Informatik Forsch. Entw. (2001) 16: 125–144

Informatik
Forschung und Entwicklung

Ein graphbasiertes Managementsystem für dynamische Entwicklungsprozesse

Bernhard Westfechtel

Lehrstuhl für Informatik III, RWTH Aachen, D-52056 Aachen

Eingegangen am 9. Juni 2000 / Angenommen am 20. März 2001

Zusammenfassung. Wir präsentieren ein Managementsystem, das insbesondere hinsichtlich der Dynamik von Entwicklungsprozessen über die Funktionalität kommerzieller Systeme zum Projektmanagement, Workflowmanagement und Engineering Data Management hinausgeht. *AHEAD* (*A*daptable and *H*uman-Centered *E*nvironment for the M*A*nagement of *D*evelopment Processes) unterstützt die Koordination von Entwicklern durch integriertes Management von Produkten, Aktivitäten und Ressourcen. Für das Management von Aktivitäten werden dynamische Aufgabennetze angeboten, die sich durch nahtloses Verschränken von Planen, Analysieren und Ausführen auszeichnen. Alle Managementdaten (z.B. Aufgabennetze, Versionshistorien oder Produktkonfigurationen) werden intern in einheitlicher Weise durch Graphen repräsentiert. Managementwerkzeuge werden aus einer formalen Spezifikation generiert, die auf programmierten Graphersetzungsregeln basiert.

Schlüsselwörter: Softwarekonfigurationsmanagement, Workflowmanagement, Softwareprozess

Abstract. We present a management system whose functionality goes beyond commercial systems for project management, workflow management, and engineering data management in particular with respect to the dynamics of development processes. *AHEAD* (*A*daptable and *H*uman-Centered *E*nvironment for the M*A*nagement of *D*evelopment Processes) supports the coordination of engineers through integrated management of products, activities, and resources. For activity management, dynamic task nets are provided which are characterized by seamless interleaving of planning, execution, analysis, and monitoring. All management data (e.g., task nets, version histories, and product configurations) are internally represented by graphs in a uniform manner. Management tools are generated from a formal specification which is based on programmed graph rewriting.

Key words: Software configuration management, Workflow management, Software process

CR Subject Classification: D.2.9, H.4.1, K.6.3

1 Einleitung

Unternehmen stehen unter ständig wachsendem Druck, ihre Geschäftsprozesse zu optimieren. Kosten müssen gesenkt, Zeiten verkürzt, und die Qualität muss verbessert werden. Insbesondere bei repetitiven Prozessen wurden bezüglich dieser Faktoren bereits signifikante Erfolge erzielt. Dies gilt beispielsweise für Fertigungsprozesse, aber auch für Routinetätigkeiten im Bürobereich.

Entwicklungsprozesse zeichnen sich dagegen durch einen vergleichsweise geringen repetitiven Anteil aus. Sie sind hochgradig kreativ und damit schwer planbar, wiederholbar und beherrschbar. Während die Produkte von Fertigungsprozessen beliebig oft repliziert werden können, sind die Produkte von Entwicklungsprozessen Unikate. Insofern ist jeder Entwicklungsprozess einmalig. Auch bei der Entwicklung eines ähnlichen Produkts ändert sich der Prozess durch erfahrungsbasiertes Lernen und Wiederverwendung.

Aus diesem Grund sind Entwicklungsprozesse hochgradig *dynamisch*. Der Ablauf eines Entwicklungsprozesses kann nur beschränkt vorab geplant werden. Welche Schritte auszuführen sind, kann häufig erst während des Ablaufs entschieden werden. Auch die Dauer der einzelnen Schritte lässt sich oft nur schwer vorhersagen. Schließlich können Rückgriffe auftreten, die dazu führen, dass bereits ausgeführte Schritte erneut durchlaufen werden müssen.

Diese Feststellungen gelten gleichermaßen für Entwicklungsprozesse in unterschiedlichen Anwendungsbereichen. In diesem Aufsatz werden wir die *Softwareentwicklung* betrachten (und damit implizit auch immer die Softwarewartung mit einschließen). Außerdem gilt unser Interesse Entwicklungsprozessen im *Maschinenbau* (Verfahrenstechnik und Fertigungstechnik).

Aus den genannten Gründen erweist sich das *Management* von Entwicklungsprozessen als eine schwierige Aufgabe. Dabei verstehen wir unter Management “alle Aktivitäten und Aufgaben, die von einer oder mehreren Personen durchgeführt werden, um die Aktivitäten anderer zu planen und zu kontrollieren, so dass ein Ziel erreicht wird, das diese anderen alleine nicht erreichen könnten” [62]. M.a.W.: Management ist die Koordination von Entwicklern.

Das Management von Entwicklungsprozessen muss Produkte, Aktivitäten und Ressourcen integriert betrachten:

- Als *Produkte* bezeichnen wir alle Ergebnisse von Aktivitäten in den verschiedenen Phasen der Entwicklung. In der Softwareentwicklung schließt dies beispielsweise Anforderungsdefinitionen, Softwarearchitekturen, Modulschnittstellen und -rümpfe, Dokumentationen, Testdaten etc. ein.
- *Aktivitäten* sind die Schritte, aus denen sich ein Entwicklungsprozess zusammensetzt. Beispielsweise werden in der Verfahrenstechnik Anlagen durch Fließbilder beschrieben, chemische Prozesse simuliert, Kostenschätzungen durchgeführt etc.
- Der Begriff *Ressourcen* umfasst die Aktoren bzw. Hilfsmittel, die Aktivitäten ausführen bzw. zur Ausführung benötigt werden. Dies schließt sowohl menschliche Ressourcen (das Entwicklerteam) als auch technische Ressourcen (Werkzeuge und Rechner) ein.

Um das Management von Entwicklungsprozessen zu unterstützen, steht eine breite Palette von Systemen zur Verfügung. Allerdings weisen diese Systeme eine Reihe von Nachteilen auf, die im Folgenden kurz besprochen werden.

Seit langer Zeit werden *Projektmanagementsysteme* [34, 33, 38] eingesetzt, um Entwicklungsprozesse zu planen, zu organisieren und zu überwachen. Zu diesem Zweck werden unterschiedliche Arten von Projektplänen verwendet (z.B. Vorgangsnetzwerke, PERT- oder GANTT-Diagramme). Mit der Verwendung von Projektmanagementsystemen sind folgende Probleme verbunden: zu grobgranulare Projektpläne, keine adäquate Unterstützung der Dynamik von Entwicklungsprozessen (z.B. keine Rückgriffe), fehlende Integration in die Arbeit der Entwickler und keine Berücksichtigung der Produkte des Entwicklungsprozesses.

Zur Verwaltung der Produkte von Entwicklungsprozessen dienen Systeme für Engineering Data Management (EDM [48, 6, 37]), Product Data Management (PDM [21]), Dokumentenmanagement [19] und Software Configuration Management (SCM [63, 10]). Insbesondere lassen sich mit diesen *Produktmanagementsystemen* Versionen von Dokumenten verwalten und zu Konfigurationen zusammensetzen. Das Management von Aktivitäten wird jedoch häufig nur rudimentär unterstützt (z.B. durch Änderungsprozesse, die durch endliche Automaten gesteuert werden).

Workflowmanagementsysteme [65, 26, 44] werden vor allem für repetitive Geschäftsprozesse im Bürobereich eingesetzt (z.B. Sachbearbeitertätigkeiten in Banken und Versicherungen). Eine wesentliche Restriktion von Workflowmanagementsystemen besteht in ihrer mangelnden Flexibilität: Sie unterstützen i.W. vordefinierte Abläufe, deren Ausführung durch eine Workflowmaschine gesteuert wird. Die Workflowmaschine ermittelt die zur Ausführung anstehenden Schritte, bestimmt die verantwortlichen Bearbeiter, bietet Aufgaben in Arbeitslisten an und ruft zur Bearbeitung benötigte Anwendungen auf. Abweichungen von der Workflowdefinition werden nur beschränkt zugelassen, z.B. durch Überspringen oder Wiederholen einzelner Schritte.

In diesem Aufsatz präsentieren wir ein Managementsystem, das mit dem Akronym *AHEAD* [30] bezeichnet wird (*A*daptable and *H*uman-Centered *E*nvironment for the M*A*nagement of *D*evelopment Processes). Es unterscheidet sich in folgenden Punkten von kommerziellen Systemen zum Projekt-, Produkt- und Workflowmanagement:

- AHEAD deckt das Management von Produkten, Aktivitäten und Ressourcen gleichermaßen ab. Dagegen liegt der Schwerpunkt von EDM-, PDM- und SCM-Systemen auf dem Produktmanagement, während der primäre Fokus von Projektmanagement- und Workflowmanagementsystemen den Aktivitäten gilt.
- AHEAD unterstützt das Management dynamischer Entwicklungsprozesse. Insbesondere lassen sich Planen, Analysieren, Ausführen und Überwachen nahtlos miteinander verschränken. Dagegen gibt es bei Workflowmanagementsystemen in der Regel eine scharfe Trennung zwischen Definitions- und Ausführungszeit.
- Sowohl Entwicklern als auch Managern werden interaktive Werkzeuge zur Verfügung gestellt. Insbesondere wird – im Gegensatz zu vielen Workflowmanagementsystemen – das Management nicht automatisiert, d.h. der Manager kann stets in den Entwicklungsprozess eingreifen und ihn ggf. umstrukturieren.
- Intern werden alle Managementdaten in einheitlicher Weise durch *Graphen* dargestellt (an der Benutzeroberfläche werden unterschiedliche Repräsentationen benutzt, z.B. Diagramme, Bäume oder Tabellen). Die Funktionalität der vom AHEAD-System angebotenen Werkzeuge ist formal mit Hilfe von *Graphersetzungsregeln* auf einer hohen Abstraktionsebene spezifiziert. Kommerzielle Systeme basieren in der Regel nicht auf formalen Spezifikationen.
- Graphbasierte Spezifikationen erleichtern nicht nur das präzise Verständnis der Funktionalität von Werkzeugen. Darüber hinaus werden die Werkzeuge aus der Spezifikation generiert. Dies reduziert den Aufwand zur Erstellung und Modifikation der Werkzeuge erheblich.
- AHEAD lässt sich an unterschiedliche Anwendungsbereiche anpassen. Domänenexperten bedienen sich der *Unified Modeling Language* (UML [5]), einer weitverbreiteten Standardnotation für objektorientierte Modellierung. Im Gegensatz dazu bieten z.B. Workflowmanagementsysteme proprietäre Sprachen für die Definition von Workflows an, die Domänenexperten – z.B. Verfahrenstechniker – erst erlernen müssen.

Die wissenschaftlichen Beiträge des AHEAD-Systems liegen auf zwei Ebenen. Zum einen basiert es auf einem innovativen Modell, das aus drei miteinander integrierten Teilmodellen für das Management von Produkten, Aktivitäten und Ressourcen besteht und insbesondere die Dynamik von Entwicklungsprozessen unterstützt. Somit bietet AHEAD seinen Benutzern neuartige Funktionalität (externe Ebene). Zum anderen demonstriert es den Nutzen von Graphersetzungssystemen für die formale Spezifikation und Generierung von Werkzeugen (interne Ebene).

Das AHEAD-System wird z.Zt. im Rahmen des SFB 476 IMPROVE auf verfahrenstechnische Entwicklungsprozesse angewendet. Es ist der Nachfolger eines Managementsystems, das im SUKITS-Projekt entstanden ist, in dem Entwicklungsprozesse in der Fertigungstechnik untersucht wurden. AHEAD und sein Vorgänger sind im Rahmen einer Habilitation entstanden [68]; SUKITS und IMPROVE sind beide in [52] beschrieben.

Abschnitt 2 vermittelt einen Überblick über das AHEAD-System. In den Abschnitten 3 und 4 wird das zugrunde lie-

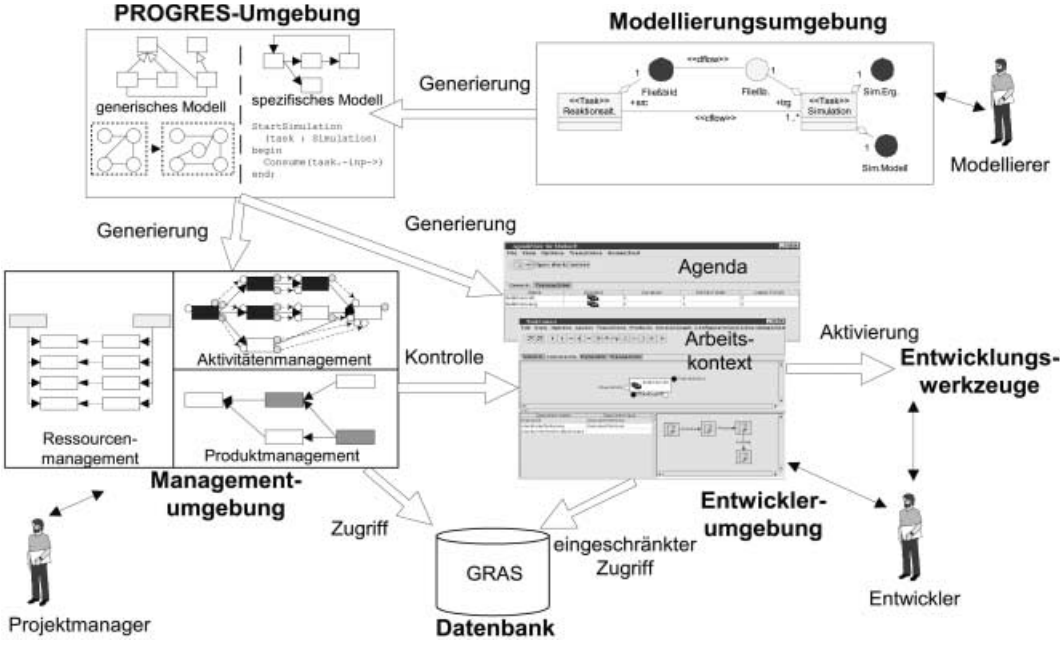


Abb. 1. Überblick über das AHEAD-System

gende Managementmodell informell eingeführt bzw. formal beschrieben. Abschnitt 5 geht auf die Anpassung des Managementmodells in der UML ein. Abschnitt 6 ist den Werkzeugen gewidmet, die das AHEAD-System anbietet. In Abschnitt 7 werden Anwendungen im Maschinenbau besprochen. Abschnitt 8 diskutiert verwandte Ansätze. Schließlich folgen in Abschnitt 9 eine Zusammenfassung und ein Ausblick auf zukünftige Arbeiten.

2 Überblick über das AHEAD-System

Abbildung 1 vermittelt einen Überblick über die Komponenten des AHEAD-Systems.

Die *Managementumgebung* (unten links) stellt graphische Werkzeuge zur Planung, Steuerung, Analyse und Überwachung von Entwicklungsprozessen zur Verfügung. Ein Manager definiert das Projektteam, baut Aufgabennetze auf, weist den Aufgaben Entwickler zu, überwacht die Ausführung von Aufgaben und modifiziert das Aufgabennetz (z.B. im Falle von Rückgriffen). Ferner nutzt er graphische Werkzeuge zum Produktmanagement, z.B. um Versionshistorien zu betrachten oder Konfigurationen zu bearbeiten.

Die *Entwicklerumgebung* (unten rechts) zeigt einem Entwickler eine *Agenda* an, in der die ihm zugewiesenen Aufgaben aufgelistet sind. Über die Agenda kann der Entwickler Aufgaben starten, unterbrechen, fortsetzen und beenden. Wählt er eine Aufgabe zur Bearbeitung aus, so wird ein *Arbeitskontext* angezeigt, der die zu bearbeitenden Dokumente enthält. Auf diesen Dokumenten lassen sich dann entsprechende Entwicklungswerkzeuge aktivieren (z.B. CASE-Werkzeuge, Editoren, Compiler und Debugger für die Softwareentwicklung).

Alle Managementdaten (Versionshistorien, Produktkonfigurationen, Aufgabennetze etc.) werden in einer graphbasierten Datenbank abgespeichert. Dazu wird das Datenbanksystem *GRAS* [39] verwendet, das auf die Anforderungen zugeschnitten ist, die aus dem Bau strukturbezogener interaktiver Werkzeuge resultieren.

Managementumgebung und Entwicklerumgebung unterstützen ihre Benutzer zur *Laufzeit* eines Projekts. Im Folgenden gehen wir darauf ein, auf welche Weise die den Managern und Entwicklern zur Verfügung gestellten Werkzeuge entstehen und wie die Modelle der zu unterstützenden Entwicklungsprozesse definiert werden (*Definitionszeit*).

Managementwerkzeuge werden aus einer formalen Spezifikation generiert, die auf programmierten Graphersetzungsregeln basiert. Zu diesem Zweck wird die *PROGRES-Umgebung* [60, 61] verwendet (oben links). PROGRES bezeichnet zum einen eine Spezifikationssprache und zum anderen eine Umgebung, die integrierte Werkzeuge zum syntaxgestützten Edieren, Analysieren und Interpretieren anbietet. Aus einer mit der PROGRES-Umgebung erstellten Spezifikation wird mit Hilfe eines Compilers effizienter Code erzeugt, der in die Managementumgebung und die Entwicklerumgebung eingebunden wird.

