Wartungsunterstützung in heterogenen Sprachumgebungen

Ein Überblick zum Projekt GUPRO

Jürgen Ebert
Universität Koblenz-Landau
Institut für Softwaretechnik
Rheinau 1
56075 Koblenz
ebert@informatik.uni-koblenz.de

Rainer Gimnich
IBM Deutschland
Informationssysteme GmbH
Wissenschaftliches Zentrum
Institut für Datenbanken und
Softwareengineering
Vangerowstr. 18
69115 Heidelberg
gimnich@vnet.ibm.com

Andreas Winter
Universität Koblenz-Landau
Institut für Softwaretechnik
Rheinau 1
56075 Koblenz
winter@informatik.uni-koblenz.de

Januar 1996

erscheint in:
Franz Lehner:
Softwarewartung und Reengineering
Erfahrungen und Entwicklungen
Gabler: Wiesbaden, 1996

Tagungsband zur Fachtagung
„Softwarewartung und Reengineering“
der GI-Fachgruppe 5.1.3
(Reengineering und Wartung betrieblicher Anwendungssysteme),
11.-12. März 1996,
Regensburg.

Zusammenfassung

Wirtschaftliche Wartung und Weiterentwicklung von Anwendungssoftware setzt ein grundsätzliches Verstehen vorhandener Quelltexte voraus. Diese sind oft wenig strukturiert, schwach kommentiert und in unterschiedlichen Umgebungen entstanden. Im Projekt GUPRO - Eine Generische Umgebung zum Programmverstehen - wird ein benutzerkonfigurierbarer Generator zur Erzeugung sprachübergreifender Programmverstehenswerkzeuge entwickelt, die das Nachvollziehen und Verstehen auch heterogener Software beliebiger Sprachen (Programmiersprachen, Anfragesprachen, Sprachen der „4. Generation“) unterstützen. Hierzu dienen Anfrage- und Browsing-Werkzeuge, die über eine gemeinsame, graphbasierte Datenstruktur integriert sind.

1 Software-Wartung und Programmverstehen in der Praxis

Die Praxis der Software-Entwicklung zeigt, daß vorhandene Programm-Quelltexte trotz besseren Wissens unter dem üblichen Zeitdruck meist *schwach dokumentiert* oder gar *unkommentiert* bleiben. Zudem führt die korrektive und adaptive Wartung der Software mit der Zeit häufig zu *schlecht strukturierten* Programmtexten, da der ursprüngliche Programmentwurf durch die notwendigen Korrekturen, Anpassungen und Erweiterungen erheblich verändert wird. Diese Situation erschwert die Wiederverwendung von Programmen bzw. Programmteilen.

Der Aufwand, sich mit den vorhandenen Programmtexten zu beschäftigen, um sie wiederzuverwenden, erscheint oft höher als eine völlige Neuentwicklung, aus denen dann häufig die gleichen Schwächen bezüglich Dokumentation und Klarheit resultieren. Daher sind dringend Hilfsmittel erforderlich, die das Überarbeiten der vorhandenen Quellen erleichtern, um so deren Wiederverwendung und Weiterentwicklung zu unterstützen.

In gängigen Programmierumgebungen stehen zu diesen Zwecken verschiedene Optionen zur Erzeugung von übersichtlichen Listings, zur farblichen Hervorhebung der Programmstruktur oder zur Erstellung von Querverweislisten zur Verfügung. Darüber hinaus bieten Debugger zur Analyse des dynamischen Programmverhaltens Hilfen bei der Verfolgung der Programmausführung, z.B. um die Belegung von Variablen oder den Aufruf von Funktionen während des Programmablaufs zu verfolgen. Zusätzliche Hilfe liefern

- *Formatierungswerkzeuge*, die auch spezifische durch den Einzelfall festgelegte Kommentierungskonventionen berücksichtigen, und
- *Analysewerkzeuge*, die auch globale versteckte Abhängigkeiten zwischen Bausteinen erkennen.

