PRO-ART[1]: Erfassung und Verwaltung von Anforderungshistorien

Klaus Pohl, Ralf Dömges, Peter Haumer, Ralf Klamma, Klaus Weidenhaupt

RWTH Aachen, Informatik V
52056 Aachen, Germany
email: {pohl}@informatik.rwth-aachen.de

Zusammenfassung: *Die Nachvollziehbarkeit von Anforderungen (Requirements Traceability) ist eine essentielle Grundvoraussetzung für die Entwicklung hochwertiger Softwaresysteme. Hierbei wird allgemein zwischen der Nachvollziehbarkeit der Verwendung einer Anforderung (Post-Traceability oder Vorwärts-Nachvollziehbarkeit) und der Entstehung einer Anforderung (Pre-Traceability oder Rückwärts-Nachvollziehbarkeit) unterschieden. Für die Unterstützung von Pre-Traceability sind drei Fragen von zentraler Bedeutung: (1) Welche Informationen müssen aufgezeichnet werden? (2) Wie werden diese Informationen strukturiert? (3) Wie werden die Informationen aufgezeichnet?*
In diesem Beitrag stellen wir einerseits drei Rahmenwerke vor, die generische Antworten auf diese Fragen geben. Andererseits skizzieren wir eine detaillierte Ausarbeitung dieser Rahmenwerke, auf der die Requirements Engineering Umgebung PRO-ART basiert. PRO-ART unterstützt die Erfassung und Strukturierung von Trace-Informationen entlang der drei Dimensionen des Requirements Engineerings. Die Aufzeichnung der Daten in einem Trace-Repository erfolgt durch die Werkzeuge der Entwicklungsumgebung, die gemäß einem neuen Ansatzes für prozeßintegrierte Werkzeuge implementiert wurden.
Wir berichten von unseren Erfahrungen mit PRO-ART 1.0, einer ersten prototypischen Implementierung unserer Ansätze, dem daraus resultierenden Re-Design sowie der Re-Implementierung der Umgebung (PRO-ART 2.0).

1 Einleitung

Die Nachvollziehbarkeit (Traceability) von Anforderungen ist eine Grundvoraussetzung für die Erstellung hochqualitativer Softwaresysteme, so daß die Erfassung und Wartung von Trace-Informationen eine essentielle Aktivität während der Erstellung und Verwendung einer Anforderungsdefinition darstellt. Einen umfassenden Überblick über den potentiellen Einsatz von Trace-Informationen und deren Nutzen findet man in [11, 33, 26]. Diese Berichte zeigen unter anderem auf, daß nachvollziehbare Spezifikationen zu weniger Fehlern während der Systementwicklung führen, die Integration von Änderungen erleichtern, eine wichtige Rolle in Vertragssituationen spielen und die Akzeptanz des Softwaresystems erhöhen.

[1] Process and ReprOsitory based Approach for Requirements Traceability

Auf der einen Seite ist die Nachvollziehbarkeit einer Anforderungsspezifikation hin zum Design und zur Implementierung (und umgekehrt) für das Verständnis des aktuellen Systems wichtig. Auf der anderen Seite muß aber auch die Entstehungshistorie einer Anforderungsspezifikation nachvollziehbar sein, um das Verständnis der Anforderungen selbst zu erleichtern. Daher wird allgemein zwischen zwei Arten von Nachvollziehbarkeit differenziert (vgl. [15, 7, 19, 12, 27]). Während die Nachvollziehbarkeit der Verfeinerung, Verwendung und Umsetzung einer Anforderung während des Designs und der Implementierung Post-Traceability oder Vorwärts-Nachvollziehbarkeit bezeichnet wird, spricht man bei der Verfolgbarkeit einer Anforderung zurück zu ihren Ursprüngen von Pre-Traceability oder Rückwärts-Nachvollziehbarkeit (vgl. [11]). Erst durch Pre-Traceability wird die Grundlage für das Management evolvierender Systeme geschaffen, da Pre-Traceability die Identifikation der Teile einer Spezifikation ermöglicht, die von einer Änderung in der Systemumgebung betroffen sind, z.B. einer Änderung der organisatorischen Regeln, des Geschäftsprozesses oder einer erweiterten Nutzung des Systems.

Inzwischen existieren einige Ansätze [1, 9, 2] und sogar kommerzielle Werkzeuge (z.B. *RT* von Teledyne Brown Engineering, *RTM* von Marconi Systems Technology, *RDD 100* von Ascent Logic Cooperation), die die Nachvollziehbarkeit innerhalb von Anforderungsspezifikationen (mit Fokus auf die Nachvollziehbarkeit hierarchischer Dekompositionen) sowie zwischen Anforderungsspezifikationen und Dokumenten späterer Phasen, z.B. Entwurfsbeschreibungen, unterstützen. Leider tendieren diese Ansätze dazu, den Aspekt der Pre-Traceability zu vernachlässigen, d.h. die Erhebung, Definition und Evolution von Anforderungen.

Im Gegensatz dazu zeigen unabhängige empirische Untersuchungen (vgl. Ramesh & Edwards [35] und Gotel & Finkelstein [11]) auf, daß Pre-Traceability mindestens genauso wichtig ist wie Post-Traceability; insbesondere für Systeme, die in sich ständig ändernden Umgebungen eingebettet sind. Jüngste Forschungsbeiträge (vgl. [34, 35, 33, 20, 13, 36]) und Systeme (z.B. *DOORS* von QSS) haben sich daher verstärkt der Pre-Traceability zugewandt. Allerdings betrachten all diese Ansätze nur einzelne Aspekte von Pre-Traceability, z.B. Annotationen von Anforderungsspezifikationen mittels Hypertext oder die Erfassung von Argumentationshistorien. Somit bleibt das aufgezeichnete Wissen und die angebotene Unterstützung im wesentlichen auf einige Teilaspekte beschränkt; ein umfassendes Rahmenwerk für Pre-Traceability fehlt dagegen (vgl. [35, 12, 36]).

Auf dem Weg zu einem solchen Rahmenwerk organisieren wir unseren Beitrag entlang von drei Fragestellungen, auf die jeder Ansatz für Pre-Traceability implizite oder explizite Antworten geben muß:

- Welche Art von Information muß aufgezeichnet werden? (vgl. Abschnitt 2)
- Wie wird diese Information strukturiert? (vgl. Abschnitt 3)
- Wie wird diese Information erfaßt? (vgl. Abschnitt 4)

Wir behandeln diese Fragen auf zwei unterschiedlichen Ebenen. Zum einen stellen wir für jede Fragestellung ein generisches Rahmenwerk vor, das als Wegweiser für konkrete Ansätze zur Unterstützung von Pre-Traceability dient. Zum anderen skizzieren wir die detaillierte Ausarbeitung der Rahmenwerke, auf denen die Requirements Enginee-

ring Umgebung PRO-ART basiert (Abschnitt 2 – 4). In Abschnitt 5 beschreiben wir die resultierende Architektur der PRO-ART Umgebung und ihre Implementation. Die gewonnenen Erfahrungen und offenen Probleme werden in Abschnitt 6 zusammengefaßt.