Die von einem konkreten Anwendungsbereich unabhängigen Konzepte für das Management von Produkten, Aktivitäten und Ressourcen werden in einem *generischen Modell* spezifiziert. Diese Spezifikation wird nur einmal erstellt und dann in unterschiedlichen Domänen wiederverwendet. Um das Managementsystem an einen bestimmten Anwendungsbereich anzupassen, wird ein auf dem generischen Modell basierendes *spezifisches Modell* definiert. Die Gesamtspezifikation setzt sich somit aus einem generischen und einem spezifischen Anteil zusammen.

PROGRES ist eine Spezifikationssprache, die sich in erster Linie an Werkzeugbauer richtet. Man kann i.A. nicht erwarten, dass Domänenexperten (z.B. aus dem Maschinenbau) diese Sprache beherrschen. Deshalb wird Domänenexperten eine auf der UML basierende *Modellierungsumgebung* zur Verfügung gestellt (oben rechts). In der Modellierungsumgebung erstellte UML-Modelle werden automatisch in den spezifischen Anteil der PROGRES-Spezifikation

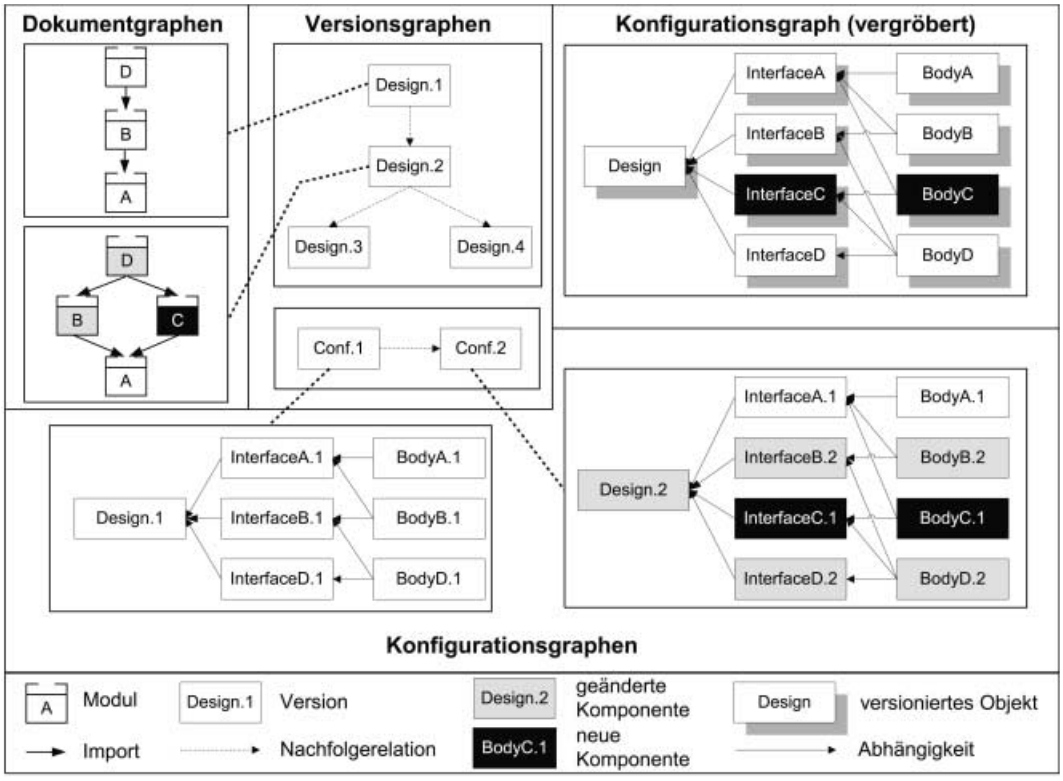


Abb. 2. Graphen für die Versions- und Konfigurationsverwaltung

transformiert. Somit wird die für den Werkzeugbau verwendete Spezifikationssprache vor den Domänenexperten verborgen.

3 Managementmodell

Das Managementmodell [68, 42] setzt sich aus drei Teilmodellen (für Produkte, Aktivitäten und Ressourcen) zusammen, die im Folgenden auf informeller Ebene beschrieben werden. Danach gehen wir auf die Integration der Teilmodelle ein. Zur Illustration wird ein durchgängiges Beispiel aus dem Bereich der Softwareentwicklung verwendet.

Bei der Beschreibung konzentrieren wir uns auf die Eigenschaften der generischen Modelle (obwohl wir konkrete Beispiele verwenden, die entsprechende spezifische Modelle voraussetzen). Wie ein spezifisches Modell auf der Basis eines generischen Modells definiert wird, ist in Abschnitt 5 erläutert.

3.1 Produktmodell

Das Produktmodell *CoMa* (*Co*nfiguration *Ma*nagement [67]) dient zur Verwaltung der Dokumente, in denen die Ergebnisse von Entwicklungsaktivitäten festgehalten werden. Dokumente können mit Hilfe beliebiger Entwicklungswerkzeuge erstellt werden (Heterogenität). Dabei werden Dokumente nicht isoliert voneinander betrachtet, sondern es werden die Abhängigkeiten zwischen den Dokumenten berücksichtigt. Dadurch leistet das CoMa-Modell einen Beitrag zur dokumentübergreifenden Konsistenzkontrolle. Aus Dokumenten und ihren Abhängigkeiten werden Konfigurationen gebildet. Sowohl Dokumente als auch Konfigurationen unterstehen der Versionskontrolle [10]. Der Begriff "Version" umfasst sowohl Revisionen (zeitliche Entwicklung) als auch Varianten (Alternativen).

Für das Produktmanagement werden unterschiedliche Arten von Graphen miteinander gekoppelt (Abb. 2). Ein *Versionsgraph* (oben in der Mitte) repräsentiert die Versionen eines Objekts (Dokument oder Konfiguration) und die zwischen ihnen bestehenden Beziehungen. Eine Beziehung von v_1 nach v_2 drückt aus, dass v_2 aus v_1 entstanden ist (Nachfolgerelationen). Nachfolgerelationen dürfen keinen Zyklus bilden. Es sind sowohl Verzweigungen (Varianten) als auch Zusammenführungen erlaubt.

Der Inhalt einer Dokumentversion wird durch einen *Dokumentgraphen* (oben links) dargestellt. In unserem Beispiel ist dies eine Softwarearchitektur, die sich aus Modulen und zwischen ihnen bestehenden Importbeziehungen zusammensetzt. Das CoMa-Modell schreibt nicht vor, wie Dokumentinhalte zu modellieren sind (grobgranularer Ansatz). Ferner müssen auf der Realisierungsebene keine Graphen für die Darstellung von Dokumentinhalten verwendet werden, sondern es können die Datenrepräsentationen heterogener Entwicklungswerkzeuge verwendet werden.

Analog wird der Inhalt einer Konfigurationsversion durch einen *Konfigurationsgraphen* (unten) repräsentiert. Im Gegensatz zu den Dokumentgraphen ist die Struktur von Konfigurationsgraphen im CoMa-Modell festgelegt (lässt sich aber domänenspezifisch anpassen). Ein Konfigurationsgraph besteht aus Versionen von Dokumenten (oder geschachtelten Konfigurationen) und zwischen ihnen bestehenden Abhängigkeiten (z.B. ist der Rumpf eines Moduls von der Schnittstelle abhängig). Die Abhängigkeiten legen fest, wel-

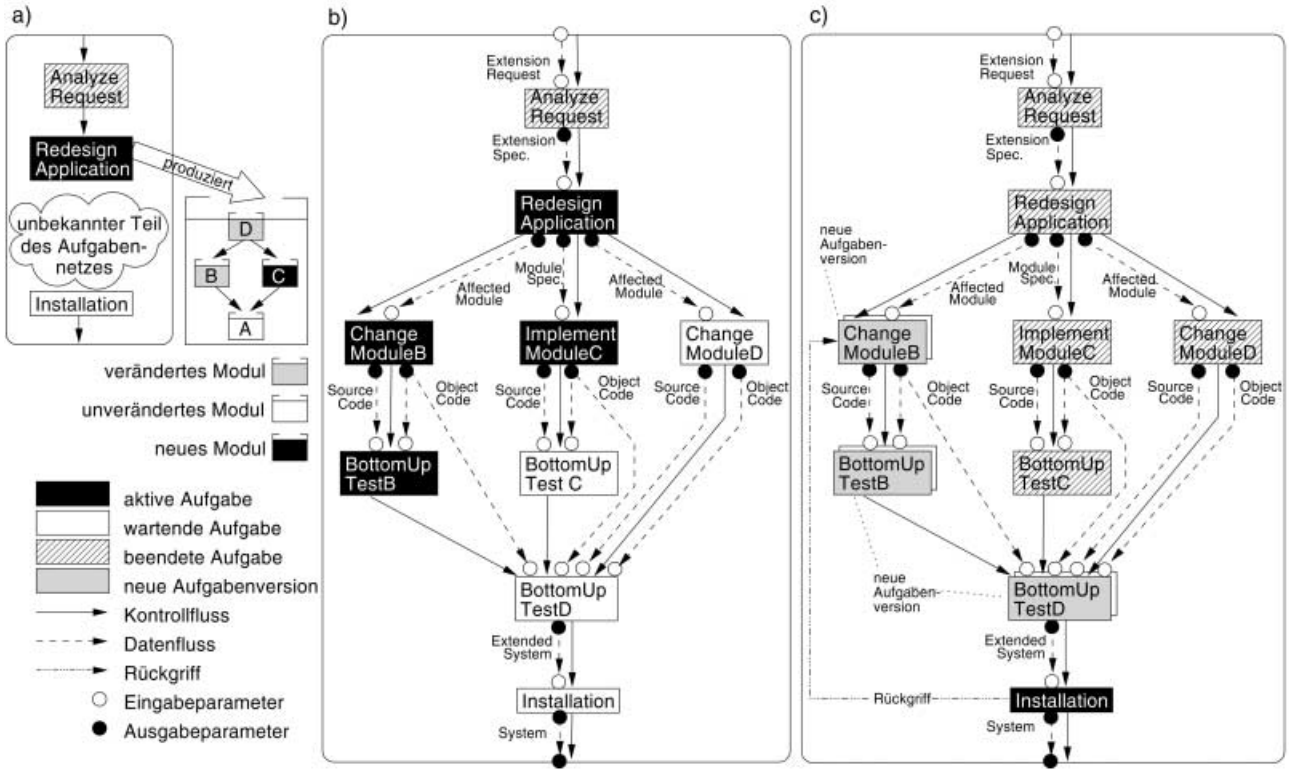


Abb. 3. Dynamische Aufgabennetze

che Dokumentversionen miteinander konsistent zu halten sind.

Im Beispiel besteht die Konfiguration Conf.1 aus einer Softwarearchitektur, welche die Module A, B und D enthält, sowie entsprechenden Modulschnittstellen und -rümpfen. Beim Übergang zu Conf.2 wird das Modul C in die Architektur eingefügt, die Module B und D werden geändert.

Zusätzlich wird ein *vergröberter Konfigurationsgraph* (oben rechts) verwaltet, der eine abstrahierende Sicht auf eine Menge von Konfigurationsversionen anbietet und sowohl invariante Teile (InterfaceA) als auch variable Komponenten (InterfaceC) enthält. Mit Hilfe dieser Vergröberung lassen sich Konfigurationsfamilien darstellen (näheres dazu in [67, 68]).

3.2 Aktivitätenmodell

Das Aktivitätenmodell *DYNAMITE* (*DYNAMI*c *T*ask N*E*ts [23, 41]) definiert *dynamische Aufgabennetze*, die sich durch nahtloses Verschränken von Planen, Ausführen, Analysieren und Überwachen auszeichnen. Ein Aufgabennetz setzt sich aus Aufgaben zusammen, die durch unterschiedliche Arten von Beziehungen verbunden sind. Kontrollflüsse regeln die Reihenfolge der Bearbeitung von Aufgaben in ähnlicher Weise wie in einem Netzplan. Mit Hilfe von Rückgriffsbeziehungen, die in entgegengesetzter Richtung zu den Kontrollflüssen verlaufen, lassen sich Rücksprünge im Entwicklungsprozess ausdrücken. Aufgaben haben Ein- und Ausgaben, die durch Datenflüsse verbunden sind. Eine Aufgabe ist entweder atomar (Blattaufgabe), oder sie ist komplex und wird durch ein Teilnetz verfeinert (Aufgabenhierarchie). Schließlich hat jede Aufgabe einen Lebenszyklus, der durch ein Zustandsübergangsdiagramm beschrieben wird.

Abbildung 3 zeigt als Beispiel einen Prozess zur Erweiterung der Funktionalität eines Softwaresystems. Der Prozess wird durch eine Änderungsanforderung ausgelöst. Zu Beginn sind nur einige Fragmente des Gesamtprozesses bekannt. Abbildung 3a zeigt den Zustand des Prozesses, nachdem die eingehende Änderungsanforderung analysiert wurde und die Überarbeitung der Architektur begonnen hat. Die Aufgaben Analyze Request und Redesign Application sind beendet bzw. aktiv. Der weitere Teil des Aufgabennetzes ist, abgesehen von der abschließenden Installationsaufgabe, noch unbekannt.

Nachdem die Architektur modifiziert wurde, kann das Aufgabennetz vervollständigt werden (Abb. 3b). Da die Module B und D geändert und Modul C neu implementiert werden müssen, werden entsprechende Änderungs- und Implementierungsaufgaben in das Aufgabennetz eingefügt (*Netzerweiterung*). Danach folgen Testaufgaben, die in einer Bottom-up-Reihenfolge angeordnet sind. Aufgaben, die durch Kontrollflüsse verbunden sind, können zeitlich überlappend bearbeitet werden. Beispielsweise können Entwurf, Implementierung und Test parallelisiert werden (*Simultaneous Engineering* [15]).

Abbildung 3c zeigt das Aufgabennetz zu einem späteren Zeitpunkt. Bei der abschließenden Installationsaufgabe wird ein Fehler in der Implementierung von Modul B festgestellt. Dadurch wird ein *Rückgriff* ausgelöst, der eine Reaktivierung der Aufgabe ChangeModuleB erforderlich macht. Im Interesse der Nachvollziehbarkeit wird eine neue Aufgabenversion erzeugt; dies gilt auch für die nachgelagerten Aufga-

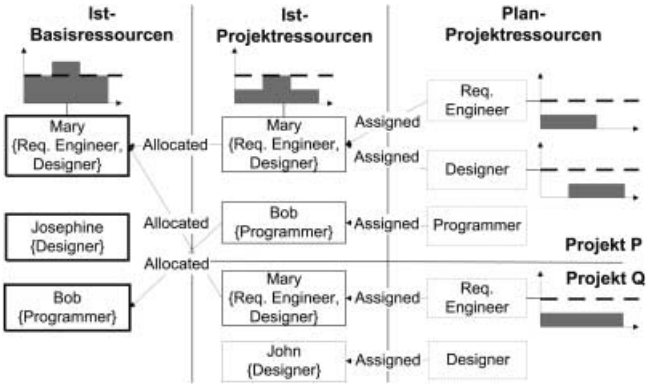


Abb. 4. Projektübergreifendes Ressourcenmanagement

ben. Die Aufgabenversionen repräsentieren somit die Konsequenzen des Rückgriffs im Aufgabennetz.

3.3 Ressourcenmodell

Im Ressourcenmodell *RESMOD* (*RES*ource *MOD*el [43, 42]) werden menschliche und technische Ressourcen in einheitlicher Weise behandelt. RESMOD definiert allgemeine Basiskonzepte, die sich auf beliebige Arten von Ressourcen anwenden lassen. Ressourcen lassen sich hierarchisch zu Ressourcenkonfigurationen gruppieren; ferner können Abhängigkeiten zwischen Ressourcen dargestellt werden. Um die Planung zu unterstützen, wird zwischen *Plan-Ressourcen* und *Ist-Ressourcen* unterschieden. Mit Hilfe von (abstrakten) Plan-Ressourcen wird der Ressourcenbedarf geplant. Zu einem späteren Zeitpunkt werden den Plan-Ressourcen geeignete Ist-Ressourcen zugeordnet. Schließlich unterstützt RESMOD auch das Multiprojektmanagement. Zu diesem Zweck werden *Basis-* und *Projektressourcen* eingeführt. Basisressourcen repräsentieren die Ressourcen eines Unternehmens, die dauerhaft verfügbar sind und bei Bedarf zeitlich befristeten Projekten zugeordnet werden.