Für die Zwecke der Wartung sind jedoch weitergehende Hilfsmittel erforderlich, die gezielte Unterstützung in Wartungssituationen leisten. Hierzu gehören

- *Anfragewerkzeuge*, die es erlauben, verborgene Informationen aus dem gesamten System auf eine kompakt gestellte Frage hin zusammengefaßt und übersichtlich strukturiert zu liefern, und
- *Navigationswerkzeuge*, mit denen speziellen Abhängigkeiten interaktiv nachgegangen werden kann.

Die Werkzeuge müssen in einer integrierten, auf die Anforderungen der Anwender hin *konfigurierbaren Wartungsumgebung* zusammengefaßt sein. Diese dient der effizienten Durchführung von Software-Wartungsaktivitäten, indem insbesondere das *Verstehen* der Software unterstützt wird. Die Unterstützung sollte programmübergreifend und möglichst auch *programmiersprachenübergreifend* realisiert sein.

In der Software-Wartung wird das *Verstehen* der Anwendungen, ihrer Programme, Datenbestände und deren Zusammenhänge als zeitaufwendigste Aktivität berichtet (u.a. in [Corbi89, Sneed91, Lehn91, McCl92]). Die meisten über Jahre gewachsenen Anwendungen enthalten eine Vielzahl schwer aufdeckbarer Querbeziehungen (*Cross-Referenzen*) unterschiedlicher Art, z.B. Aufruf-, Einschluß-, Verwendungsabhängigkeiten. Dieses Geflecht aus nicht ausreichend dokumentierten Zusammenhängen erschwert das Verstehen und erklärt zum Teil die heute von vielen Unternehmen (u.a. von großen Finanz-, Versicherungs- und Handelsunternehmen) berichteten hohen Wartungsaufwände. Dabei ist zu berücksichtigen, daß die zu wartenden Anwendungen oft einen hohen wirtschaftlichen Wert haben, der zur Marktposition des Unternehmens entscheidend beiträgt.

2 GUPRO-Projektziele

Im Projekt **GUPRO**[1] — Eine **G**enerische **U**mgebung zum **Pro**grammverstehen — wird ein benutzerkonfigurierbarer Generator zur Erzeugung sprachübergreifender Programmverstehenswerkzeuge entwickelt. Das Projekt basiert auf den praktischen Anforderungen eines Anwenders. Die GUPRO-Verbundpartner sind die Volksfürsorge Versicherungsgruppe, Hamburg, das Institut für Softwaretechnik der Universität Koblenz und das Institut für Datenbanken und Software Engineering des Wissenschaftlichen Zentrums Heidelberg der IBM.

In das Projekt fließen Erfahrungen mit heutigen Programmverstehenswerkzeugen ein. Diese leisten in der Regel Unterstützung für genau eine Programmiersprache oder eine fest vorgegebene Menge von Programmiersprachen. Damit können meist nur Teilbereiche der täglichen Software-Wartung im Unternehmen unterstützt werden. Erforderlich sind hier angepaßte Werkzeuge, die die jeweilige Sprachumgebung des Benutzers berücksichtigen.

Häufig basieren Anwendungen auf verschiedenartigen Sprachen, z.B. eine Anwendung aus COBOL-Hauptprogramm, COBOL- und Assembler-Unterprogrammen und eingebetteten SQL-Anweisungen, oder eine Anwendung aus PL/I-Programmen, nativenKOBOL-Programmen und über eine 4GL generierten COBOL-Programmen, wobei die Abläufe jeweils über JCL (Job Control Language) -Prozeduren gesteuert werden. Diese *Heterogenität* muß geeignet berücksichtigt werden. Es sind nicht nur Programmiersprachenkonzepte zu repräsentieren, sondern auch Anfrage- und 4GL-Sprachen. Außerdem ist die beim Anwender vorliegende Sprachumgebung nicht vorhersehbar, so daß es sinnvoll ist, das geforderte Verstehenswerkzeug über einen Generator-Ansatz zu erstellen.