2 Welche Trace-Informationen müssen erfaßt werden?

Zur Beantwortung dieser Frage haben wir ein Rahmenwerk für das Requirements Engineering vorgeschlagen: *Die drei Dimensionen des Requirements Engineering* (vgl. [24, 25]). In diesem Rahmenwerk wird RE als ein Prozeß verstanden, der sich in dem durch die drei Dimensionen *Repräsentation*, *Spezifikation* und *Übereinstimmung* aufgespannten Raum bewegt (vgl. Abb. 2.1). Danach muß sich Requirements Engineering mit technischen (Repräsentation), kognitiven (Spezifikation) und sozialen Problemen (Übereinstimmung) befassen. Ein traditioneller RE-Prozeß beginnt nahe dem Ursprung mit unterschiedlichen persönlichen Sichten, einem geringen Systemverständnis und einer informellen Repräsentation. Ziel eines jeden RE-Prozesses ist jedoch eine gemeinsame Sicht auf eine gutverstandene, vollständige und zumindest teilweise formalisierte Anforderungsspezifikation.

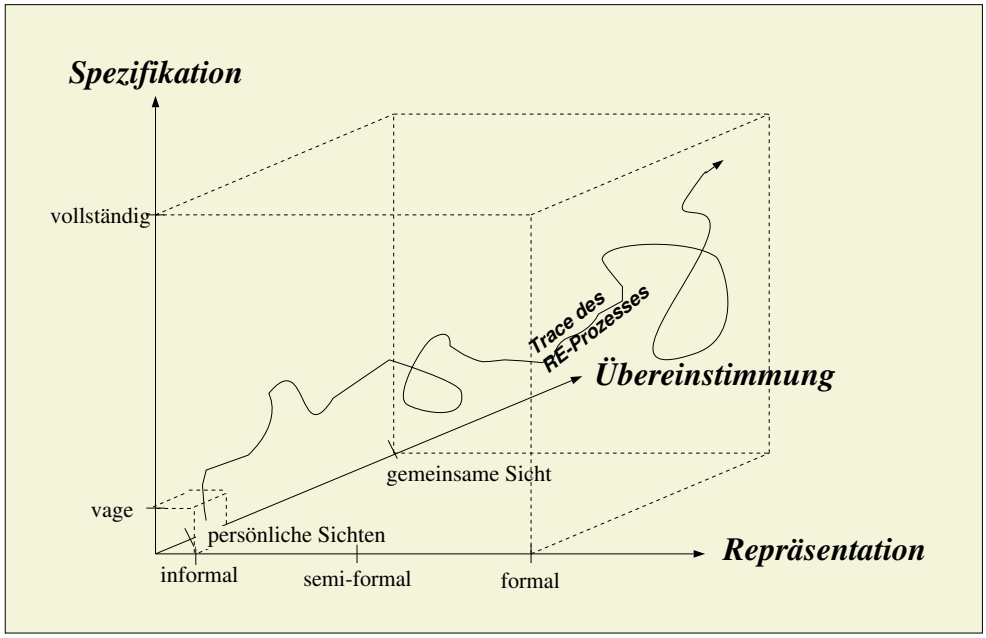

Abb. 2.1: Die drei Dimensionen des Requirements Engineering

Der Trace des RE-Prozesses kann nun als eine beliebige Kurve innerhalb des von den drei Dimensionen aufgespannten Raumes betrachtet werden. Die Erfassung von Trace-Informationen als Grundlage für Pre-Traceability erfordert somit die Aufzeichnung von

- Informationen entlang der Repräsentationsdimension, d.h. die während des Prozesses verwendeten Repräsentationsformalismen und ihre Beziehungen untereinander;
- Informationen entlang der Spezifikationsdimension, d.h. der Inhalt der Spezifikation gemäß bestimmter Standards oder Domänenmodelle;

- Informationen entlang der Übereinstimmungsdimension, d.h. die Standpunkte der unterschiedlichen beteiligten Personen(gruppen), Argumentationen, alternative Lösungsvorschläge und Entscheidungen während des Prozesses;
- Beziehungen zwischen diesen Informationen, d.h. Abhängigkeiten, die zwischen den aufgezeichneten Trace-Objekten bestehen (z.B. welche Entscheidung hatte die Einführung einer neuen Anforderung zur Folge);
- Informationen über die eigentliche Prozeßdurchführung, d.h. Verknüpfung der aufgezeichneten Informationen mit den dafür verantwortlichen Prozeßschritten und Agenten.

3 Wie werden Trace-Informationen strukturiert?

Bei der Wiederverwendung der aufgezeichneten Trace-Informationen während des Systementwicklungsprozesses ergeben sich drei Hauptprobleme, die nur durch eine geeignete Strukturierung der erfaßten Informationen gelöst werden können.

Zum ersten ist bekannt, daß die Verwendung von Trace-Informationen stark von der jeweiligen Personengruppe und der Phase des Softwareentwicklungsprozesses abhängt (vgl. [11]). Das heißt, daß Trace-Informationen später i.a. in einem anderen Kontext verwendet werden, als sie vorher aufgezeichnet wurden. Die Strukturierung und Verwaltung der erfaßten Daten muß also spezifische Sichten und ein selektives Retrieval gemäß dem aktuellen Bedarf bei der Verwendung unterstützen.

Zweitens bietet aufgrund der großen Informationsmenge, die während der Prozeßdurchführung anfällt, nur eine inhaltsorientierte, in einen breiteren Prozeßkontext eingebettete Erfassung von Trace-Informationen eine Basis für geeignete Wiederverwendung (vgl. [27]).

Drittens sind die Personen, die in die Trace-Erfassung involviert sind, i.a. nicht identisch mit den Verwendern der aufgezeichneten Trace-Informationen (vgl. [11]). Daher wird eine standardisierte Traceability-Struktur benötigt, die garantiert, daß Trace-Informationen stets auf die gleiche Art erfaßt werden, um dadurch eine einfachere Verwendung durch Dritte zu ermöglichen.

Zur Lösung der genannten Probleme schlagen wir im folgenden den IRD-Standard (ISO/IEC 10027 [16]) als Rahmenwerk für die Strukturierung von Trace-Informationen vor. Auf der Grundlage des IRDS entwickeln wir eine generische Traceability-Struktur entlang der drei Dimensionen des RE (vgl. Abschnitt 3.1). In Abschnitt 3.2 skizzieren wir die konkrete Traceability-Struktur der PRO-ART Umgebung.