Ein Beispiel zur Unterscheidung von Plan- und Ist-Ressourcen bzw. von Basis- und Projektressourcen ist in Abb. 4 angegeben. Auf der linken Seite sind drei zur *Basisorganisation* gehörende Entwickler dargestellt (Mary, Josephine und Bob). Die Rollen, die sie in Entwicklungsprojekten ausfüllen können, sind jeweils in geschweiften Klammern angegeben (z.B. kann Mary im Requirements Engineering und im Entwurf arbeiten). Auf der rechten Seite sind die *Plan-Projektressourcen* angegeben, mit denen ein Projektmanager seinen Ressourcenbedarf planen kann. Z.B. werden in Projekt P drei Stellen für Requirements Engineering, Entwurf und Implementierung benötigt. In der Mitte werden nun mit Hilfe von *Ist-Projektressourcen* die Plan-Projektressourcen und die Ist-Basisressourcen zueinander in Beziehung gesetzt. Dabei kann eine Person (Mary) in mehreren Projekten und innerhalb eines Projekts auf mehreren Stellen eingesetzt werden. Ferner kann ein Entwickler (John) auch für ein spezifisches Projekt eingestellt werden, ohne der Basisorganisation anzugehören.

Für die Ressourcenplanung können erwartete *Auslastungen* angegeben werden. Auf der rechten Seite von Abb. 4 zeigen die Auslastungskurven, dass Requirements Engineering und Entwurf zeitlich versetzt bearbeitet werden. In der Mitte werden die Kurven zur projektspezifischen Auslastung addiert, und auf der linken Seite erhält man schließlich die globale Auslastung. Im Falle von Mary ergibt sich eine Überlast durch den gleichzeitigen Einsatz in mehreren Projekten. Diese muss durch geeignete Reorganisationsmaßnahmen beseitigt werden.

3.4 Integration der Teilmodelle

Bei der Integration der Teilmodelle wird besonderer Wert auf eine lose Kopplung gelegt. Die Schnittstellen zwischen den Teilmodellen sind so gestaltet, dass ein Teilmodell mit möglichst geringen Auswirkungen auf andere Teilmodelle geändert bzw. ersetzt werden kann. Beispielsweise werden im Aktivitätenmodell die Ein- und Ausgaben von Aufgaben durch Marken (Token) repräsentiert, die auf beliebige Objekte verweisen können.

Die Integration zwischen Teilmodellen wird jeweils in entsprechenden *Integrationsmodellen* beschrieben:

- *Aktivitätenmodell* ↔ *Produktmodell*: Datenflüsse zwischen Aufgaben werden mit Hilfe von Marken modelliert, die auf Versionen von Objekten verweisen. Jeder Aufgabe ist ein logischer Workspace zugeordnet, der außer den importierten und exportierten Objektversionen (Eingaben bzw. Ausgaben) auch lokale Versionen enthält. Die Workspaces überlappen sich physisch, d.h. beim Produzieren von Ausgaben bzw. Konsumieren von Eingaben werden keine Kopien erzeugt (man beachte, dass freigegebene Versionen unveränderlich sind).
- *Aktivitätenmodell* ↔ *Ressourcenmodell*: Den Aufgaben werden menschliche Ressourcen (Entwickler) und technische Ressourcen (Werkzeuge) zugeordnet. Dabei geschieht die Zuordnung nicht auf direktem Weg (Aufgaben ↔ Ist-Ressourcen), sondern indirekt über Plan-Ressourcen. Dadurch kann man z.B. eine Aufgabe einer Stelle bereits zuordnen, bevor sie besetzt ist.
- *Produktmodell* ↔ *Ressourcenmodell*: Diese Teilmodelle sind z.Zt. nur rudimentär gekoppelt. Erzeugt ein Entwickler eine Version, so wird automatisch eine entsprechende Verantwortlichkeitsbeziehung geschaffen.

4 Formale Spezifikation

Das AHEAD-System zeichnet sich dadurch aus, dass die von ihm angebotenen Werkzeuge nicht von Hand implementiert, sondern aus einer formalen Spezifikation generiert sind. Zu diesem Zweck werden *Graphen* und *Graphersetzungsregeln* eingesetzt. Managementdaten wie z.B. Versionshistorien, Produktkonfigurationen oder Aufgabennetze werden in einheitlicher Weise als Graphen modelliert. Die von der Manager- und der Entwicklerumgebung angebotenen Operationen werden mit Hilfe von Graphersetzungsregeln auf einer hohen Abstraktionsebene formal spezifiziert.

Als Spezifikationssprache wird *PROGRES* [60, 61] benutzt. In einem Graphschema wird die Struktur von Graphen einer bestimmten Klasse statisch beschrieben. Ein Graphschema definiert Knotenklassen, die in einer Vererbungshierarchie angeordnet sind. Den Knoten können sowohl ei-

genständige als auch abgeleitete Attribute zugeordnet werden. Analog wird zwischen eigenständigen und abgeleiteten Relationen unterschieden (Kanten bzw. Pfade). Graphveränderungen werden durch Graphersetzungsregeln beschrieben. Dabei bietet PROGRES mächtige Hilfsmittel zur Definition komplexer Graphmuster an (z.B. optionale und Mengenknoten, Attributbedingungen und negative Anwendbarkeitsbedingungen). Ferner lassen sich Graphersetzungsregeln mit Hilfe von Kontrollstrukturen zu komplexen Graphtransformationen zusammensetzen.

Zur Illustration des Spezifikationsansatzes werden im Folgenden Ausschnitte aus der Spezifikation des Aktivitätenmodells beschrieben; auf das Produkt- und Resourcenmodell gehen wir aus Platzgründen nicht ein (s. dazu [67, 43, 42]). Wir beschränken uns auf die Spezifikation des generischen Modells; auf spezifische Modelle gehen wir in Abschnitt 5 ein.

Es sei explizit darauf hingewiesen, dass die Spezifikation auf der Ebene der *internen Modellierung* liegt. Es wird also die interne Repräsentation der Managementdaten beschrieben, die von den Werkzeugen des AHEAD-Systems benutzt wird. An der Benutzeroberfläche können diese Daten auf unterschiedliche Arten repräsentiert werden (z.B. als Diagramme, Bäume oder Tabellen; s. Abschnitt 7).

4.1 Graphen und Graphschemata

Intern werden *attributierte Graphen* zur Repräsentation von Managementdaten benutzt. Abbildung 5 zeigt als Beispiel einen Ausschnitt aus der internen Repräsentation des Aufgabennetzes aus Abb. 3b. Knoten sind getypt; die Knotentypen werden in der Abbildung durch unterschiedliche Symbole dargestellt. Kanten sind gerichtet und getypt; der Kantentyp wird durch einen Bezeichner ausgedrückt. Aufgaben und ihre Ein- und Ausgabeparameter werden durch Knoten repräsentiert, die durch `has`-Kanten verbunden sind. Attribute werden verwendet, um Knoten weitere Detailinformationen zuzuordnen (z.B. Name und Zustand einer Aufgabe). Kanten können jedoch nicht attributiert werden; ferner unterstützt das PROGRES zugrunde liegende Datenmodell keine Beziehungen zwischen Kanten. Aus diesen Gründen werden Beziehungen wie Kontrollflüsse, Datenflüsse und Rückgriffsbeziehungen durch Knoten mit adjazenten Kanten dargestellt (z.B. `fromSourceT`- und `toTargetT`-Kanten im Falle von Kontrollflüssen).

Die Modellierung von Datenflüssen bedarf noch einer weitgehenderen Erläuterung. Datenflüsse verfeinern Kontrollflüsse bzw. Rückgriffe: Dieser Sachverhalt wird durch `refines`-Kanten ausgedrückt. Ein Datenfluss verbindet einen Ausgabeparameter einer Vorgängeraufgabe mit einem Eingabeparameter einer Nachfolgeraufgabe. Daten werden durch Marken repräsentiert, die ihrerseits auf Objekte (hier: Dokumentversionen) verweisen. Erzeugt die Vorgängeraufgabe eine neue Ausgabe, so wird eine neue Marke erzeugt, die durch eine `produced`-Kante mit dem Ausgabeparameter verbunden wird. Die Marke kann selektiv für Nachfolgeraufgaben freigegeben werden (`releasedFor`-Kanten). Eine berechtigte Nachfolgeraufgabe kann die Marke lesen (`read`-Kante), d.h. die entsprechende Dokumentversion wird in den logischen Workspace

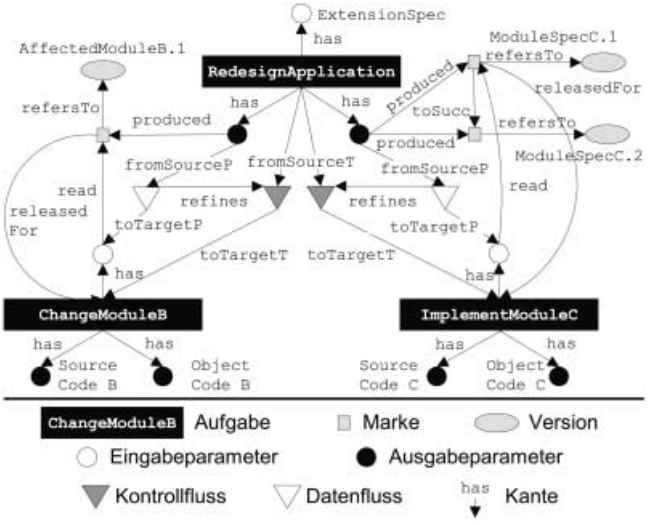


Abb. 5. Interne Repräsentation eines Aufgabennetzes

eingefügt. Produziert die Vorgängeraufgabe danach eine neue Marke (z.B. im Falle eines Rückgriffs), so wird diese an die bisherige Markenliste angehängt (`toSucc`-Kante), um das Produzieren und Lesen nachvollziehbar zu machen. Beispielsweise hat die Aufgabe `RedesignApplication` zwei Versionen der Schnittstelle von Modul `C` produziert, von denen die erste für `ImplementModuleC` freigegeben und konsumiert wurde[1].

Während Abb. 5 einen Graphen auf der Instanzebene zeigt, wird in Abb. 6 die entsprechende Definition auf der Typebene dargestellt. Im *Graphschema* werden Knotenklassen in einer Vererbungshierarchie angeordnet. Eine Unterklasse erbt die Attribute (in Abb. 6 nicht dargestellt) und Kantentypen ihrer Oberklassen. Ein Kantentyp definiert die Klassen von Quell- und Zielknoten sowie die Kardinalität (ebenfalls nicht dargestellt). Graphinstanzen müssen mit dem Schema konsistent sein. Um dies sicherzustellen, werden in PROGRES spezifizierte Graphersetzungsregeln auf ihre Konsistenz mit dem Schema überprüft.

`ITEM` dient als Wurzel der Knotenklassenhierarchie. Mit Hilfe des Kantentyps `toSucc` lässt sich die Entwicklungsgeschichte von Aufgabennetzen modellieren (z.B. um Aufgabenversionen – s. Abb. 3c – und Markenlisten – s. Abb. 5 – darzustellen). Auf der nächsten Ebene der Klassenhierarchie wird zwischen Entitäten und Relationen unterschieden. Entitäten werden in Aufgaben und Parameter klassifiziert, die ihrerseits in Ein- und Ausgabeparameter unterteilt werden. Relationen verbinden entweder Aufgaben (`TASK_RELATION`) vertikal (`DECOMPOSITION`) oder horizontal (`DEPENDENCY`-Relationen, die in Kontrollflüsse und Rückgriffe unterteilt sind), oder sie stellen Beziehungen zwischen Parametern her (`DATA_FLOW`). Die über Datenflüsse transportierten Marken werden durch Knoten der Klasse `TOKEN` repräsentiert. Knoten der Klasse `TASK_GRAPH` dienen dazu, alle Bestandteile von Aufgabennetzen zu verankern. Schließlich sind auf der rechten Seite der Abbildung die Verbindungen zu den Produkten und Ressourcen angedeutet.

[1] Diese Details wurden in Abb. 3 nicht dargestellt; dort wurde von den Dokumentversionen abstrahiert.

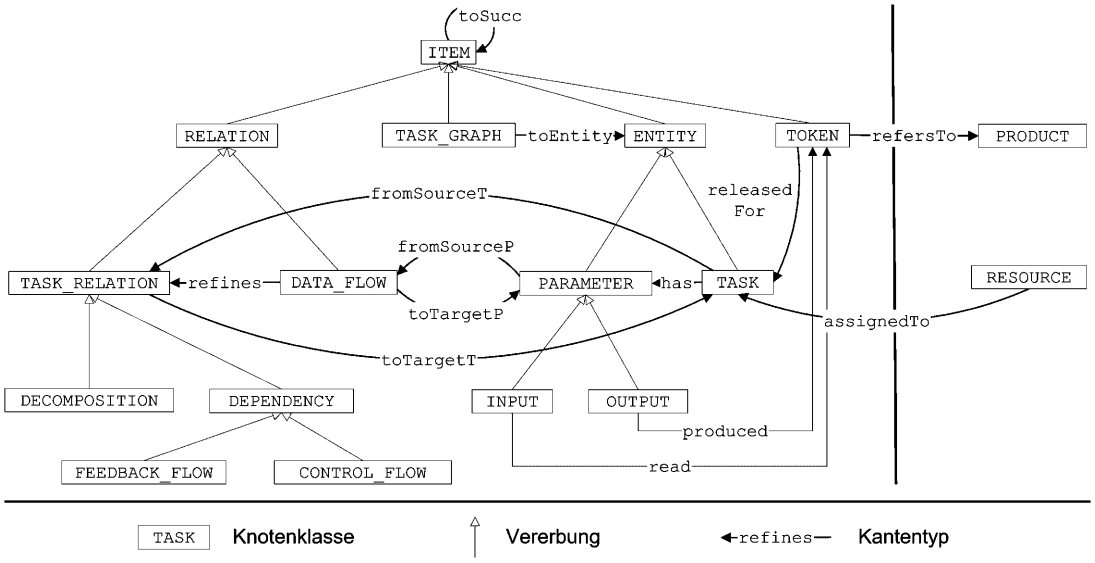


Abb. 6. Graphschema

4.2 Graphersetzungsregeln

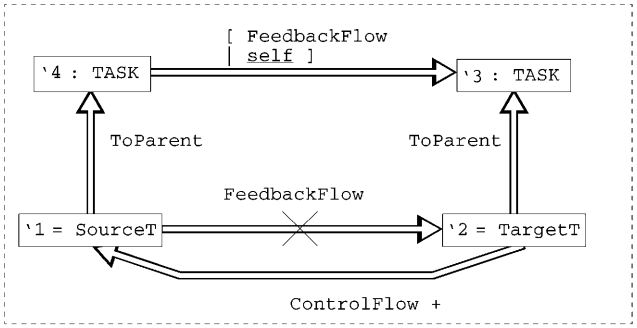
Operationen auf Graphen werden mit Hilfe von *Graphersetzungsregeln* spezifiziert. Eine Graphersetzungsregel beschreibt die Ersetzung eines Graphmusters auf einer hohen Abstraktionsebene. Sie wird in Elementaroperationen zur Mustersuche und zum Erzeugen/Löschen von Knoten/Kanten bzw. zum Setzen von Attributwerten übersetzt. In PROGRES besteht eine Graphersetzungsregel aus folgenden Bestandteilen:

- dem Namen der Regel,
- Parametern, die z.B. dazu benutzt werden können, Knoten der linken Seite zu fixieren,
- der linken Seite, die das zu ersetzende Graphmuster beschreibt,
- Bedingungen an Attributwerte von Knoten der linken Seite,
- der rechten Seite, die das einzusetzende Graphmuster festlegt,
- Regeln zur Berechnung von Attributwerten der Knoten der rechten Seite und
- dem Rückgabeteil zur Berechnung der Ausgabeparameter.

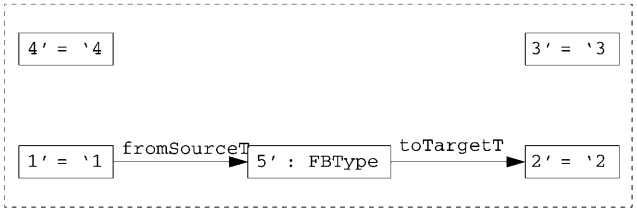
Im Folgenden diskutieren wir zwei Beispiele für Graphersetzungsregeln. Die erste Regel führt eine strukturelle Änderung eines Aufgabennetzes aus, die zweite wird zur Ausführung einer Aufgabe benötigt. Edieren und Ausführen von Aufgabennetzen werden in einheitlicher Weise durch Graphersetzungsregeln spezifiziert. Im Gegensatz zu anderen Ansätzen (insbesondere Workflowmanagementsysteme, s. Abschnitt 1) lässt sich zwischen Edieren und Ausführen keine scharfe Grenze ziehen.

