einheiten des REDUX-Modells sind Goal, Operator, Decision und Assignment. *Goal* und *Operator* repräsentieren (im Sinne des vorangehenden Abschnittes) Prozesse bzw. Methoden des CoMo-Kit. Eine *Decision* steht für eine vom Planer getroffene Dekompo-

sitionsentscheidung (d.h. die Auswahl einer Methode), *Assignments* für Wertzuweisungen an verwendete Variablen (z.B. formale Parameter des CoMo-Kit-Modells). Für Goals verwaltet REDUX ein als TMS-Netzwerk realisiertes Zustandsmodell, welches auf niedriger Ebene Aussagen über die Gültigkeit (*valid*/ *invalid*) und den Abarbeitungszustand (*reduced* - Dekompositionsentscheidung getroffen/ *unreduced* - Dekompositionsentscheidung noch nicht getroffen/ *blocked* - Dekomposition nicht möglich/ *satisfied* - Goal bearbeitet) eines Goals erlaubt und die Basiskomponente des CoMo-Kit-Zustandsmodells ist.
Wird ein Goal durch eine Decision (Dekompositionsentscheidung) für einen Operator (Methode) reduziert, so werden im im TMS modellierten Abhängigkeitsnetzwerk sowohl die Subgoals des Operators als auch die durch seine Anwendung verursachten Wertzuweisungen bzw. Assignments (d.h. die Outputs der Methode) als von der Decision abhängig eingetragen. Wird die Dekompositionsentscheidung für einen Operator irgendwann einmal wiederrufen (Entscheidungsrückzug), so werden auch die abhängigen Goals und Assignments als ungültig markiert. Wir unterscheiden den Rückzug einer Entscheidung (*retracted-decision*) und den Rückzug eines Operators (*rejected-decision*); letzterer umfaßt natürlich auch den Rückzug der Entscheidung für diesen Operator, dieser kann dann allerdings nicht nochmals ausgewählt werden. Wird nach einem Entscheidungsrückzug derselbe Operator nochmals ausgewählt, so werden damit auch die von ihm abhängigen Subgoals und Assignments wieder validiert.

CoMo-Kit-Erweiterungen zu REDUX

Diese Zusammenhänge werden durch ein Zustandsmodell für Decisions operationalisiert, welches die Zustände *SupportedValidity* (d.h. Validität durch Design Rationales gestützt), *UnsupportedValidity*, *SupportedRetraction* (d.h. Entscheidungsrückzug durch Design Rationales gestützt) und *UnsupportedRetraction* umfasst. Ein Operatorrückzug führt in den Zustand *SupportedRetraction*, ein Entscheidungsrückzug in den Zustand *UnsupportedRetraction*.
Im CoMo-Kit werden zudem auf der Basis des von REDUX verwendeten TMS-Netzwerks auch Abhängigkeiten bezüglich der Ausführbarkeit eines Goals/Operators verwaltet. Insbesondere Datenflußabhängigkeiten (z.B. müssen bestimmte, Prozeßinputs repräsentierende Assignments bei Prozeßbeginn verfügbar sein) und Abhängigkeiten bezüglich Delegierungsentscheidungen werden einbezogen. Darüberhinaus können weitere Abhängigkeiten zwischen Goals, Decisions und Assignments über das Design-Rationale-Konzept spezifiziert werden.

Das Design-Rationale-Konzept

Die den Design Rationales zugrundeliegende Idee ist es, Begründungen für Entscheidungen und Sachverhalte festzuhalten, die zum einen diese dokumentieren und zum anderen Änderbarkeit und Verfolgbarkeit von Entscheidungsverläufen über eine Verwaltung der Abhängigkeiten erlauben. Fallen z.B. die Begründungen für einen Operatorrückzug weg, so kann der betroffene Operator so auch wieder als zulässig erkannt werden.
Im CoMo-Kit können Assignments, Decisions und textuelle Repräsentationen als Begründungen für Entscheidungen bzw. deren Rückzug spezifiziert werden. In [DKM96] wird am Beispiel der Software-Engineering-Domäne neben diesem Konzept auch der Wegfall von Abhängigkeiten behandelt.

Wir werden bei der Beschreibung der Operationalisierung der in dieser Arbeit erstellten Konzepte auch auf weitere Details der CoMo-Kit-Implementierung eingehen.

2.5 Dynamisches Umplanen im CoMo-Kit

Nach der Beschreibung der bereits in der aktuellen Version möglichen Umplanungen konkretisieren wir unsere Aufgabenstellung.

Umplanungsmöglichkeiten im CoMo-Kit

Auch in der aktuellen CoMo-Kit-Version sind bereits Umplanungen während des Ablaufes eines Prozesses möglich. Neben dem dynamischen Generieren von Agentenbindungen über das Matchen von Kompetenzen, dem Rückzug von getroffenen (Dekompositions-)Entscheidungen und dem anschließenden Treffen neuer Entscheidungen können als global sichtbare Modellmodifikationen Methoden zu einem Prozeßtyp hinzugefügt oder gelöscht werden. Diese Modifikationen sind dann zwar sofort bei anstehenden Entscheidungen sichtbar, beeinflussen jedoch nicht bereits getroffene Entscheidungen. Auch gibt es keine Möglichkeit, in einer Methode bearbeitete Aufgaben in die andere, neue Methode zu übernehmen, da im CoMo-Kit davon ausgegangen wird, daß zwei verschiedene Methoden eine Aufgabe in disjunkte Mengen von Teilaufgaben zerlegen.
Das Löschen einer Aufgabe während ihrer Abarbeitung hat ebenfalls keine direkten Auswirkungen, da ihre Löschung in der Modellierungskomponente der Ausführungskomponente nicht mitgeteilt wird.
Mit Hilfe des Konzeptes der Design Rationales können zudem bestimmte Ummodellierungen des Datenflusses realisiert werden. Wird z.B. eine Decision über ein Design Rationale mit einer Wertzuweisung (einem Assignment) begründet, so entspricht dies genau der Semantik des Hinzufügens eines Planning-Inputs (repräsentiert durch das Assignment) zu der Task, zu der die Decision gehört. Problematisch hierbei ist natürlich, daß das Design Rationale an und für sich keine solche Semantik repräsentieren kann, und daß dieser zusätzliche Input nicht im Workflow-Modell sichtbar ist.

Konkretisierte Aufgabenstellung: Ein Ummodellierungsmanagement für den CoMo-Kit

Das oben beschriebe Löschen von Methoden und Aufgaben stellt schon eine erste abhängigkeitsmodifizierende Umplanung im Sinne der in der Einleitung beschriebenen Problematik dar. Die Abhängigkeitsverwaltung (das TMS-Netzwerk im Scheduler) wird jedoch nicht sofort von der Modifikation informiert, so daß die Abhängigkeiten dort zunächst unverändert bleiben. Nur wenn die betroffene Methode/Aufgabe noch nicht Teil der Ablaufplanung des Schedulers war, wird die Änderung (im Falle der Auswahl der betreffenden Methode) in der Abwicklungskomponente sichtbar.
Es ist also nötig, eine *koordinierte Interaktion* zwischen ummodelliertem Modell und dem Scheduler, in dem das Abhängigkeitsnetzwerk der Workflowmodell-Instanz verwaltet wird, zu erlauben.
Ist eine von einer Ummodellierung betroffene Aufgabe/Methode bereits Teil des aktuellen Ablaufplanes, so ändert sich durch die Ummodellierung eventuell auch dieses Abhängigkeitsnetzwerk, und damit indirkt auch die Zustände von Prozessen, Methoden, Decisions und Variablen. Die Anforderung der Automatisierung und der Transparenz solcher technischen Vorgänge für Benutzer des CoMo-Kit verlangen hier nach einem *Transformationsalgorithmus*, mit dessen Hilfe das TMS-Abhängigkeitsnetzwerk im Scheduler vor der Ummodellierung in ein TMS-Abhängigkeitsnetzwerk umgewandelt wird, in welchem die Ummodellierungen berücksichtigt wurden.

Um die Bearbeiter von überflüssig werdenden Prozessen frühzeitig von der Wertlosigkeit der Arbeitsfortsetzung zu informieren (Aspekt der *Arbeitsersparnis*), muß es dem Prozessdesigner möglich sein, bereits während der Modellierung im Modeler auf die Planabarbeitung im Scheduler Einfluß nehmen zu können.
Um die *Wiederverwendung bereits geleisteter Arbeit* zu ermöglichen, müssen wir uns überlegen, anhand welche Eigenschaften von Prozessen und Methoden es uns ermöglichen, über die Wiederverwendbarkeit von Produkten und Prozessen in einer konkreten Ummodellierungssituation zu entscheiden.

3 Umplanung und Ummodellierung: Begriffsbildung

Zunächst ist es wichtig, festzustellen, was wir unter einem Umplanungs-/Ummodellierungsprozeß verstehen, da in der Literatur keine einheitliche Terminologie zu erkennen ist. Deshalb geben wir in eine Definition für Umplanungen und Ummodellierungen an und erläutern sie an einigen Beispielen. An einem weiteren Beispiel wollen wir untersuchen, welche Möglichkeiten ein Benutzer eigentlich prinzipiell hat, um auf die Notwendigkeit einer Ummodellierung zu reagieren. Aus dem Bereich Software Engineering führen wir dann einige Klassifikationen von Ummodellierungen (dort unter dem Stichwort „Prozessevolution“ betrachtet) an, die dabei behilflich sind, die Natur einer Ummodellierung zu erfassen. Wir werden später auf diese Eigenschaften und ihre Verwendbarkeit für die von uns betrachtete Problematik zurückkommen.

3.1 Der Umplanungsprozess: Definition und Beispiele

Eine genauere Definition des Begriffes „Umplanungsprozeß“ ist vor allen Dingen deshalb nötig, weil es in der Literatur zwar ein allgemeines Bild der beschriebenen Problematik gibt, jedoch keine feste Begrifflichkeit. Wir konkretisieren damit gleichzeitig den etwas laxen Umgang mit dem Begriff Umplanung in der Einleitung.

> Ein *Umplanungsprozeß* in einem WFMS ist ein Prozeß, bei dem eine früher getroffene Entscheidung wiederrufen und evtl. eine neue Entscheidung gefällt wird. Wir nennen ihn *dynamisch*, wenn es sich um Veränderungen handelt, die während der Abarbeitung einer Instanz des Workflowmodells auftreten und für diese Instanz Geltung haben. Wird bei einem Umplanungsprozeß das im Workflowmodell gespeicherte Wissen bezüglich Agenten, Prozessen oder Produkten verändert, reden wir von einer *Ummodellierung*.

Bei Umplanungen, die nicht Ummodellierungen sind, handelt es sich entweder um Reallokationen (d.h. Neuverteilung und -zuweisung) von Ressourcen oder um Replanning, d.h. eine bestimmte Aufgabe konnte nicht auf die zunächst ausgewählte Methode gelöst werden, so daß eine andere Methode zum Einsatz kommt. Diese Umplanungsprozesse sind Bestandteil des CoMo-Kit-Schedulingkonzeptes und nicht Gegenstand dieser Arbeit.
Nicht-dynamische Umplanungsprozesse verlangen letztendlich nur nach Erhaltung der Modellkonsistenz; diese Aspekte sind nicht Gegenstand unserer Betrachtungen. Wir betrachten aus den in der Einleitung dargelegten Gründen nur dynamische Ummodellierungsprozesse.

Zur Veranschaulichung hier einige Beispiele zu unserer Terminologie für Umplanungsprozesse:

(1) (Gegenbeispiel!) Die Erweiterung des WFMS um die Möglichkeit, Constraints über Produkten zu definieren, ist kein Umplanungsprozeß in unserem Sinne, sondern eine tiefergreifende Metamodelländerung, deren Auswirkungen nicht Gegenstand dieser Arbeit sind.

(2) Der für einen Workflow zuständige Agent fährt längere Zeit in Urlaub. Die Verantwortung für diesen Workflow wird vorübergehend auf einen anderen Agenten übertragen. Hierbei handelt es sich zwar um eine Umplanung (Reallokation von Ressourcen), *nicht* aber um eine Ummodellierung.

(3) Der für einen Workflow zuständige Agent wird von diesem Projekt abgezogen und einem anderen zugeteilt. Auch hierbei handelt es sich nur um eine Reallokation von Ressourcen.

(4) Der für einen Workflow zuständige Agent verläßt die Firma. Die Verantwortung für diesen Workflow wird endgültig auf einen anderen Agenten übertragen. Hierbei handelt es sich ebenfalls nur um eine Reallokation von Ressourcen.

(5) Der für einen Workflow zuständige Agent wird wegen Inkompetenz entlassen. Die Verantwortung für diesen Workflow wird endgültig auf einen anderen Agenten übertragen. Hierbei handelt es sich nicht nur um eine Reallokation von Ressourcen, vielmehr schlägt sich die Inkompetenz in der Löschung des Agenten bei den Aufgaben und Methoden, für die er inkompetent geworden ist, nieder. Eventuell müssen die Entscheidungen des inkompetenten Agenten (z.B. bezüglich Methodenauswahl) überprüft und ggfs. revidiert werden.

(6) Der Rolle, aufgrund derer ein Agent einem Workflow zugeordnet wurde, wird die Kompetenz zur Bearbeitung dieses Workflowtyps entzogen (z.B. wird in der Kreditabteilung einer Bank die Summe, über deren Zuteilung ein Sachbearbeiter noch selbst entscheiden darf, von 100.000 DM auf 50.000 DM gesenkt. In diesem Fall sind u.U. nur einige Instanzen des Workflowtyps „Selbstständige_Kreditvergabe“ betroffen. Genausogut könnten alle Instanzen betroffen sein, wenn die Bank generell darauf verzichtet, Kreditzuteilungen auf Sachbearbeiterebene entscheiden zu lassen.). Entweder werden nun Agenten einer für die neue Situation kompetenten Rolle mit der Verantwortung für diese Workflowinstanzen betreut, oder es wird ein anderer Workflowtyp (Methode) für die zu lösende Aufgabe ausgewählt, der auch von der nunmehr „inkompetent“ gewordenen Rolle betreut werden kann (z.B. ein Workflowtyp, in dem der Sachbearbeiter die Erlaubnis zur Kreditzuteilung von der verantwortlichen Stelle einholt). In letzterem Fall ist es natürlich wünschenswert, daß Zwischenprodukte, die in beiden Workflowtypen vorkommen, im „neuen“ Workflow wiederverwendet werden.[1]

(7) Zu einer Aufgabe wird eine neue Methode spezifiziert. Sie ist lediglich eine zusätzliche Alternative zu den vorhandenen Methoden, braucht aber Daten, die bisher nicht zur Lösung der Aufgabe benötigt wurden.

(8) Zu einer Aufgabe wird eine neue Methode spezifiziert. Sie ist eine Verbesserung einer bereits vorhandenen Alternative und soll diese ersetzen. Hier muß darüber entschieden werden, ob Output-Produkte der fehlerhaften Methode z.B. auch bei

1. Wobei man festhalten muß, daß bei hierarchischen Workflowmodellen die Zwischenprodukte zweier Methoden bei korrekter Modellierung nie gleich sein sollten, denn wären sie gleich (d.h. gleiche Datenstruktur und gleiche Entstehungsgeschichte), so müßte der Subworkflow, durch den sie geschaffen wurden, korrekterweise eine Hierarchieebene höher stattfinden.

bereits abgeschlossenen Prozessen invalidiert werden müssen und evtl. eine Neubearbeitung der entsprechenden Aufgabe notwendig ist.

(9) Beim Entwurf einer Methode wurden bestimmte benötigte Daten nicht berücksichtigt. Diese Daten können Output-Produkte eines bereits beendeten Workflows sein (in diesem Fall ist nur die Abhängigkeit der beiden Workflows festzuhalten), sie können Output-Produkte eines in Abarbeitung befindlichen Workflows sein (in dem Fall muß zusätzlich die Abarbeitung des aktuellen Workflows so lange außer Kraft gesetzt werden, bis der datenliefernde Workflow fertig ist) oder die Daten sind bisher gar nicht vorhanden, so daß mit der Beschaffung dieser Daten eine zusätzliche Subaufgabe (mit einer potentiell neuen Klasse von Output-Produkten) zum laufenden Workflow hinzukommt, die vor einer Fortsetzung des Workflows gelöst werden muß.

(10) Bei der Ausführung einer Methode wird klar, daß bei der Entscheidung über die Dekomposition der zugehörigen Aufgabe eine Information eine wichtige Rolle spielt, die im aktuellen Plan nicht beachtet wurde. Möglicherweise wird dadurch die Entscheidung für diese Methode ungültig.

(11) Zum Lösen einer komplexen Aufgabe (z.B. Erstellung eines Berichtes) ist im Workflow-System das Starten eines bestimmten Tools vorgesehen. In einem besonderen Fall kann diese Aufgabe jedoch nicht in den Räumlichkeiten der Firma erledigt werden, sonder an einem externen Rechner, der nicht über dieses Tool verfügt. Das so erstellte Dokument kommt also zustande, allerdings nicht unter der Kontrolle des Workflow-Systems.

(12) Bei der Implementierung eines Softwaresystems wird aus Zeitgründen eine Anforderung der Spezifikation an das System fallengelassen. Die Implementierungs-Task wird geändert und der Code wird entsprechend adaptiert. Das Dokument, in dem die Anforderungen formuliert sind, muß geändert werden. Diese Änderung ist jedoch so lokal, daß eigentlich keine früher getroffenen Bearbeitungsentscheidungen davon betroffen sind.

(13) Bei der Planung eines Entwurfsprozesses für einen LKW wurde der Workflow „Getriebeentwurf“ zunächst nicht weiter spezifiziert, da seine konkrete Ausgestaltung von zu vielen Informationen abhängt, die erst im Laufe des Projekts erarbeitet werden. Liegen diese Informationen vor, kann auch der Workflow formuliert werden, d.h. er wird verfeinert.

Bei den Beispielen 5 bis 13 handelt es sich um echte Ummodellierungen, während Beispiele 2 bis 4 Umplanungen, aber keine Ummodellierungen sind. Wir werden auf diese Beispiele für Umplanungen im weiteren Text mit dem Kürzel UP<x> verweisen (mit <x> als entsprechende Nummer eines Beispieles).

3.2 Möglichkeiten des Umgangs mit Ummodellierungen

Wir wollen hier untersuchen, welche Möglichkeiten es prinzipiell gibt, mit einer Ummodellierung fertigzuwerden. Betrachten wir dazu ein Beispiel aus dem Bereich Software Engineering, anhand dessen wir einige Problematiken bei Ummodellierungsprozessen betrachten können. Wir haben an dieser Stelle darauf verzichtet, daß in dieser Domäne als Referenzbeispiel postu-

lierte ISPW-6-Beispiel (bzw. das erweiterte ISPW-7-Beispiel) zu verwenden, da es für CoMo-Kit schlecht geeignet ist, da in diesem Beispiel im CoMo-Kit-Modell enthaltene Planungsvorgänge wie Delegierungsentscheidungen explizit als Prozesse formuliert werden.

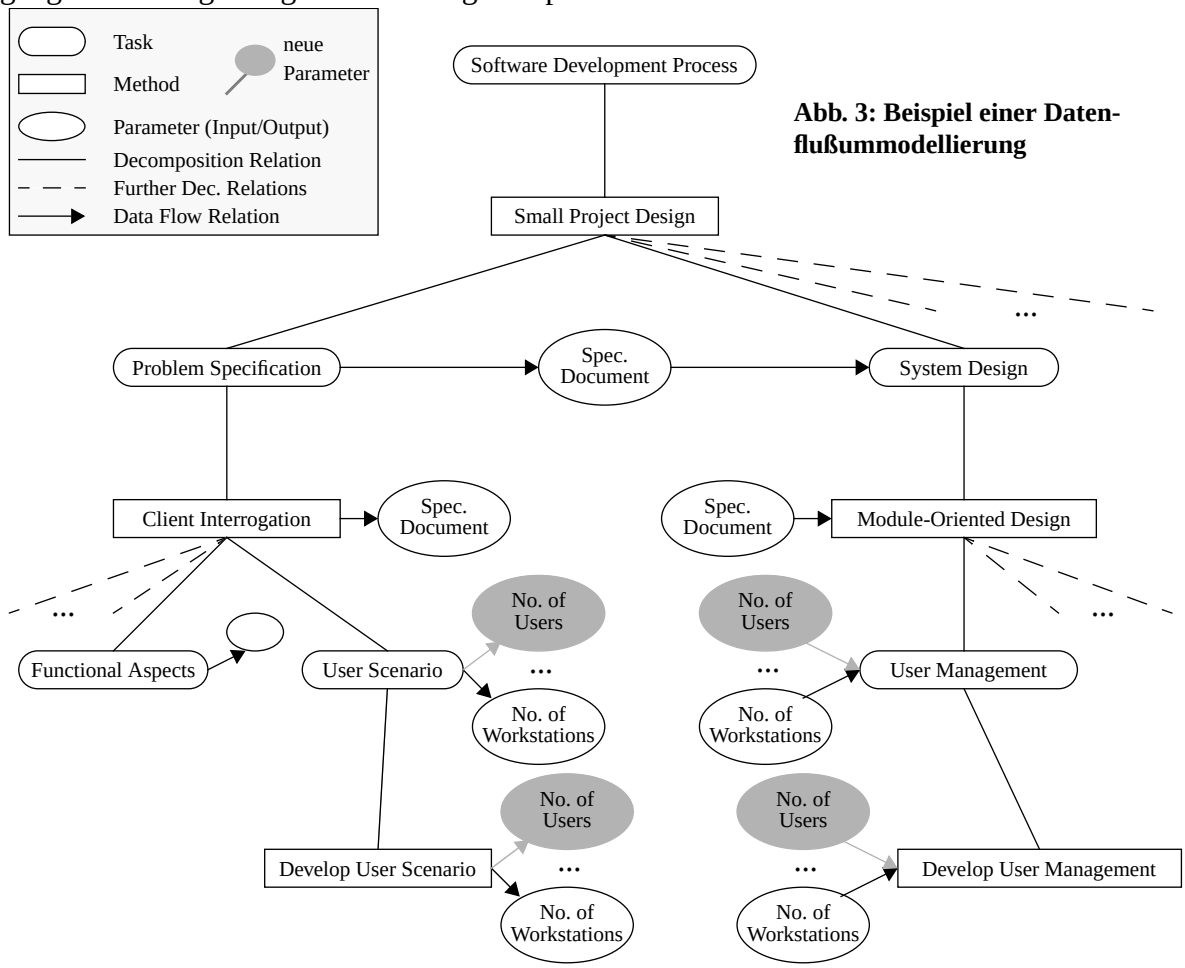


Abb. 3: Beispiel einer Datenflußummodellierung

Unser Beispiel (Abb. 3) deckt nicht alle Möglichkeiten von Ummodellierungen ab, sondern soll aufkommende Problemstellungen verdeutlichen. In einem kleineren Softwareentwicklungsprojekt stellt der Bearbeiter der (atomaren) Methode „Develop User Management“, in der er eine Administrator-Werkzeug zur Verwaltung der Benutzer des zu erstellenden Softwaresystems (z.B. Verwaltung von Zugriffsrechten, Benutzergruppen, usw.) konzipieren soll, fest, daß es bei der Entscheidung für oder gegen eine Indexierung der Benutzerdaten die Anzahl der Benutzer in die Betrachtungen einbezogen werden sollte. Diese Information liegt ihm nicht vor, d.h. sie gehört nicht zu den Inputs der Methode, die er bearbeitet. Wir gehen weiter davon aus, daß die Information im Verlaufe der bisherigen Abarbeitung nicht erstellt wurde, daß es aber einen „natürlichen“ Prozeß gibt, in dem sie hätte erstellt werden können (die atomare Methode „Develop User Scenario“).
Der Benutzer hat nun verschiedene Möglichkeiten:

- *Unkontrollierte Ummodellierung*: Es ist ihm möglich, daß Problem ohne Hilfe des WFMS zu lösen, indem er einfach beim Kunden anruft und dann die Arbeit fortsetzt. Die Nachteile liegen auf der Hand: Wird in der realen Welt dieser nicht spezifizierte Input wiederrufen (weil sich z.B. die Benutzeranzahl ändert), so erfährt der Agent dies nicht und hat auch keine Möglichkeit, auf diese Änderung zu reagieren. Außerdem steht in einem späteren Projekt ein anderer Agent u.U. vor demselben Problem.

- *Lokale Ummodellierung des Workflows*: Der Agent könnte in der Beschreibung der Methode „Develop User Management“ die Ermittlung der Anzahl der zu erwartenden Systembenutzer als Teil der Methode (bei einer komplexen Methode wäre dies eine Subtask) verankern. Auch hier sind die Nachteile offensichtlich: Zum einen „gehört“ die Ermittlung der Anzahl der Benutzer nicht in das Design der Komponente, zum anderen könnte diese Information auch für andere Subtask erforderlich werden, so daß sie sowieso explizit generiert werden sollte. Gleichzeitig ist diese Information nicht in der Abhängigkeitsverwaltung enthalten, ändert sich also etwas an dem Benutzerszenario, so würde sie ihre Validität nicht verlieren[1].
- *Globale Ummodellierung des Workflow-Modells*: Das Workflow-Modell wird so modifiziert, daß der Informationsfluß explizit ausgedrückt wird. Die „Anzahl der Benutzer“ muß dazu Teil der Outputs der Aufgabe „User Scenario“ und der ihr zugeordneten Methode(n) werden und Teil der Inputs der konsumierenden Aufgabe „User Management“ und deren Methode(n). Da in der Methode „Client Interrogation“ alle relevanten Outputs der Subtasks zu einem komplexen Outputprodukt „Specification Document“ zusammengefasst werden, müßte die Anzahl der Benutzer dort als neuer Slot auftauchen (was eine Produkttypänderung darstellt). Auch könnten als zusätzliche Folge die Postcondition der „Develop User Scenario“-Methode geändert werden (z.B. erweitert um eine Plausibilitätsbedingung „NoOfUsers >= NoOfWorkstations“), ebenso die Precondition der Aufgabe „User Management“ (z.B. könnte es sinnvoll sein, die Auswahl einer Methode von der Benutzeranzahl abhängig zu machen) und vielleicht sogar die Invarianten der Methoden „Client Interrogation“, „Module-Oriented Design“ oder „Small Project Design“.

Die globale Lösung ist somit zwar konzeptuell runder, verusacht allerdings einen höheren Modellierungsaufwand. Auch bei der eigentlichen Durchführung der Ummodellierung durch Umsetzung im Projektablaufplan entsteht bei der globalen Lösung ein deutlich höherer Aufwand. Trotzdem ist sie auf lange Sicht der sinnvollste Ansatz, da sich mit der Zeit ein zumindest fehlerarmes Workflow-Modell ergibt. Wir sollten darum bemüht sein, die entstehende Komplexität für den Benutzer so handhabbar wie möglich zu machen.

3.3 Ansätze zur Beschreibung von Ummodellierungsprozessen

Will man die genaue Semantik von Ummodellierungsprozessen festlegen, so muß man sich Kriterien erarbeiten, mit denen man Ummodellierungsprozesse einteilen kann. In der Literatur sind verschiedene Ansätze zu finden, verschiedene Arten von Ummodellierungsprozessen anhand von bestimmten Eigenschaften zu unterscheiden. Wir können dabei grob unterscheiden zwischen *modellierungsorientierten* Klassifikationen und *realitätsorientierten* Klassifikationen. Erstere unterscheiden eher technische Aspekte der Ummodellierung, während letztere versuchen zu erfassen, welcher reale Vorgang einer Ummodellierung zugrunde lag.

1. Diesen Nachteil könnte man im CoMo-Kit durch Einfügen eines Design-Rationales beheben, was jedoch der Semantik der Ummodellierung nicht gerecht würde.

Modellierungsorientierte Klassifikationen

Die folgenden Klassifikationen stammen aus dem EPOS- und dem GOODSTEP-Projekt, auf die wir in den Abschitten 4.4.2 bzw. 4.4.1 noch näher eingehen.

- *Offline- vs. Online-Änderungen* (nach [BFN94]): Ein Prozeß[1] wird Offline geändert, wenn sich keine Instanz von ihm in Abarbeitung befindet, ansonsten Online. Nur letztere sind für uns relevant.
- *Modell- vs. Instanzänderungen* (nach [BFN94]): Es kann das Modell eines Prozesses geändert werden, ohne das laufende Instanzen geändert werden („lazy propagation“), oder alle Instanzen sofort mitgeändert werden („eager propagation“). Natürlich sind auch Kompromisse zwischen den beiden Extremen denkbar, daß also z.B. nur einige Instanzen geändert werden, oder die Änderung erst mit einer gewissen zeitlichen Verzögerung propagiert wird. Es muß auch möglich sein, Instanzen zu ändern, ohne das Modell zu ändern, z.B. als Maßnahme im Exception Handling eines Workflows.
- *Geplante vs. Spontane Änderungen* (nach [BFN94]): Manche Änderungen (z.B. Verfeinerungen zunächst nur grob beschriebener Strukturen) lassen sich bereits vor der Abarbeitung des Prozeßmodells insofern planen, als das sie für das Scheduling verwaltbar gemacht werden; andere Änderungen entstehen erst spontan während der Abarbeitung, auf sie kann man sich nur begrenzt vorbereiten. Insbesondere kann es bei spontanen Änderungen vorkommen, daß es zu einer vorübergehenden Inkonsistenz[2] zwischen dem Zustand des realen Prozesses und dem Zustand der Prozeßplaninstanz kommt.
- *Kontrollierte vs. Unkontrollierte Änderungen* (nach [NWC96]): Kontrollierte Änderungen sind solche, die unter der Kontrolle des WFMS ablaufen, bei unkontrollierten Änderungen wird dem System nur das Ergebnis der Änderungen mitgeteilt. Typisches Beispiel für letztere sind Prozeßevolutionsschritte, die bei einer Post-Mortem-Analyse eines Projektes vollzogen werden. Prozeßverbesserungen werden dem WFMS zwar mitgeteilt (d.h. die verbesserten Prozesse werden modelliert), es wird aber kein expliziter Zusammenhang zwischen alter und neuer Prozeßversion hergestellt.
- *Verhaltensorientierte vs. Strukturelle Änderungen* (nach [NWC96], [JC93]): Im EPOS-Projekt (siehe Abschnitt 4.4.2 auf Seite 43) werden Änderungen, die nur das Verhalten eines Prozesses ändern (also z.B. Vorbedingungen), von solchen unterschieden, die strukturelle Modifikationen mit sich bringen (z.B. Hinzufügen eines Subprozesses oder eines Input-Produktes). Erstere heißen auch *weiche Änderungen*, letztere auch *harte Änderungen*.

Weitere Ansätze in [BFN94] sind die Klassifizierung nach der Metaprozeß-Phase (wobei Metaprozesse den Verlauf eines Designprojektes an sich beschreiben) ein , in der die Änderungen auftauchen oder die die Einteilung nach der Sichtbarkeit der Ummodellierungen für einzelne am Prozeß beteiligte Rollen bzw. Agenten.

---

1. Auch wenn im folgenden immer von geänderten *Prozessen* die Rede ist, gelten die Eigenschaften grob auch für Agenten/Rollen und Produkte.
2. Man beachte, daß die hier angesprochene Inkonstistenz zwischen tatsächlicher und virtueller WFMS-Realität hat nichts mit dem Begriff eines konsistenten Workflow-Modells zu tun hat, bei dem es um die referentielle Integrität innerhalb eines Modells geht.

Realitätsorientierte Beschreibung von Ummodellierungsprozessen in EPOS

Jeder Ummodellierung liegt entweder eine Veränderung der modellierten Realwelt, die Entdeckung neuer Sachverhalte in der Realwelt oder ein Modellierungfehler zugrunde, bei dem die Realwelt nicht korrekt im Modell abgebildet wurde.

Im EPOS-Projekt war man bemüht, eine realitätsorientierte Beschreibung von Ummodellierungsprozessen zu finden, mit deren Hilfe man die Natur eines Ummodellierungsprozesses im Hinblick auf den Aufbau einer Erfahrungsdatenbank (Schlagwort „Experience Factory“) erfassen kann. Die Untersuchungen zur Semantik von Prozeßevolution (d.h. zur Frage, welchem Realweltereignis eine Veränderung des Prozeßmodells zuzuordnen ist) umfassten in EPOS nicht nur theoretische Überlegungen, sondern auch eine Modellvalidierung an einer Fallstudie.

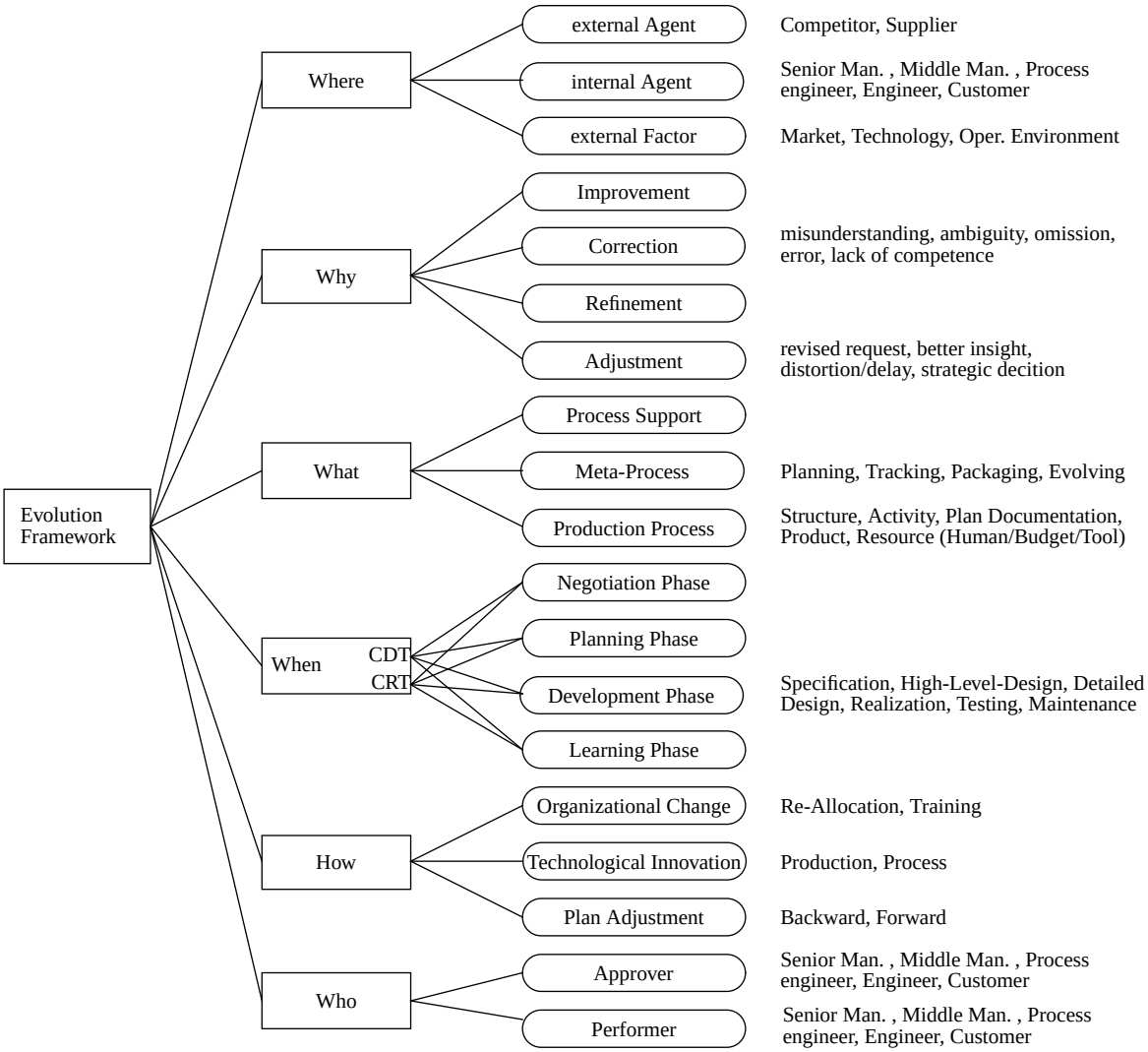


Abb. 4: Kategorisierung von Ummodellierungsprozessen in EPOS

Ummodellierungsprozesse werden in 6 Kategorien unterschieden:

- *Wo*: Durch welche Quelle wurde der Evolutionsprozeß initiiert?
- *Warum*: Aus welchem Grund wurde der Evolutionsprozeß initiiert?
- *Was*: Was wird durch die Prozeßevolution verändert?

- *Wann*: Wann (in welcher Phase) findet der Evolutionsschritt statt? Hier werden „Change Detection Time“ (CDT - der Zeitpunkt, an dem die Notwendigkeit einer Umplanung erkannt wird) und „Change Realization Time“ (CRT - der Zeitpunkt, zu dem die Umplanung tatsächlich im aktuellen Modell umgesetzt wird) unterschieden.
- *Wie*: Durch welche Art von Maßnahme wird der Evolutionsprozeß realisiert?
- *Wer*: Wer verantwortetet den Evolutionsschritt bzw. wer führt ihn durch?

Die gesamte Kategoriesierungshierarchie ist in Abb. 4 (nach [NWC96]) dargestellt.
In einer Fallstudie wurde diese Art und Weise der Beschreibung erprobt, um verschiedene Muster von häufig vorkommenden Prozeßevolutionstypen festzuhalten. Gefundene Muster umfassten beispielsweise:

- *Aufschub durch Kunden*: Der Kunde verschiebt das Projekt, um vorher noch Umstrukturierungen (z.B. Rechner- oder Netzwerkumstellungen) vorzunehmen. CDT und CRT ist der Zeitpunkt nach Ablauf der Planungsphase („When“ - „Planning Phase“); Reaktion ist eine vorwärtsgerichtete Plankorrektur („How“ - „Plan Adjustment: Forward“) des Produktionsprozesses („What“ - „Production Process“), da keine früheren Ergebnisse in Frage gestellt werden.
- *Mißverständnisse des Kunden*: Der Kunde versteht Details der Spezifikation miß oder vergißt ihre Spezifikation. Späteres Einbringen dieser Details erfordert eine rückwärtsgerichtete Plankorrektur („How“ - „Plan Adjustment: Backward“) mit Wiederaufarbeitung bereits abgeschlossener Teilaufgaben.
- *Neuverteilung von Ressourcen/Training*: Das mittlere Management („Who“ - „Internal Agent: Middle Management“) entscheidet sich für eine Ressourcenneuverteilung, um ein verspätetes Projekt zu retten („When“ - „CDT&CRT: Development Phase“). Einige Mitarbeiter müssen u.U erst auf ihre neuen Aufgaben vorbereitet werden („How“ - „Organizational Change: Training“), so daß auch Trainingsaktivitäten bei der vorwärtsgerichtete Plankorrektur („How“ - „Plan Adjustment: Forward“) berücksichtigt werden müssen.

Solche Prozeßevolutionsmuster sollen insbesondere dabei helfen, Ähnlichkeiten in den Prozeßveränderungen mehrerer Projekte zu erkennen und Erfahrungen aus dem einen Projekt im anderen zu nutzen (z.B. zwecks Abschätzung des Aufwandes eines Evolutionsschrittes).

Einordnung der Beschreibungsansätze in die Problematik unserer Arbeit

Die beiden beschriebenen Richtungen der Klassifikation von Ummodellierungen unterscheiden sich letztendlich auch dadurch, daß auf der einen (modellorientierten) Seite formale Eigenschaften im Vordergrund stehen, und auf der anderen (realitätsorientierten) Seite auch die Intentionen einer Ummodellierung eine wichtige Rolle spielen. Da wir uns um einen hohen Automatisierungsgrad des Ummodellierungsmanagements bemühen, müssen wir uns um eine möglichst scharfe und eindeutige Unterscheidung der Ausprägungen einer Klassifikation kümmern. Dies ist aufgrund der wesentlich einfacheren Spezifikation eher mit den formaleren Eigenschaften machbar. Bei realitätsorientierten Beschreibungsansätzen hingegen müssen wir stets damit rechnen, daß entweder nicht alles für die Beschreibung der realen Ummodellierungssituation wichtige in der Begriffswelt beschreibbar ist, zum anderen kann es aufgrund von sprachlichen Mehrdeutigkeiten leichter zu Mißverständnissen kommen. Sind die Ausprägungen solcher Eigenschaften Grundlage für automatisierte Reaktionen des Ummodellierungsmanagements, kann es zu Fehlreaktionen kommen.

Trotzdem müssen wir feststellen, daß z.B. die (realitätsorientierte) Unterscheidung Correction/ Improvement im EPOS-Projekt auch für uns nützlich sein kann, da nach der Ausprägung dieser Eigenschaft beurteilt werden kann, ob Ummodellierungen an einem Workflow dessen sofortige Unterbrechung erfordern (bei einer Correction sollte er nicht fortgesetzt werden).

4 Stand der Forschung

Problematiken, wie sie bei Systemen zur Steuerung von Arbeitsabläufen auftreten, werden in verschiedenen Teilbereichen der Informatik, der Wirtschaftswissenschaften und des Maschinenbaus betrachtet. In diesem Kapitel wollen wir untersuchen, inwieweit die in diesen Forschungsbereichen erarbeiteten Ergebnisse zur Lösung der in Abschnitt 1.2 beschriebenen Problematik beitragen können. Wir beschränken uns dabei weitgehend auf die Informatik, da eine Untersuchung anderer Wissenschaftsgebiete den Rahmen dieser Arbeit sprengen würde.
Der Abschnitt „Bewertung“ soll die beschriebenen Forschungsaktivitäten in Relation zum Thema dieser Arbeit und zum CoMo-Kit setzen.

4.1 Ummodellierungsprozesse in den Wirtschaftswissenschaften

Auch wenn wir hier Ansätze aus anderen Wissenschaften nicht gründlich erörtern können, gehen wir kurz auf den Bereich Wirtschaftswissenschaften (Stichwort Business Process Reengineering - BPR) ein, um festzuhalten, daß nach einem oberflächlichen Eindruck die besondere Fragestellung dieser Arbeit (Rechnergestützte Verwaltung von Umplanungsprozessen) noch nicht Gegenstand von Untersuchungen im Bereich Wirtschaftswissenschaften war.
In [GSVR94] wird dem Grundtenor nach argumentiert, daß Business Process Reengineering im Allgemeinen eine radikale Änderung der Geschäftsprozesse nach sich zieht. Alte Abhängigkeiten sollten bei der Planung nicht beachtet werden, um eine nicht vorbelastete Sicht auf die Geschäftsprozesse zu garantieren. Die Verantwortung für eine ordnungsgemäße Umsetzung des Reengineering-Prozesses (und damit für die Abhängigkeiten zwischen altem und neuem Prozeßsystem) wird einem besonders dafür qualifiziertem Agenten aufgebürdet („Reengineering Champion“). Im Allgemeinen thematisiert das Buch (wie viele andere Publikationen im Bereich BPR auch) eher Motivation und Einführung einer prozessorientierten Sicht auf die Unternehmung.
In [Hofm95] wird dem BPR als weitere Möglichkeit der Unternehmensentwicklung „Continious Improvement“ (CI) gegenübergestellt. Hierbei wird immerhin festgestellt, daß Prozeßverbesserung meist daran scheitert, daß sie „bei einer lokale Betrachtungsweise nicht auf ihre Auswirkungen auf vor- und nachgelagerte Abläufe untersucht wird“. Als Werkzeug zur Verdeutlichung dieser Auswirkungen wird jedoch nicht die Modellierung und Verwaltung von Prozeßabhängigkeiten vorgeschlagen, sondern Simulation der verbesserten Prozeßsysteme.
In [Ober94] bzw. [Ober96] geht es hauptsächlich um allgemeinere Aspekte der flexiblen Modellierung von Arbeitsabläufen in WFMS. Als wichtiger Punkt dabei wird die organisierte

Planung von Workflows über ein festgelegtes Vorgehen bei seiner Erstellung und über die Möglichkeit zur schrittweisen Verfeinerung (auch aus bereits existierenden Mustern - sog. „Referenzabläufen“ - heraus) herausgestellt. Dies alles geschieht aber immer *vor* der Instanziierung des Workflowsystems. Zwar wird die prinzipielle Notwendigkeit der Unterstützung von dynamischen Migrationsprozessen als eine Anforderung an WFMS formuliert, bei der Ausarbeitung eines Petrinetz-basierten Ansatzes wird jedoch nicht mehr gesondert auf diesen Aspekt eingegangen.
Eine etwas allgemeinere Einordnung von Prozeßbeschreibungsmodellen außerhalb der Informatik in [TC95] führt allerdings auch „Evolution“ als wichtigen Aspekt des BPR auf, jedoch ohne genauer zu spezifizieren, was genau damit gemeint ist. Dort findet man auch Verweise auf weitere verwandte Wissenschaftsgebiete.