3.1 Entwicklung eines Trace-Repositories

Die Struktur eines nach dem IRD-Standard organisierten Trace-Repositories orientiert sich an der Klassifikationsabstraktion semantischer Datenmodelle. In einem geschichteten Typuniversum mit vier Ebenen stellt die Ebene *n+1 (die definierende Ebene)* ein Typsystem, d.h. die Sprache für die Definition der Ebene *n (die definierte Ebene)* zur Verfügung. In abnehmendem Grad der Abstraktion werden die vier Ebenen *IRD Definition Schema Level*, *IRD Definition Level*, *IRD Level* und *IRD Application Level* genannt.

Um den IRDS-Ansatz auf das Problem der Strukturierung von Trace-Informationen für RE-Prozesse anzuwenden, müssen die oberen drei Ebenen (vgl. Abb. 3.1) mit geeigneten Modellen (Schemata) gefüllt werden. Auf dem IRD Definition Schema Level muß ein Modell (Schema) die Konstrukte bereitstellen, mit denen auf dem IRD Definition Level konkrete Traceability-Modelle erstellt werden. Die Traces selbst werden als Instanzen der Traceability-Modelle auf dem IRD Level aufgezeichnet.

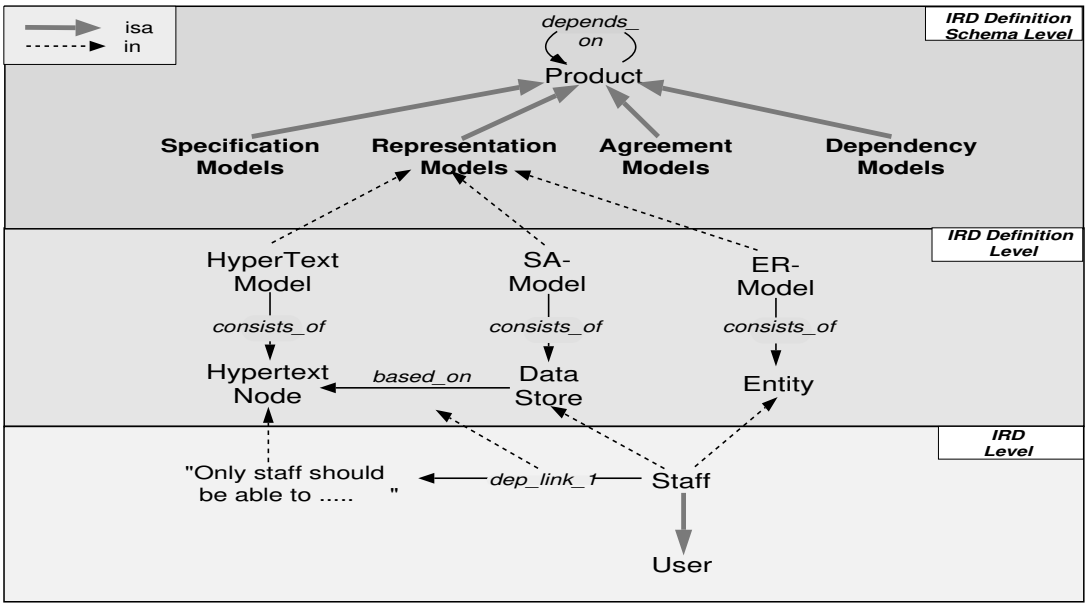

Abb. 3.1: Strukturierung von Trace-Informationen durch Metamodelle

IRD Definition Schema Level: Auf dieser Ebene ist das *Traceability-Metamodell* angesiedelt, das die Konzepte für eine konkrete Traceability-Struktur auf dem IRD Definition Level bereitstellt. Prinzipiell sind zwei Möglichkeiten für die Definition einer konkreten Traceability Struktur entlang der drei Dimensionen des RE denkbar. Eine Möglichkeit besteht darin, ein monolithisches, aus unzusammenhängenden Konzepten bestehendes Modell definieren. Aufgrund der Orthogonalität der anfallenden Informationen führt dieser Ansatz zu einer Explosion von Konzepten. Geeigneter erscheint daher die zweite Möglichkeit, bei der eine Menge von in sich geschlossenen, orthogonalen Modellen entlang jeder Dimension definiert wird und modellübergreifende Beziehungen durch Abhängigkeitsverweise ausgedrückt werden.

Ein wichtiger Beitrag unseres Traceability-Metamodells ist somit die Unterscheidung zwischen vier Arten von Traceability-Modellen (in Abb. 3.1 vereinfacht dargestellt durch das Konzept *Product* und dessen Spezialisierungen). Es sind dies Modelle für die Trace-Informationen entlang der Repräsentations-, Spezifikations- und Übereinstimmungsdimension sowie Abhängigkeitsmodelle, die grundlegende Beziehungstypen zwischen Trace-Objekten definieren (vgl. [27] für eine detaillierte Beschreibung des Traceability-Metamodells).

IRD Definition Level: Auf dem IRD Definition Level wird eine *konkrete Traceability-Struktur* als Instantiierung des Traceability-Metamodells definiert. Diese konkrete

Traceability-Struktur sollte auf die konkreten Bedürfnisse einer Organisation ausgerichtet sein. Sie beschreibt explizit die Typen von Trace-Informationen und Beziehungen, die während des RE-Prozesses aufgezeichnet werden.

Für die Erfassung von Trace-Informationen entlang einer einzelnen Dimension sollten wiederum orthogonale Modelle definiert werden. In Abb. 3.1 sind z.B. drei vereinfachte Modelle für Informationen entlang der Repräsentationsdimension dargestellt: ein Hypertext-Modell, ein Structured-Analysis-Modell und ein Entity-Relationship-Modell. Darüberhinaus müssen mögliche Beziehungen zwischen Trace-Objekten innerhalb eines Abhängigkeitsmodells spezifiziert werden. So wird z.B. der Abhängigkeitstyp (Beziehungstyp) *based_on* in Abb. 3.1 innerhalb eines Abhängigkeitsmodells definiert. Neben der Definition einer Menge von Abhängigkeitstypen sollte das Abhängigkeitsmodell den Gebrauch der Typen restringieren (d.h. die Typen von Trace-Objekten, zwischen denen eine bestimmte Abhängigkeit gezogen werden kann) und ihre automatische Herleitung ermöglichen, z.B. durch deduktive Regeln.