Abbildung 7 zeigt eine Graphersetzungsregel – in PROGRES auch als *Produktion* bezeichnet – zum Erzeugen eines Rückgriffs. Die Quelle (der Auslöser) und das Ziel des Rückgriffs sowie der Typ der zu erzeugenden Beziehung werden als Parameter übergeben. PROGRES hat ein geschichtetes Typsystem: Knoten sind Instanzen von Knotentypen, die ihrerseits Instanzen von Knotenklassen sind. Der Knotentyp `FBType` muss Instanz (`type in`) der Klasse `FEEDBACK` sein. Bei erfolgreicher Regelanwendung wird der erzeugte Rückgriffsknoten (`NewFeedback`) als Ausgabeparameter (`out`) zurückgeliefert.

```
production CreateFeedbackFlow
            ( SourceT : TASK ; TargetT : TASK ;
              FBType : type in FEEDBACK ;
              out NewFeedback : FEEDBACK)
                =
```




```
::=
```




```
   folding { '3, '4 };
   condition [...];
   transfer 5'.active := true;
   return NewFeedback := 5';
end;
```

Abb. 7. Graphersetzungsregel zum Erzeugen eines Rückgriffs

Durch `SourceT` und `TargetT` werden die Knoten `'1` und `'2` der linken Seite bereits eindeutig festgelegt. Die übrigen Elemente der linken Seite dienen dazu, Anwendbar-

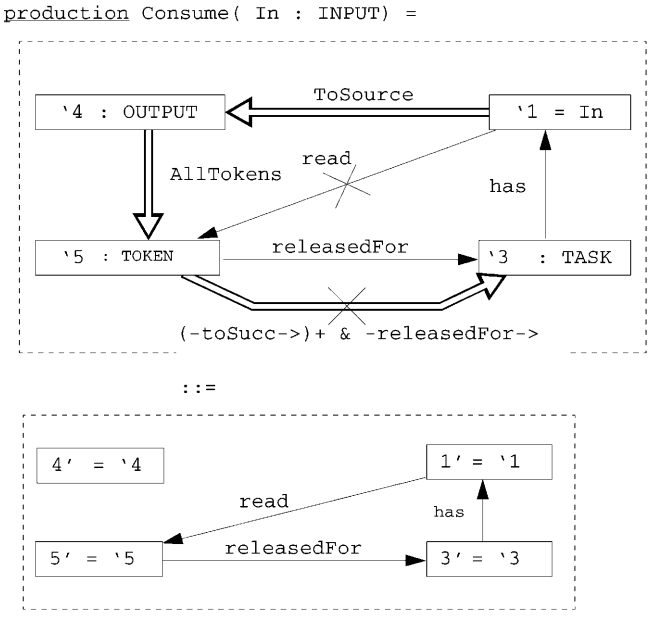


Abb. 8. Graphersetzungsregel zum Konsumieren einer Eingabe

keitsbedingungen zu überprüfen (s.u.). Sind diese Bedingungen erfüllt, so wird die linke Seite durch die rechte ersetzt. Alle Knoten der linken Seite werden identisch ersetzt (z.B. `1' = '1`); der Knoten `5'` sowie seine adjazenten Kanten werden neu erzeugt.

Im `transfer`-Teil wird der Rückgriff durch Setzen eines Booleschen Attributs als aktiv (d.h. zu bearbeitend) gekennzeichnet. Der `return`-Teil dient dazu, den Knoten für den Rückgriff als Ausgabeparameter zu übergeben.

Auf der linken Seite werden Pfade (abgeleitete Relationen) benutzt, um Anwendbarkeitsbedingungen zu spezifizieren. Pfade werden durch Doppelpfeile dargestellt. Der von `'2` nach `'1` verlaufende Pfad stellt sicher, dass das Ziel des Rückgriffs bzgl. der Kontrollflussbeziehungen (`ControlFlow`) ein transitiver Vorgänger (+) der Quelle ist (d.h. Rückgriffe verlaufen entgegengesetzt zu den Kontrollflüssen). Der durchkreuzte Pfad von `'1` nach `'2` repräsentiert eine negative Anwendbarkeitsbedingung und schließt mehrfache Rückgriffe aus.

Schließlich führt der Pfad `toParent` jeweils zur übergeordneten Aufgabe. Falls Knoten `'3` und `'4` – die jeweils zur Klasse `TASK` gehören müssen – zusammenfallen (die `folding`-Klausel erlaubt dies), handelt es sich um einen Rückgriff innerhalb eines Teilnetzes. Andernfalls müssen die übergeordneten Aufgaben bereits durch einen Rückgriff verbunden sein (Balancierung von Rückgriffsbeziehungen). Die Notation `[p|q]` (Pfad von `'4` nach `'3`) steht für einen alternativen Pfad, d.h. die Knoten müssen durch `p` oder `q` verbunden sein.

Die Graphersetzungsregel aus Abb. 8 dient zum Konsumieren einer Eingabe. Sie wird auf einen Knoten angewendet, der einen Eingabeparameter repräsentiert und über einen `ToSource`-Pfad mit einem entsprechenden Ausgabeparameter verbunden ist. Der Pfad `AllTokens` führt zu allen Marken, die bisher produziert worden sind. Aus dieser Menge wird die jüngste Marke ausgewählt, die für die Aufgabe freigegeben ist, welcher der Eingabeparameter zugeordnet ist. Die negative `read`-Kante schließt aus, dass die Marke bereits konsumiert wurde. Die Kante `releasedFor` stellt sicher, dass die Freigabe erfolgt ist. Der negative Pfad von `'3` nach `'5` schließt die Existenz einer jüngeren freigegebenen Marke aus. Die Notation `p & q` steht dabei für eine Sequenz von Pfaden (bzw. Kanten). Ferner wird im Bedingungsteil überprüft, ob die konsumierende Aufgabe sich im Zustand `Active` befindet. Sind diese Bedingungen erfüllt, so wird eine `read`-Kante vom Eingabeparameter zur Marke gezogen. Dadurch wird das Objekt, auf das die Marke verweist, im Workspace der konsumierenden Aufgabe sichtbar.

5 Modellierung in der UML

Im letzten Abschnitt wurde die formale Spezifikation des dem AHEAD-System zugrunde liegenden Managementmodells beschrieben. Bei unserer Erläuterung haben wir uns auf die Ebene des generischen Modells beschränkt. Ein spezifisches Modell lässt sich auf der Basis des generischen Modells definieren, indem man das Graphschema erweitert und die Graphtransformationen anpasst (z.B. durch Komposition von Basisoperationen). Dies setzt jedoch detaillierte Kenntnisse der Spezifikationssprache PROGRES voraus.

Um Domänenexperten (z.B. Verfahrenstechnikern) den Zugang zu erleichtern, wird für die Definition spezifischer Modelle die *Unified Modeling Language* (UML [5]) benutzt. Folgende Gründe sprechen für die Verwendung der UML:

- Die UML ist eine weitverbreitete, standardisierte Modellierungssprache.
- Die Dynamik von Entwicklungsprozessen lässt sich mit Hilfe eines objektorientierten Ansatzes adäquat modellieren.
- Die UML stellt eine reichhaltige Menge von Diagrammen zur Verfügung, um sowohl die Struktur als auch das Verhalten von Entwicklungsprozessen zu beschreiben.

Um die UML für die Prozessmodellierung einzusetzen, haben wir zunächst eine geeignete Teilmenge von Diagrammarten ausgewählt [31]. Zur Beschreibung der Struktur werden Klassendiagramme, zur Beschreibung des Verhaltens Zustands- und Kollaborationsdiagramme verwendet. Diese Diagramme können jedoch nicht in beliebiger Weise benutzt werden, sondern es muss der Bezug zum generischen Modell hergestellt werden. Beispielsweise muss dafür gesorgt werden, dass in einem Klassendiagramm für das Aktivitätenmodell Klassen von Aufgaben sowie Ein- und Ausgabeparametern definiert werden und durch Kontroll- oder Datenflussassoziationen verbunden werden. Dies wird mit Hilfe von *Stereotypen* [4] erreicht. Durch Stereotype lassen sich UML-Modellelemente als Instanzen von Metaklassen auszeichnen.

Stereotypisierten Diagrammen wird eine formale Semantik zugeordnet, indem sie auf programmierte Graphersetzungsregeln abgebildet werden [58]. Dadurch wird sowohl die statische als auch die dynamische Semantik der Diagramme formal definiert. Diese Definition ist auf unser Modell für das Management von Entwicklungsprozessen zugeschnitten und erhebt nicht den Anspruch der Allgemeingültigkeit (man beachte, dass der UML-Standard nur

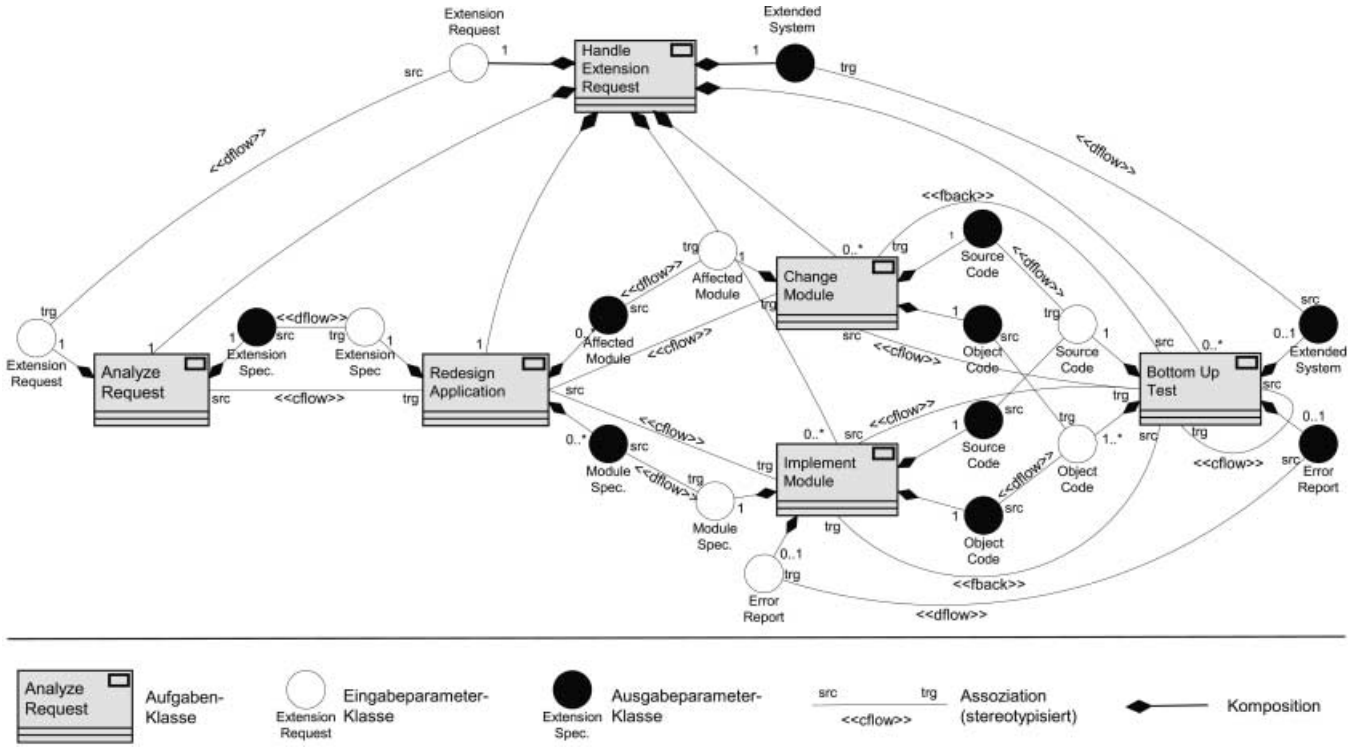


Abb. 9. Klassendiagramm

die statische Semantik (mit OCL) und nicht die dynamische Semantik definiert).

In den folgenden Abschnitten über die Struktur und das Verhalten gehen wir jeweils auf die Modellierung in der UML und die Transformation nach PROGRES ein. Dabei beschränken wir uns wiederum auf das Aktivitätenmodell. Abschließend diskutieren wir kurz die Konsequenzen der Anpassung für die Benutzer der Management- und der Entwicklerumgebung.

5.1 Modellierung der Struktur

Zur Modellierung der Struktur werden *Klassendiagramme* eingesetzt. Das Klassendiagramm von Abb. 9 beschreibt den Beispielprozess aus Abb. 3 auf der Typebene. Klassen für Aufgaben, Ein- und Ausgabeparameter werden durch unterschiedliche graphische Symbole dargestellt. Die Metaklasse einer Assoziation wird durch eine in spitze Klammern eingeschlossene Zeichenkette repräsentiert (z.B. ≪cflow≫ für Kontrollflüsse). Durch schwarze Rauten werden Kompositions-Assoziationen gekennzeichnet, auf deren explizite Stereotypisierung in der Abbildung verzichtet wurde. Kompositionen werden z.B. verwendet, um eine Klasse für eine komplexe Aufgabe mit den Klassen für ihre Unteraufgaben zu verbinden.

Im Klassendiagramm für den Beispielprozess wird festgelegt, dass zunächst genau eine AnalyzeRequest-Aufgabe auszuführen ist, an welche die Eingabe der komplexen Aufgabe HandleExtensionRequest weitergeleitet wird. Danach folgt genau eine RedesignApplication-Aufgabe, an die sich jeweils beliebig viele ChangeModule- und ImplementModule-Aufgaben anschließen. Schließlich folgen beliebig viele Testaufgaben, deren Reihenfolge durch entsprechende Kontrollflüsse festgelegt wird[2]. Die potenziellen Rückgriffe im Prozess werden durch ≪fback≫-Assoziationen repräsentiert, von denen in der Abbildung nur einige exemplarisch dargestellt sind.

Um die Verbindung zum Graphersetzungssystem des letzten Abschnitts herzustellen, wird das Klassendiagramm durch ein automatisch arbeitendes Transformationswerkzeug in eine Erweiterung des PROGRES-Graphschemas übersetzt. Die im generischen Modell definierten Graphersetzungsregeln greifen auf das Graphschema zu und überprüfen damit die domänenspezifischen Konsistenzbedingungen. Die Graphersetzungsregeln selbst brauchen nicht verändert zu werden. Für detailliertere Informationen sei der Leser auf [24, 58] verwiesen.

5.2 Modellierung des Verhaltens

Das Verhalten wird mit Hilfe von *Zustandsdiagrammen* festgelegt. Jede Aufgabe hat einen Zustand, der die jeweils ausführbaren Operationen einschränkt. Im generischen Modell wird ein Zustandsdiagramm vordefiniert, das domänenspezifisch angepasst werden kann. Das im generischen Modell definierte Standardverhalten legt fest, welche Kombinationen von Zuständen benachbarter Aufgaben erlaubt sind, in welchem Zustand ein Aufgabennetz strukturell verändert werden kann, unter welchen Bedingungen Eingaben konsumiert bzw. Ausgaben produziert werden können etc.

Das *generische Zustandsdiagramm* aus Abb. 10a gilt gleichermaßen für komplexe und atomare Aufgaben. Der Le-

[2] Man beachte, dass das Klassendiagramm nicht ausdrucksstark genug ist, um die Teststrategie (bottom-up) zu beschreiben.

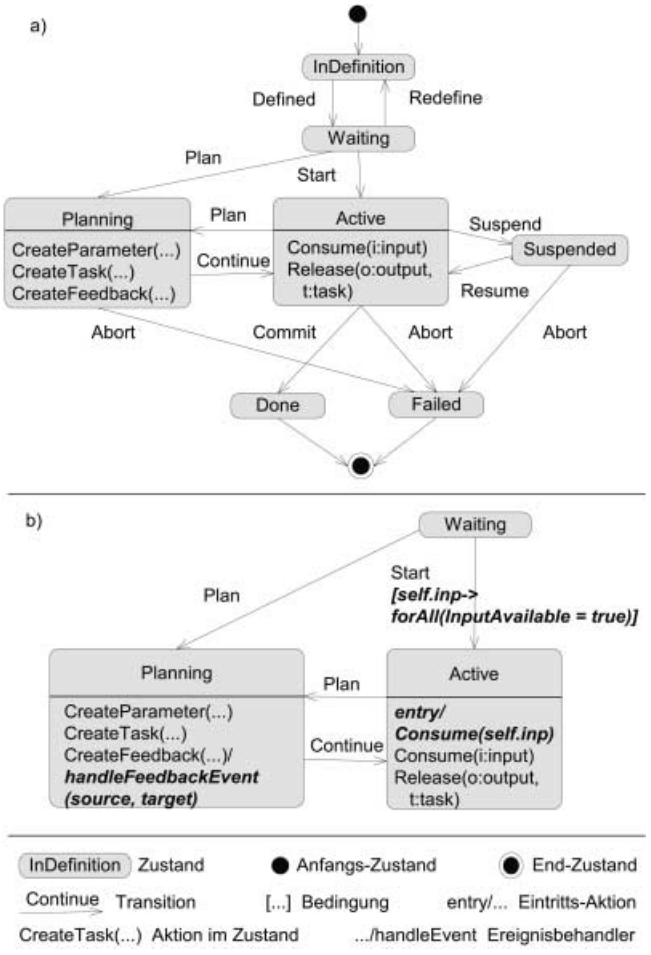


Abb. 10. Generisches und spezifisches Zustandsdiagramm

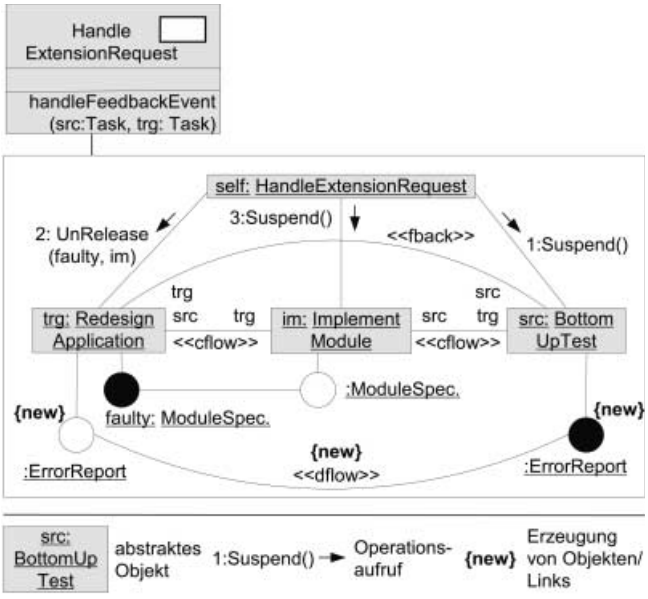


Abb. 11. Kollaborationsdiagramm

benszyklus einer Aufgabe beginnt im Zustand InDefinition, in dem zunächst die Aufgabenstellung sowie die Ein- und Ausgaben festgelegt werden[3]. Im Zustand Waiting wartet die Aufgabe auf ihre Aktivierung (Übergang nach Active). Während eine Aufgabe aktiv ist, können Eingaben konsumiert bzw. Ausgaben produziert werden. Durch den Übergang nach Suspended wird die Bearbeitung unterbrochen. Im Zustand Planning kann die Schnittstelle einer Aufgabe (Ein- und Ausgabeparameter) verändert werden. Ferner kann das Subnetz einer komplexen Aufgabe ediert werden (initiale Planung bzw. spätere Umplanung). Schließlich wird eine Aufgabe entweder erfolgreich beendet (Zustand Done) oder abgebrochen (Zustand Failed).