4.2 Datenbanken

Im Bereich Datenbanken ist Workflow-Management eine der meistuntersuchtesten fortgeschrittenen Anwendungen („Advanced Applications“) von Datenbanksystemen. Im Vordergrund stehen dabei relativ implementierungsnahe Problematiken, die aus der Heterogenität der betrieblichen Hard- und Softwarelandschaften, der Verteiltheit von Agenten, Informationsquellen und Ressourcen und der Notwendigkeit neuer Verarbeitungskonzepte jenseits von ACID-Transaktionen (sog. „Advanced Transaction Models“ - [Elma92],[GR93]) resultieren (Übersicht in [GHS95]). In den ersten beiden Punkten sind Fragen der Workflowmodellierung eher unwichtig, weshalb wir auf Heterogenität und Verteiltheit hier nicht weiter eingehen.
Im letzten Punkt nimmt die Diskussion über die Vergleichbarkeit bzw. den Grad der Äquivalenz von Transaktionen und Workflows breiten Raum ein. Allgemeiner kann man von einer Diskussion über die notwendigen Konsistenzeigenschaften von Workflows reden.[Mo94]
[RS94]
In [SR93] wird eine Einteilung in transaktionale (für die gewisse Konsistenz- und Verläßlichkeitsanforderungen bestehen) und nicht-transaktionale (für die Konsistenz- und Verläßlichkeitsanforderungen nicht sinnvoll sind) Workflows beschrieben. Dabei wird unterstrichen, daß auch Zusammenhänge zwischen Aktivitäten modelliert und bei der Workflowausführung verwaltet werden müssen. Dabei entstehen Anforderungen, die viele der z.B. in [Elma92] beschriebenen erweiterten Transaktionsmodelle nicht erfüllen können.
Die meisten entwickelten Konzepte befassen sich mit der Erweiterung eines Transaktionsmodells (siehe auch die unten beschriebenen Projekte) sowohl um Aspekte, die die innere Struktur einer Transaktion genauer beschreiben ließen (*erweiterte* Transaktionsmodelle), als auch um solche, die die Zusammenarbeit mehrerer Transaktionen ermöglichten und organisierten (*relaxierte* Transaktionsmodelle). Dazu gehören sowohl Kommunikation und Koordination zweier Transaktionen als auch Modellierungsmöglichkeiten für Transaktionsanordnungen. Es wird also versucht, A (Atomizität) und I (Isolation) zu relaxieren, während C (Konsistenz) und D (Dauerhaftigkeit) unberührt bleiben.
Wir betrachten nun einige der entwickelten Transaktionskonzepte und versuchen herauszuarbeiten, inwieweit Umplanungsprozesse unterstützt werden. Danach untersuchen wir einige Ansätze genauer: das ConTracts-Modell, weil es für das WFMS elementare Erweiterungen in die bis dahin transaktionsorientierte Sichtweise auf Zustandsänderungen in Datenbanken ein-

führte und das CONCORD-Modell, da es speziell für den Entwurfsbereich (also für eine viel Flexibilität fordernde Domäne) entwickelt wurde.

4.2.1 Alternative Transaktionskonzepte

Transaktionen bzw. Transaktionssprogramme können als Prozeßbeschreibungen für atomare Zustandsübergänge von Datenbanken aufgefaßt werden. Es ist also interessant zu untersuchen, inwieweit Veränderungen bzw. Variationen während der Abarbeitung von Transaktionen möglich sind und wie sie verwaltet werden.
Das klassische ACID-Transaktionskonzept (z.B. in [GR93]) modelliert atomare Zustandsübergänge und deshalb potentiell sehr kurze, potentiell sehr einfache Aktivitäten, bei deren Abarbeitung Umplanungsprozesse keine Rolle spielen, da die Verwaltung dieser Prozesse aufwendiger wäre als Rücksetzen und Neubeginn der Transaktion.
Da fortgeschrittene Datenbankapplikationen wie Wissensbanken, andere Non-Standard-Datenbankmanagementsysteme, verteilte und parallele Systeme und nicht zuletzt auch WFMS eben doch ausgefeiltere Abarbeitungsmodelle benötigen, um z.B. inakzeptabel lange Sperren zu vermeiden, wurden zahlreiche erweiterte und relaxierte Transaktionsmodelle entwickelt, die in irgendeiner Weise die genaue Semantik der Abarbeitung einer oder mehrerer Transaktionen in einem feineren Granulat spezifizieren zu können. Da dann häufig eine hierarchische Struktur von (Sub-)Transaktionen entsteht, kommt es bei diesen Modellen durchaus zu Workflow-ähnlichen Strukturen. Die folgende kurze Übersicht ergibt sich aus den Untersuchungen in [Moha94] und [RS94].

- *Nested Transactions* ([Moss85]) waren der erste Ansatz mit hierarchisch gegliederten Transaktionen. Elterntransaktionen starten vor und enden (committen) nach der Abarbeitung aller Subtransaktionen. Ein Abort der Elterntransaktion führt zu einem Abort der Subtransaktionen. Bei *Open Nested Transactions* ([WS92]) wurde es möglich, Ergebnisse von Subtransaktionen bereits vor dem Commit der Elterntransaktion freizugeben. Da bei beiden Ansätzen alle Subtransaktionen parallel und isoliert voneinander ablaufen, ist bei einer Änderung der zugrundeliegenden Ablaufschemata bzw. Datentypen nur ein Abort mit anschliessendem Neustart der Transaktion sinnvoll. Umplanungen sind also nicht vorgesehen.
- *Split Transactions* ([KP92]) sind das einzige Transaktionsmodell, in dem eine oder mehrere Transaktionen tatsächlich während ihrer Abarbeitung verändert werden. Allerdings ist diese Veränderung nur sehr eng gefasst: Letztendlich werden nur die Ressourcen einer Transaktion (d.h. Lese- und Schreibrechte) aufgeteilt (Split-Operator) bzw. die mehrerer Transaktionen zusammengeführt (Join-Operator). Durch das Kombinieren beider Operatoren können Ressourcen von einer Transaktion zur anderen verschoben werden. Da auch hier keine Möglichkeit besteht, die innere Struktur von Transaktionen zu spezifizieren, treten auch hier keine Umplanungsproblematiken auf.
- *Generalized Sagas* ([GK87], [GGK+91]) erlauben verschachtelte Transaktionen, bei denen Ergebnisse bereits vor dem Commit der Elterntransaktion exportiert werden können, da zu jeder Transaktion eine kompensierende Transaktion definiert wird, die bei Scheitern der Elterntransaktion ausgeführt wird. Es ist möglich, in der Struktur einer Transaktion über einen Bind-Operator einen Steuerfluß zwischen zwei Subtransaktionen zu spezifizieren. Eine gezielte Verwaltung der Datenflußabhängigkeiten findet nicht statt: Nachfolgende Transaktionen werden

abgebrochen, wenn vorhergehende Transaktionen, deren Daten vorschnell freigegeben wurden, nachträglich doch noch abgebrochen werden. Trotzdem hier innere Strukturen einer Transaktion genauer spezifiziert werden, ist eine dynamische Änderung dieser Strukturen nicht vorgesehen.

- *Flexible Transactions* ([ELLR90]) wurden für den Bereich der Multidatenbanken konzeptioniert. In der Struktur einer Transaktion können - über die Möglichkeiten von Sagas hinaus - auch Alternativen bei Versagen einer Subtransaktion spezifiziert werden, die Ergebnisse kompensierbarer Subtransaktionen können bereits vor dem Commit der Elterntransaktion freigegeben werden und ein Zeitbegriff über Dauer und Startzeitpunkt einer Transaktion wird eingeführt. Die spezifizierten Abhängigkeiten werden in ein Petrinetz übersetzt und mit dessen Hilfe abgearbeitet. Hier können immerhin gewisse antizipierte Änderungen am „üblichen“ Kontrollfluß (Failures) spezifiziert werden, aber eine echte dynamische Modifikation ist nicht möglich.
- Das *Activity Transaction Model (ATM)* ([DHL91]) wurde für langandauernde Aktivitäten ausgelegt. Die Autoren formen einen neuen Aktivitätsbegriff, der zusätzlich zu „normalen“ (auch geschachtelten) Transaktionen im System existiert. Kontroll- und Datenfluß innerhalb einer Aktivität können in einem Skript oder mit einer eventgetriggerten Regelmenge spezifiziert werden. Auch wenn dadurch erheblich mehr Flexibilität bei der Spezifikation einer Aktivität realisiert wird, müssen auch hier die möglichen Berichtigungen des Vorgehens beim Abarbeiten einer Aktivität bereits bei der Modellierung bekannt sein.

Insgesamt muß man also feststellen, daß die bei erweiterten Transaktionsmodellen (gegenüber klassischen Transaktionen) konzipierten Erweiterungen zwar die Flexibilität bezüglich Daten- und Kontrollflußspezifikationen innerhalb einer Transaktion und das Failure Handling verbessern, den Anforderungen eines flexiblen WFMS insbesondere in Bezug auf dynamische Ummodellierungsprozesse genügen sie allerdings nicht. Immerhin sind einfachere Aspekte von Umplanungsprozessen wie die Spezifikation von Alternativen zu fehlgeschlagenen Subtransaktionen möglich, und die Recovery-Mechanismen der Transaktionsmodelle sind für Umplanungsprozesse wichtige Low-Level-Tools, da sie immer einen konsistenten Zustand garantieren. Man muß zudem feststellen, das die Erstellung neuer Transaktionssysteme von ihrer Grundidee her (nämlich bei Datenbankzustandsübergängen gewisse *für alle Transaktionen/Workflows gleiche* Eigenschaften zu garantieren) nicht unbedingt das geeignete Mittel zur Verwaltung von Umplanungsprozessen sind, da unterschiedliche Workflows z.B. stark unterschiedliche Grade der Isolation erfordern können, Transaktionsmodelle jedoch nur einen bestimmten Grad der Isolation realisieren.

4.2.2 Das ConTract-Modell

Das ConTract-Modell (allgemein: [WR92], Bezug zu WFMS: [RS95]) und seine prototypische Implementierung APRICOTS (Operatordarstellung: [Schw93], aktueller Entwicklungsstand: [Schw95]) wurden unter Prof. A. Reuter am Institut für parallele und verteilte Höchstleistungsrechner der Universität Stuttgart entwickelt. Das Modell geht die Probleme an, die üblicherweise im Datenbankbereich bei langlebigen Transaktionen auftreten (Vermeidung langlebiger Satz- bzw. Objektsperren, mögliche längere Unterbrechung einer Transaktion). Es ist als Low-Level-Mechanismus für die Implemtierung von z.B. einem WFMS als fortgeschrittene Daten-

bankapplikation gedacht. Der Designbereich als Sonderfall in WFMS wird dabei nicht gesondert berücksichtigt.

Modellbeschreibung

Mit dem ConTract-Modell wird neben klassischen ACID-Konstrukten (sog. *Steps*) eine abstraktere, übergeordnete transaktionsähnliche Verarbeitungseinheit, ein *ConTract*, eingeführt. Die transaktionalen Steps werden innerhalb eines ConTracts nicht, wie sonst bei Transaktionen üblich, als zusammenhangslose Übergänge von einem Datenbankzustand in einen anderen modelliert, sondern in einen Kontroll- und Datenfluß eingebunden, so daß ein Workflow-ähnliches Konstrukt entsteht. Auch können mehrere Steps zu einer Einheit (einem „atomaren Step“) zusammengefasst werden. Hierarchien von ConTracts sind jedoch nicht modellierbar.

Die Beschreibung des Kontrollflusses heißt das *Skript* eines ConTracts. Es sind Sequenzen (Step-Sequenzen als atomare Einheiten definierbar), Verzweigungen, Schleifen und parallele Abarbeitungen von Steps modellierbar (Abb. 5). Der Kontrollfluß kann dabei auch über explizite Abhängigkeiten (z.B. der Form `IF Abort(`S_1`) THEN Begin(`S_2`))` spezifiziert werden.

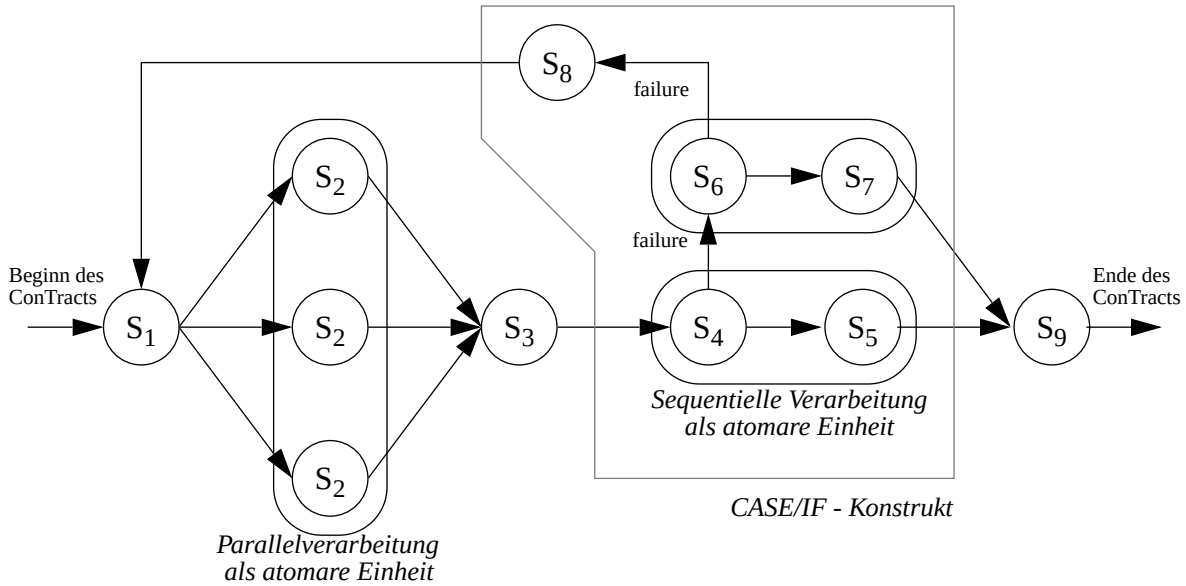


Abb. 5: Kontrollfluß zwischen den Steps eines ConTracts (nach [WR92])

Die von den Steps erstellten bzw. veränderten Datenelemente, die in der ConTracts-Begriffswelt Kontextelemente heißen, werden als updatefreie Versionen (d.h. es wird nichts überschrieben) gespeichert und über ein 5-Tupel (<Elementname>, <ConTract-Id>, <Step-Id>, <Schleifenzähler>, <Parallel-Index>) adressiert. Der Datenfluß wird jedoch nicht explizit modelliert.

Relaxierung von Atomizität und Isolation

ConTracts sind weder atomar noch laufen sie isoliert voneinander ab. Es werden insbesondere durch Steps eines ConTracts veränderte Datenelemente freigegeben, bevor der ConTract als ganzes abgearbeitet ist; diese Datenelemente werden u.U. von späteren ConTracts als Eingaben benutzt. Natürlich möchte man auch in dieser Situation ein gewisses Maß an Konsistenz

garantieren. Im ConTracts-Modell werden dazu die Konzepte *Kompensation* und *Invarianten* verwendet.
Da die Datenelemente vor Beendigung („Commit“) eines ConTracts freigegeben werden, ist bei einem Scheitern des ConTracts der sonst übliche *Rollback* des ConTracts als korrektive Aktion nicht akzeptabel. Vielmehr wird ein konsistenter Zustand vorwärtsorientiert über *kompensierende Operationen* erreicht, welche für jeden einzelnen Step definiert werden müssen.
Um für die Steps und ConTracts trotz dieser flexiblen Regelung ein gewisses Maß an Konsistenz zusichern zu können, werden diese mit Hilfe von Prädikaten - sog. *Invarianten* - synchronisiert. Dabei geben *Entry-Invarianten* die globalen Konsistenzanforderungen des Steps/ ConTracts *vor* dessen Abarbeitung an, und *Exit-Invarianten* die Konsistenzanforderungen *nach* dessen Abarbeitung. Abb. 6 gibt ein Beispiel für die Abarbeitung eines Steps an, mit dessen Hilfe in einem ConTract „Geschäftsreisenreservierung“ (aus [WR92]) bei ausreichendem Budget eine Flugreservierung gemacht und ein Ticket bezahlt (incl. Wertstellung im Budget) werden.
Treten bei der Überprüfung der Invarianten Inkonsistenzen zutage, kann ein aus Sicht des Steps/ConTracts konsistenter Zustand mit Hilfe von konfliktresolvierenden Aktionen (welche wieder als Steps realisiert sind) wiederhergestellt werden. Diese Mechanismen können auch dazu benutzt werden, den Ablauf eines ConTracts zu optimieren ([RS95]).

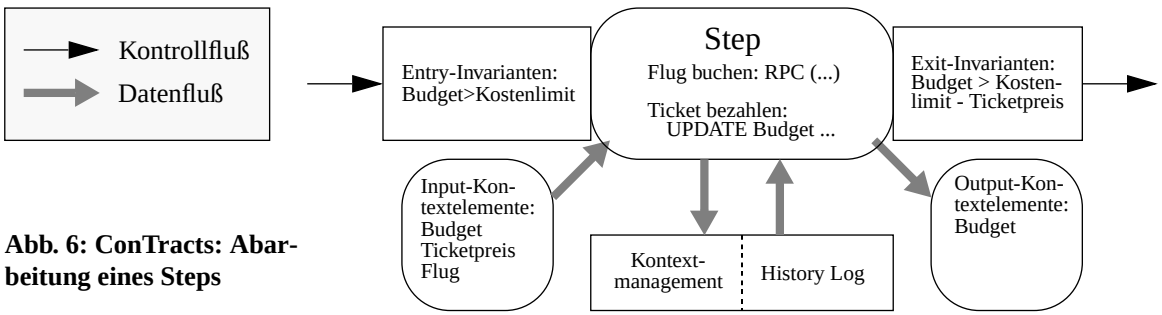


Abb. 6: ConTracts: Abarbeitung eines Steps

Darüberhinaus bietet das ConTract-Modell noch weitere bei der Implementierung eines WFMS nützliche Funktionalitäten an, wie z.B. die Abrufbarkeit der Geschichte eines Objekts bzw. ConTracts oder die Unterstützung von Kommunikation bei verteilten Systemen einschließlich der erforderlichen Robustheit bei Systemfehlern. Da diese Eigenschaften die Umplanungsthematik nicht berühren, gehen wir hier nicht weiter darauf ein und verweisen auf [WR92].

Modellierung von Umplanungsprozessen

Die Zielsetzung des ConTracts-Modells war, einen *Low-Level*-Mechanismus für fortgeschrittene Datenbank-Anwendungen zu spezifizieren. Es fehlen deshalb wichtige WFMS-Funktionalitäten. So gibt es weder die Möglichkeit, Zusammenhänge zwischen ConTracts zu spezifizieren, noch einen Agentenbegriff, noch ein Konzept zur Unterstützung von Ummodellierungsprozessen.

4.2.3 Das CONCORD-Modell

Das CONCORD-Modell ([RMH+93]) wurde im Rahmen des Sonderforschungsbereiches 124 „VLSI-Entwurfsmethoden und Parallelität“ am Lehrstuhl für Datenbanken im Fachbereich Informatik der Universität Kaiserslautern unter Prof. T. Härder entwickelt. Es steht in der Tra-

dition des ConTract-Ansatzes, geht jedoch noch mehr auf besondere Anforderungen im Entwurfsbereich insbesondere bei der Modellierung ein. CONCORD ist kein erweitertes Transaktionsmodell, sondern ein Konzept zur Modellierung und Verwaltung von Designprozessen, welches auf den klassischen Transaktionen aufsetzt.
Einige Aspekte von CONCORD (insbesondere die Modellierung von Kooperation zwischen Workflows) findet man auch in ASSET ([BDG+94]) wieder.

Das Drei-Ebenen-Modell

Die Modellierung von Designprozessen wird auf drei Abstraktionsebenen durchgeführt. Auf dem *Administration/Cooperation Level* (AC-Level) werden Aufbau und Organisation der Designprozesse beschrieben, auf dem *Design Control Level* (DC-Level) wird der Daten- und Kontrollfluß von Designprozessen genauer modelliert und auf dem *Tool Execution Level* (TE-Level) wird die transaktionale Abarbeitung einzelner Designschritte verwaltet (Siehe auch Abb. 7 auf Seite 34).
Die Designprozesse können auf dem AC-Level als Design Activities (DAs) modelliert werden. DAs bestehen aus dem zugeordneten *Design Object Type* (DOT), welcher das zu entwerfende Objekt modelliert und von dem im Laufe der DA-Abarbeitung verschiedene Versionen (*Design Object Versions* - DOVs) instanziiert werden, dem zugehörigen *Design Goal*, mit dem spezifiziert wird, welche Eigenschaften das fertige Design Object besitzen sollte, dem für die Abarbeitung der DA verantwortlichen *Designer* und dem *Design Control Parameter DC*, welcher die interne Struktur einer DA spezifiziert. Es wird dabei sowohl Aufgabendekomposition als auch Kooperation zwischen DAs unterstützt. Über die Beziehung *Delegation*[1] lassen sich Hierarchien von DAs definieren. Dabei kann - aber muß nicht - die gesamte von der Super-DA zu lösende Aufgabe an Sub-DAs zur Bearbeitung delegiert werden. Die Super-DA gilt erst dann als korrekt abgearbeitet, wenn alle Sub-DAs ordnungsgemäß abgearbeitet wurden. Auch die Designobjekte (DOTs) müssen auf zulässige Weise auf Sub-DAs verteilt werden, insbesondere bilden sie Untermengen des DOTs der Super-DA. Kooperation zwischen DAs läßt sich über die Beziehungen *Negotiation* zur Abstimmung von Designzielen und *Usage* zur Übergabe von Zwischenergebnissen (d.h. Freigabe bestimmter DOVs an kooperierende DAs) modellieren.
Auf dem DC-Level werden neben Sub-DAs *Design Operations* (DOPs) als atomare Abarbeitungsschritte, die die eigentlichen Datenmanipulationen durchführen, eingeführt. Zwischen diesen können Daten- und Kontrollfluß auf drei verschiedene Möglichkeiten spezifiziert werden:

- Explizit über *Scripts*, d.h. über die Spezifikation mit üblichen Programmierkonstrukten wie Schleifen oder Verzweigungen, oder
- Implizit über *Constraints* oder *ECA-Regeln* (ON Event IF Condition THEN Action), welche nicht nur Vereinbarungen über Daten spezifizieren können, sondern auch über den Kontrollfluß (z.B. wenn nach Abarbeitung einer DOP eines bestimmten Typs immer DOPs eines anderen bestimmten Typs zur Abarbeitung kommen sollen).

Alle drei Spezifikationsarten können auch gleichzeitig in einer DA eingesetzt werden. Der Datenfluß wird nicht explizit modelliert, der Kontrollfluß muß so spezifiziert werden, daß die zur Abarbeitung einer DA bzw. einer DOP notwendigen Input-DOVs auch zu diesem Zeitpunkt verfügbar sein können.

1. Man beachte die Begriffsverwendung: Bei der „Delegation“ in CONCORD handelt es sich um die „Dekomposition“ im CoMo-Kit. Der CoMo-Kit-Begriff „Delegation“ kommt in CONCORD nicht vor, da es in CONCORD kein Agentenmodell gibt.

Auf dem TE-Level wird die Abarbeitung der DOPs geregelt. Diese besitzen im Prinzip die ACID-Eigenschaften, erweitert durch Operatoren für langandauernde Transaktionen, die Zweiphasen-Commit (`Begin-DOP/Abort-DOP/Commit-DOP`), Arbeitsunterbrechung (`Suspend-DOP/Resume-DOP`), Savepoints (`Save/Restore`) und Versioning (Versionsableitungsgraphen, Checkin/Checkout-Konzept mit Schreibsperren auf den DOVs) realisieren.

CONCORD-Abstraktionsebenen

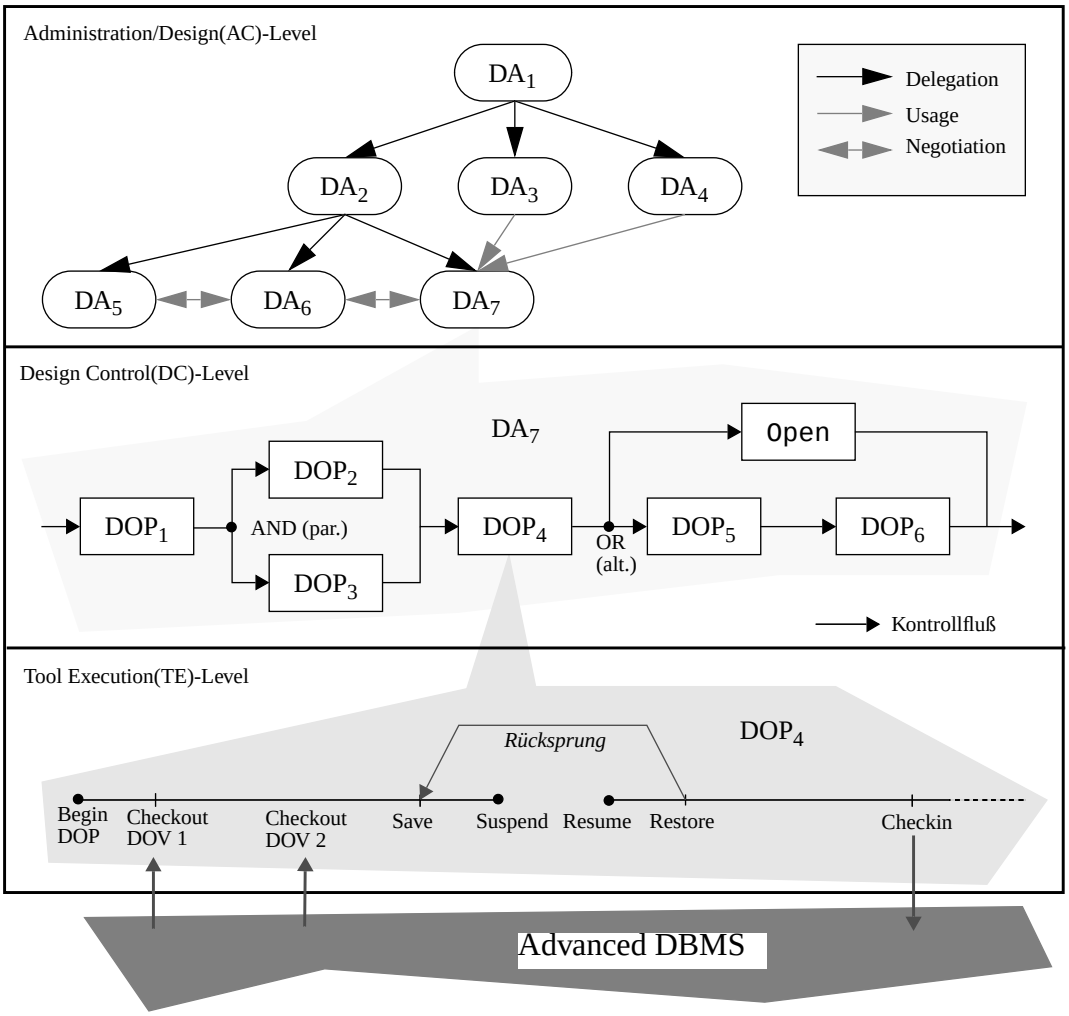


Abb. 7: Das CONCORD-Modell

Kooperation zwischen Design Activities

Die Kooperationsbeziehungen in CONCORD erlauben Kooperation zwischen DAs bezüglich Design*zielen* (Negotiation) und Design*objekten* (Delegation/Usage).

Delegation beinhaltet neben der Abstraktionsbeziehung zwischen DAs auch eine Beziehung zwischen den DOTs (und damit natürlich auch den DOVs) der beteiligten DAs, da die Super-DA der Sub-DA bei deren Spezifikation die Modellierung von Teilen des eigenen Designobjektes überträgt und eine initiale Sub-DA-DOV festlegt. Im Prinzip teilen sich die beiden DAs diese DOV. Aufgabendelegation und Designobjektdekomposition sind semantisch aneinander gebunden.

Designziele können verschiedene Formen haben, z.B. können ideale Wertbereiche für Objektslots vorgegeben werden oder auch Testtools angegeben werden, mit deren Hilfe ein Designziel als in einem bestimmten Maße „erreicht“ erkannt werden kann. Werden zwei DAs miteinander über die Kooperationsbeziehung *Negotiation* verbunden, so können diese DAs miteinander über ihre Designziele verhandeln. Dazu existieren die Operatoren `Propose` (für Vorschläge über neue Designziele) und `Agree` bzw. `Disagree` (für Vorschlagsannahme bzw. -ablehnung). Dabei kann jede DA ihre Designziele nicht beliebig verändern, sonder diese nur weiter einschränken. Damit wird verhindert, daß eine vorher konsistente Menge von Designzielen inkonsistent werden kann. Konflikte zwischen Sub-DAs werden dabei von den Super-DAs bereinigt.
Über die Beziehung *Usage* wird eine weitere Möglichkeit für das gemeinsame Nutzen von DOVs definiert. Mit Hilfe des Operators `Require` kann eine DA ein DOV mit bestimmten Eigenschaften bei einer anderen DA anfordern. Über den Operator `Propagate` kann die die DOV besitzende DA den Zugriff gewähren, auch wenn die freigegebene DOV noch nicht den Designzielen der besitzenden DA entspricht bzw. die DA noch nicht fertig abgearbeitet ist.
Bei dieser Art von Version Sharing tritt die übliche Problematik der vorzeitigen Datenfreigabe auf: DOVs werden freigegeben und später invalidiert. In CONCORD werden zwei Fälle unterschieden.
Der erste Fall tritt auf, wenn klar wird, daß eine bestimmte propagierte DOV *nicht* Vorfahr der endgültigen Output-DOV der propagierenden DA wird. In diesem Fall die wird invalidierte DOV durch eine gültige DOV, die den Anforderungen der `Require`-Operation entspricht, ersetzt.
Im zweiten Fall können Existenz (z.B. bei Abort der propagierenden DA) oder angeforderte Eigenschaften (z.B. bei Änderung der Spezifikation einer propagierenden DA derart, daß die neuen Designziele im Widerspruch zu den angeforderten Eigenschaften stehen) einer DOV nicht mehr garantiert werden. Die möglichen Reaktionen reichen dann vom Rücksetzen der anfordernden DAs über die Verwendung von Zwischenversionen beim Wiederaufsetzen einer zurückgesetzten anfordenden DA bis zum expliziten Validieren der auf der zurückgezogenen propagierten DOV basierenden DOVs durch einen Designer.

Weitere, für unsere Betrachtungen weniger wichtige Merkmale von CONCORD sind eine Evaluate-Funktion, mit deren Hilfe festgestellt werden kann, wie weit ein DOV bereits dem Designziel der besitzenden DA genügt, sowie Recoveryfunktionen nach Systemausfällen.

Modellierung von Ummodellierungsprozessen

Wie im letzten Abschnitt bereits angedeutet, existieren in CONCORD begrenzte Möglichkeiten der Ummodellierung.
Ein Skript ist in CONCORD letztendlich ein Muster für mögliche Abarbeitungspfade der DA. Teile von Skripts können durch Verwendung des `Open`-Operators zunächst unspezifiziert belassen werden und werden dann erst zur Laufzeit vervollständigt. Auch können über OR-Verzweigungen alternative Pfade festgelegt werden, zwischen denen sich der Designer erst bei der Abarbeitung entscheiden muß.
Es ist möglich, während der Abarbeitung einer DA die Designziele dieser DA einzuschränken bzw. genauer zu fassen, z.B. bei einer Veränderung der Sub-DA-Spezifikation durch die übergeordnete Super-DA oder bei einer nicht zu erfüllenden Designspezifikation. In einem solchen Fall wird die veränderte DA neu gestartet, wobei u.U. eine bei der vorherigen Abarbeitung entstandene DOV als Ausgangspunkt verwendet wird.

Die Verwaltung der in diese Vorgänge involvierten Abhängigkeiten wird nicht durchgängig von der CONCORD-Architektur unterstützt. So kann der Zusammenhang zwischen nach Verhandlungen zweier DAs (mit Beziehung Negotiation) geänderten Designzielen und den durch diese Änderungen eventuell implizierten (oder aus Effizienzgründen zu empfehlenden) Änderungen des Kontrollflusses bzw. des gesamten Aufbaus (Skripts) einer DA nicht im Modell nachvollzogen werden (wenn z.B. nach den Verhandlungen für einen Slotwert des Designobjektes sowieso nur noch ein gültiger Wert übrigbleibt, erübrigt sich eine Sub-DA, die den optimalsten Wert dieses Slots ermitteln soll).

4.2.4 Ad-Hoc-Workflows und die Behandlung sematischer Fehler

Auf der Suche nach mehr Flexibilität in WFMS ist die Integration von Ad-Hoc-Workflows ebenso wie die Behandlung semantischer Fehler ein weiteres Anliegen im Datenbankbereich. Ein Ad-Hoc-Workflow ist (nach [GHS95]) ein Prozeß, bei dem es kein a priori bekanntes Muster von Informationsfluß und Vorgehensweise gibt. Vielmehr ergibt sich das genaue Vorgehen erst während der Abarbeitung. Mit Hilfe des `Open`-Operators im CONCORD-Modell kann z.B. ein solcher Workflow spezifiziert werden.
Ein semantischer Fehler in einem Workflow liegt vor, wenn der Workflow aus einem nicht näher spezifizierten Grund nicht wie vorgesehen abgearbeitet werden kann.
Beide Themen fallen in den Themenbereich von Umplanungsprozessen, da sich jeweils die konkrete Ausprägung eines Abarbeitungsplanes für Workflowinstanzen erst während der Modellabarbeitung ergibt. Wir skizzieren hier kurz einige Ansätze aus verschiedenen Projekten.

Ad-Hoc-Workflows in MOBILE

Im MOBILE-Projekt (Überblick in [Jabl94], Ad-Hoc-Workflows in [HSS96]) unterscheidet man (wie im WFMS-Bereich üblich) zwischen *präskriptiven* Workflows, bei welchen das genaue Vorgehen im Modell genau (z.B. mit Hilfe einer Prozeßbeschreibungssprache) spezifizierbar ist, und *deskriptiven* Workflows, bei denen das genaue Vorgehen nur in Form einer (natürlichsprachlichen) Beschreibung festgehalten wird. Ein Workflow wird über seine Dekompositionsbeziehungen zu Subworkflows (funktionaler Aspekt), über den Kontrollfluß zwischen den Subworkflows (verhaltensorientierter Aspekt), über die Workflow-lokalen Variablen und den Datenfluß zwischen den Subworkflows (informationsorientierter Aspekt), über zugriffsberechtigte Agenten bzw. Rollen (organisatorischer Aspekt) und über die verwendeten Tools (operationaler Aspekt) spezifiziert. Es existieren Workflow- und Datentypen, mit deren Instanzvariablen die verschiedenen Aspekte in einer Beschreibungssprache ausgedrückt werden können.
Während die Modellierung von Ad-Hoc-Workflows als präskriptive Workflows nicht möglich ist, ist eine Realisierung als deskriptiver Workflow oft unangebracht, da z.B. durchaus bekannt sein kann, welche Subaktivitäten *auf jeden Fall* ausgeführt werden müssen. Um auch Ad-Hoc-Workflows mit der gewünschten Granularität beschreiben zu können, werden folgende Maßnahmen getroffen:

- Die Angabe von Subworkflowtypen ohne die normalerweise zur Instanziierung von Subworkflows benutzten Workflow-Instanzvariablen wird möglich und drückt aus, daß diese Subworkflowtypen durchlaufen werden *können*, aber nicht müssen.

- Die Instanzvariablen von Subworkflowtypen, die durchlaufen werden *müssen*, deren Reihenfolge aber nicht absehbar ist, werden bei der Kontrollflußspezifikation ohne irgendeinen Zusammhang angegeben.
- Der Datenfluß wird soweit wie möglich spezifiziert. Bei den optionalen Subworkflows wird bei der Datenflußspezifikation statt der nicht vorhandenen Instanzvariablen das Prädikat `INSTANCES_OF (<Workflowtype>)` verwendet.

Die größte Einschränkung des Ansatzes resultiert daraus, daß von der konkreten Art der Ausführung eines solchermaßen spezifizierten Ad-Hoc-Workflows absolute Lokalität (d.h. Kapselung gegenüber parallel laufenden Workflows) verlangt wird. Insbesondere können die optionalen Subworkflows nur auf Daten zugreifen, die vom Elternworkflow auch geliefert werden können. Würden umgekehrt alle *potentiell* notwendigen Daten vom Elternworkflow angefordert, so entständen auf der nächsthöheren Hierarchieebene eventuell Datenabhängigkeiten, die in der tatsächlichen Anwendung gar nicht vorhanden sind und beim Scheduling zu unnötigen Verzögerungen führen. Dies wiederum könnte durch eine laufende Verfeinerung während der Abarbeitung des Workflowmodells abgefangen werden, die entsprechenden Abhängigkeiten (wenn z.B. aufgrund einer Entscheidung innerhalb eines vorhergehenden Workflows die Abarbeitung eines bestimmten optionalen Workflows nicht mehr sinnvoll ist und die auf diesem begründeten Datenabhängigkeiten somit hinfällig werden) können jedoch in MOBILE nicht spezifiziert werden.
Insgesamt erreicht MOBILE damit die gleiche Flexibilität, wie sie auch durch geschickten Einsatz des `Open`-Operators im CONCORD-Ansatz möglich wäre.

Behandlung semantischer Fehler in MOBILE

Im MOBILE-Projekt sollen semantische Fehler nach einem Vier-Phasen-Konzept aus der Fertigungstechnik (nach [JWZ88]) behandelt werden ([HS96]). Die Phasen Erkennung, Eindämmung, Klassifikation und Behebung können dabei je nach Anwendung zusammenfallen. Da die möglichen Ursachen semantischer Fehler als zu Vielfältig für eine systematische Modellierung angesehen werden, wird als Konzept für die Implementierung dieser Fehlerbehandlung das oben beschriebene Ad-Hoc-Workflow-Konzept vorgeschlagen.

Behandlung semantischer Fehler in EXOTICA

Im IBM Exotica Projekt (Überblick in [MAG+95]) verfolgt man einen anderen Ansatz. Grundidee ist es, die Recovery-Mechanismen von Transaktionsmodellen als „Musterworkflows“ zur Behandlung semantischer Fehler zu modellieren ([AKA+94], [AAE+96]). Als Beleg für die Machbarkeit dieser Idee werden in [AKA+94] die Transaktionskonzepte „Sagas“ und „Flexible Transactions“ (siehe auch Abschnitt 4.2.1 auf Seite 29) mit Hilfe vom Workflowsystem IBM FlowMark nachgebaut. Beim Saga-Konzept kann so das Rücksetzten eines fehlgeschlagenen Workflows modelliert werden, bei der Umsetzung des Flexible-Transactions-Konzept wird die Spezifikation von Alternativ-Workflows gezeigt. So sollen WFMS dazu gebracht werden, die Methoden der Fehlerbehandlung in Transaktionsmodellen zu übernehmen.

4.2.5 Bewertung

Daß die Verwaltung dynamische Umplanungsprozesse im Datenbankbereich noch nicht angegangen wird, zeigen auch einige neuere Veröffentlichungen aus verschiedenen Projekten, in denen dieses Problem als noch offen bezeichnet wird ([Schw95], [WKM+95], [Jabl95a], [KR96], [WWWK96]).

Problematisch bei der Umsetzung dieser Ansätze in CoMo-Kit ist auch, das dort von vorneherein mehr Abhängigkeiten bereits im Metamodell enthalten sind (z.B. im Task-Methoden-Konzept, in dem von vorneherein die Spezifikation von Alternativen zur Lösung einer Aufgabe möglich ist) als bei den im Datenbankbereich entstandenen WFMS-Ansätzen üblich ist. Umplanungsprozesse im CoMo-Kit können dadurch auf einer höheren Abstraktionsebene ablaufen. Auch sind „echte“ Rollbacks im CoMo-Kit nicht notwendig, da die Produkte im CoMo-Kit keine festen, von Workflows modifizierten Datenstrukturen sind, sondern alle Output-Produkte immer neu erzeugt werden.
Einige technische Details sind jedoch allgemein für die Verwaltung von Umplanungsprozessen im CoMo-Kit interessant:

- *Versionierung*: Das Konzept der Versionierung ermöglicht es, strukturiert auf früher erarbeitete Produkte zuzugreifen. Bei Umplanungsprozessen hilft Versionierung von Prozessen und Produkten dabei, die zueinander passenden Versionen von Prozessen und Input- bzw. Output-Produkten festzuhalten.
- *Scheduling-Events*: Zur Behebung der Effekte eines Workflows ist es nützlich, auf Scheduling-Events (Begin/Commit/Abort) eines Workflows (einer Methode), insbesondere deren Abort (Operator- bzw. Entscheidungsrückzug im CoMo-Kit), reagieren zu können, um korrektive Aktionen für externe (d.h. nicht der Kontrolle des WFMS unterliegende) Effekte zurückgesetzter Subworkflows zu veranlassen (z.B. die Zusendung eines Entschuldigungsschreibens wegen zeitlicher Verzögerungen).

Der CONCORD-Ansatz wiederum verfügt bei der Spezifikation kooperativer Workflows über mehr Semantik als der CoMo-Kit. Immerhin scheinen auch dynamische Modifikationen des Aufbaus von Design Activities vorgesehen zu sein, wenn auch nur in eingeschränktem Maße in Form von Verfeinerungen. Andere Ummodellierungsprozesse scheitern (wie auch beim Contracts-Ansatz) schon allein daran, daß die entsprechenden Begriffe (Agenten, Rollen, etc.) nicht vorhanden sind.