Heutige Werkzeuge zum Programmverstehen beinhalten in der Regel eine fest vorgegebene Menge von Anfrage- und Analysemöglichkeiten, z.B. nach Labels, Prozeduren, Variablen. Doch eigene Untersuchungen am Beispiel der Programmiersprache C [Schü94] haben gezeigt, daß Benutzer in der Regel ein breites Spektrum von Analyse-

[1] GUPRO wird gefördert durch das BMBF, Initiative zur Förderung der Software-Technologie in Wirtschaft, Wissenschaft und Technik, Förderkennzeichen 01 IS 504 B 9.

funktionen sowie eine flexible Anfrageformulierung, d.h. eine Anfragesprache, wünschen. Daher sind in GUPRO leistungsfähige **Anfrage- und Navigationswerkzeuge** (Browser) vorgesehen, die auch kombiniert benutzt werden können.

Der in GUPRO verfolgte Generator-Ansatz kommt damit der vom Anwender gewünschten Flexibilität entgegen, sowohl in bezug auf die vorhandene Sprachumgebung als auch in bezug auf die geforderten Analyse- und Anfragemöglichkeiten.

3 Der GUPRO-Ansatz

Der GUPRO-Ansatz zur Wartungsunterstützung ist durch die Generierung und Anpassung von Werkzeugen geprägt. Es wird ein Rahmen vorgegeben, innerhalb dessen der Anwender seine Arbeitsumgebung und Informationswünsche selbst festlegen bzw. spezialisieren kann.

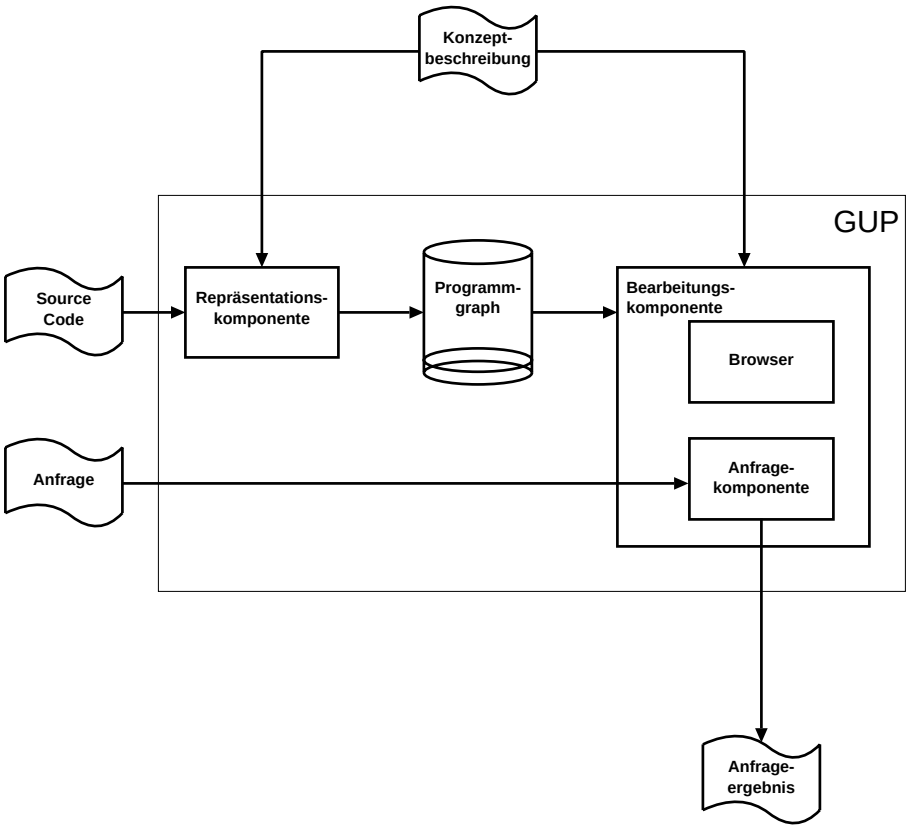

Abbildung 1: GUPRO-Architektur

Dieser universelle Ansatz läßt sich auf zwei grundlegende Komponenten zurückführen:

- Mit einer Repräsentationskomponente werden die zu untersuchenden Programme in eine von konkreten Programmiersprachen unabhängige Repräsentationsform gebracht.