Das generische Zustandsdiagramm lässt sich nur in eingeschränkter Weise anpassen. Es beschreibt in allgemeiner Weise den Ablauf bei der Bearbeitung einer Aufgabe. Ein *spezifisches Zustandsdiagramm* hat stets die gleiche Topologie wie das generische Zustandsdiagramm. Das generische Zustandsdiagramm wird angepasst, indem die Bedingungen für Zustandsübergänge, die Reaktionen auf die Ausführung von Operationen und die Aktionen beim Betreten und Verlassen von Zuständen redefiniert werden. Nach unserer Erfahrung vereinfacht diese Art der Anpassung die Modellierung des Verhaltens, ohne die Flexibilität unangemessen einzuschränken.

In Abb. 10b wird ein Ausschnitt aus dem Zustandsdiagramm für Aufgaben der Klasse HandleExtensionRequest gezeigt. Beim Start-Übergang wird nun überprüft, ob alle Eingaben verfügbar sind (das Standardverhalten schreibt dies nicht vor). Die entsprechende Bedingung ist in der OCL (Object Constraint Language der UML) formuliert. Beim Eintritt (entry) in den Zustand Active werden die Eingaben automatisch konsumiert. Schließlich wird im Zustand Planning beim Aufruf der Operation CreateFeedback der Ereignisbehandler handleFeedbackEvent angestoßen.

Um den Rumpf eines Ereignisbehandlers zu spezifizieren, wird eine weitere Diagrammart der UML benutzt: das *Kollaborationsdiagramm*. Ein Kollaborationsdiagramm besteht aus einer Menge von (abstrakten) Objekten und Links. Durch den Links zugeordnete beschriftete Pfeile werden Aufrufe von Operationen dargestellt. Ferner lassen sich strukturelle Änderungen spezifizieren (z.B. Erzeugen von Objekten und Links mit {new}).

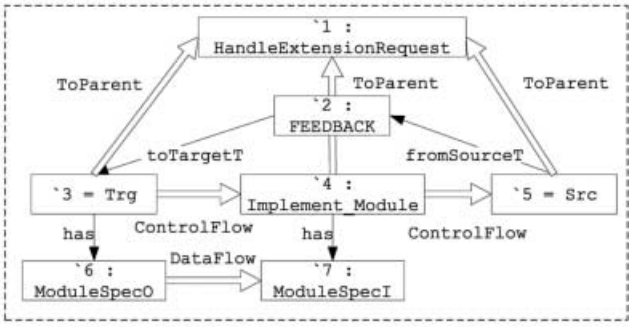
Das Kollaborationsdiagramm aus Abb. 11 beschreibt die Reaktion auf einen Rückgriff von einer Aufgabe der Klasse BottomUpTest auf eine Aufgabe der Klasse RedesignApplication. Zur Übermittlung eines entsprechenden Fehlerreports werden ein Ausgabeparameter, ein Eingabeparameter und ein Datenfluss erzeugt. Ferner werden sowohl die Testaufgabe als auch die vorgelagerte Implementierungsaufgabe suspendiert. Die Freigabe der fehlerbehafteten Modulschnittstelle wird zurückgenommen. Im weiteren Verlauf des Entwicklungsprozesses wird der Fehler behoben und eine neue Schnittstellenversion freigegeben. Danach können Implementierung und Test fortgesetzt werden.

Auch die UML-Diagramme zur Modellierung des Verhaltens werden automatisch in PROGRES übersetzt. Abb. 12 zeigt dies exemplarisch für das Kollaborationsdiagramm aus Abb. 11. Durch einen *Graphtest* wird zunächst die Suche nach dem Objektmuster realisiert[4]. Anschließend werden die im Kollaborationsdiagramm spezifizierten Operationen auf-

[3] Diese Festlegungen können ggf. später noch verändert werden.

[4] Ein Test lässt sich als eine identische Graphersetzung auffassen.

```
test ThandleCreateFeedbackEvent
      ( Src : Bottom_Up_Test;
        Trg : Redesign_Application;
        out Strd : Standard;
        out Impl_Module: Implement_Module;
        out FB : FEEDBACK;
        out Mspec1 : ModuleSpecO;
        out Mspec2 : ModuleSpecI; )
=
```




```
  return Strd := `1; Impl_Module := `4;
         FB := `2 ; Mspec1 :=`6; Mspec2 := `7;
end;

transaction handleFeedbackEvent(Src, Trg : TASK) =
  (* Deklaration lokaler Variablen *)
    ThandleCreateFeedbackEvent
      (Src : Bottom_Up_Test,
       Trg : Redesign_Application, ...)
  & CreateParameter(Src, Error_ReportO, out ER1)
  & CreateParameter(Trg, Error_ReportI, out ER2)
  & CreateDataflow(ER1, ER2, FB)
  & Suspend(Src)
  & UnRelease(Mspec1, Impl_Module)
  & Suspend(Impl_Module)
end;
```

Abb. 12. Übersetzung nach PROGRES

gerufen. Die Aufrufe erfolgen in einer *Transaktion*. Der Operator & steht für einen sequenziellen Aufruf (der in prozeduralen Programmiersprachen üblicherweise durch ein Semikolon dargestellt wird).

Man beachte, dass die im Kollaborationsdiagramm beschriebenen strukturellen Änderungen nicht in einer spezifischen Graphersetzungsregel zusammengefasst werden können. Statt dessen werden einzelne Operationsaufrufe erzeugt. Auf diese Weise wird garantiert, dass die Konsistenzbedingungen des generischen Modells eingehalten werden. Eine domänenspezifische Graphersetzungsregel, die an den Basisoperationen des generischen Modells vorbei auf den Aufgabengraphen angewendet wird, könnte diese Bedingungen verletzen.

5.3 Domänenspezifisches Managementsystem

Für die Benutzer der Management- bzw. der Entwicklerumgebung äußert sich die Definition eines Klassendiagramms wie in Abb. 9 darin, dass auf der Instanzebene domänenspezifisch getypte Aufgabennetze bearbeitet werden, deren Typkonsistenz überwacht und gewährleistet wird. Baut etwa ein Manager ein Aufgabennetz wie in Abb. 3 auf, so stehen dafür die in Abb. 9 definierten Klassen und Assoziationen zur Verfügung. Mit dem domänenspezifisch angepassten AHEAD-System lassen sich dann nur noch Aufgabennetze aufbauen, die mit dem Klassendiagramm konsistent sind. Insofern dient das strukturelle Modell der *Kontrolle* und sorgt für die Einhaltung eines definierten Prozesses.

Mit Hilfe des Verhaltensmodells lassen sich domänenspezifische Operationen definieren, die die *Unterstützung* der Benutzer verbessern sollen. Beispiele dafür wurden in den Abb. 10 und 11 gegeben. In beiden Fällen werden Basisoperationen des generischen Modells zu komplexen Operationen zusammengefasst. Der Benutzer braucht also die Sequenz der Basisoperationen nicht mehr von Hand zu aktivieren. So werden in Abb. 10 beim Starten einer Aufgabe automatisch alle Eingaben konsumiert, und in Abb. 11 werden beim Erzeugen eines Rückgriffs automatisch weitere Operationen (zum Suspendieren von Aufgaben und Übertragen eines Fehlerreports) ausgeführt.

6 Realisierung

In diesem Abschnitt gehen wir kurz auf die Realisierung des AHEAD-Systems ein, dessen Aufbau in Abb. 1 dargestellt wurde.

Zur Erstellung domänenspezifischer Prozessmodelle wird eine kommerzielle CASE-Umgebung (*Rational Rose*) verwendet. Dies geschieht aus zwei Gründen: Zum einen erscheint der Aufwand für eine Eigenimplementierung nicht gerechtfertigt; zum anderen können Domänenexperten eine weit verbreitete, ihnen ggf. ohnehin vertraute Modellierungsumgebung benutzen. Die UML-Diagramme werden an die Prozessmodellierung angepasst, indem Stereotype definiert und diesen ggf. Ikonen für die graphische Darstellung zugeordnet werden. Die vom Modellierer erstellten Diagramme werden von einem Compiler auf syntaktische und semantische Korrektheit geprüft und in PROGRES-Code übersetzt. Als Ausgabe erzeugt der Compiler eine Textdatei mit PROGRES-Quelltext. Der Compiler ist in Visual C++ geschrieben und umfasst ca. 17.000 Zeilen Quelltext[5].

Die *PROGRES-Umgebung* [60, 61] ist seit einigen Jahren als Public-Domain-Software verfügbar[6]. Sie bietet integrierte, strukturbezogene Werkzeuge für das Erstellen, Analysieren und Interpretieren von PROGRES-Spezifikationen an. Edieren, Analysieren und Interpretieren lassen sich nahtlos miteinander verschränken. Die der PROGRES-Umgebung zugrunde liegende Basistechnologie wurde im Rahmen des IPSEN-Projekts [49] entwickelt.

PROGRES wurde in einer Reihe von Forschungsprojekten auf unterschiedliche Probleme der Softwaretechnik angewendet. Im Rahmen der Entwicklung des AHEAD-Systems wurden Spezifikationen der generischen Modelle zum Management von Produkten, Aktivitäten und Ressourcen erstellt. Der Gesamtumfang dieser Spezifikationen beträgt etwa 200 Seiten PROGRES-Quelltext. Die spezifischen Modellanteile werden, wie bereits oben erläutert, automatisch erzeugt. Die Gesamtspezifikation wird vom PROGRES-Compiler in C-Code übersetzt, der in die Management- und in die Entwicklerumgebung eingebunden wird.

Der generierte Code operiert auf einer graphbasierten Datenbank, die vom Datenbanksystem *GRAS* [39] verwaltet wird. GRAS stellt (wie PROGRES) eine Eigenentwicklung dar, die bereits Anfang der 80er Jahre initiiert wurde. GRAS verwaltet attributierte Graphen, deren Struktur durch ein

[5] Bisher wurde der Compiler nur für das Aktivitätenmodell realisiert.
[6] `http://www-i3.informatik.rwth-aachen.de`

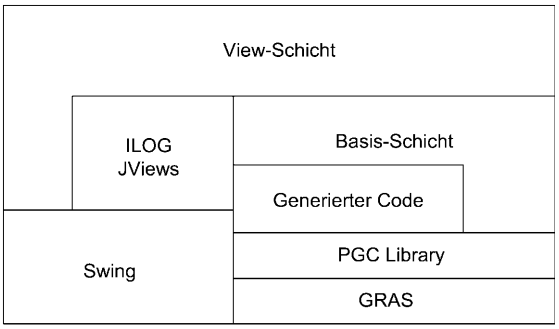


Abb. 13. Architektur des UPGRADE-Rahmenwerks

Graphschema definiert wird. Die prozedurale Schnittstelle von GRAS bietet Elementaroperationen zum Lesen und Navigieren, zum Erzeugen/Löschen von Knoten und Kanten sowie zum Schreiben von Attributwerten an. In PROGRES spezifizierte Graphersetzungsregeln werden auf diese Elementaroperationen zurückgeführt. GRAS stellt eine Reihe von Funktionen zur Verfügung, die in kommerziellen Datenbanksystemen fehlen, z.B. inkrementelle Berechnung von abgeleiteten Attributen und Pfaden sowie unbeschränktes Undo/Redo.

Die Managementumgebung und die Entwicklerumgebung sind mit Hilfe eines graphbasierten Rahmenwerks realisiert, das mit *UPGRADE* bezeichnet wird (*U*niversal *P*latform for *GRA*ph-Based *DE*velopment [32]). UPGRADE ist der Nachfolger des ECU-Rahmenwerks [29], das derzeit noch mit der PROGRES-Umgebung ausgeliefert wird, um auf einfache Weise zu einer PROGRES-Spezifkation ein graphisches, interaktives Werkzeug erzeugen zu können. Diverse Beschränkungen dieses Rahmenwerks (insbesondere hinsichtlich Wartbarkeit und Erweiterbarkeit) machten eine Neuentwicklung erforderlich.

Mit Hilfe des UPGRADE-Rahmenwerks lassen sich interaktive Anwendungen erzeugen, die auf dem aus einer PROGRES-Spezifikation generierten Code operieren. Im einfachsten Fall kann die Anwendung nahezu vollständig generiert werden und bietet dann dem Benutzer eine Standard-Graphsicht, die sich interaktiv konfigurieren lässt. Für die Realisierung der Management- und der Entwicklerumgebung reicht dieser Ansatz jedoch nicht aus. Hier werden Graphsichten unterschiedlichen Inhalts sowie andersartige Sichten (Bäume und Tabellen) benötigt, die auf der Basis des Rahmenwerks implementiert werden müssen (s. Abschnitt 7).

Das Rahmenwerk setzt sich aus generierten und selbstentwickelten Anteilen sowie aus Komponenten von Drittanbietern zusammen (Abb. 13). Auf der Datenbankseite (rechts) liegt oberhalb des Datenbanksystems GRAS eine Bibliothek, die vom generierten Code benutzt wird. Die Basisschicht implementiert diverse Filter, um jeweils sichtenrelevante Ereignisse an die View-Schicht hinaufzureichen. Für die Benutzeroberfläche wird Java Swing verwendet. Darauf basiert das kommerzielle System ILOG JViews, das in UPGRADE für die Repräsentation von Graphen eingesetzt wird. Die View-Schicht dient schließlich zur Kopplung von Benutzerschnittstelle und Datenbank.

Das UPGRADE-Rahmenwerk umfasst z.Zt. ca. 70.000 Zeilen Java-Code. Die spezifischen Erweiterungen für das Managementsystem haben einen Umfang von ca. 10.000 Zeilen.

7 Anwendungen

Zur Präsentation unseres Ansatzes haben wir in den vorherigen Abschnitten ein Beispiel aus der Softwaretechnik gewählt. Darüber hinaus wurden in zwei interdisziplinären Forschungsprojekten Anwendungen in der Fertigungstechnik und in der Verfahrenstechnik untersucht.

Im Rahmen des *SUKITS-Projekts* [52] wurde ein Vorläufer des AHEAD-Systems entwickelt. Dabei stand das Management von Produkten, das auf dem in Abschnitt 3.1 beschriebenen CoMa-Modell basierte, im Mittelpunkt. Für das Management von Aktivitäten und Ressourcen wurden Vorläufer des DYNAMITE- bzw. RESMOD-Modells eingesetzt. Das SUKITS-System wurde auf das Management von Entwicklungsprozessen in der *Fertigungstechnik* (Metall- und Kunststoffverarbeitung) angewendet. Es ist ausführlich in [52, 68] beschrieben.

Im Sonderforschungsbereich 476 *IMPROVE* [50, 52] werden Entwicklungsprozesse in der *Verfahrenstechnik* studiert. Gegenstand der Entwicklung sind verfahrenstechnische Anlagen. Am SFB 476 sind Lehrstühle der RWTH Aachen aus der Informatik und den Ingenieurwissenschaften beteiligt. Als konkretes und umfassendes Fallbeispiel wird derzeit die Entwicklung einer Polyamid6-Anlage untersucht; dies geschieht in Zusammenarbeit mit einem Industriepartner (Bayer AG).

Das AHEAD-System entsteht im Rahmen eines der Teilprojekte des SFB. Es ist ein zentraler Bestandteil eines Prototyps zur Unterstützung verfahrenstechnischer Entwicklungsprozesse, der eine Reihe weiterer Werkzeuge umfasst, die von anderen Projektpartnern entwickelt werden. Der Prototyp wurde im Mai 2000 bei der (erfolgreichen) Fortsetzungsbegehung vorgeführt. Darüber hinaus wurde er im November 2000 auf einem vom SFB 476 veranstalteten Workshop der verfahrenstechnischen Industrie und in diesem Anwendungsbereich aktiven Werkzeuganbietern demonstriert.