4.3 Künstliche Intelligenz

In der KI findet man Vorgehensmodelle vor allen Dingen im Bereichen der Planungssysteme.
Bei Planungssystemen (z.B. in [Hert89]) findet man workflow-ähnliche Strukturen vor allen Dingen beim mehrstufigen, nicht-linearen Planen. Dort werden auch abstrakte Planschritte benutzt, die erst zu einem späteren Zeitpunkt weiter konkretisiert werden (Planverfeinerung). Allerdings verändert sich dabei das Wissen, daß man über die möglichen Planausführungsschritte hat (d.h. die Menge der Planoperatoren) normalerweise nach Planungsbeginn nicht mehr. Es wäre aber genau die zu dynamischen Ummodellierungsprozessen äquivalente Problemstellung für Planungssysteme, wenn die Menge der Planoperatoren während der Planerstellung noch verändert werden würde. Wir konnten jedoch keine Hinweise auf Forschungstätigkeit zu dieser Problemstellung finden.
Ein weiterer wichtiger Unterschied zu unserer Problematik ist, daß in Planungssystemen üblicherweise kein Datenfluß spezifiziert wird, da die Planoperatoren auf einem globalen Weltmodell operieren, d.h. weder die „Geschichte“ der vorliegenden Plansituation noch irgendwelche Abhängigkeiten von früher getroffenen Planungsentscheidungen festgehalten werden können. Wir werden im Rahmen der Beschreibung des EPOS-Projektes noch einmal kurz auf Planungskomponenten eingehen.

4.4 Software Engineering

Der Bereich Software Engineering ist vor allem deshalb für unsere Arbeit interessant, weil der Software Design Process einer der meistuntersuchtesten Entwurfsprozesse in der Literatur und das Software Engineering auch eine aktuelle Anwednungsdomäne des CoMo-Kit ist. Für das Entwickeln von Programmen wurden zahlreiche Modelle entwickelt und verglichen (Übersicht z.B. in [GJM91]). Die Unterstützung der Softwareentwicklung nach diesen Modellen mittels Computer entwickelte sich über isolierte Tools, integrierte Entwicklungsumgebungen für Compiler und Konfigurationsmanagementsysteme zur teamorientierten Softwareentwicklung hin zu prozeßorientierten Softwareentwicklungsumgebungen (Process-Centered Software Engineering Environments, PSEEs; Übersicht und wichtige Artikel in [GJ96]). Erst letztere bieten ähnliche Möglichkeiten wie WFMS, wie z.B. explizite Prozeßmodellierung und Verwaltungsmöglichkeiten zur organisierten Durchführung von Softwareentwicklungsprozessen, und müssen die entsprechenden Problematiken behandeln.
Wir betrachten nun einige PSEEs insbesondere in Hinsicht auf die vorhandenen Möglichkeiten zur Verwaltung von Umplanungsprozessen.

4.4.1 Das SPADE-Environment

Das PSEE SPADE ([BFGL94]) mit der Prozeßmodellierungssprache SLANG ([BF94], [BFGG92]) wurde im Rahmen des ESPRIT-Projekts GOODSTEP (General Object Oriented Database for Software Engineering Purposes, Projekt-Nr. 6115) entwickelt. Das Projekt war ein Gemeinschaftsprojekt mehrerer Universitäten und Unternehmen, in welchem eine datenbankbasierte Plattform für ein Software Engineering Environment (SEE) realisiert werden sollte. Es ist im Dezember 1995 ausgelaufen, die wesentlichen Ergebnisse können dem Abschlußbericht ([GOOD95]) entnommen werden.

Workflow-Modellierung mit SLANG

Mit SLANG wurde in SPADE ein vollständig-reflektiver[1] Ansatz verfolgt. SLANG ist eine auf Environment-Relationship-Petrinetzen (ERP-Netze, [GMMP91], kurz auch in [BFGL94], sind gefärbten und Prädikat-Transition-Petrinetzen sehr ähnlich) basierende Sprache. Grundelemente sind *Transitionen*, die Input- und Output-*Places* haben, und feuern, wenn *Token* entsprechend der Spezifikation der Transition in den Input-Places vorhanden sind. Feuern erzeugt die spezifizierte Art und Anzahl von Tokens in den Output-Places. Tokens und Places sind typisiert gemäß der Typhierarchie eines Modells (s.u.). Transitionen nehmen einen spezifizierbaren Zeitraum in Anspruch. Durch sogenannte *verdeckte Transitionen* (*Black Transitions*) können asynchron ablaufende Prozesse (z.B. die Abarbeitung eines Dokuments mit Hilfe eines Editors oder andere Tools) spezifiziert werden. Transitionen und Places sind über Arcs verbunden, deren Gewichte angeben, wieviele Token auf einmal durch den Arc weitergeleitet werden. Es ist auch möglich, diese Gewichte erst zur Laufzeit zu bestimmen.
Ein Modell in SLANG besteht aus einem Mengentupel (*Prozeßtypen*, *Prozeßaktivitäten*), wobei *Prozeßtypen* eine Menge von in eine Vererbungshierarchie eingebundenen Typdefinitionen ist und *Prozeßaktivitäten* eine Menge von mittels ERP-Netzen spezifizierten Prozeßdefini-

1. Wir bezeichnen ein System als *reflexiv*, wenn es über ein Verhaltensmodell von sich selber verfügt und auf einer Metaebene darüber räsonieren kann. Wir bezeichnen ein System als *vollständig-reflexiv*, wenn System- und Metaprozesse bzw. -elemente in der gleichen Weise (z.B. gleiche Modellierungssprache) beschrieben werden können.

tionen. Auch die zur Definition eines Prozesses benötigten Dinge (Token, Place, Arc, Transition) sowie die Aktivität und die Aktivitätsinstanz selber sind in der Typhierarchie enthalten (siehe Abb. 8). Alle modellspezifischen Typen werden als Subtypen von ModelType spezifiziert. Dieser Teil der Hierarchie allein ist durch den Benutzer modifizierbar.

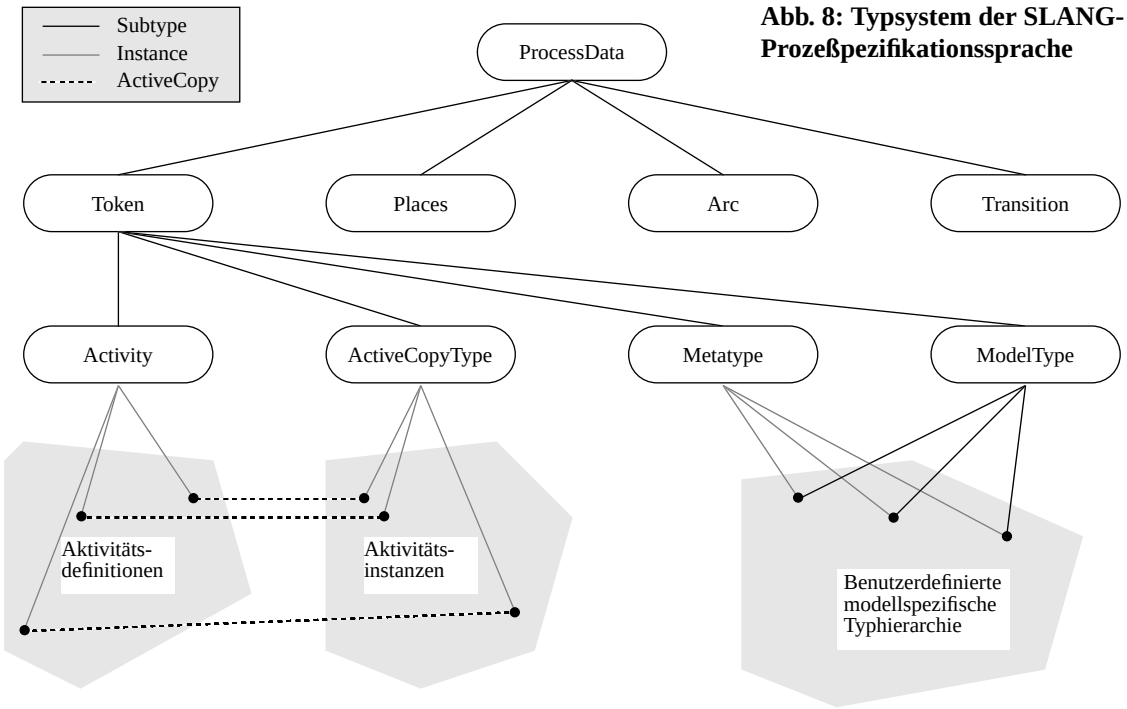


Abb. 8: Typsystem der SLANG-Prozeßpezifikationssprache

Die Reflektivität von SLANG kommt dadurch zustande, daß einerseits sowohl Aktivitätsdefinitionen und -laufzeitinstanzen (sog. „ActiveCopies“) als auch klassische Datentypspezifikationen (String, Number, etc.) als Token(-typen) definiert sind, andererseits die Aktivitäten als Petrinetze (mit Token, die wieder die aus der Hierarchie ersichtlichen Subtypen haben können) modelliert werden. Letztendlich sind damit Prozesse auf beliebigen Abstraktionsstufen modellierbar.
Ein über Datenflußabhängigkeiten hinausgehender Steuerfluß muß, wie bei Petrinetzen üblich, künstlich über Token und Transitionen modelliert werden. Auch in den Ablauf eingreifende Events werden über Transitionen modelliert.
Prinzipiell sind Aktivitäten atomare Aktionen, d.h. es werden keine Zwischenzustände einer Aktivität für aussenstehende Aktivitäten sichtbar. Kommunikation zwischen Aktivitäten wird mit Hilfe von *Shared Places* (Places, die weder notwendige Eingaben noch resultierende Ausgabe enthalten) realisiert, über die Token im- und exportiert werden können. Wichtig für die spätere Betrachtung von Umplanungsprozessen ist der Hinweis, daß insbesondere auch Aktivitäten verschiedener Abstraktionsebenen so miteinander kommunizieren können.
Die Möglichkeit zum Aufruf und das Beenden von Subaktivitäten wird über spezielle Start- und Ende-Transitionen definiert. Zwischen zwei Aktivitäten entsteht durch die Verwendung solcher Aufruf-Transitionen eine *uses*-Beziehung (Superaktivität *uses* Subaktivität), die so eine Dekompositionshierarchie (d.h. insbesondere azyklisch) über alle Aktivitäten definiert. Wurzel des Hierarchie-Graphen ist eine *Root Activity*, die von keiner anderen Aktivität aufgerufen werden darf.

Ausführung eines SLANG-Workflows

Ein SLANG-Workflow wird zur Ausführung interpretiert. Jeder Workflow wird in einer eigenen Interpreter-Instanz (die *Process Engine* PE) abgearbeitet, dessen Aufruf als verdeckte Transition mit je einem Input- und Output-Place vom Typ `ActiveCopyType` modelliert wird. In dieser grundlegenden Datenstruktur sind alle Informationen zur Abarbeitung des Workflows gegeben: die Spezifikation der Aktivität (vom Typ Activity), die benötigte Typinformation, der Zustand der ActiveCopy (determiniert durch Art und Anzahl der Token im Petrinetz der Aktivität) sowie eine Referenz auf das Start- bzw. Ende-Event zusammen mit den Tokens, die das Event ausgelöst haben.

Sind zu einem Zeitpunkt der Abarbeitung mehrere Transitionen feuerbereit, wird automatisch oder im Benutzerdialog eine Transition ausgewählt, die dann feuern darf. Verdeckte Transitionen werden asynchron abgearbeitet: Beim Start werden die Input-Token entfernt, nach dem Ende werden die Output-Token in die entsprechenden Places kopiert.

Eine Aktivität hat Start- und Ende-Events. Tritt eines der Start-Events ein und liegen in den weiteren Input-Places brauchbare Token bereit, so kann die Aktivität starten. Sie endet, wenn eines der Ende-Events auftritt, die Output-Places werden dann gemäß der Workflowspezifikation gefüllt. Das Abarbeiten von Sub-Aktivitäten ist ein Prozeß (*Invoke-Subactivity-Prozeß*), der ebenfalls in SLANG beschrieben (und abgearbeitet) wird. Kernstück ist die Schaffung einer weiteren PE (als verdeckte Transition modelliert), die als Input-Token die Beschreibung der auszuführenden ActiveCopy erhält. Dabei werden erst beim Starten der Subaktivitäts-PE die nötigen Informationen zur Erstellung der ActiveCopy (s.o.) gesammelt (Late Binding). Die PE wird erst gelöscht, wenn die Output-Places ordnungsgemäß belegt wurden.

Abhängigkeiten zwischen ActiveCopies werden ähnlich beschrieben wie Abhängigkeiten zwischen Aktivitätsdefinitionen. Auch hier existiert eine *uses*- bzw. *invokes*-Beziehung zwischen aufrufender und aufgerufener Aktivität. Diese Beziehungen liefern auch hier einen azyklischen Graphen (Datenstruktur `DynTree`). Allerdings werden die Aktivitäten nicht mit dessen Hilfe synchronisiert, vielmehr enthält jede Aktivität einen Zähler über die aktiven Subaktivitäten, um zu garantieren, daß eine Aktivität erst terminiert, wenn alle Subaktivitäten beendet sind (entspricht Zähler auf 0).

Realisierung von dynamischen Ummodellierungsprozessen mit SLANG

Der Typ ActiveCopyType ist die Datenstruktur, in der die komplette Information über eine in der Abarbeitung befindliche Aktivität enthalten ist. Ein (dynamischer) Ummodellierungsprozeß ist hier immer eine Modifikation dieser Datenstruktur. Auch wenn die Modifikationen vorgeplant werden können ist die Modifikation der Datenstruktur nur dann möglich, wenn sich die ActiveCopy nicht mehr in Abarbeitung befindet. Dazu existiert ein Suspend/Restart-Mechanismus in den Abarbeitungsstrukturen von SLANG-Activities.

Der Invoke-Subactivity-Prozeß verfügt über Shared Places (Typ ActiveCopyType), über die ActiveCopies, die suspendiert wurden, exportiert werden können. Genauso gibt es einen Restart-Place, um suspendierte Aktivitäten wieder zur Abarbeitung zu bringen. Die Suspendierung wird ausgelöst über eine Suspend-Request, die über eine entsprechende Transition direkt an die aktive Aktivitätsinstanz gestellt werden kann. Eine explizite Restart-Request wird nicht formuliert, es reicht, das geänderte ActiveCopy-Token in den Restart-Place des Invoke-Subactivity-Prozesses zu geben, wo es dann wieder eine PE zugewiesen bekommt und so abgearbeitet wird.

Eine suspendierte Aktivität wird also als Token exportiert und zur Weiterabarbeitung wieder importiert. Dazwischen kann das Token (wie jede andere Datenstruktur) beliebig manipuliert werden. Wichtig dabei ist, daß *alle* zur Abarbeitung des Prozesses notwendigen Informationen

(auch Typ- und Aktivitätsdefinitionen) im Token lokal enthalten sind, und zwar in der Version, in der sie zum Zeitpunkt der Suspendierung für die Aktivität gültig waren. Weder über die Art und Weise der Änderungen noch über die damit verbundene Semantik werden Vorgaben (Change policies) gemacht.
In SPADE unterscheidet man prinzipiell zwischen drei Arten von Prozeß- bzw. Typänderungen (nach den Überlegungen aus [BFN94]). Zum einen Modifikationen von Prozeßdefinitionen, zum anderen Modifikationen von Prozeßinstanzen (ActiveCopies). Erstere kann man weiter unterteilen in eifrige („eager“) und faule („lazy“) Modifikationen, je nachdem, ob aktive Prozeßinstanzen sofort ebenfalls modifiziert werden (eifrig) oder die Modifikationen nur an neu entstehende Prozeßinstanzen propagiert werden (faul). Natürlich sind hier auch Zwischenlösungen denkbar, z.B. sofortige Propagierung an eine Untermenge der aktiven Prozeßinstanzen. Diese unterschiedlichen Änderungsmodi werden wiederum in einem SLANG-Netz beschrieben und abgearbeitet.
Bei Typmodifikationen tritt als besonderes Problem noch auf, daß dem Inhalt eines Places plötzlich eine andere Version der Typdefinition zugrundeliegt als dem Place selber. Dies passiert z.B. bei einem Output-Place einer Aktivität, welcher von einer anderen Aktivität als Input-Place genutzt wird. Für die Token-produzierende Aktivität könnte nach einer Typmodifikation noch die alte Version relevant sein, während für die Token-konsumierende Aktivität der modifizierte Typ zählt. Das Dilemma wird z.Zt. in SPADE mit Hilfe von benutzerdefinierten Transformationsfunktionen gelöst.

Kritik am SPADE-Ansatz

Das GOODSTEP-Team selbst führt in [BFGL94] folgende Punkte als noch zu behebende Schwächen an:

- *Komplexität*: Die Anforderungen, die an ein PSEE aufgrund der Komplexität des Software Prozesses gestellt werden, werden noch nicht voll erfüllt (z.B. Verwaltung von Zugriffsrechten durch ein ausgefeiltes Rollensystem, Einbindung und Implementierung von High-Level-Konstrukten wie z.B. Change Policies).
- *Tolerierung von Abweichungen vom Prozeßmodell*: Auch kurzzeitige Abweichungen vom Prozeßmodell, die aus praktischen Erwägungen manchmal sinnvoll sein können (z.B. Ad-Hoc-Workflows oder Exception Handling), müssen in das Modell integriert werden, um den SLANG-Interpreter nicht zu verwirren.
- *Konsistenz*: Die Konsistenz eines SLANG-Netzwerks kann nicht für den allgemeinen Fall zugesichert werden. Es sind daher Tools wünschenswert, die helfen, ein Netzwerk auszutesten.

Darüberhinaus wirft auch die Realisierung von Datenstrukturen als Token Probleme auf, da es insbesondere bei Tokens in Shared Places zu unerwünschten Hotspots kommen kann. So ist z.B. die Datenstruktur `DynTree`, in der die invokes-Beziehungen aller aktiven Prozeßinstanzen als Hierarchie abgelegt ist, als Token realisiert. Sie wird häufig verändert (bei Start/Ende einer Prozeßinstanz sowie bei Suspend/Restart-Operationen) und kann sich, da sie nur komplett als Token gegen konkurrierende Zugriffe gesperrt werden kann, in einer komplexeren Anwendung als echter Flaschenhals erweisen.
Die im SPADE-Projekt entwickelten Konzepte zur Umplanung sind auf einem eher niedrigen Abstraktionslevel angesiedelt. Genaue Untersuchungen darüber, wie genau Umplanungen verwaltet werden, wie z.B. die Agenten von sie betreffenden Ummodellierungsprozessen benachrichtigt werden oder wie Daten- oder Steuerflußabhängigkeiten verwaltet werden, gab es im SPADE-Projekt offensichtlich nicht.

4.4.2 Der EPOS Process Model Ansatz

EPOS (Expert system for program and System Development, [C+94], [MCL+95] und mit Details der Implementierung: [EPOS]) ist ein PSEE, welches seit 1986 am Institut für Datentechnik und Telematik der Norwegischen Universität für Wissenschaft und Technologie in Trondheim unter Leitung von Prof. Conradi entwickelt wurde. Im Leistungsumfang ähnelt es dem GOODSTEP-Projekt, insbesondere umspannen die Entwicklungen sowohl die datenbanktechnischen Grundlagen (EPOSDB) als auch High-Level-Konzepte zur Prozeßpezifikation und -evolution (Spell-Sprache, EPOSPM Planner & Enactment Machine).
Ähnlich zu SLANG in SPADE gibt es in EPOS die reflektive Prozeßmodellierungssprache SPELL ([C+92]). Es handelt sich dabei um eine objekt-relationale Erweiterung der PROLOG-Anfrage- und Modellierungssprache der zugrundeliegenden Datenbank EPOSDB. Die Mächtigkeit der Sprachkonstrukte entspricht der von SLANG und auch in architektonischen Details wie Typ- bzw. Klassensystem, Abarbeitung durch Interpretation oder Aufgabenbeschreibung über Dekompositionshierarchien, Skripte, Prä- und Postbedingungen sowie Verwendung formaler Parameter ähneln sich die Ansätze. Wir gehen deshalb hier nicht weiter auf SPELL ein.
In der Architektur bzw. den Komponenten ist der größte Unterschied zu vergleichbaren Ansätzen, daß in EPOS die automatische Dekomposition mit Hilfe eines Planers teilweise automatisiert werden kann.
Wichtiger sind Unterschiede in den Beschreibungsmöglichkeiten für den Software Process, die in EPOS erweitert sind. Neben Agenten- bzw. Rollenmodell, Produktmodell und Aktivitätsmodell sind folgende Begriffe modellierbar:

- *Tools*: Toolmodelle enthalten Informationen über Toolnamen, Anzahl und Art benötigter Parameter sowie Informationen zum Toolaufruf.
- *Kooperation*: Im Kooperationsmodell können allgemeine Strategien über die Sichtbarkeit von Informationen in verschiedenen Teilbäumen der Aktivitätshierarchie definiert werden (Wann werden Änderungen in einer Aktivität für andere sichtbar? Wann werden Produkte einer Aktivität für nachfolgende sichtbar? Wie kann während der Abarbeitung - z.B. bei hochgradig voneinander abhängigen Aktivitäten - über die Sichtbarkeit von Informationen verhandelt werden? Etc.).
- *Projekt*: Das Projektmodell wird dazu benutzt, die Aktivitätshierarchie weiter zu strukturieren (Aufteilung in Subprojekte). Insbesondere sind Metriken zur Bewertung der Performance an das Projektmodell gebunden.

Auch in diesen Modellen können Veränderungen während der Laufzeit stattfinden (s.u.).
Im Rahmen des EPOS-Projektes ist die Semantik von Prozeßevolution (d.h. die Bedeutung bzw. Ursache von Prozeßevolutionsschritten) genauer untersucht worden als es im SPADE-Ansatz der Fall war ([JC93] - Grundlegende Konzepte und Tools; [LC93] - Automatismen zur Unterstützung von Prozeßevolution; [NC94], [NWC96] - Semantische Aspekte von Umplanungsprozessen). Eine Beschreibung der Attribute, mit deren Hilfe Prozessevolutionsschritte in EPOS beschrieben werden sollen, haben wir bereits in Abschnitt 3.3 auf Seite 22 geliefert.

Prozeßevolution in EPOS

Das Problem Prozeßevolution an und für sich wird in EPOS mit Hilfe eines vierschrittigen Metaprozesses verstanden. In der Hauptsache werden Umplanungen danach Unterschieden, ob sie geplant/ungeplant bzw. kontrolliert/unkontrolliert (siehe auch Abschnitt 3.3 auf Seite 22) auftreten.

- **MP1:** Planning & Instantiation. Ein neues Projekt wird unter Verwendung eines generischen Planmodells und einer Erfahrungsdatenbank, in der sowohl Informa-

tionen über Modellierung und Abarbeitung bzw. Bewertung ähnlicher, abgeschlossener Projekte als auch über deren Evolutionsgeschichte gespeichert sind, modelliert und mögliche Veränderungen gegenüber der Modellbasis werden antizipiert. Prozeßevolution vom Typ *Planned/Controlled* findet also in dieser Phase statt.

- **MP2:** Execution & Tracking. Während der Abarbeitung werden wichtige Daten über Modellierung und Performance aufgezeichnet. Wenn während der Abarbeitung eine Modellveränderung (d.h. ein Evolutionsschritt) ansteht, wird in früheren, ähnlichen Projekten nach ähnlichen Evolutionspattern gesucht, um Zeit- und Kostenaufwand abschätzen zu können. Hierbei handelt es sich dann um Evolution des Typs *Unplanned/Controlled*.
- **MP3:** Packaging & Assessment. Die in MP2 gewonnenen Daten werden verdichtet (d.h. zu aussagefähigen Maßzahlen zusammengefaßt) und mit bisherigen Erfahrungen verglichen. Kosten- und Nutzeneffekte von Prozeßevolutionsschritten werden ebenfalls verglichen. Sowohl der MP2-Phase (zwecks Plankorrektur) als auch der MP4-Phase (zwecks Hinzulernen) werden die Maßzahlen übermittelt.
- **MP4:** Evolving & Learning. Aus dem Projekt gezogene Erfahrungen werden auf das generische Modell verallgemeinert und Korrekturen werden vorgenommen. Die Erfahrungsdatenbank wird mit den Ergebnissen dieser Post-Mortem-Analyse gefüttert. Gemeinsam mit MP3 wird so Prozeßevolution vom Typ *Unplanned/Uncontrolled* behandelt.

In diesem Modell wird also die Projektplanung und -durchführung formuliert. Aktionen in diesem Metamodell sind häufig Modifikationen des Softwareentwicklungsprozesses. Modellevolution *während* der Abarbeitung des Softwareentwicklungsprozesses findet dabei also in Phase MP2 statt.
Die Existenz dieses Metaprozesses wird im SPADE-Projekt als Einschränkung und als wichtigen Unterschied zur SLANG-Methodik gesehen, wo kein spezielles Metaprozeß-Modell zugrundeliegt. Diese Kritik ist mittlerweile nicht mehr berechtigt, da nach einer Überarbeitung des Typsystems in EPOS dieser Metaprozeß nicht mehr fix implementiert ist, sondern in SPELL (nunmehr vollständig-reflektiv) als ein Metaprozeß von vielen möglichen formuliert ist.

Ummodellierungsprozesse in EPOS

Generell können in EPOS in jedem der oben genannten Modelle Ummodellierungsprozesse auftreten, auf die jeweils entsprechend reagiert werden muß (nach [NWC96]). Die Untersuchungen diesbezüglich sind jedoch noch auf einem recht allgemeinen Stand:

- *Activity Model*: Im Activity Model wird grundsätzlich zwischen weichen und harten Umplanungen unterschieden (siehe Abschnitt 3.3 auf Seite 22).
  Weiche Umplanungen können Änderungen am Code, den Attributen oder den Vor-/Nach-Bedingungen einer Task sein. Mögliche Reaktionen umfassen einen Neustart der Task, eine Neubewertung betroffener nachfolgender Tasks und eine Neuberechnung der Zeitvariablen der betroffenen Projekte.
  Harte Umplanungen verändern die Struktur des Tasknetzwerks. Mögliche Veränderungen sind das Hinzufügen/Entfernen einer Task aus dem Task-Netzwerk sowie das Hinzufügen/Entfernen eines (Input- oder Output-)Produktes einer Task. Das Verschieben einer Task wird dabei als Kombination von Entfernen und Hinzufügen einer Task realisiert. Diese Veränderungen können auch während der Abarbeitung der Tasks vorgenommen werden.

- *Product Model*: Den Ummodellierungsprozessen in Produktmodellen wird man mit Hilfe eines Versionierungskonzeptes herr, welches über die richtige Zuordnung von Produktinstanzen wacht. Konflikte, die aus der Versionierung von Produkten resultieren, werden mit Hilfe der im Kooperationsmodell festgelegten Tools und Richtlinien gelöst.
- *Tool Model*: Änderungen im Toolmodell resultieren in einem Neustart der von den Änderungen betroffenen Tasks.
- *Role Model*: Änderungen im Agenten-/Rollenmodell machen häufig eine Reallokation von Ressourcen erforderlich. Bei schwerwiegenderen Änderungen können jedoch auch Projekt- oder Taskneuansetzungen nötig werden.
- Cooperation Model: Änderungen im Kooperationsmodell erfordern u.U. einen Neustart der betroffenen Projekte sowie weitere Umplanungen.

Tiefergehende Beschreibungen dieser Konzepte sowie der Stand der Implementierung sind aus den Publikationen des Projektes nicht ersichtlich.
In [Liu91] werden Planverfeinerungen als Ummodellierungsprozesse im Kontext der Planungskomponente von EPOS beschrieben. Die Notwendigkeit von Umplanungen wird an die erfüllbarkeit von Vorbedingungen (über die auch der Datenfluß spezifiziert wird) geknüpft. Wird z.B. ein Input-Produkt geändert, oder wurde die dieses Produkt erstellende Task abgebrochen, so wird durch ein nicht erfüllbares Existenz-Prädikat in der Vorbedingung der Task, die das Produkt benötigte, eine Umplanungsaktivität als Vorlauf zur Taskabarbeitung ausgelöst.

4.4.3 Weitere Projekte

In anderen Projekten im Bereich Software Engineering werden weitere Tools und Mechanismen vorgeschlagen, um auch die praktische Umsetzung von Umplanungsprozessen zu vereinfachen. In [ACF96] wird das OPSIS-Sichtenmodell vorgestellt und dargelegt, daß ein Sichtmechanismus nicht nur die Komplexität von Prozeßmodellen reduzieren hilft, sondern auch bei Verständnis und Umsetzung von Prozeßevolution von Vorteil ist.
Im DYNAMITE-Projekt ([HJKW96]) wird ein dem REDUX-System ähnlicher Ansatz auf der Basis eines Graph Rewriting Systems verfolgt. Dynamische Tasknetze erlauben die inkrementelle Entwicklung von hierarchischen Aufgabenstrukturen. Allerdings wird auch hier die zentrale Problematik dynamischer Ummodellierungsprozesse - das Umplanen während der Abarbeitung einer Aufgabe - dadurch vermieden, daß nach dem Start einer Aufgabe keine Änderungen ihrer Definition mehr zugelassen werden, so daß im Falle einer Änderung die Aufgabe komplett zurückgesetzt werden muß. Interessant am DYNAMITE-Modell ist, daß von vorneherein neben dem üblichen vorwärtsgerichteten Datenfluß ein rückwärtsgerichteter Feedback-Datenfluß modelliert wird, über den erst in späteren Projektphasen festgestellte Fehler rückgemeldet werden.

4.4.4 Bewertung

Ummodellierungsprozesse werden im Bereich Software Engineering eher aus dem Blickwinkel der Prozeßevolution betrachtet. Bei der Verwaltung von Prozeßevolution sind offensichtlich drei Eckpunkte wichtig:

- *Formulierung von Metaprozessen*: Bestimmte Prozeßevolutionsarten, wie z.B. Aufgabenverfeinerung, können bereits im Voraus in einem Metamodell eingeplant

werden. Eine Bestrebung bei PSEEs ist es, dieses Wissen über den Metaprozeß ebenfalls ins Prozeßmanagement zu integrieren.

- *Bewertung von Prozeßevolutionsschritten*: Eine genaue Charakterisierung und Analyse von Ummodellierungsprozessen kann - in einer Erfahrungsdatenbank gespeichert - wertvolle Hinweise bei der Beurteilung zukünftiger Ummodellierungsprozesse geben.
- *Durchführung der Umplanungen*: Der zentrale Punkt ist die Umsetzung von Umplanungen (d.h. einer Menge von atomaren Umplanungsschritten) im konkreten Projekt. Es müssen sowohl alle notwendigen Prozeßmodellmodifikationen vorgenommen (Modellkonsistenz erhalten) werden als auch die betroffenen Instanzen (Agenten) von der Umplanung benachrichtigt werden.

Beim ersten Punkt ist eine vollständig-reflektive Prozeßspezifikationssprache von Vorteil, da so ein integriertes Prozeßmanagementsystem realisierbar ist. Der zweite Punkt gewinnt unter dem Stichwort „Experience Factory“ an Bedeutung, bei der es darum geht, für aktuell anstehende Entscheidungen auf bereits gemachte Erfahrungen effektiv zugreifen zu können. Bei vielen Umplanungsprozessen hat der Prozeßdesigner die Wahl abzuwägen, ob der durch die Umplanung entstehende Nutzen die Kosten rechtfertigt. In diesem Fall können früher gewonnene Erfahrungen aus ähnlichen Situationen herangezogen werden. Ob die in Abb. 4 auf Seite 24 dargelegte Begriffswelt für diese Zwecke ausreicht, muß wohl noch näher untersucht werden. Beim dritten Punkt steht vor allen Dingen die Verwaltung der zwischen verschiedenen Subtasks bestehenden Abhängigkeiten im Vordergrund.
Ein weiterer wichtiger Punkt ist die Tolerierung von Inkonsistenzen zwischen Modellinstanzen und der Prozeßrealität durch das PSEE dar, wie sie selbstkritisch im SPADE-Modell als Defizit dargestellt wird.

4.5 Resumee

Wir sind bereits bei den einzelnen Fachrichtungen darauf eingegangen, welche Aspekte für ein Ummodellierungsmanagement interessant sein könnten. Hier wollen wir allgemeinere Lehren aus der Literaturarbeit ziehen.
Es hat sich gezeigt, daß der CoMo-Kit-Ansatz der *expliziten* Modellierung von Abhängigkeiten zwischen Aufgaben und Produkten sowie der Möglichkeit, Planungsentscheidungen und deren Rückzug zu verwalten, relativ ungewöhnlich ist. Desweiteren findet die Problematik der Verwaltung *dynamischer* Umplanungen bisher wenig Beachtung (Im Grunde eigentlich nur im EPOS-Projekt, und dort mit unklarer Operationalisierung). Zusammen ergibt sich damit für unsere konkrete Fragestellung der Konzeption und Operationalisierung eines ummodellierungsmanagements kein großer Nutzen[1].
Immerhin können wir aus den Erfahrungen und Ideen des EPOS-Projekts lernen, daß wir für das Ummodellierungsmanagement drei Richtungen unterscheiden können. Auf der untersten Ebene (und im Vordergrund unserer Arbeit) befindet sich das Ermöglichen des Einbringens von Ummodellierungen in eine sich aktuell in der Abwicklung befindlichen Instanz eines Prozessmodells. Eine darüberliegende Ebene ist das Beschreiben einer Ummodellierung eines

1. Natürlich können wir aus dem Bereich der Datenbanken noch das Transaktionskonzept als Rahmen für die Durchführung von Änderungen auf (im CoMo-Kit: Modellierungs-) Daten mitnehmen.

Prozesses durch einen übergeordneten (Meta-)Prozess (dessen Erstellung natürlich auch wieder als Prozess beschrieben werden könnte, usw.). Solche Beschreibungen erhöhen durch die explizite Modellierung zum einen das Verständnis für solche (Meta-) Prozesse, zum anderen könnten sie dazu benutzt werden, auch die Durchführung bestimmter Umplanungen/Ummodellierungen - soweit erforderlich oder sinnvoll - zu automatisieren oder wenigsten geordneter abarbeitbar zu machen.
Eine dritte Richtung geht dahin, mit Hilfe einer Erfahrungsdatenbank praxisrelevante Daten von Ummodellierungsprozessen zu sammeln, um z.B. Kostenabschätzungen machen zu können oder andere Informationen aus früheren, verwandten Projekten in einem aktuellen Projekt zu verwerten oder gar wiederzuverwenden.
In dieser Arbeit beschäftigen wir uns im wesentlichen mit dem Einbringen der Ummodellierungen in die Prozessinstanz. Wir werden auf die anderen Aspekte im „Ausblick“ (Abschnitt 6.2) noch einmal kurz eingehen.

5 Ein Modell zur Verwaltung von Ummodellierungsprozessen

Wir wollen nun unser Konzept des Ummodellierungsmanagements im CoMo-Kit darstellen. Dazu beschäftigen wir uns zunächst nochmal mit der Begriffswelt und dem Workflow-Modell, auf welchem wir unsere Betrachtungen aufbauen wollen (Abschnitt 5.1). Dann gehen wir noch einmal ausführlicher auf die Anforderungen an ein solches Ummodellierungsmanagement ein (Abschnitt 5.2). Danach führen wir auf, welche Beschreibungsformalismen für Ummodellierungen wir benötigen, um die Anforderungen zu erfüllen (Abschnitt 5.3). Im Hauptteil dieses Abschnittes (Abschnitt 5.4) versuchen wir, für jede Anforderung Konzepte zu erarbeiten, die dieser genügen. Wir beschließen diesen Abschnitt mit Apekten der Operationalisierung des Ummodellierungsmanagements(Abschnitt 5.5).

5.1 Das CoMo-Kit-Workflow-Modell und seine Ummodellierungen

Da wir unsere Untersuchungen im Prinzip nicht speziell auf den CoMo-Kit ausrichten wollen, wäre es gut, ein allgemeineres Workflow-Modell unseren Anforderungen zugrunde legen zu können. Der Workflow-Bereich zeichnet sich jedoch durch im Detail recht heterogene Begriffswelten aus. Einen Standardisierungsversuch gibt es im Datenbankbereich durch die Workflow Management Coalition [WFMC96]. Das dortige Modell unterstützt jedoch keine hierarchischen Workflows (jedenfalls nicht explizit), welche dem CoMoKit-Konzept zugrundeliegen, und welche auch im Datenbankbereich (z.B. [KR95]) als z.B. im Entwurfsbereich unabdingbar bestätigt wurden. Deshalb orientieren wir uns bei unseren Untersuchungen im wesentlichen an der aus Abschnitt 2 bekannten CoMoKit-Systematik und -Begriffswelt, welche wir leicht modifizieren wollen.

Modifiziertes CoMo-Kit-Modell

Wir wollen das CoMo-Kit-Modell verallgemeinern, um die Diskussion zunächst auf einem nicht-projektspezifischen Niveau zu halten und uns auf das Wesentliche zu konzentrieren. In folgenden Punkten wollen wir verallgemeinern:

- Wir verwenden kein besonderes Ressourcen- oder Agenten-Modell. Mögliche Agenten werden einfach als bei den Aufgaben und Methoden eingetragen betrachtet, obwohl ein flexibles Matching anhand von Kompetenzen natürlich komfortabler ist. Es reicht jedoch für unsere Zwecke aus, die Weg- oder Hinzunahme von Agenten in den entsprechenden Aufgaben- und Methodenattributen als relevante

Ummodellierung zu betrachten. Selbst bei einem gut ausgearbeiteten Ressourcenmodell ist der Nettoeffekt des Hinzukommens bzw. Wegfallens eines kompetenten/ inkompetenten Agenten das für das Scheduling relevante Ereignis.

- Im Prozeßmodell verzichten wir auf die Einbeziehung optionaler Parameter, der Context Information und der „weiteren Attribute“ da sie im Zusammenhang mit Ummodellierungen irrelevant sind, da über sie keine verbindlichen Abhängigkeiten definiert werden.

Andere Aspekte wollen wir hier nicht von vorneherein ausschliessen, da wenigstens untersucht werden sollte, ob Ummodellierungen in diesen Bereichen erlaubt sein sollten oder nicht. Dieses ist das Workflow-Modell, welches als Basis für unsere Diskussion dient.
Wir betrachten nun zunächst allgemein die Ummodellierungsmöglichkeiten, die im CoMo-Kit-Modell existieren. Wir gehen dann auf die Problematik von Ummodellierungen im Produktmodell gesondert ein, da wir ansonsten den Schwerpunkt unserer Arbeit auf Ummodellierungen im Prozeßmodell legen. Zum Schluß dieses Abschnittes betrachten wir das Konzept der Design Rationales im Kontext von Ummodellierungen.

5.1.1 Ummodellierungsmöglichkeiten im CoMo-Kit

Im CoMo-Kit können Modellierungseinheiten Aufgabe, Methode und Produkt(typ) modifiziert werden. Wir betrachten nun, wie erfaßt werden kann, was genau im Rahmen der Ummodellierung passiert. Der Umplaner erstellt während der Modellierung der Modifikationen praktisch die *Feinspezifikation* der Ummodellierung (d.h. welche atomaren Ummodellierungen vorgenommen werden). Die Möglichkeiten zur Modifikation der Modellierungseinheiten ergeben die Menge der *atomaren Ummodellierungsoperatoren* (siehe Tabelle 14), aus denen solch eine Feinspezifikation im CoMo-Kit bestehen kann.
Dabei können bei Aufgaben Parameter hinzugefügt/weggenommen werden (Plan-Input-added/ -retracted; Exec-Input-added/-retracted; Output-added/-retracted), Methoden oder mögliche Bearbeiter hinzugefügt/weggenommen werden (Method-/Agent-added/-retracted), Parametertypen (Plan-Input-Type-/Exec-Input-Type-/Output-Type-modified) oder Bedingungskonstrukte (Precondition-/Invariant-/Postcondition-modified geändert werden. Auch bei Methoden können sich Input- und Output-Parameter und Bedingungskonstrukte entsprechend denen der Aufgaben ändern. Darüberhinaus sind noch Änderungen am Skript einer (komplexen) Methode möglich. Dies umfaßt sowohl das Hinzufügen und Löschen von Subtasks (Task-added-/retracted) und Formalen Parametern (Formal-Parameter-Added/-Retracted) als auch das Hinzufügen/Löschen von Input- (Parameter-Task-Relation-added/-retracted) und Outputrelationen (Task-Parameter-Relation-added/-retracted) zwischen Subtasks und Formalen Parametern. Auch der Typ Formaler Parameter kann modifiziert werden (Formal-Parameter-Type-Changed).
Im Produkt(typ)modell können bei komplexen Typen entweder Slots und Attribute hinzugefügt und gelöscht (Slot-/Attribute-added/-retracted) oder Typ und Kardinalität von Slots und Attributen modifiziert werden (Slottype-/Attribute-Type-/Slotcardinality-modified).

5.1.2 Ummodellierungen im Produktmodell

Wir haben in der Literatur gesehen, daß in vielen Ansätzen versucht wird, den Produktmodifikationen mit einer Versionierungskomponente herr zu werden. Insbesondere in den Ansätzen

Aufgabenmodell	Methodenmodell	Produktmodell
Plan-Input-added	Input-added	Slot-added
Plan-Input-retracted	Input-retracted	Slot-retracted
Plan-Input-Type-modified	Input-Type-modified	Slottype-modified
Exec-Input-added	Output-added	Slotcardinality-modified
Exec-Input-retracted	Output-retracted	Attribute-added
Exec-Input-Type-modified	Output-Type-modified	Attribute-retracted
Output-added	Task-added	Attribute-Type-Modified
Output-retracted	Task-retracted	-
Output-Type-modified	Formal-Parameter-Added	-
Method-added	Formal-Parameter-Retracted	-
Method-retracted	Formal-Parameter-Type-Changed	-
Agent-added	Parameter-Task-Relation-added	-
Agent-retracted	Parameter-Task-Relation-retracted	-
Precondition-modified	Task-Parameter-Relation-added	-
Invariant-modified	Task-Parameter-Relation-retracted	-
Postcondition-modified	Precondition-modified	-
-	Invariant-modified	-
-	Postcondition-modified	-

Tabelle 3: Atomare Ummodellierungen im CoMo-Kit

aus dem Datenbankbereich greift man auf diese Funktionalität zurück, da sie im Verständnis der dortigen Workflowansätze sowieso gebraucht wird. Die Erstellung eines Produktes wird dort häufig als das Ausfüllen einer zu Beginn der Abarbeitung leeren Datenstruktur, von der mit jedem Verarbeitungsschritt eine neue Version entsteht. Schon diese Versionen müssen verwaltet werden. Das Produktkonzept des CoMo-Kit hingegen modelliert Produkte als temporäre Einheiten, die irgendwann produziert und woanders konsumiert werden und dort in anderen Produkten aufgehen. Die Abhängigkeiten, die durch dieses Vorgehen entstehen, werden indirekt durch das TMS von CoMo-Kit mitverwaltet. CoMo-Kit enthält also eine implizite Instanzenversionierung.

Auch bei der Ummodellierung von Produkttypen bietet es sich daher an, einen Ummodellierungsprozeß im Produktmodell nicht als Typmodifikation aufzufassen, sondern als Modellierung eines neuen Typen. Dies hat dann natürlich auch zur Folge, daß Typänderungen bei Slot- und Attributtypen auch zu neuen Konzepten bzw. Konzepttypen führt.