- Auf dieser Repräsentation können dann in einer Bearbeitungskomponente diverse Werkzeugfunktionen etwa zur Formatierung und Analyse von Programmsystemen aufsetzen. In GUPRO sind hier insbesondere Anfrage- und Navigationswerkzeuge geplant.

Die für GUPRO entwickelte Architektur ist grob in Abbildung 1 dargestellt.

Da das gesuchte Werkzeug zur Unterstützung des Programmverstehens für zahlreiche konkrete Programmiersprachen anwendbar sein soll, erfordert die Repräsentationskomponente eine konzeptionelle Vorbereitung. Hierzu sind die für die Anwendung relevanten Konzepte bei der zu analysierenden Software herauszuarbeiten und für die Überführung der konkreten Software in die einheitliche Repräsentation zugänglich zu machen.

Das Vorgehen in GUPRO beinhaltet eine einheitliche *Konzeptmodellierung* mit einer erweiterten Entity-Relationship-Beschreibungform (EER-Diagramme). Diese Art der Konzeptbeschreibung erlaubt es, Sprachkonzepte (Entities), Beziehungen (Relationships), Fallunterscheidungen (Generalisierungen) und Komposition (Aggregationen) auszudrücken [CEW94]. Die Modellierung einer gegebenen Sprachumgebung (z.B. PL/I, COBOL, CSP) liefert Konzeptdiagramme für jede einzelne Sprache, die zusammengefügt werden können und die die Grundlage zum Generieren von Parsern für diese Sprachumgebung bilden.

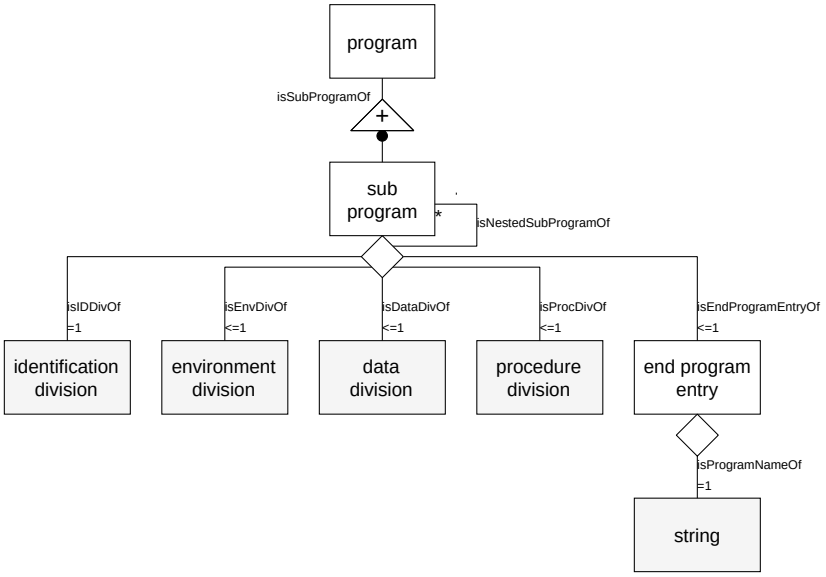

Abbildung 2: Konzeptdiagramm zum Aufbau eines COBOL-Programms

Als Beispiele für Konzeptdiagramme sind in den Abbildungen 2 und 3 Ausschnitte der Modellierung der Programmiersprache COBOL dargestellt. Abbildung 2 beschreibt grob den Aufbau eines COBOL-Programms; Abbildung 3 beinhaltet den Aufbau der PROCEDURE DIVISION in einer grobgranularen Form. Doch bereits auf dieser Abstraktionsstufe sind Anfragen zur SECTION- bzw. Paragraphenstruktur eines COBOL-Programms und zum Aufbau der Paragraphen aus Sätzen und Einzelanweisungen möglich.