Im SFB wird eine frühe Phase des Entwicklungsprozesses betrachtet. Sie wird als *Basic Engineering* bezeichnet und schließt mit einem Grobkonzept für die verfahrenstechnische Anlage ab. Dieses Grobkonzept wird durch ein Fließbild beschrieben, das die Verfahrensschritte und deren Zusammenspiel wiedergibt. Basic Engineering umfasst außer der Fließbilderstellung z.B. Simulationen, Laborversuche und Kostenschätzungen. Im (hier nicht betrachteten) *Detail Engineering* wird das Verfahrenskonzept so weit ausgearbeitet, dass die Anlage anschließend gebaut werden kann (z.B. Auswahl und Auslegung von Apparaten, Planung des Layouts der Anlage etc.).

Entwicklungsprozesse im Basic Engineering sind hochgradig dynamisch. Verfahrensalternativen werden ausgearbeitet und anschließend bewertet. Die Struktur des Fließbildes liegt nicht a priori fest, sondern hängt von den getroffenen Entwurfsentscheidungen ab. Mit Hilfe von Simulationen und Laborexperimenten werden Verfahrensalternativen untersucht und bewertet. Es gibt vielfältige Rückgriffe, die dazu führen, dass Entwurfsentscheidungen revidiert und neue Alternativen untersucht werden müssen.

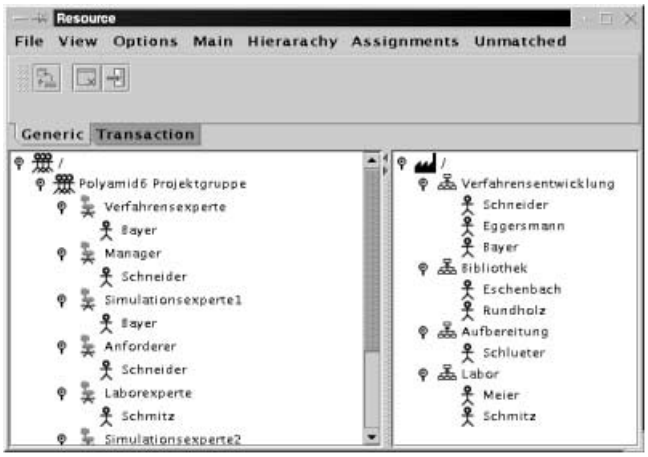


Abb. 14. Ressourcenmanagement

Die folgende Beispielsitzung deckt nur einen kleinen (und vereinfachten) Ausschnitt aus der Entwicklung einer Polyamid6-Anlage ab. Sie bezieht sich auf die Untersuchung der Reaktion (von Monomeren zum gewünschten Polymer). Mit Hilfe der Beispielsitzung sollen Funktionalität und Benutzerschnittstelle des AHEAD-Systems illustriert werden. Dabei beschränken wir uns auf die Laufzeitunterstützung durch die Managementumgebung bzw. die Entwicklerumgebung. Zur Anpassung des AHEAD-Systems an die Verfahrenstechnik wurden mit Hilfe von Rational Rose entsprechende UML-Diagramme erstellt; darauf gehen wir aus Platzgründen jedoch im Folgenden nicht ein.

Mit Hilfe der Beispielsitzung soll insbesondere gezeigt werden, dass AHEAD das Management von Produkten, Aktivitäten und Ressourcen in integrierter Weise unterstützt, während kommmerzielle Systeme in der Regel nur Teilbereiche abdecken. Ein weiterer Schwerpunkt liegt auf der Dynamik von Entwicklungsprozessen, die insbesondere von kommerziellen Workflowmanagementsystemen nur unzureichend berücksichtigt wird.

Die Managementumgebung stellt ein Werkzeug zum Ressourcenmanagement zur Verfügung, mit dessen Hilfe der Projektmanager sein Entwicklungsteam definiert (Abb. 14). Zu diesem Zweck werden baumartige Sichten auf die Ist- und Plan-Ressourcen angeboten (vgl. auch Abb. 4). In der rechten Sicht werden die Ist-Basisressourcen angezeigt. Die Mitarbeiter werden (permanent) unterschiedlichen Abteilungen zugeordnet. In der linken Sicht erscheinen die Plan-Projektressourcen, d.h. die im Entwicklungsteam vorhandenen (auf die Dauer des Projekts befristeten) Stellen, und die Ist-Projektressourcen, d.h. die den Stellen jeweils zugeordneten Mitarbeiter. Ein Mitarbeiter darf mehrere Stellen einnehmen; z.B. fungiert Bayer als Verfahrensexperte und Simulationsexperte.

Darüber hinaus bietet die Managementumgebung ein Werkzeug zum Management von Aktivitäten an (Abb. 15), das auf dynamischen Aufgabennetzen basiert (vgl. auch Abb. 3). Die obere Baumsicht auf der linken Seite repräsentiert die Aufgabenhierarchie. In der rechts angezeigten Graphsicht wird die Hierarchie durch das Layout visualisiert: Übergeordnete Aufgaben erscheinen oberhalb der Unteraufgaben. Um die Graphik übersichtlich zu halten, wird auf die explizite Darstellung von Hierarchiebeziehungen verzichtet. Ein- und Ausgaben sowie Kontroll- und Datenflüsse werden wie in Abb. 3 dargestellt; Aufgabenzustände werden durch Ikonen repräsentiert.

Auf der obersten Ebene wird der Entwicklungsprozess in die Definition von Anforderungen, die Untersuchung der Reaktion (und weiterer, hier nicht gezeigter Verfahrensschritte) und das anschließende Detail Engineering zerlegt. Die Aufgabe ReaktionUntersuchen befindet sich im Zustand Active, ihre Unteraufgaben Reaktionsalternativen und Zusammenfassung nehmen die Zustände Waiting und InDefinition ein. In dieser frühen Phase der Entwicklung steht lediglich fest, dass zunächst Reaktionsalternativen ausgearbeitet werden müssen und abschließend die jeweiligen Ergebnisse zusammengefasst und bewertet werden müssen. Das Aufgabennetz ist also zu diesem Zeitpunkt noch unvollständig.

Die untere Baumsicht auf der linken Seite stellt die Projektressourcen dar, die bereits in Abb. 14 zu sehen waren. Sie ermöglicht es dem Projektmanager, den Aufgaben Mitarbeiter zuzuordnen. Auf diese Weise werden Aktivitäten- und Ressourcenmanagement integriert. In unserem Beispielprozess wird der Aufgabe Reaktionsalternativen die Mitarbeiterin Bayer zugeordnet.

Die Entwicklerumgebung bietet eine Agenda an, die eine Liste von persönlichen Aufgaben anzeigt (oberes Fenster in Abb. 16). Ein Entwickler kann eine Aufgabe aus der Agenda auswählen und Zustandsübergänge ausführen (Starten, Beenden etc.). Zur Bearbeitung einer aktiven Aufgabe baut die Entwicklerumgebung einen Arbeitskontext auf (unteres Fenster). Dort werden Eingaben, Ausgaben und Zustand der ausgewählten Aufgabe graphisch angezeigt. Ferner erscheint links unten eine Liste der für die Bearbeitung der Aufgabe relevanten Dokumente. Für ein ausgewähltes Dokument wird der entsprechende Versionsgraph rechts unten angezeigt (der im hier gezeigten Beispiel allerdings nur eine Version enthält).

Die Verwaltung des Arbeitskontextes erfordert eine Integration von Aktivitäten- und Produktmanagement. In den Aufgabennetzen werden Ein- und Ausgaben von Aufgaben festgelegt; diese müssen im Arbeitskontext erscheinen. Durch Konsumieren von Eingaben (diese Operation ist formal in Abb. 8 spezifiziert) werden Versionen von Eingabedokumenten in den Arbeitskontext importiert; umgekehrt werden Ausgaben durch Produzieren und Freigeben aus dem Arbeitskontext exportiert. Durch die Verwaltung des Arbeitskontextes wird verhindert, dass die Entwickler auf den falschen (z.B. veralteten) Versionen arbeiten. Insbesondere beim Simultaneous Engineering spielt die Versionskontrolle für den Arbeitskontext eine zentrale Rolle.

Vom Arbeitskontext aus lassen sich externe Entwicklungswerkzeuge (z.B. Fließbildwerkzeuge oder Simulatoren) aktivieren. Dies geschieht mit Hilfe von CORBA-Wrappern, welche die zum Aufruf benötigten Dokumentversionen bereitstellen und das vom Entwickler ausgewählte Werkzeug starten. Auf diese Weise werden technische Ressourcen in die Entwicklungsumgebung integriert.

Abbildung 17 zeigt das Aufgabennetz zu einem späteren Zeitpunkt der Entwicklung. Nachdem Reaktionsalternativen in das Fließbild eingetragen wurden, wurde das Aufgabennetz entsprechend erweitert, um die Reaktion in ei-

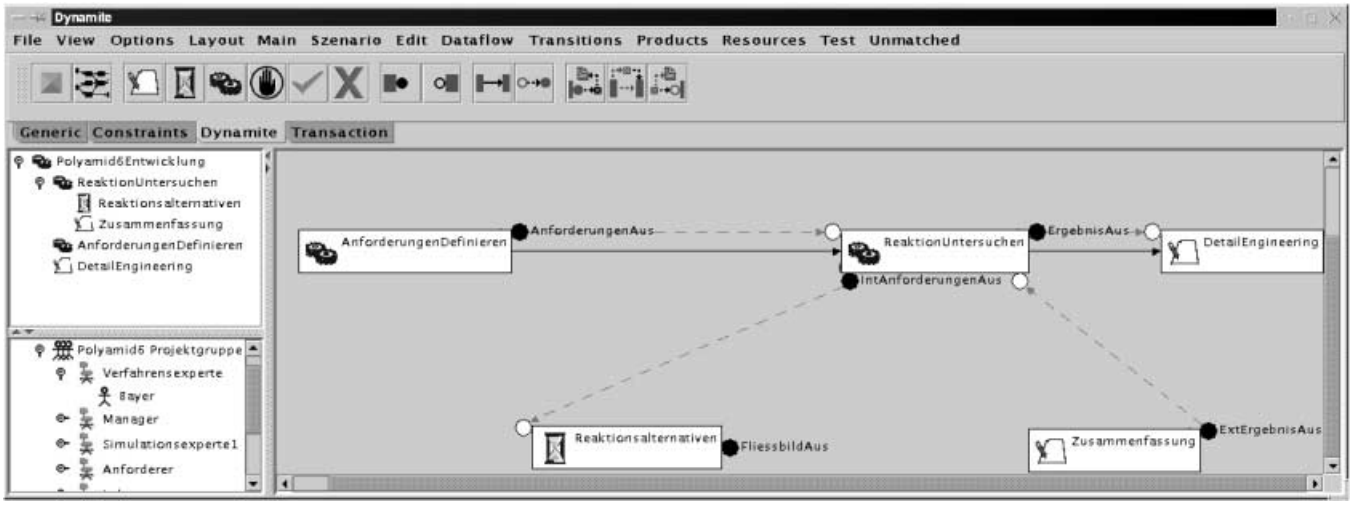


Abb. 15. Initiales Netz

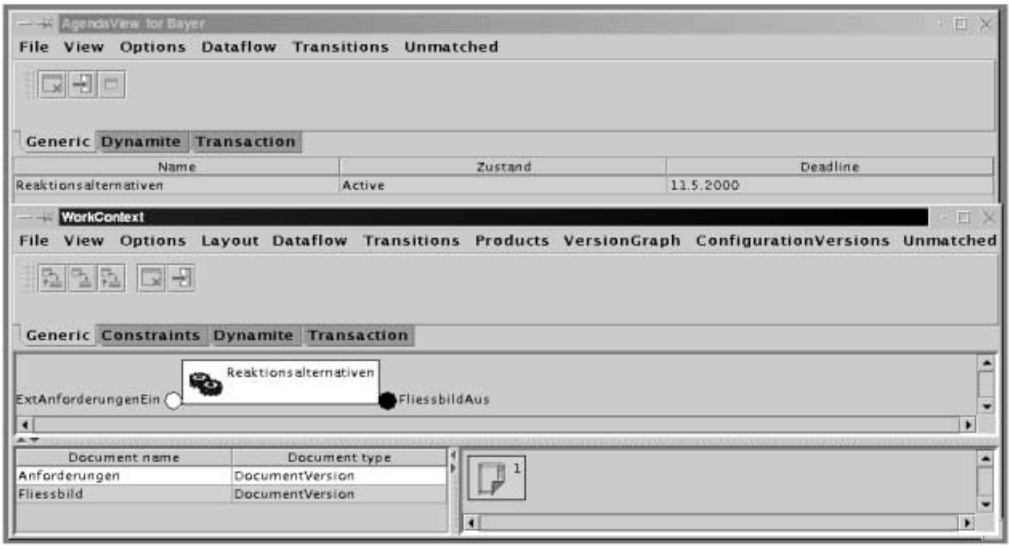


Abb. 16. Agenda

nem Rührkesselreaktor (CSTR) bzw. in einem Rohrreaktor (PFR) zu untersuchen. Zu diesem Zweck werden jeweils Simulationsmodelle erstellt und ausgeführt. Im Falle des Rührkesselreaktors wurde die Simulation durch Laborexperimente validiert. Die Ergebnisse der Simulation werden ins Fließbild übertragen. Dies geschieht über eine Rückgriffsbeziehung und einen entsprechenden Datenfluss. Die Aufgabe Reaktionsalternativen bleibt aktiv, solange sie Feedback von nachgelagerten Aufgaben erwartet (Simultaneous Engineering).

Schließlich bietet die Managementumgebung auch ein Werkzeug zum Management von Produkten an. Im Schnappschuss aus Abb. 18 wird nur ein Teil der Funktionalität dieses Werkzeugs illustriert; insbesondere fehlen die Konfigurationsgraphen aus Abb. 2. Für jede Aufgabe wird ein Workspace angelegt, der die jeweils relevanten Dokumente enthält. Die Baumsicht auf der linken Seite stellt die hierarchische Organisation der Workspaces dar. In der Graphsicht auf der rechten Seite ist darüber hinaus zu erkennen, dass ein Dokument in mehreren Workspaces enthalten sein kann (was zu Duplikaten in der Baumsicht führt). Ferner sind die Versionen eines Dokuments durch Nachfolgerelationen verbunden.

8 Verwandte Ansätze

Der folgende Literaturvergleich ist in drei Unterabschnitte gegliedert, die sich auf die Inhalte der Abschnitte 3–5 beziehen.