Grundsätzlich kann man bei Produkttypmodifikationen unterscheiden zwischen strukturverändernden Modifikationen (bei Konzeptbeschreibungen) und Transformationen von einem Grundtyp in einen anderen. Wir gehen davon aus, daß bei Konzeptänderungen auch das Prozeßmodell modifiziert wird, und die Reaktionen, die aufgrund der Prozeßmodellmodifikationen erfolgen, die Variable automatisch auf einen korrekten Wert setzen. Kommt also in einer Konzeptbeschreibung beispielsweise ein Slot hinzu, so wird auch im Prozessmodell ein neuer Output auftauchen, der die Wertbelegung dieses Slots durch eine Task repräsentiert.

Dies gilt auch für Typwechsel von einem Grundtyp in einen Konzepttyp und umgekehrt. Transformationen von Grundtypen hingegen müssen separat durchgeführt werden. Wir lassen dabei nur solche Typmodifikationen zu, für die Transformation der alten Werte in die neuen automatisierbar ist (Wir gehen darauf näher in Abschnitt 5.4.5.1.4 und Abschnitt 5.4.5.2.3 ein). Zur Zeit gibt es im CoMo-Kit die Grundtypen Boolean, Text, Graphic, Real, Video, String, Sound, Integer und Symbol. Wir betrachten Boolean, Graphic, Video und Sound als komplett inkompatibel zu allen anderen Typen, es sind keine Ummodellierungen erlaubt, bei denen Variablen in diese Typen oder von diesen Typen gewandelt werden. Bei den anderen Typen können automatische Transformationen begrenzt stattfinden. Welche Modifikationen u.E. sinnvol und möglich sind, kann Tabelle 4 entnommen werden. Zusätzlich sind bei den Typwechseln noch Verallgemeinerungen entlang der Is-A-Hierarchie der Konzepte erlaubt, sowie Typwechsel zwischen zwei Konzepten, die sich nur in einzelnen kompatiblen Slot- oder Attributtypen unterscheiden.

von \ nach	Text	Real	String	Integer	Symbol
Text	-	n	j	n	n
Real	j	-	j	j	n
String	j	j	-	j	j
Integer	j	j	j	-	n
Symbol	j	n	j	n	-

Tabelle 4: Mögliche Grundtyptransformationen

Wir gehen davon aus, daß Grundtyptransformationen nicht an der Semantik einer Variablen ändern. Sollte dies erwünscht sein, muß die alte Variable entfernt werden und eine neue Variable erzeugt werden.
Neben den Part-Of-Beziehungen bei Konzeptbeschreibungen, die durch die Verwendung von Konzeptbeschreibungen als Slottypen einer anderen Konzeptbeschreibung entstehen, gibt es noch die Is-A-Beziehung (auch: Spezialisierung/Generalisierung) zwischen Typobjekten. Wir lassen die Verallgemeinerung entlang der Is-A-Hierarchie als Standardtransformation zu.
Obwohl wir den Fokus dieser Arbeit auf Ummodellierungen im Prozeßmodell richten, werden wir immer wieder auch Aspekte der Produktmodellummodellierung betrachten, da aus den obigen Ausführungen hervorgeht, daß zwischen Produkt- und Prozeßmodell Zusammenhänge bestehen, die wir bei der Konzeption eines Ummodellierungsmanagements in Betracht ziehen müssen.

5.1.3 Ummodellierungen und Design Rationales

Letztendlich sind alle Abhängigkeiten zwischen Goals, Operatoren, Assignments und Decisions, die im CoMo-Kit aus der Task-Methoden-Hierarchie und den Angaben zum Datenfluß generiert werden, dem Konzept nach Design Rationales. Das besondere an ihnen ist, daß wir ihre Semantik genau erfassen können, da dem System bekannt ist, welchen Grund es für eine Abhängigkeit gibt (z.B. Input-Beziehung zwischen einem Produkt/formalen Parameter und einer Methode). Deswegen ist es bei Ummodellierungen auch möglich zu entscheiden, welche Teile des Abhängigkeitsnetzwerkes weiterverwendet werden und welche nicht.

In Abschnitt 2.4.2 auf Seite 13 haben wir jedoch beschrieben, daß es im CoMo-Kit darüberhinaus die Möglichkeit gibt, zusätzliche Design Rationales einzufügen und existierende Design Rationales aus dem Abhängigkeitsnetzwerk zu entfernen. Für Ummodellierungen ergeben sich daraus höchst problematische Konsequenzen.
Stellen wir uns vor, die Entscheidung für eine Methode „Objektorientiertes Design“ wurde als Design Rationale einer Entscheidung (in einer anderen Task) zugunsten der Methode „Objektorientierte Implementierung“ spezifiziert. Wird nun die Methode „Objektorientiertes Design“ in irgendeiner Weise modifiziert, so können wir nicht automatisch entscheiden, ob das Design Rationale noch Bestand hat oder nicht, da wir die Semantik der Beziehung zwischen den beiden Methoden bzw. Entscheidungen nicht kennen. Genauso verhält es sich bei gelöschten systemgenerierten Design Rationales. In beiden Fällen müssen wir den Benutzer (in dem Fall den Planer, der die durch das Design Rationale begründete Entscheidung getroffen hatte) explizit nach der Beibehaltung/Löschung[1] fragen.

5.2 Anforderungen an die Verwaltung von Ummodellierungen

Wir haben bereits in der Einleitung kurz einige Anforderungen an Ummodellierungen geschildert. Erste Konkretisierungen der Art und Weise, wie wir diesen Anforderungen im CoMo-Kit genügen können, haben wir bereits in Abschnitt 2.5 vorgenommen.
Wir wollen in diesem Abschnitt die Anforderungen nochmals konkretisieren, um uns in den darauffolgenden Abschnitten an diesen Anforderungen zu orientieren.

5.2.1 Anforderungen an ein CoMo-Kit-Ummodellierungsmanagement

Stellen wir zunächst die Anforderungen nochmals dar und konkretisieren Realisierungsmöglichkeiten im CoMo-Kit:

- *Hohe WFMS-Verfügbarkeit*: Das Grundproblem für eine Verwaltung von Ummodellierungen ist es, den Benutzern eine möglichst große Flexibilität bei Modellmodifikationen zu offerieren. Gleichzeitig sollte die Planabarbeitung möglichst wenig durch die Umsetzung einer Ummodellierung (d.h. das Einbringen der Änderungen in den Scheduler) gestört werden, damit sich die betroffenen Agenten nicht gegängelt und bevormundet fühlen.
  Im CoMo-Kit wollen wir dies durch *koordinierte Interaktionen* von Modeler und Scheduler realisieren.
- *Korrektheit der Umsetzung*: Eine Ummodellierungsverwaltung muß dafür sorgen, daß die Ummodellierungen in der Abwicklungskomponente zu den richtigen Modifikationen der Abhängigkeiten führen und daß die Benutzer von der veränderten Situation in Kenntnis gesetzt werden.
  Im CoMo-Kit können wir dies mit Hilfe eines *Transformationsalgorithmus* erreichen, der ausgehend von den tatsächlich angefallenen Ummodellierungen die als

1. Die Löschung von systemgenerierten Design Rationales können wir daran erkennen, daß an den Stellen, an denen aufgrund der Modellierung Design Rationales sein müßten, keine mehr sind.

TMS-Strukturen gespeicherten Abhängigkeiten des Workflow-Modells entsprechend modifiziert.

- *Wiederverwendung von Arbeit*: Bei einer Ummodellierung sollten nicht alle Ergebnisse des durch die Ummodellierung modifizierten Modellteiles verworfen werden, sondern sie sollten (wenn möglich automatisch) wiederverwendet werden. Dies schließt auch die Wiederverwendung bereits angefangener Subprozesse nach der Umodellierung ein.
  Wir werden untersuchen, welche Eigenschaften von Tasks und Methoden im CoMo-Kit uns beim Erkennen von Wiederverwendungemöglichkeiten helfen.
- *Transparenz von Ummodellierungen:* Unter diesem Punkt fordern wir sowohl Maßnahmen zur Arbeitsersparnis (Wird durch eine Ummodellierung die Bearbeitung bestimmter Tasks überflüssig, so sollten die betroffenen Agenten *frühestmöglichst* von dieser Tatsache erfahren) als auch allgemein frühzeitige Benachrichtigung aller Agenten von für sie relevanten Ereignissen. Es ist wünschenswert, daß Erkennen von und Reagieren auf überflüssige Tasks möglichst weit zu automatisieren.
  Hier müssen wir uns erarbeiten, welche Informationen im CoMo-Kit zum Erkennen von überflüssigen Arbeiten nötig sind und welche Reaktionen genau angebracht sind.
- *Konsistenzüberwachung*: Das Workflow-Modell sollte vor und nach der Ummodellierung konsistent sein. Dies umfaßt sowohl referentielle Integrität (z.B. das Vorhandensein aller benutzten Produkttypen oder Versionskontrolle, d.h. die richtige Zuordnung von Produkttyp- und Prozeßtypversionen) als auch semantische Integrität (z.B. daß Invarianten von Methoden Spezialisierungen von Invarianten der zugehörigen Prozesse sind). Diese Konsistenzprüfungen können ebenfalls von einer Ummodellierungskomponente vorgenommen oder wenigstens unterstützt werden.
  Im CoMo-Kit existiert zur Zeit keine Konsistenzprüfungskomponente. Wir werden prüfen, wie daß Ummodellierungsmanagement die Konsistenzprüfung unterstützen kann.
- *Flexibilität bei der Spezifikation einer Ummodellierung*: Die Ummodellierung wird dem Umplaner zu Beginn zwar in groben Zügen bekannt sein, alle Auswirkungen der Ummodellierung (d.h. alle von ihr betroffenen Teile des Workflow-Modells) wird er erst im Laufe der Modellierung der Ummodellierung feststellen. Das WFMS muß ihm erlauben, bei der Modellierung über einen längeren Zeitraum (z.B. einige Tage) hinweg auch dem Aspekt der Arbeitsersparnis Rechnung tragen zu können, indem jeder an einer von der Ummodellierung betroffenen Aufgabe arbeitende Agent davon unterrichtet wird, falls eine Weiterarbeit an dieser Aufgabe durch die Ummodellierung sinnlos wird - und auch, falls die Bearbeitung wieder Sinn hat, da sich der Umplaner geirrt hat.
  Es mag manchmal sinnvoll sein, eine Modifikation des Modells in einen in Abarbeitung befindlichen Projektplan *gar nicht* einzubringen (z.B. weil die mit dem Einbringen der Änderungen verbundenen Kosten - insbesondere der Zeitaufwand - den Nutzen der Ummodellierung übertreffen). Wird ein zweiter auf diesem Workflow-Modell basierendes Projekt initialisiert, so sollten die Modifikationen dort sichtbar werden. In diesem Fall sind zeitgleich in verschiedenen Instanzen eines Workflow-Modells u.U. verschiedene Versionen von Tasks, Methoden und Produkten in Abarbeitung. Diese Versionen müssen auseinandergehalten werden; insbe-

sondere muß auf Versionskompatibilität (z.B. zwischen Methodenversion und Produkttypversion ihrer Outputs) geachtet werden.
Im CoMo-Kit müssen wir darauf achten, daß die Konzepte für eine koordinierte Interaktion (s.o.) zwischen Modeler und Scheduler dem Streben nach Flexibilität Rechnung tragen.

- *Protokollierung von Ummodellierungen*: Für den Aufbau einer „Experience Factory“, mit deren Hilfe die Konzeption und Planung von Projekten durch Verwendung bereits gesammelter Erfahrungen verbessert werden kann, sind auch statistische Daten von und über Ummodellierungen wertvoll. Eine Verwaltungskomponente sollte die Sammlung entsprechender Informationen unterstützen.
  Wir brauchen für die „Korrektheit der Umsetzung“ genaue Informationen darüber, welche Ummodellierungen vorgekommen sind. Eine Verwaltungskomponente sollte die Sammlung entsprechender Informationen unterstützen.
  Für den CoMo-Kit müssen wir festlegen, welche Informationen wie protokolliert werden sollen. Für den für die Korrektheit der Umsetzung zuständigen Transformationsalgorithmus müssen wir die Ummodellierungen mitloggen.

Wir werden bei der Beschreibung des Ummodellierungsmanagements (Abschnitt 5.4) sehen, daß wir nicht alle Anforderungen berücksichtigen konnten.

5.3 Beschreibungsansätze für Ummodellierungen im CoMo-Kit

Wir wollen in diesem Abschnitt untersuchen, welche Eigenschaften von Ummodellierungsprozessen wir zur Erarbeitung von Konzepten, die den im letzten Abschnitt beschriebenen Anforderungen genügen, benötigen.
In Anlehnung an Abschnitt 3.3 können wir zunächst sagen, daß die meisten Anforderungen des letzten Abschnittes eher mit Modellierungseinheiten oder mit modellorientierten Problematiken zu tun haben. Insbesondere muß genau spezifiziert werden können, welche Teile des Modells und welche Instanzen von einer Ummodellierung betroffen sind (Abschnitt 5.3.1). Es liegt deshalb nahe, für diese Anforderungen auch eher modellorientierte Beschreibungsansätze ins Auge zu fassen, auch wenn wir eine realitätsorientierte Alternative in unseren Überlegungen berücksichtigen werden.
Unser vorrangiges Ziel ist die Operationalisierung eines Ummodellierungsmanagements. Würden wir dabei versuchen, konsequent eher realitätsnahe Eigenschaften von Ummodellierungen zu nutzen, so müßten wir uns zudem auch auf den Facettenreichtum der Realität einlassen. Da dies uns unangemessen komplex erscheint, gehen wir in unseren Überlegungen von einer stark modellorientierten Sicht auf Eigenschaften von Ummodellierungen aus.
Da bei der Anforderung „Arbeitsersparnis“ ist von einem „frühestmöglichen“ Zeitpunkt die Rede ist, haben wir uns auch ein Zeitmodell für Ummodellierungen erarbeitet (Abschnitt 5.3.2).

5.3.1 Beschreibung des Ausmaßes von Ummodellierungen

Unter dem Ausmaß von Ummodellierungen verstehen wir die zwei Dimensionen „Betroffene Instanzen“ und „Betroffene Modellteile“. Wir haben für die Dimension „Betroffener Instan-

zen“ die Eigenschaften Modifikationstyp und Sichtbarkeit erarbeitet (Abschnitt 5.3.1.1) und diskutieren dann die Dimension „Betroffene Modellteile“ (Abschnitt 5.3.1.2). Wir werden jeweils auf Verwendungsmöglichkeiten bezüglich der Anforderungen verweisen.

5.3.1.1 Die Dimension „Betroffene Instanzen“

Wir lassen wir uns zunächst doch von den „realitätsorientierten Beschreibungsansätzen“ inspirieren und diskutieren die auf der realen Bedeutung einer Ummodellierung basierende Eigenschaft des Modifikationstyps. Dann beschäftigen wir uns mit der technischeren Eigenschaft der Sichtbarkeit von Ummodellierungen.

Der Modifikationstyp von Ummodellierungen

Wir können versuchen, die Entscheidung darüber, welche Instanzen eines Workflowmodells daran festzumachen, welche Tragweite eine Ummodellierung von ihrer Semantik her hat. Wir können erwarten, daß die Korrektur eines Modellierungsfehlers beispielsweise in allen laufenden Workflow-Instanzen korrigiert werden sollte, während die Spezifikation einer neuen Methode keine direkten Auswirkungen auf die Instanzen hat. Wir müssen jedoch feststellen, daß diese Erwartung trügt.

Unterstellen wir, daß Ummodellierungen immer der Verbesserung eines Workflow-Modells dienen, so sind Ummodellierungsprozesse immer entweder die Behebung einer Fehlmodellierung (*Debug*-Ummodellierung) oder eine Effizienzsteigerung durch das Einbringen neuer Alternativen in das Modell (*Improve*-Ummodellierung). Wir wollen diese Eigenschaft als *Modifikationstyp* (mit den Ausprägungen Debug/Improve) von Ummodellierungsprozessen festhalten. Der Modifikationstyp repräsentiert in gewisser Weise auch die Intention einer Ummodellierung.

Standardmäßig werden also Improve-Ummodellierungen nicht unbedingt in allen Instanzen des Workflow-Modells sichtbar, während dies bei Debug-Ummodellierungen (als Abweichungen und Korrekturen) in der Regel so sein wird. Es lassen sich jedoch Gegenbeispiele für diese Regeln finden. So kann es wünschenswert sein, eine Debug-Ummodellierung nicht in die ablaufenden Instanzen weiterzupropagieren , falls die mit der Ummodellierung verbundenen Kosten in der realen Welt den Nutzen der Ummodellierung nicht rechtfertigen. Und auch bei Improve-Ummodellierungen wird u.U. bereits zum Zeitpunkt der Modellierung der Ummodellierung feststehen, daß sie in allen Instanzen sichtbar wird, z.B. wenn die durch eine deutlich effizientere Modellierung mögliche Zeitersparnis es nahelegt, statt der alten Methode - trotzdem sie sich eventuell bereits in Abarbeitung befindet - die neue Alternative im Projekt einzusetzen.

Wir können also nicht davon ausgehen, allein aufgrund des Modifikationstyps einer Ummodellierung auf deren Auswirkungen auf Workflowmodell-Instanzen Rückschlüsse ziehen zu können. Trotzdem werden wir für den Modifikationstyp Verwendung finden (s.u.).

Die Sichtbarkeit von Ummodellierungen

Mit der Sichtbarkeit von Ummodellierungen können wir direkt bezeichnen, welche Instanzen des Workflow-Modells von einer Ummodellierung betroffen sind. Dazu formalisieren wir zunächst einmal die Beziehung Modell-Instanz (siehe Abb. 9).

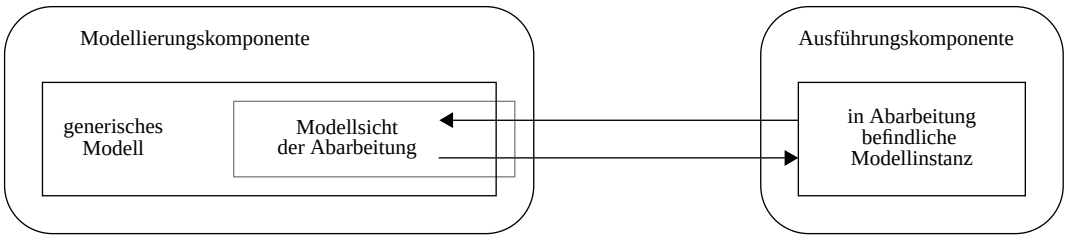


Abb. 9: Modellebenen bei der Workflow-Abarbeitung

Bei einem in Abarbeitung befindlichen WFMS kann man allgemein drei Modellebenen unterscheiden[1] . Die Modellinstanz enthält alle zur Abarbeitung notwendigen Informationen einschließlich der bisher erarbeiteten Ergebnisse bzw. Variablenbelegungen sowie der Zustände der Workflows. Die Modellsicht enthält einen für das in der Ausführungskomponente abgearbeitete Projekt relevanten Ausschnitt des generischen Modells; u.U. mit Sichtspezifischen Modifikationen. Das generische Modell enthält die abstrakten Beschreibungen (Produkt-, Aufgaben- und Methodentypen inklusive der zwischen ihnen bestehenden Beziehungen, Agentenmodell, etc.). Dem (aktuellen) generischen Modell mag noch eine Erfahrungsdatenbank angeschlossen sein, in der auch „alte“ Modellversionen aus Zeiten vor früheren Ummodellierungen, eventuell mit zusätzlichen statistischen Daten über Projektverläufe und Ummodellierungen, gespeichert sind.

Wir können Ummodellierungen nun danach einteilen, wo in diesen Modellebenen sie sichtbar werden sollen:

- *nur Modellinstanz*: In diese Bereich fallen Umplanungen wie Ressourcen-Reallokationen oder Revision von Planungsentscheidungen, die keine Ummodellierung erfordern. Sie sind nicht Gegenstand unserer Betrachtungen.
- *Modellsicht & Modellinstanz*: Hierhinein fallen Ummodellierungen, die projektlokale Ursachen haben, allerdings auch in den Ablaufplan einfließen müssen. Das kann z.B. das Hinzufügen einer Subtask (zu einer Methode) sein, die eine normalerweise zur Lösung der Aufgabe nicht notwendige Information beschafft, die von einer anderen Subtask in diesem speziellen Fall jedoch benötigt wird. Wir nennen eine solche Ummodellierung eine *Abweichung*.
- *Generisches Modell, Modellsicht & Modellinstanz*: Dieser Fall umfasst z.B. die Behebung wirklicher Modellierungsfehler, wie die Modellierung einer Methode, die die ihr zugeordnete Aufgabe so nicht lösen kann und deshalb ummodelliert werden muß. Solche Ummodellierungen nennen wir *Korrektur.*
- *nur Modellsicht*: Dies betrifft Ummodellierungen, die zum einen lokal begründet sind und zum anderen nicht unbedingt in den Ablaufplan integriert werden müssen. Hängt z.B. die Dekomposition einer Aufgabe in starkem Maße von projektlokalen Gegebenheiten ab, so kann sie im generischen Modell nicht beschrieben werden und wird erst während der Projektabarbeitung verfeinert. Werden im Rahmen dieser Ummodellierung zwei mögliche Methoden für diese Aufgabe spezifi-

1. Dieses Drei-Ebenen-Modell kommt in ähnlicher Form auch z.B. in EPOS und in SPADE vor, kann also als relativ allgemeingültig angesehen werden.

ziert, so ist für keine der Methoden die Umsetzung in den Plan garantiert. Solche Ummodellierungen nennen wir also *Verfeinerungen*.

- *Generisches Modell & Modellsicht*: Ummodellierungen, die nicht unbedingt in den Plan eingebracht werden müssen, wie z.B. die Modellierung einer neuen, zusätzlichen Methode fallen in diese Kategorie. Wir reden dann von einer *Optimierung*.
- *nur Generisches Modell*: Es kann sinnvoll sein, eigentlich notwendige Korrekturen am Workflow-Modell in der konkreten Projektabarbeitung „kosmetisch“ in einem eigenen Ummodellierungsprozeß zu umgehen (z.B. die „lokale Lösung“ aus dem Beispiel aus Abschnitt 3.2). In diesem Fall wird bei der eigentlichen Modellberichtigung (in einem anderen Ummodellierungsprozeß) nur das generische Modell geändert. Diese „Offline-Ummodellierungen“ werden von uns hier nicht betrachtet.

Natürlich können sich Ummodellierungen, die zunächst nur in der Modellsicht und nicht in der Modellinstanz sichtbar waren, durch eine entsprechende Umplanung in der Modellinstanz (z.B. Abbruch einer Methode und Auswahl einer neuen Methode) doch noch auf den Ablaufplan auswirken. Werden Generisches Modell und Modellsicht nicht gleichzeitig geändert, so entsteht eine *tolerierte Inkonsistenz* in der Modellsicht. Inkonsistenzen zwischen Modellinstanz und Modellsicht sind nicht erlaubt; es ist auch kein Grund dafür denkbar.
Diese Eigenschaft der *Sichtbarkeit* von Ummodellierungen impliziert letztendlich auch deren Persistenz: Nur Ummodellierungen, die im generischen Modell sichtbar sind, sind auch persistent[1].
In Sinne obiger Ausführungen wollen wir nun die in Abschnitt 3.1 auf Seite 19 beschriebenen Ummodellierungsbeispiele klassifizieren:

- UP2-4 beeinflußen nur die laufende Modellinstanz und sind keine Ummodellierungen.
- UP5 ist eine Korrektur und Debug-Ummodellierung.
- UP6 entspricht letztendlich UP5, da Agenten für diese Methode für inkompetent erklärt werden. Möglicherweise wurde auch die Precondition der Methode geändert, was dann zwar eine komplett andere Modellierungsvariante darstellt, die sich allerdings bezüglich Sichtbarkeit und Modifikationstyp nicht von UP5 unterscheidet.
- Bei UP7 ist nicht klar, ob das generische Modell (Optimierung) oder nur die Modellsicht (Verfeinerung) modifiziert wird. Klar ist, daß es eine reine Improve-Ummodellierung ist und das die Modellinstanz zunächst nicht betroffen ist.
- UP8 hat bzgl. der Sichtbarkeit ähnliche Probleme wie UP7, jedoch wissen wir hier, daß auch die Modellinstanz betroffen sein wird (also Korrektur oder Abweichung). Es handelt sich um eine Debug-Ummodellierung, da die alte Methode offensichtlich in irgendeiner Form falsch war.
- UP9 ist ein etwas schwieriger gelagerter Fall. Seine Einteilung ist davon abhängig, welche Semantik der neue Input (das fehlende Datum) hat. Ist es prinzipiell zur Erreichung des Zieles nötig, so wird die Ummodellierung auch im generischen Modell sichtbar; gibt es lediglich projektspezifische Gründe für die Notwendigkeit dieser Information, so ist dies nicht der Fall. Stellt man jedoch fest, daß die konkrete Ausprägung der Information weitreichende Folgen für die Dekomposition

1. Natürlich erhalten auch alle anderen Ummodellierungen eine gewissen Grad an Persistenz, wenn sie z.B. in einer Erfahrungsdatenbank gespeichert werden.

dieser Aufgabe hat, so muß man die Information als Planungsinput einstufen, und entsprechend UP10 behandeln. Darüberhinaus ist UP9 eine Debug-Ummodellierung.

- UP10 beschreibt die Hinzunahme eines für die Planung (d.h. für die Dekomposition einer Aufgabe in Subaufgaben) notwendigen Inputs. Auch dies ist eine Debug-Ummodellierung, da die alte Modellierung der entsprechenden Aufgabe modifiziert wurde und auch mindestens die der darüberliegenden Methode. Alle drei Ebenen sind betroffen (Korrektur).
- UP11 ist eine Improve-Ummodellierung (die alte Art, diese Aufgabe zu lösen, wird nicht ungültig!) und betrifft lediglich Modellsicht und Modellinstanz (also eine zur Abweichung gewordene Verfeinerung). Die Änderungen an der Art der Aufgabenausführung sind projektspezifisch. Würde diese Änderung häufiger vorkommen, so würde sich eine Ausweitung auf das generische Modell anbeiten.
- UP12 betrifft ebenfalls Sicht und Instanz. Dieser besondere Fall liegt erheblich komplizierter: Die Ummodellierung ist die Einführung einer neuen Subtask, die das Spezifikationsdokument ändert. Darüberhinaus werden Design Rationales modifiziert, da durch die Ummodellierung eine domänenspezifische Konsistenzbeziehung verletzt (Die Spezifikation wurde mißachtet!) wurde, die in irgendeiner Form wiederhergestellt werden sollte.
- Die Improve-Ummodellierung UP13 betrifft in jedem Fall nicht das generische Modell. Ob sie die Modellinstanz betrifft hängt davon ab, ob bereits zu Zeitpunkt der Modellierung bekannt ist, daß die Ummodellierung auch in den Plan eingebracht wird. Werden z.B. zwei Verfeinerungsmöglichkeiten modelliert, so steht nicht fest, welche der beiden letztendlich umgesetzt wird.

Einsatz von Modifikationstyp und Sichtbarkeit

Die beiden Eigenschaften Modifikationstyp und Sichtbarkeit sind leicht spezifizierbar. Sie sind dabei behilflich, die Art der Auswirkungen einer Ummodellierung auf den in Abarbeitung befindlichen Ablaufplan des WFMS festzustellen:

- *Die Sichtbarkeit und der Modifikationstyp einer Ummodellierung beeinflußen ihre Rücksetzbarkeit in den laufenden Modellinstanzen.* Wir haben grundsätzlich bei Umsetzung einer Ummodellierung im Abhängigkeitsgraph des Schedulers die Wahl, alle Strukturen, die sich auf den Modellzustand vor der Ummodellierung beziehen, zu löschen oder sie beizubehalten, um die Ummodellierung u.U. schnell rückgängig machen zu können. Bei Improve-Ummodellierungen brauchen wir das nicht, da kein altes Wissen invalidiert wurde und somit alte und neue Modellteile nebeneinander existieren können. Debug-Ummodellierungen, die im generischen Modell sichtbar werden, stufen wir als verläßlich ein, so daß eine Rücksetzbarkeit nicht unbedingt gegeben sein muß. Für lokal sichtbare Debug-Ummodellierungen sollten wir, da die projektspezifischen Gründe für die Ummodellierung u.U. wieder wegfallen könnten, die Möglichkeit des Zurückziehens einer Ummodellierung offenhalten, d.h. insbesondere, daß die inaktiven Abhängigkeiten des „alten“ Version im Plan erhalten bleiben. Dieser Aspekt wird sich im Transformationsalgorithmus für das Abhängigkeitsnetzwerk niederschlagen.
- *Die Sichtbarkeit einer Ummodellierung beeinflußt, welche Projekte betroffen sind.* Wird nur die Modellsicht geändert, so ist nur das entsprechende Projekt betroffen,

bei einer Modifikation des generischen Modells hingegen sind zunächst einmal alle laufenden Projekte betroffen.[1]

- *Die Sichtbarkeit einer Ummodellierung beeinflußt die Konsequenz ihrer Durchsetzung.* Während bei einer Optimierung/Verfeinerung offensichtlich keine Notwendigkeit bestehen muß, die Ummodellierung auch im konkreten Projekt anzuwenden, verlangt eine Korrektur/Abweichung nach einer strengeren Behandlung des existierenden Plans, z.B. bezüglich der Benachrichtigung der von der Ummodellierung betroffenen Agenten (z.B. unter dem Aspekt der Arbeitsersparnis).

Während der Modifikationstyp also nur noch für den Transformationsalgorithmus Bedeutung hat wird uns die Sichtbarkeit als wichtige Eigenschaft von Ummodellierungen häufiger begegnen.

5.3.1.2 Die Dimension „Betroffene Modellteile“

In diesem Abschnitt erarbeiten wir uns ein methodenbasiertes Denkmodell für Ummodellierungen, mit dem wir beschreiben können, welcher Teil des Prozessmodells in die Ummodellierung verwickelt war. Außerdem beschreiben wir, warum wir die Unterscheidung zwischen weichen und harten Änderungen aus Abschnitt 3.3 übernehmen.

Ummodellierungen als Methodendefinition

Betrachtet man sich das Beispiel aus Abschnitt 3.2 auf Seite 20 näher, so fällt auf, daß die dort spezifizierten Ummodellierungen - sieht man von Änderungen des Produktmodells ab - auf eine gewisse Weise lokal bleiben, da alle Änderungen in der Methode „Small-Project-Design“ stattfinden. Aufgrund des von Tasks und Methoden in einem Modell aufgebauten Und-Oder-Baums kann man festhalten, daß alle Änderungen entweder in irgendeiner Methode lokal sind oder die Wurzeltask verändern. Ummodellierungen an der Wurzeltask unterliegen prinzipiell den gleichen Problematiken wie alle anderen Tasks auch (siehe dazu Abschnitt 5.4.5.1). Werden darüber hinaus weitere Teile der Dekompositionshierarchie modifiziert, so gelten die folgenden Anmerkungen entsprechend. Da die eventuellen Folgen von Ummodellierungen bei der Wurzeltask ziemlich gravierend sein können (Rücksetzen des gesamten Projektes), sollten die Ummodellierungen dort selbstverständlich wohlüberlegt sein.
Gibt es eine Methodenebene, auf der alle Modifikationen als lokal (d.h. methodenintern auftretend) betrachtet werden können, so kann man eine Ummodellierung als Spezifikation einer neuen Methode auffassen. Der Unterschied zur üblichen Methodenspezifikation ist, daß man sich u.U. hier *sicher* sein kann (wenn die Ummodellierung in der Modellinstanz sichtbar sein soll), daß diese Methode später auch in den aktuellen Plan umgesetzt wird, und daß die u.U. momentan in Abarbeitung befindliche Methode in der vorliegenden Form zurückgezogen wird. Bleiben wir bei diesem Bild, so ergeben sich weitere Unterschiede zur üblichen Methoden(neu-)-spezifikation:

- Die neue Methode wird nicht völlig neu spezifiziert, sondern wir gehen von der aktuellen Version der Methode aus und modifizieren diese.
- Aus dem Wissen heraus, daß die neue Version in den Plan umgesetzt wird, können wir u.U. die Tasks bestimmen, deren Bearbeitung durch die neue Version überflüssig werden.

---

1. Hier kann man sich noch vorstellen, daß es Sinnmachen könnte, für jede einzelne laufende Modellinstanz über die Übernahme von Ummodellierungen zu entscheiden.

- Wir können ebenfalls bestimmen, welche Subtasks des Planes von den Modifikationen unbeeiträchtigt bleiben.

Im CoMo-Kit haben zur gleichen Task gehörende Methoden disjunkte Subtaskmengen[1]. Betrachten wir uns die Menge der Ummodellierungen, die die „alte“ Methodenversion in die „neue“ überführen, so können wir *mit Sicherheit* bestimmen, daß bestimmte Subtasks der alten Version und der neuen *identisch* sind. Dies hat zur Folge, daß wir bei der Umsetzung der Ummodellierungen die entsprechenden Teile des Abhängigkeitsgraphen der alten Methodenversion in die neue übernehmen können (Wir gehen darauf genauer in Abschnitt 5.4.3 ein).
Es sei jedoch an dieser Stelle darauf hingewiesen, daß schon hinterfragt werden sollte, inwieweit das „automatische“ Übernehmen von eigentlich in einem anderen Entscheidungskontext getroffenen Delegierungs- und Dekompositionsentscheidungen in einen modifizierten Abhängigkeitsgraph statthaft ist. Es bildet sich hier ein Konflikt zwischen der „Echtheit“ einer Entscheidung (Sie wurde in einem bestimmten Kontext gefällt und es stellt sich die Frage, ob sie auch noch in einem veränderten Entscheidungskontext optimal ist - schliesslich könnte sich der Planer jetzt anders entscheiden wollen) und praktischen Gesichtspunkten (Arbeitsersparnis durch die Übernahme alter Strukturen und Produkte). Wir gehen jedoch davon aus, daß durch die Invariante einer Task (zu der auch die Invarianten der übergeordneten Tasks/Methoden gehören!) *alle* für die Bearbeitung dieser Task relevanten Kontextaspekte zugesichert werden, so daß eine für die (Nicht-)Übernahme „alter“ Entscheidungen bei einer Ummodellierung relevante Kontextänderung sich auch in einer Modifizierung der Invariante niederschlagen muß.
Wir wollen darauf hinweisen, daß unsere Verwendung der Invarianten als den Entscheidungskontext garantierende Eigenschaft durchaus nicht unproblematisch ist. Das Problem liegt darin, daß z. Zt. die genaue Semantik von Invarianten im CoMo-Kit noch nicht feststeht. Wir nehmen eine „externe“ Ausrichtung von Invarianten an, da sie uns gewisse äußere Bedingungen zusichern sollen. Es sind jedoch auch „interne“ Ausrichtungen von Invarianten denkbar. So können Invarianten z.B. auch zur Formulierung von Constraints über lokale Variablen von Methoden benutzt werden oder um temporale Eigenschaften von Methoden und Tasks zuzusichern („Dauer < 9 Tage“). Diese Prädikate gehören nicht unbedingt zum Entscheidungskontext der Decision, eine Modifikation dieser Bedingungsteile würde jedoch dieselben Folgen wie die Änderung des Entscheidungskontextes haben. Will man dies verhindern, so bietet sich an, die Invariante in entsprechende Teilbedingungen („interne“ und „externe“) zu zerlegen und getrennt zu verwalten.

Die Methode, die alle Modifikationen umfaßt, nennen wir den *Änderungskontext*. Die Task, zu der die Ummodellierung nach diesem Denkansatz als „neudefinierte“ Methode gehören würde, nennen wir den *Ummodellierungsanker.*

Harte vs. Weiche Modifikationen

Wir wollen noch auf eine weitere Eigenschaft von Ummodellierungsprozessen aus der Literatur, nämlich die Unterscheidung zwischen *verhaltensorientierten (weichen) Änderungen* (z.B. Änderung einer Invariante) und *strukturellen (harten) Änderungen* (z.B. Einfügen einer neuen Subtask in eine Methode) im Prozeßmodell, eingehen. Das wesentliche an dieser Unterscheidung ist die Tatsache, daß weiche Änderungen nur Auswirkungen innerhalb des Prozeßmodells haben, während sich harte Änderungen auch auf das Produktmodell auswirken, weil eine Umstrukturierung der Prozeßtypen auch eine Umstrukturierung der Produkttypen nach sich ziehen kann. Aufgrund dieser Beziehung zwischen Produkt- und Prozeßmodell ist auch der

1. Wobei diese Modellierungsweise keinen faktischen Grund hat, sondern aus historischen Gründen so besteht.

umgekehrte weg möglich, daß eine Produktmodellmodifikation Prozessmodellmodifikationen erzwingt, jedoch sind im Produktmodell (da Produkte kein „Verhalten“ haben) alle Änderungen „hart“.

Einsatz der Beschreibungen der betroffenen Modellteile

Die Diskussion über Invarianten im Abschnitt „Ummodellierungen als Methodendefinition“ wird auch für die Anforderung der Wiederverwendung von Arbeit (Abschnitt 5.4.3) eine Rolle spielen. Die methodenbasierten Sicht auf Ummodellierungen werden wir bei der Befriedigung der Anforderungen „Flexibilität“ und „Hohe WFMS-Verfügbarkeit“ ebenso wiederfinden, wie beim Transformationsalgorithmus für das Abhängigkeitsnetzwerk.
Die Unterscheidung zwischen harten und weichen Modifikationen benötigen wir immer dann, wenn es um Übersprechungen zwischen Änderungen am Produkt- und am Prozessmodell geht (z.B. bei der Diskussion der Modellkonsistenz in Abschnitt 5.4.1).

5.3.2 Ein Zeitmodell für Ummodellierungen

Um spezifizieren zu können, wie das System sich bei Ummodellierungsprozessen verhalten soll, benötigen wir einen Zeitbegriff, der die relevanten Zeitpunkte einer Ummodellierung erfasst. Wir verwenden für die Phasen bei Ummodellierungsprozessen das in Abb. 10 dargestellte Zeitstrahlmodell (Wir werden uns im Weiteren immer über die untenstehenden Kürzel auf die jeweiligen Events beziehen.), welches eine Ummodellierung zeitlich in eine Modellierungsphase und eine Umsetzungsphase einteilt. Mit „Umsetzung“ ist auch hier das Einbringen der Änderungen in den aktuellen Ablaufplan gemeint.

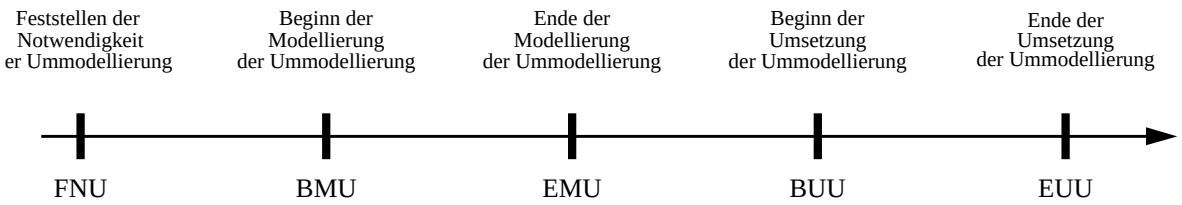


Abb. 10: Zeitstrahlmodell der Ummodellie-

Es kann durchaus vorkommen, daß die Notwendigkeit einer Ummodellierung zu irgendeinem Zeitpunkt vor BUU nicht mehr besteht (z.B. beim Abbruch der Ummodellierung durch den Umplaner). Auch diese Erkenntnis muß unter bestimmten Umständen betroffenen Agenten bekanntgemacht werden und wird deshalb als Event *Ende der Notwendigkeit einer Umplanung (ENU)* geführt.
Diese Events kann man auf das Gesamtmodell, (dann repräsentieren sie den gesamten Ummodellierungsprozeß) auf die CoMo-Kit-Modellierungseinheiten, und auf die atomaren Ummodellierungsoperatoren (weniger sinnvoll, da die Modellierungs- und Umsetzungsevents jeweils zusammenfallen) anwenden.
Wir benutzen diese Events vor allem in der Modellierungsphase zur Spezifikation des Auslösungszeitpunktes einer Reaktion auf eine Umplanung (insbesondere bei der Anforderung der Transparenz von Ummodellierungen).

5.4 Ein Ummodellierungsmanagement für CoMo-Kit

Wir wollen uns auch hier bei der Darstellung des von uns entwickelten Ummodellierungsmanagements wieder an den in Abschnitt 5.2 formulierten Anforderungen orientieren. Zunächst erläutern wir, warum wir die Anforderung „Konsistenzüberwachung“ nicht bzw. nur teilweise befriedigt haben (Abschnitt 5.4.1). Dann beschreiben wir die Aufzeichnung von Ummodellierungen (Abschnitt 5.4.2). Für die Wiederverwendung von Arbeit stellen wir das Konzept der „Shared Subgoal Spaces“ vor (Abschnitt 5.4.3). Wir betrachten Notifikationskonzepte der Transparenz von Ummodellierungen (insbesondere bei der Vermeidung überflüssig gewordener Arbeit) in Abschnitt 5.4.4 und beschreiben einen Algorithmus zur korrekten Umsetzung der CoMo-Kit-Abhängigkeitsstrukturen im Kontext der erfolgten Ummodellierung (Abschnitt 5.4.5). Abschließend (Abschnitt 5.4.6) versuchen wir den Anforderungen einer hohen WFMS-Verfügbarkeit und Flexibilität mit dem Konzept einer Ummodellierungstransaktion zu genügen, die auch die Klammer für die Operationalisierung der vorher beschriebenen Konzepte bildet.

5.4.1 Konsistenzüberwachung im Ummodellierungsmanagement

Grundsätzlich kann man in allen Bereichen, in denen Modellierungen von realen Vorgängen oder Sachverhalten benutzt werden, zwei Arten der Konsistenz unterscheiden. Zum einen die Konsistenz zwischen den realen Verhältnissen und dem Modell, zum anderen die Konsistenz im Modell selbst. Wir betrachten hier letzteren Konsistenzbegriff.
Eine Modifikation an einer Modellierungseinheit kann Modifikationen an anderen Modellierungseinheiten erforderlich machen, um das Workflow-Modell konsistent zu halten. So kommt es zu Ummodellierungskaskaden durch konsistenzkritische Auswirkungen von Ummodellierungen:

- *Modifikation eines Prozeßtyps(Task)*: Beeinflußt häufig auch die zu diesem Prozeßtyp gehörenden Methoden sowie die Methoden, in denen er vorkommt. Wird etwas an den Parametern modifiziert, so können auch deren Produkttypen in die Ummodellierung verwickelt sein.
- *Modifikation einer Methode*: Beeinflußt häufig auch den Prozeßtyp der Methode sowie in ihr vorkommende Subprozeßtypen. Wird etwas an den Parametern modifiziert, so können auch deren Produkttypen in die Ummodellierung verwickelt sein.
- *Modifikation eines Produkttyp*s: Beeinflußt alle Prozesse und Methoden, die Parameter dieses Produkttyps haben.