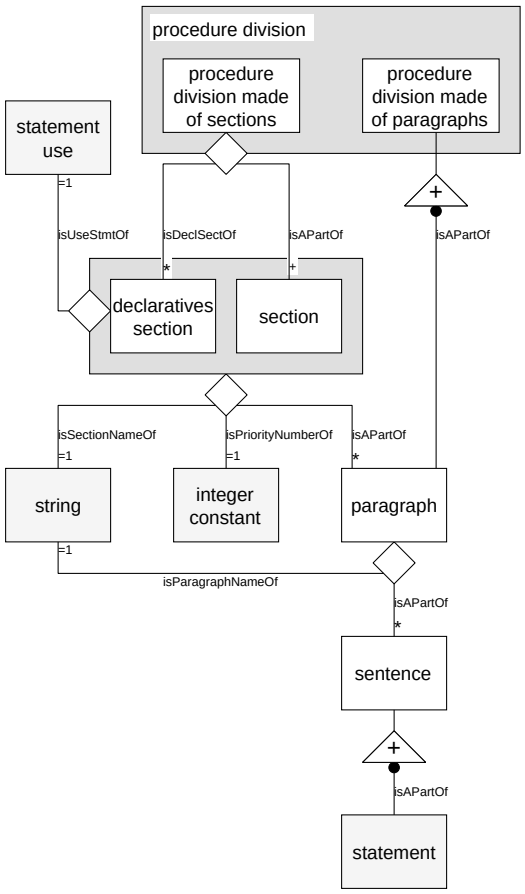

Abbildung 3: Konzeptdiagramm zum Aufbau der PROCEDURE DIVISION eines COBOL-Programms

Der Feinheitsgrad der Modellierung ist dabei frei wählbar, so daß die individuellen Anwender-Anforderungen gezielt berücksichtigt werden können. Die Modellierung kann z.B. grobgranular auf der Stufe der Dateien, Prozeduren und Module oder feingranular auf der Stufe der Variablen, Ausdrücke und Zuweisungen erfolgen. Als Beispiel wird der Aufbau der PROCEDURE DIVISION (vgl. Abbildung 3) in COBOL um zwei Stufen weiter verfeinert und dann das MOVE-Statement herausgegriffen, dargestellt in Abbildung 4. Eine vollständige Modellierung von COBOL (ANSI85-Standard) liegt in [Hümm95] vor.

Dem Anwender wird durch die Modellierung die Möglichkeit gegeben, das Werkzeug individuell auf seine Anforderungen hinsichtlich der zu der wartenden Software zu konfigurieren. Insbesondere können neben Programmiersprachenkonzepten auch Konzepte von Datenbank-Anfragesprachen (z.B. SQL, DL/I, DDIL) oder von Sprachen der „4. Generation“ (z.B. CSP, Delta) in diesem einheitlichen Rahmen mitmodelliert werden.

Durch die Verwendung der EER-Diagramme als Konzeptbeschreibung erhält der Anwender ein anschauliches Beschreibungsmittel für die ihn interessierenden Konzepte. Durch das Beschreibungsmittel ist aber gleichzeitig, aufgrund seiner definierten Se-

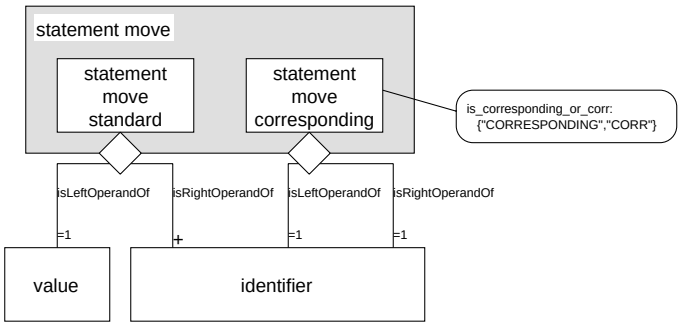

Abbildung 4: Konzeptdiagramm zum Aufbau des MOVE-Statements in COBOL

mantik, auch eine Datenstruktur formalisiert.