IRDS Level: Abb. 3.1 zeigt beispielhaft, wie ein konkreter Trace unter Verwendung der auf dem IRD Definition Level spezifizierten Schemata auf dem IRD Level abgelegt wird. In dem Beispiel gibt es fünf Trace-Objekte: *Staff*, *User*, *"Only staff should be able to ..."*, *dep_link_1* und der *isA*-Link zwischen *Staff* und *User*. Das Trace-Objekt *"Staff"* repräsentiert sowohl einen Datenspeicher als auch eine Entity und ist somit Instanz zweier unterschiedlicher Klassen. Der Abhängigkeitsverweis *dep_link_1* ist als Instanz des Beziehungstypen *based_on* abgelegt und setzt die Objekte *"Staff"* und *"Only staff should be able to ..."* zueinander in Beziehung, wobei letzteres Objekt eine Instanz des Typen *HypertextNode* darstellt. Abhängigkeiten zwischen Trace-Objekten können also auf zwei Arten repräsentiert werden: durch multiple Instantiierung (in Klassen unterschiedlicher Teilmodelle) und durch explizite Abhängigkeitsverweise (über Teilmodellgrenzen hinweg).

3.2 Die Traceability-Struktur der PRO-ART Umgebung

Die Traceability-Struktur der PRO-ART Umgebung besteht aus sieben orthogonalen Modellen und einem Abhängigkeitsmodell, das 18 Beziehungstypen zwischen Objekten definiert. Diese Modelle umfassen zur Zeit mehr als 250 Klassendefinitionen (Konzepte) sowie über 80 Integritätsbedingungen und deduktive Regeln, die eine konsistente Instantiierung der Traceability-Struktur garantieren.

Auf der Repräsentationsdimension haben wir uns auf Quasi-Industriestandards konzentriert, um zu zeigen, daß unser Ansatz auch gutbekannte Methoden wie OMT oder ER-Modellierung verbessern kann. Zur Zeit unterstützen wir:

- ein Hypertext-Modell für die Strukturierung verschiedenster informeller Repräsentationen wie natürlichsprachliche Systembeschreibungen, Bilder, Tabellen etc. (vgl. [14, 29]);
- ein erweitertes Entity-Relationship-Modell für die Modellierung der Datensicht auf ein System (vgl. [27]);

- ein Structured-Analysis-Modell (Datenflußdiagramme und Datenlexikon) für die Modellierung der funktionalen Sicht auf ein System (vgl. [27]);
- das Objekt- und Verhaltensmodell von OMT (vgl. [22, 27]);
- O-Telos für die Spezifikation von Integritätsregeln und deduktiven Regeln (vgl. [18])

Für die Strukturierung von Trace-Informationen entlang der Spezifikationsdimension haben wir 21 internationale Standards und Richtlinien für das Requirements Engineering analysiert und daraus ein umfassendes Spezifikationsmodell gewonnen, das Requirements Specification Model (RSM) (vgl. [10, 27]).

Kommunikationen, Unterredungen und Verhandlungen zwischen Personen, die in den RE Prozeß involviert sind, führen zu Entscheidungen. Zur Strukturierung solcher Argumentationen wurde von Rittel das Issue Based Information System (IBIS) vorgeschlagen, auf dem viele Ansätze zur Aufzeichnung von Begründungshistorien beruhen (z.B. [6, 32, 34, 8]). Unser Modell für die Strukturierung von Informationen entlang der Übereinstimmungsdimension übernimmt eine Teilmenge des IBIS-Modells, wobei der Fokus auf den Entscheidungen liegt, die während des RE-Prozesses getroffen werden.

Unser Abhängigkeitsmodell ist das Resultat einer umfassenden Literaturanalyse auf den Gebieten Requirements Engineering, Textanalyse, Wissensaquisition, Erfassung von Begründungshistorien und Hypertext. Die 18 Beziehungstypen unseres Abhängigkeitsmodells können in fünf disjunkte Kategorien unterteilt werden: Bedingungsbeziehungen, inhaltliche Beziehungen, Dokumentationsbeziehungen, Evolutionsbeziehungen und Abstraktionsbeziehungen.

Im Vergleich zu anderen Ansätzen bietet ein auf die oben beschriebene Art organisiertes Trace-Repository zwei Vorteile: (1) Jedes Trace-Objekt kann mit jedem anderen über multiple Instantiierung oder explizite Abhängigkeitsverweise in Beziehung gesetzt werden, z.B. kann eine Entity mit einem Abschnitt eines informellen Dokuments verknüpft werden; (2) Die geschichtete Architektur und die formale Definition der Modelle stellt eine semantisch reiche Struktur zur Verfügung, die erst die Grundlage für ein selektives Retrievel der aufgezeichneten Trace-Informationen darstellt.

4 Wie werden Trace-Informationen erfaßt?

Bei der Erfassung von Trace-Informationen spielen zwei Aspekte eine wesentliche Rolle.

Erstens müssen Trace-Informationen *während* der Prozeßausführung aufgezeichnet werden, da es nahezu unmöglich ist, den Trace nach der Fertigstellung der Anforderungsspezifikation zu rekonstruieren.

Zweitens muß die Erfassung von Trace-Informationen so weit wie möglich automatisiert werden[2]. Zum einen bedeutet manuelle Aufzeichnung von Trace-Information einen zusätzlichen Arbeitsaufwand für die am Prozeß beteiligten Personen, so daß Traceability zu arbeits- und kostenintensiv würde. Zum anderen sind manuell aufgezeich-

[2] Die Notwendigkeit einer automatischen Erfassung von Trace-Informationen wird in vielen Veröffentlichungen ausdrücklich betont (vgl. [38, 33, 31])

nete Trace-Informationen oft subjektiv und idealisiert und reflektieren die realen Abläufe nicht hinreichend genau.

Eine vollständig automatisierte Aufzeichnung ist allerdings wegen der großen Menge der im RE-Prozeß anfallenden *informellen* Information i.a. nicht möglich. So kann z.B. die Strukturierung eines Besprechungsprotokolls nur von menschlichen Experten vorgenommen werden. Nichtsdestotrotz soll dieser Abschnitt zeigen, daß eine geeignete RE-Umgebung einen Großteil der Trace-Erfassung übernehmen und die am RE-Prozeß Beteiligten bei der Erfassung der "richtigen" Traces unterstützen kann.

Wir erläutern zunächst die prinzipielle Idee hinter unserem Ansatz für automatisierte Trace-Erfassung (Abschnitt 4.1) und skizzieren dann die konkrete Umsetzung innerhalb der PRO-ART Umgebung (Abschnitt 4.2).