Durch diese Kaskadierung entstehen u.U. vielfältig verschachtelte und verknüpfte Ummodellierungen.
Auf der Ebene der atomaren Ummodellierunsoperatoren läßt sich auch der Begriff der Modellkonsistenz durch die möglichen Ummodellierungskaskaden genauer beschreiben. In den Tabellen 5, 6 und 7 versuchen wir einen Überblick zu geben, welche direkten Folgen die Anwendung eines Ummodellierungsoperators nur zur Erhaltung der Modellintegrität[1] nach sich ziehen kann. Sie machen zum Einen die Komplexität deutlich, mit der eine Komponente zur Konsistenzüberprüfung bzw. -erhaltung zurechtkommen muß. Zum Anderen sind sie Indi-

1. Die Modellintegrität ist z.B. dann verletzt, wenn ein konsumierter Input nirgendwo als Output produziert wird, die Invarianten von Subtask und Supertask einander wiedersprechen oder einer Konzeptbeschreibung ein Slot zugefügt wird, der nirgendwo durch eine Task mit einem Wert belegt wird.

Ummodellierungsoperator	mögliche konsistenzkritische Auswirkungen
Plan-Input-added	Parameter-Task-Relation-added(Supermethod)
Plan-Input-retracted	Parameter-Task-Relation-Retracted(Supermethod)
Plan-Input-Type-modified	Formal-Parameter-Type-Modified(Supermethod); *any product modification*
Exec-Input-added	Parameter-Task-Relation-added(Supermethod); Input-added(Method)
Exec-Input-retracted	Formal-Parameter-Retracted(Supermethod); Input-retracted(Method)
Exec-Input-Type-modified	Formal-Parameter-Type-Modified(Supermethod); Input-Type-modified(Method); *any product modification*
Output-added	Formal-Parameter-Added(Supermethod); Output-added(Method)
Output-retracted	Formal-Parameter-Retracted(Supermethod); Output-retracted(Method)
Output-Type-modified	Formal-Parameter-Type-Modified(Supermethod); Output-Type-modified(Method); *any product modification*
Method-added	Plan-Input-added(Task); Exec-Input-added(Task)
Method-retracted	Plan-Input-retracted(Task); Exec-Input-retracted(Task)
Agent-added	Agent-added(Method)
Agent-retracted	Agent-retracted(Method)
Precondition-modified	Precondition-modified(Method); Invariant-modified(Supermethod)
Invariant-modified	Invariant-modified(Supermethod); Invariant-modified(Method)
Postcondition-modified	Postcondition-modified(Supermethod); Postcondition-modified(Method); Invariant-modified(Supermethod)

Tabelle 5: Ummodellierungskaskaden bei Prozeßtypmodifikation

Einheit	Ummodellierungsopertor	mögliche konsistenzkritische Auswirkungen
Product	Product-Type-added	Formal-Parameter-Type-Modified(aMethod); Slottype-modified(aProduct); Output-Type-modified(aTask); Output-Type-modified(aMethod); Plan-Input-Type-modified(aTask); Exec-Input-Type-modified(aTask); Input-Type-modified(aMethod)
	Product-Type-retracted	*siehe Product-Type-added*
	Slot-added	Product-Type-added; *siehe Product-Type-added*
	Slot-retracted	Product-Type-retracted; *siehe Product-Type-added*
	Slottype-modified	Product-Type-added; *siehe Product-Type-added*
	Slotcardinality-modified	*siehe Product-Type-added*
	Attribute-added	Product-Type-added; *siehe Product-Type-added*
	Attribute-retracted	Product-Type-retracted; *siehe Product-Type-added*
	Attribute-Type-Modified	*siehe Product-Type-added*

Tabelle 7: Ummodellierungskaskaden bei Produktmodifikation

kator dafür, daß es tatsächlich für einen Umplaner schwierig ist, die Auswirkungen einer Ummodellierung einer Modellierungseinheit abzuschätzen. Gerade die Wechselwirkungen

Ummodellierungsoperator	mögliche konsistenzkritische Auswirkungen
Input-added	Exec-Input-added(Supertask)
Input-retracted	Exec-Input-retracted(Supertask)
Input-Type-modified	Exec-Input-Type-modified(Supertask); *any product modification*
Output-added	Output-added(Supertask)
Output-retracted	Output-retracted(Supertask)
Output-Type-modified	Output-Type-modified(Supertask); *any product modification*
Task-added	Formal-Parameter-Added(Method); Parameter-Task-Relation-added(Method); Task-Parameter-Relation-added(Method); Input-added(Method); Output-added(Method)
Task-retracted	Formal-Parameter-Retracted(Method); Parameter-Task-Relation-retracted(Method); Task-Parameter-Relation-retracted(Method); Input-retracted(Method); Output-retracted(Method)
Formal-Parameter-Added	Parameter-Task-Relation-added(Method); Task-Parameter-Relation-added(Method); Slot-added(aProduct)
Formal-Parameter-Retracted	Parameter-Task-Relation-retracted(Method); Task-Parameter-Relation-retracted(Method); Slot-retracted(aProduct)
Formal-Parameter-Type-Modified	Output-Type-modified(Subtask); Plan-Input-Type-modified(Subtask); Exec-Input-Type-modified(Subtask); *any product modification*
Parameter-Task-Relation-added	Exec-Input-added(Subtask); Plan-Input-added(Subtask); Formal-Parameter-Added(Method)
Parameter-Task-Relation-retracted	Exec-Input-retracted(Subtask); Plan-Input-retracted(Subtask); Formal-Parameter-Retracted(Method)
Task-Parameter-Relation-added	Output-added(Subtask); Formal-Parameter-Added(Method)
Task-Parameter-Relation-retracted	Output-retracted(Subtask); Formal-Parameter-Retracted(Method)
Precondition-modified	Precondition-modified(Subtask); Invariant-modified(Supertask)
Invariant-modified	Invariant-modified(Subtask); Invariant-modified(Supertask)
Postcondition-modified	Postcondition-modified(Subtask); Postcondition-modified(Supertask); Invariant-modified(Supertask)

Tabelle 6: Ummodellierungskaskaden bei Methodenmodifikation

zwischen dem Produktmodell und Prozeßmodell bergen mannigfaltige Möglichkeiten zu einer Kaskadierung einer Ummodellierung. Dazu ein Beispiel.
Man stelle sich vor, in einem Workflow wird ein Report erstellt, der normalerweise so kurz ist, daß kein Stichwortindex erforderlich ist. Wird dies in einem Workflow jedoch notwendig (dort würde dann der Subworkflow „Stichwortindex erstellen“ eingefügt werden), so würde dies zu einem neuen Slot „Stichwortindex“ im Produkttyp „Report“ führen. Die würde auch andere Workflows, die zwar diesen Produkttyp verwenden, aber sachlich nicht die Notwendigkeit eines Stichwortindizes beinhalten, in die Ummodellierung verwickeln.
Ein erster Ausweg wäre hier, im Rahmen eines Versionierungskonzeptes den modifizierten Produkttyp als neue Version zu führen. Dies würde immerhin den Fall, daß es sich um den Pro-

dukttyp eines internen Formalen Parameters einer Methode handelt, handhabbar machen, da die Produktummodellierung nicht außerhalb der Methode sichtbar wird. Handelt es sich jedoch um den Produkttyp eines Output-Parameters, so „versteht“ der diesen Output konsumierende Workflow diesen modifizierten Input-Parameter nicht mehr. Entweder muß der konsumierende Workflow für die modifizierte Version angepaßt (also auch modifiziert) werden, oder es muß eine geeignete Typkonvertierung stattfinden, die die Instanz der neuen Produkttypversion in ein für den konsumierenden Workflow sinnvolles Format bringt.
Diese Übersprechungen zwischen Produktmodell und Prozeßmodell (und umgekehrt) waren wohl der Grund dafür, daß im EPOS-Projekt (siehe [NC94]) strukturelle („harte“) Änderungen am Prozeßmodell zunächst nicht erlaubt wurden. Aufgrund der Komplexität dieses Gebietes (und der Tatsache, daß im CoMo-Kit auch zur Zeit keine Konsistenzprüfungen stattfinden) haben wir in dieser Arbeit darauf verzichtet, Konsistenzprüfungen in das Ummodellierungssystem zu integrieren. Wir werden mit der Kaskadierungsproblematik allerdings auch an anderer Stelle zu kämpfen haben.

5.4.2 Protokollierung der Ummodellierung

Eine Anforderung an das Ummodellierungsmanagement ist, während der Modellierungsphase der Ummodellierung auf der Basis der atomaren Ummodellierungsoperatoren den Nettoeffekt der Ummodellierung festzustellen, da diese Menge der Ummodellierungsoperatoren die Grundlage für den Transformationsalgorithmus bei der Umsetzung sind[1]. Prinzipiell haben wir zwei Methoden, diesen Nettoeffekt festzustellen: Ergebnisvergleich (Modell vor und nach der Ummodellierung vergleichen) und Mitloggen der Ummodellierungen.
Ergebnisvergleich hat den Vorteil, keine überflüssigen Modifikationen aufzuzeichnen, insgesamt entsteht während der Modellierung kein zusätzlicher Aufwand im Modeler. Vorteil des Mitloggens ist, daß die Ummodellierung auch in dem Moment festgestellt wird, in dem sie auftritt, was besonders für die in Abschnitt 5.4.4 beschriebenen möglichen Automatismen interessant ist. Zudem kann das Mitloggen „nebenbei“ geschehen, während der Ergebnisvergleich am Ende der Modellierungsphase eine gewisse Zeit in Anspruch nimmt. Wir haben uns für das Mitloggen der atomaren Ummodellierungen entschieden.

Mitzuloggende Informationen

Die Aktionen des Umplaners im Modeler werden über die verwendeten atomaren Ummodellierungsoperatoren mitgeloggt, die wir in Abschnitt 5.1.1 eingeführt haben. In Tabelle 14 führen wir auf, welche Informationen (d.h. welche zusätzlichen Parameter von atomaren Ummodellierungsoperatoren) insgesamt gespeichert werden müssen, um ein vollständiges Bild der vorgenommenen Ummodellierungen zu erhalten.
Dabei müssen neben der Art der Ummodellierung auch Informationen über die an der Ummodellierung beteiligten Modellierungseinheiten gespeichert werden, damit wir wissen, was ummodelliert wurde. Bei allen Modifikationsoperatoren (...-Modified) sind zudem alter und neuer Wert zu speichern. Bei den Bedingungskonstrukten ist z. Zt. noch nicht klar, wie sie gespeichert werden. Wir gehen hier von einer rein textuellen Repräsentation der Bedingungen aus, die dem Benutzer viele Freiheitsgrade läßt und können deshalb die Ummodellierung auch nicht genau erfassen, weshalb wir zur Feststellung der Modifikation letztendlich nur einen Stringvergleich durchführen können. Wird als Repräsentation z.B. eine (konjunktive oder dis-

1. Diese mitgeloggten Ummodellierungsoperatoren können in späteren Projektphasen für ein Versionierungskonzept oder für eine Erfahrungsdatenbank nützlich sein.

Format: Ummodellierungsoperator (Zusatzinformationen)
Task:Plan-Input-added (TaskID, ParameterID)
Task:Plan-Input-retracted (TaskID, ParameterID)
Task:Plan-Input-Type-modified (TaskID, ParameterID, FromTypeID, ToTypeID)
Task:Exec-Input-added (TaskID, ParameterID)
Task:Exec-Input-retracted (TaskID, ParameterID)
Task:Exec-Input-Type-modified (TaskID, ParameterID, FromTypeID, ToTypeID)
Task:Output-added (TaskID, ParameterID)
Task:Output-retracted (TaskID, ParameterID)
Task:Output-Type-modified (TaskID, ParameterID, FromTypeID, ToTypeID)
Task:Method-added (TaskID, MethodID)
Task:Method-retracted (TaskID, MethodID)
Task:Agent-added (TaskID, AgentID)
Task:Agent-retracted (TaskID, AgentID)
Task:Precondition-modified (TaskID, OldCondition, NewCondition)
Task:Invariant-modified (TaskID, OldCondition, NewCondition)
Task:Postcondition-modified (TaskID, OldCondition, NewCondition)
Method:Input-added (MethodID, ParameterID)
Method:Input-retracted (MethodID, ParameterID)
Method:Input-Type-modified (MethodID, ParameterID, FromTypeID, ToTypeID)
Method:Output-added (MethodID, ParameterID)
Method:Output-retracted (MethodID, ParameterID)
Method:Output-Type-modified (MethodID, ParameterID, FromTypeID, ToTypeID)
Methodscript:Task-added (MethodID, TaskID)
Methodscript:Task-retracted (MethodID, TaskID)
Methodscript:Formal-Parameter-Added (MethodID, ParameterID)
Methodscript:Formal-Parameter-Retracted (MethodID, ParameterID)
Methodscript:Formal-Parameter-Type-Modified (MethodID, ParameterID, FromTypeID, ToTypeID)
Methodscript:Parameter-Task-Relation-added (MethodID, TaskID, ParameterID)
Methodscript:Parameter-Task-Relation-retracted (MethodID, TaskID, ParameterID)
Methodscript:Task-Parameter-Relation-added (MethodID, TaskID, ParameterID)
Methodscript:Task-Parameter-Relation-retracted (MethodID, TaskID, ParameterID)
Method:Precondition-modified (MethodID, OldCondition, NewCondition)
Method:Invariant-modified (MethodID, OldCondition, NewCondition)
Method:Postcondition-modified (MethodID, OldCondition, NewCondition)
Concept:Slot-added (ConceptID, SlotID)
Concept:Slot-retracted (ConceptID, SlotID)
Concept:Slottype-modified (ConceptID, SlotID, FromTypeID, ToTypeID)
Concept:Slotcardinality-modified (ConceptID, SlotID)
Concept:Attribute-added (ConceptID, AttributeID)
Concept:Attribute-retracted (ConceptID, AttributeID)
Concept:Attribute-Type-Modified (ConceptID, AttributeID, FromTypeID, ToTypeID)

Tabelle 8: Liste mitgeloggter atomarer Ummodellierungsoperatoren

junktive) Normalform gewählt, so kann man die Ummodellierung anhand der Strukturen der Normalform genauer repräsentieren.

Die gleiche „physische“ Ummodellierung im Modell wird u.U. mehrmals durch verschiedene Ummodellierungsoperatoren in verschiedenen Sichten gezeigt (z.B. entstehen beim Hinzufügen eines Planning-Inputs zu einer Task die Ummodellierungsoperatoren Task:Plan-Input-Added und Method:Formal-Parameter-Added/Method:Parameter-Task-Relation-Added). Wir wollen dies nicht einschränken, um eventuell auf dem Logging-Konzept aufgebaute zukünftige Erweiterungen nicht zu beeinträchtigen.

Feststellen des Nettoeffekts einer Ummodellierung

Um den Nettoeffekt einer Ummodellierung festzustellen, ist es nötig, die Aufhebung eines Ummodellierungsoperators durch einen anderen Ummodellierungsoperator festzustellen (z.B. Installieren und anschliessendes Löschen eines neuen Slots) oder bei der Modifikation von Bedingungskonstrukten abzutesten, ob die Bedingung tatsächlich modifiziert wurde. Ersteres geschieht bei allen ...Added/...Retracted-Operatoren, letzteres bei allen ...Modifiied-Operatoren, bei denen zusätzlich aufeinanderfolgende Modifikationen zu einer Modifikation zusammengesetzt werden. Beides wird von der Logging-Komponente übernommen.

Das Mitloggen der Ummodellierung wird im Rahmen des Konzeptes der Ummodellierungstransaktion (Abschnitt 5.4.6) operationalisiert.

5.4.3 Wiederverwendung von Arbeit

Wir haben diesen Abschnitt bewußt „Wiederverwendung von Arbeit“ genannt und nicht z.B. Wiederverwendung von Produkten. Natürlich kann ein kompetenter Agent aufgrund seines Expertenwissens jederzeit entscheiden, ob in einer bestimmten Ummodellierungssituation einbereits erstelltes Produkt weiterverwendet werden kann. Ein Nachteil dieses Verfahrens ist, daß wir zwar Abhängigkeiten zwischen dem Produkt und einer konsumierenden Subtask aufbauen können, jedoch eventuell Abhängigkeiten, die eigentlich vorhanden sein müßten (z.B. zwischen der produzierenden Task und der Supertask). Deshalb reden wir davon, daß wir die produzierende Task weiterverwenden wollen, insbesondere auch ihre Abhängigkeitsstrukturen im Scheduler.
Zudem zielen wir auf einen hohen Automatisierungsgrad von Ummodellierungsproblematiken ab und wollen deshalb auch anhand von bestimmten Eigenschaften automatisch bestimmen können, ob ein Produkt oder Zwischenprodukt weiterhin verwertbar ist. Wir haben dazu das Konzept der „Shared Subgoal Spaces“ entwickelt.
Das Erkennen von wiederverwendbaren Tasks ist bei der Modifikation *einer* Methode noch relativ einfach, weil wir wissen, daß wir die Subtasks, die nicht verändert wurden, einfach weiterverwenden können. Finden jedoch komplexere Ummodellierungen statt, bei denen z.B. Subtasks zwischen mehreren Methoden hin- und hergeschoben werden, brauchen wir eine Eigenschaft, mit deren Hilfe wir entscheiden können, ob die innerhalb einer Task getroffenen Entscheidungen (und somit auch ihre Ergebnisse) auch weiterhin Gültigkeit haben. Trotzdem wollen wir zunächst den einfacheren Fall der Ummodellierung einer Methode betrachten und untersuchen das Verhalten der Bedingungskonstrukte Precondition und Invariante.

Die Rolle von Preconditions

Eine reguläre Abarbeitung eines Workflow-Systems vorrausgesetzt, ist die Reihenfolge des Abtestens ist im Modell eine implizite Beziehung zwischen den Preconditions einer Methode und denen ihrer Subtasks (Method.Precondition vor Subtask.Precondition). Hat sich in unse-

rem Fall die Precondition der Supermethode einer Task geändert, so müßte (bei Erhaltung der Reihenfolge der Überprüfungen der Preconditions) somit diese neue Version der Precondition vor den Preconditions der Subtask abgeprüft werden. Dies würde die Übernahme der Subtask aus der alten Methodenversion jedoch verhindern, da alle Entscheidungen im Teilbaum dieser Subtask dann auch nochmal getroffen werden müßten (Die Task müßte ja zur erneuten Überprüfung ihrer Precondition in den Zustand „Initialized“ oder „Delegated for Planning“ zurückgesetzt werden). Da sich diese strikte Reihenfolge der Überprüfung von Preconditions auch nicht durch die Precondition-Semantik rechtfertigen läßt, betrachten wir die Reihenfolge des Abtestens der Preconditions *nicht* als charakteristische Eigenschaft des CoMo-Kit-Modellierungsansatzes.
Somit können wir zwar legal auch bei modifizierter Precondition der übergeordneten Methode eine bereits bearbeitete Subtask aus der alten Version des Abhängigkeitsgraphen übernehmen, müssen aber auch erkennen, daß die Precondition keine für uns verwertbare Eigenschaft darstellt.

Die Rolle der Invariante

Zwischen den Invarianten einer Methode und denen ihrer Subtasks besteht eine engere Beziehung als zwischen den Preconditions. Invarianten einer Methode sind immer auch Teil der Invarianten der Subtasks dieser Methode. Somit ist die Übernahme von Subtasks aus der alten in die neue Methodenversion auch nur dann möglich, wenn sich die Invariante der Methode nicht geändert hat, denn hätte sie sich geändert, so müßten wir die Methode komplett zurücksetzen (d.h. auch alle in den Subtasks getroffenen Entscheidungen widerrufen), da die Gültigkeit der neuen Invariante zu allen früheren Zeitpunkten, zu denen sich die Methode in Abarbeitung befand, im Nachhinein nicht getestet werden kann.
Wichtig ist für uns hier, daß die Gültigigkeit von den in den Subtasks getroffenen Entscheidungen mit der Invariante gekoppelt sind. Wir halten es deshalb für möglich, auch für komplexere Ummodellierungen in der Dekompositionshierarchie des Prozessmodells davon auszugehen, daß Invarianten den Entscheidungskontext der in der Dekompositionshierarchie der Methode untergeordneten Entscheidungen beschreibt. Solange eine Subtask also im gleichen Entscheidungskontext bleibt (und natürlich selbst nicht verändert wird), so lange bleiben ihre Entscheidungen auch gültig[1]. Anders ausgedrückt: Solange die Invarianten der alten und neuen Supermethode gleich sind[2], so lange können wir die Subtasks der alten Methode wiederverwenden. Diese Sicht von Invarianten deckt sich auch mit der in im Abschnitt über ein methodenbasiertes Denkmodell für Ummodellierungen (Abschnitt 5.3.1.2) gemachten Überlegungen (insbesondere auch mit den Bemerkungen zur nicht festgelegten Semantik von invarianten).
Methoden mit gleicher Invariante zerlegen ihre Tasks also über der gleichen Menge von Subtasks. Diese Mengen bezeichnen wir als „Shared Subgoal Spaces“.
Im Abhängigkeitsnetzwerk ist bei Verwendung dieses Konzeptes darauf zu achten, daß eine Entscheidung nunmehr zu zwei verschiedenen Goals gehören kann und dann valid ist, wenn eines dieser Goals valid ist. Die TMS-Strukturen sind dafür entsprechend zu modifizieren.

Die Operationalisierung dieses Konzeptes geschieht im Transformationsalgorithmus, den wir in Abschnitt 5.4.5 schildern.

---

1. Natürlich nur, sofern sie nicht aufgrund anderer Ummodellierungen ungültig wird (wenn z.B. durch Ummodellierungen ihre Planning-Inputs ungültig werden).
2. Es würde eigentlich ausreichen, wenn die Invariante der alten Supermethode der Task die Invariante der neuen Supermethode impliziert. Da sich dies jedoch schlecht operationalisieren läßt, wählen wir die härtere Bedingung der Gleichheit.

5.4.4 Maßnahmen zur Transparenz von Ummodellierungen

Transparenz von Ummodellierungen hieß vor allen Dingen, daß Benutzer des WFMS stets über für sie relevante Ereignisse im Ummodellierungsmanagement schnellstmöglich benachrichtigt werden (Arbeitsersparnis!). Schnellstmöglich heißt insbesondere, daß schon während der Modellierung einer Ummodellierung das Ummodellierungsmanagement die notwendigen Reaktionen erfolgen sollen.
Grundsätzlich kann man als notwendige Reaktionen auf Ummodellierungen *Aktionen* (bei denen das System selbstständig Manipulationen am Modell oder der ablaufenden Modellinstanz bzw. dem Schedulerzustand herbeiführt) und *Notifikationen* (bei denen dem Benutzer ein bestimmter Sachverhalt gemeldet wird) unterscheiden. Aktionen finden Verwendung, wenn das WFMS sich der genauen Semantik einer Ummodellierung so sicher ist, daß es selbstständig handeln darf. Ansonsten sollten lediglich Benachrichtigungen verschickt werden, um den Planern die Möglichkeit zu geben, ihre Entscheidungen aufgrund der neuen Situation nochmals zu überdenken.
Ausgelöst werden Reaktionen Welche Reaktion erfolgen soll, hängt von der Art der Ummodellierung (Welche Ummodellierungsoperatoren fanden Verwendung?), der Sichtbarkeit der Ummodellierung und dem Zustand der betroffenen Aufgaben und Methoden.
Wir betrachten - getrennt nach der Sichtbarkeit - die Reaktionen auf Optimierungen und Verfeinerungen (die keine Umsetzung in der Modellinstanz verlangen) und die Reaktionen auf Abweichungen und Korrekturen (die in der Modellinstanze sichtbar sind). Während die Reaktionen auf erstere eher Benutzerservice als wirkliche Erfordernis sind, haben die Reaktionen auf letztere unter dem Aspekt der Arbeitsersparnis eine höhere Priorität.
Bevor wir auf die Reaktionen im Einzelnen eingehen, wollen wir unseren Beschreibungsformalismus für Reaktionen noch näher erläutern.

Darstellungsformalismus für Reaktionen

Als Formalismus zur Darstellung der Reaktionen benutzen wir *Event-Condition-Action(ECA)*-Regeln, wobei die im Zeitstrahlmodell dargestellten Ereignisse („*Events*“) über eine über die Planzustände (d.h. Zustände von Aufgaben und Methoden) definierte Bedingung („*Condition*“) die Reaktionen („*Action*“) Notifikation oder Aktion hervorrufen können. Eine Notifikation ist ein Tupel *(Empfänger, Nachricht)*; eine Aktion ein Tupel *(modifizierte Komponente, Modifikationsbeschreibung)*, wobei die „modifizierte Komponente“ das Modell oder die Planinstanz (bzw. das TMS) sein kann. In den Formeln verwenden wir intuitive Variablennamen in einer Form ähnlich der Form strukturierter Datentypen, wie sie bei vielen Programmiersprachen vorkommen:

> ***ON*** *FNU(Method-Added(Task-ID, Method-ID))*
> ***IF*** *(NewMethod.Task.State >= „AcceptedForPlanning“) AND (NewMethod.Task.State < „Finished“)*
> ***THEN*** *Notification(NewMethod.Task.PlanningAgent, „NewMethodExpected“)*

NewMethod.Task.State bezeichnet dann den „Zustand der Aufgabe, zu der die neue Methode gehört“.
Bei jeder Reaktion wird darüberhinaus die Information mitgegeben, welcher Umplanung unter wessen Verantwortung die Reaktion zugrundeliegt.
Wir gehen bei der folgenden Diskussion davon aus, daß die Änderungen, die innerhalb einer Umplanung stattfinden, aufgezeichnet werden und daß die Menge der Änderungen den *Netto-Effekt* der gemachten Änderungen repräsentiert, d.h., daß z.B. die Erschaffung und nachfol-

gende Löschung einer Subtask nicht in der Menge repräsentiert ist (siehe auch Abschnitt 5.4.2).

5.4.4.1 Reaktionen bei Optimierungen und Verfeinerungen

Optimierungen und Verfeinerungen invalidieren vorhandenes Wissen nicht. Es gibt von daher auch keinen Zwang, eine dieser Ummodellierungen im aktuellen Plan auch umzusetzen. Wichtig ist jedoch, die neuen Möglichkeiten, die durch eine Ummodellierung dieser Art hinzukommen, den relevanten Entscheidungsträgern bekannt zu machen.
Die bei Improve-Ummodellierungen notwendigen Aktionen finden nicht in der Ausführungskomponente, dem Scheduler, statt, sondern helfen dabei, in der Modellierungskomponente den Überblick über noch zu erwartende Alternativen zu behalten.
Da die einzigen Möglichkeiten, im CoMo-Kit Alternativen zu modellieren (d.h. Ummodellierungen zu spezifizieren, die *nicht* vorhandenes Wissen invalidieren) darin besteht, verschiedene Methoden für eine Aufgabe zu spezifizieren oder mehrere Agenten für eine Aufgabe vorzusehen, sind das Hinzufügen einer neuen Methode (Task:Method-added) bzw. das Hinzufügen eines neuen Agenten (Task:Agent-added) auch die einzigen echten Improve-Ummodellierungen, die zur Zeit möglich sind. Alle anderen Ummodellierungen beinhalten immer eine Modifikation bestehenden Wissens, da die Zuweisung neuen Wissens, z.B. das Hinzufügen von Inputs oder die Modifikation von Bedingungen das alte Wissen invalidiert.
Sowohl wenn neue Methoden zur Lösung einer Aufgabe verfügbar sind, als auch wenn sich die Delegationsmöglichkeiten an Agenten vergrößert haben, sollten an der betreffenden Aufgabe arbeitende Agenten (Planer) benachrichtigt werden. Ob dann tatsächlich eine Ummodellierung auch im Plan umgesetzt wird , ob also die Verantwortung für eine Subtask an einen anderen Agenten als ursprünglich geplant vergeben wird (Redelegation) oder ob die neue Methode zum Einsatz kommt, ist der Entscheidung des mit der betreffenden Aufgabe betrauten Agenten überlassen. Entscheidet sich dieser tatsächlich für die Redelegation einer Aufgabe oder die Umstellung auf die neue Methode, so kann das WFMS dies genauso behandeln wie den normalen Rückzug einer Dekompositions- oder Delegationsentscheidung. Falls mit Hilfe der Versionierung eine „Verwandtschaft“ zwischen alter und neuer Methode nachweisbar ist, kann mit Hilfe ähnlicher Konzepte, wie wir sie für die Umsetzung von Debug-Ummodellierungen vorstellen werden, durch die Wiederverwendung bereits erstellter Produkte bzw. bereits laufender Prozesse die Umplanung optimiert werden.

Hinzufügen einer Methode zu einer Aufgabe

Je nachdem, von wie großer praktischer Relevanz die neue Methode ist, sollten die im Plan mit der betreffenden Aufgabe betrauten Agenten so früh wie möglich von der neuen Methode erfahren. Relevante Zeitpunkte sind FNU (ab dem feststeht, daß eine neue Methode zur Aufgabe hinzukommen soll) und EMU (ab dem eine neue Methode tatsächlich zur Verfügung steht).
Zweierlei Maßnahmen sind nötig: Alle bereits mit der Planung dieser Aufgabe befaßten Agenten sollten zu den Zeitpunkten FNU und EMU eine Notifikation erhalten und auch alle Agenten, die erst nach dem Zeitpunkt FNU mit der Planung der entsprechenden Aufgabe betraut werden, sollten von der anstehenden Improve-Ummodellierung erfahren. Deshalb sollte die Tatsache, daß eine weitere Methode erwartet wird, bei einer Aufgabe abgespeichert werden. Das Event ENU muß zur Aufhebung dieser Maßnahmen führen. Dabei ist dann nicht zu vermeiden, daß Planer, die aufgrund des Hinweises, daß für eine Task eine neue Methode erwartet wird, ihre Planung verzögert haben, von dem Wegfall dieser neuen Methode nichts erfahren

und so u.U. unnötig warten. Das Hinzufügen einer Methode führt also den in Tabelle 9 beschriebenen Reaktionen.

ON Event	IF Condition	THEN Action
FNU	(NewMethod.Task.State >= „AcceptedFor-Planning“) AND (NewMethod.Task.State < „Finished“)	Notification(NewMethod.Task.Planner, „NewMethodExpected“)
ENU	(NewMethod.Task.State >= „AcceptedFor-Planning“) AND (NewMethod.Task.State < „Method Selected“)	Notification(NewMethod.Task.Planner, „ExpectedMethodDeleted“)
FNU	*True*	Action (Modeler, Initialize New Empty Method in NewMethod.Task)
ENU	*True*	Action(Modeler, Delete New Method Entry in NewMethod.Task)
EMU	(NewMethod.Task.State >= „AcceptedFor-Planning“) AND (NewMethod.Task.State < „Finished“)	Notification(NewMethod.Task.Planner, „NewMethodAvailable“)

Tabelle 9: Reaktionen auf das Hinzufügen einer neuen Methode

Das Event ENU wird in der Realisierungsvariante „partielle Kopie“ des Modifikationspuffers bei der Festlegung der Modifikationspufferung systemseitig getriggert, wenn sich der Änderungskontext erweitert, da dann auf einer höheren Ebene eine Methode hinzugenommen wird. Wir werden im Abschnitt über die Debug-Ummodellierungen darauf eingehen, inwieweit das System auch eine eventuelle Umsetzung der neuen Methode in den aktuellen Plan unterstützen kann.

Erweitern der Agentenkompetenzen

Bei Agenten kann es sowohl vorkommen, daß FNU, BMU und EMU fast zeitgleich passieren (z.B. bei einer spontanen Beförderung eines Agenten, bei der er schlagartig neue organisationale Kompetenzen erhält) als auch, daß sie - wie bei Methoden auch - zeitlich auseinanderliegen (z.B. wenn bekannt wird, daß er in zwei Monaten eine einwöchige Trainingsmaßnahme besuchen wird, bei der er in ein bestimmtes Tool eingeführt wird). Hier sollten deshalb Reaktionen entsprechend denen beim Hinzufügen neuer Methoden geschehen, die wir nicht weiter formalisieren, da CoMo-Kit noch kein Kompetenz- und Agentenmodell enthält. Daß diese langwierigen Ummodellierungen jedoch Probleme aufwerfen, haben wir bereits in Abschnitt 5.4.6.3 auf Seite 93 thematisiert.

5.4.4.2 Reaktionen bei Abweichungen und Korrekturen

Wir haben hier die klare Direktive des Umplaners, die Ummodellierung auch tatsächlich in der Modellinstanz (dem Abhängigkeitsgraph) sichtbar werden zu lassen. Dadurch werden zum einen u.U. bestimmte z. Zt. bearbeitete Aufgaben überflüssig, zum anderen sind produzierte Ergebnisse u.U. in der neuen Modellversion nicht sinnvoll einsetzbar. Wollen wir den Benutzer bei der Verwaltung dieser Vorgänge unterstützen, sollten wir ihm möglichst viel Arbeit abnehmen. Dazu ist zu klären, wie wir *automatisch* erkennen können, wann eine Arbeit überflüssig geworden ist und wie wir reagieren.

Das Erkennen von durch eine Ummodellierung überflüssig werdenden Aufgaben

Wir wollen zunächst einige Fakten festhalten:

- Wir können nur unter großem, nicht zu rechtfertigendem Rechenaufwand feststellen, welche Ummodellierungen durch eine bereits erfolgte Ummodellierung zur Wiederherstellung der Modellkonsistenz notwendig werden. Dies haben wir bereits in Abschnitt 5.4.1 auf Seite 62 unter dem Stichwort „Kaskadierung von Ummodellierungen“ erläutert.
- Wir können feststellen, welche direkten Auswirkungen Ummodellierungen auf den konkreten Ablaufplan haben werden. Dies können wir ableiten, indem wir ermitteln, ob bei einer Umsetzung der Ummodellierung in der Modellinstanz (aufgrund der Reaktionen, die wir in Abschnitt 5.5 beschreiben werden) die laufende Task/Methode zurückgesetzt werden würde oder ob weiterabgearbeitet werden könnte. Beispielsweise läßt sich aus der Modifikation der Invariante ableiten, daß eine Aufgabe in der im Ablaufplan des Schedulers aktiven Form nicht sinnvoll ist, da sie bei der Umsetzung der Ummodellierung sowieso zurückgesetzt werden würde.
- Andererseits können wir nicht davon ausgehen, daß jeder Ummodellierungsoperator, den der Umplaner benutzt, nie zurückgenommen wird. Der Umplaner kann sich durchaus im Laufe der Modellierungsphase noch anders entscheiden.

Wir können also prinzipiell entscheiden, wann eine Aufgabe überflüssig wird; wir können es nur nicht zum frühestmöglichen Zeitpunkt entscheiden (wenn z.B. aus Gründen der Konsistenz eine Ummodellierung eine zweite Ummodellierung irgendwann nach sich ziehen *muß* und diese zweite Ummodellierung dann eine Aufgabe überflüssig macht), und wir können es nicht auf Dauer entscheiden.

Reaktionen auf als überflüssig erkannte Aufgaben

Es ist offensichtlich notwendig, daß bereits ab dem Zeitpunkt FNU mindestens mit Notifikationen der betroffenen Agenten Einfluß auf den aktuellen Plan genommen wird, vielleicht müssen sogar Aktionen (Entscheidungsrückzüge und/oder Invalidierungen von Subprozessen und/oder Zustandsübergänge) als Reaktion in Betracht gezogen werden. Letztere sind nur dann sinnvoll, wenn das System notwendige Reaktionen automatisch generieren kann.

Wenden wir uns nun der Frage zu, welche Reaktion angebracht wäre. Gehen wir von der Situation aus, daß für eine Aufgabe sowohl eine Dekompositionsentscheidung als auch eine Delegationsentscheidung getroffen wurde. Wir nehmen weiter an, daß in einer Korrektur-Ummodellierungstransaktion eine Debug-Ummodellierung als notwendig erkannt und vom Umplaner markiert wird wird, die u.U. bedeuten könnte, daß eine oder mehrere Subtasks nicht in der bisher geplanten (und vielleicht zum Teil auch schon ausgeführten) Weise behandelt werden sollten. Untenstehende Folgen kann die Ummodellierung für die Arbeit an den Subtasks haben:

- *Invalidierung aller Subtasks*: Eine Möglichkeit ist es natürlich, alle Subtasks zu invalidieren und somit ihre Weiterbearbeitung zu unterbrechen. Der Nachteil davon ist, daß u.U. auch für die neue, ummodellierte Modellversion relevante Arbeiten unnötigerweise unterbrochen werden.
- *Laissez-Faire-Prinzip*: Das andere Extrem ist, gar nicht zu reagieren. Dabei geht man das Risiko ein, zum Zeitpunkt der Umsetzung der Ummodellierung einen großen Teil der verrichteten Arbeit einfach wegwerfen zu müssen.
- *Subtask-Spezifische Reaktionen*: Einen Kompromiß zwischen den beiden Extremen bildet die Möglichkeit, individuell für jede Subtask zu spezifizieren, ob sie

weiterbearbeitet werden sollte oder nicht, und das bis in eine beliebige Schachtelungstiefe im Teilbaum unter der ummodellierten Task. Dabei hat man die Wahl, die betroffenen Agenten nur durch Notifikationen zu entsprechendem Verhalten aufzufordern, oder den Zustand der Subtask auf valid/invalid zu setzen und so eine Weiterbearbeitung/ einen Abbruch zu erzwingen.

Wir halten den dritten Ansatz für den flexibelsten. Bei diesem Ansatz spielt der Zeitpunkt der Reaktion deshalb eine wesentliche Rolle, weil die Entscheidung über die Behandlung einer von einer Ummodellierung betroffenen Subtask nicht zu einem fest spezifizierbaren Zeitpunkt getroffen werden kann, sondern es zu einem beliebigen Zeitpunkt zwischen FNU und EMU einer Methode klar werden kann, daß eine Subtask weiter bearbeitet werden sollte bzw. überflüssig ist. Die Frage ist, wie der die Subtask bearbeitende Agent über die mögliche Invalidierung seiner Arbeit informiert bzw. instruiert werden sollte.
Prinzipiell können wir dies mit einem Notifikationssystem oder über die Einführung neuer Zustände, die die Unsicherheit symbolisieren, die bezüglich der Brauchbarkeit einer Aufgabe auftaucht, wenn an ihr eine Ummodellierung vorgenommen wurde. Bei einem Notifikationssystem hat der Umplaner keine direkte Kontrolle über den Plan, er muß sich darauf verlassen, daß der Agent entsprechend reagiert. Bei der Einführung neuer Zustände, die die Weiterarbeit an einer Aufgabe unterbinden könnten, könnte die Akzeptanz des Systems leiden, wenn sich die Bearbeiter durch mehrfaches Ein- und Ausschalten von Tasks durch den Umplaner gegängelt fühlen. Auch ist die Entscheidung, eine Aufgabe weiterzubearbeiten oder nicht möglicherweise von mehreren Faktoren abhängig, die im Zustandsmodell nicht berücksichtigt werden (z.B. wenn ein Bearbeiter eine Aufgabe „zur Übung“ fertigmacht, weil er sowieso nichts anderes zu tun hat; oder wenn ein Agent so viel Arbeit hat, daß er auch schon bei der kleinsten Unsicherheit über die Nützlichkeit seiner Bearbeitung eher eine andere Aufgabe bearbeiten würde). Alles in allem halten wir ein Notifikationssystem für angebrachter, da es mehr Freiheitsgrade offeriert.

Quintessenz

Das gravierendste Problem bleibt aber immer noch, daß erst am Schluß einer Ummodellierungstransaktion *definitiv* feststeht, welche Arbeiten überflüssig werden und welche nicht. Wir nehmen deswegen davon Abstand, die Verwaltung der Einflüsse der Modellierung auf den Plan zu automatisieren (d.h. über die Ummodellierungsevents zu triggern). Stattdessen geben wir dem Umplaner die Möglichkeit, direkt aus der Modellierung heraus mit einer Notifikation über die Unsicherheit einer Aufgabe an den Planer indirekt die Abarbeitung zu beeinflußen. Auch hier muß der Sachverhalt der Unsicherheit wieder im Modell gespeichert werden.

5.4.5 Die korrekte Umsetzung der Ummodellierung

Zur Umsetzung der Ummodellierung ist ein Algorithmus zur notwendig, der die im TMS-Netzwerk des Schedulers gespeicherten Abhängigkeiten dem durch die Ummodellierung modifizierten Modell angepaßt wird.
Wir setzen uns zunächst mit den bei den einzelnen Ummodellierungen möglicherweise auftretenden Problematiken auseinander (Abschnitt 5.4.5.1), spezifizieren dann als Ergebnis dieser Untersuchungen, welche Zustandsübergänge wir mit Hilfe des Umsetzungsalgorithmus erreichen wollen (Abschnitt 5.4.5.2) und beschreiben schliesslich den Algorithmus allgemein (Abschnitt 5.4.5.3) und zusätzlich (als Beispiel) an dem in Abschnitt 3.2 beschriebenen Szenario.

Grundsätzlich folgen wir der Idee, daß Ummodellierungen nichts mit der normalen Planabarbeitung zu tun haben. Im Abhängigkeitsgraphen des Schedulers sollte es daher nach einer Ummodellierung so aussehen, als wäre der Plan schon immer nach der neuen Version des Workflow-Modells abgearbeitet worden (von lokalen Ummodellierungen abgesehen).

5.4.5.1 Die Problematik der Umsetzung von Ummodellierungen

Im Prinzip haben wir drei verschiedene Gruppen von Ummodellierungen. Modifikationen des Datenflusses (neue Inputs/Outputs), Modifikationen der den Arbeitsablauf überwachenden Bedingungen (Pre- und Postconditions/Invarianten) und Modifikationen der Beziehungen zwischen den Modellierungseinheiten (Task/Method, Task/Method/Agent und Product/Slot). Wir wollen jetzt diese Ummodellierungen näher betrachten.
Wir werden hier zum Teil auch Lösungen für die beschriebenen Problematiken beschreiben. Die für den Umsetzungsalgorithmus letztendlich charakterisierenden Reaktionen beschreiben wir jedoch erst im nächsten Abschnitt.

5.4.5.1.1 Änderungen an Bedingungskonstrukten

Die besondere Schwierigkeit bei der Umsetzung von Modifikationen an Bedingungen entsteht aus der noch unklaren Semantik von Postconditions und dem „Dauerhaftigkeitsanspruch“ von Invarianten.

Änderungen an Preconditions

Preconditions haben im CoMo-Kit nur eine sehr eingeschränkte Funktionalität. Von ihnen wird nur verlangt, daß sie im Moment des Akzeptierens einer Aufgabe oder einer Methode (welches den Beginn der Abarbeitung der Aufgabe/Methode impliziert) durch einen Agenten wahr sein müssen. Dauerhaft zuzusichernde Bedingungen müssen in der Invariante spezifiziert werden. Daraus folgt aber auch, daß als einzige Maßnahme bei einer Modifikation der Precondition dafür gesorgt werden muß, daß im Moment des Akzeptierens einfach die aktuelle Form der Precondition zum Testen genommen wird. Dies kann von der normalen Ablauforganisation gehandhabt werden und erfordert keine speziellen Reaktionen bei der Umsetzung.