8.1 Managementmodelle

In unterschiedlichen Anwendungsbereichen wie Softwareentwicklung, Maschinenbau oder VLSI-Design ist eine große Fülle von Modellen für das Produktmanagement vorgeschlagen worden [10, 20, 36]. Das in Abschnitt 3.1 präsentierte CoMa-Modell ist das älteste der drei Teilmodelle, aus denen sich das Managementmodell des AHEAD-Systems zusammensetzt [66]. Es zeichnet sich durch die Integration von Versions-, Konfigurations- und Konsistenzkontrolle aus. Insbesondere der letzte Aspekt wird in vielen Produktmanage-

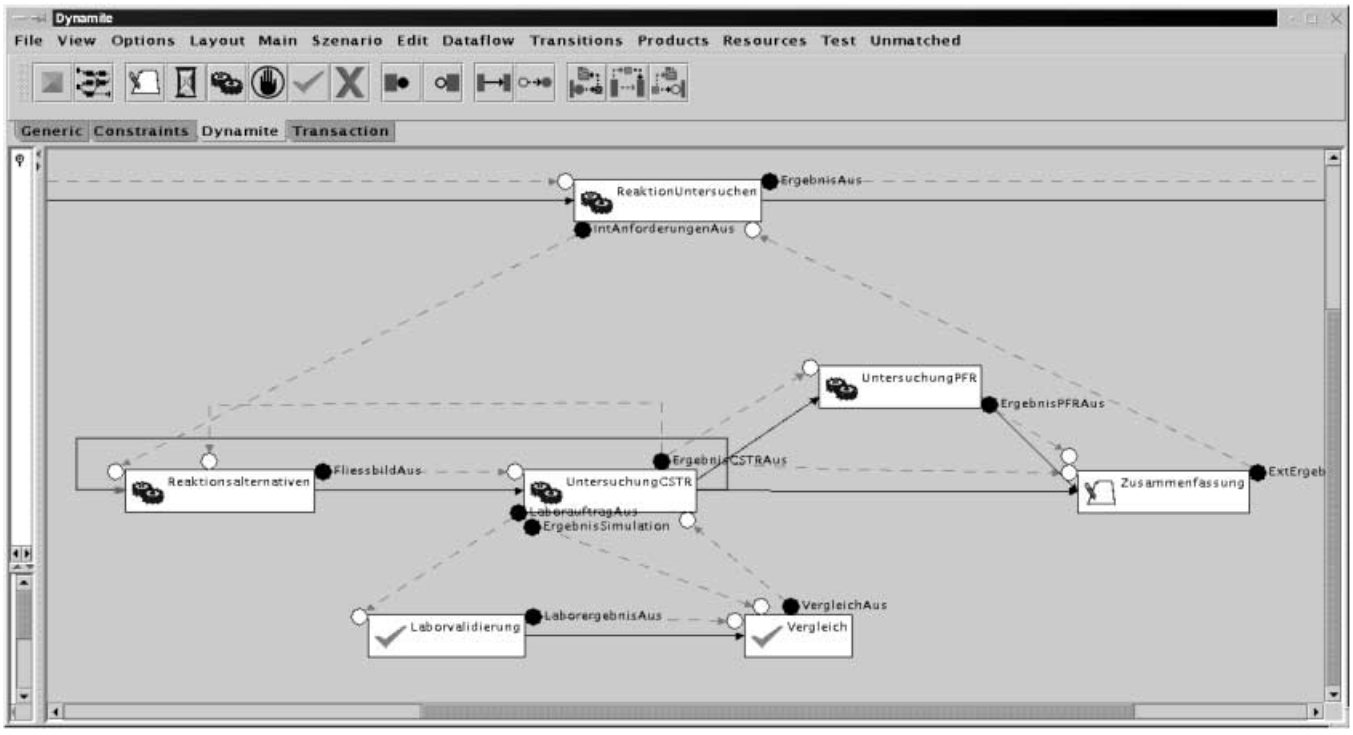


Abb. 17. Erweitertes Netz mit Rückgriff

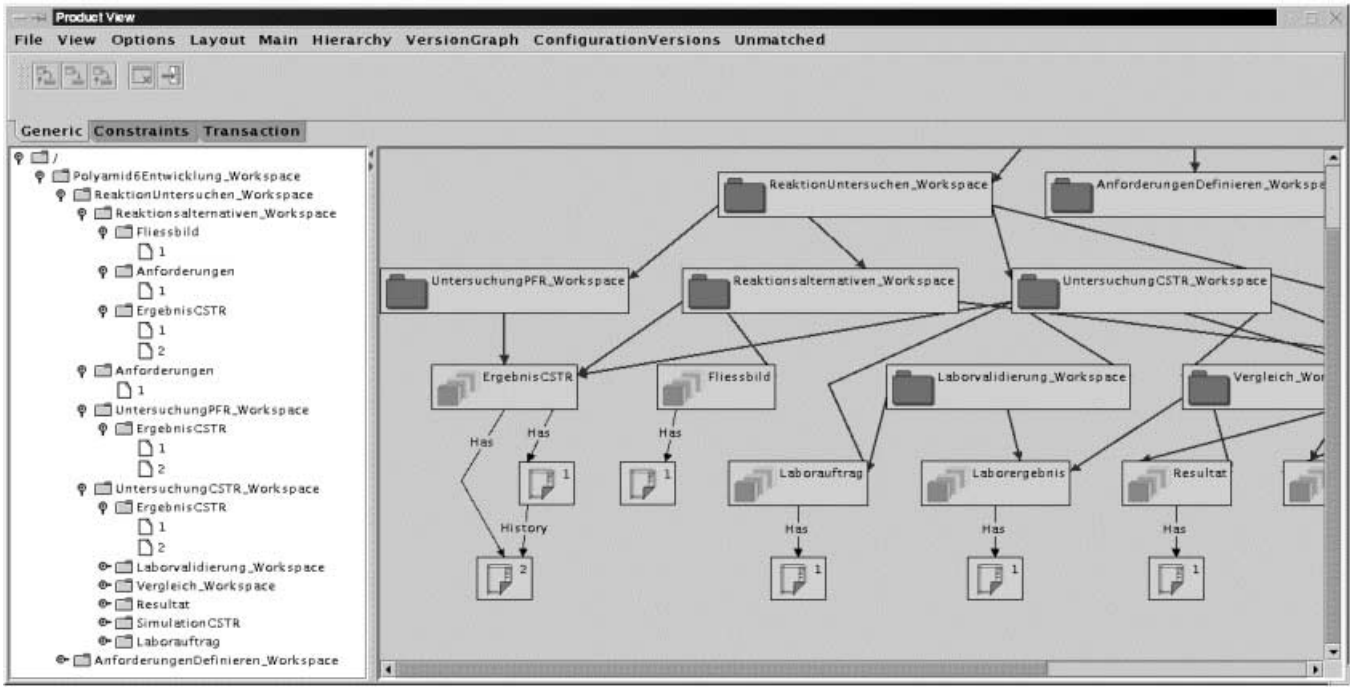


Abb. 18. Produktmanagement

mentsystemen vernachlässigt, die Dokumente hierarchisch organisieren (z.B. unterstützt das SCM-System ClearCase [45] die Versionierung von Dateien und Verzeichnissen). Die größten Ähnlichkeiten bestehen zum Objekt-Versionsmodell (OVM [35]), in dem wie in CoMa zwischen einer Objekt- und einer Versionsebene unterschieden wird und sich ebenfalls Abhängigkeitsbeziehungen darstellen lassen. Ein Unterschied besteht in der Behandlung von Konfigurationen: In CoMa werden sie auf die gleiche Weise wie Dokumente versioniert, in OVM ist dies nicht der Fall.

Das Aktivitätenmodell aus Abschnitt 3.2 (DYNAMITE) basiert auf dynamischen Aufgabennetzen, die es gestatten, Planen und Ausführen nahtlos zu verschränken. Aufgabennetze werden zur Laufzeit aufgebaut, indem Typen von Aufgaben, Kontroll- und Datenflüssen etc. instanziiert werden. Die Topologie der Netze wird erst zur Laufzeit fest-

gelegt und kann jederzeit geändert werden. In vielen – insbesondere kommerziellen – Workflowmanagementsystemen ist der Workflow dagegen i.W. statisch festgelegt [44, 27]. Es gibt jedoch eine Reihe von Systemen, welche die Verschränkung von Planung und Ausführung ebenfalls unterstützen. Dazu zählen z.B. SPADE [1], CONCORD [54] und ADEPT$_{flex}$ [53]. Dabei sind beliebige Änderungen möglich (soweit sie mit dem aktuellen Ausführungszustand verträglich sind), während in DYNAMITE die zur Laufzeit erlaubten Änderungen durch ein domänenspezifisches Schema eingeschränkt werden (s. Abb. 9). Dies ist auch im EPOS-System [28] der Fall. Dort werden jedoch Aufgabennetze automatisch erzeugt, d.h. das Management wird wie in vielen Workflowmanagementsystemen automatisiert. In DYNAMITE werden dem Manager dagegen interaktive Werkzeuge angeboten, um Aufgabennetze aufzubauen, zu analysieren und umzuplanen.

Das in Abschnitt 3.3 beschriebene Ressourcenmodell RESMOD definiert allgemeine Basiskonzepte für das Ressourcenmanagement, z.B. Soll- und Istressourcen, Basis- und Projektressourcen sowie Ressourcenhierarchien. Auf diesem Grundmodell lassen sich unterschiedliche Modelle für die Projektorganisation definieren. In Workflowmanagementsystemen werden dagegen häufig spezifische Organisationsmodelle vordefiniert, die nicht hinreichend flexibel sind, um die in einem Unternehmen etablierten Organisationsstrukturen abzubilden [11, 25, 56, 57]. Eine Ausnahme stellt das Workflowmanagementsystem MOBILE dar, das ein sehr allgemeines Organisationsmodell zur Verfügung stellt [8, 7]. Allerdings müssen in MOBILE Organisationsmodelle von Grund auf definiert werden, ohne dass man wie in RESMOD auf vordefinierte, hinreichend allgemeine Basiskonzepte zurückgreifen könnte.

Zum Abschluss dieses Unterabschnitts gehen wir noch auf die Frage ein, inwieweit in anderen Ansätzen das Management von Produkten, Aktivitäten und Ressourcen integriert behandelt wird. Auf die grundsätzlichen Defizite von Projekt-, Produkt- und Workflowmanagementsystemen haben wir bereits in der Einleitung hingewiesen. Unter diesen Systemen sind die Produktmanagementsysteme bezüglich Überdeckung und Integration am weitesten fortgeschritten. So gibt es einige Produktmanagementsysteme, die eine Komponente zum Workflowmanagement anbieten. Als Beispiel lässt sich das in der verfahrenstechnischen Industrie verbreitete System Documentum[7] anführen. Die Workflowmanagement-Komponente lässt sich jedoch in hochgradig dynamischen Entwicklungsprozessen nur beschränkt einsetzen. Ein anderer Weg wird in ClearCase [45] verfolgt. Mit Hilfe von ClearGuide [46] werden Projektmanagement und Softwarekonfigurationsmanagement integriert. Das dem Projektmanagement zugrunde liegende Modell basiert auf dynamisch erweiterbaren, hierarchischen Netzplänen und bietet weniger Ausdrucksmöglichkeiten als das DYNAMITE-Modell (z.B. keine Rückgriffe, keine selektiven Freigaben etc.).

[7] `http://www.documentum.com`

8.2 Formale Spezifikation

Für die formale Spezifikation des Managementmodells verwenden wir Graphen und Graphersetzungssysteme. Dieser Spezifikationsansatz bietet folgende Vorteile:

- Das Datenmodell der attributierten Graphen ist universell anwendbar und eignet sich für die Modellierung komplexer Strukturen wie z.B. Produktkonfigurationen, Versionshistorien und Aufgabennetze.
- Operationen auf Graphen lassen sich deklarativ mit Hilfe von Graphersetzungsregeln auf einem hohen Abstraktionsniveau spezifizieren.
- Aus der Spezifikation lässt sich Code generieren. Auf diese Weise wird der Aufwand zur Realisierung von Werkzeugen signifikant verringert.

Auf einen umfassenden Vergleich mit anderen Spezifikationsansätzen wird aus Platzgründen verzichtet (s. [68]). Wir beschränken uns hier auf die Diskussion von Petri-Netzen, auf denen eine Reihe von Workflowmanagementsystemen bzw. prozesszentrierten Softwareentwicklungsumgebungen basieren (z.B. COSA [47], LEU [11] oder SPADE [1]). Petri-Netze eignen sich für die Spezifikation nebenläufiger Systeme. Auf ihrer Grundlage lässt sich die Analyse, Simulation und Ausführung von Workflows formal beschreiben. Das Verschränken von Edieren und Ausführen erfordert jedoch die formale Beschreibung struktureller Änderungen unter Berücksichtigung des aktuellen Ausführungszustands. Wir haben gezeigt, wie man diese Aufgabe mit Hilfe von Graphersetzungssystemen lösen kann [24, 41, 68].

Während im Bereich der Graphgrammatiken und Graphersetzungssysteme eine Fülle von theoretischen Ansätzen existiert [55], wurden die praktischen Anwendungen lange vernachlässigt, finden jedoch zunehmendes Interesse [12]. Außer dem PROGRES-System stehen z.Zt. nur wenige Werkzeuge zur Verfügung, die auf Graphersetzungsregeln basieren. Als Beispiele lassen sich GENGED und DIAGEN [2], AGG [14] und Fujaba [16] anführen. Diesen Werkzeugen liegen jeweils einfachere Ersetzungskalküle als PROGRES zugrunde. Der Anwendungsbereich des Managements von Entwicklungsprozessen wird nur in wenigen auf Graphersetzungssystemen basierenden Arbeiten behandelt [70, 22, 59].

8.3 Modellierung in der UML

Es gibt eine Reihe von Ansätzen, in denen die UML zur Modellierung von Geschäftsprozessen verwendet wird [40, 64]. Dies geschieht vor allem in der Analysephase, in der informelle Modelle zur Beschreibung existierender bzw. gewünschter Geschäftsprozesse erstellt werden. Dagegen befassen wir uns mit der Erstellung formaler Prozessmodelle, denen durch die Transformation in ein Graphersetzungssystem eine Ausführungssemantik zugeordnet wird.

Ein weiterer Unterschied besteht darin, dass in den oben zitierten Ansätzen Geschäftsprozesse mit Hilfe von Aktivitätsdiagrammen modelliert werden. Diese Art der Modellierung wird jedoch der Dynamik von Entwicklungsprozessen nicht gerecht. Diese Dynamik lässt sich adäquat abbilden, indem man Aufgaben als Objekte modelliert und die

zur Laufzeit möglichen Objektstrukturen durch Klassendiagramme beschreibt. Dieser Modellierungsansatz wird ebenfalls in [18] verfolgt. Dort wird jedoch nur die Modellierung der Struktur behandelt; die Modellierung des Verhaltens wird nicht betrachtet.

Schließlich sei noch darauf hingewiesen, dass im Bereich von Workflowmanagementsystemen und prozesszentrierten Softwareentwicklungsumgebungen eine große Vielfalt von ausführbaren Prozessmodellierungssprachen entstanden ist [9]. Im Lichte dieser Vielfalt erscheint es sinnvoll, den Einsatz einer standardisierten, weitverbreiteten Modellierungssprache zu untersuchen. Die UML bietet mit ihrer breiten Palette von Modellierungselementen reichhaltige Möglichkeiten.

Die Workflow Management Coalition (WfMC) bemüht sich um die Standardisierung im Bereich von Workflowmanagementsystemen [44]. Einer dieser Standards definiert eine Prozessmodellierungssprache [69], in der Prozesse durch Netze von Aktivitäten und Transitionen beschrieben werden. Den Arbeiten der WfMC liegt die Vorstellung zugrunde, dass sich zur Laufzeit das zur Definitionszeit festgelegte Netz von Aktivitäten und Transitionen nicht ändert. Dieser Ansatz erscheint uns bezogen auf die Dynamik von Entwicklungsprozessen als zu stark eingeschränkt.

9 Zusammenfassung und Ausblick

Wir haben ein Managementsystem beschrieben, das Produkte, Aktivitäten und Ressourcen integriert behandelt und dynamische Entwicklungsprozesse unterstützt. Das AHEAD-System operiert auf Graphen; die Funktionalität der Werkzeuge wird mit Hilfe von programmierten Graphersetzungssystemen auf einer hohen Abstraktionsebene spezifiziert. Domänenexperten modellieren Entwicklungsprozesse in der UML. UML-Modelle werden in PROGRES-Spezifikationen übersetzt, aus denen anschließend Code generiert wird.

Im Folgenden geben wir einen Überblick über aktuelle und geplante Arbeiten am AHEAD-System.

9.1 Evaluierung

Wie wir bereits in Abschnitt 7 erwähnt haben, wurde das AHEAD-System auf der Fortsetzungsbegehung des SFB 476 im Mai 2000 und auf einem Workshop im November 2000 demonstriert. Es wurde auf ein Referenzszenario – Basic Engineering einer Polyamid6-Anlage – angewendet, das von den Ingenieurpartnern im SFB in Kooperation mit der Bayer AG ausgearbeitet wurde. Der Polyamid6-Entwicklungsprozess ließ sich mit den AHEAD-Managementmodellen adäquat beschreiben. Die ausgearbeitete Demonstrationssitzung zeigt den Vorteil des integrierten Ansatzes zum Management dynamischer Entwicklungsprozesse.

Da wesentliche Teile des Systems erst im Laufe des vergangenen Jahres entstanden, konnte noch keine umfassende Evaluierung durchgeführt werden. Die Evaluierung erfolgt in zwei Stufen. Zunächst wird das AHEAD-System von den Ingenieurpartnern im SFB evaluiert. Nach dieser internen Evaluierung (und entsprechender Überarbeitung) folgt eine externe Evaluierung durch Anwender aus der verfahrenstechnischen Industrie. Die Evaluierung soll zunächst mit Hilfe von Benutzungsszenarien und Befragungen durchgeführt werden, bevor das System in einem Pilotprojekt eingesetzt wird.

Auf diese Weise erhält man qualitative Aussagen über den Nutzen des AHEAD-Systems. Quantitative Aussagen – z.B. Aussagen zur Steigerung der Produktivität – erfordern umfangreiche Feldversuche und sind wegen der Einmaligkeit von Entwicklungsprozessen prinzipiell problematisch. Auch wenn Projekte mit gleichartigen Aufgabenstellungen bearbeitet werden, ändert sich der Entwicklungsprozess ständig durch Lernen und Wiederverwendung. Ferner beeinflussen viele, nur schwer erfassbare und isolierbare Faktoren die Produktivität der Entwickler.

9.2 Nutzung von kommerziellen Systemen

Zu einem großen Teil handelt es sich bei dem AHEAD-System um eine Eigenentwicklung. Lediglich für die anwenderseitige Modellierung wird ein kommerzielles System eingesetzt; auch im UPGRADE-Rahmenwerk wird Fremdsoftware benutzt. Zur Realisierung des AHEAD-Systems haben wir insbesondere die PROGRES-Umgebung benutzt, die wir auch in anderen Projekten für die Spezifikation und Generierung graphbasierter Werkzeuge verwenden. Dieser Ansatz ermöglicht es uns, Werkzeuge prototypisch zu realisieren, um neuartige Funktionalität demonstrieren zu können.