4.1 Automatisierte Trace-Erfassung: prinzipielle Idee

Die Aufzeichnung von Trace-Informationen gemäß der vordefinierten Struktur ist Aufgabe der Prozeßausführungsumgebung, d.h. der Requirements Engineering Umgebung (REU). Jede rechnerbasierte Umgebung bietet eine feste Menge von Aktionen an, durch die ein Produkt erzeugt, modifiziert und gelöscht werden kann. Diese Aktionen sind meistens durch bestimmte Programme (Module, Prozeduren) implementiert.

Die grundlegende Idee hinter unserem Ansatz besteht darin, daß die von der REU angebotenen Aktionen selbst für die Aufzeichnung von Trace-Informationen verantwortlich gemacht werden. Die REU muß also garantieren, daß

- die Ausführung jeder Aktion zusammen mit ihren Ein- und Ausgaben im Trace-Repository abgelegt wird;
- zwischen zwei oder mehreren Objekten Abhängigkeitsverweise gezogen werden, wenn diese durch eine Aktion in Beziehung gesetzt werden;
- der Prozeßausführende und der Prozeßkontext, in den die Aktion eingebettet ist (d.h. die vorangegangenen Aktionen), erfaßt werden.

Die Erfassung dieser Trace-Informationen erfordert eine Erweiterung der Traceability-Modelle auf dem IRD Definition Schema Level. Wir haben unser Modell u.a. durch die Konzepte `Tool` (z.B. *ER-Editor*), `Action` sowie `Input` und `Output` ergänzt (vgl. [27] für Details). Somit kann jede Ausführung einer Aktion als Instanz eines definierten Aktionstypen im Repository abgelegt und die Ein- und Ausgaben mit der Aktion verknüpft werden.

Bezogen auf das Beispiel aus Abschnitt 3.1 erkennt man nun in Abb. 4.1, daß die Entity *Staff* innerhalb der Aktion *Specialize_Entity_Act_12* als Spezialisierung der Enity *User* erzeugt wurde, wobei *User* und der Hypertextknoten *"Only staff should be able to ..."* als Eingaben dienten. Neben der Enity *Staff* wurde im Rahmen dieser Aktion zusätzlich noch der Abhängigkeitsverweis *dep_link_1* vom Typen *based_on* erzeugt.

4.2 PRO-ART: Trace-Erfassung durch Werkzeuge und Prozeßmaschine

Der oben beschriebene Ansatz funktioniert für die Aufzeichnung des Trace einzelner Aktionen. Allerdings ergeben sich Abhängigkeiten zwischen Trace-Objekten häufig erst

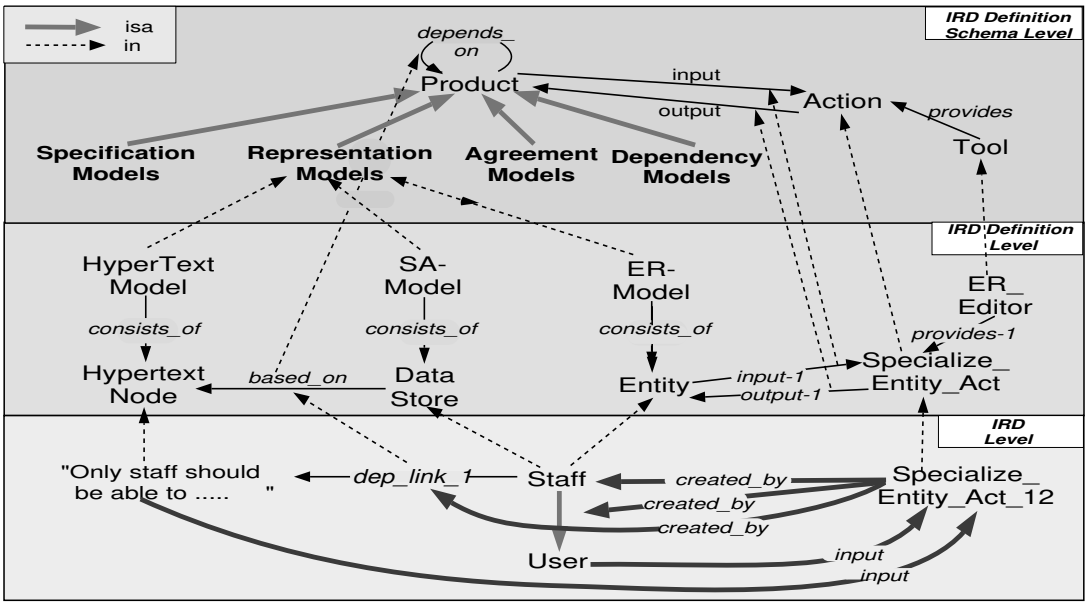

Abb. 4.1: Aufzeichnung von Trace-Informationen durch Werkzeugaktionen

im Kontext einer komplexeren Abfolge von Aktionen. Um auch aktionsübergreifende Abhängigkeiten erfassen zu können, haben wir in PRO-ART den in Abschnitt 4.1 vorgestellten Ansatz in zweifacher Hinsicht erweitert

1. Die PRO-ART Werkzeuge sind interoperabel, d.h. sie können die Resultate ihrer Aktionen untereinander austauschen und lassen eine Adaption ihres Arbeitsmodus, z.B. das Anstoßen von Aktionen, von außen zu;
2. Die Interaktionen zwischen Werkzeugen im Rahmen komplexer Aktionen werden auf der Basis expliziter Prozeßmodelle von einem Prozeßausführungsmechanismus, der Prozeßmaschine, gesteuert.

Die Aufgabe der Trace-Erfassung ist somit zweigeteilt. Während die Werkzeuge der REU die Ausführung einzelner Aktionen protokollieren, ist die Prozeßmaschine dafür verantwortlich, den Prozeßkontext und aktionsübergreifende Abhängigkeiten im Repository aufzuzeichnen. Einzelheiten des in PRO-ART verfolgten Ansatz zur prozeßorientierten Werkzeuginteroperabilität sind in [27, 21, 37, 28, 30] zu finden.

5 PRO-ART: Implementation

Die Architektur der PRO-ART Umgebung besteht aus drei wesentlichen Komponenten: einem zentralen Repository für die Traceability- und Prozeßmodelle, einem Kommunikationsserver und einer Prozeßmaschine für die aktionsübergreifende Werkzeugsteuerung sowie den eigentlichen Entwicklungswerkzeugen (vgl. Abb. 5.1).