Änderungen an Invarianten

Ist die Aufgabe/Methode bereits in einem Zustand, in dem die Invariante bereits gelten müßte, so kann nicht abgeprüft werden, ob die neue Invariante zu allen früheren Zeitpunkten, an denen die alte Invariante erfüllt war, ebenfalls erfüllt war. Eine Ausnahme bildet hier der Fall, daß ($\text{Invariante}_{alt} \Rightarrow \text{Invariante}_{neu}$) eine Tautologie darstellt, was wiederum schwierig festzustellen ist.
Invarianten sollen über die gesamte Laufzeit einer Task gültig sein. Insbesondere sind sie dann auch über die gesamte Laufzeit ihrer Methoden/Subtasks gültig. So gesehen sind Invarianten von Tasks immer Teil der Invarianten ihrer Methoden und Invarianten von Methoden immer Teil der Invarianten ihrer Subtaks. Das bedeutet, daß die Modifikation einer Invariante immer auch die Modifikation aller Invarianten im sich unter der eigentlich ummodellierten Task/Methode befindlichen Teilbaum der Dekompositionshierarchie. Das bedeutet auch, daß die Modifikation der Invariante alle in diesem Teilbaum bereits gefällte Entscheidungen und alle produzierten Produkte invalidiert. Deren TMS-Strukturen sollten dementsprechend aus dem Abhängigkeitsgraphen völlig entfernt werden[1].

Änderungen an Postconditions

Bei Postconditions muß man zwischen dem eventuell spezifizierten temporalen (z.B. Prädikate, die von Zuständen anderer Tasks abhängen) und dem nicht-temporalen Teil der Bedingung unterscheiden. Wird der nicht-temporale Teil geändert, so braucht nur die Bedingung erneut abgeprüft zu werden, um die Validität der Outputs zu beurteilen. Im temporalen Teil treten ähnliche Probleme wie bei Invarianten auf, wird z.B. die Postcodition um ein Prädikat erweitert, welches den Zustand einer anderen Task/Methode abprüft, so kann im nachhinein nicht gesagt werden, ob dieses neue Prädikat zu allen Zeitpunkten galt, an denen die Postcondition bereits erfüllt sein musste. Die Aufgabe/Methode muß u.U. erneut abgearbeitet werden.
Bei Postconditions ist die genaue Semantik z. Zt. noch nicht geklärt. So ist die Frage der Dauerhaftigkeit einer Postcondition nicht geklärt (die Extreme sind dort punktuelle Gültigkeit bei Abschluß einer Aufgaben-/Methodenabarbeitung vs. Postconditions als „ewige“ Constraints über die Produkte) und auch nicht die Frage nach dem Einfluß, den Postconditions auf die Sichtbarkeit von Ergebnissen für nachfolgende Tasks/Methoden haben. Als Alternativen stehen zur Verfügung, Outputs sofort nach ihrer Erstellung für nachfolgende Workflows sichtbar zu machen (was einen unterbrechungsfreien Arbeitsfluß ermöglicht; dies ist Stand der aktuellen CoMo-Kit-Implementierung) oder erst nach Gültigkeit der Postcondition (was der Intention der Postcondition, als Instrument der Qualitätssicherung zu dienen, entspricht). Erstere Alternative nennen wir *schwache* Postcondition, letztere *starke* Postconditions.
Wählen wir erstere Alternative, so muß stets damit gerechnet werden, daß ein Output, der sich woanders bereits in Verwertung befindet, im Nachhinein noch als ungültig erklärt werden kann, falls die Postcondition der ihn produzierenden Methode nicht erfüllt wurde. Bei letzterer Variante kann das Problem auftreten, daß nach einer Ummodellierung einer bereits abgearbeiteten Task, bei der dieser ein zusätzlicher, noch nicht erstellter Output angefügt wurde, einige über die Postcondition bereits validierte Outputs bereits an anderer Stelle Verwendung finden, die aufgrund der Tatsache, daß die Task aus dem Zustand „Finished“ (den sie im Abhängigkeitsgraph vor der Ummodellierung hatte) in den Zustand „In Execution“ (den sie in der neuen Version des Abhängigkeitsgraphen korekterweise haben sollte) wechselt, wieder ungültig werden, bis die Postcondition erneut abgeprüft wurde.
Kommt die erste Alternative zum tragen, so bleiben die Outputs der Task/Methode der Postcondition prinzipiell gültig („assigned“). Sie können aber durch einen Fehlschlag der die Task/Methode abschliessenden Überprüfung der modifizierten Postcondition wieder ungültig werden. Auch bei der zweiten Alternative wird die modifizierte Postcondition auf jeden Fall erneut abgeprüft, bis zu dieser Überprüfung müssen die Outputs hier jedoch invalidiert werden. Dieser Unterschied wird bei zusätzlich modifiziertem Datenfluß u.U. erheblich. Wir gehen im nächsten Abschnitt nochmals auf diese Problematik ein.
Bei beiden Alternativen würde das Hinzufügen eines neuen Zustandes „InValidation“ zum Produktvariablen-Zustandsmodell zumindest eine Abstufung der Validität bzw. Endgültigkeit eines Variablenwertes erlauben. Eine Variable würde diesen Zustand immer dann annehmen, wenn die Postcondition einer Methode, bei der dieser Variablenwert Output war noch nicht überprüft wurde bzw. erneut überprüft werden muß. Über die Abfrage dieses Zustandes in den Preconditions/Invarianten einer diese Variable konsumierenden Task/Methode könnten entsprechende Reaktionen der Konsumenten initiiert werden. Standardmäßig würde „InValidation“ bei schwachen Postconditions dem Zustand „Assigned“ entsprechen, bei starken dem Zustand „Unassigned“.

---

1. Die Ausnahme hier sind Abweichungen und Verfeinerungen, bei denen wir uns ein einfaches Zurücksetzen der Ummodellierungen vorbehalten wollen. Auch da ist es aber so, daß die TMS-Strukturen von einem Planer/Bearbeiter auf normalem Wege nicht wieder gültig gemacht werden können.

5.4.5.1.2 Änderungen am Produktfluß

Wir wollen uns zunächst nochmal anhand eines Beispiels die Entstehung einer Datenflußmodellierung ansehen. Wir werden untersuchen, welche Möglichkeiten es allgemein gibt und welche Probleme im Kontext der im letzten Abschnitt beschriebenen Unklarheiten bei den Postconditions auftreten.
Die natürliche Entstehung von Änderungen am Datenfluß ist wohl das Feststellen eines „Produktdefizits“ (das Fehlen eines Produktes/einer Information) beim Abarbeiten einer Methode. Wir gehen hier davon aus, daß in einer Subtask festgestellt wird, daß zu deren Planung ein weiterer Input benötigt wird (siehe auch Beispiel aus Abschnitt 3.2 auf Seite 20). Wir können jetzt vier verschiedene Möglichkeiten untersuchen:

- Fall 1: Der Input ist bereits als Output im Modell vorhanden, und zwar in der Methode einer Supertask.
- Fall 2: Der Input ist bereits im Modell vorhanden, und zwar als Formaler Parameter in einer vorhergehenden Subtask bzw. deren Methode.
- Fall 3: Der Input ist nicht vorhanden und muß von Außen (d.h. als neuer Input der Wurzeltask) spezifiziert werden.
- Fall 4: Der Input ist nicht vorhanden, kann aber in einer vorhergehenden oder in einer neuern Subtask produziert werden.

Fall 1 ist der einfachste, es müssen nur neue Abhängigkeiten zwischen den Variablen-TMS-Strukturen des Inputs und den Strukturen des Goals installiert werden. Ist die Aufgabe, für die der neue Input benötigt wird, Auch wenn die Aufgabe bereits über den Zustand „Acceptable for Planning“ hinaus ist, muß sie in den Zustand „Initialized“ zurückgesetzt werden, da Planning Inputs zum Zeitpunkt der Dekompositionsentscheidung vorliegen müssen.solange der neue Input nicht über Precondition oder Invariante auf das zeitliche Verhalten der Aufgabe Einfluß nimmt. Tut er das, so wurden Precondition oder Invariante modifiziert, und die entsprechenden Maßnahmen werden ergriffen.
In Fall 2 kommen nicht nur neue Inputs ins Modell, sondern auch neue Outputs bei den Methoden und Aufgaben, die zwischen der produzierenden Methode und der untersten gemeinsamen Supermethode von produzierender Methode und konsumierender Task liegen. Dies ist notwendig, um den Output bis zur konsumierenden Aufgabe weiterzupropagieren. Auch hier treten über die neue Abhängigkeitsbeziehungen hinausgehende Änderungen nur auf, wenn Invariante oder Postcondition ebenfalls modifiziert wurden.
Bei Fall 3 erhält die bei der Dekompositionsentscheidung der Wurzeltask ausgewählte Methode einen neuen Input; ebenso alle der konsumierenden Task übergeordneten Methoden und Aufgaben. Hier tritt zum ersten Mal das Problem auf, daß das benötigte Produkt erst noch erstellt werden muß was eine gewisse Zeit in Anspruch nehmen kann und den Abarbeitungsprozeß einer Task, für die er als Planning Input zählt, u.U. länger lahmlegen könnte.
Der komplizierteste Fall ist Fall 4. Stellen wir uns vor, die das Produkt erstellende Task wurde bereits beendet. Wir haben das Produkt in der produzierenden Subtask als Output hinzugenommen. Da dieses Produkt noch nicht erstellt wurde, wird aus der beendeten Subtask korrekterweise wieder eine arbeitende und der Zustand müßte sich von „Finished“ nach „In Execution“ ändern. Dies hat dann die Folgen, die wir bereits im Abschnitt "Änderungen an Postconditions" in Abschnitt 5.4.5.1.1 beschrieben haben, hier sogar ohne daß tatsächlich eine Änderung der Postcondition vorliegen muß. Ergänzend zu der dortigen Beschreibung zeigt dieser Fall, daß von dem Moment, in dem eine Task aus dem Zustand „Finished“ nach „In Execution“ wechselt, bis zu dem Moment, in dem die Postcondition erneut abgeprüft wird (und damit auch alle früher erstellten Outputs wieder validiert werden), durch die Notwendigkeit, einen neuen

Output erstellen zu müssen durchaus eine längere Zeitspanne vergehen kann, in der nachfolgende Tasks bei starken Postconditions auf Eis liegen.
Wir diskutieren nun die einzelnen möglichen Ummodellierungen. Die atomaren Methoden-Ummodellierungen Task-Parameter-Relation-Added/Deleted bzw. Parameter-Task-Relation-Added/Deleted betrachten wir als identisch zu den Task-Ummodellierungen Output-Added/Deleted bzw. Planning/Execution-Input-Added/Deleted und erwähnen sie nicht gesondert.

Typänderungen an formalen Parametern

Da bei Typänderungen an formalen Parametern eigentlich keine Modifikation des Datenflusses stattfindet, sondern die Beschreibung dessen, was fließt, geändert wird, tauchen keine Probleme auf, solange produzierende und konsumierende Task den Typwechsel beachten (wovon wir ausgehen, da die Ummodellierung die Modellkonsistenz nicht verletzt haben darf). Einzig die Umsetzung der bereits erstellten Instanzen ist eventuell notwendig (Wir diskutieren die Problematik ausführlicher in Abschnitt 5.4.5.1.4).

Planning- und Execution-Inputs hinzufügen/löschen

Da wir Planungsinputs als für die Planung einer Aufgabe notwendige Informationen betrachten, gehen wir davon aus, daß eine Entscheidung, die nicht auf allen Planungsinputs beruhte, zurückgezogen werden sollte. Natürlich kann auch hier dann die gleiche Entscheidung nochmal getroffen werden, womit dann auch die bereits erstellten Produkte wieder gültig werden. Die Löschung von Planungsinputs hat hingegen auf den Bearbeitungszustand keinen Einfluß, da zum Zeitpunkt der Entscheidung der Aufgabe einfach ein offensichtlich irrelevante gewordener Input berücksichtigt wurde. Natürlich müssen in beiden Fällen die TMS-Abhängigkeiten ergänzt bzw. gelöscht werden.
Execution-Inputs von Tasks werden grundsätzlich nur an die entsprechende Methode weitergereicht. Bei komplexen Methoden verhält es sich ebenso, die Inputs werden an die konsumierenden Subtasks weitergereicht. Die wirkliche Verwertung der Inputs findet als Planning-Input bei den Tasks oder als Execution-Input bei den atomaren Methoden statt. Erst diese verwertenden Instanzen müssen auf die hinzugekommenen/gelöschten Inputs reagieren. Dabei verhalten sich atomare Methoden genauso wie konsumierende Tasks: Bei hinzugekommenen Inputs muß die Methode erneut abgearbeitet werden, die Löschung von Inputs ist irrelevant.

Outputs hinzunehmen/löschen

Wie im Beispiel weiter oben bereits beschrieben, müssen hinzukommende Outputs natürlich nachträglich noch erstellt werden, falls die betreffende Task/Methode bereits abgearbeitet wurde. Dabei kommt es zu den oben beschriebenen Problemen mit den Postconditions. Gelöschte Inputs wiederum haben keinen Einfluß auf den Abarbeitungszustand, die mit ihnen verbundenen Abhängigkeiten werden einfach gelöscht.

5.4.5.1.3 Änderungen an der Dekompositionshierarchie

Änderungen an den Methoden einer Tasks sind relativ unproblematisch: Neue Methoden müssen den Agenten bekannt gemacht werden, gelöschten Methoden werden zunächst durch einen Operatorrückzug behandelt und ihre Strukturen im Abhängigkeitsgraphen eliminiert[1]. Wir gehen davon aus, daß alle neuen Methoden in einer Ummodellierungstransaktion, deren Sichtbarkeit die Modellinstanz umfasst, auch tatsächlich umgesetzt werden sollen, d.h. falls eine

1. Auch hier wieder vorbehaltlich der Maßnahmen zu einem evetuellen Rücksetzen der Ummodellierung. Zusätzlich muss noch darauf geachtet werden. daß Subtasks, die übernommen werden sollen, nicht gelöscht werden.

Entscheidung für eine Methode bereits existiert, so wird diese zurückgezogen und durch eine Entscheidung für die neue Methode ersetzt.[1]
Das Hinzufügen neuer Tasks zu einer Methode läßt sich dadurch handhaben, daß im Zustandsmodell in den Zustand „Execution Started“ zurückgeschaltet wird, falls die Methode schon über diesen Punkt hinaus war; und ansonsten die TMS-Strukturen genau so modifiziert werdender neuen Task genauso eingebracht werden, wie es gewesen wäre, wenn sie bei der ursprünglichen Dekompositionsentscheidung beteiligt gewesen wäre. Der Zustandsübergang wird natürlich weiter nach oben propagiert, so daß alle Supertasks und -methoden sich wieder in Bearbeitung befinden. Die weiter oben bereits beschriebene Postcondition-Problematik tritt durch diesen Zustandsübergang dann auch hier zutage.
Sind neue Tasks einer Methode identisch mit Tasks der alten Methodenversion, so können sie auch später im Plan übernommen werden. Prinzipiell läßt sich die Übernahme von Subtasks von einer Methode zur nächsten unter bestimmten Vorraussetzungen auf beliebige Methoden erweitern, solange diese auf der gleichen Menge von Unteraufgaben („Shared Subgoal Spaces“) operieren. Wir gehen in Abschnitt 5.4.3 auf Seite 67 näher darauf ein.

5.4.5.1.4 Änderungen am Produktmodell

Bereits in Abschnitt 5.1.2 auf Seite 49 haben wir darauf hingewiesen, daß wir versuchen wollen, Produkttypmodifikationen als Einführung eines neuen Produkttypen zu betrachten. Dementsprechend liegt bei einer Typmodifikation bei den betroffenen Variablen ein Typwechsel vor. Wir wollen hier betrachten, wie die Instanzen in ihren neuen Typ modifiziert werden können.

Transformation der Variablenwerte

Da alle im Rahmen der Prozeßabarbeitung erstellten Werte entweder initial vorgegeben sind oder in einer Subtask erarbeitet werden, bräuchten wir keine weiteren Konzepte solange Produktummodellierungen zu harten Modifikationen gehören, die sich auch im Prozeßmodell niederschlagen (z.B. wenn das Hinzufügen eines neuen Slots in einem Produkttyp einhergeht mit dem Einfügen einer neuen Subtask in alle Methoden, die Instanzen dieses Produkttyps erzeugen). In einem solchen Fall übernehmen die Maßnahmen, die aufgrund der Ummodellierungen des Prozeßmodells bei der Umsetzung getroffen werden, alles nötige, um eine korrekte Instanz dieses neuen Produktypes zu erstellen. Wir können diese Art Typmodifikation daran erkennen, daß *strukturelle* Änderungen vorgenommen werden (Slots/Attribute hinzufügen/wegnehmen).
Anders hingegen sieht es aus, wenn es sich um eine Ummodellierung ohne Bezug zum Prozeßmodell handelt, wenn es z.B. um eine Berichtigung eines Modellierungsfehlers in einem Produkttyp geht (Wenn z.B. als Attributtyp „String“ angegeben wurde, obwohl dort „Number“ stehen sollte); wenn sich also nur daß „Aussehen“ einer Variablen ändert. Wir nennen dies entsprechend eine „weiche“ Modifikation. In einem solchen Fall muß die Produkttypummodellierung von der Implementierung einer „Transformationstask“ begleitet werden. Diese muß dann wie eine normale Task in den Ablaufplan eingefügt werden, ohne in der Modellsicht sichtbar zu sein, da sie prinzipiell nicht zu der Dekomposition der zugehörigen Task gehört.[2] Sie kann sogar automatisiert ablaufen, wenn sich ein funktionaler Zusammenhang zwischen alter Produkttypversion und neuer Produkttypversion existiert. Dieser Zusammenhang muß dann bei der Ummodellierung explizit spezifiziert werden. Wird er nicht spezifiziert, so wird zwischen Produktion und Verwertung der Instanzvariablen dieses Typs eine Standard-Transformations-

1. Ist dieses Verhalten nicht erwünscht, so sollte man dies in unterschiedlichen Ummodellierungstransaktionen realisieren.
2. Diese Realisierung erzeugt eine Inkonsistenz zwischen Modellsicht und Modellinstanz und kann deshalb zu Schwierigkeiten bei der Abarbeitung führen. Sauberer wäre die Kapselung solcher Transformationen mit Hilfe einer Versionierungskomponente.

task eingesetzt, in der ein Agent die Übersetzung „von Hand“ vornehmen muß. Um diese Art von Transformationen möglichst selten anwenden zu müssen, wäre es von Vorteil, wenn das Typsystem für die Grundtypen Standardtransformatoren anbieten würde.
Diese Transformationstasks könnten sowohl als Teil der Umsetzung betrachtet werden oder, wie oben angedeutet, explizit in den Plan eingefügt werden. Erstere Möglichkeit hat als größten Nachteil, daß sie Zeit in Anspruch nimmt (*Viel* Zeit, wenn ein Agent die Transformation „von Hand“ vornehmen muß!). Dies widerspricht unserem Ziel, die Umsetzung nur so kurz wie möglich dauern zu lassen, um die Planabarbeitung nicht übermäßig zu stören (Verfügbarkeit des WFMS!). Letztere Möglichkeit hätte zur Folge, daß eine automatische, interne Improve-Ummodellierung als Abweichung für die neue Methode mit Transformationstask spezifiziert werden müßte, was eine unnötige Komplexität in den Ummodellierungsprozeß einbringen würde.
Eine weitere Möglichkeit, der Problematik von Typtransformationen bei Variablen beizukommen, wäre die Auffassung, die Typen von Outputs als *den Postconditions äquivalente* Zusicherungen über die Outputs zu sehen. In diesem Fall würden auf eine solche Typmodifikation die gleichen Reaktionen erfolgen wie auf die Änderung der Postcondition der sie produzierenden Methode. Diese Methode führt bei starken Postconditions jedoch zu unnötigen Verzögerungen im Planablauf.
Da diese Problematik nur schwer zu handhaben ist, schlagen wir vor, automatisierbare Variablentransformationen (z.B. „String“ nach „Text“) als Teil der Umsetzung zu handhaben und Typtransformationen, die menschliches Eingreifen verlangen, nicht zuzulassen. Eine solche Typmodifikation betrachten wir dann bei der produzierenden Task/Methode als Löschung eines Outputs und anschliessendes Hinzufügen eines neuen Outputs.

Die Unterstützung des CoMo-Kit-Binding-Konzeptes

Im CoMo-Kit werden im Scheduler für die Formalen Parameter der Methoden Variablen eingeführt, die nicht nur eine gültige Wertbelegung haben, sondern auch alte Wertbelegungen als sogenannte „Bindings“ speichern. So werden bei Zurücksetzen einer Methode und Anwendung einer anderen nicht die alten Variablenwerte und Assigments überschrieben. Dabei wurden alle Wertbelegungen durch verschiedene Methoden erstellt.
Alle bisherigen Überlegungen zur Produkttypummodellierung galten für die aktuelle Wertbelegung einer Variablen. Was jedoch geschieht mit den alten Wertbelegungen?
Auch bei den alten Wertbelegungen müssen wir uns um strukturverändernde Modifikationen, keine Sorgen machen; ihre Transformation wird mit Hilfe der Prozeßmodell-Maßnahmen durchgeführt. Bei Grundtyptransformationen müssen wir dafür sorgen, daß nicht nur die aktuelle Wertbelegung, sondern auch *alle* früheren Werte transformiert werden müssen.

Diese Überlegungen zeigen allerdings, daß auch im CoMo-Kit u.U. die Einführung eines Versionierungskonzeptes, in dem solche Transformationen unabhängig von Prozeßmodell und Ablaufplan gehandhabt werden können, sinnvoll sein könnte. Auch das Problem inkompatibler Typwechsel (z.B. „Text“ nach „Graphik“) könnte durch Einbinden einer mit dem Scheduling verbundenen Dialogkomponente eines Versionsmanagements, die die Transformationstask in die normale Task-Methoden-Hierarchie einbettet, gelöst werden.
Wir gehen davon aus, daß Typmodifikationen an der Semantik eines Parameters nichts ändern, d.h. falls an bereits beendeten Tasks/Methoden die Typen der Input-Parameter geändert werden, wird die Task/Methode *nicht* zurückgesetzt.

5.4.5.1.5 Änderungen an den Agentenbindungen

Kommen bei einer Task/Methode neue Agenten hinzu, so sollte der Agent, der die entsprechende Delegierungsentscheidung zu treffen hat (Bearbeiter der Supermethode/Planer der Supertask) von den neuen Möglichkeiten erfahren.
Wir können zwar nicht beurteilen, inwieweit eine Löschung eines Agenten seine Entscheidungen auch im Nachhinein noch invalidiert, jedoch muß dafür gesorgt werden, daß für eine ablaufende Task oder Methode immer ein Agent zuständig ist. Bei der Löschung eines Agenten in einer Task muß die Task zurückgesetzt werden, falls der gelöschte Agent sie als Planer akzeptiert hatte und sie noch nicht abgeschlossen wurde. Falls er bereits eine Dekompositionsentscheidung getroffen hatte, muß dann auch diese zurückgenommen werden (wobei sie von einem anderen Agenten dann wieder validiert werden kann). Entsprechend verfahren wir bei der Löschung des Bearbeiters einer Methode: Ist sie noch in Arbeit, wird sie zurückgesetzt, ist sie bereits abgeschlossen, erfolgt keine Zustandsänderung.[1]
Da es aber doch passieren kann, daß auch im Nachhinein Entscheidungen gelöschter Agenten überdacht werden müssen, sollten die jeweils übergeordneten Agenten eine Notifikation erhalten.

5.4.5.1.6 Änderungen an inaktiven Komponenten

Im Abhängigkeitsgraphen von CoMo-Kit wird der gesamte bisher verfolgte Plan inklusive aller Abhängigkeiten gespeichert. Insbesondere sind auch Wege (Tasks/Methoden) gespeichert, die zunächst versucht wurden und dann wieder zurückgezogen wurden. Da es jedoch zur Zielsetzung vom CoMo-Kit gehört, auf die in diesen Strukturen gespeicherten Ergebnisse/ Planverläufe zurückgreifen zu können, müssen wir damit rechnen, daß auch diese Strukturen irgendwann wieder aktiviert werden (d.h. die betreffende Dekompositionsentscheidung wird u.U nochmal getroffen). Aus diesem Grunde müssen wir solche Methoden genauso behandeln, als wären sie noch aktiv. Dabei kann es durchaus passieren, daß eine Methode, die bereits den Zustand „PostconditionSatisfied“ erreicht hatte, nach der Umsetzung der Ummodellierung im Zustand „ExecutionStarted“ eingefroren bleibt. Wir werden auf diese inaktiven Komponenten nicht weiter eingehen.

5.4.5.2 Reaktionen in der Umsetzungsphase

Nachdem wir im vorherigen Abschnitt die in der Umsetzungsphase auftretenden Problematiken näher beleuchtet haben, wollen wir die in dieser Phase notwendigen Reaktionen etwas formaler aufschreiben. Reaktionen sind hier Notifikationen und Zustandsänderungen. Während die Notifikationen über eine entsprechende Notifikationskomponente abgewickelt werden, werden die Zustandsänderungen durch die bei der Umsetzung notwendig werdenden TMS-Restrukturierungen nebenbei mitausgeführt, da im CoMo-Kit Zustandsgraph und Abhängigkeitsgraph im gleichen TMS-Netzwerk verwaltet werden. Aus diesem Grund erwähnen wir hier auch keine propagierten Zustandsänderungen (z.B. wenn eine Methode aus „PostconditionSatisfied“ zurück in „ExecutionStarted“ wechselt, wechseln die übergeordneten Tasks aus „Finished“ zurück in „InExecution“), da sie sich von selbst ergeben werden. Genauere Informationen über diese Zustandspropagierungen findet man in [Chri96].
Wir benutzen dazu eine ähnliche Syntax wie bei den Beschreibungen für die Modellierungsphase.

5.4.5.2.1 Reaktionen bei der Umsetzung von Task-Ummodellierungen

1. Ist dieses Verhalten unerwünscht, so kann dies dadurch verhindert werden, daß in einer Umplanung (nicht Ummodellierung!) mit der Auswahl eines anderen Agenten die getroffene Delegierungsentscheidung revidiert wird.

Bei Ummodellierungsoperator	IF Condition	THEN Action
Plan-Input-added	OTV.State > „AcceptableForPlanning“	NTV.State = „AcceptableFor Planning“
Output-added	OTV.State >= „Finished“	NTV.State = „In Execution“
Method-added	(OTV.State >= „AcceptableForPlanning“) AND (OTV.State <= „In Execution“)	Notification(NTV.Agent, „New Method applicable“)
Method-retracted	(OTV.State > „AcceptedForPlanning“) AND (OTV.Decision = RetractedMethod)	(NTV.State = „AcceptedForPlanning“) AND (NTV.Decision.State = „SupportedRetraction“)
Agent-added	(„Initialized <= OTV.State <= „AcceptedForPlanning“)	Notification(NTV.Supermethod.Agent, „Agent Added“)
Agent-retracted	(OTV.State >= „AcceptableForPlanning“) AND (OTV.State < „Finished“)	NTV.State = „AcceptableForPlanning“
	OTV.State = „Finished“	Notification(NTV.Supermethod.Agent, „Agent may be incompetent“)
Invariant-modified	(OTV.State > „Initialized“) AND (NOT (OTV.Invariant ⇒ NTV.Invariant))	NTV.State = „Initialized“

Tabelle 10: Reaktionen bei der Umsetzung von Task-Ummodellierungen

Legende: OTV = Old Task Version (vor der Umsetzung); NTV = New Task Version (Nach der Umsetzung)

Nicht alle Ummodellierungsoperatoren rufen Reaktionen hervor, da einige Reaktionen von den entsprechenden Methoden- oder Subtask-Ummodellierungsoperatoren übernommen werden. Natürlich muß für jeden Ummodellierungsoperator eine entsprechend Ummodellierung des TMS stattfinden; dies erwähnen wir hier nicht gesondert. Postconditions fallen hier aus unseren Betrachtungen heraus, da sie zwar für Tasks spezifiziert werden können, letztendlich aber immer bei den Methoden abgeprüft werden.
In Tabelle 5 beschreiben wir die notwendigen Zustandsübergänge.

5.4.5.2.2 Reaktionen bei der Umsetzung von Methoden-Ummodellierungen

Bei den Methoden-Ummdellierungen unterscheiden wir zwischen den Reaktionen bei komplexen Methoden (Tabelle 6) und denen bei atomaren Methoden (Tabelle 6). Die kursiven Zustandsübergänge würden bei der Realisierung eines weiteren Zustandes „InValidation“ im Zustandsmodell für Produkte entstehen.

5.4.5.2.3 Reaktionen bei der Umsetzung von Produkt-Ummodellierungen

Wie wir bereits in Abschnitt 5.4.5.1.4 auf Seite 78 beschrieben haben, werden harte (d.h. strukturelle) Produkttypänderungen an Konzepten von entsprechenden Prozeßummodellierungen begleitet. Über die bei diesen wirkenden Mechanismen werden auch die Instanzen der Produkttypen in die neue Form transformiert. Anders verhält es sich bei den weichen Modifikationen, also bei der Transformation von einem Grundtyp in einen anderen Grundtyp, bei denen wir noch für eine Übesetzung sorgen müssen. Diese soll automatisch im Rahmen der Umsetzung der Ummodellierung passieren und ist somit für den Benutzer transparent. Insbesondere werden keine Zustandswechsel hervorgerufen.
Welche Änderungen wir zulassen, haben wir bereits in Abschnitt 5.1.2 auf Seite 49 beschrieben. Die tatsächliche Transformation von Variablenbelegungen bildet eine Phase des Umsetzungsprozesses.

Bei Ummodellierungsoperator	IF Condition	THEN Action
Task-added	OMV.State >= „ExecutionFinished“	NMV.State >= „ExecutionStarted“
Task-retracted	OMV.State = „ExecutionStarted“ AND RetractedTask.State <> „Finished“ AND *all other Tasks „finished“*	NMV.State = „Finished“
Invariant-modified	OMV.State > „Initialized“ AND (NOT (OMV.Invariant ⇒ NMV.Invariant))	NMV.State = „Initialized“
Postcondition-modified	(OMV.State = „PostconditionSatisfied“) AND (NOT (OMV.Postcondition ⇒ NMV.Postcondition))	NMV.State = „ExecutionFinished“ *AND (FORALL Outputs in Postcondition (NMV.Output.State = „InValidation“))*
	(OMV.State = „Exception“) AND (NOT (OMV.Postcondition ⇒ NMV.Postcondition))	NMV.State = „ExecutionFinished“
	(OMV.State = „ExecutionFailed“) AND (NOT (OMV.Postcondition ⇒ NMV.Postcondition))	NMV.State = „ExecutionFinished“

Tabelle 11: Reaktionen bei der Umsetzung von komplexen Methoden

Legende: OMV = Old Method Version (vor der Umsetzung); NMV = New Method Version (Nach der Umsetzung)

Bei Ummodellierungsoperator	IF Condition	THEN Action
Input-added	OMV.State >= „AcceptedForExecution“	NMV.State = „AcceptedForExecution“
Output-added	OMV.State > „ExecutionStarted“	OMV.State = „ExecutionStarted“
Invariant-modified	OMV.State > „Initialized“ AND (NOT (OMV.Invariant ⇒ NMV.Invariant))	NMV.State = „Initialized“
Postcondition-modified	(OMV.State = „PostconditionSatisfied“) AND (NOT (OMV.Postcondition ⇒ NMV.Postcondition))	NMV.State = „ExecutionFinished“ *AND (FORALL Outputs in Postcondition (NMV.Output.State = „InValidation“))*
	(OMV.State = „Exception“) AND (NOT (OMV.Postcondition ⇒ NMV.Postcondition))	NMV.State = „ExecutionFinished“
	(OMV.State = „ExecutionFailed“) AND (NOT (OMV.Postcondition ⇒ NMV.Postcondition))	NMV.State = „ExecutionFinished“

Tabelle 12: Reaktionen bei der Umsetzung von atomaren Methoden

Legende: OMV = Old Method Version (vor der Umsetzung); NMV = New Method Version (Nach der Umsetzung)

5.4.5.3 Der Umsetzungsalgorithmus

Wir beschäftigen uns nun mit der Frage, welche Schritte im einzelnen bei der Umsetzung einer Ummodellierung zu tun sind. Wir gehen also jetzt von einer Ummodellierung aus, deren Sichbarkeit auch die Modellinstanz umfaßte, und die deshalb nun zu einem modifizierten Abhängigkeitsgraphen führt. Fassen wir nochmal zusammen, was alles im Rahmen der Umsetzung passieren muß:

- Modifikation der TMS-Strukturen
- Änderung der Zustände, wo notwendig

- Typtransformation

Nach dem Umsetzungsprozeß soll wieder ein konsistenter Zustand der Workflow-Abarbeitung erreicht werden. Aufgrund der Tatsache, daß im CoMo-Kit Zustände und Abhängigkeiten im selben TMS-Netzwerk verwaltet werden, geschieht das Ändern der Zustände letztendlich ebenfalls über die TMS-Modifikationen.
Während der Umsetzung darf der Plan nicht fortgeschrieben werden, der Zugriff auf den Abhängigkeitsgraphen bleibt verwehrt. Aus diesem Grund ist es sehr wichtig, die Umsetzung so schnell als möglich zu bewerkstelligen, damit keine zu großen Beeinträchtigungen der laufenden Arbeit stattfinden.
Die Eigenschaft der Sichtbarkeit einer Ummodellierung läßt erkennen, in welchen Modellsichten und Modellinstanzen eine Umsetzung der ummodellierung durchgeführt wird. Immer wenn für die Modellsicht eines Projektes die Ummodellierungen sichtbar sind, müssen für die zugehörige Modellinstanz die dem folgenden Algorithmus entsprechenden Modifikationen an den TMS-Strukturen dieser Instanz durchgeführt werden.

Modifikation der TMS-Strukturen

Wir weisen darauf hin, daß die genaue Implementierung des neuen Zustandsmodells noch nicht bekannt ist, so daß der hier vorgestellte Algorithmus eine gewisse Vorläufigkeit hat. Wir gehen von einer dem aktuellen Scheduler ähnlichen Implementierung aus, daß die Zustandsmodelle von Goals (Tasks), Operatoren (Methoden), Decisions und Variablen als TMS-Netzwerke realisiert sind, deren Abhängigkeiten die im CoMo-Kit üblichen Beziehungen zwischen diesen Einheiten wiederspiegeln. Dabei ergibt sich ein Baum, an dessen oberster Ebene die Goal-TMS-Struktur der Wurzeltask steht, in der zweiten Ebene sämtliche jemals für diese Task getroffenen Dekompositionsentscheidungen (darunter maximal eine gültige) in Form von Decision-TMS-Strukturen, in der dritten Ebene die zu diesen Entscheidungen gehörenden Methoden bzw. deren Operator-TMS-Strukturen, eine Ebene darunter wiederum die durch diese Operatoren dekomponierten Task-TMS-Strukturen mit den zugehörigen Assignment-TMS-Strukturen der Input- und Output-Parameter. Die Ebenen wechseln sich in dieser Reihenfolge ständig ab bis hinunter zu atomaren Methoden.
Wir nutzen auch hier das Denkmodell von der Ummodellierung als Methodenmodifikation aus Abschnitt 5.3.1.2. Gehen wir zunächst einmal von einer Ummodellierung aus, die nicht die Wurzeltask beeiträchtigt. Wir gehen von der Task aus, zu der alte und ummodellierte Methode gehören. Die Grundidee des Algorithmus ist, vom Ummodellierungsanker aus - nach der Einführung einer Decision für die neue Methode - ebenenweise aus der alten, aktiven Methode wiederverwendbare Teile in die neue Methode zu kopieren oder auf sie zu verweisen. Dies muß sowohl für alle gerade aktiven Methoden aller der neuen Methodenversion unterstehenden Subtasks geschehen, als auch für die inaktiven, die bereits zurückgezogen wurden, um bei deren möglicher erneuten Auswahl einen korrekten Zustand garantieren zu können. Die folgenden Schritte führen wir mit dem Abhängigkeitsgraphen durch (die die Aktionen auslösenden Ummodellierungsoperatoren sind kursiv dargestellt.):

- **Schritt 1**: Ummodellierung initialisieren:
  - a) Einführen einer neuen Decision-TMS-Struktur für die neue Methode der Task. (Gültigkeit von Design Rationales für diese Begründung durch Benutzerinteraktion ermitteln.)
  - b) Einführen einer neuen Operator-TMS-Struktur für die neue Methode.
- **Schritt 2**: Führe Dekomposition für diesen Operator durch:
  - a) Falls Subgoal bereits existent, so benutze alte Goal-TMS-Struktur (Verpointerung; dies bringt auch die zugehörigen Variablen-TMS-Struktu-

ren in die neue Methode ein).

b) Falls Subgoal nicht existent, so initialisiere neue Goal-TMS-Struktur (Dies beinhaltet auch die zugehörigen Variablen-TMS-Strukturen).

- **Schritt 3**: Einbringen der operatorbezogenen Modifikationen:

a) Falls *Invariant-Modified(Operator)*, so initialisiere Operator neu (d.h. Löschen und Neuschaffung der Subgoals und Variablen)

Sonst:

i) Falls Operator komplex:

1) Falls *Subtask-Added(Operator, aSubGoal)*, so initialisiere Subgoal neu.

2) Falls *Formal-Parameter-Added(Operator, Variable)*, so initialisiere neue Variablen-TMS-Struktur.

3) Falls *Subtask-Retracted(Operator, aSubgoal)*, so lösche entsprechende Subgoal-TMS-Struktur.

4) Lösche alle Variablen-TMS-Strukturen ohne Verbindungen zu Goals.

ii) Falls Operator atomar:

1) Falls *Input-Added(Operator)* oder *Output-Added(Operator)*, so initialisiere neue Variablen-TMS-Strukturen (falls bereits vorhanden, etabliere Input- und Output-Beziehungen).

2) Falls *Input-Retracted(Operator)* oder *Output-Retracted(Operator*), so lösche weggefallene Bindungen an Variablen-TMS-Strukturen.

b) Falls *Postcondition-Modified (Operator)*, so setze Methode zurück entsprechend Tabelle 6 (komplex) bzw. Tabelle 6 (atomar)[1].

- **Schritt 4**: Einbringen der goalbezogenen Modifikationen:

a) Falls *Method-Retracted(Goal, aChosenMethod)*, so lösche Decisionstrukturen für diese Methode.

b) Falls *Agent-Retracted(Goal, RetractedAgent)* und Goal.Agent = RetractedAgent, so invalidiere Decision.

c) Falls *Invariant-Modified(Goal)*[2], so initialisiere Goal neu und lösche bisherige Decisions.

d) Herstellen der neuen Parameterstrukturen:

Einführen neuer Planning-Input- und Output-Assignments zum Goal (insbes. der Variablen-TMS-Strukturen).

Löschen bestehender Verbindungen bei weggefallenen Inputs/Outputs.

Löschen der Variablen-TMS-Strukturen, falls keine Verbindung mehr zu irgendeinem Goal besteht.

e) Falls *Method-Added(Goal, Method)* und es existiert eine gültige Entscheidung für diese Goal, so:

i) mache Entscheidung ungültig (retract-decision)

ii) instanziiere neue Decision-TMS-Struktur für neue Methode

iii) instanziiere neue Operator-TMS-Struktur für neue Methode

Falls Subgoal bereits existent, so benutze alte Goal-TMS-Struk-

1. Wir legen hier schwache Postconditions zugrunde, die keine temporalen Prädikate enthalten.
2. Wir lassen hier generell die Einschränkung bezüglich der Tautologie bei den Bedingungskonstrukten außer acht.

tur (Verpointerung; dies bringt auch die zugehörigen Variablen-TMS-Strukturen in die neue Methode ein).

Falls Subgoal nicht existent, so initialisiere neue Goal-TMS-Struktur (Dies beinhaltet auch die zugehörigen Variablen-TMS-Strukturen).

- **Schritt 5**: Für alle gültigen und ungültigen Decisions der Methoden der Goals einer Ebene neue Parameterstrukturen herstellen:
  a) Falls Decision.Method komplex:
    i) Einführen neuer Input-Assignments (entsprechend hinzugekommener Planning-Inputs des zugehörigen Goals).
    ii) Löschen bestehender Inputs (entsprechend gelöschter Planning--Inputs des zugehörigen Goals)

  Falls Decision.Method atomar[1]:
    i) Einführen neuer Input-Assignments (entsprechend hinzugekommener Planning-Inputs des zugehörigen Goals *und* der Execution-Inputs der Methode).
    ii) Löschen bestehender Inputs (entsprechend gelöschter Planning-Inputs des zugehörigen Goals *und* gelöschter Execution-Inputs der Methode).

  b) Einführen neuer Output-Assignments (entsprechend hinzugekommener Outputs der zugehörigen Methode) und Löschung der Beziehungen zu gelöschten Output-Assignments.

  c) Falls Decision.Method irgendeine Modifikation erfahren hat:
  In Kooperation mit dem Benutzer ermittelt werden, ob begründende Design Rationales noch gültig sind.
- **Iteration**: Wiederhole die Schritte 3-5 Ebene für Ebene, bis atomare Methoden erreicht sind.
- **Letzter Schritt**: Typtransformationen in allen betroffenen Variablen.

Nach Durchlaufen dieses Algorithmus haben alle Goals, Operatoren, Decisions und Variablen die in den Tabellen 5, 6 und 6 beschriebenen Zustände. Wir müssen jetzt die alte Methode noch zurückziehen, und diesen Rückzug mit einem Design Rationale begründen, falls wir die Ummodellierung nochmal rückgängig machen wollen. In diesem Fall gehen wir davon aus, daß die Verpointerung in den Schritten 2a) und 4e)iii) nur dann Verpointerungen bleiben, falls die entsprechenden Goals nicht modifiziert wurden. Ansonsten müssen alle Strukturen kopiert werden. Vom Algorithmus wird dann immer die Kopie modifiziert, so daß die alten Strukturen erhalten bleiben, und bei einem Rückzug der Ummodellierung diese wieder zur Verfügung stellen zu können. Ansonsten können die nicht mehr benötigten alten Strukturen aus dem TMS-Netz entfernt werden. Gleichzeitig mit diesen Vorgängen wird die Entscheidung für die neue Methodenversion gültig, und der Plan kann weiterabgearbeitet werden.
Wird auch die Wurzeltask geändert, so werden zunächst einmal die laut Tabelle 5 notwendigen Modifikationen der Wurzeltask erledigt, bevor dann die weiter Umsetzung nach obigem Algorithmus geschieht.

1. Diese Art und Weise der Modellierung, sowohl die Methodenauswahl als auch die konsumierten Inputs und produzierten Outputs dieser Methode an *eine* Decision zu binden, wird z.Zt. überdacht und eventuell geändert.

Durchführung der Typtransformationen

Variablenbelegungen von Produkttypen, die aus anderen Produkttypen ohne Verwendung der strukturmodifizierenden Ummodellierungsoperatoren Slot-Added/Retracted oder Attribute-Added/Retracted hervorgegangen sind (weiche Produkttypmodifikationen), müssen nunmehr noch automatisch transformiert werden. Wir gehen davon aus, daß nur erlaubte Modifikatiuonen (siehe Tabelle 4 auf Seite 51) durchgeführt wurden. Variablen von Produkttypen, bei denen lediglich Slot-/Attribute-Retracted-Ummodellierungen auftauchten, sind durch daß Löschen der Abhängigkeiten im TMS-Graph und die damit verbundene Instanzmodifikation bereits konvertiert worden. Variablen von Produkttypen, die um weitere Teile ergänzt wurden, müssen von den produzierenden Aufgaben und Methoden zunächst noch mit einer Wertergänzung versehen werden; die Produzenten wurden durch die TMS-Restrukturierung in entsprechende Zustände versetzt.