In aktuellen Arbeiten untersuchen wir nun den Einsatz kommerzieller Produkt-, Projekt- oder Workflowmanagementsysteme. Dies geschieht aus folgenden Gründen:

- Kommerzielle Systeme decken zwar nicht alle Probleme des Managements von Entwicklungsprozessen ab. Sie bieten aber eine Reihe von Diensten an, die mit Vorteil genutzt werden können (z.B. effiziente Verwaltung großer Datenbestände durch Produktmanagementsysteme).
- Die Nutzung kommerzieller Systeme erleichtert den Transfer von Konzepten in die industrielle Praxis, z.B. weil diese Systeme in den Unternehmen ohnehin eingesetzt werden und die Anwender mit ihnen vertraut sind.

Bisher haben wir die Integration von AHEAD mit einem kommerziellen Workflowmanagementsystem untersucht [3]. Zukünftige Arbeiten sind dem Einsatz von Produkt- und Projektmanagementsystemen gewidmet.

9.3 Modellierung von Entwicklungsprozessen

Da Entwicklungsprozesse hochgradig dynamisch sind, bereitet deren Modellierung Probleme. Nach unseren Erfahrungen erfordert etwa die Erstellung von Klassendiagrammen zur Beschreibung von Aufgabennetzen auf der Typebene (vgl. Abb. 9) validiertes Domänenwissen, das nicht immer vorausgesetzt werden kann. Aufgrund dieser Erfahrungen haben wir ein Ad-hoc-Modell entwickelt, das sich aus vordefinierten Standardtypen zusammensetzt. Mit Hilfe dieses Modells kann man das AHEAD-System auch ohne domänenspezifische Anpassungen unmittelbar einsetzen. Ein

Domänenmodell, mit dem sich Entwicklungsprozesse spezifisch kontrollieren und unterstützen lassen, kann a posteriori auf der Basis bereits ausgeführter Entwicklungsprozesse erstellt werden (Bottom-up-Ansatz zur Prozessmodellierung).

Unsere aktuellen Arbeiten im Bereich der Prozessmodellierung widmen sich folgenden Themen:

- Um den Aufwand zur Erstellung von Prozessmodellen zu verringern, arbeiten wir an einem semi-automatischen Verfahren zum erfahrungsbasierten Lernen, mit dessen Hilfe sich aus Prozessmodellinstanzen entsprechende Prozessmodelldefinitionen ableiten lassen.
- Wie viel Mühe auch immer in die Erstellung eines domänenspezifischen Prozessmodells fließen mag – es kann sich stets bei der Anwendung als inadäquat erweisen. Deshalb wird das AHEAD-System derzeit so erweitert, dass Abweichungen vom Prozessmodell toleriert werden können.
- Eine Abweichung kann dazu führen, dass das Prozessmodell geändert werden muss. Zu diesem Zweck werden Prozessmodellversionen verwaltet, und es wird die Migration einer Prozessmodellinstanz zu einer neuen Version unterstützt.

References

1. Bandinelli, S., Fuggetta, A., Ghezzi, C.: Software Process Model Evolution in the SPADE Environment. IEEE Trans Softw Eng, 19(12): 1128–1144 (1993)
2. Bardohl, R., Minas, M., Schürr, A., Taentzer, G.: Application of Graph Transformation to Visual Languages. In: Ehrig, H. et al. [12], pp 105–180
3. Becker, S.: Integration von Werkzeugen für das Management übergreifender Prozesse. Diplomarbeit, RWTH Aachen, Jan. 2001
4. Berner, S., Glinz, M., Joos, S.: A Classification of Stereotypes for Object-Oriented Modeling Languages. In: France, R., Rumpe [17], pp 249–264
5. Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Reading, Mass: Addison Wesley 1998
6. Bullinger, H.-J., Hartmann, R., Marcial, F.: Engineering Data Management Systeme – EDM als strategischer Erfolgsfaktor im innovativen Unternehmen. Fraunhofer-Institut für Arbeitswissenschaft und Organisation, Stuttgart, Mai 1996
7. Bußler, C.: Organisationsverwaltung in Workflow-Management-Systemen. Techn. Ber. 30–3, Universität Erlangen, März 1997
8. Bußler, C., Jablonski, S.: An Approach to Integrate Workflow Modeling and Organization Modeling in an Enterprise. In: Proceedings of the Third Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, pp 81–95, Morgantown, West Virginia, Apr. 1994. IEEE Computer Society Press
9. Conradi, R., Jaccheri, M.L.: Process Modelling Languages. In: Derniame, J.-C., Baba, A.K., Wastell, D. (Hrsg.): Software Process: Principles, Methodology, and Technology, LNCS 1500, pp 26–52. Berlin Heidelberg New York: Springer 1998
10. Conradi, R., Westfechtel, B.: Version Models for Software Configuration Management. ACM Computing Surveys, 30(2): 232–282 (1998)
11. Dinkhoff, G., Gruhn, V., Saalmann, A., Zielonka, M.: Business Process Modeling in the Workflow Management Environment Leu. In: Loucopoulos, P. (Hrsg.): Proceedings of the 13th International Conference on Object-Oriented and Entity-Relationship Modeling (ER '94), Manchester, UK, LNCS 881, pp 46–63, Berlin Heidelberg New York: Springer 1994
12. Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. (Hrsg.): Handbook on Graph Grammars and Computing by Graph Transformation: Applications, Languages, and Tools, Bd. 2. World Scientific, Singapur, 1999
13. Engels, G., Rozenberg, G. (Hrsg.): TAGT '98 – 6th International Workshop on Theory and Application of Graph Transformation, Paderborn, LNCS 1764, Nov. 1998. Berlin Heidelberg New York: Springer 1998
14. Ermel, C., Rudolf, M., Taentzer, G.: The AGG Approach: Language and Environment. In: Ehrig, H. et al. [12], pp. 551–602
15. Eversheim, W., Bochtler, W., Laufenberg, L. (Hrsg.): Simultaneous Engineering. Berlin Heidelberg New York: Springer 1995
16. Fischer, T., Niere, J., Torunski, L., Zündorf, A.: Story Diagrams: A New Graph Grammar Language Based on the Unified Modeling Language and Java. In: Engels, G., Rozenberg, G. [13], pp. 296–309
17. France, R., Rumpe, B. (Hrsg.): UML '99 – The Unified Modeling Language, Fort Collins, Colorado, LNCS 1723, Okt. 1999. Berlin Heidelberg New York: Springer 1999
18. Franch, X., Ribo, J.M.: Using UML for the Static Part of a Software Process. In: France, R., Rumpe, B. [17], pp. 292–307
19. Gulbins, J., Seyfried, M., Strack-Zimmermann, H.: Dokumentenmanagement – Vom Imaging zum Business-Dokument. Berlin Heidelberg New York: Springer 1999
20. Hamer, P. van den, Lepoeter, K.: Managing Design Data: The Five Dimensions of CAD Frameworks, Configuration Management, and Product Data Management. Proceedings of the IEEE, 84(1): 42–56 (1996)
21. Harris, S.B.: Business Strategy and the Role of Engineering Product Data Management: A Literature Review and Summary of the Emerging Research Questions. Proceedings of the Institution of Mechanical Engineers, Part B (Journal of Engineering Manufacture), 210(B3): 207–220, 1996
22. Heidenreich, G.: Ein generisches Organisationsschema der Konfigurationsverwaltung im Software-Engineering. Techn. Ber. 29/13, Universität Erlangen, Sep. 1996
23. Heimann, P., Joeris, G., Krapp, C.-A., Westfechtel, B.: DYNAMITE: Dynamic Task Nets for Software Process Management. In: Proceedings of the 18th International Conference on Software Engineering, pp. 331–341, Berlin, März 1996. IEEE Computer Society Press
24. Heimann, P., Krapp, C.-A., Westfechtel, B., Joeris, G.: Graph-Based Software Process Management. Int J Softw Eng Knowledge Eng 7(4): 431–455 (1997)
25. IBM, Böblingen: IBM FlowMark: Modeling Workflow, Version 2.1, März 1995
26. Jablonski, S., Böhm, M., Schulze, W. (Hrsg.): Workflow-Management: Entwicklung von Anwendungen und Systemen. Heidelberg: dpunkt.verlag 1997
27. Jablonski, S., Bußler, C.: Workflow Management – Modeling Concepts and Architecture. Bonn: International Thomson Publishing 1996
28. Jaccheri, M.L., Conradi, R.: Techniques for Process Model Evolution in EPOS. IEEE Trans Softw Eng, 19(12): 1145–1156 (1993)
29. Jäger, D.: Generating Tools from Graph-Based Specifications. Information Software and Technology, 42(2): 129–140 (2000)
30. Jäger, D., Schleicher, A., Westfechtel, B.: AHEAD: A Graph-Based System for Modeling and Managing Development Processes. In: Nagl, M. et al. [51], pp. 325–339
31. Jäger, D., Schleicher, A., Westfechtel, B.: Using UML for Software Process Modeling. In: Nierstrasz, O., Lemoine, M. (Hrsg.): Software Engineering – ESEC/FSE '99, Toulouse, LNCS 1687, pp. 91–108, Sep. 1999. Berlin Heidelberg New York: Springer 1999
32. Jäger, D., Schleicher, A., Westfechtel, B.: UPGRADE: A Framework for Building Graph-Based Software Engineering Tools. Techn. Ber. AIB 00-3, RWTH Aachen, 2000
33. Jungbluth, V.: Alles im Griff: Projektmanagementsysteme im Vergleich. c't, (7): 178–189 (1997)
34. Jungbluth, V.: Teamwork: Einführung in die EDV-gestützte Projektplanung. c't, (7): 172–177 (1997)
35. Käfer, W., Mitschang, B.: Flexible Entwurfsdatenverwaltung für CAD-Frameworks: Konzept, Realisierung und Bewertung. In: Oberweis, A., Stucky, W. (Hrsg.): Datenbanksysteme in Büro, Technik und Wissenschaft, Informatik aktuell, pp. 144–163, Braunschweig, Springer 1993
36. Käfer, W., Ritter, N., Schöning, H.: Konfigurierungskonzepte für datenbank-basierte, technische Entwurfsanwendungen. Inf Forsch Entw, 13(1): 1–17 (1998)
37. Katz, R.H.: Toward a Unified Framework for Version Modeling in Engineering Databases. ACM Computing Surveys, 22(4): 375–408 (1990)

38. Kerzner, H.: Project Management: A Systems Approach to Planning, Scheduling, and Controlling. New York: Wiley 1998
39. Kiesel, N., Schürr, A., Westfechtel, B.: GRAS, a Graph-Oriented Software Engineering Database System. Inf Syst, 20(1): 21–51 (1995)
40. Korthaus, A.: Using UML for Business Object Based Systems Modeling. In: Schader, M., Korthaus, A. (Hrsg.): The Unified Modeling Language – Technical Aspects and Applications, pp. 220–237. Heidelberg: Physica-Verlag 1998
41. Krapp, C.-A.: An Adaptable Environment for the Management of Development Processes. Nr. 22 in Aachener Beiträge zur Informatik. Augustinus Buchhandlung, Aachen, 1998
42. Krapp, C.-A., Krüppel, S., Schleicher, A., Westfechtel, B.: Graph-Based Models for Managing Development Processes, Resources, and Products. In: Engels, G., Rozenberg, G. [13], pp 455–474
43. Krüppel, S.: Ein Ressourcenmodell zur Unterstützung von Software-Entwicklungsprozessen. Diplomarbeit, RWTH Aachen, Feb. 1996
44. Lawrence, P. (Hrsg.): Workflow Handbook. Chichester, UK: Wiley 1997
45. Leblang, D.: The CM Challenge: Configuration Management that Works. In: Tichy, W.F. [63], pp. 1–38
46. Leblang, D.: Managing the Software Development Process with ClearGuide. In: Conradi, R. (Hrsg.): Software Configuration Management: ICSE'97 SCM-7 Workshop, Boston, Mass., Mai 1997. LNCS 1235, pp. 66–80, Berlin Heidelberg New York: Springer 1997
47. LEY GmbH, Pulheim: COSA Workflow, Version 2.0, Sep. 1998
48. McIntosh, K.G.: Engineering Data Management – A Guide to Successful Implementation. Maidenhead, England: McGraw-Hill 1995
49. Nagl, M. (Hrsg.): Building Tightly-Integrated Software Development Environments: The IPSEN Approach. LNCS 1170. Berlin Heidelberg New York: Springer 1996
50. Nagl, M., Marquardt, W.: SFB-476 IMPROVE: Informatische Unterstützung übergreifender Entwicklungsprozesse in der Verfahrenstechnik. In: Jarke, M., Pasedach, K., Pohl, K. (Hrsg.): Informatik '97: Informatik als Innovationsmotor, Informatik aktuell, pp. 143–154, Aachen, Sep. 1997. Berlin Heidelberg New York: Springer 1997
51. Nagl, M., Schürr, A., Münch, M. (Hrsg.): AGTIVE – Applications of Graph Transformations with Industrial Relevance, Castle Rolduc, The Netherlands, LNCS 1779, Sep. 1999. Berlin Heidelberg New York: Springer 1999
52. Nagl, M., Westfechtel, B. (Hrsg.): Integration von Entwicklungssystemen in Ingenieuranwendungen. Berlin Heidelberg New York: Springer 1998
53. Reichert, M., Dadam, P.: ADEPTflex – Supporting Dynamic Changes Without Loosing Control. J Intell Inform Syst, 10(2): 93–129 (1998)
54. Ritter, N., Mitschang, B., Härder, T., Gesmann, M., Schöning, H.: Capturing Design Dynamics – The CONCORD Approach. In: Proceedings of the 10th International Conference on Data Engineering, pp 440–451, Houston, Texas, 1994. IEEE Computer Society Press
55. Rozenberg, G. (Hrsg.): Handbook on Graph Grammars and Computing by Graph Transformation: Foundations, Vol. 1. Singapur: World Scientific 1997
56. Rupietta, W.: Organisationsmodellierung zur Unterstützung kooperativer Vorgangsbearbeitung. Wirtschaftsinformatik, 34(1): 26–37 (1992)
57. Rupietta, W.: Organization Models for Cooperative Office Applications. In: Karagiannis, D. (Hrsg.): Proceedings of the 5th International Conference on Database and Expert Systems Applications, Athen, 1994. LNCS 856, pp 114–124, Berlin Heidelberg New York: Springer 1994
58. Schleicher, A.: Formalizing UML-Based Process Models Using Graph Transformations. In: Nagl, M. et al. [51], pp. 341–358
59. Schneider, K.: SESAM: A Hybrid Simulation Formalism Based on Graph Grammar Concepts. In: Ehrig, H. et al. [12], pp. 287–306
60. Schürr, A., Winter, A., Zündorf, A.: Graph Grammar Engineering with PROGRES. In: Schäfer, W., Botella, P. (Hrsg.): Proceedings of the European Software Engineering Conference (ESEC '95), Barcelona, Sep. 1995. LNCS 989, pp. 219–234, Berlin Heidelberg New York: Springer 1995
61. Schürr, A., Winter, A., Zündorf, A.: The PROGRES Approach: Language and Environment. In: Ehrig, H. et al. [12], pp. 487–550
62. Thayer, R.H. (Hrsg.): Tutorial: Software Engineering Project Management. IEEE Computer Society Press, Washington, D.C., 1988
63. Tichy, W.F. (Hrsg.): Configuration Management, Vol. 2 Trends in Software series. New York: Wiley 1994
64. Versteegen, G.: Objektorientierte Geschäftsprozeßmodellierung mit der UML: die Innovator Business Workbench. OBJEKTspektrum, (1): 62–67 (1998)
65. Vossen, G., Becker, J. (Hrsg.): Geschäftsprozessmodellierung und Workflow-Management. Thomson Publishing, Bonn, 1996
66. Westfechtel, B.: Using Programmed Graph Rewriting for the Formal Specification of a Configuration Management System. In: Mayr, E., Schmidt, G., Tinhofer, G. (Hrsg.): Proceedings WG '94 Workshop on Graph-Theoretic Concepts in Computer Science, Herrsching, Juni 1994. LNCS 903, pp. 164–179, Berlin Heidelberg New York: Springer 1995
67. Westfechtel, B.: A Graph-Based System for Managing Configurations of Engineering Design Documents. Int J Softw Eng Knowledge Eng, 6(4): 549–583 (1996)
68. Westfechtel, B.: Models and Tools for Managing Development Processes. LNCS 1646. Berlin Heidelberg New York: Springer 1999
69. Workflow Management Coalition, Brüssel: Workflow Management Coalition Interface 1: Process Definition Interchange Process Model, WfMC TC-1016-P, Okt. 1999
70. Zamperoni, A.: GRIDS – Graph-Based Integrated Development of Software: Integrating Different Perspectives of Software Engineering. In: Proceedings of the 18th International Conference on Software Engineering, pp 48–59, Berlin, März 1996. IEEE Computer Society Press

Bernhard Westfechtel studierte Diplom-Informatik in Erlangen und promovierte 1991 an der RWTH Aachen, wo er sich 1999 auch habilitierte. Seine Forschungsgebiete umfassen Softwareentwicklungsumgebungen, Softwarekonfigurationsverwaltung, Prozessmodellierung, Workflowmanagement, objektorientierte Modellierung, Engineering/Product Data Management, Datenbanken für Ingenieranwendungen und Softwarearchitekturen.