Ein EER-Diagramm legt stets eine bestimmte *Graphklasse* fest [CEW94], die als Konzeptschema für die zu realisierende Anwenderumgebung gesehen werden kann. Konkrete Software-Systeme werden entsprechend dem definierten Konzeptschema analysiert und jeweils als Instanzen (d.h. Instanzgraphen) repräsentiert. Auf diesen Graphen erfolgt dann die weitere Bearbeitung zur Unterstützung des Programmverstehens. Das Anfragen und Navigieren wird auf der Basis dieser Datenstrukturen realisiert, wobei dem Anwender weiterhin die Konzeptbeschreibung für seine Interaktion mit dem Werkzeug dient. Grundlage für die Entwicklung der Repräsentations- und Bearbeitungskomponenten bildet das *Graphenlabor* der Universität Koblenz [Ebert87, DEL94].

Die Anfragen bzw. Informationswünsche von Anwendern beim Programmverstehen sind vielfältig und können durch die Modellierung der geforderten Konzepte auf unterschiedlichen Abstraktionsstufen abgedeckt werden. Die Formulierung konkreter Anfragen auf dem Konzeptschema muß in einer geeigneten Anfragesprache textuell oder graphisch einfach durchführbar sein. Im Beispiel der grobgranularen Modellierung von COBOL-Prozedurteilen (Abbildung 3) können z.B. Anfragen der Art „Welche Statements enthält Paragraph A?“ oder „Wie viele Sätze sind in jeder Section dieses Programms enthalten?“ gestellt werden. Verfeinerungen des Konzeptschemas beinhalten z.B. eine Unterscheidung nach der Art der Statements (Arithmetik, Datenveränderung, Ein-/Ausgabe, Kontrollfluß, u.a.) und die Einführung von variablen-bezogenen Konzepten (z.B. Deklaration, Verwendung von Variablen), so daß sich die Anfragemöglichkeiten erheblich erweitern.

Charakteristische Anfragen auf hoher Abstraktionsstufe beziehen sich auf Aufrufbeziehungen zwischen Programmen und Prozeduren/Funktionen, wobei hier heterogene Strukturen (z.B. von COBOL-Programmen aufgerufene Assembler-Programme) mitberücksichtigt werden, und auf Beziehungen zwischen Programmen und externen Datenbeständen (Dateien, Datenbanken). Ergebnisse aus Anfragen dieser Art sind nicht nur in der täglichen Wartungsarbeit, sondern bei anstehenden Reengineering-Maßnahmen sehr nützlich (z.B. Migration der Datenhaltung für eine Anwendung aus 50 Programmen).

Die Entwicklung einer leistungsfähigen textuellen und graphischen *Anfragesprache*

für die Zwecke des Programmverstehens und einer ergonomisch angemessenen *Benutzerschnittstelle* des Systems bilden wesentliche Arbeitspakete des Projekts. Hier fließen Erfahrungen aus vorangegangenen Projekten in der Software-Ergonomie und Software-Wiederverwendung ein. So wird insbesondere eine geeignete Kombination aus Anfrage- und Browsing-Möglichkeiten angestrebt [GCGW93], bei der Browsing-Ergebnisse z.B. den Suchbereich der weiteren Anfragen festlegen.

4 Projektstand

Nach [IBM93, IBM94] legen wir folgende Begriffsbestimmungen innerhalb Reengineering und Wartung zugrunde:

Bestandsanalyse (Inventory Analysis): Erstellung eines Inventars aller Anwendungen und ihrer Komponenten und Bewertung dieser Anwendungen (z.B. nach Komplexität, Bedeutung für das Unternehmen), vgl. auch [GiHo95]. Die Ergebnisse der Bewertung können als Entscheidungsgrundlage für das weitere Vorgehen, u.a. Weiterentwicklung, Reengineering (Restrukturierung, Redokumentation) oder Migration, dienen.

Anwendungsverstehen (Application Understanding) Verstehen der Anwendung und ihrer Komponenten (Programme, Datenbestände, Prozesse) sowie der involvierten Hardware-/Software-Systeme (u.a. Rechnertypen, Datenhaltungssysteme, Kommunikation, Benutzungsoberflächen).

Programmverstehen (Program Understanding) Verstehen der einzelnen Programme, u.a. im Hinblick auf Datendefinitionen, Datenfluß, Kontrollfluß, Aufrufstrukturen.