Ist dies alles geschehen, so gilt die Ummodellierung als abgeschlossen (d.h. die Ummodellierungs-TA hat committed), die Arbeit kann normal fortgesetzt werden.

5.4.5.3.1 Ein Beispiel zur Transformation des Abhängigkeitsnetzes

Wir betrachten auch hier wieder als Beispiel die schon aus Abschnitt 3.2 bekannte Situation, die wir hier nochmals in Abb. 11 darstellen. Wir betrachten zunächst lediglich die Änderung

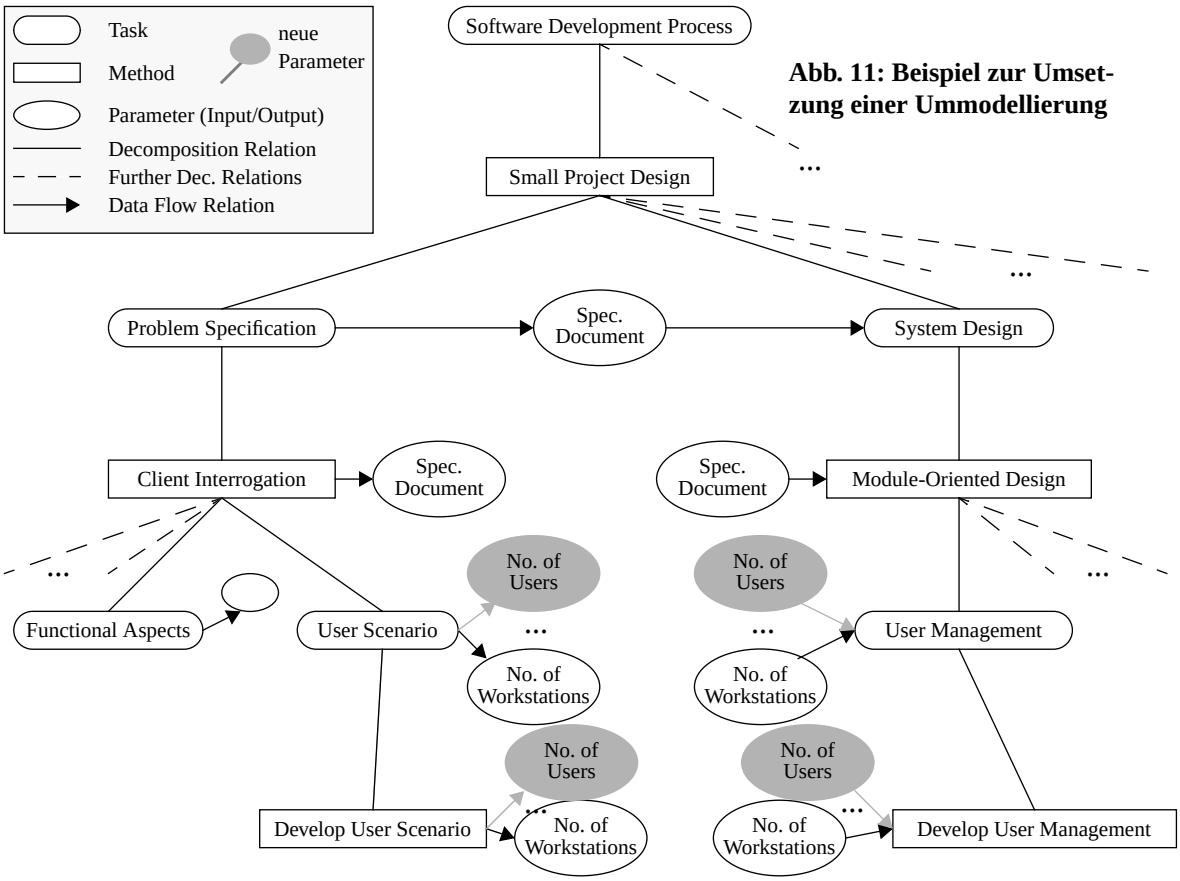


Abb. 11: Beispiel zur Umsetzung einer Ummodellierung

des Datenflusses aus dem Beispiel, in Variationen nehmen wir u. a. noch an, daß sich eine Invariante bzw. eine Postcondition geändert haben. Die verwendeten (und vom Modeler aufgezeichneten) Ummodellierungsoperatoren des Grundbeispiels und der Variationen können Tabelle 13 entnommen werden. Bei der gesamten Ummodellierung handele es sich um eine Debug-Ummodellierung und Korrektur, d.h. die Ursache der Ummodellierung waren Model-

Atomare Ummodellierungsoperatoren	
Method:Output-Added	(DevelopUserScenario, NoOfUsers)
Task:Output-Added	(UserScenario, NoOfUsers)
Method:Output-Type-Modified	(ClientInterrogation, SpecificationDocument, SpecWithoutNoOfUsers, SpecWithNoOfUsers)
Task:Output-Type-Modified	(ProblemSpecification, SpecificationDocument, SpecWithoutNoOfUsers, SpecWithNoOfUsers)
Method:Formal-Parameter-Type-Modified	(SmallProjectDesign, SpecificationDocument, SpecWithoutNo-OfUsers, SpecWithNoOfUsers)
Task:Execution-Input-Type-Modified	(SystemDesign, SpecificationDocument, SpecWithoutNoOfUsers, SpecWithNo-OfUsers)
Method:Input-Type-Modified	(ModuleOrientedDesign, SpecificationDocument, SpecWithoutNoOfUsers, SpecWithNoOfUsers)
Task:Input-Added	(UserMangement, NoOfUsers)
Method:Input-Added	(DevelopPasswordSystem, NoOfUsers)
Method:Input-Added	(DevelopUserManagement, NoOfUsers)

Tabelle 13: Ummodellierungsoperatoren im Beispiel für die Umsetzung einer Ummodellierung

lierungsfehler und die Änderungen sollen auf allen Modellierungsebenen Generisches Modell, Modellsicht und Modellinstanz sichtbar werden.
Wir gehen von der aus Abb. 11 ersichtlichen Situation aus. Das Bild zeigt die TMS-Strukturen des Abhängigkeitsnetzwerkes zu dem Zeitpunkt, in dem das Scheduling der Arbeitsprozesse für die Umsetzung der Ummodelierung eingefroren wurde. Die Beziehungen zwischen den TMS-Strukturen sollen nur ausdrücken, daß diese sich in irgendeiner Art und Weise direkt aufeinander auswirken[1]. Die eigentlich bestehenden direkten Beziehungen zwischen Goals und Operatoren wurden der Übesichtlichkeit halber weggelassen. Es sind nur für das Beispiel relevante Objekte ausgedrückt; über weiter Inputs/Outputs machen wir keine Aussage.
Aus dem Netz geht hervor, daß früher versucht wurde, die Aufgabe „Software Development Process“ mit Hilfe der Methode „Medium Project Design“ versucht wurde, zu lösen. Da diese jedoch einen dem Problem unangemessenen Aufwand darstellt, wurde sie mit einem entsprechenden Design Rationale zurückgezogen und die Methode „Small Project Design“ ausgewählt. Innerhalb deren Bearbeitung wurde die Subtask „Problem Specification“ abgearbeitet und die Aufgabe „System Design“ wird gerade bearbeitet. Dort wurde für die Subtask „User-Management“ zunächst die Methode „DevelopPasswordSystem“ und erst später die Methode „DevelopUserManagement“ ausgewählt. Zur Zeit befindet sich diese letzte Methode im Zustand „AcceptedForExecution“, d.h. ein Agent hat sich für diese Task/Methode zuständig erklärt, aber noch nicht mit der Arbeit begonnen.
Bei den Produkten sind sowohl das vorhandene Produkt „NoOfWorkstations“ als auch das hinzukommende Produkt „NoOfUsers“ Slots im Konzept „SpecificationDocument“ bzw. „SpecificationDocumentNew“.
Durch den oben beschriebenen Algorithmus soll dieses Netzwerk in ein der neuen Modellierung genügendes Netzwerk umgewandelt werden. Wir wollen diese Umsetzung Schritt für Schritt verfolgen. Die Entwicklung des Algorithmus ist aus Abb. 11 ersichtlich.

- **Startphase**: Wir stellen als Ummodellierungsanker zunächst die Wurzeltask fest. Folgen wir dem ersten Schritt und initialisieren sowohl eine neue Decision als auch

1. Es existiert z.Zt. noch keine genauere Spezifikation dieser Beziehungen. Sie sollen allgemein sicherstellen, daß die Zustände der TMS-Strukturen konsistent miteinander sind.

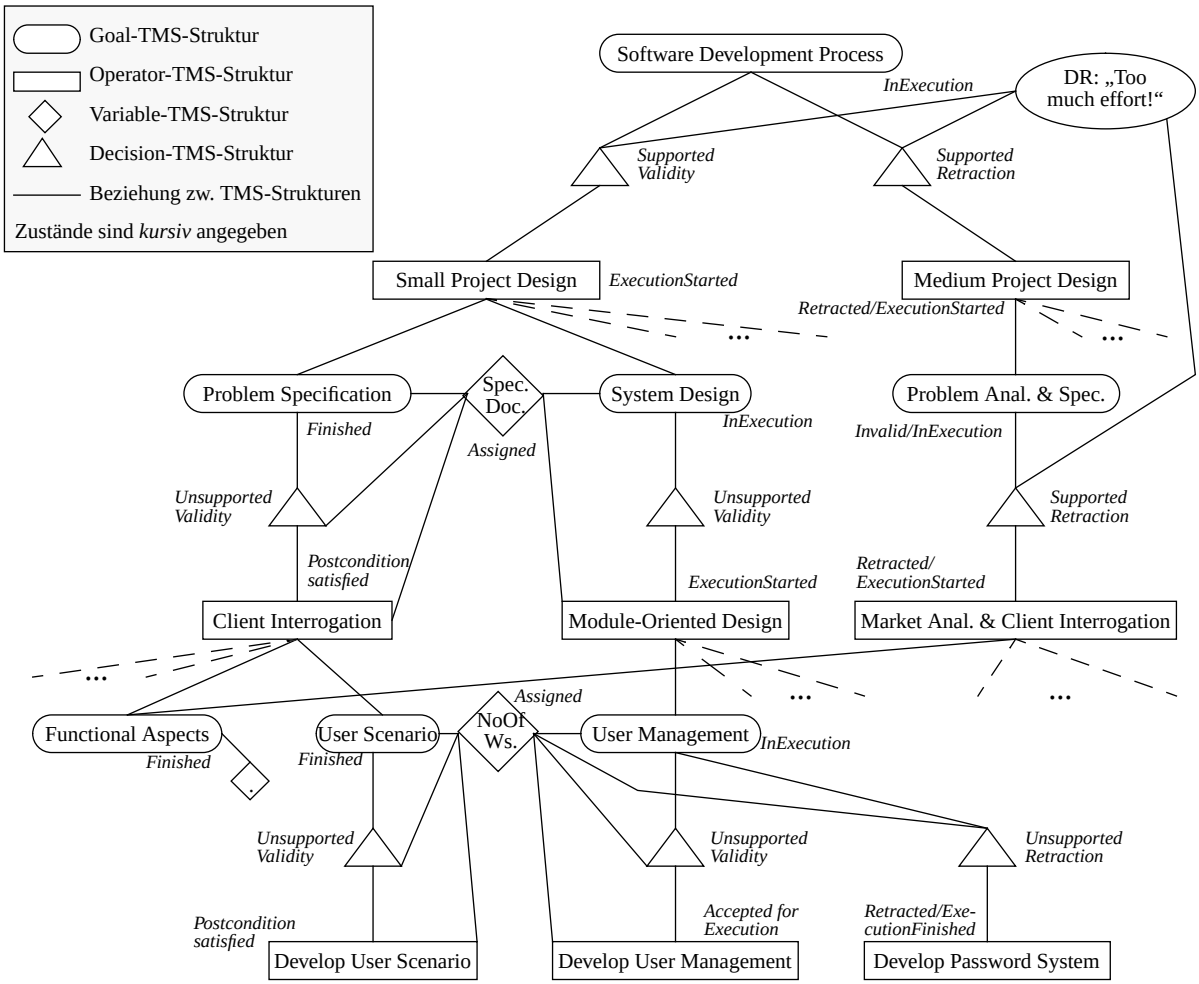


Abb. 12: Zustands- und Abhängigkeitsgraph vor der Ummodellierung

einen neuen Operator „SmallProjectDesignNew“. Bei dessen Dekomposition (zweiter Schritt) können alle Aufgaben übernommen werden (also identische TMS-Strukturen mit der alten Version, d.h. von hier an operieren wir nur auf bestehenden Strukturen), da sich an der Struktur der Dekomposition nichts geändert hat. Beim Erstellen der neuen Decision muß die Übernahme des Design Rationales geregelt werden (Hier entscheidet sich der Umplaner, daß DR zu übernehmen).

- **Phase 1**: Der erste Durchlauf von Schritt 3 bleibt leer, da in dem Operator keine Änderungen dieser Art aufgetreten ist. Auch in den Schritten 4 und 5 passiert nichts, da keine entsprechenden Modifikationen gemacht wurden. Zu diesem Zeipunkt sind wir bereits auf der Ebene der Methoden „Client Interrogation“ und „Module-Oriented Design“ angekommen.
- **Phase 2**: Im zweiten Durchlauf von Schritt 3 ergibt sich wiederum keine Ummodellierung, da der neue Output der Task nicht als Formaler Parameter in der Methode instanziiert wird.
  In Schritt 4 endlich kommen das Hinzufügen eines Outputs zur Task „User Scenario“ zum tragen: eine entsprechende neue Variablen-TMS-Struktur wird eingebracht und in den Zustand „Unassigned“ versetzt. Als Folge hiervon werden durch die im CoMo-Kit übliche Zustandspropagierung auch die Tasks „User Scenario“ und „Problem Specification“ in den Zustand „InExecution“ zurückversetzt. Da die

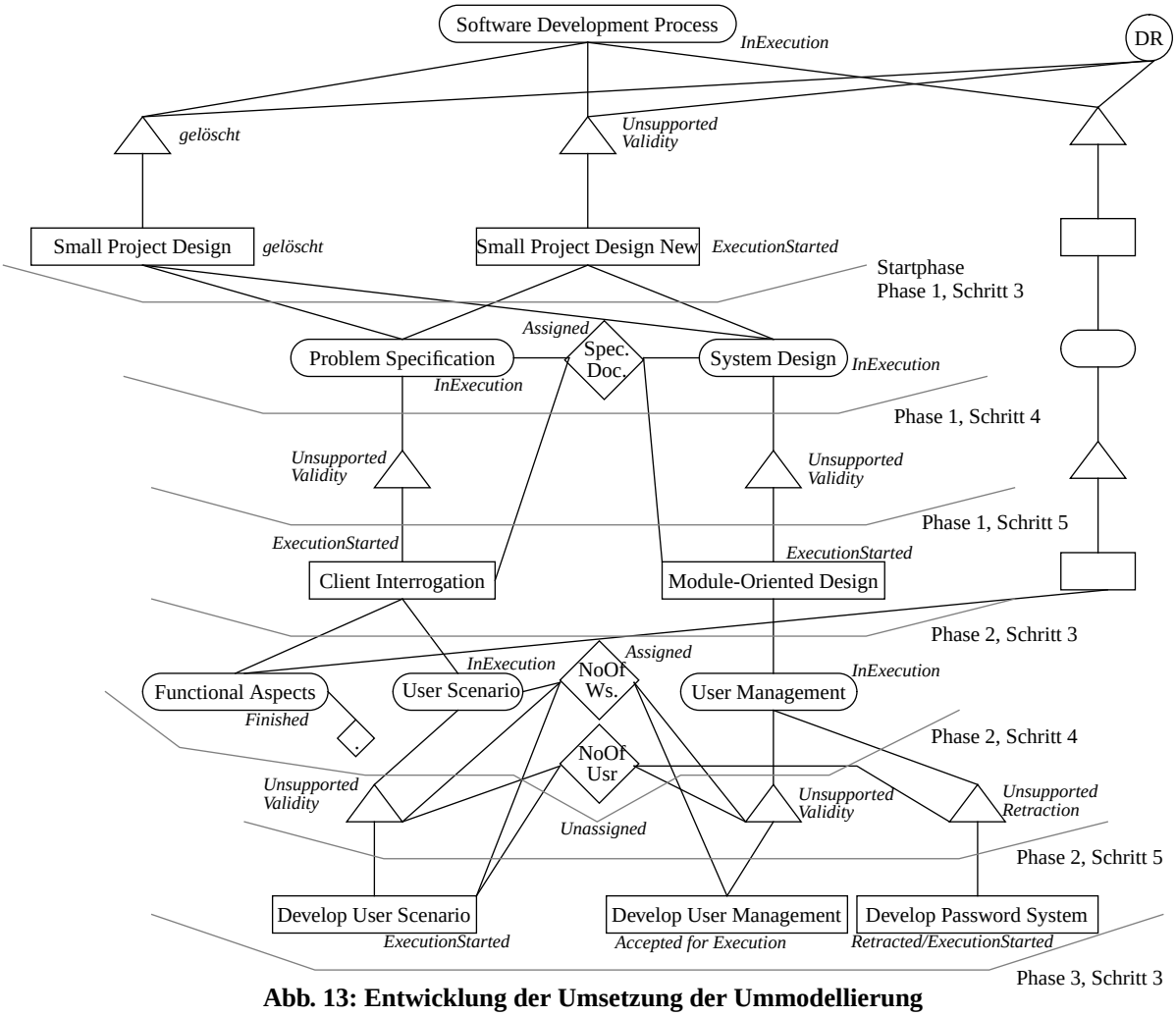


Abb. 13: Entwicklung der Umsetzung der Ummodellierung

(Die Legende entspricht der aus Abb. 11.)

neue Variable für die Task „User Management“ nur einen Execution-Input darstellt, entsteht zwischen der Goal-TMS-Struktur und der neuen Variablen-TMS-Struktur keine Beziehung.

(Variation 1: Wir weisen hier darauf hin, daß in diesem Schritt natürlich auch alle anderen Tasks überprüft werden, insbesondere auch die Task „Functional Aspects“, die jedoch im Rahmen der betrachteten Ummodellierung-TA keine Modifikationen erlebte. Würde dies nicht so sein, nähmen wir z.B. an, daß diese Task ebenfalls einen neuen Output bekommen würde, so würde sich auch dort dadurch der Zustand ändern, und diese Zustandsänderung würde *in einem völlig anderen Teil des Abhängigkeitsgraphen* (Methode „Method Analysis & Client Interrogation“ und Vorgänger) andere Zustandsübergänge triggern, da diese Aufgabe auch Subtask der Methode „Market Analysis & Client Interrogation“ wäre.)

(Variation 2: Nehmen wir an, daß die Invariante der Task „User Management“ modifiziert wurde. In diesem Fall würden als Reaktion alle Decisions und darunterliegende Strukturen gelöscht. Diese Strukturen müßten bei einer erneuten Auswahl völlig neu initialisiert werden.)

(Variation 3: Sei nun die Postcondition der Task „User Scenario“ modifiziert worden. Dies hätte in diesem Schritt keine weiteren Folgen. Die notwendigen

Umstrukturierungen würden in Phase 3, Schritt 3 durchgeführt.)
In Schritt 5 hätten wir es bei allen in Abb. 11 aufgeführten Decisions dieser Ebene mit Entscheidungen für atomare Methoden zu tun. In 5a) kommt also zur Decision für die Methoden „Develop User Management“ und „Develop Password System“ jeweils eine Beziehung zur in Schritt 4 neu geschaffenen Variablen-TMS-Struktur hinzu. In 5b) erhält auch die Decision für die Methode „Develop User Scenario“ eine entsprechende (Output-)Beziehung dorthin.

- **Phase 3**: Der dritte Durchlauf von Schritt 3 ist für die in Abb. 11 dargestellten Modellierungseinheiten der letzte Teil der Iteration. Die entsprechenden Operatoren erhalten neue Beziehungen zu der neuen Variablen-TMS-Struktur. Durch diese Installation von Beziehungen werden auch hier die Zustände der Operatoren modifiziert. So wechselt der Operator „Develop Password System“ von „ExecutionFinished“ nach „ExecutionStarted“, da ein neuer Input hinzukam.
  (Variation 3: Die Modifikation der Postcondition der Supertask bedeutet auch immer die Modifikation der Postcondition der Methoden. Involviert in diesem Fall der neue Teil der Postcondition den neuen Output, so muß mit dem Abtesten der Postcondition gewartet werden, bis dieser erstellt ist. Ist dies nicht der Fall, so müßte die Postcondition nur nochmal überprüft werden; und die Zustände müßten sich nicht unbedingt ändern. Da es jedoch schwierig ist, zu bestimmen, ob eine Postcondition nochmal überprüft werden sollte oder nicht, gehen wir hier einfach von einer Notwendigkeit zur Überprüfung aus.)
  Natürlich kann die Iteration für andere Teilbäume noch weiter durchlaufen (so noch nicht bearbeitete Ummodellierungen noch vorhanden sind).
- **Phase X**: Typmodifikation muß hier nur in der Variablen „Specification Document“ durchgeführt werden.

Die in Abb. 11 angegebenen Zustände repräsentieren den Abarbeitungszustand nach der Umsetzung der Ummodellierung.
Wir möchten nochmals darauf hinweisen, daß es sich bei diesem Algorithmus um eine „erste Näherung“ handelt. Die genaue Vorgehensweise kann erst nach Festlegung des Zusammenspiels der Modellierungseinheiten erfolgen. In Abschnitt 5.5.2 adaptieren wir diesen Algorhitmus für das „alte“, der aktuellen Implementierung entsprechende Zustandsmodell des CoMo-Kit.

5.4.6 Das Konzept der Ummodellierungstransaktion

Mit dem Konzept der Ummodellierungstransaktion wollen wir den Anforderungen einer hohen Verfügbarkeit des WFMS und einer möglichst großen Freiheit bei der Spezifikation von Ummodellierungen Rechnung tragen. Außerdem sollen die in den vorhergehenden Abschnitten von Abschnitt 5.4 erarbeiteten Konzepte im Rahmen dieser Ummodellierungstransaktion operationalisiert werden. Wir beschreiben diese Operationalisierungen zunächst zusammen mit der Beschreibung der Ummodellierungstransaktion. Danach gehen wir im Rahmen der Rollback- und Recoveryproblematik auf das Scheitern und das Rücksetzen einer Ummodellierungstransaktion ein. Zum Schluß betrachten wir noch Aspekte der verteilten und parallelen Ummodellierens.

5.4.6.1 Beschreibung der Ummodellierungstransaktion

Wir betrachten eine Ummodellierungstransaktion als eine Einheit, in der alle zu einer Ummodellierung gehörenden Maßnahmen gekapselt werden. Wir können eine Ummodellierungstransaktion in eine Spezifikations- und eine COMMIT-Phase unterteilen (entspricht Modellierungs- und Umsetzungsphase unseres Zeitmodells). Im Rahmen der Spezifikationsphase wird die gesamte Ummodellierung vom Umplaner spezifiziert. Beginn und Ende dieser Phase müssen vom Umplaner explizit spezifiziert werden, da es keine Möglichkeit gibt, das Ende einer Ummodellierung anhand von Useraktionen automatisch abzuleiten. Dabei ist es sinnvoll, die Sichtbarkeit der Ummodellierung ebenfalls zu Beginn anzugeben, da eine Sichtbarkeit, die auch die Modellinstanz umfaßt, eine Umsetzung der Ummodellierung impliziert, die, wie wir in Abschnitt 5.4.3 (Arbeitersparnis) und Abschnitt 5.4.4 (Wiederverwendung) gesehen haben, bestimmte Reaktionen hervorrufen sollte.
Uns ist bewußt, daß damit dem Umplaner gestattet wird, in den Verantwortungsbereich eines Planers einzugreifen (Nämlich dann, wenn er die Modellinstanz als von einer Ummodellierung betroffen spezifiziert). Wir halten diesen Aspekt für nicht weiter bedenklich, da wir davon ausgehen, daß das Modifizieren eines Planes eine größere Kompetenz erfordert als die Auswahl eines Planes unter mehreren Alternativen. Zudem kann der Planer von der Modifikation informiert werden.
Der Umplaner ist dann absolut frei in der Spezifikation der von ihm gewünschten Ummodellierungen (Flexibilität!). Ab diesem Zeitpunkt müssen alle Modifikationen in Form der verwendeteten atomaren Ummodellierungsoperatoren aufgezeichnet werden („Protokollierung der Ummodellierung“, Abschnitt 5.4.2). Innerhalb dieser Ummodellierungstransaktion können die Events FNU, BMU, EMU und ENU bezüglich jeder zu modifizierenden Modellierungseinheit auftreten, z.B. also BMU(Task „Report schreiben“). Das Event FNU sollte explizit vom Umplaner initiiert werden können oder bei Beginn der Modellierung (bzw. Modifikation) einer Modellierungseinheit systemseitig ausgelöst werden. Eine weitere Struktur innerhalb der Ummodellierungstransaktion ist nicht determinierbar. Die FNU einer Supertask muß nicht zeitgleich der FNU der zugehörigen Methoden sein und das FNU-Event einer in einer dieser Methoden vorkommenden Subtask muß nicht mit einem der beiden übergeordneten FNU-Events zusammenfallen. Es mag sogar vorkommen, daß das FNU-Event einer Subtask vor dem FNU-Event ihrer Supertask auftritt. Dies liegt daran, daß der Umplaner nur selten in der Lage ist, die Folgen seiner Ummodellierungen (also die Kaskadierung der Ummodellierungen) bereits zum Zeitpunkt FNU der initialen Ummodellierung zu übersehen. Deshalb läßt sich *keine zeitliche Reihenfolge der Ummodellierungsevents* und *keine Verschachtelungshierarchie* zwischen verschiedenen in einem Ummodellierungsprozeß vorkommenden Ummodellierungen spezifizieren. Wir können aus diesem Grunde Ummodellierungstransaktionen auch nicht an feinen Granulaten wie z.B. Modellierungseinheiten festmachen, sondern betrachten stets die Modifikation des Modells als Ganzes.
Um die Weiterverarbeitung im WFMS während der Spezifikationsphase zu gewährleisten (Verfügbarkeit!), sollte die Ummodellierungstransaktion nicht auf den Originalstrukturen operieren. Wir gehen darauf in Abschnitt 5.4.6.2 ein.
Zum Ende der Spezifikationsphase der Ummodellierungstransaktion sollte das Modell wieder in einem konsistenten Zustand sein. Dies könnte zu diesem Zeitpunkt von einer Systemkomponente entweder durch Prüfen des neuen Modells oder, ausgehend von der Annahme, zu Beginn der Ummodellierung ein konsistentes Modell vorliegen zu haben, durch Überprüfung der mitgeloggten Änderungen.
In der COMMIT-Phase sollen die spezifizierten Ummodellierungen abhängig von der Sichtbarkeit der Ummodellierung entweder einfach nur ins Generische Modell/ in die Modellsicht

eingebracht werden oder zusätzlich die Umsetzung der Ummodellierung in der Modellinstanz anhand des in beschriebenen Algorithmus durchgeführt werden. Dies sollte vor dem Einbringen der Modifikationen in das Workflow-Modell geschehen, da dann im Falle des Scheiterns der Umsetzung das Einbringen nicht wieder rückgängig gemacht werden muß (auch dazu siehe Abschnitt 5.4.6.2).

5.4.6.2 Rollback- und Recoveryproblematik der Ummodellierungstransaktion

Wir müssen damit rechnen, daß eine Ummodellierungstransaktion u.U. (gewollt oder ungewollt) abgebrochen wird. In diesem Fall müssen wir gewährleisten, daß das eigentliche Modell von diesen Aktionen unbehelligt bleibt. Wir müssen also eine Art *Modifikationspufferung* durchführen, zu der wir drei Realisierungsalternativen haben:

- *Modellkopie*: Eine Möglichkeit ist, Änderungen nur auf einer Kopie des Modells durchzuführen (was bei großen Modellen sehr speicherintensiv werden kann) und erst beim COMMIT der Ummodellierungs-TA das eigentliche Modell zu ersetzen.
- *partielle Modellkopie*: Bei einer zweiten Möglichkeit gehen wir prinzipiell genauso vor, versuchen jedoch, durch Ausnutzen des methodenbasierten Denkmodells ein feineres Kopiegranulat zu erreichen. Dabei fassen wir den Ummodellierungsprozeß tatsächlich als eine neue Methode auf. Da zunächst unbekannt ist, zu welcher Subtask die den Ummodellierungsprozeß repräsentierende Methode gehört (wir wissen nicht, welche Modellierungseinheiten von dieser Ummodellierung betroffen sein werden), müssen wir den Änderungskontext (d.h. die Methodenebene, von der wir annehmen, daß sie alle Änderungen umfassen wird) laufend erweitern, wenn eine Modifikation ausserhalb des bisher festgelegten Änderungskontextes stattfindet. Fängt die Ummodellierung im Beispiel in Abschnitt 3.2 auf Seite 20 mit dem Hinzufügen eines Inputs zur Methode „Develop User Management“ an, gehen wir zunächst von einer Modifikation aus, die in der Methode „Module-Oriented Design“ gekapselt werden kann und kopieren deren Struktur. Wird dann irgendwann der Output der Task „User Scenario“ hinzugefügt, so ist der neue Änderungskontext die Ebene der niedrigsten Methode, die alle Änderungen umfasst, also die Ebene der Methode „Small Project Design“ und die fehlenden Strukturen werden dann der bestehenden Kopie beigefügt. So erreichen wir ein feineres Granulat der Modifikationskopie.
- *Versionierung*: Die dritte Möglichkeit wäre ein ausgefeiltes, die Semantik der Modellierungsmöglichkeiten im CoMo-Kit berücksichtigendes Versionierungskonzept für Modellierungseinheiten. Dies hätte vor allen Dingen Vorteile bei der Realisierung paralleler Ummodellierungsprozesse (s.u.).

Mit dieser ModifikationspufferungKohärenzprobleme bei den ersten beiden Möglichkeiten können wir solange ausschliessen, wie nicht mehrfach sich überschneidende Ummodellierungen parallel durchgeführt werden (hierzu auch s.u.).

Das Scheitern einer Ummodellierung

Scheitert die Ummodellierungstransaktion aus technischen (z.B. Systemabsturz) oder semantischen (z.B. Abbruch durch Benutzer) Gründen, so müssen (neben einer Rücknahme der Reaktionen, die wir in Abschnitt 5.4.4 beschreiben) am Modell selbst keine Änderungen erfolgen, sondern lediglich die Modifikationskopie bzw. die neuen Versionen gelöscht werden.
Es stellt sich die Frage, was passiert, wenn die *Umsetzung* einer Ummodellierung (d.h. die Transformation der Abhängigkeitsstrukturen) in der Modellinstanz scheitert. Die Alternativen

sind, die gesamte Ummodellierungstransaktion bei einem Scheitern der Umsetzung als gescheitert zu betrachten, oder Ummodellierung und Umsetzung als unabhängige Schritte zu betrachten. Wir halten erstere Möglichkeit für die bessere, da zum einen bereits zu Beginn der Ummodellierungstransaktion bestimmt wird, ob die Modellinstanz betroffen sein soll oder nicht, dies also eine charakterisierende Eigenschaft der Ummodellierungstransaktion ist; zum anderen, weil es sonst eine Inkonsistenz zwischen Modellsicht und Modellinstanz geben könnte, was jedoch nicht erlaubt ist. So gesehen gehört die Umsetzung in der Modellinstanz zur COMMIT-Phase einer Ummodellierungstransaktion.
Gerade bei Abweichungen mag es nützlich sein, Ummodellierungen auch nach der gelungenen Umsetzung (nach dem COMMIT der Ummodellierungstransaktion) zurücknehmen zu können. In dem Fall sollten die „alten“ Strukturen im Plan mit Hilfe eines speziellen, nicht durch Benutzer manipulierbaren Design Rationales invalidiert werden, um sie notfalls wieder validieren zu können. Im Falle der Rücknahme solcher Ummodellierungen muß (zumindest bei Debug-Ummodellierungen) zudem die alte Modellversion wiederhergestellt werden. Dies kann prinzipiell allerdings nur mit Hilfe einer Versionierungskomponente geschehen, die die verschiedenen Modellversionen verwaltet.

Rücksetzbarkeit von Ummodellierungen

Zur angestrebten Rücksetzbarkeit von lokal sichtbaren Ummodellierungen benötigt man neben den Maßnahmen im Ablaufplan natürlich auch eine gespeicherte Version des Zustandes vor der Ummodellierung. Wir orientieren uns hier an dem Kopiegranulat des Modifikationspuffers, dessen ursprüngliche Version wir vor der Ummodellierung als Backup abspeichern. Eine Versionierung wäre auch hier die elegantere und fexiblere Lösung.

5.4.6.3 Verteiltes und paralleles Ummodellieren

Paralleles Ummodellieren ist problemlos möglich, solange nur verschiedene Modellsichten ummodelliert werden und nicht das generische Modell. Wir betrachten nun die Problematik paralleler Ummodellierungen auf der gleichen Modellsicht bzw. dem Generischen Modell.
Werden mehrere Ummodellierungstransaktionen parallel gestartet, so werden aus dem Datenbankbereich bekannte Synchonisationsmechanismen notwendig. Hier können die im Zeitmodell beschriebenen Events zur Initiierung von Reservierungs- (FNU, BMU) und Freigabeaktionen (EMU) verwendet werden. Wie im Bereich Datenbanken stellt sich auch hier die Problematik, lange Sperren vermeiden zu wollen, und wie dort auch, wird man, um die gewünschte Fexibilität zu erreichen, ein Versionierungskonzept für die Modellierungseinheiten zu konzipieren haben.
Es sollte jedoch hinterfragt werden, inwieweit paralleles Ummodellieren sinnvoll ist. Unsere Darstellung der Kaskadierung von Ummodellierungen in Abschnitt 5.4.1 und im Beispiel in Abschnitt 3.2 weisen darauf hin, daß bereits relativ simple Ummodellierungen weitreichende Folgeänderungen im Modell nach sich ziehen können. Andererseits liegt es in der Natur von Workflow-Modellen, beim Design der Zerlegung von Aufgaben den einzelnen Subtasks einen möglichst hohen Grad an Autonomie zuzuweisen. Wir müssen uns dann fragen, wie hoch die Konfliktwahrscheinlichkeit bei parallelen Ummodellierungen ist. Die Folge einer hohen Konfliktwahrscheinlichkeit wäre, daß die schnelleren, optimistischen Synchronisationsverfahren nicht sinnvoll einsetzbar wären, und es bei einer Synchronisation über pessimistische Verfahren (Sperren) bzw. bei einem Versionierungskonzept mit hoher Wahrscheinlichkeit zu großen Behinderungen durch langlebige Sperren bzw. zu viele parallel gültigen Versionen käme. Da wir aber *ein* gültiges Modell benötigen, müssten vor einem sinnvollen Arbeiten mit dem

Modell alle Sperrkonflikte ausgeräumt bzw. alle parallelen Versionen zusammengeführt werden. Es wären gründlichere Untersuchungen über die Konfliktwahrscheinlichkeit notwendig, um diesen Aspekt abschliessend zu klären. Wir werden aufgrund dieser Problematik von der Zulassung paralleler Ummodellierungstransaktionen zunächst absehen[1].

5.5 Operationalisierung des Ummodellierungsmanagements

Wir können hier noch keine genauen Aussagen über die sämtliche Aspekte der Operationalisierung des Ummodellierungsmanagements machen, da die Implementierung des in Abschnitt 2 dargelegten Zustandsmodell nicht abgeschlossen ist.
Wir beschreiben zunächst die Einbettung des Ummodellierungsmanagements in den CoMo-Kit und schließen mit der Konzeptualisierung eines Prototypen, der ein Ummodellierungsmanagement auf der Basis der aktuellen CoMo-Kit-Implementierung realisiert.

5.5.1 Einbettung des Ummodellierungsmanagements in CoMo-Kit

Die wesentlichen Funktionalitäten des Ummodellierungsmanagements sind in der CoMo-Kit-Modellierungskomponente der *Ummodellierungs-Transaktionsmanager*, der die Ummodellierungstransaktionen verwaltet und das Protokollieren der Ummodellierungsoperatoren übernimmt, und in der CoMo-Kit-Schedulingkomponente der *Replanner*, der die Umsetzung eines Abhängigkeitsgraphen entsprechend der Ummodellierung vornimmt.

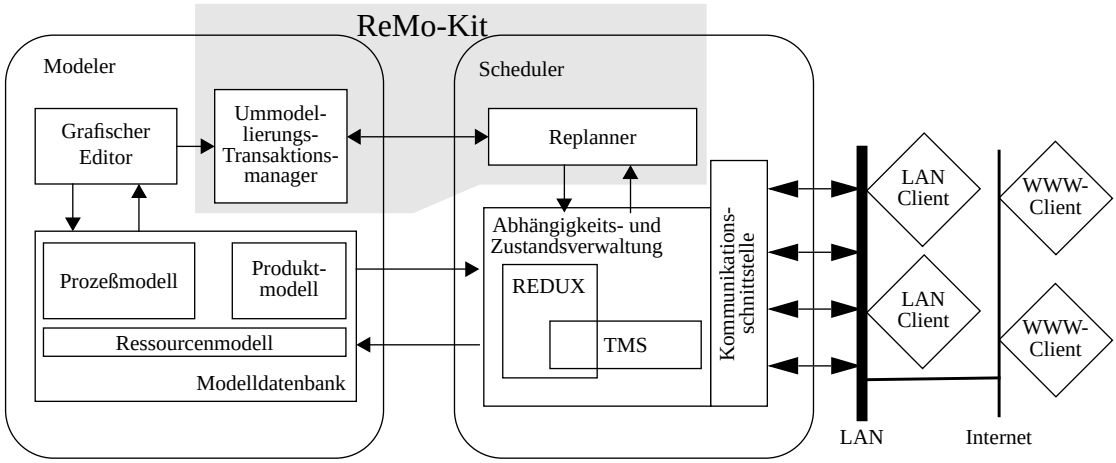


Abb. 14: Die erweiterte CoMo-Kit-Architektur

Wird im Grafischen Editor des Modelers eine Modifikation vorgenommen, impliziert dies das BEGIN einer Ummodellierungstransaktion (und damit auch das FNU-Event für die , deren genaue Eigenschaften (Sichtbarkeit und Modifikationstyp) der Benutzer zunächst spezifizieren

1. Es besteht eine tatsächliche Notwendigkeit für parallele Ummodellierungen, da diese z.B. bei den Agenten aufgrund der Beziehungen zu einem Realweltprozeß (Training zum Erlernen einer neuen Fähigkeit) durchaus monatelang dauern können (im Gegensatz zu z.B. Methodenmodifikationen, bei denen die Dauer der Ummodellierung der Willkür des Umplaners unterliegt). An dieser Stelle muß man sich dann fragen, ob der Nutzen einer präzisen Realitätsabbildung im Modell (hier: bei der Anmeldung des Agenten zum Lehrgang FNU-Event im System) die Kosten (Notwendigkeit der Verwaltung paralleler Ummodellierungen) rechtfertigt.

muß. Entsprechend der Modifikationspuffer-Strategie (nach Abschnitt 5.4.6.2 auf Seite 92) wird ein Duplikat des Modells angefertigt, auf dem dann die Ummodellierung modelliert wird. Die Änderungen auf dieser Kopie werden im Modeler erst vollständig wirksam, wenn eine ordnungsgemäße Umsetzung der Ummodellierung in der Schedulingkomponente sichergestellt ist.

5.5.2 ReMoKit: Ein Prototyp des Ummodellierungskonzeptes

In diesem Abschnitt beschreiben wir den das Konzept des *ReMo*delling-*Kit*, der als Erweiterung zum CoMo-Kit viele der im letzten Abschnitt beschriebenen Konzepte prototypisch realisieren soll.
Das Projekt CoMo-Kit erfährt z.Zt. erhebliche Erweiterungen und Überarbeitungen. Wir waren im vorangehenden Abschnitt bemüht, unsere Konzepte basierend auf den neuesten Spezifikationen aufzubauen. Da die Implementierung des CoMo-Kit mit der Konzeptentwicklung nicht immer Schritt halten konnte, müssen wir unsere Überlegungen für eine Implementierung in der bestehenden CoMo-Kit-Version adaptieren. Folgende Unterschiede zwischen dem neuen und dem alten CoMo-Kit-Modell müssen wir dabei beachten:

- CoMo-Kit unterstützt die in Abschnitt 5.3.1 aufgezeigte Sicht auf unterschiedliche Modellierungsebenen noch nicht. Z. Zt. wird bei jeder Modellmodifikation die Modellsicht aller Projekte gleichzeitig geändert. Eine übergeordnete Ebene mit einem „generischen Modell“ existiert zur Zeit nicht.
- Die aktuelle CoMo-Kit-Implementierung unterstützt das neue Zustandsmodell noch nicht.
- Der Scheduler unterstützt die im Modeler bereits spezifizierbaren Neuerungen wie z.B. Bedingungen und an Methoden gebundene Inputs noch nicht.

Aus diesen Gründen beschreiben wir hier einen Prototyp im Rahmen der alten CoMo-Kit-Version, die ein der alten Modellierung angepasstes Ummodellierungsmanagement realisiert.
Zum Teil sind zum Verständnis der dargestellten Modifikationen Kenntnisse der TMS-Strukturen im CoMo-Kit notwendig. Wir verweisen dafür auf die CoMo-Kit-Dokumentation und [Dell95].

5.5.2.1 Das alte CoMo-Kit-Modell

Im alten CoMo-Kit-Modell haben wir keine Bedingungen und keine Parameter für Methoden. Alle benötigten Eingaben und alle erstellten Ausgaben werden an die Aufgaben gebunden; es wird nicht mehr zwischen für die Planung einer Aufgabe und für die Ausführung einer Methode wichtigen Inputs unterschieden (d.h. alle Inputs werden bei komplexen Methoden immer als Planungs-Inputs betrachtet). Damit verbunden sind auch weniger Ummodellierungsmöglichkeiten (siehe Tabelle 14) und ein einfacheres Zustandsmodell (aus [Dell95], siehe Abb. 15).
Goals und Operatoren werden also in einem gemeinsamen Zustandsmodell und auch in einer gemeinsamen TMS-Struktur verwaltet (genaugenommen werden nur die Zustände des Goals betrachtet). In Variablen-TMS-Strukturen sind lediglich alle Assignments, die zu der Variable gehören, enthalten, gemeinsam mit einer TMS-Struktur, die als Zustände der Variablen „Assigned“ und „Unassigned“ zuläßt. Für Decisions existiert kein explizites Zustandsmodell, in die REDUX-TMS-Strukturen bezüglich Decisions kann man jedoch die Zustände „Valid“, „Retracted“ und „Rejected“ hineininterpretieren.