Das Trace-Repository, in dem die Traceability-Modelle, die eigentlichen Trace-Daten sowie die Prozeßdefinitionen abgelegt sind, wurde mit Hilfe des ConceptBase-Systems [17], einer Implementierung von O-Telos [23, 18], realisiert. Es besteht zur Zeit aus über 400 O-Telos-Klassendefinitionen, die die Traceability-Struktur und die von den

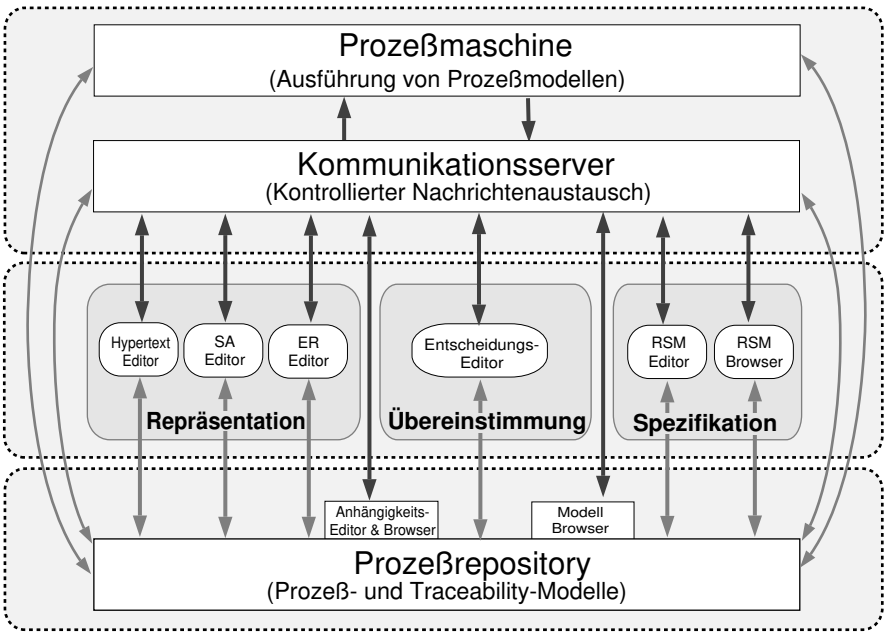

Abb. 5.1: Die Architektur des PRO-ART Umgebung

verschiedenen Aktionen zu speichernden Trace-Informationen spezifizieren. Weiterhin garantieren mehr als 80 deduktive Regeln und Integritätsbedingungen eine korrekte Instantiierung (vgl. [27]).

Die Prozeßmaschine interpretiert die im Repository abgelegten Ablaufdefinitionen, die in dem an der Universität Mailand entwickelten Petrinetzderivat SPADE (vgl. [5, 3, 4]) spezifiziert werden. Dabei interagiert die Prozeßmaschine über einen zentralen IPC-basierten Kommunikationsserver mit den Entwicklungswerkzeugen, erfaßt Informationen über komplexe Aktionen und generiert Trace-Daten gemäß den im Repository abgelegten Traceability-Modellen. Die Implementierung der Prozeßmaschine wird in [21] beschrieben.

Die eigentliche Arbeitsumgebung besteht aus einer Reihe von Werkzeugen, und zwar

- einem Hypertext-Editor für die Erfassung informal notierter Informationen;
- einem SA- und ER-Editor;
- einem Entscheidungs-Editor für die Erfassung von Entscheidungen und Begründungshistorien;
- einem RSM-Editor und -Browser für die Klassifikation von Anforderungen und die Generierung von Spezifikationsdokumenten auf der Basis unseres umfassenden Spezifikationsmodells (RSM);
- einem Abhängigkeits-Editor und -Browser für die Erzeugung und das Retrieval von Beziehungen zwischen Trace-Objekten.

Die Werkzeuge, die Prozeßmaschine und der Kommunikationsserver der PRO-ART Umgebung wurden in C++ unter Verwendung des Andrew-Toolkits und X11/R5 implementiert und umfassen mehr als 120.000 Quellkodezeilen.

In Abb. 2 ist die Erzeugung eines Attributs für die Enity *proceedings* dargestellt. Hier basiert die Definition eines Attributs für die spezielle Ausleihdauer von *proceedings*

(rechts unten) auf einer informell notierten Aussage eines Bibliothekars während eines Benutzerinterviews (*"proceedings should only be checked out for three days"*, links oben). Während der Erzeugung des Attributs selektiert der Systemanalytiker dieses Textstück und die PRO-ART Umgebung setzt diese als Hypertextknoten verwaltete Aussage mit dem erzeugten Attribut in Beziehung. Ein ausführliches Beispiel findet man in [27].

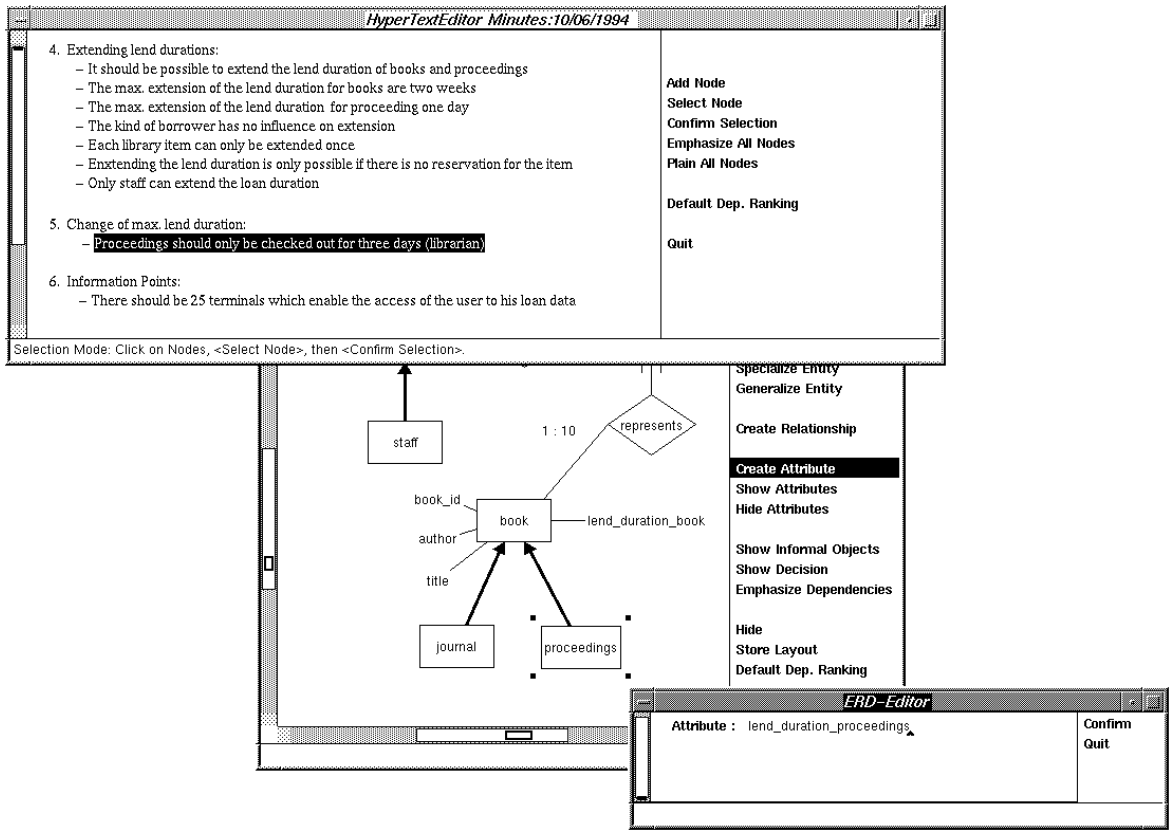

Abb. 2: Erzeugung einer Abhängigkeit zwischen einem ER-Attribut und einem informellen Text