Die Bestandsanalyse wird demnach überwiegend auf der Ebene des Unternehmens bzw. der IT-Organisation durchgeführt; die Verstehensunterstützung erfolgt auf der Ebene der Anwendungen und auf der Ebene der Programme. Aufgrund der Ähnlichkeit der Problemstellungen auf diesen drei Ebenen wird in GUPRO ein möglichst einheitliches Vorgehen in der Modellierung und Werkzeugfunktionalität entwickelt. Beispielsweise gibt es sogenannte „weite Cross-Referenzen“, die Abhängigkeiten auf Bestandsanalyse-Ebene erfassen, z.B. alle — auch indirekten — Aufrufbeziehungen, die von einem Programm ausgehen und die ein bestimmtes (aufgerufenes) Programm betreffen. Diese Zusammenhänge können auf Programmverstehens- ebene durch weitere Cross-Referenzen (z.B. Deklarationen und Verwendungen von Variablen oder Prozeduren) ergänzt werden, die mit den gleichen Sprachmitteln darstellt und bearbeitet werden können.

In GUPRO wird derzeit das folgende Vorgehen erforscht: Die konzeptuelle Modellierung beginnt auf der Ebene des *Anwendungsverstehens*. Die noch grobgranularen Modelle der betreffenden Sprachen werden dann für die *Bestandsanalyse*-Ebene zu einem sprachübergreifenden Modell zusammengefügt und weiter vergröbert. Das Modell

auf Anwendungsebene bildet auch die Grundlage für das *Programmverstehen*smodell. Hierfür werden die grobgranularen Modelle der einzelnen Sprachen weiter verfeinert, bis auf eine vom Anwender gewünschte Feinheitsstufe.

GUPRO wurde Mitte 1995 begonnen und hat eine Laufzeit von 3 Jahren. Vorarbeiten liegen in der Modellierung graphischer Sprachen und in der Graphenlabor-Software der Universität Koblenz sowie in dem dort entwickelten C-Analyse-System CANAL [DaHo92] und in feingranularen Modellen der Sprachen COBOL und PL/I. Weiterhin bringt die Universität Koblenz umfangreiche Erfahrungen zur graphbasierten Modellierung, zur Entwicklung von Anfragesprachen und zur Parsergenerierung in das Projekt ein.

Am Wissenschaftlichen Zentrum Heidelberg der IBM Deutschland werden Reengineering-Forschungsprojekte auf der Basis praktischer Anforderungen durchgeführt. Aktuelle Reengineering-Projekte liegen in den Gebieten Anwendungs- und Programmverstehen, Sprachmigration und Software-Wiederverwendung. Schwerpunkte bilden dabei die Sprachen COBOL, PL/I und FORTRAN. Die entwickelten Reengineering-Methoden und -Werkzeuge werden unmittelbar in Kundenprojekten eingesetzt. Neben der Reengineering-Basis werden für GUPRO auch Erfahrungen in der Entwicklung von Datenbank-Anfragesprachen sowie Methodenwissen aus der Software-Ergonomie genutzt.

Die Volksfürsorge Versicherungsgruppe bringt langjährige und umfassende Software-Entwicklungserfahrungen mit mehreren Programmier-, Datenbank- und Benutzerschnittstellen-Systemen in das Projekt ein. Auch liegen bei der Volksfürsorge umfangreiche Erfahrungen in der Datenmodellierung und Ablaufsteuerung großer Anwendungskomplexe vor. Im Hause Volksfürsorge befinden sich über 16 000 Programmeinheiten in unterschiedlichen Sprachen im Einsatz, so daß eine realistische Umgebung zur Entwicklung und zum Einsatz der GUPRO-Werkzeuge gegeben ist.