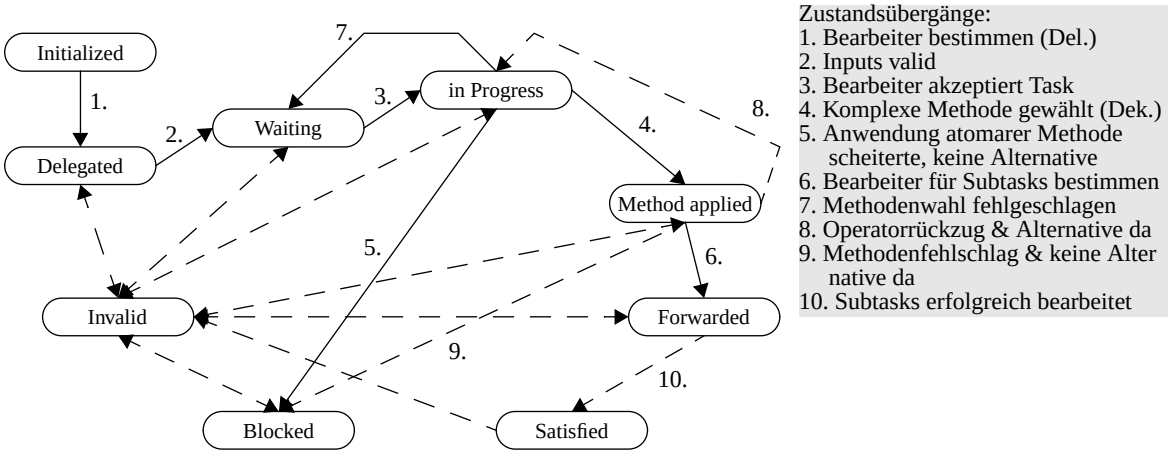


Abb. 15: Das alte CoMo-Kit-Zustandsmodell

Aufgabenmodell	Methodenmodell	Produktmodell
Input-added	Task-added	Slot-added
Input-retracted	Task-retracted	Slot-retracted
Input-Type-modified	Formal-Parameter-Added	Slottype-modified
Output-added	Formal-Parameter-Retracted	Slotcardinality-modified
Output-retracted	Formal-Parameter-Type-Modified	Attribute-added
Output-Type-modified	Parameter-Task-Relation-added	Attribute-retracted
Method-added	Parameter-Task-Relation-retracted	Attribute-Type-Modified
Method-retracted	Task-Parameter-Relation-added	-
Agent-added	Task-Parameter-Relation-retracted	-
Agent-retracted	-	-

Tabelle 14: Atomare Ummodellierungen im alten CoMo-Kit-Modell

Während sich damit für die Modellierungsphase nur eingeschränkt Änderungen gegenüber Abschnitt 5 ergeben, gilt unser Interesse vermehrt den Zusammenhängen, die bei der Umsetzung der Ummodellierung beachtet werden müssen und die sich aufgrund der anderen Semantik des alten CoMo-Kit-Modells auch anders verhalten.

5.5.2.1.1 Aspekte der Arbeitsersparnis und -wiederverwendung

Den Aspekt der Vermeidung überflüssig werdender Arbeit operationalisieren wir über eine neue bei den Tasks verankerte Notifikationsmöglichkeit zur Benachrichtigung von an dieser Task arbeitenden Agenten. Als Standardnotifikationen sind vorgesehen:

- *Task Insecure*: Die Task wird modifiziert, eine abschließende Bewertung des Ausmaßes der Modifikation ist z.Zt. nicht möglich. Diese Notifikation soll dem bearbeitenden Agenten lediglich die Möglichkeit lassen, zunächst andere, sichere Tasks zu bearbeiten.
- *Task Inapt*: Die Task wird modifiziert/gelöscht, eine weitere Bearbeitung scheint nicht sinnvoll.

- *Task Secure*: Erklärt die Task als sicher, d.h. der Bearbeitung wert. Dies macht vor allem dann Sinn, wenn eine übergeordnete Task „Insecure“ erklärt wurde, bestimmte Subtasks dieser Task jedoch auch später Verwendung finden werden.

Die Wiederverwendung bereits erledigter Aufgaben konnten wir mit Hilfe des Konzeptes der „Shared Subgoal Spaces“ im neuen CoMoKit-Modell an den Invarianten festmachen. Im alten CoMoKit-Modell gibt es noch keine Invarianten, und es läßt sich auch keine Methodeneigenschaft finden, die in ähnlicher Form eine Aussage über den Entscheidungskontext einer Methode repräsentieren könnte. Wir müssen deshalb dieses Konzept aufgeben und den Benutzer dazu zwingen, explizit zu spezifizieren, welche der Subtasks zweier Methoden miteinander identisch sind. Dazu wird dem CoMo-Kit-Methodeneditor die aus Standard-Texteditoren bekannte Technik des „Kopierens und Einfügens“ („Cut & Paste“) mitgegeben.

5.5.2.1.2 Umsetzung von Ummodellierungen im alten CoMo-Kit-Modell

Aufgrund der geänderten Semantik müssen wir auch die Reaktionen adaptieren. Da es nur ein Zustandsmodell für Goals und Operatoren gibt, betrachten wir Reaktionen auf Ummodellierungen an Tasks und Methoden auch gemeinsam. Auch hier gehen wir davon aus, daß Reaktionen bezüglich Formaler Parameter von den Subgoals gehandhabt werden, die diese produzieren oder konsumieren. Die sich ergebenden Reaktionen kann man Tabelle 15 entnehmen.

Ummodellierung	IF Condition	THEN Action
Task: Input-added	OTV.State >= „Waiting“	NTV.State = „Delegated“
Task: Output-added	*Atomic only:* OTV.State = „Satisfied“	NTV.State = „InProgress“
Task: Method-added	OTV.State <= „Satisfied“	Notification (NTV.Agent, „New Method Applicable“)
	OTV.State = „Blocked“	NTV.State = „Waiting“
Task: Method-retracted	(OTV.State >= „MethodApplied“) AND (OTV.Decision = RetractedMethod)	NTV.State = „InProgress“
Task: Agent-added	OTV.State = „Initialized“	Notification(NTV.Supertask.Agent, „Agent Added“)
Task: Agent-retracted	(„InProgress“ >= OTV.State >= „Delegated“) AND (OTV.Agent = RetractedAgent)	NTV.State = „Initialized“
	(„InProgress“ < OTV.State < „Satisfied“) AND (OTV.Agent = RetractedAgent)	Notification(NTV.Supertask.Agent, „Responsible Agent Lost“)
Method: Task-added	OTV.State >= „Forwarded“	NTV.State = „MethodApplied“
Method: Task-retracted	OTV.State = „MethodApplied“ AND RetractedTask.State <> „Satisfied“ AND *all other Subtasks „Satisfied“*	NTV.State = „Satisfied“

Tabelle 15: Reaktionen bei Task-/Methoden-Ummodellierungen

Für die Produkte ergeben sich hier keine Änderungen, es gelten die Ausführungen zur Typmodifikation von Seite 86.

Im Unterschied zu Abschnitt 5 sind hier die die Zustände repäsentierenden Strukturen bekannt. Deshalb können wir hier auch genauer darauf eingehen, welche Modifikationen an diesen

Strukturen die erwünschten Reaktionen (Zustandsübergänge) hervorruft[1]. Wir benutzen für das TMS die bereits aus [Dell95] bekannte Repräsentation als Regelmenge.

Task: Input-added

Hier ist es nicht ausreichend, das TMS einfach um den Assigned-Knoten und die Verbindung zum Executable-Knoten des Goals zu erweitern, da der neue Input ja bereits vorhanden (d.h. „assigned“) sein könnte und sich die neue, erweiterte Regel

$$Assigned(I_1) \wedge \ldots \wedge Assigned(I_n) \wedge \boldsymbol{Assigned(I_{neu})} \wedge Valid-Delegation(T) \Rightarrow Executable(T)$$

im Wert nicht ändert und deshalb auch keinen erwünschten Zustandsübergang verursacht. Da wir zwischen Inputs und Outputs jedoch einen kausalen Zusammenhang postulieren, müssen wir auch bei neu hinzugekommene Inputs die Task neu abarbeiten. Um dies zu erzwingen, müssen wir zusätzlich den Knoten „Retracted-Decision“ für diese Task aktivieren. Dies Maßnahme hat zudem den Vorteil, daß nach der erneuten Auswahl der Methode auch alle in den Subtasks dieser Task getroffenen Entscheidungen automatisch wieder gültig gemacht werden.

Task: Output-added

Auch hier ist es nicht damit getan, das entsprechende Output-Assignment einzufügen, denn nach momentaner Realisierung des Zustandsmodells würde sich der Zustand der Task nicht ändern. Deshalb ergänzen wir (dauerhaft!) folgende Regel für atomare Aufgaben:

$$Assigned(O_1) \wedge \ldots \wedge Assigned(O_n) \Rightarrow Satisfied(T)$$

Somit ist eine atomare Aufgabe erst dann abgeschlossen, wenn alle Outputs belegt sind. Fügen wir nun den neuen (noch nicht erstellten) Output ein, so rutscht die Task automatisch in den erwünschten Zustand „InProgress“.

Task: Method-added

Grundsätzlich muß hier die die alternativen Zerlegungen einer Aufgabe repräsentierende Datenstruktur ConflictSet um die neue Alternative erweitert werden, damit bei allen zukünftigen Entscheidungen diese Methode auch aufgelistet wird. Im TMS wird der für alle Methoden obligatorische Known-Bad-Knoten, der zur Feststellung einer Blockade dient, neu eingeführt. Dies führt bei einer blockierten Task zu der in Tabelle 15 beschriebenen erwünschten Zustandsveränderung. Ansonsten wird lediglich eine Notifikation an den Agenten diese Aufgabe gesendet, um ihm die Möglichkeit zu geben, eine u.U. bereits ausgewählte Methode durch die neue zu ersetzten.

Task: Method-retracted

In diesem Fall wird der ConflictSet entsprechend um die weggefallene Alternative verkleinert. War die Methode bereits in der Abarbeitung, so muß der Rejected-Decision-Knoten der gelöschten Methode mit einer nicht-zurückziehbaren Premise-Justification versehen werden, um die erneute Auswahl der Methode zu verhindern.

Task: Agent-added

Erfordert keine dierkten Manipulationen am TMS.

---

1. Wir setzen nunmehr Kenntnisse über die TMS-Strukturen voraus, z.B. aus [Dell95]

Task: Agent-retracted
Diese Ummodellierung wirft die Frage auf, was mit einer bereits begonnenen Task geschehen soll, wenn der bearbeitende Agent plötzlich wegfällt. Da wir keine allgemeine Semantik für diese Ummodellierung finden können, müssen wir während der Ummodellierung erfragen, ob und unter wessen Verantwortung die Task weiter bearbeitet werden soll.

Method: Task-added
Wird eine Task einer bereits in Abarbeitung befindlichen Zerlegung zugefügt, so wird sie exakt so behandelt, als würde die Task neu eingefügt, d.h. es werden neue TMS-Strukturen für ihren Zustand, mögliche Decisions und produzierte/konsumierte Produkte angelegt. Durch das Anbinden an die Supertask wird diese dann in den korrekten Zustand zurückversetzt.

Method: Task-retracted
Wird eine Task gelöscht, so reicht es aus, ihre TMS-Strukturen (einschliesslich derer ihrer Subtasks zu löschen. Natürlich sollte dies erst geschehen, nachdem alle weiterzuverwendenden Subtasks an ihre neuen Supertasks angebunden wurden.

Wir verzichten hier auf eine erneute Darlegung des Algorithmus, da er (natürlich unter Weglassen der Schritte, in denen Bedingungskonstrukte behandelt werden) im Wesentlichen dem in Abschnitt 5.4.5.3 geschilderten gleicht.

5.5.2.2 Realisierungsentscheidungen für ReMo-Kit
Während wir die erarbeiteten Aspekte der Modellierung und Umsetzung von Prozeßmodelländerungen (bis auf die Problematik der Bedingungskonstrukte) voll berücksichtigen können, werden wir bei den Produktmodellmodifikationen begrenzte Möglichkeiten offerieren. Insbesondere ermöglichen wir keine automatischen Typtransformattionen von Instanzen .
Bei der Verwaltung der Ummodellierungstransaktionen werden wir eine vollständige Modellkopie als Modifikationspuffer benutzen. Wir nehmen ausgehend von der aktuellen Implementierung des CoMo-Kit schwache Postconditions an, was in unserem Fall heißt, daß Outputs weiterverwendet werden können, ohne daß die produzierende Task im Zustand „Satisfied“ sein muß.
Als Notifikationskomponente nehmen wir zunächst die Message-Komponente, die bisher Zustandsänderungen im Scheduler an die Clients der Agenten weitergibt.
Grundsätzlich behält die in Abschnitt 5.5.1 und Abb. 14 dargestellt Architektur des CoMo-Kit-Systems ihre Gültigkeit, so daß auch ReMo-Kit in zwei Komponenten zerfällt: ein im Modeler angesiedeltes Ummodellierungs-Transaktionsmanagement und eine im Scheduler befindliche Replanning-Komponente.

Ummodellierungs-Transaktionsmanagement im Modeler
Die wesentliche Modifikation im Modeler ist die Einführung eines Ummodellierungs-Transaktions-Managements, welches die volle Kontrolle über alle Modellmodifikationen erhält. Es sollte nicht mehr möglich sein, Änderungen an Modellen vorzunehmen, ohne daß eine Ummodellierungstransaktion angestossen wird.
Vor allen Modellmodifikationen muß also eine Ummodellierungstransaktion eröffnet werden, deren Sichtbarkeit und Modifikationstyp angegeben werden muß. Im Transaktionsmanager wird eine Kopie des aktuellen Modells angelegt, auf der die Modifikationen vorgenommen werden. Eine Logging-Komponente schreibt die Änderungen mit.

Da es sich um das Konzept einer prototypischen Implementierung handelt, verzichten wir zunächst auf eigentlich wünschenswerte Eigenschaften wie eine feingranularige Rücknahme von Modifikationen. Ein Rücksetzen von Änderungen ist somit nur durch einen Abort der Ummodellierungstransaktion zu bewerkstelligen.

Replanning-Komponente im Scheduler

Neben der in Abschnitt 5.5.2.1.2 (Absatz „Task:Output-Added“) beschriebenen dauerhaften Modifikation der TMS-Strukturen durch Hinzufügen einer neuen Regel wird die wesentliche Funktionalität (das Umsetzen des Abhängigkeitsnetzwerkes in die neue Version) in der Replanningkomponente gekapselt.

Wird eine Ummodellierung beendet, so muß zu Beginn der Umsetzungsphase der Scheduler von der Arbeit suspendiert werden. In dieser COMMIT-Phase der Ummodellierungstransaktion können die Clients der Agenten nicht mit dem Scheduler interagieren, sie müssen ihre Aktionen puffern.

Bei der Implementierung der Umsetzungsalgorithmus selber verzichten wir zunächst auf die Rücksetzbarkeit von Ummodellierungen.

6 Zusammenfassung und Ausblick

Neben einer Zusammenfassung der erstellten Arbeit im Bereich Ummodellierungen deuten wir mögliche Entwicklungslinien für das Projekt CoMo-Kit als Werkzeug zur Modellierung und Verwaltung von prozeßorientierten Problemlöseprozessen an.

6.1 Zusammenfassung

Unsere Zielsetzung war, Ummodellierungsprozesse, die während der Abarbeitung eines Workflow-Modells auftreten, auf eine für den Benutzer transparente, geordnete Art und Weise in der Scheduling-Komponente umzusetzen. Praktische Aspekte, die dabei zu beachten waren, umfaßten, sowohl durch die Ummodellierung überflüssig werdende Arbeiten zu einem möglichst frühen Zeitpunkt zu erfassen und den bearbeitenden Agenten von der Überflüssigkeit seines Tuns in Kenntnis zu setzen als auch einen möglichst hohen Grad der Wiederverwendung bereits erarbeiteter Ergebnisse zu erreichen.
Unsere Literaturarbeit hat in diesem Zusammenhang wenige Erkenntnisse gebracht. Das Problem dynamischer Ummodellierungen in WFMS wurde zwar häufig als „wichtig“ bezeichnet, jedoch konnten wir nur im EPOS-Projekt Ansätze finden, die sich mit dieser Thematik bereits konkreter auseinandersetzten. Zudem trat zutage, daß die *explizite* Verwaltung der Abhängigkeiten zwischen Workflow-Prozessen und -Produkten in der Form, in der sie im CoMo-Kit repräsentiert wird (entscheidungsbasierte Task-Methoden-Dekomposition) sich so in anderen Ansätzen nicht wiederfindet, weswegen auch eine Adaption der Ergebnisse aus anderen Projekten im Bereich WFMS schwierig sein dürfte.
Zur Erreichung obiger Ziele wir zunächst die Anforderungen konkretisiert und uns dann bemüht, Konzepte zu erarbeiten, die diesen Anforderungen gerecht werden. Dies ist mit unterschiedlichem Erfolg gelungen: Während wir bei der Konsistenzüberwachung und der Anforderung der Transparenz von Ummodellierungen (d.h. der Unterrichtung der Benutzer von sie betreffenden Ereignissen im Ummodellierungsmanagement) nur teilautomatisieren konnten, haben wir mit dem Konzept der Wiederverwendung von Arbeit, der Ummodellierungstransaktionen und der Transformation der Abhängigkeitsstrukturen des CoMo-Kit die Anforderungen erfüllen können.
Wir haben feststellen müssen, daß sich die Frage der „Überflüssigkeit“ einer Aufgabe in der Praxis nicht ausschliesslich an den modellierbaren Gegebenheiten der Ummodellierung (z.B. den verwendeten Ummodellierungsoperatoren) festmachen läßt und deshalb eine automatisierte Detektion und Reaktion nicht wünschenswert erscheint. Trotzdem haben wir dem

Umplaner über Notifikation der betroffenen Planer/Bearbeiter die Möglichkeit gegeben, auf die Abarbeitung des Workflow-Modells Einfluß zu nehmen. Mit dem Konzept der Ummodellierungstransaktion haben wir einen Rahmen für die Operationalisierung von Modellierung und Umsetzung der Modellmodifikationen geschaffen.
Schliesslich haben wir für die Umsetzung der Ummodellierung in das die Abarbeitung des Workflow-Modells verwaltende TMS-Netzwerk (die „COMMIT-Phase“ der Ummodellierungstransaktion) einen Algorithmus gebaut, der ausgehend von der Idee, eine Ummodellierung als Einführung einer neuen Methode (inklusive deren sofortiger Anwendung im aktuellen Ablaufplan) zu betrachten, schrittweise auch über die Wiederverwendung bereits erledigter Arbeiten entscheidet. Mit den „Shared Subgoal Spaces“ haben wir ein Konzept entwickelt, daß es mit Hilfe der Invarianten einer Aufgabe erlaubt zu entscheiden, ob in einem anderen Zusammenhang erarbeitete Ergebnisse übernommen werden können oder nicht.
Während damit Ummodellierungen im Prozeßmodell erfolgreich bearbeitet werden konnten, müssen wir unsere Versuche, Ummodellierungen an Produkten in unsere Betrachtungen einzubeziehen *ohne* ein Versionierungskonzept auch auf der Ebene der Konzeptbeschreibungen einzuführen, aufgeben (siehe dazu Abschnitt 6.2).

6.2 Ausblick

Wie bereits des öfteren angedeutet, werden prozeßorientierte Systeme in vielen Fachgebieten unter den verschiedensten Gesichtspunkten erforscht. Neben der Darstellung von Weiterentwicklungen der von uns entwickelten Konzepte lassen wir uns hier von diesen Untersuchungen beim Auffinden von Möglichkeiten zur Weiterentwicklung von CoMo-Kit inspirieren.

Ummodellierung anderer Modellierungseinheiten
Ähnlich dem EPOS-Projekt gibt es neben dem Prozess-Modell und dem Produkt-Modell noch weitere Modelle, die wir bei unseren Betrachtungen vernachlässigt haben. In weiteren Arbeiten sollten auch das Agenten-Modell und das in Ansätzen vorhandene Tool-Modell in die Betrachtung der Ummodellierungsproblematik einbezogen werden.

Einführung einer Versionierungskomponente
Bisher konnte aufgrund der CoMo-Kit-Philosophie, das Produktmodell ähnlich den Petrinetzen mit produzierten/konsumierten Instanzen zu realisieren, nicht die Notwendigkeit eines komplexen Versionsmanagements begründet werden. Im Kontext von Ummodellierungen ändert sich dies. Es hat sich in unseren Untersuchungen (Abschnitt 5.1.2 und Abschnitt 5.4.5.1.4) gezeigt, daß eine für das Ummodellierungsmanagement transparente Unterstützung von Ummodellierungen im Produktmodell durch eine Versionierungskomponente die Problematik der Interaktion zwischen Prozeß- und Produktmodell lindern würde. Folgende Funktionalitäten wären wünschenswert:

- Verwaltung von Produkttypversionen und den zugehörigen Typtransformatoren
- Unterstützung von automatischen Typtransformationen sowohl bei Grundtypen als auch bei Konzeptbeschreibungen
- Anbindung an das Ummodellierungsmanagement des Prozessmodells bei harten Modifikationen (Abschnitt 5.3.1.2)

- Unterstützung verzögerter Typtransformationen

Der letzte Punkt würde die Möglichkeit eröffnen, daß obwohl eine Ummodellierung eigentlich bereits umgesetzt wurde, trotzdem noch die alte Version einer ummodellierten Task weiterbearbeitet werden kann und erst die Outputs dieser Task mit einer entsprechenden Typtransformation an die neuen Gegebenheiten angepaßt werden. Dies kann dann Sinn machen, wenn die Kosten einer Ummodellierung der betreffenden Task sehr hoch sind (z.B. viel verlorene Arbeit) im Vergleich zu den Kosten einer (zur Not nicht automatisierten, von Hand vorgenommenen) Typtransformation der Outputs.
Packt man die Beschreibung von Prozessen und Methoden auch noch in die Konzeptbeschreibungen hinein, so könnte man auch versuchen, die Ummodellierungen im Prozessmodell mit Hilfe der Versionierungskomponente zu verwalten.

Anwendung der „Shared Subgoal Spaces“ bei Methoden

Das Konzept der „Shared Subgoal Spaces“ kann auch bei ausschliesslich in der Modellinstanz lokalisierten Umplanungen, z.B. Rückzug und Neuauswahl einer Methode die Wiederverwendung von Arbeit regeln. Es gelten dieselben Überlegungen, wie wir sie in Abschnitt 5.4.3 für Ummodellierungen angestellt haben.

Unterstützung bei der Konsistenzüberwachung

Im CoMo-Kit wird z. Zt. keine Konsistenzprüfung auf einem Workflow-Modell durchgeführt, und wir haben in Abschnitt 5.4.1 dargelegt, daß so gut wie unmöglich ist, diese global durchzuführen. Neben der Einführung einer kompletten Konsistenzprüfung würde sich eine Erweiterung der Benutzeroberfläche (im Modeler) um eine Komponente, in der dem Benutzer offene referentielle Abhängigkeiten (z.B. ein nicht versorgter Input; ein Produkttyp, der mit keinem Formalen Parameter assoziiert ist oder umgekehrt) angezeigt bekommt, anbieten.

Einführung einer Erfahrungsdatenbank

Wir haben uns in Abschnitt 5.3 gegen die Verwendung realitätsorientierter Beschreibungen ausgesprochen, weil sie uns für den Zweck der Operationalisierung eines Ummodellierungsmanagements unangemessen komplex erschien. Es ist aber natürlich möglich, daß sich bei der Betrachtung von Erfahrungen aus verschiedenen Instanziierungen eines Workflow-Modells verschiedene nützliche Indikatoren z.B. über die Anfälligkeit von Prozessen für Ummodellierungen, die bei der Planung eines (neuen) Projektes sinnvoll sein können. Beschreibungsansätze, mit denen solche Informationen modelliert werden können dürften jedoch im Sinne der in Abschnitt 3.3 angeführten Klassifikation von Beschreibungsansätzen eher realitätsorientiert sein, so daß die im EPOS-Projekt (Abschnitt 3.3 und Abschnitt 4.4.2) vorgeschlagene Entwicklung von PCEs in Richtung „Experience Factory“ auch für die Zielsetzung flexiblen Workflowmanagements im CoMo-Kit als sinnvoll erweisen könnte.
Für den Bereich unserer Arbeit wäre es vielleicht sogar lohnend zu prüfen, z.B. mit fallbasierten Ansätzen bzw. Adaption, inwieweit Ummodellierungen von einem maschinellen Umplaner durchgeführt werden könnten.

Verzeichnis der Abbildungen

Verzeichnis der Tabellen

Literaturverzeichnis

Wo bekannt, ist der Zugriffspfad fuer ein Dokument über WWW *kursiv* angegeben. Über die Dauerhaftigkeit dieser Links kann generell keine Aussage gemacht werden.

[AAE+96] Alonso, G.; Agrawal, D.; El Abbadi, A..; Kamath, M.; Guenthoer, R.; Mohan, C.: „Advanced Transaction Models in Workflow Contexts“, in: Proc. 12th International Conference on Data Engineering, New Orleans, February 1996, available via WWW: *http://www.almaden.ibm.com/cs/exotica/exotica_papers.html*

[ACF96] Avrilionis, D.; Cunin, P.-Y.; Fernström: „OPSIS: A View Mechanism for Software Processes which Supports their Evolution and Reuse“ in: 18th International Conference on Software Engineering, S. 38 - 47, IEEE Comp. Soc. Press, 1996

[AKA+94] Alonso, G.; Kamath, M.; Agrawal, D.; El Abbadi, A.; Günthör, R.; Mohan, C.: „Failure Handling in large Scale Workflow Management Systems“, IBM Research Report, 1994, available via WWW: *http://www.almaden.ibm.com/cs/exotica/exotica_papers.html*

[BDG+94] Biliris, A.; Dar, S.; Gehani, N.; Jagadish, H.V.; Ramamritham, K.: „ASSET - A System for Supporting Extended Transactions“ in: Proc. of the SIGMOD International Conference on Management of Data, May 1994

[BF94] Bandinelli, S.; Fuggetta, A.: „Computational Reflection in Software Process Modeling: the SLANG Approach“ Technical Report No. 13 , ESPRIT-III Project GOODSTEP (6115), May 1994. Appeared in: Proc. of the 15th Int. Conf. on Software Engineering, Baltimore, Md, IEEE Computer Society Press, 1993, available via *http://www.dbis.informatik.uni-frankfurt.de/REPORTS/GOODSTEP/goodstep_library.html*

[BFG93] Bandinelli, S.; Fugetta, A.; Ghezzi, C.: „Software Process Evolution in the Spade Environment“,. Technical Report No. 14 , ESPRIT-III Project GOODSTEP (6115), May 1994. Appeared in: IEEE Transactions on Software Engineering 19(12):1128--1144, 1993, also in [GJ96], available via *http://www.dbis.informatik.uni-frankfurt.de/REPORTS/GOODSTEP/goodstep_library.html*

[BFGG92] Bandinelli, S.; Fugetta, A.; Ghezzi, C.; Grigolli, S.: „Process Enactment in SPADE“ in: Proc. of the 2nd European Workshop on Software Process Technology, Trondheim/Norway, Springer Verlag, 1992

[BFGL94] Bandinelli, S.; Fuggetta, A.; Ghezzi, C.; Lavazza, L.: “ SPADE: An Environment for Software Process Analysis, Design and Enactment“ ,Technical Report No. 20 , ESPRIT-III Project GOODSTEP (6115), May 1994. Appeared in: A. Finkelstein and J. Kramer and B. Nuseibeh (eds.) Software Process Modeling and Technology, Research Studies Press Limited, 1994. available via *http://www.dbis.informatik.uni-frankfurt.de/REPORTS/GOODSTEP/goodstep_library.html*

[BFN94] Bandinelli, S.; Fugetta, A.; Di Nitto, E.: „Policies and Mechanisms to Support Process Evolution in PSEEs“ in: 3rd International Conference on the Software Process, IEEE Comp. Soc. Press 1994

[C+92] Conradi, R. et al.: „Design, Use and Implementation of SPELL, A Language for Software Process Modeling and Evolution“ in: Derniame, J.-C. (ed.), Proc. of the 2nd Eropean Workshop on Software Process Technology, Trondheim, Norway, Springer Verlag, 1992, available via WWW: *http://www.idt.unit.no/~epos/bibliografia.html*

[C+94] Conradi, R. et al.: „EPOS: Object-Oriented and Cooperative Process Modeling“, in: Finkelstein, A.; Kramer, J.; Nuseibeh, B.A. (eds.): „Software Process Modelling and Techniques“, Advanced Software Development Series, Research Studies Press/John Wiley & Sons, 1994, available via WWW: *http://www.idt.unit.no/~epos/bibliografia.html*

[Chri96] Christofidelli, S: „“, Projektarbeit, Universität Kaiserslautern, 1996

[Dell95] Dellen, B.: „Verwaltung von Abhängigkeiten bei der Operationalisierung konzeptueller Modelle“, Diplomarbeit, Universität Kaiserslautern, 1995

[DHL91] Dayal, U.; Hsu, M.; Ladin, R.: „A Transaction Model for Long Running Activities“ in: Proc. of the 17th Int. Conference on Very Large Data Bases, August 1991

[DKM96] Dellen, B.; Kohler, K.; Maurer, F.: „Design rationales and Software Process Models“, Technical Report SFB-Bericht SFB 501-01-95, Universität Kaiserslautern, 1996

[DM96] Dellen, B.; Maurer, F.: „Integrating Planning and Execution in Software Development Processes“, Proc. of the Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE‘ 96), Stanford, 1996

[DMMV96] Dellen, B.; Maurer, F.; Münch, J.; Verlage, M.: „Enriching Software Process Support by Knowledge-Based Techniques“, in: Special issue of Int. Journal of Software Engineering and Knowledge Engineering, 1996. Also available as Technical Report SFB-501-TR-06-96, available via WWW: *http://wwwagse.informatik.uni-kl.de/Personalia/jm/publications.html*

[DMP95] Dellen, B., Maurer, F., Paulokat, J.:„Verwaltung von Abhängigkeiten in kooperativen wissensbasierten Arbeitsabläufen“ in: Richter, M.M., Maurer, F. (Hrsg.): „Expertensysteme 95: Beiträge zur 3. deutschen Expertensystemtagung XPS-95, 1.-3. März 1995, Kaiserslautern“, infix-Verlag, Sankt Augustin, 1995 (in German)

[DMP96] Dellen, B.; Maurer, F.; Pews, G.: „Knowledge-Based Techniques to increase the Flexibility of Workflow Management“, In: Special Issue of the Data & Knowledge Egnieering Journal, 1996

[Doyl79] Doyle, J.: „A Truth Maintenance System“ in: Journal on Artificial Intelligence, Vol. 12, No. 3, 1979

[ELLR90] Elmagarmid, A.; Leu, Y; Litwin, W.; Rusinkiewicz, M.: „A Multidatabase Transaction Model for Interbase“ in: Proc. of the 16th Int. Conference on Very Large Data Bases, August 1990

[Elma92] Elmagarmid, A.K. (ed.): „Database Transaction Models for Advanced Applications“, Morgan Kaufmann Pub., San Mateo, 1992

[EN96] Ellis, C.A.; Nutt, G.J.: „Workflow: The Process Spectrum“ in: Proc. of the NFS Workshop on Workflow and Process Automation in Information Systems, Athens/Georgia, 1996, available via WWW: *http://optimus.cs.uga.edu:5080/activities/NSF-workflow/finaldocs.html*

[EPOS] The EPOS Team: General Overview, only available via WWW: *http://www.idt.unit.no/~epos/OVERVIEW/epos/epos.html*

[GGK+91] Garcia-Molina, H.; Gawlick, D.; Klein, J.; Kleissner, K.; Salem, K.: „Modelling Long-Running Activities as Nested Sagas“ in: Data Engineering, Vol. 14, No. 1, March 1991

[GHS95] Georgakopoulos, D.; Hornick, M.; Sheth, A.: „An Overview of Workflow-Management: From Process Modelling to Workflow Automation Infrastructure“ in: Distributed and Parallel Databases, No. 3, 1995, available via WWW: *http://optimus.cs.uga.edu:5080/publications/pub_WORKFLOW.html*

[GJ96] Garg, P.K.; Jazayeri, M.: „Process-Centered Software Engineering Environments“, IEEE Computer Society Press, Los Alamitos, 1996

[GJM91] Ghezzi, C.; Jazayeri, M.; Mandrioli, D.: „Foundations of Software Engineering“, Prentice Hall, 1991

[GK87] Garcia-Molina, H.; Salem, K.: „Sagas“ in: Proc. of the SIGMOD International Conference on Management of Data, May 1987

[GMMP91] Ghezzi, C.; Mandrioli, D.; Morasca, S.; Pezze, M.: „A unified high-level petri net formalism for time-critical systems“ in: IEEE Transactions on Software Engineering, February 1991

[Gold96] Goldmann, S.: „PROCURA - A Project Management Model for Concurrent Planning and Design“, submitted to the Workshop „Project Coordination“ on WET ICE ’96: „Collaborating on the Internet: The World Wide Web and Beyond“ (Stanford University, 19.-21.6.96) *(http://wwwagr.informatik.uni-kl.de/~maurer/WETICE96.html)*

[GOOD95] Das GOODSTEP-Team: „The Goodstep Project - Final report“, ESPRIT-III Project GOODSTEP (6115), December 1995, available via *http://www.dbis.informatik.uni-frankfurt.de/REPORTS/GOODSTEP/goodstep.html*

[GSVR94] Gaitanides, M.; Scholz, R.; Vrohlings, A.; Raster, M.: „Prozeßmanagement: Konzepte, Umsetzungen und Erfahrungen des Reengineering“, Hanser Verlag, München/Wien, 1994

[Hert89] Hertzberg, J.: „Planen: Einführung in die Planerstellungsmethoden der Künstlichen Intelligenz“, BI Wissenschaftsverlag, Mannheim/Wien/Zürich, 1989

[HJKW96] Heimann, P.; Joeris, G.; Krapp, C.-A.; Westfechtel, B.: „DYNAMITE: Dynamic Task Nets for Software Process Management“ in: 18th International Conference on Software Engineering, S. 331 - 341, IEEE Comp. Soc. Press, 1996

[Hofm95] Hofmann, M.: „Konzeption eines Prozeßinformations- und -managementsystems“, Gabler Verlag, Wiesbaden 1995

[HSS96] Heinl, P.; Schuster, H.; Stein, K.: „Behandlung von Ad-Hoc-Workflows im MOBILE Workflow-Modell“ in: 4. GI-Fachtagung zur Softwaretechnik in Automation und Kommunikation (STAK), München, 1996

[HS96] Heinl, P.; Stein, K.: „Behandlung semantischer Fehler im Workflow Management“ in: Teilnehmerunterlagen zum 7. GI-Workshop „Transaktionskonzepte“, 1996

[Jabl94] Jablonski, S.: „MOBILE: A Modular Workflow Model and Architecture“ in: Proceedings of the 4th International Workshop on Dynamic Modelling and Information Systems, September 1994, Noordwijkerhout, The Netherlands

[Jabl95a] Jablonski, S.: „Workflow-Management-Systeme: Motivation, Modellierung, Architektur“ in: Informatik Spektrum 18 (1995), S.13-24

[Jabl95b] Jablonski, S.: „Workflow-Management-Systeme: Modellierung und Architektur“, Int. Thompson Publ., 1995“

[JC93] Jaccheri, M.; Conradi, R.: „Techniques for Process Model Evolution in EPOS“ in: IEEE Transactions on Software Engineering 19(12): S. 1145-1156, 1993, available via WWW: *http://www.idt.unit.no/~epos/bibliografia.html*

[JWZ88] Jablonski, S.; Wedekind, H.; Zörntlein, G.: „Fehlerbehandlung in flexiblen Fertigungssystemen“ in: Informatik Forschung und Entwicklung 3, 1988

[Kim94] Kim, W. (ed.): „Modern Database Systems: The Object Model, Interoperability and Beyond“, ACM Press, 1994

[KP92] Kaiser, G; Pu, C.: „Dynamic Restructuring of Transactions“ in: [Elma92]

[KR95] Kamath, M.; Ramamritham, K.: „Modeling Correctness and System Issues in Supporting Advanced Database Applications using WFMS“, Technical Report, Univ. of Massachusettes, USA, June 1995, available via WWW: *ftp://ftp.cs.umass.edu/pub/techrept/techreport/1995/UM-CS-1995-050.ps*

[KR96] Kamath, M; Ramamritham, K.: „Bridging the Gap between Transaction Management and Workflow Management“ in: Proc. of the NFS Workshop on Workflow and Process Automation in Information Systems, Athens/Georgia, 1996, available via WWW: *http://optimus.cs.uga.edu:5080/activities/NSF-workflow/finaldocs.html*

[LC93] Liu, C.; Conradi, R.: „Automatic Replanning of Task Networks for Process Model Evolution in EPOS“ in: Sommerville, I.; Paul, M. (eds.): Proc. of the 4th European Software engineering Conference, Garmisch-Partenkirchen, Germany, Springer Verlag, 1993, available via WWW: *http://www.idt.unit.no/~epos/bibliografia.html*

[Liu91] Liu, C.: „Software Process Planning and Execution: Coupling vs. Integration“, in: Rudolf Andersen, Janis A. Bubenko jr., Arne Solvberg (Eds.): CAiSE‘91, the 3rd International Conference on Advanced Information Systems, Trondheim, Norway, 13-15 May 1991. LNCS 498, Springer Verlag, 578 p. Also available as EPOS TR 132.

[MAG+95] Mohan, C.; Alonso, G.; Günthör, R.; Kamath, M.; Reinwald, B.: „An Overview of the Exotica Research Project on WFMS“ in: Proc. of the 6th International Workshop on High Performance Transaction Systems, Asolimar 1995, available via WWW: *http://www.almaden.ibm.com/cs/exotica/exotica_papers.html*

[Maur93] Maurer, F.: „Hypermediabasiertes Knowledge-Engineering für verteilte wissensbasierte Systeme“, Dissertation, Universität Kaiserslautern, 1993

[Maur96a] Maurer, F.: „Modeling the Knowledge Engineering Process“, submitted to the 2nd Knowledge Engineering Forum 1996, University of Karlsruhe, *(http://wwwagr.informatik.uni-kl.de/~keforum/)*

[Maur96b] Maurer, F.: „Project Coordination in Design Processes“, submitted to the Workshop „Project Coordination“ on WET ICE ’96: „Collaborating on the Internet: The World Wide Web and Beyond“ (Stanford University, 19.-21.6.96) *(http://wwwagr.informatik.uni-kl.de/~maurer/WETICE96 .html)*

[MCL+95] Munch, B.P.; Conradi, R.; Larsen, J.-O.; Nguyen, M.N.; Westby, P.H.: „Integrated Product and Process Management in EPOS“ in: Journal of Integrated CAE, Special Issue on Integrated Product and Process Management, 1995, available via WWW: *http://www.idt.unit.no/~epos/bibliografia.html*

[Moha94] Mohan, C.: „Advanced Transaction Models - Survey and Critique“, Tutorial presented at ACM SIGMOD International Conference on Management of Data, Minneapolis, May 1994, available via WWW: *http://www.almaden.ibm.com/cs/exotica/exotica_papers.html*.

[Moss85] Moss, E.: „Nested Transactions: An Approach to Reliable Distributed Computing“, MIT Press, 1985

[MP95] Maurer, F.; Pews, G.: „Ein Knowledge-Engineering-Ansatz für kooperatives Design am Beispiel der Bebauungsplanung“ in: KI 1/95, interdata Verlag, 1995

[MP96] Maurer, F.; Pews, G.: „Supporting Cooperative Work in Urban Land-Use Planning“ in: Proc. COOP‘96, 1996

[NC94] Nguyen, M.N.; Conradi, R.: „Classification of Meta-Processes and their Models“ in: Proc. of the 3rd International Conference on the Software Process, Washington, USA, October 1994

[NWC96] Nguyen, M.N.; Wang, A.I.; Conradi, R.: „Total Software Process Model Evolution in EPOS“, submitted to the 4th Int. Conference on the Software Process, December 1996, Brighton, UK, available via WWW: *http://www.idt.unit.no/~epos/bibliografia.html*

[Ober94] Oberwies, A.: „Verteilte betriebliche Abläufe und komplexe Objektstrukturen: Integriertes Modellierungskonzept für WFMS“, Dissertation, Institut für Angewandte Informatik und Formale Beschreibungsverfahren der Universität Karlsruhe, 1994

[Ober96] Oberwies, A.: „Modellierung und Ausführung von Workflows mit Petri-Netzen“, Teubner Verlagsgesellschaft, Stuttgart/Leipzig, 1996

[Petr91] Petrie, C.: „Planning and Replanning with Reason Maintenance“, PhD Thesis, MCC AI Lab, Austin/Texas, 1991

[RMH+93] Ritter, N.; Mitschang, B.; Härder, T.; Gesmann, M.; Schöning, H.: „Capturing Design Dynamics - The CONCORD Approach“, Proceedings of the 10th Int. IEEE Data Engineering Conference, Houston, 1994, also available as Technical Report of the SFB 124 Nr. 24/93, University of Kaiserslautern, June 1993

[RS94] Rusinkiewicz, M.; Sheth, A.: „Specification and Execution of Transactional Workflows“ in: [Kim94]

[RS95] Reuter, A.; Schwenkreis, F.: „ConTracts - A low-level mechanism for building general-purpose Workflow Management Systems“, in: Hsu, M. (ed.): IEEE Bulletin of the Technical Committee on Data Engineering, Special Issue on Workflow Systems, Vol. 18 No. 1, March 1995, *http://www.informatik.uni-stuttgart.de/ipvr/as/publikationen/reuter-95-1.html*

[Schm94] Schmitz, W.-G.: „Multiple Aufgabenzerlegung von konzeptuellen Modellen“, Diplomarbeit, Universität Kaiserslautern, 1994

[Schw93] Schwenkreis, F.: „APRICOTS - Management of the Control Flow and the Communication System“, in: Proc. of the IEEE Symposium on Reliable Distributed Systems, Princeton, October 1993, *http://www.informatik.uni-stuttgart.de/ipvr/as/publikationen/schwenkreis-93-1.html*

[Schw95] Schwenkreis, F.: „APRICOTS - a workflow programming environment“, in: Proc. of the 6th Workshop on High Performance Transaction Processing Systems (HPTS), 1995, available via WWW: *http://www.informatik.uni-stuttgart.de/ipvr/as/publikationen/schwenkreis-95-1.html*

[TC95] Totland, T.; Conradi, R.: „A Survey and Classification of Some Research Areas Relevant to Software Process Modeling“, EWSPT‘95, Leiden, 3-5 April 1995; available via WWW: *http://www.idt.unit.no/~epos/bibliografia.html*

[VDMM96] Verlage, M.; Dellen, B.; Maurer, F.; Münch, J.: „A Synthesis of Process Support Approaches“, Proc. Int. Conference on Software Engineering, 1996

[WFMC96] Workflow Management Coalition: Glossary, available via WWW: *ftp://ftp.aiai.ed.ac.uk/pub/projects/WfMC/glossary.ps*

[WKM+95] Wodtke, D., Kotz Dittrich, A., Muth, P., Sinnwell, M., Weikum, G.:„Mentor - Entwurf einer Workflow-Management-Umgebung basierend auf State- und Activitycharts“ in: Lausen, G. (Hrsg.): „BTW95 - Datenbanksysteme in Büro, Technik und Wissenschaft“ , Tagungsband zur GI-Fachtagung, Springer Verlag, Heidelberg, 1995 (in German)

[WR92] Wächter, H.; Reuter, A.: „The ConTract Model“ in [Elma92]

[WS92] Weikum, G.; Schek, H.-J.: „Concepts and Applications of Multilevel Transactions and Open Nested Transactions“ in: [Elma92]