6 Zusammenfassung und Ausblick

In diesem Beitrag haben wir einen Ansatz für Requirements Pre-Traceability vorgestellt, der auf drei Grundpfeilern beruht:

- einem dreidimensionalem Rahmenwerk für das Requirements Engineering, das der Identifikation der aufzuzeichnenden Trace-Informationen dient (Abschnitt 2);
- einem Trace-Repository für die Strukturierung und das selektive Retrieval von Trace-Informationen (Abschnitt 3);
- einem neuartigen Ansatz zur Werkzeuginteroperabilität, der eine weitgehend automatische Erfassung von Trace-Informationen erlaubt (Abschnitt 4).

Die darauf aufbauende RE-Umgebung PRO-ART stellt Werkzeuge für jede der drei Dimensionen zur Verfügung, leitet Trace-Informationen weitgehend automatisch ab und legt diese entsprechend einer gemeinsamen Traceability-Struktur in einem Repository ab.

Die Implementierung von PRO-ART hat uns in die Lage versetzt, praktische Experimente zur Trace-Erfassung durchzuführen. Erste Erfahrungen bei der Modellierung kleinerer Systeme haben gezeigt, daß einige Funktionalitäten der PRO-ART Umgebung im täglichen Umgang Wünsche offen ließen, z.B. die textuelle Darstellung von Abhängigkeitsverweisen. Nach der Behebung solcher frühen Mängel, z.B. durch die Entwicklung eines komfortablen, graphischen Abhängigkeitsbrowsers, haben wir die Umgebung auf neun internationalen Konferenzen und Ausstellungen, darunter die ICRE'94, ICSE'94 und CAiSE'94, vorgeführt.

Diese Präsentationen stießen sowohl seitens der Forschung als auch der Industrie auf großes Interesse und zogen signifikante Rückmeldungen nach sich. Die hilfreichen Kommentare betrafen vor allem folgende Punkte:

- Einige Trace-Information sind so offensichtlich, daß die nicht aufgezeichnet werden sollten, z.B. daß eine Entity durch eine *create_entity*-Aktion erzeugt wurde;
- Trace-Strategien sollten explizit definierbar sein, abhängig von projektspezifischen Bedürfnissen;
- Mehrbenutzer-Erweiterung, z.B. Aufgabenlisten und Kooperationshilfsmittel;
- Zugriffsrechte auf Trace-Informationen.

Weiterhin hat das Interesse an unseren Forschungsresultaten und der PRO-ART Umgebung zu einem Kooperationsprojekt auf dem Gebiet der Verfahrenstechnik geführt. Hier zeigte sich, daß die Nachvollziehbarkeitsideen hinter PRO-ART auch bei der Erstellung von Simulationsmodellen für chemische Prozesse hilfreich waren. obwohl die erste Implementation nicht in der Lage war, die dabei anfallenden großen Datenmengen hinreichend effizient zu verwalten.

Sowohl die Rückmeldungen nach den zahlreichen Präsentationen als auch die Erfahrungen aus dem oben genannten Kooperationsprojekt haben uns zu einem Re-Design und einer Re-Implementierung der Umgebung veranlaßt. Die wesentlichen Vorteile der neuen Version PRO-ART 2.0, die im Oktober 1995 fertiggestellt sein wird, sind:

- Handhabung großer Datenmengen und Transaktionskontrolle durch Verwendung relationaler Datenbanktechnologie;
- Explizite Definition von Trace-Strategien;
- Standardisierte, einfach anpaßbare Werkzeugarchitektur;
- Einheitliche Benutzeroberfläche der PRO-ART Werkzeuge;
- Einfachere Portierbarkeit auf andere Plattformen;
- Vollständig prozeßintegrierte Werkzeuge

Langfristige Forschungsaufgaben sehen wir zum ersten in dem Konfigurations- und Versionsmanagement der anfallenden, großen Trace-Datenmengen. Außerdem werden bestimmte standardisierte Trace-Strategien benötigt, die einen Projektleiter bei der Definition der aufzuzeichnenden projektspezifischen Trace-Informationen unterstützen. Ein weiteres Problem stellt die feingranulare Integration existierender Werkzeuge in die Umgebung dar.

Literaturreferenzen

[1] M. W. Alford. Software Requirements Engineering Methodology (SREM) at the Age of Two. In *Proc. of the 4th Intl. Computer Software & Applications Conference*, pages 866–874, New York, NY, 1980. IEEE Computer Society Press.

[2] M. W. Alford. Software Requirements Engineering Methodology (SREM) at the Age of Eleven – Requirements Driven Design. In P. A. Ng and R. T. Yeh, editors, *Modern Software Engineering*. Van Nostrand Reinhold, 1990.

[3] S. Bandinelli, L. Baresi, A. Fuggetta, and L. Lavazza. Requirements and Early Experiences in the Implementation of the SPADE Repository. In W. Schäfer, editor, *Proc. of the 8th Intl. Software Process Workshop: State of the Practice in Process Technology*, pages 30–33, Wadern, Germany, Mar. 1993. IEEE Computer Society Press.

[4] S. Bandinelli, A. Fuggetta, and C. Ghezzi. Software Process Model Evolution in the SPADE Environment. *IEEE Transactions on Software Engineering*, 19(12):1128–1144, Dec. 1993.

[5] S. Bandinelli, A. Fuggetta, C. Ghezzi, and S. Grigolli. Process Enactment in SPADE. In J.-C. Derniame, editor, *Proc. of the 2nd Europ. Workshop on Software Process Technology*, number 635 in LNCS, pages 67–83, Trondheim, Norway, Sept. 1992. Springer-Verlag.

[6] J. Conklin and M. J. Begeman. gIBIS: A Hypertext Tool for Exploratory Policy Discussion. *ACM Transactions on Office Information Systems*, 6(4):303–331, 1988.