Derzeit wird in einer gemeinsamen Entwicklung der drei Verbundpartner ein grobgranulares, integriertes Modell für die Sprachumgebung PL/I, COBOL, CSP und JCL mit DB2- und IMS-Anbindung fertiggestellt. Gleichzeitig wird für dieses Modell ein Parser entwickelt, der zur Unterstützung der Bestandsanalyse im Hause Volksfürsorge genutzt werden soll. Weitere aktuelle Projektarbeiten beinhalten die Entwicklung einer textuellen, SQL-ähnlichen Anfragesprache auf Graphen, die Entwicklung einer geeigneten Report-Struktur auf der Basis der Anfrage-Ergebnisse sowie den Entwurf einer adäquaten graphischen Benutzerschnittstelle für den Wartungsarbeitsplatz. Die Projektzwischenergebnisse werden laufend in der Praxis validiert.

In der zweiten Hälfte des Projektzeitraums werden die graphische Anfragesprache sowie ein geeignetes Browserkonzept realisiert und in die Benutzerschnittstelle integriert. Gleichzeitig wird die Generator-Umgebung für das Programmverstehen weiterentwickelt und in verschiedenen Gebieten der Anwendungsentwicklung erprobt. Die Konsolidierung der Projektergebnisse und der Transfer der Ergebnisse auf neue Anwendungssituationen bilden wesentliche Bestandteile der Projektarbeit. Die Arbeitspakete sind „verzahnt“ konzipiert, so daß jeder Partner sein System- und Methodenwissen und seine Erfahrungen möglichst gut einbringen und austauschen kann.

Literatur

[CEW94] Carstensen, Martin; Ebert, Jürgen; Winter, Andreas: Deklarative Beschreibung von Graphensprachen. GI-Fachgruppe 2.1.4 Alternative Konzepte für Sprachen und Rechner, Bad Honnef, Mai 1994 (Fachbericht der Universität Kiel).

[Corbi89] Corbi, Thomas A.: Program Understanding: Challenge for the 1990s. In: IBM Systems Journal 28 (1989), 2, S. 294-306.

[DaHo92] Daute, Oliver; Horn, Georg: CANAL - Ein Reverse-Engineering-Werkzeug für die Programmiersprache C. Diplomarbeit, Universität Koblenz, Fachbereich Informatik, 1992.

[DEL94] Dahm, Peter; Ebert, Jürgen; Litauer, Christoph: Benutzerhandbuch EMS-Graphenlabor. Universität Koblenz, Fachbereich Informatik, 1994. (verfügbar über FTP: ftphost.uni-koblenz.de/outgoing/GraLab)

[Ebert87] Ebert, Jürgen: A Versatile Data Structure for Edge-Oriented Graph Algorithms. In: Comm. ACM 30 (1987), 6, S. 513-519.

[GCGW93] Gimnich, Rainer; Convent, Bernd; Günauer, Jürgen; Wernecke, Wolfgang: Supporting Reuse by Software Documents Management. In: IBM Software Engineering ITL, Toronto, Kanada, 1993, S. 123-142.

[GiHo95] Gimnich, Rainer; Hofinger, Alois: Bestandsanalyse als Grundlage der Software-Redevelopment-Strategie. GI-Fachgruppe 5.1.3 Reengineering und Wartung, Münster, März 1995.

[Hümm95] Hümmerich, Martin: Entwicklung und prototypische Implementation eines konzeptionellen Modells zum Reverse Engineering von ANSI85-COBOL- Programmen. Studienarbeit, Universität Koblenz, Fachbereich Informatik, 1995.

[IBM93] IBM: COBOL Code Reengineering. Doc. No. GG24-3979-0. International Technical Support Center, San Jose, CA., May 1993.

[IBM94] IBM Deutschland Informationssysteme GmbH: Lösungsangebot Software Redevelopment. Heidelberg, April 1994.

[Lehn91] Lehner, Franz: Softwarewartung. Hanser, München, 1991.

[McCl92] McClure, Carma: The 3 Rs of Software Automation - Reengineering, Repository, Reusability. Prentice-Hall, Englewood Cliffs, NJ, 1992.

[Schü94] Schütze, Jürgen: Konzept für ein interaktives, graphbasiertes Werkzeug zum Verstehen von Programmen. Studienarbeit, Universität Koblenz, Fachbereich Informatik, 1994.

[Sneed91] Sneed, Harry M.: Softwarewartung. Müller, Köln, 1991.