[7] A. M. Davis. The Analysis and Specification of Systems and Software Requirements. In R. Thayer and M. Dorfman, editors, *Systems and Software Requirements Engineering*, pages 119–134. IEEE Computer Society Press — Tutorial, 1990.

[8] G. Fischer. Integrating Construction and Argumentation in Domain-Oriented Design Environments. In *Proc. of the 1st Intl. Symposium of Requirements Engineering*, page 284, San Diego, CA, Jan. 1993. IEEE Computer Society Press.

[9] R. F. Flynn and D. Dorfmann. The Automated Requirements Traceability System (ARTS): An Experience of Eight Years. In R. Thayer and M. Dorfman, editors, *Systems and Software Requirements Engineering*, pages 423–438. IEEE Computer Society Press — Tutorial, 1990.

[10] F. Gibbels. *Ein Modell für die Spezifikation von Informationssystemen: Definition und Unterstützung im Requirements Engineering*. Diplomarbeit, RWTH Aachen, 1994.

[11] O. Gotel and A. Finkelstein. An Analysis of the Requirements Traceability Problem. Technical Report TR-93–41, Imperial College, Department of Computing, 1993.

[12] O. Gotel and A. Finkelstein. An Analysis of the Requirements Traceability Problem. In *Proc. of the 1st Intl. Conference on Requirements Engineering*, pages 94–102, Colorado Springs, CO, Apr. 1994. IEEE Computer Society Press.

[13] O. Gotel and A. Finkelstein. Modeling the Contribution Structure Underlying Requirements. In *Proc. of the 1st Intl. Workshop on Requirements Engineering: Foundation of Software Quality*, Utrecht, The Netherlands, Aug. 1994. Augustinus-Verlag.

[14] P. Haumer. *Strukturierung und Integration von informalem Wissen in Anforderungsspezifikationen mittels Hypertext*. Diplomarbeit, RWTH Aachen, 1994.

[15] IEEE-830. Guide to Software Requirements Specification. 1984. ANSI/IEEE Std. 830.

[16] . ISO/IEC. *Information Technology – Information Resource Dictionary Systems (IRDS) – Framework*. ISO/IEC International Standard, 1990.

[17] M. Jarke, S. Eherer, R. Gallersdörfer, M. Jeusfeld, and M. Staudt. ConceptBase – A deductive Object Base Manager. *Journal on Intelligent Information Systems*, 1994. Ebenfalls erschienen als: RWTH-Aachen, Aachener Informatik Berichte, No. 93–14.

[18] M. Jeusfeld. *Änderungskontrolle in deduktiven Objektbanken*. INFIX Pub, Bad Honnef, Germany, 1992.

[19] W. L. Johnson, M. S. Feather, and D. R. Harris. Integrating Domain Knowledge, Requirements, and Specifications. *Journal of Systems Integration*, 1:283–320, 1991.

[20] H. Kaindl. The Missing Link in Requirements Engineering. *ACM SIGSOFT Software Engineering Notes*, 19(2):30–39, 1993.

[21] R. Klamma. *Präskriptive Prozeßbeschreibungen: Definition, Implementierung und Validierung in PRO-ART*. Diplomarbeit, RWTH Aachen, 1995.

[22] C. Lenzen. *Objektorientierte Analyse: Auswahl und Integration in die NATURE-Umgebung*. Diplomarbeit, RWTH Aachen, 1994.

[23] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: Representing Knowledge about Information Systems. *Transactions on Information Systems*, 8(4):325–362, 1990.

[24] K. Pohl. The Three Dimension of Requirements Engineering. In *Proc. of the 5th Intl. Conference on Advanced Information Systems Engineering*, pages 275–292, Paris, France, June 1993. Springer-Verlag.

[25] K. Pohl. The Three Dimension of Requirements Engineering: A Framework and Its Application. *Information Systems*, 3(19):243–258, June 1994.

[26] K. Pohl. *A Process Centered Requirements Engineering Environment*. PhD thesis, RWTH Aachen, 1995.

[27] K. Pohl. *Process Centered Requirements Engineering*. RSP marketed by John Wiley & Sons Ltd, UK, 1995. In Vorbereitung.

[28] K. Pohl, R. Dömges, and M. Jarke. Decision Oriented Process Modelling. In *Proc. of the 9th Intl. Software Process Workshop*, Arlie, VA, Oct. 1994. IEEE Computer Society Press.

[29] K. Pohl and P. Haumer. HYDRA: A Hypertext Model for Structuring Informal Requirements Representations. In *Proc. of the 2nd Int. Workshop on Requirements Engineering: Foundation of Software Quality (REFSQ'95)*, June, 12–13, Jyväskylä, Finland, 1995. Augustinus, Aachen, Germany.

[30] K. Pohl, M. Jarke, and R. Dömges. Unterstützung schwach strukturierter geschäftsprozesse. In *GI-Fachtagung "Geschäftsprozesse und Workflow-Systeme in der evolutionären Unternehmung", GI-FG 5.2.1 (MobIS)*, 1995.

[31] K. Pohl, G. Starke, and P. Peters. REFSQ'94: Workshop Summaries. *EMISA Forum*, (2):87–95, Aug. 1994. also appeared in ACM-Sigsoft Notes.

[32] C. Potts and G. Bruns. Recording the Reasons for Design Decisions. In *Proc. of the 10th Intl. Conference on Software Engineering*, Singapore, Apr. 1988.

[33] B. Ramesh. A Model of Requirements Traceability for Systems development. Technical report, Naval Postgraduate School, Monterey, CA, Sept. 1993.

[34] B. Ramesh and V. Dhar. Process-Knowledge Based Group Support in Requirements Engineering. *IEEE Transactions on Software Engineering*, 18(6), 1992.

[35] B. Ramesh and M. Edwards. Issues in the Development of a Requirements Traceability Model. In *Proc. of the 1st Intl. Symposium on Requirements Engineering*, San Diego, CA, Jan. 1993. IEEE Computer Society Press.

[36] B. Ramesh, T. Powers, C. Stubbs, and M. Edwards. Implementing Requirements Traceability: A Case Study. In *Proc. of the Second IEEE Int. Symposium On Requirements Engineering (RE'95)*, pages 89–95, March 27–29, York, UK, 1995.

[37] K. Weidenhaupt. *Adaptabilität in Entwicklungsumgebungen: Modellierung und Programmierung*. Diplomarbeit, RWTH Aachen, 1995.

[38] A. L. Wolf and D. S. Rosenblum. A Study in Software Process Data Capture and Analysis. In L. Osterweil, editor, *Proc. of the 2nd Intl. Conference on the Software Process*, pages 115–124, Berlin, Germany, Feb. 1993. IEEE Computer Society Press.